Inżynieria oprogramowania
Wybrane problemy

Redakcja naukowa:

Zdzisław Szyjewski, Jakub Swacha

Konferencje naukowe organizowane przez

Polskie Towarzystwo Informatyczne:

VIII edycja Sejmiku Młodych Informatyków

XV edycja Krajowej Konferencji Inżynierii Oprogramowania

XX edycja Systemów Czasu Rzeczywistego

zostały dofinansowane

przez Ministra Nauki i Szkolnictwa Wyższego

w ramach programu związanego z realizacją zadań upowszechniających naukę

(decyzja nr 1064/P-DUN/2013 z dnia 24.07.2013)

Dziękujemy!

POLSKIE TOWARZYSTWO INFORMATYCZNE

Inżynieria oprogramowania
Wybrane problemy

Redakcja naukowa:

Zdzisław Szyjewski, Jakub Swacha

Warszawa 2013

Recenzenci

prof. dr hab. inż. Wojciech Cellary, dr inż. Jakub Chabik, prof. dr hab. inż. Zbigniew Czech, prof. dr hab. Zygmunt Drążek, Piotr Fuglewicz, prof. dr hab. inż. Janusz Górski, dr inż. Bogumiła Hnatkowska, dr inż. Marek Hołyński, prof. dr hab. inż. Jacek Kitowski, prof. dr hab. inż. Zdzisław Kowalczuk, prof. dr hab. inż. Stanisław Kozielski, prof. dr hab. Jan Madey, prof. dr hab. inż. Jan Magott, dr inż. Marcin Mastalerz, dr Karolina Muszyńska, prof. dr hab. inż. Edward Nawarecki, Dariusz Nogalski, dr hab. inż. Cezary Orłowski, dr Łukasz Radliński, dr inż. Andrzej Stasiak, dr Krzysztof Świder, prof. dr hab. inż. Antoni Wiliński

Redakcja naukowa

prof. dr hab. Zdzisław Szyjewski, dr hab. Jakub Swacha

Autorzy

Cezary Orłowski , Bartosz Chrabski, Kamil Dowgielewicz – ROZDZIAŁ 1
Ilona Bluemke, Anna Stepień – ROZDZIAŁ 2
Karol Kempa, Anna Wawszczak – ROZDZIAŁ 3
Bartosz Chrabski, Robert Siara – ROZDZIAŁ 4
Iwona Dubielewicz, Bogumiła Hnatkowska, Zbigniew Huzar, Lech Tuzinkiewicz – ROZDZIAŁ 5
Grzegorz Timoszuk – ROZDZIAŁ 6
Włodzimierz Dąbrowski, Andrzej Stasiak, Krzysztof Wnuk – ROZDZIAŁ 7
Irena Bach-Dąbrowska, Artur Ziółkowski – ROZDZIAŁ 8
Bogdan Bereza, Krzysztof Wnuk – ROZDZIAŁ 9

Redakcja techniczna

Karolina Muszyńska

Projekt okładki

Łukasz Piwowarski

ISBN 978-83-7518-598-0

Wydanie: I. Nakład: 200 egz. Ark. wyd. 6,65. Ark. druku 8,31.
Wydawca, druk i oprawa: PPH ZAPOL, al. Piastów 42, 71-062 Szczecin

Spis treści

Wstęp

Każda dziedzina aktywności przeżywa różne okresy rozwoju. Po początkowym okresie burzliwego, nieuporządkowanego rozwoju, następuje faza stabilizacji i porządkowania procesów wytwarzania. Później następuje próba standaryzacji rozwiązań i metod wytwarzania, tworzenie narzędzi wspomagania procesów wytwórczych, wypracowywania metod i technik podnoszących sprawność wytwarzania i jakość rozwiązań. Analogicznie możemy obserwować etapy rozwoju informatyki a w szczególności oprogramowania komputerów, co stanowi podstawową aktywność informatyczną. Po okresie dynamicznego rozwoju metod i technik indywidualnego, „artystycznego", podejścia do problematyki programowania komputerów, następuje etap polegający na ujednoliceniu i systematyzacji procedur i procesu programowania. Standaryzacja programowania i wypracowanie uniwersalnych metod i technik programowania podnoszą sprawność procesu tworzenia oprogramowania i dają lepsze wyniki ilościowe i jakościowe wytwarzania oprogramowania. Wzorcem staje się inżynierskie podejście do procesu wytwarzania, stosowane w innych aktywnościach wytwórczych.

Zasadnicza różnica, która wyróżnia wytwarzanie oprogramowania od wytwarzania innych produktów technicznych, polega na niematerialnym charakterze produktu końcowego. Wymaga to dostosowania procedur, wykorzystywania specyficznych standardów postępowania oraz sprawdzonych wzorców inżynierskiego procesu wytwarzania. Podstawowym problemem wytwarzania oprogramowania w porównaniu z innymi obiektami inżynierskimi jest brak powszechnie akceptowalnych i stosowanych miar, co nie tylko utrudnia określenie wielkości produktu końcowego, ale również skali jego złożoności i wynikającej z tego pracochłonności jego wytworzenia.

Innym istotnym problemem inżynierii oprogramowania, gdyż tak przyjęto określać ten obszar aktywności zmierzających do wypracowania metod i technik mających na celu podniesienie sprawności i jakości programowania, jest problematyka specyfikacji wymagań produktu końcowego, jakim jest program komputerowy lub system oprogramowania. Konieczność znalezienia wspólnego i zrozumiałego dla wszystkich zainteresowanych sposobu komuni-

kowania się w środowisku informatycznym jest obiektem stałych prób i badań. Hermetyczny język stosowany w różnych obszarach aktywności stanowi poważny problem dla zrozumiałej dla wszystkich komunikacji i wyrażania swoich potrzeb, tak aby powstała specyfikacja była jednoznaczna i zrozumiała, ale również łatwa w implementacji.

Inżynieria oprogramowania posiada stosunkowo długą, bo kilkudziesięcioletnią, historię rozwoju i ma na koncie niewątpliwe sukcesy dla rozwoju informatyki. Mimo to, w dalszym ciągu prowadzone są intensywne prace w tym zakresie i odkrywane nieznane dotąd możliwości dalszego rozwoju w coraz to nowszych obszarach. Prowadzone są stale badania nad podnoszeniem sprawności wytwarzania oraz doskonalenia metod wspomagania i podnoszenia jakości procesów wytwarzania oprogramowania.

Wybranym i aktualnym problemom inżynierii oprogramowania poświęcamy tę monografię. Podzielono ją na trzy części, podejmujące odpowiednio tematykę modeli i metod wykorzystywanych w inżynierii oprogramowania, jakości oprogramowania i edukacji w zakresie inżynierii oprogramowania.

Część pierwszą otwiera rozdział autorstwa grupy badaczy pod kierunkiem prof. Cezarego Orłowskiego, w którym opisano koncepcję maszyny wnioskującej, która umożliwiać będzie wspomaganie rozwoju narzędzi *open source* oraz proces ich doboru u dostawcy usług informatycznych. W rozdziale drugim opisano wzorzec projektowy DCI, w szczególności jego architekturę i sposób wykorzystania na przykładzie aplikacji w języku Ruby. W zbliżonym nurcie tematycznym mieści się rozdział trzeci, w którym para młodych badaczy z Politechniki Częstochowskiej pokazała, jak zaawansowane mechanizmy języka Objective-C mogą być wykorzystane do tworzenia łatwych w integracji komponentów. Z kolei w rozdziale czwartym zaprezentowano odważną koncepcję zastosowania mechanizmu profili UML w testowaniu opartym na modelach, bazującą na podejściu „4+1" dla modelowania architektury oprogramowania.

Część drugą rozpoczyna rozdział autorstwa niezwykle silnego składem zespołu badawczego z Politechniki Wrocławskiej, w którym przedstawiono wyniki pracy nad identyfikacją praktyk stosowanych w metodykach zwinnych, które mogą wspierać procesy cyklu życia oprogramowania związane z zapewnieniem jakości. Nowe światło na problematykę jakości oprogramowania rzuca z pewnością rozdział szósty, w którym Grzegorz Timoszuk przedstawia

autorską metodę wizualnej prezentacji oprogramowania, która umożliwia szybką ocenę jego jakości.

Część trzecia i ostatnia monografii zaczyna się rozdziałem opisującym kształcenie inżynierii wymagań i procesy inżynierii wymagań według organizacji *International Requirements Engineering Board* (IREB). Rozdział ósmy, opierając się na doświadczeniach Politechniki Gdańskiej, opisuje zasady organizowania i działania Uniwersyteckiego Centrum Kompetencyjnego Technologii Oprogramowania. W kończącym monografię rozdziale dziewiątym, z wielką swadą Bogdan Bereza i Krzysztof Wnuk rozprawiają na temat współpracy między przemysłem informatycznym i uczelniami, ilustrując swoje przemyślenia doświadczeniami szwedzkimi.

Życząc czytelnikom pouczającej i interesującej lektury, dziękujemy jednocześnie wszystkim osobom, które przyczyniły się do powstania niniejszej monografii: autorom, którzy dostarczyli jakże wartościowe teksty, recenzentom, którzy mimo wielu obowiązków znaleźli czas na wnikliwe ich przestudiowanie i wskazanie możliwości istotnych poprawek, oraz redaktor technicznej, dr Karolinie Muszyńskiej, której zawdzięczamy ostateczną postać niniejszego dzieła.

Zdzisław Szyjewski
Jakub Swacha

CZĘŚĆ I.
MODELE I METODY INŻYNIERII OPROGRAMOWANIA

Rozdział 1

Koncepcja budowy maszyny wnioskującej modelu referencyjnego rozwoju i doboru narzędzi Open Source

Rozdział przedstawia koncepcję budowy maszyny wnioskującej dla modelu referencyjnego wspomagającego rozwój narzędzi Open Source oraz proces ich doboru u dostawcy usług IT. Koncepcja silnika maszyny wnioskującej bazuje na wykorzystaniu mechanizmów wnioskowania wprzód. Prezentuje się zależności pomiędzy typami narzędzi, a możliwościami ich wykorzystania lub rozbudowy w oparciu o dostępne funkcjonalności modelu referencyjnego. Autorzy (celem weryfikacji opracowanej koncepcji) analizują także działanie maszyny wnioskującej na podstawie wykorzystywanych przez decydentów w organizacjach dostawcy usług IT praktyk w dziedzinie doboru i rozwoju narzędzi Open Source. Zakłada się, że kolejnym etapem budowy maszyny wnioskującej będzie opracowanie reguł wnioskowania opartych o mechanizmy sztucznej inteligencji dla doboru funkcjonalności dla rozwijanego w organizacjach dostawcy usług IT oprogramowania.

Dobór narzędzi u dostawcy usług IT jest z reguły poprzedzony subiektywnymi ocenami kierownika projektu oraz programistów uczestniczących w realizacji projektu. Wybierane są albo narzędzia najbardziej znane albo te najlepiej dostępne. Z tego też powodu znaczna część dostawców usług IT wykorzystuje narzędzia Open Source. Narzędzia te nie są jednak, aż tak zaawansowane (liczba funkcjonalności) jak narzędzia komercyjne. Dlatego też, organizacje dostawcy usług IT chcące korzystać z Open Source wdrażają po kilka różnych ich typów, aby wypełnić zbiór wymaganych przez organizację oczekiwań w stosunku do dostępnych funkcjonalności [1][10]. Często takie praktyki pociągają za sobą powielanie funkcjonalności narzędzi i problemy z ich

integracją [12][13]. Zainteresowanie narzędziami Open Source jest także konsekwencją kosztów licencji narzędzi komercyjnych, często także problemami integracji z pozostałym portfolio oraz wysokimi kosztami wsparcia.

Problemem rynku informatycznego jest brak wzorców rozwoju i integracji narzędzi Open Source [3]. Dlatego też autorzy pracy podjęli się próby budowy modelu referencyjnego rozwoju i doboru narzędzi informatycznych. Wybrali grupę narzędzi do wspomagania pozyskiwania wymagań. Zaproponowali także możliwości weryfikacji opracowanego modelu referencyjnego w procesach pozyskiwania wymagań w projektach informatycznych.

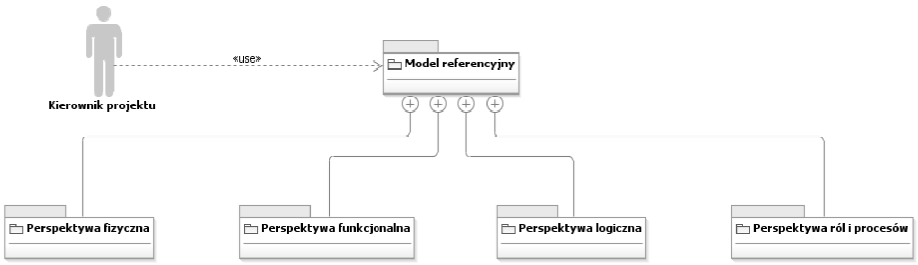
1.1. Koncepcja modelu referencyjnego rozwoju i doboru narzędzi informatycznych

Opracowana przez autorów koncepcja koncentruje się na formalnym wsparciu kierunków rozwoju stosowanych przez organizacje dostawcy usług IT narzędzi informatycznych (dobru narzędzi lub rozbudowy istniejących). W trakcie budowy koncepcji maszyny wnioskującej modelu referencyjnego dokonano analizy wielu dostawców narzędzi. Kierowano się wynikami raportów grupy analitycznej i doradczej Gartner'a oraz Ovum z lat 2008, 2010 i 2012 wskazujących na kierunki rozwoju środowisk wytwórczych dla wsparcia cyklu wytwarzania oprogramowania [9]. To wyniki tych analiz wskazywały na potrzebę uwzględnienia proponowanej w pracy koncepcji (budowy modelu referencyjnego i wsparcia jego wykorzystania maszyną wnioskującą) produktów firmy IBM Rational. Dokonano w tym celu procesu dekompozycji funkcjonalności produktów IBM Rational (wspierających procesy pozyskiwania wymagań) dla potrzeb modelu referencyjnego.

Dekompozycja została dokonana dzięki szczegółowej analizie tych narzędzi oraz bezpośredniej komunikacji autorów z osobami odpowiedzialnymi za produkty (ang. Product Manager) ze strony firmy IBM. Proces dekompozycji narzędzi do pozyskiwania wymagań ukazał ograniczenia budowanego modelu referencyjnego już na początku badań. Bazując tylko na dekompozycji funkcjonalnej narzędzi niemożliwe staje się ocenienie ich wpływu dla organizacji dostawcy usług IT. W takim ujęciu budowa modelu referencyjnego uniezależnia funkcjonalności narzędzi od ustalonego procesu formalnego lub

zwinnego zarządzania organizacją. W takim ujęciu model referencyjny może być stosowany zarówno dla organizacji stosujących metodyki ciężkie dobierając większą pulę funkcjonalności lub też kierujących się w stronę metodyk lekkich (Scrum, EclipseWay) [2]. Można go zatem stosować w każdej organizacji niezależnie od wdrożonego procesu, kierując się jedynie oczekiwaniami jakie posiada organizacja względem procesów doboru/rozwoju wykorzystywanych narzędzi.

W budowie modelu referencyjnego autorzy zdecydowali się zastosować koncepcje perspektyw architektonicznych. Zastosowanie perspektyw pozwoliło na łatwiejsze zarządzanie (także z perspektywy interesariuszy) wielowarstwową architekturą modelu referencyjnego. Wykorzystano w tym celu koncepcję opracowaną przez P. Kruchtena [8]. W ramach tej koncepcji wprowadza 5 perspektyw (ang. 4+1 Architectural View Model), które dotykają różnych widoków (ang. views) oraz punktów widzenia (ang. points of views) budowanego modelu referencyjnego. Opracowano perspektywy: funkcjonalną, procesów biznesowych, logiki narzędzi, infrastruktury organizacji oraz struktury wewnętrznej narzędzi informatycznych. Zastosowanie perspektyw do budowy modelu referencyjnego pozwala na współpracę (w procesie rozwoju narzędzi informatycznych) wielu interesariuszy projektu. Pozwala także na wyodrębnienie tych składowych perspektyw/perspektyw, które mają największe znaczenie w procesie budowy modelu referencyjnego. Struktura proponowanego w pracy modelu referencyjnego ma charakter dynamiczny, pozwala na dowolny dobór perspektyw, dzięki czemu można uzyskać mniej lub bardziej szczegółowy proces doboru/rozwoju narzędzi informatycznych.



Rys. 1.1. Struktura modelu referencyjnego dla doboru/rozwoju narzędzi w cyklu wytwarzania oprogramowania.

Do specyfikacji i implementacji zaproponowanych perspektyw w modelu referencyjnym zostały zastosowane elementy notacji języka modelowania

UML 2.4 w postaci diagramów przypadków użycia, wdrożenia, pakietów oraz komponentów.

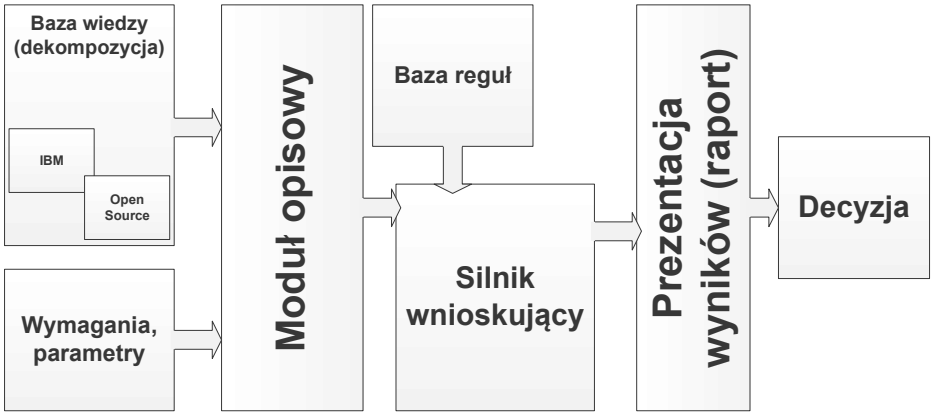
Dysponując modelem referencyjnym doboru/rozwoju narzędzi dla wsparcia procesu wytwarzania oprogramowania, należy stosować (w przypadku znacznej liczby funkcjonalności dekomponowanych w wyodrębnionych pięciu perspektywach) operację wnioskowania i prezentowania pewnych zależności pomiędzy elementami modelu. Wydaje się, że opracowanie reguł wnioskowania może decydować o przydatności opracowanego modelu referencyjnego. Nie wydaje się, aby analityk odpowiedzialny za dobór/rozwój narzędzi Open Source w przypadku znacznej liczby funkcjonalności przeszukiwał odpowiednie perspektywy i kojarzył je ze sobą.

Autorzy opracowali dwa przykłady wnioskowania wprost wykorzystując opracowany model referencyjny doboru/rozwoju narzędzi. Opracowana maszyna wnioskująca pozwoliła na weryfikację modelu referencyjnego. Opracowana wersja maszyny wnioskującej nie jest aktualnie produktem mogącym wspomóc firmę w podejmowaniu decyzji. Aby móc wykorzystywać maszynę wnioskującą, jako element wspomagający decyzje, musi ona być wyposażona w zaawansowany algorytm wnioskujący. Do sprecyzowania algorytmu wnioskującego należałoby sprecyzować elementy wejściowe i wyjściowe algorytmu. W przypadku wspomagania decyzji doboru narzędzi Open Source danymi wejściowymi są między innymi elementy inżynierii wymagań stawiane przy wytwarzaniu produktu przez użytkownika korzystającego z maszyny wnioskującej, jak również możliwości integracji aktualnie wykorzystywanych narzędzi, możliwości firmy w zakresie zasobów, doświadczenia, itp. Natomiast danymi wyjściowymi są propozycje funkcjonalności lub narzędzi spełniających te funkcjonalności, które najlepiej odpowiadają stawianym wymaganiom. Proces doboru wymagań dla maszyny wnioskującej jest jednym z etapów badawczych autorów. Trenowanie algorytmów wnioskujących powinno opierać się na wnioskach płynących z wytwarzania rzeczywistych projektów w dużych firmach. Autorzy dysponują możliwością przetestowania maszyny właśnie w takim środowisku. Autorzy dysponując doświadczeniem w wytwarzaniu oprogramowania chcą rozbudować maszynę wnioskującą, implementując różne algorytmy wnioskowania i badając ich przydatność w procesie wnioskowania.

1.2. Model doboru narzędzi w procesie wytwarzania oprogramowania

Na rysunku 1.2. przedstawiono koncepcję proponowanej w pracy maszyny wnioskującej dla modelu referencyjnego doboru/rozwoju narzędzi informatycznych. Moduł bazy wiedzy (lewa strona rysunku) zawiera zdekomponowane funkcjonalności zarówno narzędzi Open Source jak i narzędzia wzorcowego (w analizowanym przypadku IBM Rational) wspierających proces wytwarzania oprogramowania.

Moduł wymagań zawiera wszystkie parametry wejściowe dla silnika wnioskującego: wykorzystywane w organizacji dostawcy usług IT narzędzia informatyczne, stosowane metodyki zarządzania organizacją, zasoby informatyczne, infrastruktura oraz wymagania funkcjonalne odnośnie wytwarzanego w organizacji oprogramowania, itp. Moduł wymagań został sparametryzowany (dla wprowadzania danych o dowolnych organizacjach). Autorzy zebrali wiedzę na temat parametrów posiłkując się ankietami wykorzystywanymi do zbierania wymagań o organizacjach dostawcy usług IT.



Rys. 1.2. Koncepcja maszyny wnioskującej w oparciu o model referencyjny narzędzi.

Moduł opisowy jest elementem będącym przedmiotem aktualnych badań autorów rozdziału. Każdy z parametrów wejściowych/wyjściowych został opisany w sposób jakościowy ze względu na sposób wykorzystania tych parametrów przez silnik wnioskujący. W analizowanych przez autorów przypadkach autorzy do wytwarzania/modyfikacji modułu opisowego wykorzystali

program Java oraz raporty Birt (ang. Business Intelligence and Reporting Tools) oraz mechanizmy wnioskowania wprost. W badanym przypadku moduł jakościowy nie odgrywał dużej wartości, ponieważ relacje między danymi bazy wiedzy nie są brane pod uwagę. Autorzy planują w modelu bardziej zaawansowaną implementację algorytmów wnioskowania (rozmywanie wartości, algorytmy SI). Planują w takim przypadku wzięcie pod uwagę np. stopnia przynależności funkcjonalności narzędzia Open Source (x1) ze stopniem przynależności funkcjonalności y1 narzędzia wzorcowego. Innymi słowy należy określić czy stopień przynależności funkcjonalności x1 jest tożsamy ze stopniem funkcjonalności y1 lub też, w jakim stopniu te funkcjonalności (w oparciu o analizę stopni przynależności) są tożsame (1).

$$(\forall x \in X)(\exists y \in Y)(x \cong y) \tag{1}$$

Moduł opisowy zawiera także język zapytań dla silnika wnioskującego. Autorzy przyjęli zasadę generowania języka zapytań na podobnej zasadzie jaka panuje w środowiskach bazodanowych (język SQL wykonuje zapytanie na zbiorze danych – baza wiedzy, parametry we/wy). Baza reguł tworzona jest przez eksperta.

Zadaniem silnika wnioskującego jest wnioskowanie na podstawie parametrów z bazy wiedzy, modułu parametrów oraz języka zapytań z bazy reguł. Działanie silnika wnioskującego poddawane jest opisowi jakościowemu (transformacji) dla modułu prezentacji danych. Przedstawiona na rysunku 1.2 koncepcja działania maszyny wnioskującej nie zakłada poddania danych opisowi jakościowemu w silniku wnioskującym.

Wynik działania silnika wnioskującego jest prezentowany w postaci raportu (dokumentu PDF/Word). Raport ten zawiera zestawienie szukanych i wymaganych parametrów narzędzi informatycznego oraz propozycję doboru wymaganych funkcjonalności. W oparciu o ten raport analityk może podejmować decyzje dotyczące rozwoju/doboru narzędzi informatycznych. W przypadku, jeżeli wynik działania maszyny wnioskującej nie jest zadowalający, analityk może dokonać zmian parametrów lub reguł w bazie reguł i rozpocząć proces wnioskowania ponownie.

1.3. Implementacja przykładowego modelu

Celem weryfikacji proponowanego rozwiązania (zastosowania maszyny wnioskującej do analizy modelu referencyjnego) autorzy wykorzystali narzędzia Birt [7] i Java [11] do implementacji reguł wnioskowania w silniku maszyny wnioskującej.

Birt jest narzędziem Open Source pozwalającym na przetwarzanie zbiorów danych (np. w postaci XML) na ich reprezentację graficzną w postaci raportu. W badanym przypadku autorzy stworzyli kilka raportów (bazujących na oddzielnych plikach XML) pozwalających na pozyskiwanie informacji o funkcjonalnościach udostępnianych przez brane pod uwagę przez organizację narzędzie oraz jego kwalifikację, jako zdolnego do rozwoju. Analiza przykładów doboru/ rozwoju narzędzi informatycznych w oparciu o opracowaną koncepcje maszyny wnioskującej i jej implementację w oparciu o narzędzie Birt potwierdziła możliwość wnioskowania w oparciu o model referencyjny.

Kolejnym przykładem implementacji było napisanie programu w języku Java. Autorzy stworzyli mechanizm wnioskowania identyczny jak w przypadku narzędzia Birt. Takie podejście miało na celu potwierdzenie możliwości budowania maszyny wnioskującej niezależnie od technologii. Program w języku Java pobiera parametry narzędzi informatycznych z modelu referencyjnego i traktuje je, jako parametry wejściowe dla maszyny wnioskującej. Następnie zwraca wynik w postaci sugerowanych funkcjonalności dla rozwoju istniejących w organizacji dostawcy usług IT narzędzi.

W analizowanym przypadku stworzono prosty silnik wnioskujący. Takie podejście miało na celu sprawdzenie czy możliwe jest w przyszłości budowanie maszyny wnioskującej w oparciu o model referencyjny oraz o zaawansowane algorytmy wnioskowania.

1.4. Opracowanie reguł wnioskowania

Proponowana koncepcja wnioskowania (na potrzeby budowy bazy reguł maszyny wnioskującej) polega na analizie relacji pomiędzy funkcjonalnościami narzędzi Open Source, a funkcjonalnościami pochodzącymi z dekompozycji narzędzi wzorcowych zawartych w modelu referencyjnym. Na tej podstawie tworzone jest zapytanie do bazy wiedzy, w wyniku, którego uruchamiany

jest proces wnioskowania w przód (czy zachodzi relacja pomiędzy funkcjonalnościami). W oparciu o opracowany proces wnioskowania maszyna wnioskująca dostarcza raportów w oparciu o jeden z czterech zbiorów:

- funkcjonalności wspierane przez narzędzie – zbiór 1
$$y \in Y \quad (2)$$
- funkcjonalności wymagane przez klienta – zbiór 2
$$x \in X \quad (3)$$
- funkcjonalności wymagane przez klienta i wspierane przez narzędzie – zbiór 3
$$X \cap Y = \{x : x \in X \wedge x \in Y\} \quad (4)$$
- funkcjonalności wymagane przez klienta i niewspierane przez narzędzie – zbiór 4
$$x \neq y \rightarrow X \cap Y = \emptyset \quad (5)$$

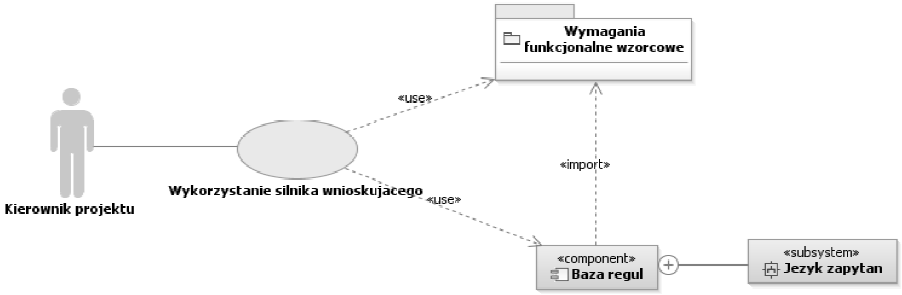
Następnie maszyna wnioskująca dokonuje losowej klasyfikacji wykorzystywanego w organizacji dostawcy usług IT narzędzia na trzy grupy:

- **A** – brak możliwości rozwoju narzędzia o funkcjonalności ze zbioru 4
- **B** – możliwość rozwoju tylko jednej funkcjonalności ze zbioru 4
- **C** – możliwość rozwoju narzędzia o wszystkie funkcjonalności ze zbioru 4

Bazując na wnioskowaniu w przód autorzy ocenili, że wynik działania maszyny wnioskującej będzie poprawny dla ściśle sprecyzowanego zapytania (typ i parametry). Okazało się, że wynik działania maszyny wnioskującej, na podstawie zrealizowanych dwóch sesji z maszyną wnioskującą, dostarcza jedynie ogólnej informacji na temat relacji w bazie wiedzy (dostępnych funkcjonalności). Istotnym czynnikiem okazał się brak informacji o wszystkich narzędziach wspierających cykl wytwarzania oprogramowania (w analizowanym przypadku wsparcia procesu wymagań). Dlatego też ciągle trwają prace mające na celu dekompozycje narzędzi Open Source dla uzupełnianie modelu referencyjnego. Napotkano także problemy przetwarzania w środowisku Birt podczas tworzenia reguł. Okazało się, że zbiór funkcjonalności środowiska Birt nie posiada możliwości przeszukiwania referencyjnego po dodaniu nowych funkcjonalności do modelu referencyjnego.

Mając na uwadze powyższe ograniczenia wnioskowania (przy zastosowaniu technologii Birt) autorzy planują zaimplementowanie mechanizmów sztucznej inteligencji w maszynie wnioskującej do tworzenia modułu opiso-

wego dla wszystkich parametrów silnika wnioskującego. Należy także zwrócić uwagę, że w przeprowadzonym badaniu dekompozycja funkcjonalności elementów bazy wiedzy i na tej podstawie tworzenie relacji w bazie wiedzy opiera się na wiedzy eksperta. Oznacza to że jakościowe porównanie funkcjonalności wzorcowej z dekomponowaną leży także po stronie eksperta wiedzy.



Rys. 1.3. Schemat wykorzystania języka zapytań jako parametru wejściowego dla bazy reguł maszyny wnioskującej.

Stwierdzono także w przeprowadzonych badaniach, że moduł opisowy powinien jednoznacznie opisywać zarówno parametry wymagań, jak i reguły bazy wiedzy. Wykorzystanie przez autorów np. logiki rozmytej spowoduje opisanie w sposób matematyczny parametrów silnika wnioskującego (uporządkuje tę jednoznaczność). Stąd też prowadzone prace badawcze skupiają się na próbie implementacji reguł wnioskowania w postaci języka zapytań i traktowania je na potrzeby maszyny wnioskującej, jako parametru wejściowego. W takim podejściu język zapytań mógłby być generowany np. przez zewnętrzny system ekspertowy.

1.5. Weryfikacja metody w środowisku projektowym

Działanie maszyny wnioskującej zostało zweryfikowane przez porównanie wyników jej działania z wynikami diagramów RSA (ang. Rational Software Architect) Topic Diagram [5] oraz Browse Diagram [6]. W procesach weryfikacji zastosowano aplikację języka JAVA oraz raporty opracowane w oparciu o Birt. W obu przypadkach na podstawie danych dotyczących narzędzi informatycznych organizacji dostawcy usług IT dobierano funkcjonalno-

ści/nowe narzędzia. W przypadku diagramów RSA na potrzeby prezentacji zastosowania modelu referencyjnego zostały opracowane raporty:

- Porównanie funkcjonalne
- Implementacja wyznaczonych funkcjonalności

Analizując otrzymane wyniki udało się zaobserwować zbieżność na poziomach funkcjonalnych oraz integracyjnych. Nie analizowano kosztów rozwoju narzędzi, które także mogą odgrywać znaczenie przy doborze funkcjonalności. Autorzy przyjęli, że z perspektywy możliwości rozwoju oraz realizacji potrzeb biznesowych, mają one znaczenie drugorzędne.

Zaproponowana metoda zostanie przetestowana w środowisku dużych firm działających w branży IT. Równocześnie trwają prace nad uzupełnianiem bazy wiedzy o kolejne narzędzia Open Source i dekompozycję ich funkcjonalności.

1.6. Podsumowanie

W rozdziale zaprezentowano koncepcję maszyny wnioskującej dla doboru oraz rozwoju narzędzi Open Source w oparciu o model referencyjny dla dyscypliny zarządzania wymaganiami dla potrzeb organizacji dostawcy usług IT. Przedstawiono aplikacje wzorcowe – spełniające wymagania i potrzeby klientów. Za takie wzorcowe rozwiązania autorzy uznali narzędzia IBM (na podstawie wyników raportów firm konsultingowych).

W ramach prowadzonych badań dokonano implementacji prostych reguł wnioskowania wprzód na podstawie relacji w opracowanym modelu referencyjnym rozwoju/doboru narzędzi informatycznych. Wynikiem prowadzonych prac jest stworzenie silnika wnioskującego w maszynie wnioskującej w oparciu o mechanizmy raportów Birt oraz opracowanego na potrzeby maszyny wnioskującej aplikacji w technologii Java. Opracowanie maszyny wnioskującej pozwoliło na weryfikację przydatności modelu referencyjnego. Konstrukcja maszyny wnioskującej ma charakter modułowy, dzięki czemu może być dowolnie dostosowywana dla potrzeb dowolnych metod wnioskowania.

Rozdział przedstawia ogólną koncepcję zastosowania maszyny wnioskującej dla doboru i rozwoju narzędzi wspierających procesy wytwarzania

oprogramowania. Autorzy bazując na przykładach zastosowania maszyny, wykazali jej przydatność w praktyce oraz możliwości jej oceny.

Prowadzone przez autorów analizy wykazały także, że istnieje możliwość zastosowania innych mechanizmów wnioskowania w maszynie wnioskującej. Autorzy sądzą, że takie podejście przyczyni się do bardziej precyzyjnego określania kierunku rozwoju narzędzi Open Source, a co za tym idzie usprawni proces wyznaczania systemów informatycznych przez organizacje dostawcy usług IT.

1.7. Bibliografia

1. Broy M.: *Requirements engineering as a key to holistic software quality*, Proceedings of 21th International Symposium on Computer and Information Sciences, 2006.
2. Chrabski B.: *Zwinne wytwarzanie oprogramowania – praktyki wspierane w środowisku IBM Jazz*, Integracja systemów informatycznych – nowe wyzwania, red. Zdzisław Kowalczuk, Janusz Górski, Cezary Orłowski. - Gdańsk: Pomorskie Wydawnictwo Naukowo-Techniczne, 2011.
3. Chrabski B.: *Integration and Dependency in Software Lifecycle based on Jazz platform*, Problems of Dependability and Modelling. Monographs of System Dependability, red Mazurkiewicz J., Sugier J., Walkowiak T., Michalska K, Oficyna Wydawnicza Politechniki Wrocławskiej. Wrocław 2011.
4. IBM Rational Innovate: https://jazz.ideajam.net 2012.
5. IBM, Rational Software Center: http://publib.boulder.ibm.com/infocenter/radhelp/v7r0m0/index.jsp?topic=/com.ibm.xtools.topicbrowse.doc/topics/ctopic.html 2013.
6. IBM, Rational Software Center: http://publib.boulder.ibm.com/infocenter/rsdvhelp/v6r0m1/index.jsp?topic=%2Fcom.ibm.xtools.modeler.doc%2Ftopics%2Fcbrowsd_m.html 2013.
7. Hague N., Tatchell J.: *BIRT: A Field Guide to Reporting (2nd ed.)*, Addison-Wesley Professional. p. 794, 2008.

8. Kruchten P.: *Architectural Blueprints - The "4+1" View Model of Software Achitecture*, IEEE Software 12 (6), 1995.
9. Ovum, Ovum Application Lifecycle Management Report: http://www.rationalindia.in/ibm-rational-the-recognized-leader-in-alm 2006.
10. Wiegers K. E.: *More about Software Requirements*, Microsoft Press, 2006
11. Oracle: http://www.oracle.com/technetwork/java/javase/overview/index.html 2013.
12. Orłowski C., Chrabski B., Szczygielski Ł., Szczygielski J., Madej P.: *Integracja narzędzi do zarządzania wymaganiami oraz zarządzania projektem na przykładzie IBM Rational Requirements Composer i Team Concert,* Integracja systemów informatycznych — Nowe wyzwania / eds. Janusz Górski, Cezary Orłowski. - Gdańsk: Pomorskie Wydawnictwo Naukowo-Techniczne, 2011.
13. Ziółkowski A., Chrabski B., Szwarczewska A.: *Integracja narzędzi do zarządzania wymaganiami oraz modelowania na przykładzie IBM Rational Requirements Composer i Rational Software Architect*, Integracja systemów informatycznych — Nowe wyzwania / eds. Janusz Górski, Cezary Orłowski. - Gdańsk: Pomorskie Wydawnictwo Naukowo-Techniczne, 2011.

Rozdział 2

Wzorzec DCI

W niniejszym rozdziale przedstawiono wzorzec DCI (ang. Data, Context, Interaction). Opisano najważniejsze założenia architektury tego wzorca i jego elementy składowe. Wzorzec DCI zastosowano praktycznie w przykładowej aplikacji. Przestawiono wady i zalety zastosowania DCI w implementacji aplikacji.

Wzorce projektowe zostały wprowadzone w architekturze w latach siedemdziesiątych przez Alexandra [1]: „*wzorzec opisuje problem, który ciągle pojawia się w naszej dziedzinie, a następnie określa zasadniczą część jego rozwiązania w taki sposób, by można było zastosować je nawet milion razy za każdym razem w nieco inny sposób*". Wprowadzenie do projektowania obiektowego zasad i strategii opartych na wzorcach projektowych zaproponowali: Gamma, Helm, Johnson, Vlissides, zwani "Bandą Czworga" [2]. Stwierdzili oni, iż tworząc programy spotykamy się z problemami, które rozwiązać można w podobny sposób. Wykorzystanie istniejących, wcześniej sprawdzonych wzorców, przyspiesza pracę nad projektem i pozwala uniknąć błędów gdyż nie musimy wymyślać rozwiązań typowych problemów. Autorzy - zaproponowali sposób opisu wzorców a także skatalogowali wiele wzorców projektowych. Opisy wielu wzorców projektowych można znaleźć w [3]. Wzorce te, na przestrzeni lat na stałe wpisały się w kanony programowania obiektowego.

DCI (ang. *Data – Context – Interaction*) jest wzorcem programowania obiektowego opracowanym przez norweskiego profesora Trygve Reenskauga [4] w 2008 roku. Jest to efekt wieloletnich prac autora nad projektem Baby-UML [5]. DCI [6,7,8,9] umożliwia modelowanie i implementację systemu informatycznego w oparciu o role, które mogą być przypisywane dynamicznie do obiektów w czasie wykonania. Od momentu swojej oficjalnej publikacji w

2008 roku, DCI zdobywa coraz większą popularność, szczególnie w środowisku osób związanych z dynamicznymi językami programowania oraz metodyką *Agile* [6].

W niniejszym rozdziale opisano i oceniono przydatność wzorca projektowego DCI. Badanie przydatności nowego wzorca powinno być poprzedzone jego praktycznym wykorzystaniem, umożliwiającym dokonanie obiektywnej oceny i porównanie wyciągniętych wniosków z założeniami teoretycznymi. Dostępne w literaturze i publikacjach naukowych informacje odnośnie możliwości zastosowania DCI oparte były dotychczas na rozważaniach teoretycznych oraz niewielkich przykładach, na podstawie których autorzy starali się przedstawić ogólną ideę i zasadę działania wzorca. Stworzono realną aplikację DCI-Project, wykorzystująca mechanizmy i założenia DCI a opisaną w [10].

DCI-Project jest narzędziem umożliwiającym grupie osób wspólną pracę nad projektem. Mogą one tworzyć i przypisywać zadania współpracownikom, przeglądać postępy prac oraz redagować dokumenty i teksty powstałe na różnych etapach realizacji projektu. Jednym z założeń aplikacji jest przypisanie projektu do konkretnej osoby – właściciela projektu. Osoba ta jest odpowiedzialna za nadzorowanie przebiegiem prac oraz zarządza utworzonymi projektami, dokumentami, współpracownikami i zadaniami. Użytkownik systemu może także komunikować się poprzez system komentarzy, oraz obserwować interesujące go zadania czy też projekty.

W kolejnych podrozdziałach przedstawiono DCI oraz przykład wykorzystania w rzeczywistej aplikacji a także wnioski.

2.1. DCI

Reenskaug zaproponował nowe podejście do programowania obiektowego szczególnie dogodne w „zwinnych" (ang. *agile*) metodykach tworzenia oprogramowania [8]. Podczas gdy tradycyjne podejście do programowania obiektowego przedstawia obiekty świata rzeczywistego jako klasy i ich metody tworzące pojedynczy interfejs, Reenskaug proponuje podział systemu na:

- część **danych** odpowiedzialną za to „*czym system jest*" oraz
- część **zachowania i interakcji**, czyli to „*co system robi*".

W klasycznym programowaniu obiektowym funkcjonalność systemu jest rozproszona pomiędzy szeregiem klas modelu dziedziny. DCI pozwala dokonać separacji zachowania systemu na poziomie przypadków użycia. Zachowania te reprezentowane są przez **role**, które w kontekście przypadku użycia identyfikują dany obiekt i jego funkcjonalność. Role są bezstanowymi identyfikatorami, których zadaniem jest zdefiniowanie funkcji systemu. Kluczową cechą wyróżniającą DCI od innych technik programistycznych, jest fakt, iż **funkcjonalność zdefiniowana przez role dodawana jest do obiektów dynamicznie,** dokładnie w chwili, w której następuje wykonanie przypadku użycia. Takie podejście pozwala na:

- Oddzielenie kodu statycznego, od kodu dynamicznego opisującego funkcje i zachowania systemu.
- Skupienie kodu odpowiedzialnego za realizację pojedynczego przypadku użycia w jednym miejscu.
- Umożliwienie modelowania systemu informatycznego w kontekście widzianym przez użytkownika, z uwzględnieniem interakcji pomiędzy elementami systemu co jest szczególnie przydatne w „zwinnych" (ang. *agile*) metodach tworzenia.
- Zwiększenie czytelności kodu.
- Uproszczenie modelu danych.

2.1.1. Architektura DCI

Celem DCI jest separacja statycznego kodu opisującego model dziedziny od dynamicznego, opisującego funkcjonalność. W tym celu Reenskaug wydzielił trzy główne części systemu: dane, kontekst i interakcje. Każda z nich w jasny sposób określa zakres swoich obowiązków.

Dane (ang. *Data*)

Na część systemu odpowiedzialną za dane składają się podstawowe klasy modelu dziedziny. Klasy te pozbawione są szczegółowej funkcjonalności powiązanej z konkretnym przypadkiem użycia, jednak mogą zawierać implementacje metod zapewniających dostęp do właściwości obiektu czy też ogólnego zachowania, wspólnego dla całego systemu. Część danych często nazywana jest „mikro bazą danych", ponieważ zawiera jedynie klasy modelu dziedziny i powiązania pomiędzy nimi. Głównym celem takiej reprezentacji da-

nych, dzięki odseparowaniu zachowania, jest umożliwienie modelowania systemu w kontekście widzianym przez użytkownika, a nie w kontekście klas i ich interfejsów.

Kontekst (ang. *Context*)

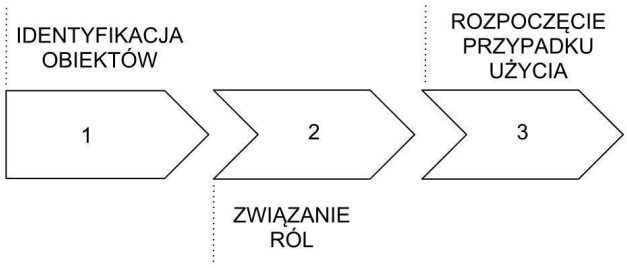
Z programistycznego punktu widzenia kontekst jest klasą lub jej instancją, która skupia kod odpowiedzialny za realizację pojedynczego przypadku użycia. Podstawowym zadaniem kontekstu jest wyróżnienie ról oraz związanie ich z obiektami biorącymi udział w przypadku użycia. Kontekst stanowi również przestrzeń nazw dla ról, które nie są samodzielnymi bytami, lecz stanowią pojęcie ściśle związane z konkretną klasą kontekstu. Role wyróżnione w klasie kontekstu często odpowiadają aktorom wyspecyfikowanym na poziomie przypadku użycia.

Interakcje (ang. *Interaction*)

Interakcje definiują zachowanie i funkcjonalność systemu, czyli „to co system robi". Na pojedynczą interakcję składają się role oraz zachodzące pomiędzy nimi relacje. Wykonanie interakcji polega na „odegraniu" ról poprzez obiekty – aktorów oraz komunikacji pomiędzy nimi prowadzącej do realizacji przypadku użycia.

2.1.2. Działanie systemu opartego na DCI

Działanie systemu opartego o mechanizmy DCI może zostać opisane jako sekwencja trzech następujących po sobie kroków, które zostały przedstawione na Rys. 2.1.

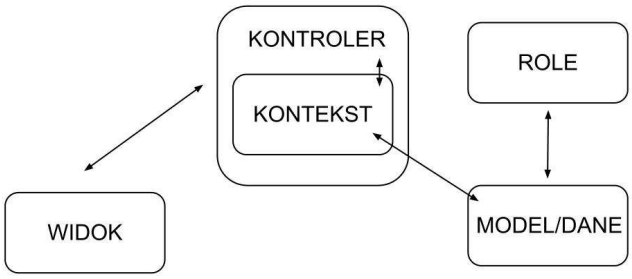


Rys. 2.1. Schemat działania systemu opartego na DCI

Pierwszym z kroków jest identyfikacja obiektów biorących udział w wykonaniu kontekstu. Identyfikacja może odbywać się wewnątrz kontekstu lub poza nim. Po identyfikacji obiektów, zadaniem kontekstu jest związanie ich z odpowiadającymi im rolami. Każda ze zdefiniowanych przez kontekst ról, może zostać przypisana do dokładnie jednego obiektu i stanowi jego identyfikator. Nie ma natomiast ograniczenia na liczbę ról przypisanych do pojedynczego obiektu. Przypadek użycia rozpoczyna się wywołaniem metody jednej z ról i dalszej komunikacji pomiędzy rolami prowadzącej do realizacji przypadku użycia.

2.2. DCI a inne wzorce

Koncepcje zbliżone do tych, które są prezentowane przez DCI są od wielu lat wykorzystywane w powszechnie znanych wzorcach. Silnie powiązany z DCI jest wzorzec projektowy Model – Widok – Kontroler [11]. Zadaniem tego wzorca jest odseparowanie modelu danych od warstwy prezentacji. W tym celu wyróżnione zostały trzy podstawowe elementy: model, widok, kontroler. Porównując oba wzorce można zauważyć, iż istnieje pomiędzy nimi pewne podobieństwo dotyczące separacji warstwy odpowiedzialnej za dane od pozostałych komponentów systemu. Jednak wzorzec Model – Widok – Kontroler nie wprowadza ograniczeń dotyczących sposobu implementacji modelu. Podstawową różnicą pomiędzy przedstawionymi technikami są zdefiniowane przez nie cele – w odróżnieniu od MVC, w DCI oddzielone są funkcje oraz zachowanie systemu od modelu danych. DCI może zostać uznany jako dopełnienie i rozszerzenie wzorca Model – Widok – Kontroler. Wydaje się być wygodne połączenie obu wzorców architektonicznych i zostało to praktycznie sprawdzone w aplikacji DCI Project [10].



Rys. 2.2. Schemat architektury systemu wykorzystującego DCI oraz MVC

Na Rys. 2.2 przedstawiony został schemat architektury systemu wykorzystujący obie techniki. Zastosowanie wzorca DCI w połączeniu z Model – Widok – Kontroler wprowadza dodatkową warstwę abstrakcji pomiędzy kontroler oraz część systemu odpowiedzialną za dane. Kontroler, podobnie jak w przypadku wzorca MVC, stanowi centralną część aplikacji oraz stanowi środowisko wykonania dla klas kontekstu. Najważniejszą różnicą, którą wprowadza przedstawiona architektura jest brak bezpośredniej komunikacji pomiędzy kontrolerem a warstwą danych – odbywa się ona jedynie za pośrednictwem kontekstu.

Delegacja [12] jest wykorzystywana przez wiele wzorców projektowych. Wyróżnia ona dwa podstawowe obiekty – odbiorcę oraz delegata, do którego kierowane są wykonania operacji. Stosowane w DCI role są koncepcyjnie zbliżone do delegatów – to one definiują dodatkową funkcję obiektu odbiorcy. Jednak pomiędzy obiektami delegatów a rolami przedstawionymi przez DCI istnieje zasadnicza różnica. Role są bezstanowe i nie istnieją jako samodzielne byty, podczas gdy „delegaci" stanowią pełnoprawne obiekty klas istniejących w systemie. Koncepcja ról w DCI rozwiązuje jeden z problemów wiążących się ze stosowaniem wzorca delegacji – zjawiska tzw. "self-schizofrenii", w której to tożsamość obiektu jest rozdzielona pomiędzy dwa obiekty – odbiorcy i delegata. W przypadku DCI wszelka "delegacja" funkcji odbywa się zawsze w kontekście obiektu powiązanego z przypisaną do niego rolą. Rozwiązanie to zapewnia, iż tożsamość obiektu jest zawsze jednoznaczna.

2.3. Przykład

Na listingu 1 przedstawiono fragment aplikacji DCI-Project [10], napisanej w języku Ruby [13] oraz wykorzystującej szkielet aplikacyjny *Ruby on Rails* [14]. Klasa *CreateIssueContext* implementuje przypadek użycia odpowiedzialny za tworzenie nowego zadania w systemie zarządzania projektami. Aktorem biorącym udział w przypadku użycia jest osoba zgłaszająca, która opisuje zadanie oraz wybiera osobę do niego przypisaną. Po utworzeniu zadania zarówno osoba zgłaszająca jak i osoba, która została do niego przypisana – jako osoby powiązane z zadaniem stają się jego obserwatorami. Dla podanego przypadku użycia wyróżnione zostały trzy **role**, które odpowiadają kolejno osobie zgłaszającej (*Reporter*), osobie przypisywanej do realizacji zadania (*Assignee*) oraz osobom związanym z zadaniem (*IssueCollabolators*). Zdefiniowano trzy role: *Reporter*, *Assignee* oraz *IssueCollabolators* (linie 17-38). Role te, zaimplementowane jako osobne moduły (*Reporter* – od linii 17, *Assignee* – od linii 27 oraz *IssueCollabolators* – od linii 30), definiują funkcje specyficzne dla tego przypadku użycia. Powiązanie ról z odpowiednimi obiektami odbywa się w metodzie *initialize* (początek w linii 3) – nazwy ról stają się identyfikatorami powiązanych obiektów oraz udostępniają interfejs do zdefiniowanych przez siebie funkcji. Role definiowane na poziomie kontekstu mogą być powiązane z pojedynczym obiektem lub z kolekcja obiektów jako całością – role *Reporter* oraz *Assignee* związane są z pojedynczymi obiektami, natomiast rola *IssueCollabolators* została związana z kolekcją dwóch obiektów (linia 6). Role oraz definiowane przez nie funkcje związane są zawsze z pewnym kontekstem i nie istnieją jako samodzielne jednostki. Oznacza, to iż nie istnieje możliwość „wydzielenia" roli czy też samych definiowanych przez nią funkcjonalności do fragmentu kodu, który mógłby być wykorzystany przez inny kontekst.

```
1. class CreateIssueContext
2.  include Context
3.  def initialize(reporter, assignee, params)
4.    role Reporter, reporter
5.    role Assignee, assignee
6.    role IssueCollabolators, [reporter, assignee]
7.    @project_id = params[:project_id]
```

```
8.    @params = params[:issue]
9.  end
10.  def execute
11.    interaction do
12.      issue = Reporter.create_issue(@project_id, @params)
13.      IssueCollabolators.watch_issue(issue.id)
14.      Response.new(issue)
15.    end
16.  end
17.  module Reporter
18.    extend Role
19.    class << self
20.      def create_issue(project_id, params)
21.      project = Project.find(project_id)
22.  issue  =  project.issues.create(params.merge(:reporter_id  =>
id))
23.       issue
24.      end
25.    end
26.  end
27.  module Assignee
28.    extend Role
29.  end
30.  module IssueCollabolators
31.    extend Role
32.    class << self
33.    def watch_issue(issue_id)
34.    context.roles[self].uniq.each do |collabolator|
35.    WatchIssueContext.new(collabolator, issue_id).execute
36.    end
37.   end
38.  end
39. end
```

Listing 1. Fragment aplikacji wykorzystującej DCI

Ważny element DCI – **interakcja**, jest przedstawiona w metodzie *execute*. Wewnątrz bloku *interaction* (linie 11-15) następuje wykonanie przypad-

ku użycia polegające na wywołaniu metod ról. Należy zaznaczyć, iż **metody te dodawane są do obiektów** – aktorów **jedynie na czas wywołania interakcji i nie są dostępne poza jej zasięgiem**. Zdefiniowane w aplikacji **konteksty** stanowią samodzielne i niezależne jednostki kodu i mogą zostać ponownie wykorzystane w innych częściach aplikacji. Przykładem jest kontekst *WatchIssueContext* (linia 35), który został zagnieżdżony w metodzie roli *watch_issue* (od linii 33). Konteksty, podobnie jak przypadki użycia, mogą pozostawać pomiędzy sobą w relacjach. Przedstawiony na listingu 1 przykład ilustruje możliwość implementacji relacji zawierania dla dwóch przypadków użycia – *CreateIssueContext* oraz *WatchIssueContext*. Kontekst *WatchIssueContext*, który istnieje również jako samodzielny przypadek użycia, jest wywoływany wewnątrz kontekstu *CreateIssueContext*.

2.4. Podsumowanie

DCI jest stosunkowo nową techniką programistyczną, jednak oferuje szereg nowatorskich rozwiązań w obszarze programowania obiektowego. Najważniejszymi cechami są: oddzielenie statycznego kodu modelu dziedziny od kodu opisującego zachowanie i funkcjonalność systemu, skupienie zachowania i funkcji systemu wewnątrz ról dynamicznie przypisywanych do obiektów, uproszczenie modelu dziedziny, komplementarność i łatwość zastosowania ze wzorcem Model – Widok – Kontroler.

Podstawową zaletą wzorca architektonicznego DCI jest zwiększenie czytelności kodu. Poprzez wprowadzenie kontekstów, kod odpowiedzialny za realizację pojedynczego przypadku użycia czy też algorytmu znajduje się w jednym miejscu, co znacząco ułatwia jego analizę. Wydzielenie funkcji i zachowania systemu do dynamicznie przypisywanych do obiektów ról upraszcza klasy modelu dziedziny, są one łatwiejsze w utrzymaniu i testowaniu.

DCI ma także słabe strony. Podział systemu na trzy części: dane, kontekst oraz interakcje, wprowadza szereg zmian w projektowaniu oraz implementacji systemu. Stosowanie DCI w istniejących systemach informatycznych wiąże się z koniecznością zmian w architekturze aplikacji, co jest czasochłonne i kosztowne. Kolejnym problemem, jest trudność realizacji DCI w niektórych językach programowania. Łatwo go zrealizować w języku *Ruby*, trudno

w językach programowania nie zezwalających na dynamiczne wiązanie ról z obiektami. Języki takie wymagają zastosowania specjalnych narzędzi: np. *Qi4j*[15] dla języka *Java*.

Poprzez podział systemu na niezależne jednostki kodu, DCI wymusza konsekwentne stosowanie dobrych praktyk programistycznych. Jednym z wielu założeń stawianych przed paradygmatem DCI jest umożliwienie programistom komunikacji z użytkownikiem, poprzez wyrażenie funkcji systemu jako ról, które musi on spełniać. Kluczowe jest modelowanie w sekwencji kroków prowadzących do pewnego celu, bardzo naturalne dla użytkownika końcowego. DCI ułatwia przejście pomiędzy przypadkami użycia i wymaganiami użytkownika, a algorytmami i kodem aplikacji. Podział kodu na dane, kontekst i interakcje zapewnia wysoki stopień oddzielenia kodu podlegającego intensywnym zmianom, od kodu statycznego. Zachowana jest jednocześnie możliwość łatwego testowania i dodawania nowych funkcji. DCI może być rozwiązaniem wielu problemów związanych z nadmiernym rozproszeniem kodu algorytmu i redundantnym kodem w klasach modelu dziedziny. Umieszczenie kodu algorytmów w jednym miejscu – odpowiedniej klasie kontekstu ułatwia jego analizę i weryfikację.

2.5. Bibliografia

1. Alexander C., Ishikawa S., Silverstein M.: *A Pattern Language*, New York, Oxford University Press, 1977
2. Gamma E., Helm R., Johnson R., Vlissides J.: *Design Patterns: Elements of Reusable Software*, Reading, Mass., Addison-Wesley, 1995
3. Shaloway A., Trott J.R.: *Projektowanie zorientowane obiektowo – wzorce projektowe*, Helion 2001
4. Reenskaug T.: http://heim.ifi.uio.no/trygver/ (dostęp 2013)
5. http://heim.ifi.uio.no/trygver/themes/babyuml/babyuml-index.html (dostęp 2012)
6. *The DCI Architecture: A New Vision of Object-Oriented Programming*, http://www.artima.com/articles/dci_vision.html (dostęp 2012)
7. http://en.wikipedia.org/wiki/Data,_context_and_interaction (dostęp 2012)

8. Coplien J.O, Bjørnvig G.: *Lean Architecture for Agile Software Development*, Wiley, 2010
9. *DCI – Data Context Interaction*, http://fulloo.info/ (dostęp 2012)
10. Stepień A.: *DCI – Data Context Interaction – ocena przydatności*, praca inżynierska, Instytut Informatyki PW, luty 2013
11. MVC: http://en.wikipedia.org/wiki/Model-view-controller (dostęp 2012)
12. http://en.wikipedia.org/wiki/Delegation_pattern (dostęp 2013)
13. Ruby: http://www.ruby-lang.org/ (dostęp 2013)
14. Ruby on Rails: http://rubyonrails.org/ (dostęp 2013)
15. Qi4j, http://qi4j.org/ (dostęp 2012)

Rozdział 3

Wykorzystanie zaawansowanych mechanizmów języka Objective-C do tworzenia łatwych w integracji komponentów

Rozwój języków programowania daje programistom ciągle nowe możliwości oraz mechanizmy pozwalające na wygodniejsze i łatwiejsze tworzenie aplikacji. W niniejszym rozdziale przedstawiona została dynamiczna charakterystyka języka Objective-C oraz przykład wykorzystania możliwości jakie daje programistom. Zaawansowane mechanizmy języka zostały wykorzystane do utworzenia łatwego w integracji komponentu interfejsu użytkownika, pozwalającego na dodanie do aplikacji dla systemu iOS baneru z aktualną temperaturą w Polsce. W najprostszym rozwiązaniu, integracja tego komponentu nie wymaga od programisty napisania ani jednej linii kodu. Wymagane jest jedynie dodanie do projektu klasy WeatherBanner. Przedstawione w pracy rozwiązanie może być z powodzeniem stosowane między innymi do tworzenia innych komponentów, na przykład banerów reklamowych, pasków z aktualnościami.

Zastosowanie różnego rodzaju technik inżynierii oprogramowania pozwala na szybsze tworzenie bardziej niezawodnych aplikacji. Często wiąże się to jednak z nieznacznym spadkiem wydajności aplikacji. Jedną z takich sytuacji jest wykorzystanie paradygmatu obiektowego (OOP - *Object Oriented Programming*) [1]. Wykorzystanie programowania obiektowego, abstrakcji danych oraz mechanizmów takich jak dziedziczenie, hermetyzacja czy polimorfizm znacznie ułatwia tworzenie oprogramowania. Obecnie jest to powszechnie stosowana i akceptowana metodologia wytwarzania oprogramowania. Należy jednak pamiętać, że zastosowanie OOP powoduje wystąpienie tzw. *OO performance penalty*, a więc narzutu czasowego związanego z wyko-

rzystaniem obiektowości. W początkowych latach rozwoju paradygmatu obiektowego, wydajność aplikacji obiektowych napisanych w języku Smalltalk [2] była szacowana na 5%-20% wydajności zoptymalizowanej aplikacji w języku C. Rozwój technologii obiektowych spowodował znaczne zmniejszenie tych dysproporcji, jednak całkowite zlikwidowanie narzutu związanego z obiektowością nie było możliwe.

Dynamiczny rozwój technologii i towarzyszący temu wzrost dostępności urządzeń o dużej mocy obliczeniowej sprawił, że czas pracy programisty jest znacznie droższy niż czas pracy urządzeń. Z tego powodu wykorzystanie możliwości nowoczesnych języków programowania, nawet tych powodujących pewien spadek wydajności, stało się standardem. Obecnie rzadkością jest, aby programista rezygnował z zastosowania ich w celu poprawy wydajności aplikacji.

Jedną z możliwości oferowanych między innymi przez języki wywodzące się z C, która często była krytykowana i w wielu przypadkach traktowana jako zła praktyka, jest zastosowanie dynamicznego wiązania metod, dynamicznego rzutowania, czy wykorzystania mechanizmu RTTI (*Run-Time Type Identification*). Nieumiejętnie wykorzystanie dynamicznej charakterystyki języka może prowadzić do powstania trudnych do zlokalizowania błędów pojawiających się dopiero podczas działania programu. Okazuje się jednak, że świadome i poprawne wykorzystanie dynamicznej charakterystyki nowoczesnych języków programowania daje olbrzymie możliwości.

Celem niniejszego rozdziału jest analiza możliwości jakie daje wykorzystanie dynamicznej charakterystyki języka programowania w zakresie modyfikacji i rozbudowy interfejsu użytkownika. W kolejnych podrozdziałach zaprezentowany został sposób wykorzystania tych możliwości w języku Objective-C do budowy łatwych w integracji komponentów interfejsu, których integracja z istniejącym programem, w najprostszym przypadku nie wymaga dodania ani jednej linii kodu. Dzięki wykorzystaniu mechanizmu nazywanego potocznie *method swizzling* [7] komponenty są automatycznie dołączane do każdego tworzonego kontrolera widoku, natomiast wykorzystanie kategorii [6] pozwala na ich kontrolę i dostosowanie.

Podobne podejście do problemu modyfikacji interfejsu użytkownika zostało zaprezentowane w pracy [4]. Przedstawiony w tej pracy projekt Scotty również wykorzystuje *swizzling* dostępny w języku Objective-C, jednak jest

zaimplementowany w języku Python z wykorzystaniem PyObjC. Autorzy wskazują również możliwość wykonania podobnych operacji w języku Java dla aplikacji wykorzystujących bibliotekę Swing oraz JavaScript dla aplikacji internetowych. Wykorzystanie introspekcji w języku Java pozwala na zmianę implementacji jedynie metod publicznych. W języku JavaScript podobny efekt uzyskano dzięki wykorzystaniu modelu delegacji.

Innym ciekawym przykładem wykorzystania dynamicznej charakterystyki języka Objective-C jest projekt zaprezentowany w [5]. Mechanizm kategorii oraz *swizzling* zostały tam wykorzystane do budowy narzędzia wspomagającego analizę interfejsu użytkownika. W opisanym przez autorów rozwiązaniu nowa implementacja metod dodawana jest do klasy za pomocą kategorii, a następnie zamieniana z oryginalną implementacją w automatycznie wywoływanej metodzie statycznej `load`.

3.1. Dynamiczna charakterystyka języka Objective-C

Język Objective-C z każdym dniem zyskuje na popularności. Obecnie, według wielu rankingów znajduje się w pierwszej trójce najpopularniejszych języków programowania. Jest to przykład nowoczesnego i szybko rozwijającego się języka, którego twórcy starają się z roku na rok wprowadzać nowe rozwiązania będące odpowiedzią na zmieniające się potrzeby programistów.

Jedną z charakterystycznych cech Objective-C [3] jest to, że odkłada moment podjęcia decyzji na jak najpóźniej wszędzie, gdzie tylko jest to możliwe. Nie są one podejmowane podczas kompilacji, ani podczas linkowania, a dopiero w czasie działania programu. Wszystkie możliwe operacje są wykonywane dynamicznie. Oznacza to, że język Objective-C wymaga do działania nie tylko kompilatora, ale również odpowiedniego środowiska uruchomieniowego, które wykona skompilowany kod. Środowisko to pełni funkcję pewnego rodzaju systemu operacyjnego dla języka Objective-C.

Programy w języku Objective-C współpracują z środowiskiem uruchomieniowym na trzech poziomach:

1. bezpośrednio poprzez kod Objective-C,
2. poprzez metody zdefiniowane w klasie `NSObject`,

3. poprzez bezpośrednie wywołanie funkcji środowiska uruchomieniowego.

Pierwsza z tych metod jest realizowana w sposób automatyczny – programista jedynie pisze i kompiluje kod źródłowy. Podczas kompilacji kodu zawierającego klasy i metody Objective-C, kompilator tworzy struktury danych
i wywołania funkcji, które wykorzystują dynamiczną charakterystykę języka.

Drugi poziom to wykorzystanie metod zdefiniowanych w klasie `NSObject`. Znaczna większość obiektów w środowiskach Cocoa oraz Cocoa Touch wykorzystywanych do tworzenia aplikacji dla MacOS oraz iOS, to obiekty klas dziedziczących po `NSObject`. Obiekty te dziedziczą więc metody klasy `NSObject`. Niektóre z tych metod proszą system uruchomieniowy o podanie informacji o obiekcie. Pozwala to na przeprowadzenie tzw. introspekcji, a więc na uzyskanie przez obiekt informacji „o sobie samym". Ta grupa metod pozwala między innymi na zidentyfikowanie klasy obiektu oraz przynależności do hierarchii dziedziczenia (`isKindOfClass:` oraz `isMemberOfClass:`). Inną często wykorzystywaną funkcją jest `respondsToSelector:`. Funkcja ta pozwala na uzyskanie informacji czy dany obiekt akceptuje określony komunikat (czy udostępnia metodę). Metoda `methodForSelector:` pozwala na uzyskanie adresu implementacji metody o podanym selektorze.

Ostatnim poziomem współpracy programów napisanych w języku Objectie-C z środowiskiem uruchomieniowym jest bezpośrednie wywołanie funkcji tego środowiska z kodu programu. Środowisko uruchomieniowe jest dynamiczną biblioteką współdzieloną posiadającą publiczny interfejs oraz udostępniającą zestaw funkcji i struktur danych w pliku nagłówkowym. Funkcje te w dużym stopniu stanowią podstawę implementacji metod udostępnianych przez klasę `NSObject`. W większości przypadków wykorzystanie tej grupy funkcji nie jest konieczne, jednak dają one olbrzymie możliwości.

3.2. Dynamiczna modyfikacja klasy i jej metod

Jedną z ciekawych możliwości jakie daje programistom język Objective-C jest dynamiczna modyfikacja ciała metod klasy, potocznie nazywana *method swizzling*. W tym celu do istniejącej klasy dodawana jest nowa meto-

da, a następnie jej ciało jest zamieniane z oryginalną implementacją innej metod. W przeciwieństwie do API introspekcji zaproponowanego w [5] do realizacji podobnego rozwiązania w języku Java, *swizzling* pozwala na zmianę implementacji wszystkich metod klasy – nie tylko publicznych.

3.2.1. Dodawanie nowych metod do istniejących klas

Podczas implementacji tego mechanizmu często wykorzystywane są kategorie, pozwalające na rozszerzenie klasy o nowe metody bez wykorzystywania w tym celu dziedziczenia. Kategorie pozwalają na zmniejszenie ilości nadmiarowego kodu oraz modyfikacji, które trzeba wprowadzić do już istniejącego kodu, ponieważ nowe metody dodawane są do już istniejących klas. Co więcej całość wykonywana jest podczas działania programu i nie jest wymagana ponowna kompilacja bibliotek.

Z punktu widzenia aplikacji, nowe metody nie różnią się niczym od metod oryginalnej klasy. Posiadają ponadto dostęp do wszystkich pól składowych rozszerzanej klasy. Wykorzystując kategorie należy pamiętać, że próbując nadpisać metodę o tej samej nazwie kompilator nie wie, którą metodę ma wywołać, co prowadzi do niejednoznaczności i żadna z metod nie zostaje wywołana. Mechanizm ten pozwala na dodanie nowych metod, jednak nie jest możliwe dodanie nowego pola klasy. Kategorie pozwalają również na podzielenie metod ze względu na ich wykorzystanie i przeznaczenie, uzyskując w ten sposób logiczne pogrupowanie funkcjonalności.

Objective-C udostępnia dwa sposoby wykorzystania kategorii. Pierwszą z nich jest dołączenie pliku nagłówkowego zawierającego implementację nowych metod. Drugą możliwością pozwalającą na dodanie metody do istniejącej klasy jest funkcja `BOOL class_addMethod(Class cls, SEL name, IMP imp, const char *types)`, która dodaje do klasy *`cls`* metodę o nazwie *`name`*, implementacji *`imp`* oraz argumentach *`types`*. Zastosowanie tej metody nie wymaga dodawania dodatkowych plików nagłówkowych.

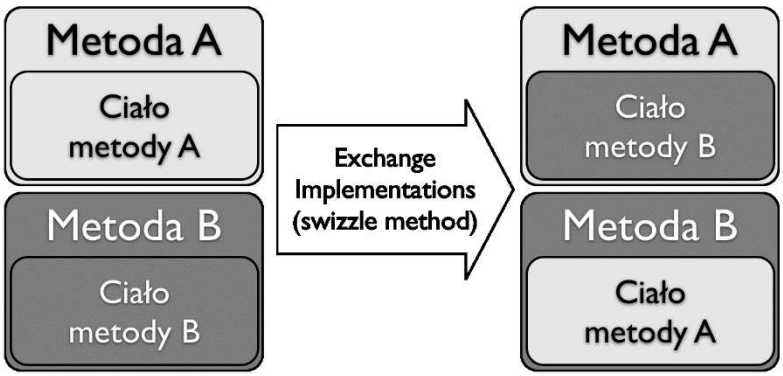
3.2.2. *Method swizzling*

W niektórych sytuacjach potrzebne jest rozszerzenie działania jednej z metod klasy lub całkowita zmiana jej zachowania. Język Objective-C pozwa-

la na dynamiczną zmianę implementacji istniejących funkcji. Umożliwia to funkcja `IMP class_replaceMethod(Class cls, SEL name, IMP imp, const char *types)`, zastępująca oryginalną implementację metody nową implementacją, często dodawaną do klasy za pomocą mechanizmu kategorii. Takie rozwiązanie pozwala na całkowite zastąpienie standardowej metody. Często jednak istnieje konieczność pozostawienia oryginalnej funkcjonalności metody i jedynie rozszerzenie jej o dodatkowe elementy. W tym celu w nowej wersji metody konieczne jest nie tylko wykonanie dodatkowych czynności, ale również wywołanie jej oryginalnej wersji. Aby taka operacja była możliwa, zamiast funkcji `class_replaceMethod`, wykorzystywana jest funkcja `void method_exchangeImplementations(Method m1, Method m2)`, pozwalająca na zamianę implementacji dwóch metod. Działanie tej funkcji zostało przedstawione na rysunku 3.1.

Osiągnięcie zamierzonego efektu możliwe jest wewnątrz ciała nowej metody (*Metoda B*) konieczne jest wywołanie metody o takiej samej nazwie (*Metoda B*).

```
- (void) metodaB{
    /*
       dodatkowe operacje
    */
    [self metodaB];
}
```



Rys. 3.1. Zamiana implementacji metod klasy

Takie wywołanie z pozoru wygląda na błędne i powodujące nieskończoną rekurencję, gdyż funkcja wywołuje samą siebie. Należy jednak pamiętać, że po wykonaniu instrukcji `method_exchangeImplementations` ciała obu funkcji zostały zamienione. Wywołanie funkcji *Metoda B* jest więc w rzeczywistości wywołaniem pierwotnej implementacji *Metoda A*.

3.3. Modyfikacja interfejsu użytkownika

W większości środowisk GUI podstawowe elementy interfejsu użytkownika, takie jak okna, widoki itp., posiadają cykl życia, definiowany przez zestaw metod wywoływanych automatycznie podczas ich tworzenia, wyświetlania czy usuwania. W Cocoa Touch jednym z takich elementów jest kontroler widoku `UIViewController`. Metodą wywoływaną po utworzeniu i załadowaniu kontrolera widoku jest `viewDidLoad`. Wykorzystanie tej metody pozwala na wykonanie dodatkowych czynności inicjujących kontroler widoku. W funkcji tej można utworzyć dodatkowe elementy interfejsu użytkownika, a następnie dodać je do głównego widoku.

W celu zaprezentowania możliwości automatycznej integracji komponentów interfejsu użytkownika, metoda `viewDidLoad` wykorzystana została do utworzenia oraz dodania do widoku banera z aktualną temperaturą w Polsce (pobraną z serwisu worldweatheronline.com). Metoda `newViewDidLoad` należąca do klasy *`WeatherBanner`* została wykorzystana do rozszerzenia standardowej implementacji `viewDidLoad` z klasy `UIViewController`. Wewnątrz tej metody tworzony i dodawany do widoku jest baner, do którego ładowana jest aktualna temperatura. W ostatniej linii metody `newViewDidLoad` wywoływana jest oryginalna wersja metody `viewDidLoad` z klasy `UIViewController`. Podstawową implementację klasy *`WeatherBanner`,* pozwalającą na automatyczne dodanie banera z aktualną temperaturą w Polsce, przedstawia listing 1.

```
      - (void)newViewDidLoad{
          WeatherBanner   *   banner   =   [[[NSBundle   mainBundle]
loadNibNamed:@"WeatherBanner" owner:nil options:nil] objectAtIndex:0];
          [[(UIViewController*)self view] addSubview: banner];
          NSData *data = [[NSData alloc] initWithContentsOfURL:[[NSURL
alloc]
```

```
initWithString:@"http://free.worldweatheronline.com/feed/weather.ashx?
q=Poland&format=json&num_of_days=1&key=***"]];
          NSError *error;
          NSDictionary       *dict       =       [NSJSONSerialization
JSONObjectWithData:data options:kNilOptions error:&error];
          NSString    *    temp    =[[[[dict    objectForKey:@"data"]
objectForKey:@"current_condition"]
objectAtIndex:0]objectForKey:@"temp_C"];
          banner.temp.text   =   [NSString   stringWithFormat:@"%@°C",
temp];
          // wywołanie oryginalnej wersji metody viewDidLoad - po za-
mianie implementacji
          [self newViewDidLoad];
      }
      + (void) swizzle {
          Method   newMethod=   class_getInstanceMethod([self   class],
@selector(newViewDidLoad));
          //dodanie nowej metody (newViewDidLoad z klasy WeatherBan-
ner) do klasy UIViewController
          class_addMethod([UIViewController                    class],
@selector(newViewDidLoad), method_getImplementation(newMethod), meth-
od_getTypeEncoding(newMethod));
          //pobranie implementacji oryginalnej i nowej metody z klasy
UIViewController
          Method origMethod= class_getInstanceMethod([UIViewController
class], @selector(viewDidLoad));
          newMethod= class_getInstanceMethod([UIViewController class],
@selector(newViewDidLoad));
          if (origMethod && newMethod){
              // zamiana implementacji metod
              method_exchangeImplementations(origMethod, newMethod);
          }
      }
```

Listing. 1. Implementacja zamiany metody viewDidLoad

Ostatnią czynnością, która musi zostać wykonana jest wywołanie metody statycznej *swizzle* klasy `WeatherBanner`. Wywołanie to może zostać wyko-

nane w dowolnym momencie działania programu. Od chwili wykonania metody *swizzle*, do każdego kontrolera widoku dodany zostanie baner.

Możliwe jest jednak całkowite wyeliminowanie konieczności pisania choćby jednej linii kodu przez programistę, który chce wykorzystać komponent w swojej aplikacji. Pozwala na to metoda statyczna `load` klasy `NSObject`. Metoda ta jest automatycznie wywoływana w momencie gdy klasa jest dodawana do środowiska uruchomieniowego Objective-C. Po zdefiniowaniu metody `load` w klasie `WeatherBanner`, do każdego kontrolera widoku dodany zostanie baner z prognozą pogody. Jedyną czynnością jaką musi wykonać programista korzystający z klasy `WeatherBanner` jest dodanie tej klasy do projektu aplikacji.

3.4. Alternatywne rozwiązania

Wykorzystanie *method swizzling* to tylko jedna z metod pozwalających na dodanie komponentu do okna aplikacji. Najprostszym, jednak posiadającym wiele ograniczeń rozwiązaniem jest utworzenie klasy dziedziczącej po klasie `UIViewController` oraz przeładowanie metody `viewDidLoad`. W metodzie klasy potomnej możliwe jest dodanie dowolnych elementów interfejsu użytkownika oraz wywołanie metody z klasy bazowej. Rozwiązanie takie posiada jednak wiele ograniczeń. Pierwszym, utrudniającym integrację komponentu z już istniejącą aplikacją jest konieczność modyfikacji kodu każdego kontrolera widoku poprzez zmianę klasy, po której dziedziczy dany kontroler widoku na klasę dołączającą komponent. Ze względu na brak wielodziedziczenia w języku Objective-C nie możliwe jest również dodanie w ten sposób kilku różnych komponentów. Rozwiązaniem tego problemu jest co prawda zastosowanie wzorca dekorator pozwalającego na wielokrotne rozszerzanie obiektu o nowe funkcjonalności lub zastosowanie mechanizmu wstrzykiwania zależności. Oba te podejścia mogą być z powodzeniem użyte podczas tworzenia aplikacji od podstaw, jednak próba zastosowania ich w istniejącej już, nieprzystosowanej do tego celu aplikacji wymaga znacznych modyfikacji kodu programu. Rozwiązanie zaprezentowane w niniejszej pracy pozwala na dodanie komponentu do ukończonego projektu bez konieczności dopisania ani modyfikowania ani jednej linii programu.

3.5. Podsumowanie

Nowoczesne języki programowania dają programistom dużo większe możliwości tworzenia komponentów oprogramowania, które mogą być w bardo prosty sposób integrowane z już istniejącymi aplikacjami. Dzięki wykorzystaniu zaawansowanych mechanizmów języka oraz jego dynamicznej charakterystyki, czasami taka integracja jest możliwa nawet bez pisania ani jednej linii kodu. Przedstawiona w niniejszej pracy metoda pozwala na tworzenie różnych komponentów interfejsu użytkownika i może być z powodzeniem stosowana nie tylko do dołączania do aplikacji baneru z prognozą pogody, ale również różnego typu pasków z aktualnościami czy banerów reklamowych.

3.6. Bibliografia

1. Apple Inc., Object-Oriented Programming with Objective-C, www.developer.apple.com, 2009
2. D. Ungar, Performance Comparison of C, SELF, and Smalltalk Implementations., 1989
3. Apple Inc.,Programming with Objective-C, www.developer.apple.com, 2009
4. James R. Eagan, Michel Beaudouin-Lafon, and Wendy E. Mackay. 2011. Cracking the cocoa nut: user interface programming at runtime. In Proceedings of the 24th annual ACM symposium on User interface software and technology (UIST '11). ACM, New York, NY, USA, 225-234.
5. Mona Erfani Joorabchi and Ali Mesbah. 2012. Reverse Engineering iOS Mobile Applications. In Proceedings of the 2012 19th Working Conference on Reverse Engineering (WCRE '12). IEEE Computer Society, Washington, DC, USA, 177-186.
6. “Categories and Extensions,” https://developer.apple.com/library/mac/#documentation/General/Conceptual/DevPedia-CocoaCore/Category.html
7. Cocoa developer community, “Method Swizzling,” http://cocadev.com/wiki/MethodSwizzling.

Rozdział 4

Zastosowanie mechanizmu profili UML w modelowaniu pokrycia architektury aplikacji przez testy

Rozdział prezentuje koncepcje modelu artefaktów związanych z procesem testowania i zarządzania jakością oprogramowania, oraz systemów w oparciu o podejście widoków architektonicznych. Koncepcja rozszerza idee testowania opartego o modele (ang. Model-Based Testing) [8], które opisuje proces testowania oprogramowania lub systemów. Publikacja bazuje na podejściu „4+1" dla modelowania architektury oprogramowania, która definiuje różne perspektywy, aby zaadresować potrzeby wszystkich udziałowców projektu. Adaptując to podejście w celu jego osadzenia w ramach procesów testowania oprogramowania wzbogacono architekturę tworzonego rozwiązania oraz nadano kontekst dla testów. W niniejszej publikacji jest promowane podejście do modelowania artefaktów procesu testowego przy zastosowaniu Unified Modeling Language oraz mechanizmu profili, które pozwala na definiowanie planu testów, przypadków testowych i strategii testów. Koncepcja pozwala na możliwość analizy wpływu pomiędzy wymienionymi elementami czy analizę pokrycia na różnych poziomach złożoności.

Stosowana aktualnie notacja obiektowa UML (ang. *Unified Modeling Language*) [1] została stworzona tak by wspierać modelowanie strukturalne i behawioralne dla szerokiej gamy rozwiązań. Pomimo uniwersalnego podejścia do projektowania, nie jest możliwe aby wspierać potrzeby każdego użytkownika uwikłanego bezpośrednio w cały cykl wytwarzania oprogramowania. Często można zaobserwować adaptacje notacji UML do zastosowań które nie

były jego docelowym przeznaczeniem. Powodem takiej sytuacji jest niejednorodność procesów wytwarzania oprogramowania oraz specyficzne wymagania użytkownika i przeznaczenie systemu. Język UML wykorzystywany do modelowania systemów opartych o obiekty nie obejmuje nowych technologii inżynieryjnych i wymaga czasu, aby zostały włączone do specyfikacji. Powoduje to, że UML najprawdopodobniej nigdy nie będzie dokładnie dopasowany do wymagań dowolnej technologii czy metodyki wytwórczej.

Uzupełnieniem specyfikacji jest rozwiązanie dostarczane przez Domain-Specific Modeling (DSM) [12], koncepcje projektowania i tworzenia systemów o strukturze wykraczającej ponad możliwości notacji UML. Przez uchwycenie podstawowych pojęć i ograniczenia dla problemu DSL promuje wiedzę o dziedzinie i ułatwia tworzenie odpowiednich rozwiązań opisywanych za pomocą modelu (ang. *Domain-Specific Model*), będącego odzwierciedleniem artefaktów dziedziny. Wizualizacją Domain-Specific Language jest metamodel dziedziny, który opisuje jego semantykę oraz atrybuty.

Mechanizm profili umożliwia dostosowanie UML dla różnych platform i specyficznych zastosowań nie przewidzianych wcześniej przez jego twórców [4]. Zastosowanie techniki modelowania Domain-Specific Modeling Languages (DSML) w dziedzinie problemu jakim jest proces testowania, umożliwia stworzenie modelu dziedzinowego, którego zakres jest ograniczony do kontekstu procesu testowania i jego artefaktów. Zastosowanie koncepcji DSL ogranicza sposób w jaki może być rozwijane rozwiązanie problemu, co umożliwia uniknięcie błędów w projektowaniu, przygotowania i realizacji dla procesu testowania. Mechanizm profilu umożliwia efektywniejszą komunikację niż model ogólnego przeznaczenia, dzięki dostarczaniu bardziej szczegółowych diagramów opisu architektury z perspektywy jej testowania oraz możliwości zapewnienia optymalnego poziomu szczegółowości. Użytkownikami modelu mogą być architekci projektujący architekturę tworzonego systemu i planują jej weryfikację i walidacje, testerzy implementujący przypadki testowe, analitycy definiujący wymagania oraz udziałowcy potrzebujący wyjaśnienia i zrozumienia problemów związanych z przebiegiem testów.

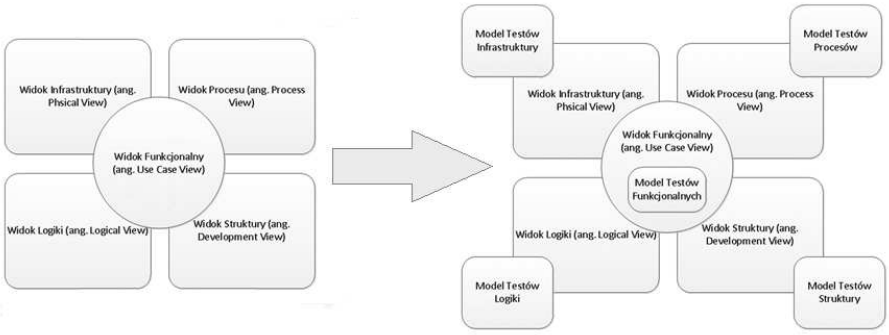
4.1. Koncepcja widoków dla procesu testowania

Zastosowanie techniki modelowania DSL omawianej wcześniej, wymagało określenia odpowiedniej metody wizualizacji modelu. Koncepcja wykorzystania modelu powinna umożliwiać wsparcie dla aktualnych wyzwań rynkowych przez profil opisujący proces testowania. Model powinien uwzględniać każdy z typów oraz technik testowania stosowanych na różnych fazach wytwarzania oprogramowania, co ma zapewnić możliwość planowania testów w całym cyklu życia oprogramowania i nie został ograniczony do modelowania testów na poziomie kodu (ang. *unit test*).

W trakcie konstrukcji profilu główna uwaga została skupiona na modelowaniu architektury systemu oraz jej połączeniu z artefaktami procesu testowania, które miało na celu udoskonalić proces w obszarach takich jak:

- komunikacja między testerami a innymi grupami wewnętrznych interesariuszy,
- projektowanie testów regresyjnych oraz określanie ich zakresu,
- testowanie w ograniczonych ramach czasowych,
- równoległa aktualizacja testów ze zmianami w architekturze systemu,
- redukcja czasu na wyszukiwanie i opracowanie błędów.

Osiągnięcie oczekiwanych rezultatów wymagało przyjęcia koncepcji, pozwalającej na wizualizacje procesu testowania na odpowiednim poziomie szczegółowości. W tym celu do opisu architektury systemu dla procesu testowania jest stosowany styl zaproponowany przez P. Kruchtena charakteryzujący się podziałem na zestaw pięciu widoków architektonicznych (ang. *4+1 Architectural View Model*) [7]. Autorzy starając się zachować jak największy stopień integralności z modelem budowanego systemu, zastosowali ten podział modelu na perspektywy. Pozwoliło to na ujednolicenie obu koncepcji i ich równoległy rozwój przez współdzielenie tych samych diagramów. Korzystanie z jednakowych diagramów w trakcie rozwoju testów jak i samego system, pozwala na tworzenie relacji pomiędzy modelami, jednocześnie ułatwiając identyfikacje zmian i prowadzenie testów regresyjnych. Efekt przystosowania koncepcji widoków dla modelowania testów przedstawiono na rysunku 4.1.



Rys. 4.1. Schemat ilustrujący integracje modelu testów z architekturą systemu

Szczególny nacisk został położony na wsparcie dla dobrych praktyk w prowadzeniu testów w oparciu o specyfikacje wymagań (ang. *Requirement Based-Testing*) [10]. Realizacja tej techniki w profilu pozwala na śledzenie oczekiwanych rezultatów z wynikami testów. Pozostałe cechy modeli testów nie są wspólne, lecz są ściśle związane z widokiem architektury systemu. Poniższy fragment zawiera charakterystykę każdej z perspektyw modelu testów.

Perspektywa testów funkcjonalnych – widok ten jest opisany poprzez diagramy przypadków użycia i diagramów aktywności. Przedstawia przede wszystkim testy na poziomie testów akceptacyjnych (ang. *user acceptance tests*), a także testów systemowych. Widok jest kluczowy dla zapewnienia poprawności działania systemu, gdyż przedstawia testy sprawdzające czy system dostarcza wszystkich wymagań funkcjonalnych zapisanych w dokumentacji.

Perspektywa testów logicznych – w przeciwieństwie do widoku testów funkcjonalnych, widok logiczny testów pokazuje testy dotyczące wewnętrznej struktury systemu. Dotyczy głównie testów integracyjnych. Perspektywa może być stosowana w obszarze testów jednego systemu, a także systemów rozproszonych składających się z wielu pojedynczych komponentów.

Perspektywa infrastruktury – pespektywa bezpośrednio wspierająca realizacje procesu testowania od strony infrastrukturalnej. W ramach widoku są stosowane elementy diagramu wdrożeniowego (ang. *deployment diagram*)

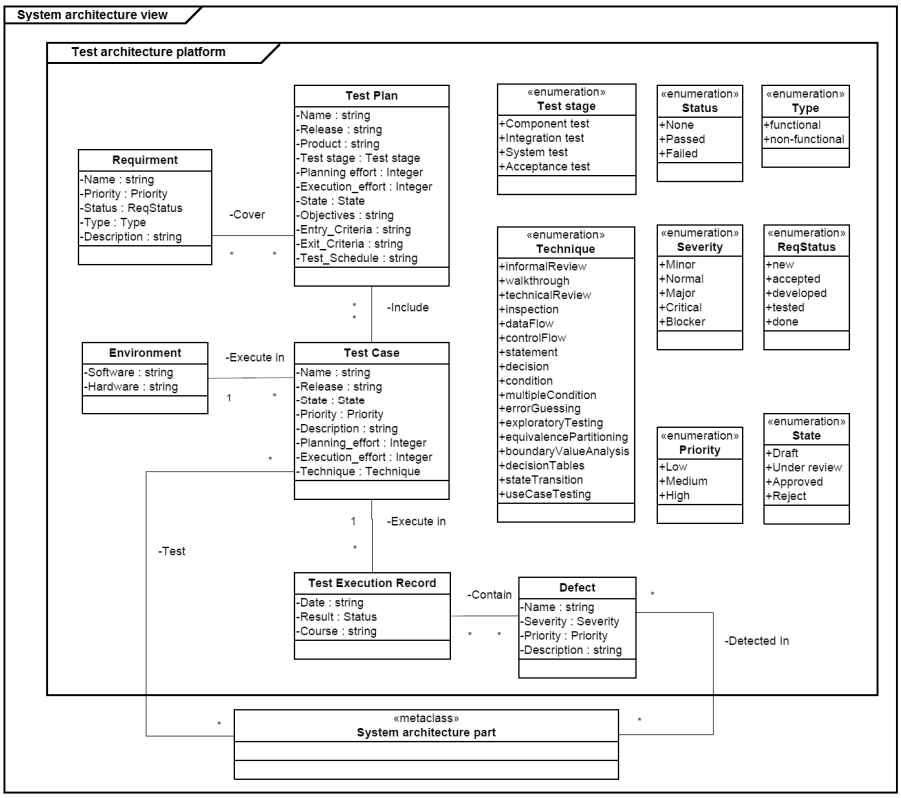
w celu opisania specyfikacji środowisk testowych oraz relacji pomiędzy ich elementami.

Perspektywa testów jednostkowych – Przedstawia testy na poziomie kodu (ang. *unit test*), potrzebne do sprawdzenia poprawności implementacji. Testy jednostkowe są obrazowane za pomocą diagramów klas i obiektów.

Perspektywa procesów – opisuje testy systemowe oraz funkcjonalne na poziomie realizacji kolejnych kroków scenariuszy testowych. Perspektywa procesowa pozwala na opisanie pełnego scenariusza kroków testowych, który musi zostać wykonany przez testera w ramach przypadku testowego.

4.2. Struktura zdefiniowanego profilu

Wszystkie kluczowe abstrakcyjne elementy, które zostały zidentyfikowane podczas analizy dziedziny procesu testowania, zostały zebrane i zamodelowane za pomocą języka UML jako metamodel DSL (ang. *Domain Specific Language*). Metamodel ten jest przydatny niezależnie od tego w jaki sposób język został zaimplementowany. Mimo tego, że nie jest bezpośrednio związany z generowaniem artefaktów dotyczących procesu testowania, pozwala na zrozumienie i przeanalizowanie jego struktury.



Rys. 4.2. Metamodel DSL dziedziny procesu testowania

Metamodel zawiera informacje o abstrakcyjnych elementach procesu testowania oraz ich wzajemnych relacjach, a także atrybutach które zostały zdefiniowane do opisania właściwości elementów. Określa także typy danych specyficzne dla tej dziedziny.

Szczególnie istotną cechą zobrazowaną przez metamodel DSL jest forma koegzystencji pomiędzy modelem procesu testowania a modelem architekturą systemu. Każdy widok architektury systemu zdefiniowany według koncepcji “4+1” P.Kruchtena dopuszcza dołączenie wewnątrz własnej struktury uniwersalnego widoku procesu realizacji testów. Konstrukcja uniwersalnego widoku procesu testowania oznacza że zestaw elementów potrzebnych do opisu przebiegu testów jest ten sam dla każdego z poziomów abstrakcji reprezentowanych przez widoki architektoniczne systemu . Konsekwencją tego

podejścia jest możliwość identyfikacji aktualnego stanu testów w każdym z widoków, przez interesariuszy pracujących na różnych poziomach abstrakcji.

Kluczowa dla kształtu profilu procesu testów jest ekspertyza dziedziny. Podczas określania struktury profilu posłużono się metamodelem opisującym składnie elementów użytych w modelu testów. Dlatego pierwszy krok na drodze do powstania profilu procesu testowania z metamodelu wymagał określenia, które meta-klasy powinny zostać przekształcone w stereotypy i które meta klasy UML zostaną rozszerzone o atrybuty dziedziny testowania. Najlepszym rozwiązaniem dla tego podejścia było wybranie meta-klas, które były najbliższe semantyki względem koncepcji profilu procesu testowania.

4.2.1. Podstawowe elementy profilu procesu testowania

TestPlan – reprezentuje dokument planu testów, który można określić jako dokument opisujący zakres, podejście, zasoby i harmonogram planowanych działań procesu testowania. Artefakt ten pozwala także na umieszczenie w modelu testów informacji na temat szacowanego czasu potrzebnego do wykonania przypadków testowych, definiuje również kryteria określające rozpoczęcie i zakończenie testów.

TestCase – reprezentuje pojedynczy przypadek testowy, który stanowi podstawową jednostkę wykonywalną dla procesu testowania. Element ten pozwala na dowiązanie do istniejących części systemu, określając cel i zakres testów. Pozwala również na określenie techniki jego wykonania jak np. warunki zakończenia. Równocześnie informuje o szacowanym czasie poświęconym na planowanie i wykonanie.

Requirement – przedstawia pojedyncze wymaganie zdefiniowany w specyfikacji. Zastosowanie go już na poziomie projektowania testów, ułatwia prowadzenie testów w oparciu o specyfikacje. Element zawiera przede wszystkim treść wymagania, typ oraz status. Użycie na diagramie pozwala na śledzenie planu testów i przypadków testowych.

TestExecutionRecord – przechowuje informacje o wyniku wykonania pojedynczego przypadku testowego. Identyfikuje czas wykonania testu oraz pozwala na zawarcie szczegółowe informacje dotyczące jego przebiegu.

Defect – pozwala na umieszczenie w modelu architektury systemu informacji na temat wady, którą stanowi nieprawidłowy lub nieoczekiwany wyniki realizacji przypadku testowego. Zawarty w profilu „Defect" wiązany jest z uszkodzonymi elementami systemu oraz z wynikiem wykonania testu.

Environment – obrazuje środowisko wykonania przypadków testowych, czyli ustawienia dotyczące oprogramowania i sprzętu, na których powinien odbyć się test.

TestArchitecture – wewnętrzny diagram, pozwalający na separacje artefaktów procesu testowania od architektury systemu.

4.2.2. Struktura zdefiniowanego profilu

Aby umożliwić praktyczne użycie opisanego profilu oraz weryfikację jego funkcjonalności, przeanalizowano narzędzia pozwalające na modelowanie architektury systemowej w celu selekcji aplikacji umożliwiającej implementacje profilu. Na wybór narzędzia miało wpływ jego rozpowszechnienie oraz możliwości rozbudowanej implementacji profili UML. Odpowiednia implementacja powinna dostarczać ułatwiający wdrożenie interfejs użytkownika, a także możliwość generowania kodu czy raportowania danych z modelu.

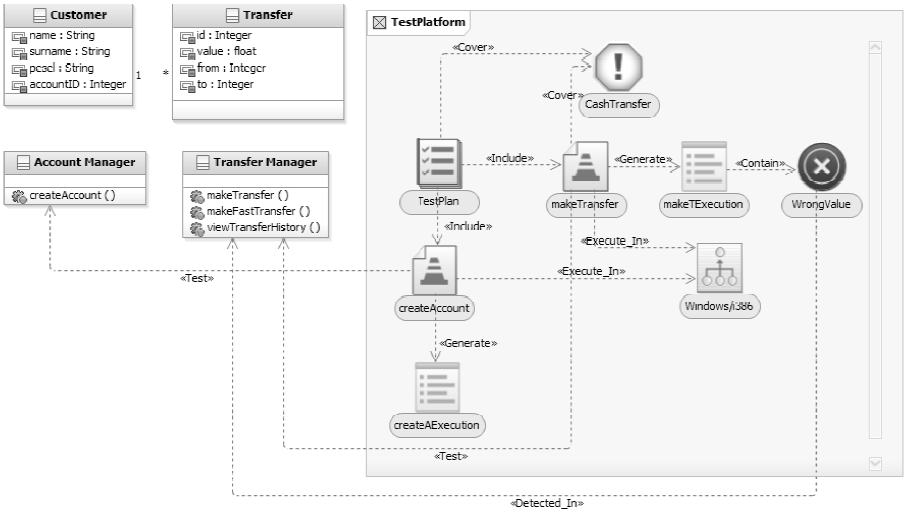
Na rynku istnieje wiele sposobów na implementacje profili UML, zależą one głównie od wykorzystanego narzędzia. W tym przypadku zastosowano użycie bardzo popularnej platformy „Eclipse", a dokładnie zbudowanej w oparciu o nią aplikacji IBM Rational Software Architect (RSA) [6] [9]. Implementacja opisanego wcześniej profilu została wykonana w postaci rozszerzenia dla programu Rational Software Architect. Udało się w ten sposób zrealizować interfejs użytkownika dostosowany dla modelowania procesu testowania. Rozszerzenie interfejsu ułatwia modelowanie testów, oraz edycję i weryfikacje zawartych w nim danych. Zbudowana wtyczka rozszerza cztery

podstawowe elementy interfejsu platform, zakładki właściwości, menu kontekstowe, palety narzędzi i komunikaty.

Konstrukcja samego modelu testów jest inicjowana od momentu dodania głównego elementu jakim jest plan testów, za pomocą którego określane są rozpoczęcia/zakończenia testów, czas ich wykonania, ich faza, a także atrybuty zależne od przebiegu takie jak np. aktualny status. Natomiast przypadek testowy charakteryzują takie cechy jak sposób wykonania czy szacowany czas na planowanie i realizacje. Dzięki temu model przypadku testowego reprezentuje więcej niż tylko jego wykonanie dla wskazywanego składnika sytemu, ale także definiuje jak test powinien być przeprowadzony.

Odrębnie opisywane są środowiska testowe tak aby móc łączyć z nimi grupy przypadków testowych. Zależnie od potrzeb umożliwiają sprecyzowanie dowolnej ilości żądanych cech sprzętu i oprogramowania. Profil pozwala na przeniesienie wymagań pochodzących z dokumentacji bezpośrednio na model testów, a także powiązanie ich z przypadkami testowymi i planem testów. Zastosowane rozwiązanie realizuje wsparcie dla koncepcji RBT (ang. *Requirements Based Testing*) zapewniając kontrole pokrycia wymagań, oraz pełne przejście między przedmiotem testu, testami a dalej rezultatami oraz potencjalnymi defektami które mogą zostać wykryte.

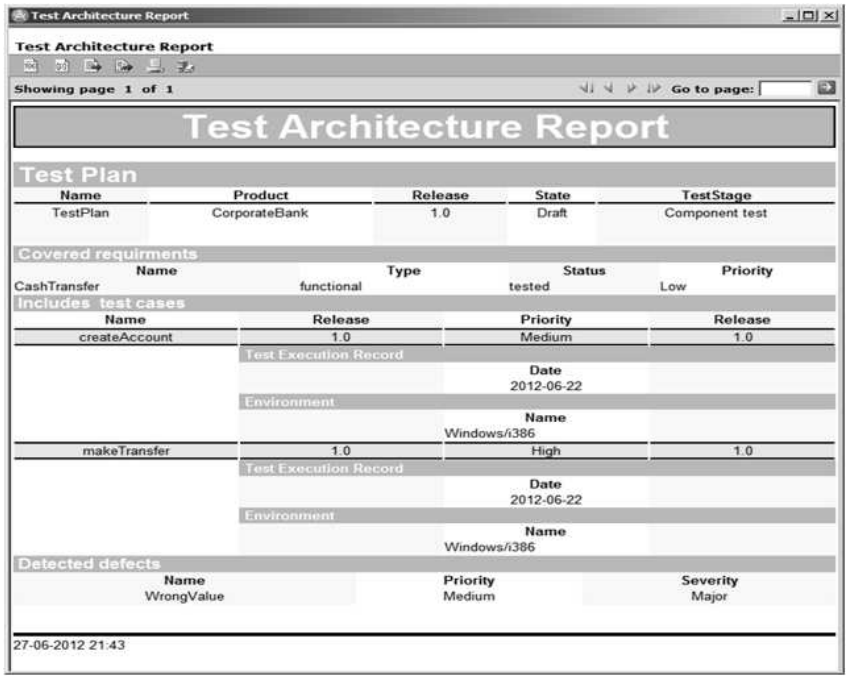
Wykorzystanie profilu zapewnia śledzenie efektów prowadzenia testów, prócz atrybutów przechowuje informacje o aktualnym stanie testów. Model uzupełniają artefakty przedstawiające wyniki przeprowadzonych testów dla przypadków testowych, także tych historycznych. Rezultatem testu zakończonego niepomyślnie jest defekt dołączany do wyniku jako osobny artefakt. Dzięki temu model opisuje wadę systemu, identyfikuje dotknięte wymagania oraz fragmentu systemu. Poniżej przedstawiono model architektury procesu testowania, który ilustruje przykład z testami struktury wbudowanymi w diagram klas.



Rys. 4.3. Model testów zbudowany w RSA za pomocą wtyczki implementującej profil

Aby budowany model zachowywał odpowiednią strukturę oraz znaczenie merytoryczne, wykorzystano ograniczenia stworzonych stereotypów, oraz informacje pochodzące bezpośrednio z modelu. W pierwszym przypadku narzędzie blokuje błędne konstrukcje informując o tym komunikatami, w drugim przypadku graficzny interfejs wskazuje poprawne zachowania oraz informuje o niepokojącym stanie planowanych testów np. przekroczony planowany czas na wykonanie przypadków testowych.

Ze względu na to że na model nakładane są informacje z przebiegu testu, dane w nim zawarte mogą być źródłem danych do tworzenia raportów na temat realizacji procesu testowania. W tym celu wykorzystanie Rational Softwae Architect, jako platformy dla implementacji profilu testowego, pozwoliło na stosowanie omawianego modelu testów w całym cyklu projektowania aplikacji. Pozwoliło również na użycie innych narzędzi opartych o platformą Eclipse, w tym do projektowania raportów. Narzędziem, które użyto w tym przypadku jest Business Intelligence and Reporting Tools (BIRT). Za pomocą, którego możliwe stało się generowanie raportów gromadzących informację w postaci sformatowanej bezpośrednio z modelu architektury systemu. W ten sposób uzyskano rozwiązanie, pozwalające na analizę przebiegu testów bezpośrednio związanego z architekturą systemu.



Rys. 4.4. BIRT – raport wykonany na bazie testów w postaci dokumentu HTML

Szereg funkcjonalności narzędzia, poprawia komfort pracy przy projektowaniu modelu, nie tylko testów. Wykorzystano te funkcjonalności w trakcie tworzenia interfejsu. W rezultacie profil został obudowany zestawem pomocnych graficznych komponentów. Model znajduje zastosowanie także w analizie przebiegu procesu testowania dostarczając raporty w pełni dostosowywalne zależnie od potrzeb interesariuszy.

4.3. Podsumowanie

W niniejszym rozdziale został przedstawiony uniwersalny profil dla modelowania procesu testowania, wspierający projektowanie architektury w oparciu o wymagania, gromadzenie danych z przebiegu testów oraz monitorowanie zmian.

Zastosowanie widoku pięciu perspektyw, prezentujących realizacje testów pozwala na ich projektowanie w trakcie całego cyklu rozwoju oprogramowania na różnych poziomach abstrakcji. Dzięki temu profil sprzyja ograni-

czeniu czasu na planowanie testów, a także zachęca do wykorzystania w niewielkich projektach i instytucjach. Istotne zaletą badań nad profilem okazała się możliwość śledzenia ścieżki od wymagania przez testy i defekt do fragmentu systemu. Aspekt ten został zrealizowany między innymi dzięki modelowaniu testów na diagramie umieszczanym wewnątrz diagramów architektury systemów. Budowanie przypadków testowych w taki sposób, czyli bezpośrednio z modelu, pozwala na zachowanie spójności między rzeczywistym przebiegiem testów a ich projektem.

Wykorzystanie narzędzia bazującego o platformę Eclipse zapewniło dostęp do bogatego narzędzia raportującego BIRT [5], z którego pomocą wygenerowano dokumenty zawierające informacje z procesu testowania. Rozwiązanie to sugeruje dalszą pracę badawczą nad możliwościami wykorzystania modelu testów oraz jego znaczeniem w trakcie rozwoju oprogramowania.

Rozdział omawia możliwość wyboru między różnymi koncepcjami dla modelowania testów. Opisywany profil jest natomiast przykładem połączenia MBD i DSL oraz wsparcia dla RBT. Odnosząc się do najsilniej rozwijanego dotychczas profilu testowego jakim jest rozwijany przez OMG UTP [14], zaproponowano badanie innych metod. Budowanie nowych propozycji i ich upublicznianie jest konieczne, ponieważ nie istnieje rozwiązanie uniwersalne, idealnie pasujące do wymagań procesowych każdej z instytucji czy technologii.

4.4. Bibliografia

1. Booch G., Jacobson I., Rumbaugh J.: *The Unified Modeling Language Reference Manual*, Addison-Wesley, 1999.
2. Fowler M.: *Domain-Specific Languages,* Addison-Wesley, 2010.
3. Fuentes-Fernández L., Vallecillo-Moreno A.: *Introduction to UML Profile,* UPGRADE, Vol. V, No. 2, 2004.
4. Giachetti G., Marín.B, Pastor.O.: *Using UML Profiles to Interchange DSML and UML Models*, Universidad Politécnica de Valencia, 2009.
5. IBM, *Generating Specification Documents from Models using BIRT*, http://www.omg.org/news/meetings/tc/mn/special-events/ecl/Elaasar-BIRT.pdf, 2010

6. IBM, *IBM Rational Software Architect Datasheet*, ftp://ftp.software.ibm.com/software/rational/web/datasheets/rsa.pdf, 2007.
7. Kruchten P.: *Architectural Blueprints - The "4+1" View Model of Software Achitecture*, IEEE Software 12 (6) ,1995.
8. Legeard B., Utting M.: *Practical Model-Based Testing: A Tools Approach*, Morgan-Kaufmann, 2006.
9. Misic.D.: *Authoring UML profiles using Rational Software Architect and Rational Software Modeler*, http://www.ibm.com/developerworks/rational/library/05/0906_dusko/, 2005
10. Mogyorodi G.E., Math. B., *Requirements-Based Testing: An Overview*, BIT, 2003.
11. OMG, *Unified Modeling Language, Infrastructure* , http://www.omg.org/spec/UML/2.1.2/, 2011
12. Steven K., Tolvanen J.:, *Domain-Specific Modeling*, Wiley-IEEE Computer Society, 2008.
13. OMG, *UML Object Constraint Language OCL Specification*, http://www.omg.org/spec/OCL/2.0/,2006.
14. OMG, UML Testing Profile, http://www.omg.org/spec/UTP/, 2012.

CZĘŚĆ II.
JAKOŚĆ OPROGRAMOWANIA

Rozdział 5

Dobre praktyki w procesach zapewniania jakości

W rozdziale przeprowadzono identyfikację praktyk stosowanych w metodykach zwinnych, które mogą wspierać procesy cyklu życia oprogramowania związane z zapewnieniem jakości. Zbiór zidentyfikowanych praktyk został ulokowany na poszczególnych poziomach modelu dojrzałości CMMI. Alokacja rozpatrywanych praktyk skupia się na drugim i trzecim poziomie. Stosowanie wszystkich praktyk zwinnych nie jest wystarczające, aby osiągnąć drugi poziom dojrzałości CMMI. Dodatkowo, rozważone metodyki zwinne pomijają niektóre praktyki, które mają wpływ na jakość końcowego produktu. Przykładem takiej praktyki jest modelowanie, które jest rekomendowane tylko przez nieliczne metodyki zwinne.

Do wytworzenia oprogramowania wysokiej jakości prowadzą dwa, powiązane ze sobą podejścia. Pierwsze – polega na jawnej specyfikacji wymagań jakościowych przed rozpoczęciem procesu wytwarzania a następnie na weryfikacji wytworzonego produktu programowego. Drugie – koncentruje się na jakości samego procesu wytwarzania oprogramowania. W drugim podejściu zaleca się stosowanie sprawdzonych sposobów postępowania, zwanych dobrymi praktykami, które co prawda bezpośrednio nie dają gwarancji uzyskania produktu oczekiwanej jakości, lecz ich pomijanie ma negatywny wpływ na jakość wytwarzanego produktu.

Rozdział ma dwa cele. Pierwszym jest analiza, które z praktyk stosowanych przez metodyki zwinne wspomagają procesy projakościowe w cyklu życia produktu programowego w rozumieniu normy ISO/IEC 12207: 2008 [2]. Drugim celem jest zbadanie jak wskazane w ten sposób praktyki – praktyki

projakościowe – lokują się w modelach dojrzałości CMMI [1]. Modele CMMI służą bowiem ulepszaniu procesów w organizacji wytwarzającej oprogramowanie, a ich głównymi elementami są właśnie praktyki odnoszące się do aktywności w całym cyklu życia przedsięwzięcia. Podobne prace dokumentowano m.in. w [12], gdzie autorzy próbowali określić stopnień pokrycia podzbioru procesów zdefiniowanych w ISO 12207:2008 przez praktyki zwinne metodyki SCRUM. Wg nich wdrożenie SCRUM zaspokaja 83% celów zdefiniowanych w procesie planowania przedsięwzięcia (Project Planning) oraz 75% celów zdefiniowanych w procesie kontroli i oceny przedsięwzięcia (Project Assessment and Control Process).

W podrozdziale 5.1., w oparciu o normę ISO/IEC 12207: 2008 [2], zostały wybrane praktyki stosowane w najbardziej popularnych metodykach zwinnych, które są związane z zapewnianiem jakości w cyklu życia oprogramowania. Rozdział 5.2. wskazuje na możliwości zastosowania praktyk projakościowych na poszczególnych poziomach dojrzałości modelu CMMI. Ostatni rozdział 5.3 stanowi podsumowanie rozdziału.

5.1. Procesy cyklu życia oprogramowania a zapewnianie jakości

Punktem wyjścia do rozważań jest norma ISO/IEC 12207:2008 [2], która definiuje wszystkie procesy związane z szeroko rozumianym wytwarzaniem systemów oprogramowania. Obejmuje ona siedem grup zawierających łącznie 43 procesy. Są to grupy procesów uzgodnień (Agreement Processes), ustanowienia przedsięwzięcia (Organizational Project-Enabling Processes), przedsięwzięcia (Project Processes), techniczne (Technical Processes), implementacyjne (Software Implementation Processes), ponownego użycia (Software Reuse Processes), oraz wspomagające (Software Support Processes).

Spośród tych grup głównie ostatnia grupa obejmuje procesy, które odnoszą się jawnie do jakości produktów informatycznych.

1. Proces zarządzania dokumentacją oprogramowania (*Software Documentation Management Process* – SDMP).
2. Proces zarządzania konfiguracją oprogramowania (*Software Configuration Management Process* – SCMP).

3. Proces zapewnienia jakości oprogramowania (*Software Quality Assurance Process* – SQAP).
4. Proces weryfikacji oprogramowania (*Software Verification Process* – SVerP).
5. Proces walidacji oprogramowania (*Software Validation Process* – SValP).
6. Proces przeglądu oprogramowania (*Software Review Process* – SRP).
7. Proces audytu oprogramowania (*Software Audit Process* – SAP).
8. Proces rozwiązywania problemów (*Software Problem Resolution Process* – SPRP).

Z pozostałych grup warto zwrócić uwagę na grupę procesów ustanowienia projektu, wśród której są wyróżnione procesy zarządzania jakością (*Quality Management* – QM). Właśnie wymienionych wyżej osiem procesów oraz proces QM stanowią tło dla określenia związku rozpatrywanych dalej praktyk z zapewnianiem jakości.

Zapewnianie jakości należy tu rozumieć jako działania polegające na monitorowaniu i ocenie różnych aspektów projektu w celu maksymalizacji prawdopodobieństwa uzyskania przez produkt programowy ustalonych oczekiwań jakościowych. Należy podkreślić, że działania zapewnienia jakości nie stanowią gwarancji uzyskania jakości, lecz ich brak zwiększa ryzyko niepowodzenia. Spośród procesów wspomagających bezpośrednio zapewnianie jakości wiążą się procesy SQAP, natomiast pośrednio, przez odwoływanie się jakości oprogramowania, procesy SVerP, SValP, SRP oraz SAP. Pozostałe procesy, SDMP, SCMP oraz SPRP, mają również istotny wpływ na jakość oprogramowania.

Każdy z procesów jest opisywany przez wskazanie listy aktywności i zadań, które należy wykonać w celu uzyskania wymaganych artefaktów wynikowych. Opisy te nie przesądzają o wykorzystywanych technikach i sposobach wykonania zadań. Doświadczenia praktyczne doprowadziły do wyłonienia pewnych sprawdzonych i zalecanych sposobów realizacji zadań – są to tak zwane praktyki lub najlepsze praktyki (*the best practices*).

Praktyki są definiowane różnie, ale dla celów dalszych rozważań przyjmujemy za Amblerem [3], że praktyka jest niezależnym, przemieszczalnym

komponentem procesu (*a practice is a self-contained, deployable component of a process*).

W dalszej części skupiamy się na praktykach wyłonionych w ramach stosowania metodyk zwinnych. Od momentu powstania manifestu zwinnego wytwarzania oprogramowania [4] upłynęło ponad dziesięć lat. Ogłoszenie manifestu było pewnego rodzaju przeciwstawieniem się metodykom ciężkim. Obecnie problem wyboru podejść zwinnych lub klasycznych (ciężkich) wiąże się z racjonalnym powiązaniem ze skalą realizowanych projektów informatycznych. Problem jakości oprogramowania wyraźnie uwzględniany przez metodyki klasyczne nie ma jawnego odniesienia w manifeście zwinności. Nie oznacza to, że metodyki zwinne problem ten pomijają – biorą go pod uwagę pośrednio, właśnie przez stosowanie odpowiednich praktyk.

Rozpatrujemy sześć popularnych metodyk zwinnych: *Extreme Programming* (XP2) [5], *Crystal Clear* [6], *Scrum* [7], *Dynamic Software Development Methodology* (DSDM) [8], *Agile Unified Process* (AUP) [9], *Agile Modeling* (AM) [10]. Związane z nimi praktyki projakościowe są wyselekcjonowane w tabeli 5.1.

Porównując opisy praktyk oraz opisy wybranych procesów, zdefiniowanych w normie ISO 12207, wybranych ze względu na ich powiązanie z zapewnieniem jakości, dokonano analizy, czy dana praktyka może stanowić instancję aktywności w obrębie danego procesu. Wyniki analizy zostały przedstawione w tabeli 5.1. Powiązanie praktyk z procesami zostało oznaczone symbolem X. Różne nazwy praktyk występujące w tym samym polu pierwszej kolumny tabeli są synonimami.

Należy zwrócić uwagę, że wśród praktyk nie ma żadnej związanej z procesem SAP (*Software Audit Process*). Tylko 23 praktyki są związane z procesami zapewniania jakości ISO, pozostałe praktyki nie mają żadnego z nimi związku.

Tab. 5.1. Praktyki wspomagające zapewnianie jakości w procesach ISO/IEC 12207

Nr	Praktyka	SDMP	SCMP	SQAP	SVerP	SValP	SRP	SAP	SPRP	QM
1	Wspólna przestrzeń informacyjna			x						x
2	Programowanie w parach				x					

Nr	Praktyka	SDMP	SCMP	SQAP	SVerP	SValP	SRP	SAP	SPRP	QM
3	Cykliczność pracy									x
4	Ciągła integracja		x	x	x					
5	Wytwarzanie przyrostowe									x
6	Analiza źródeł problemu								x	
7	Koduj i testuj	x								
8	Rejestr produktowy	x*								
9	Priorytyzacja wymagań	x*								
10	Historie użytkownika	x*			x	x				
11	Codzienne spotkania na stojąco			x					x	
12	Wykres wypalania dla sprintu			x						x
13	Częste wdrożenia					x				
14	Wersja demo na koniec sprintu					x				
15	Przegląd sprintu			x		x			x	
16	Spotkanie retrospekcyjne			x			x		x	
17	Zarządzanie konfiguracją		x							
18	Wizja architektury	x								
19	Włączenie klienta					x				
20	Modelowanie	x*								
21	Wykonywalne specyfikacje	x			x					
22	Refaktoryzacja	x								
23	Wytwarzanie sterowane testami	x			x					

* Praktyka jest opcjonalna; może być wymagana przez plan zarządzania dokumentacją w danym procesie.

5.2. CMMI a zwinne praktyki projakościowe

Wydaje się, że swojego rodzaju weryfikacją projakościowości wyselekcjonowanych praktyk jest zbadanie, czy i w jakim zakresie są one zgodne (są kompatybilne) z praktykami zalecanymi przez rozważany dalej model CMMI

for Development (CMMI-Dev), będący jednym z 3 modeli CMMI [1-3]. Modele te wprowadzają pięć poziomów dojrzałości organizacji: 1 – początkowy (initial), 2 – zarządzany (managed), 3 – zdefiniowany (defined), 4 – zarządzany ilościowo (quantitatively managed), 5 – optymalizujący (optimizing). Modele są wykorzystywane do oceny dojrzałości procesów w organizacji oraz wspomaganiu ulepszania procesów w organizacji wytwarzającej oprogramowanie.

Model CMMI-Dev określa, odmienne do ISO12207, kategorie procesów, a kryterium tego podziału są cztery kategorie aktywności wykonywane w przedsięwzięciu, tzn. zarządzanie projektem, inżynieria (wytwarzania produktu), zarządzanie procesem i wspomaganie, służące polepszania tych procesów. W ramach wymienionych kategorii aktywności model definiuje 22 obszary procesowe, z których 16 wchodzi w skład tzw. bazowego zestawu obszarów, jeden jest obszarem współdzielonym z innymi modelami CMMI, a pięć jest obszarami specyficznymi dla wytwarzania: opracowywanie wymagań, rozwiązanie techniczne, integracja produktu, weryfikacja i walidacja.

Każdemu z obszarów procesowych, na każdym poziomie dojrzałości, są przyporządkowane cele specyficzne, z którymi są stowarzyszone zalecane praktyki służące realizacji tych celów. Dopuszcza się użycie innych, ale odpowiednich do realizacji celów specyficznych, praktyk alternatywnych.

Za praktyki alternatywne można uważać praktyki zwinne wyłonione w podrozdziale 2. Każdą z tych praktyk można powiązać z jednym lub wieloma obszarami procesowymi. Powiązanie to wyznacza się przez analizę możliwości realizacji celów specyficznych obszaru przez daną praktykę. Na przykład, dla obszaru procesowego walidacja (*Validation*) kategorii inżynieria, który ma przypisane dwa cele specyficzne:

SG1: Przygotowanie do walidacji (poprzez wskazanie walidowanych produktów, ustanowienie środowiska walidacji oraz procedur i kryteriów)

SG2: Przeprowadzenie walidacji produktu (wykonanie walidacji i przeanalizowanie otrzymanych wyników)

można zaproponować praktyki zwinne *Włączenie klienta, Historie użytkownika* dla celu SG1, oraz *Wersja demo na koniec sprintu; Częste wdrożenia* dla celu SG2.

Ponieważ aspekty jakości w modelu CMMI są rozproszone między wymienione cztery kategorie procesów, i tym samym trudno alokować praktyki

do procesów, to nie można przeprowadzić analizy analogicznej do pokazanej w podrozdziale 2. W tym przypadku, najpierw dokonano analizy celów specyficznych dla wszystkich obszarów procesowych a następnie przypisano im odpowiednie, według autorów, zwinne praktyki projakościowe. Rezultaty analizy podano w tabeli 5.2. Ponieważ przeprowadzona analiza wykazała, że rozważane praktyki projakościowe (za wyjątkiem jednej) odnoszą się do drugiego i trzeciego poziomu dojrzałości, to jedynie te poziomy zostały uwzględnione w tabeli 5.2. Analizy nie prowadzono dla poziomu pierwszego, bowiem nie stosuje on żadnych praktyk, natomiast na pozostałych poziomach, 4 i 5, ma zastosowanie tylko jedna z analizowanych praktyk – *Analiza źródeł problemu* (poziom 5 – obszar procesowy *Analiza i rozwiązywanie incydentów*).

Tab. 5.2. Poziomy dojrzałości obszarów procesowych modelu CMMI a praktyki projakościowe

Poziom	Kategoria	Obszar procesowy	Zwinne praktyki projakościowe
2	Inżynieria (*Engineering*)	Zarządzanie Wymaganiami (*Requirements Management* – REQM) SG1:Zarządzaj wymaganiami	Włączenie klienta
2	Zarządzanie Projektem (*Project Management*)	Planowanie projektu (*Project Planning* – PP) SG1: Ustal szacunki (kosztów, czasu) SG2: Opracuj plan projektu SG3:Uzyskaj potwierdzenie planu	Cykliczność pracy; Wytwarzania przyrostowe
		Monitorowanie i kontrola projektu (*Project Monitoring and Control* – PMC) SG1: Monitoruj projekt względem planu SG2: Zarządzaj działaniami korygującymi prowadzącymi do zamknięcia projektu	Codzienne spotkania na stojąco; Wykres wypalania dla sprintu; Wspólna przestrzeń informacyjna; Przegląd sprintu
		Zarządzanie Uzgodnieniami z dostawcami (*Supplier Agreement Management* – SAM) SG1: Uzgodnij umowy z dostawcami SG2: Wykonuj umowy z dostawcami	

2	Wspomaganie (*Support*)	Pomiary i analiza (*Measurement and Analysis* – MA) SG1: Dopasuj aktywności pomiarowe i analityczne SG2: Dostarcz rezultaty pomiarów	Wspólna przestrzeń informacyjna; Wykres wypalania dla sprintu;
		Zapewnienie jakości procesu i produktu (*Process and Product Quality Assurance* – PPQA) SG1: Przeprowadź obiektywną ewaluację procesów i produktów pracy SG2: Dostarcz obiektywną ocenę	Wspólna przestrzeń informacyjna; Przegląd sprintu; Spotkanie retrospekcyjne; Ciągła integracja
		Zarządzanie konfiguracją (*Configuration Management* – CM) SG1: Ustal linię bazową SG2: Śledź i kontroluj zmiany SG3: Ustal integralność konfiguracji	Zarządzanie konfiguracją; Ciągła integracja
3	Inżynieria (*Engineering*)	Opracowywanie wymagań (*Requirements Development* – RD) SG1: Rozwijaj wymagania klienta SG2: Rozwijaj wymagania na produkt SG3: Analizuj i waliduj wymagania	Rejestr produktowy; Priorytetyzacja wymagań; Historie użytkownika; Modelowanie
		Rozwiązanie techniczne (*Technical Solution* – TS) SG1: Wybierz rozwiązanie dla komponentów produktu SG2: Opracuj projekt SG3: Implementuj projekt produktu	Wytwarzanie przyrostowe; Modelowanie; Wizja architektury; Refaktoryzacja; Wytwarzanie sterowane testami
		Integracja Produktu (*Product Integration* – PI) SG1: Przygotuj się do integracji produktu SG2: Zapewnij zgodność interfejsów SG3: Połącz komponenty produktu i dostarcz produkt	Ciągła integracja
		Weryfikacja (*Verification* – VER) SG1: Przygotuj się do weryfikacji SG2: Przeprowadź przeglądy parami (peer) SG3: Zweryfikuj wybrane produkty pracy	Koduj i testuj; Refaktoryzacja; Wykonywalne specyfikacje; Wytwarzanie sterowane testami; Programowanie w parach ; Ciągła integracja; Historie użytkownika; Wykonywalne specyfikacje

		Walidacja (*Validation* – VAL) SG1: Przygotuj się do walidacji SG2: Waliduj produkt	Włączenie klienta; Wersja demo na koniec sprintu; Częste wdrożenia; Przegląd sprintu; Historie użytkownika
3	Zarządzanie procesem	Orientacja na procesy organizacyjne (*Organizational Process Focus* – OPF) SG1: Określ możliwości ulepszania procesów SG2: Planuj i implementuj ulepszenia procesów SG3: Rozlokuj aktywa procesów organizacyjnych i stosuj "lessons learned"	Spotkanie retrospekcyjne
		Definiowanie procesów organizacyjnych (*Organizational Process Definition* – OPD) SG1: Ustal aktywa procesów organizacyjnych	
		Szkolenie organizacyjne (*Organizational Training* – OT) SG1: Ustal potencjał szkoleniowy organizacji SG2: Dostarcz niezbędnych szkoleń	
		Zintegrowane zarządzanie projektem (*Integrated Project Management* – IPM) SG1: Użyj zdefiniowanego procesu dla projektu SG2: Koordynacja i współpraca z interesariuszami	Włączenie klienta
		Zarządzanie ryzykiem (*Risk Management* – RSKM) SG1: Przygotuj się do zarządzania ryzykiem SG2: Zidentyfikuj i analizuj ryzyko SG3: Przenieś ryzyka	
3	Wspomaganie (Support)	Analiza i podejmowanie decyzji (*Decision Analysis and Resolution* – DAR) SG1: Dokonaj ewaluacji alternatyw	

W najnowszym modelu CMMI [1] niektóre z praktyk podanych w tabeli 5.2 są wymienione explicite jako praktyki wspomagające osiągnięcie odpo-

wiednich celów np. *Ciągła integracja* (dla SG1 Weryfikacji), *Historie użytkownika* (dla SG1 Opracowywanie wymagań). Praktyka modelowania (*Prototypes and models*) jest wymieniana jako metoda stosowana w realizacji celów specyficznych SG1 i SG2 w obszarze *Opracowywanie wymagań*.

5.3. Podsumowanie

W rozdziale przeprowadzono identyfikację praktyk stosowanych w popularnych metodykach zwinnych, które mogą wspierać procesy cyklu życia oprogramowania ISO/IEC 12207:2008 związane z zapewnieniem jakości. Następnie zbiór zidentyfikowanych praktyk został ulokowany na poszczególnych poziomach modelu dojrzałości CMMI. Z uwagi na ograniczenie objętościowe, przedstawiono tylko wyniki analiz, które doprowadziły do identyfikacji i ich ulokowania. Alokacja rozpatrywanych praktyk skupia się głównie na drugim i trzecim poziomie. Analiza wykazała, iż zestaw zidentyfikowanych praktyk projakościowych nie jest wystarczający, aby w pełni wesprzeć wszystkie cele tych poziomów.

Popularne metodyki zwinne pomijają niektóre praktyki, które mają istotny wpływ na jakość końcowego produktu. Przykładem takiej praktyki jest modelowanie. Praktyka ta, jawnie wskazywana przez model CMMI, jest rekomendowana tylko przez nieliczne metodyki zwinne. Z doświadczeń autorów, potwierdzonych badaniami [11] wynika, że stosowanie tej praktyki ma istotny wpływ na rozwijanie i utrzymanie produktu programowego – modele tworzone w procesie wytwarzania oprogramowania mogą być podstawą automatyzacji procesu i jednocześnie stanowią dokumentację.

5.4. Bibliografia

1. *CMMI® for Development, Version 1.3*. 2010. Software Engineering Institute. Dostępne pod adresem: http://www.sei.cmu.edu/reports/10tr033.pdf
2. ISO/IEC 12207:2008, Software and systems engineering – Software lifecycle processes.

3. Ambler, S. *Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process*. John Wiley & Sons, New York, Wiley Computing Publishing, 2002.
4. *Manifesto for Agile software development*, 2001, Dostępne pod adresem: http://www.agilemanifesto.org/
5. Beck, K., Andres, C.. *Extreme Programming Explained: Embrace Change* (2nd Edition), Addison-Wesley Professional, 2004.
6. Cockburn, A. *Crystal Clear. A Human-Powered Methodology For Small Teams, including The Seven Properties of Effective Software Projects*. Alistair Cockburn Humans and Technology copyright 1998-2004, Dostępne pod adresem: http://st-www.cs.illinois.edu/users/johnson/427/2004/crystalclearV5d.pdf.
7. Schwaber, K. *Agile Project Management with Scrum*. Washington, Microsoft Press, 2003.
8. *DSDM Atern Handbook v.1.0*. Dostępne pod adresem: http://www.dsdm.org/atern-handbook/flash.html#/50/
9. Ambler, S., *AmbySoft Agile Unified Process page*, 2005. Dostępne pod adresem: http://www.ambysoft.com/unifiedprocess/agileUP.html
10. Ambler, S. *Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process*. John Wiley & Sons, 2002, New York, Wiley Computing Publishing.
11. Dubielewicz I., Hnatkowska B., Huzar Z., Tuzinkiewicz L., *Quality Assurance in Agile Software Development*, in: Advances and Applications in Model-Driven Software Engineering, in print (August, 2013).
12. Irrazbal E,. Vásquez F., Díaz R., Garzás J., *Applying ISO/IEC 12207:2008 with SCRUM and Agile Methods*, Software Process Improvement and Capability Determination Communications in Computer and Information Science, Vol. 155, 2011, pp 169-180

Rozdział 6

Zobaczyć jakość oprogramowania

Za podstawę organizacji oprogramowania uznaje się powszechnie architekturę systemu informatycznego. Spełnia ona szereg istotnych funkcji, m.in. opisuje składniki oprogramowania i relacje pomiędzy nimi oraz przedstawia sposób, w jaki system informatyczny integruje się z innymi systemami. Architektura systemu powinna również opisywać zasady regulujące proces rozwoju oprogramowania. Aby móc sprawnie zarządzać procesem wytwórczym skomplikowanego oprogramowania, konieczne jest – z punktu widzenia architekta – stworzenie całościowego modelu oprogramowania, najlepiej zintegrowanego z narzędziami do ciągłej integracji i pozwalającego weryfikować wszystkie artefakty systemu. Z reguły wraz ze wzrostem systemu następuje wzrost komplikacji jego architektury. W poprzednich publikacjach autora został zaproponowany grafowy model pozwalający na gromadzenie informacji o wszystkich artefaktach oprogramowania. W niniejszym rozdziale przedstawiona zostanie metoda wizualnej prezentacji oprogramowania, która umożliwia szybką ocenę jakości danego oprogramowania. W dokonanej analizie wykorzystane będzie oprogramowanie stworzone w obiektowym języku programowania Java. Ocena będzie prowadzona na widokach zawierających klasy i relacje pomiędzy nimi, gdyż są one najważniejsze z punktu widzenia zapewnienia jakości. Zwizualizowane klasy będą posiadały dwie podstawowe własności: rozmiar (przedstawiający ważność klasy w systemie) oraz kolor (obrazujący jakość klasy w sensie metryk oprogramowania). Celem takiej prezentacji jest dostarczenie architektowi (lub innym członkom projektu informatycznego) łatwej i szybkiej metody całościowej oceny jakości systemu oraz wskazanie elementów systemu wymagających największej uwagi. Zaprezentowane w pracy badania zostały przeprowadzone na projektach open source różniących się zarówno rozmiarem, jak i poziomem skomplikowania.

Celem niniejszej pracy jest przedstawienie korzyści płynących z wizualnej analizy oprogramowania. Współcześnie systemy informatyczne są coraz bardziej skomplikowane, co utrudnia zagwarantowanie ich wysokiej jakości. W klasycznym modelu zapewnienie oczekiwanej (akceptowalnej) jakości oprogramowania wymaga od architekta systematycznej analizy każdej zmiany wprowadzanej do kodu. W przypadku większych zespołów programistów jest to zadanie bardzo trudne. W tej sytuacji pomocna może okazać się zaproponowana w niniejszym rozdziale metoda wizualizacji modelu grafowego, która pozwala architektowi „zobaczyć" jakość oprogramowania i – w konsekwencji – dokonać jego szybszej oceny . Jest to możliwe dzięki odpowiednim algorytmom wizualizacji grafów i danym przedstawianym na wizualizacji, co umożliwia szybką identyfikację tych elementów systemu, które wymagają poprawy. Dodatkowo zaproponowana metoda pozwala na przeprowadzanie wizualizacji na różnych poziomach granularności. Pozwala to na uzyskiwanie rysunków z taką liczbą informacji, która umożliwia danej osobie na dokonanie efektywnej i sprawnej oceny danego systemu lub jego elementu.

Rozdział składa się z czterech części: na początku dokonano przegląd literatury przedmiotu, następnie zaś zdefiniowany został grafowy model oprogramowania. W kolejnej części opisano eksperymenty przedstawiające przykładowe wizualizacje przy użyciu prototypowego narzędzia. Opracowanie zamyka opis implementacji narzędzia (przedstawionego w części trzeciej) oraz rekomendacje odnośnie do dalszego rozwoju zaprezentowanej metody.

6.1. Przegląd literatury przedmiotu

W opracowaniu modelu opisanego w niniejszej pracy wykorzystano istniejące w teorii informatyki podejścia i metodyki wytwarzania oprogramowania. Ponieważ systemy informatyczne są często rozbudowanymi i syndromatycznymi strukturami, problematyka formalnej metody klasyfikacji i zarządzania nimi poruszana jest w literaturze przedmiotu od wielu lat. Zunifikowane podejście do wytwarzania oprogramowania zostało zaproponowane już przez Osterweila [1]. Opracowano także formalne języki służące do opisu architek-

tury systemów, np. ADL [2]. Warto zauważyć, że w zaproponowanym modelu opis w języku ADL byłby jednym z artefaktów przechowywanych w grafie.

W przeszłości pojawiały się również propozycje grafowego modelu wspomagającego rozwój oprogramowania, np. modele przechowywania decyzji architektonicznych [3], wykrywania decyzji architektonicznych na podstawie opisów zmian [4] lub całościowe rozwiązania wspierające podejmowanie decyzji, jak i śledzenie zmian [5]. Podsumowanie zmian w podejściu do zarządzania architekturą systemów informatycznych znajduje się w [6].

Określanie jakości oprogramowania jest dziedziną, w ramach których przeprowadzanych jest współcześnie wiele badań. Popularne obecnie zestawy metryk oprogramowania to między innymi zestaw Chidambera i Kemerera [14], zestawy metryk MOOD [15], metryki Roberta Martina [16], metryki Halsteada [17] czy zestaw metryk odnoszących się do skomplikowania oprogramowania zaproponowany przez McCabe'a [18]. Sama wizualizacja oprogramowania stanowi również przedmiot niektórych badań. W ich wyniku powstają narzędzia takie jak Bauhaus [7], Source Viewer 3D [8], Gevol [9], JIVE [10], evolution radar [11], code_swarm [12] and StarGate [13]. Należy jednak podkreślić, iż żadne z wymienionych powyżej rozwiązań nie umożliwia prezentacji oprogramowania na różnych poziomach systemu (modułach, pakietach, klasach, metodach) w połączeniu z wizualizacją jego jakości.

6.2. Model grafowy

Przeprowadzone badania oparte zostały o grafowy model opisujący strukturę programu, który został zdefiniowany w [19]. W modelu zawarte są informacje na temat kodu źródłowego i architektury systemu. Jednym z głównych powodów wprowadzenia takiego modelu była potrzeba posiadania łatwo skalowalnego modelu mogącego zawierać różne typy artefaktów, tak aby móc zaprezentować całość oprogramowania wraz z procesem jego wytwarzania.

Model jest zdefiniowany jako skierowany etykietowany multigraf. Multigraf zdefiniowano jako trójkę: (V, L, E), gdzie:

1. V jest zbiorem wierzchołków opisujących artefakty powstałe w procesie wytwórstwa oprogramowania;
2. L jest zbiorem etykiet opisujących wierzchołki i krawędzie grafu;

3. $E \subseteq V \times L \times V$ jest zbiorem etykietowanych skierowanych krawędzi, które reprezentują zależności pomiędzy artefaktami.

Pomiędzy artefaktami może być wiele krawędzi, gdyż artefakty mogą być w więcej niż jednej relacji. Relacje z reguły nie są zwrotne (graf jest skierowany).

Wierzchołki reprezentujące elementy kodu programu to na przykład: moduły, pakiety, klasy, metody, testy (jednostkowe, integracyjne, wydajnościowe). Innymi przykładami wierzchołków są wymagania, przypadki użycia, zmiany, kod źródłowy w językach wyższego poziomu (np.: gramatyki vacc), pliki konfiguracyjne itp. Przykładowe krawędzie w grafie to:

1. zawieranie – klasa zawiera metody i pola, pakiet zawiera klasy;
2. wywołanie – klasa wywołuje metodę, metoda wywołuję metodę;
3. definiowanie – gramatyka definiuje język.

Etykiety mogą opisywać różne właściwości wierzchołków i krawędzi. Na przykład klasa może być „abstrakcyjna" i napisana w języku „Java" oraz być klasą „testową". Etykiety zawierają też właściwości mierzalne artefaktów [20], klasa może mieć „465 linii kodu" i „złożoność cyklometryczną 30".

Dzięki wykorzystaniu grafu można łatwo wykonywać różne operacja na grafie. Korzystając z pojęć abstrakcyjnych można obliczać metryki oprogramowania w wyabstrahowanym modelu jak również transformować graf do widoków użytecznych w określonych celach. Przykładowe widoki to: graf zawierający klasy i zależności między mini zdefiniowane jako wywołania metod między klasami. Widok ten można dla projektów o większej złożoności uogólnić do widoku pakietów i zależności między nimi (zdefiniowanych analogicznie) lub nawet dla bardzo dużych projektów widoku modułów wraz zależnościami.

6.3. Eksperymenty i wyniki

Badania zostały skoncentrowane wokół wizualnej analizy oprogramowania. Ocenie poddanych zostało wiele projektów, analizy wybranych z nich przedstawiono poniżej. W rozdziale zawarto również analizę porównawczą dwóch małych projektów (Ted i JUnit) oraz dwóch średnich projektów (JHotDraw ora JLoXiM). W pracy nie zostały opisane duże projekty (np: Apache

Camel) ze względu na ograniczenia formalne związane z rozmiarem obrazów. W przypadku dużych projektów rysunek przestawiający widok na poziomie pakietów jest już nieczytelny, natomiast widok modułów jest zbyt ogólny.

6.3.1. Klasyfikacja projektów

W tabeli 6.1 przedstawiono podstawowe informacje dotyczące projektów wraz z klasyfikacją ich rozmiaru.

Tab. 6.1. Podstawowe informacje na temat projektów oraz klasyfikacja projektów.

Język programowania	Nazwa projektu	Liczba plików zawierających kod źródłowy	Liczba linii kodu w projekcie	Liczba linii komentarzy w projekcie	Klasyfikacja
Java	Ted	222	29645	7287	Mały projekt
	Junit	305	15356	4057	
	JLoXiM	1862	113966	42103	Średni projekt
	JHotDraw	630	80691	40362	
	Apache Camel	6464	345580	166646	Duży projekt

Źródło: opracowanie własne.

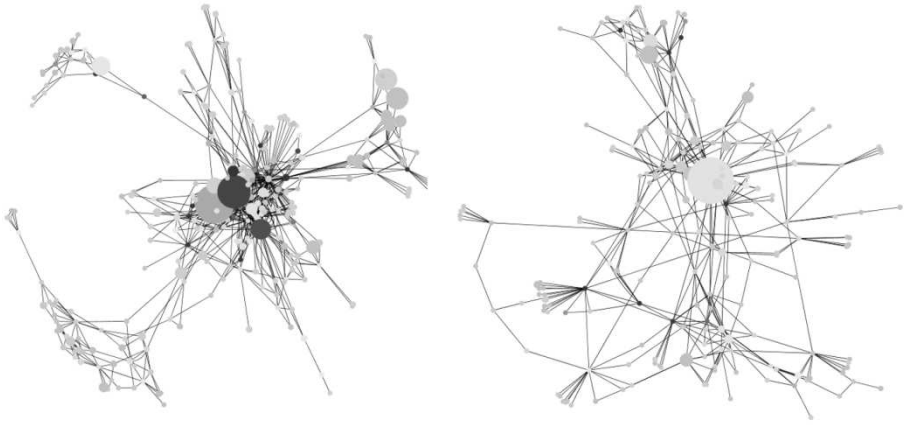
6.3.2. Analiza porównawcza Ted i JUnit

W niniejszej części zaprezentowano porównanie dwóch popularnych projektów – Ted oraz JUnit. Na rysunkach 6.1 i 6.2 przedstawione jest ich zwizualizowane porównanie. Oba rysunki przedstawiają klasy wraz z zależnościami między nimi, które zdefiniowane są jako wywołania metod pomiędzy klasami. Każdy wierzchołek posiada dwie własności: wielkość i kolor. Wielkość wierzchołka symbolizuje ważność klasy i w obu rysunkach jest zdefiniowany jako wartość obliczona przez algorytm PageRank uruchomiony na pełnym grafie oprogramowania. Kolor wierzchołka przedstawia trudność/złożoność klasy, która obliczana jest na dwa sposoby:

1. na rysunku 6.1 trudność jest zdefiniowana jako wartość metryki Halsteada trudność (difficulty) [17] – im ciemniejszy kolor wierzchołka tym jego trudność jest większa;

2. na rysunku 6.2 za kolor klasy odpowiada jej złożoność cyklometryczna [18] – im większa złożoność cyklometryczna tym ciemniejszy kolor wierzchołka.

Dodatkowo, aby pokazać lepiej zależności między poszczególnymi klasami, podczas tworzenia wizualizacji grafów zdefiniowano przyciąganie między wierzchołkami oparte na ilości wywołań metod pomiędzy klasami – im bardziej klasy się wywołują tym są bliżej.

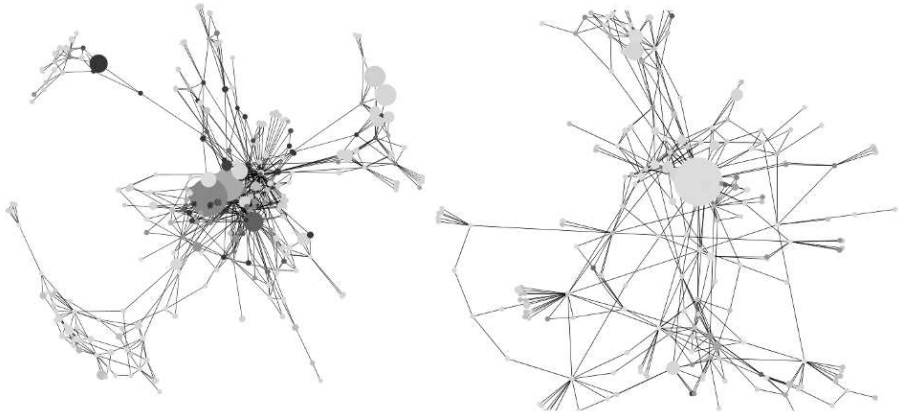


Rys. 6.1. Porównanie trudności programu Ted (po lewej) oraz biblioteki JUnit (po prawej).
Źródło: opracowanie własne.

Już z pobieżnej analizy rysunków wynika, że projekt JUnit jest projektem o lepszej jakości. Na wizualizacji JUnit nie ma dużych ciemnych wierzchołków, które oznaczają ważne i skomplikowane zarazem klasy. Takie klasy są widoczne w przypadku projektu Ted. Co więcej, projekt Ted posiada duże skupisko klas w centralnej części rysunku. Oznacza to występowanie wielu blisko powiązanych ze sobą klas. Architekt systemu obserwując taki stan rzeczy, powinien zalecić pogłębioną analizę oprogramowania w tym obszarze i jego refaktoryzację. W przypadku Ted przeanalizowano kod klas z centralnej części rysunków – duże ciemne kropki okazały się wynikiem wymieszania logiki biznesowej projektu z funkcjami wyświetlania okien. W przypadku JUnit charakterystycznym elementem jest bardzo duży wierzchołek w centralnym punkcie wizualizacji. Jest to klasa odpowiadająca za uruchamianie i za-

rządzanie testami. Warto też zwrócić uwagę na korelację pomiędzy trudnością a złożonością cyklometryczną widoczną na obu rysunkach.

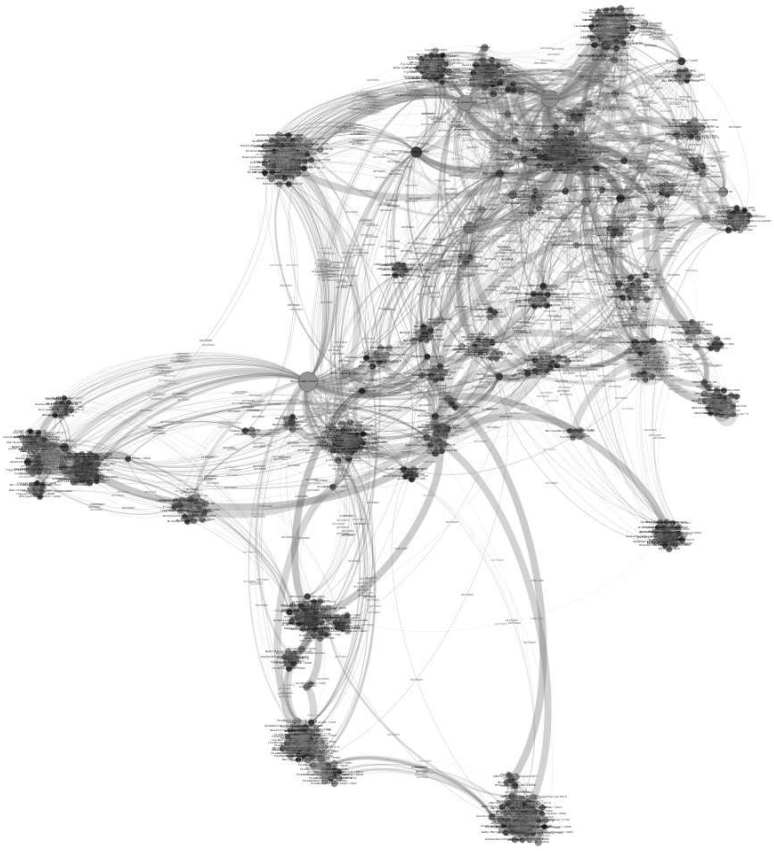
Podsumowując, z wizualnej analizy widoków grafu wynika, iż JUnit jest projektem o lepszej jakości – nie ma trudnych (w sensie metryk) i ważnych klas oraz jest dość modularny – istnieje kilka skupisk klas. Nie można powiedzieć tego samego o projekcie Ted , który potrzebuje interwencji poprawiającej jego jakość. Pożądane byłoby uproszczenie klas przedstawionych jako ciemne wierzchołki oraz zmniejszenie stopnia zależności między klasami, na przykład wprowadzając odpowiednie interfejsy lub stosując techniki takie jak inwersja kontroli.



Rys. 6.2. Porównanie złożoności cyklometrycznej program Ted (po lewej) oraz biblioteki JUnit (po prawej).
Źródło: opracowanie własne.

6.3.3. Analiza porównawcza JHotDraw oraz JLoXiM

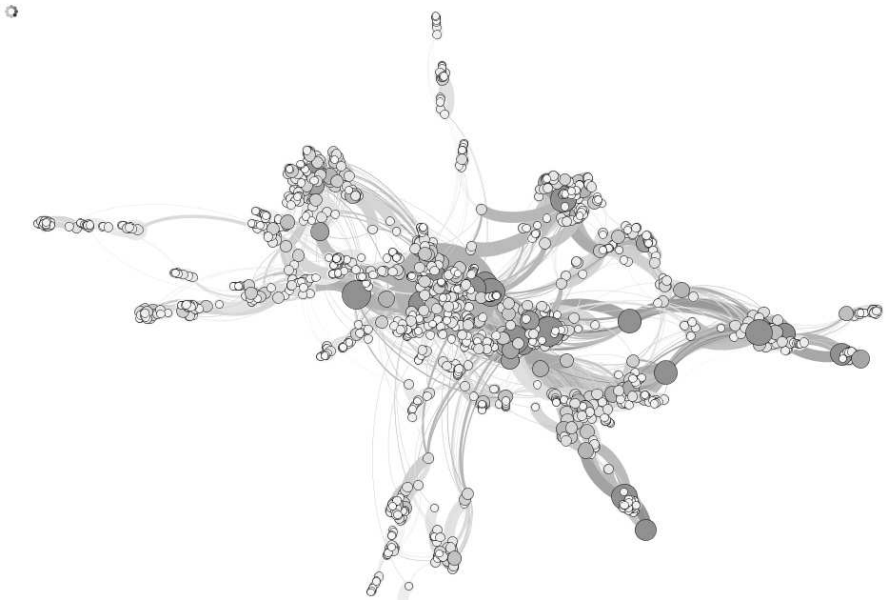
W kolejnym kroku przeanalizowane zostaną dwa projekty średniej wielkości, JLoXiM i JHotDraw. JLoXiM [21] to semistruktrulna eksperymentalna baza danych pisana przez studentów MIMUW, podczas gdy JHatDraw to graficzny edytor, często wykorzystywany do poszukiwania różnych wzorców projektowych.



Rys. 6.3. Program JHotDraw podzielony na moduły w oparciu na ilość wywołań metod między klasami
Źródło: opracowanie własne

W tym przypadku oceniono oprogramowanie pod kątem jego modularności. Modularność analizowana była jako wynik działania algorytmu wizualizującego Force Atlas 2 [22] na widoku klas w przypadku JHotDraw, natomiast dla JLoXiMa został zmieniony poziom abstrakcji na pakiety ze względu na to, że wizualizacja klas byłaby nieczytelna. Wizualizacje przedstawiające wyniki działania algorytmu są – przedstawione na rysunku 6.3 i 6.4.

Jak widać na rysunku 6.3 JHotDraw jest w porównaniu do JLoXiMa lepiej podzielony na moduły – łatwo jest zauważyć skupiska klas powiązanych ze sobą mocno, a z innymi grupami wierzchołków jedynie relatywnie niedużą liczbą krawędzi. W przypadku JLoXiMa sytuacja jest bardziej skomplikowana. W centralnym elemencie rysunku 6.4 znajduje się dużo mocno powiązanych ze sobą pakietów. Na obrzeżach rysunku można zauważyć dobrze zdefiniowane moduły. Należy zwrócić uwagę, że takie zależności nie są widoczne w przypadku klasycznego widoku programów drzewa modułów i pakietów – oba projekty jako projekty są podzielone na moduły i pakiety.



Rys. 6.4. Pakiety kodu źródłowego bazy danych JLoXiM pogrupowane według ilości wywołań metod między nimi.
Źródło: opracowanie własne.

Po analizie struktury kodu, na rysunku przedstawiającym JLoXiMa powinno być widoczne:

- 13 modułów testowych;
- 15 modułów definiujących interfejsy programistyczne (API);
- 32 modułów odpowiadających za implementację (niektóre moduły mają kilka implementacji – na przykład skład bazy danych);

- 15 modułów pomocniczych zawierających na przykład definicje protokołu komunikacyjnego bazy, narzędzia do pomiaru wydajności.

Jak widać z rysunku 6.4, modularność zaplanowana przez architekta poprzez podział kodu źródłowego nie została zachowana podczas implementacji. W tym przypadku rekomendowana jest analiza kodu źródłowego pod kątem zgodności z planowaną oryginalnie strukturą.

6.3.4. Implementacja

Zaprezentowane wyniki eksperymentów są rezultatem uzyskanym z prototypowego narzędzia – stworzonego w języku programowania Java – służącego do analizy projektów. W chwili obecnej narzędzie korzysta z grafowej bazy danych Neo4J do składowania modelu zdefiniowanego w podrozdziale 6.2. W chwili obecnej prototyp jest w stanie analizować kody źródłowe programów napisanych w języku Java. Do wizualizacji są wykorzystywane narzędzia oparte na otwartym kodzie źródłowym (open source) takie jak Cystoscape i Gephi. System może być łatwo rozszerzalny do obsługi kolejnych języków programowania – wystarczy stworzyć parser tłumaczący kod do zaproponowanego modelu grafowego.

6.4. Podsumowanie i możliwe rozszerzenia

Wizualna analiza oprogramowania jest możliwa i pozwala odkrywać zależności, które nie są widoczne w klasycznym podejściu do oceny oprogramowania. Dzięki wizualizacji architekt i programiści są w stanie relatywnie łatwo zlokalizować najważniejsze elementy systemów informatycznych. Możliwość transformacji i tworzenia widoków pozwala analizować oprogramowania na różnym poziomie modułów, pakietów, klas a nawet metod. Dzięki temu w przypadku potrzeby dokładniejszej analizy określonej części systemu informatycznego można stworzyć odpowiedni do tego zadania widok.

W przyszłości planowane jest rozbudowanie omówionego narzędzie tak, aby zwiększyć jego możliwości, jakość wizualizacji oprogramowania oraz rozszerzyć jego funkcjonalność. Jednym z możliwych rozszerzeń jest stworzenie modułu automatycznie przepakowującego klasy między pakietami, co

pozwoliłoby minimalizować liczbę wywołań metod między pakietami – czyli reorganizującego oprogramowanie tak, aby jego statyczna struktura odpowiadała strukturze wynikłej z analizy. Innym przykładem możliwych kierunków rozwoju narzędzia jest dodanie kolejnych modułów do obliczania nowych metryk, wyszukiwania wzorców i antywzorców projektowych.

6.5. Bibliografia

1. Bastian, Mathieu, Sebastien Heymann, and Mathieu Jacomy. *Gephi: An open source software for exploring and manipulating networks.* International AAAI conference on weblogs and social media. Vol. 2. Menlo Park, CA: AAAI Press, 2009.
2. Osterweil, Leon: *Software processes are software too,* Proceedings of the 9th international conference on Software Engineering. IEEE Computer Society Press, 1987
3. Dashofy, Eric M., André Van der Hoek, Richard N. Taylor: *A highly-extensible, XML-based architecture description language,* Proceedings of the Working IEEE/IFIP Conference on Software Architecture, 2001
4. That, Minh Tu Ton, Salah Sadou, Flavio Oquendo *Using Architectural Patterns to Define Architectural Decisions,* Proceedings of the 2012 Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture, IEEE Computer Society, 2012
5. Tang, Antony, Peng Liang, Hans van Vliet: *Software architecture documentation: The road ahead,* Proceedings of the 2011 Ninth Working IEEE/IFIP Conference on Software Architecture, IEEE Computer Society, 2011
6. Breivold, Hongyu Pei, Ivica Crnkovic, Magnus Larsson: *Software architecture evolution through evolvability analysis,* Journal of Systems and Software, 2012
7. Garlan, David, Mary Shaw: *Software architecture: reflections on an evolving discipline,* Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering, ACM, 2011

8. Koschke, Rainer: *Software visualization for reverse engineering,* Revised Lectures on Software Visualization, International Seminar, Springer-Verlag, 2001
9. Maletic, Jonathan I., Andrian Marcus, Louis Feng: *Source viewer 3d (sv3d): a framework for software visualization*, Proceedings of the 25th International Conference on Software Engineering, IEEE Computer Society, 2003
10. Collberg, C., Kobourov, S., Nagra, J., Pitts, J., Wampler, K.: *A system for graph-based visualization of the evolution of software,* Proceedings of the 2003 ACM symposium on Software visualization, ACM, 2003
11. Reiss, Steven P.: *Dynamic detection and visualization of software phases,* ACM SIGSOFT Software Engineering Notes. Vol. 30. No. 4. ACM, 2005
12. D'Ambros, Marco, Michele Lanza, Mircea Lungu: *The evolution radar: Visualizing integrated logical coupling information,* Proceedings of the 2006 international workshop on Mining software repositories, ACM, 2006
13. Ogawa, Michael, Kwan-Liu Ma: *code_swarm: A design study in organic software visualization,* Transactions on Visualization and Computer Graphics 15.6, IEEE Computer Society, 2009
14. Ma, Kwan-Liu: StarGate: A unified, interactive visualization of software projects, Visualization Symposium, IEEE Computer Society, 2008
15. Chidamber, Shyam R., Chris F. Kemerer: *A metrics suite for object oriented design*, IEEE Transactions on Software Engineering 20.6, IEEE Computer Society, 1994
16. Brito e Abreu, F.: *The MOOD metrics set,* Proceedings of European Conference on Object-Oriented Programming Vol. 95, Springer-Verlag, 1995
17. Martin, Robert Cecil: Agile software development: principles, patterns, and practices, Prentice Hall PTR, 2003
18. Halstead, Maurice H.: Elements of Software Science (Operating and programming systems series), Elsevier Science Inc., 1977

19. McCabe, Thomas J.: *A complexity measure,* Proceedings of the 2nd international conference on Software engineering, IEEE Computer Society Press, 1976
20. Dąbrowski, Robert, Krzysztof Stencel, Grzegorz Timoszuk: *Software is a directed multigraph,* Software Architecture, Springer-Verlag, 2011
21. Dabrowski, Robert, Krzysztof Stencel, Grzegorz Timoszuk: *One Graph to Rule Them All – Software Measurment and Managemen,* Proceedings of the 21th International Workshop on Concurrency, Specification and Programming, 2012
22. Tabor, Piotr, Krzysztof Stencel: *Stream Execution of Object Queries.* Grid and Distributed Computing, Control and Automation, Springer-Verlag, 2010
23. Bastian, Mathieu, Sebastien Heymann, Mathieu Jacomy: *Gephi: An open source software for exploring and manipulating networks,* International AAAI conference on weblogs and social media, AAAI Press, 2009

CZĘŚĆ III.
EDUKACJA W ZAKRESIE INŻYNIERII OPROGRAMOWANIA

Rozdział 7

Kształcenie inżynierii wymagań i procesy inżynierii wymagań według IREB

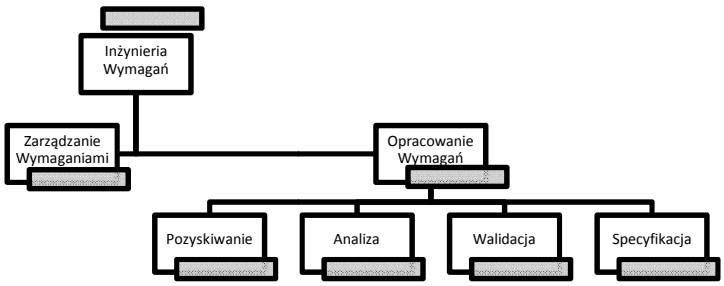
Inżynieria wymagań jest ważnym działem inżynierii oprogramowania. Znaczenie inżynierii wymagań i prawidłowego przebiegu procesów zarządzania wymaganiami jest często kluczowym czynnikiem sukcesu dla projektów informatycznych. Znane są przypadki, gdzie porażka projektu wynikała z błędów podczas analizy wymagań. [1][2][3][4] Świadomość i znajomość tych procesów dla współczesnych inżynierów informatyków nie zawsze jest wystarczająca. W rozdziale został dokonany przegląd nauczania inżynierii wymagań w programach nauczania wybranych uczelni wyższych kształcących studentów na kierunku informatyka oraz metody kształcenia w tym zakresie. W ostatnich latach obserwuje się rozwój edukacji inżynierii wymagań nie tylko na wyższych uczelniach, ale również w kształceniu ustawicznym. Powstają nowe inicjatywy i organizacje szerzące wiedzę i dobre praktyki inżynierii wymagań. Na rynku pojawiają się też certyfikaty z tego obszaru wiedzy. W rozdziale przedstawiono jedną z takich organizacji - International Requirements Engineering Board (IREB) oraz jej podejście do procesów inżynierii wymagań i certyfikacji dostępnej od 2012 roku również w Polsce, a certyfikat IREB CPRET ma około 11 000 osób [14].

Inżynieria wymagań oprogramowania jest relatywnie nowym pojęciem w informatyce. Zostało ono stworzone w celu określenia wszystkich czynności związanych z szeroko rozumianym pozyskiwaniem, dokumentowaniem i zarządzaniem wymaganiami dla systemów komputerowych. Słowo „inżynieria" w tym kontekście należy rozumieć jako systematyczną i zdefiniowaną działalność zapewniająca powodzenie tych procesów. Widoczne jest tu zatem odmienne do tradycyjnego podejścia, w którym inżynieria rozumiana jest jako „używanie właściwości materii, energii oraz obiektów abstrakcyjnych dla two-

rzenia konstrukcji, maszyn i produktów, przeznaczonych do wykonywania określonych funkcji lub rozwiązania określonego problemu" [5].

W tradycyjnych podejściach do prowadzenia projektów informatycznych, takich jak V-model czy model kaskadowy zakłada się, że wszystkie wymagania zostaną zdefiniowane, a cały proces zakończy się, przed fazą projektowania i implementacji. Wymaga się, aby wszystkie wymagania zostały zdefiniowane i odpowiednio dobrze opisane. Nie dopuszcza się możliwości zmiany wymagań po ich pierwotnym ustaleniu, choć na przykład Royce dopuszcza iteracje i podkreśla rolę sprzężenia zwrotnego w procesie wytwarzania w modelu kaskadowym [6].

Metodyki zwinne określane także jako lekkie, zakładają inne podejście do pozyskiwania wymagań. W projektach prowadzonych zgodnie z ich wytycznymi dopuszcza się zmiany w trakcie trwania projektu jako jego naturalny element. W początkowej fazie definiowane się wymagania zgodne z obecnym stanem wiedzy. W czasie trwania projektu mogą one ulec zmianie, co kontrolowane jest przez wyznaczonych do tego członków zespołu.



Rys. 7.1. Podział dziedzinowy inżynierii oprogramowania, [5]

Zwinne wytwarzanie oprogramowania oparte jest o zasadę iteracyjnego (przyrostowego) wytwarzania oprogramowania. Kolejne etapy procesu zamknięte są w iteracjach, w których za każdym razem przeprowadza się testowanie wytworzonego kodu, zebranie wymagań, planowanie rozwiązań itd. W odróżnieniu od tradycyjnego modelu kaskadowego w podejściach zwin-

nych nie wymaga się zdefiniowania wszystkich wymagań na samym początku trwania projektu. Do pierwszej iteracji przystępuje się z definicją najważniejszych oczekiwań klienta w ramach następnych iteracji definiowane są kolejne wymagania, oraz następuje ich stopniowa implementacja i jak najszybsze zaimplementowanie "najmniejszej sprzedawalnej jednostki" (ang. smallest saleable unit), czyli minimum systemu które można sprzedać klientowi.

Metody zwinne (na przykład Scrum [15]) nastawiona są wytwarzanie oprogramowania wysokiej jakości oraz na jak najszybsze przedstawienie klientowi działającego prototypu produktu. W projekcie następują częste inspekcje wymagań i rozwiązań włącznie z procesem adaptacji. Pozawala to na wczesne wykrywanie błędów. Cechą charakterystyczną dla metodyk zwinnych jest angażowanie klienta w proces powstawania oprogramowania. Dzięki aktywnemu uczestnictwu interesariuszy, polegającym przede wszystkim na definicji wymagań i testowaniu przygotowanych rozwiązań, możliwe jest natychmiastowe reagowania na zaistniałe nieprawidłowości.

Metodyka ta najczęściej znajduje zastosowanie w małych zespołach programistycznych, w których nie występuje problem komunikacji, przez co nie trzeba tworzyć rozbudowanej dokumentacji kodu.

Najistotniejszym pojęciem używanym w inżynierii wymagań jest „wymaganie". Według słownika języka polskiego „wymaganie jest to zespół warunków, które ktoś lub coś musi spełniać, jako synonimy tego pojęcia wymieniając słowa: pretensja, żądanie, oczekiwanie" [18]. W słowniku poświęconym terminom z dziedziny inżynierii oprogramowania można przeczytać że, wymaganie to:

(1) Stan lub zdolność potrzebna użytkownikowi w celu rozwiązania problemu lub osiągania celów.

(2) Stan lub zdolność, które muszą być spełnione przez system lub element, tego systemu dla spełnienia umowy, standardu, specyfikacji, lub innej formalnie nałożonej dokumentacji.

(3) Udokumentowane przedstawienie warunków lub zdolność opisanych w (1) i (2) [8].

Dean Leffingwell w swoje publikacji określa wymaganie jako: „możliwość rozwiązanie problemu i osiągnięcia celu, wymaganego przez użytkownika" [5], a także „możliwość spełnienia umowy, normy specyfikacji lub innej narzuconej dokumentacji, która musi mieć system lub komponent" [5].

Dla łatwiejszego zarządzania wymaganiami i odróżnienia ich ze względu na ich istotę dokonuje się podziału wymagań. Najczęściej spotykanym w literaturze podziałem jest ten na wymagania funkcjonalne i wymagania niefunkcjonalne, określane także jako jakościowe [7].

Wymagania funkcjonalne dotyczą tego co ma realizować system: jakie ma spełniać funkcje, jakich dostarczać usług, jak zachowywać się w określonych sytuacjach.

Ograniczenia dotyczące tego, jak system powinien realizować swoje zadania; np. wymagania dotyczące koniecznych zasobów, współpracy z określonymi narzędziami i środowiskami, zgodności z normami i standardami, a także przepisami prawnymi, w tym dotyczącymi tajności i prywatności. Wymagania niefunkcjonalne można dalej poddać dodatkowej klasyfikacji, w zależności od kategorii jakiej dotyczą. Bardzo powszechny jest podział FURPS. Pierwsza litera oznacza wymagania funkcjonalne, natomiast pozostałe - wymagania poza funkcjonalne.

U – użyteczność (ang. usability), oznacza łatwość użytkowania systemu. Takie wymagania można precyzować np. poprzez maksymalny czas szkolenia pracownika, liczba kontaktów ze wsparciem klienta, liczbą sytuacji, w których konieczne jest skorzystanie z systemu pomocy.

R – niezawodność (ang. reliability), może być mierzona poprzez: średnią liczbę godzin pracy bez awarii (ang. MTBF - Mean Time Between Failure), maksymalną liczbą godzin w miesiącu w ciągu których system może być wyłączony w celach pielęgnacyjnych (ang. maintenance) - ma to znaczenie szczególnie w przypadku systemów, które muszą pracować na okrągło - np. systemy bankowości elektronicznej

P – wydajność (ang. performance) - mierzona np. liczbą transakcji, którą system jest w stanie obsłużyć w ciągu godziny, liczbą użytkowników, którzy mogą być zalogowani jednocześnie do portalu.

S – bezpieczeństwo (ang. security), to wymagania związane z szyfrowaniem, polityką praw, itp. [8].

Często spotykanym rodzajem wymagań wyodrębnionym z wymagań niefunkcjonalnych są ograniczenia, które określają granice rozwiązania (ang. constraints). Niezależnie od tego jak problem jest rozwiązany, ograniczenia muszą być respektowane. Dla przykładu Amerykańskie Stowarzyszenie Inżynierów Kolei publikuje najlepsze praktyki dotyczące budowy systemów

w kolejnictwie. Systemy produkowane dla kolejnictwa (również systemy IT) muszą być zgodne z określoną normą. [10]

7.1. Kształcenie w obszarze inżynierii wymagań

7.1.1. Kształcenie na uczelniach polskich

Przegląd programów nauczania polskich uczelni technicznych pod kątem kształcenia w obszarze inżynierii wymagań wskazuje, że zarówno zakres, jak i intensywność edukacji w dziedzinie inżynierii w znacznym stopniu zależy od: poziomu edukacji (stopnia studiów), jak i specjalizacji na kierunku informatyka. W większości uczelni inżynieria wymagań nauczana jest w ramach szerszych zagadnień inżynierii oprogramowania lub analizy i modelowania systemów. W ramach badań autorzy dokonali przeglądu metod nauczania inżynierii wymagań na czterech uczelniach: Politechnice Warszawskiej [19], Wojskowej Akademii Technicznej [20], Polsko-Japońskiej Wyższej Szkole Technik Informacyjnych [21] i Wyższej Szkole Informatyki i Zarządzania [22] .

Zagadnienia inżynierii wymagań zasadniczo prezentowane są na studiach I stopnia. Na studiach drugiego stopnia przyjmuje się na ogół założenie pełnej znajomości zagadnień związanych z inżynierią wymagań wynikającą z dwóch obowiązujących obecnie nurtów w zakresie procesów wytwórczych, a mianowicie zwinnej (agile – zwykle w wydaniu SCRUM [15]) i tradycyjnej (czy tzw. klasycznej – reprezentowanej przez RUP) inżynierii wymagań. Warto tu również zaznaczyć, że zasadniczym sposobem weryfikacji wiedzy i umiejętności w tym zakresie jest realizacja projektów zespołowych, które znacznie wykraczają swoją złożonością poza ”tzw. projekty akademickie” – angażując zasoby niejednokrotnie przekraczające poj. grupę studencką, tj. których liczność jest większa niż 20.

Studenci studiów I stopnia w procesach pozyskiwania wymagań w ramach grup projektowych wyodrębniają ze swego składu interesariuszy, reprezentujących zamawiającego projekt i przygotowują wspólnie z wykładowcą dokumenty definicji projektu, które następnie zespół projektowy przekształca w artefakty etapu rozpoczęcia projektu (zgodnie z procesem

wytwórczym RUP). Ten krok jest ważny, ponieważ dokumenty te są poddawane analizie lingwistycznej przez zespół projektowy na podstawie, której ustalani są kandydaci na obiekty projektowe (w pierwszej kolejności stanowią one kandydatów na pojęcia słownikowe, a następnie po odpowiedniej filtracji grup rzeczownikowych i czasownikowych stanowią kandydatów na: aktorów, klasy, atrybuty oraz przypadki użycia i operacje). W ramach projektu studenci, dla uzyskania akceptacji opracowanych przez zespół projektowy scenariuszy, wykorzystują sesje CRC (zwykle tylko dla kluczowych przypadków użycia systemu) odgrywając „role" obiektów projektowych zgodnie ze zidentyfikowanym scenariuszem). Warto tu zaznaczyć, że obie te techniki: mechanistyczna analiza lingwistyczna i systematyczna, jako sesje CRC, są stosowane przez zespoły realizujące procesy inżynierii wymagań zarówno zwinnie, jak i klasycznie.

Ze względu na dużą liczność zespołów projektowych dużą wagę przywiązujemy do narzędziowego wsparcia procesów gromadzenia, zarządzania i śledzenia wymagań przez zespoły projektowe. W tym zakresie wykorzystujemy zarówno środowiska desktopowe (do pracy lokalnej) jak i serwerowe, dostępne przez platformę WEB. Nauka narzędzi realizowana jest w formie samokształcenia na podstawie przygotowanych materiałów szkoleniowych i praktycznie weryfikowana jest podczas warsztatów i przeglądów projektu. Warto tu również zaznaczyć, że każdy uczestnik projektu nie tylko uczestniczy w jego realizacji, ale również mierzy swój wysiłek, korzystając z metryk wspieranych przez platformy narzędziowe. Dzięki temu, jako wykładowcy, mamy szansę zobiektywizowanej oceny nakładów pracy poniesionych przez poszczególnych studentów jako członków zespołów projektowych. Aby osiągnąć cele realizowanych przedmiotów, zespoły część wymagań modelują:

1. zwinnie: dla metodyki SCRUM tworząc opisy sytuacji (ang. user stories), artefakty w postaci wymaganych planów, jako zaległości produktowe, wersji i sprintu poznając odpowiednie praktyki: spotkania produktowe, sprintu, retrospektywne;
2. zgodnie z procesem RUP: w oparciu o model FURPS, tworzą: wymagania słownikowe - TERM, żądania udziałowca - STRQ, cechy systemu – FEAT, wymagania typu przypadki użycia – UC, wymagania poza funkcjonalne (dodatkowe) – SR;

Istotne w procesie edukacji inżynierii wymagań jest również nauczenie dokumentowania wymagań. Dlatego studenci opracowują specyfikacje wymagań w oparciu o standardowe wzorce: formalny, nieformalny, tabelaryczny, czy RUP. Zwykle najwięcej trudności przysparza studentom opracowanie poprawnego modelu wymagań wyrażonego w języku UML, a w nim: procesy strukturalizacji wymagań oraz zarządzanie ich złożonością.

Z tego względu szczególnie istotne jest dokonanie przeglądu wymagań, najlepiej w oparciu o jednolite kryteria (Tabela 7.1).

Tab. 7.1. Formularz oceny wymagań opracowanych zgodnie z procesem wytwórczym RUP.

	Artefakty oceniane podczas przeglądu	**Max**
1	Dokument: przedstawienie problemu - dokument wykonawcy	5
2	Dokument: wizja projektu Dyplom	5
3	Raport: wyniki analizy lingwistycznej - lista kandydatów na usługi (przypadki użycia) i klasy systemu	5
4	Dokument: Słownik systemu Dyplom – z pojęciami w notacji BNF.	5
5	Opracowanie planu (zaległości produktowych i sprintu)	5
6	Raport: wymagania typu STRQ (żądania udziałowców)	5
7	Raport: wymagania typu TERM (słownikowe)	5
8	Raport: wymagania typu FEAT (cechy systemu)	5
9	Raport: wymagania typu: UC (wymagania funkcjonalne)	5
10	Raport: widok śledzenia żądań udziałowców na cechy systemu	5
11	Raport: model przypadków użycia	5
12	Raport: modele aktywności (dla wybranych przypadków użycia)	5
13	Raport: Definicje kart CRC + przykładowe scenariusze	5
14	Tablice: jako widoki realizacji planów, przez członków zespołu, na platformie jazz	5
	Razem:	70

Drugi stopień edukacji w zakresie inżynierii wymagań, zasadniczo związany jest z formalizacją wymagań oraz wprowadzeniem specjalizowanych procesów wytwórczych (np.: SOMA) – dla zachowania spójności

w programach nauczania[1] oraz standaryzacją koncepcji architektonicznych (od MDA, do MDSD) - ze względu na specjalizację kierunków nauczania (sieci komputerowe, multimedia, systemy wbudowane, bezpieczeństwo SI) niezbędne jest rozszerzenie zagadnień z inżynierii oprogramowania o zagadnienia dotyczące inżynierii systemów (ang. Systems Engineering).

W bieżącym roku przeprowadziliśmy eksperyment, którego wyniki są zadziwiająco pozytywne, polegający na realizacji przez studentów II stopnia podczas zajęć laboratoryjnych środowiska do modelowej weryfikacji semantyki modelu wymagań opracowanego przez studentów stopnia I, wykorzystując opracowany przez siebie DSL (i odpowiadający mu – zdefiniowany przez studentów profil UML[2]).

7.1.2. Kształcenie na uczelniach szwedzkich

System kształcenia inżynierii oprogramowania i inżynierii wymagań w Szwecji oparty jest na dwóch celach edukacyjnych: po pierwsze na połączeniu teorii z praktyką poprzez użycie wykładanych procesów, technik i metod w projektach i po drugie na pracy grupowej w formie projektu studenckiego. Oba te założenia powodują przesunięcie ciężaru edukacyjnego w stronę projektu i zminimalizowanie ilości materiału wykładanego na rzecz samodzielnej edukacji w oparciu o materiały z kursów. Wszystkie kursy z inżynierii oprogramowania oferowane na uniwersytecie w Lund są oparte o projekt studencki w grupach do sześciu studentów . Dzięki położeniu nacisku na projekty, studenci doświadczają typowych problemów w projekcie jak problemy z komunikacją, nierówne zaangażowanie członków zespołu, brak motywacji oraz nieprzewidziane sytuacje. Każdy z projektów zakończony jest raportem, w którym studenci opisują swoje doświadczenia oraz analizują co mogliby zrobić inaczej, a co zrobili dokładnie tak samo.

---

[1] W ramach studiów I stopnia studenci zwykle budują złożone rozwiązania portalowe, natomiast w ramach drugiego stopnia rozbudowują (bądź tylko modyfikują) je do modelu usługowego, budując rozwiązanie w architekturze SOA.

[2] Projekt dotyczył walidacji wymagań, wobec systemu wspomagania obsługi egzaminów dyplomowych na uczelni wyższej. Weryfikacja narzędzia polegała na zapisaniu przez wszystkich członków zespołu w utworzonym profilu modelu swojego egzaminu dyplomowego (na studiach I stopnia).

Kursy na szwedzkich uczelniach są intensywniejsze w stosunku do polskiego systemu. Rok akademicki dzieli się na cztery „okresy nauczania", każdy zakończony tygodniową sesją egzaminacyjną. W każdym okresie, studenci intensywnie studiują niewielką ilość przedmiotów. Dla przykładu, kurs inżynierii wymagań oferowany przez Uniwersytet w Lund [11] jako odrębny kurs na poziomie zawansowanym trwa siedem tygodni: zawierając w sobie sześć wykładów po dwie godziny, pięć ćwiczeń po dwie godziny i projekt, którego oszacowany wkład każdego członka zespołu wynosi około trzy tygodnie pracy. W rezultacie Szwedzcy studenci w większości studiują dwa kursy po 7.5 punktów ECTS jednocześnie. Większa liczba kursów stanowi poważne obciążenie czasowe. Intensyfikacja przebiegu kursów wraz ze zmniejszeniem ich ilości minimalizuje liczbę równoległych tematów studiowanych przez studentów. Dzięki temu mogą oni zagłębić się w szczegóły zagadnień poruszanych na kursach i w czasie projektu. Poprzez większą ilość zajęć w tygodniu, studenci częściej powracają do wykładanego materiału i dzięki temu prawdopodobieństwo zapamiętania go na dłużej wzrasta.

Cele kursu inżynierii wymagań na uniwersytecie w Lund są podzielone na trzy kategorie. Do pierwszej kategorii zalicza się cele związane z pozyskaniem odpowiedniej wiedzy w inżynierii wymagań. W tym przypadku studenci kończący kurs powinni rozumieć podstawowe cele i zakorzenione problemy inżynierii wymagań dla systemów informatycznych. Powinni oni rozumieć role interesariuszy w procesie inżynierii wymagań oraz poprawnie zdefiniować typ projektu w którym pracują (kontrakt lub produkt przeznaczony dla danego rynku). Studenci powinni rozumieć pojęcie wymagań funkcjonalnych oraz umieć rozróżnić wymagania funkcjonalne od niefunkcjonalnych oraz zdefiniować je dla danego projektu. Studenci powinni również rozumieć role dokumentacji wymagań oraz specyfikacji wymagań, włączając w to najważniejsze kryteria jakościowe dobrej specyfikacji wymagań. Wymaga się również od studentów zrozumienia różnicy pomiędzy inżynierią wymagań oraz procesem projektowania oprogramowania jak również podstawowej wiedzy na temat narzędzi używanych w inżynierii wymagań oraz praktyki w przemyśle w oparciu o podaną literaturę empiryczną.

Do celów powiązanych ze zdobyciem określonych umiejętności poprzez uczestnictwo w kursie zaliczone zostały: umiejętność skutecznego definiowania, dokumentowania, walidacji oraz nadawania priorytetów wymaganiom.

Dodatkowo kurs zawiera naukę umiejętności używania kilku technik do wyżej wymienionych zadań w zależności od rodzaju kontekstu i projektu oraz umiejętności określania jakości już istniejących specyfikacji wymagań. Kurs uczy również umiejętności czytania ze zrozumieniem artykułów naukowych z dziedziny inżynierii wymagań. Studenci otrzymują osiem artykułów do przeczytania i przedyskutowania w ramach kursu.

Do celów powiązanych z ostatnią kategorią zalicza się cele związane z przekazaniem odpowiedniego nastawienia studentów do zagadnień inżynierii wymagań. W tym przypadku przekazuje się studentom opinie, że lepiej jest zrobić coś dobrze od początku niż później poprawiać oraz przekazuje się opinie, że wysoka jakość wymagań pozytywnie wpływa na wysoką jakość produktu. Kurs stara się również przekazać, że użyteczność oraz dopasowanie metody do kontekstu są bardzo ważne w inżynierii wymagań oraz, że należy zawsze brać pod uwagę czynnik ludzki. Dodatkowo, przekazywane jest przeświadczenie, że inżynier wymagań powinien pomagać interesariuszom w dążeniu do zdefiniowania realistycznego produktu który można wytworzyć w ramach podanego budżetu.

Kurs inżynierii wymagań umiejscowiony jest na poziomie zaawansowanym (w dniu publikacji tej pracy wydział informatyki uniwersytetu w Lund oferował tylko jednostopniowe studia magisterskie zakończone tytułem magistra inżynieria cywilnego ze specjalizacją w jednej z poniższych dziedzin). Kurs jest kursem obieralnym i obejmuje studentów kierunków programu inżynierii danych, ekonomii przemysłowej, elektroniki oraz telekomunikacji i komunikacji danych. Kurs jest również otwarty dla studentów zagranicznych, osób pracujących w przemyśle oraz studentów innych kierunków na uniwersytecie w Lund. W przypadku słuchaczy pracujących w przemyśle posiadających duże praktyczne doświadczenie w pracy jako inżynierowie wymagań, projekt kursu jest indywidualnie dopasowywany do potrzeb i zainteresowań słuchaczy. Często temat projektu jest bezpośrednio powiązany z wyzwaniami z którymi zmierzają się słuchacze pracujący w przemyśle. Certyfikat IREB nie jest oferowany dla studentów kursy inżynierii wymagań ze względu na rozbieżności pomiędzy zakresem kursu, w szczególności w zakresie inżynierii wymagań dla produktów przeznaczonych na otwarty rynek a zakresem podstawowego kursu certyfikatu IREB.

Uniwersytet w Lund nie umożliwia kształcenia inżynierów wymagań. Studenci mogą jednak wybrać temat pracy dyplomowej z zakresu inżynierii wymagań. Prace te często odbywają się w ścisłej współpracy z przemysłem i skupiają się na zrozumieniu autentycznych sytuacji i wyzwań inżynierii oprogramowania jak również zaproponowanie i sprawdzenie rozwiązań do tych problemów. Do firm w których prace dyplomowe się odbyły lub obecnie odbywają zaliczmy: Sony Mobile, ST Ericsson, Ericsson, ABB, Anoto, Axis Communications oraz wiele firm doradczych. Ze względu na stopień zaawansowania projektów z przemysłem a co za tym idzie wymagany większy wkład pracy niż budżet kursu inżynierii wymagań, projekty z przemysłem realizowane są głownie w oparciu o prace magisterskie w wymiarze dwudziestu tygodni pracy. Podczas pracy magisterskiej w przemyśle, studenci są opłacani zgodnie z obowiązującym wynagrodzeniem dla studentów oraz otrzymują miejsce pracy, dostęp do zasobów firmy i bardzo często szybko integrują się z nowym środowiskiem. Dzięki pomocy doświadczonych nauczycieli prace dyplomowe zawierają użyteczne dla przemysłu analizy lub rozwiązania jak i prezentują wymaganą oryginalność naukową. Większość prac magisterskich jest po obronie publikowana jako artykuł na konferencji lub w czasopiśmie, np. Information and Software Technology (IST).

W ramach kursu inżynierii wymagań studenci poznają i pracują z narzędziami otwartymi jak i dostępnymi za licencją. Z narzędzi dostępnych w ramach licencji studenci mogą używać IBM Rational Focal Point do ustalenia priorytetów wymagań oraz IBM Rational Doors [16] do dokumentacji wymagań. Dodatkowo wprowadzone są do kursu dwa narzędzia stworzone przez ośrodki badawcze w Lund i Calgary. Studenci mogą użyć prototypowego narzędzia reqT [12] do modelowania wymagań tekstowych. ReqT napisany jest w języku Scala i umożliwia tworzenie i zarządzanie modelami wymagań wykorzystując uniwersalne kolekcje, połączenie języka modelowania i bezpiecznego typu. ReqT umożliwia również współpracę z aplikacjami arkusza kalkulacyjnego oraz publikowanie wymagań w Internecie lub jako dokumenty PDF. ReqT oferuje rozszerzenie otwartego, wewnętrznego DSL (języka specyficznej domeny) o własne semantyki modelowania. Drugim narzędziem stworzonym w ośrodku badawczym w Calgary jest Release Planner stworzony przez ekspertów w zakresie planowania kilku wersji oprogramowania [13]. Release planner używa algorytmów ogólnych do zoptymalizowania ilości

i zakresu wymagań które powinny być zawarte w każdej z wersji oprogramowania i umożliwia analizę „co by było gdyby" na przykład co by były gdyby jakieś wymaganie zostało usunięte, jak zmienia to ogólne plany i czy powoduje jakieś opóźnienia. Studenci używają tego narzędzia do określenia zakresu kilku wersji oprogramowania w oparciu o określone wymagania.

7.2. International Requirements Engineering Board

IREB (International Requirements Engineering Board), czyli Międzynarodowa Rada Inżynierii Wymagań powstała w 2006 roku w Fürth w Niemczech. IREB stworzyła i nadzoruje międzynarodowy system akredytacji i certyfikacji w dziedzinie inżynierii wymagań, zwany CPRE (Certyfikowany Specjalista Inżynierii Wymagań (Certified Professional in Requirements Engineering). Celem IREB jest stworzenie i dalsza budowa jednolitego, powszechnego, międzynarodowego systemu kwalifikacji zawodowych dla inżynierów wymagań [9][12].

Jako podstawowe objawy złego podejścia do inżynierii wymagań IREB określa przede wszystkim brak lub niedoprecyzowanie wymagań [1][2][3][4]. Za typowe przyczyny takiego stanu rzeczy uznaje pochopne przyjmowanie pewnych założeń co do definicji systemu za pewne i z góry oczywiste, uznając że nie ma potrzeby konsultacji tych aspektów miedzy wykonawcą systemu a zamawiającym. Problemem są również zbyt wygórowane lub ambitne oczekiwania klientów.

Zgodnie z podejściem IREB istnieją cztery główne działania związane z wymaganiami, które mają kluczowe znaczenie w procesie wytwarzania oprogramowania. Tymi czterema filarami są:

- pozyskiwanie wymagań,
- dokumentacja,
- walidacja i negocjowanie,
- zarządzanie wymaganiami.

7.2.1. Pozyskiwanie wymagań

Jako podstawę do pozyskania wymagań możemy traktować określony kontekst systemu, w ramach którego działają inżynierowie wymagań. Określa on domenę działania oraz definiuje źródła wymagań. Za źródła wymagań uznać można wszystkie osoby bezpośrednio lub pośrednio zainteresowane w powodzenie przedsięwzięcia jakim jest tworzenie nowego systemu informatycznego, a także wszelkie rodzaje artefaktów pojawiające się w tym procesie. W świetle tej definicji za źródła wymagań uznać możemy np. udziałowców, dostępna dokumentacje czy tak zwane systemy spadkowe. Obecnie ponad 80% obecnych systemów to systemy oparte o rozwinięcie już istniejącego rozwiązania.

Udziałowcy, jako najważniejsze źródło wymagań, musza być obdarzeni szczególną uwagą ze strony zespołu projektowego. Bardzo istotne jest ich aktywizowanie tak, aby brali oni czynny udział w projekcie, a nie byli tylko jego biernymi obserwatorami. W tym celu zgodnie z kulturą organizacyjną firmy, należy uzgodnić ustnie lub też w formie pisemnej, role i kompetencje udziałowców w całym procesie. Zabezpiecza to przed brakiem motywacji i konfliktami, mogącymi negatywie wpłynąć na powodzenie przedsięwzięcia.

Istotnie jest, aby przy pozyskiwaniu wymagań można było poznać jak ważne dla zadowolenia interesariuszy są poszczególne wymagania. Pozwala to na ustalenie priorytetów wymagań tak, aby jak najbardziej spełnić oczekiwania udziałowców. Pomocny w tym procesie jest model Kano. Dzieli on wymagania na trzy grupy:

- czynniki podstawowe (synonim: wzbudzające niezadowolenie),
- czynniki wydajności (synonim: wzbudzające zadowolenie),
- czynniki entuzjazmu (synonim: wzbudzające zachwyt).

W celu odpowiedniego pozyskania (świadomych, nieświadomych oraz pozostałych) wymagań udziałowców stosuje się szereg technik zbierania wymagań. Istnieje wiele czynników definiujących wybór odpowiedniej techniki. IREB zaleca wzięcie pod uwagę m. in. czynniki ryzyka, wpływy osobowe, wpływy organizacyjne, wpływy funkcyjno-treściowe oraz zamierzony poziom szczegółowości wymagań. Dobór odpowiedniej technik jest bardzo ważną umiejętnością i jest kluczowy dla powodzenia projektu. Dlatego należy po-

święci odpowiednio dużo czasu na wybór adekwatnej technik, pamiętając o tym, że najlepsze efekty osiągnąć można przy ich łączeniu.

7.2.2. Dokumentacja wymagań

W inżynierii wymagań konieczne jest dokumentowanie wszelkich istotnych informacji. Techniki dokumentacji są w zależności od przyjętego modelu bardziej lub mniej formalnymi sposobami reprezentacji wymagań, począwszy od zapisu w formie języka naturalnego po diagramy o zdefiniowanej semantyce. Ze względu na to, że nad pozyskiwaniem wymagań pracuje zazwyczaj wiele osób, istotna jest zapewnienie właściwej komunikacja między nimi.

Oddzielną kwestią pozostają formy dokumentacji wymagań. IREB proponuje trzy skuteczne metody w zależności od perspektywy, dla której opisujemy wymagania: (1) dokumentacja wymagań w języku naturalnym, (2) koncepcyjne modele wymagań takie jak diagramy przypadków użycia, diagramy klas, diagramy aktywności czy diagramy stanów, (3) mieszane formy dokumentacji wymagań.

Zgodnie z założeniami IREB, główną częścią dokumentu wymagań pozostają wymagania dla systemu. Poza samymi wymaganiami, w zależności od celu powstania dokumentacji, dokumenty wymagań zawierają także informacje na temat kontekstu systemu, warunków zatwierdzenia lub, przykładowo właściwości implementacji technicznej. Aby zapewnić możliwość zarządzania dokumentami wymagań, dokumenty te muszą mieć formalną strukturę. W tym zakresie IREB zaleca stosowanie referencyjnej struktury dokumentu zgodnej z IEEE-830 [17].

W praktyce okazuje się, że istnieje dużo pozytywnych skutków wykorzystywania struktur referencyjnych dla dokumentów wymagań. Przykładowo wykorzystanie struktur referencyjnych ułatwia korzystanie z dokumentów wymagań w późniejszych czynnościach konstrukcyjnych (np. w definiowaniu przypadków testowych). Na ogół struktury referencyjne nie mogą być stosowane dla dokumentu wymagań na zasadzie „jedna struktura dla jednego dokumentu”, jako że struktura treści konkretnego dokumentu wymagań musi być często szczegółowo dopasowywana do okoliczności właściwych dla domeny, firmy czy projektu.

7.2.3. Negocjowanie i walidacja wymagań

Celem walidacji wymagań jest sprawdzenie czy wymagania spełniają zdefiniowane kryteria jakości (np. poprawność lub kompletność) tak, aby wykryć i skorygować błędy w wymaganiach tak wcześnie, jak to możliwe na etapie ich definicji.

Nierozwiązane konflikty w wymaganiach systemu oznaczają, że jakiś zbiór wymagań interesariuszy nie może być zrealizowany, lub, że działający system albo nie zostaje w ogóle przyjęty, nie będzie w ogóle używany, lub będzie używany w niedostatecznym stopniu. Celem negocjowania wymagań jest określenie wspólnego i uzgodnionego porozumienia, które pozwoli uniknąć takich sytuacji.

Drugim aspektem poruszanym równolegle z walidacją wymagań jest ich negocjowanie. Celem negocjowania wymagań jest ustalenie wspólnej dla wszystkich interesariuszy interpretacji wymagań wobec tworzonego systemu. Czynności wchodzące w skład negocjowania wymagań to:

- rozpoznanie konfliktów,
- analiza konfliktów,
- rozwiązywanie konfliktów,
- dokumentowanie rozwiązań konfliktów.

7.2.4. Zarządzanie wymaganiami

Aby zarządzać wymaganiami systemowymi podczas całego cyklu życia, konieczne jest zebranie informacji o wymaganiach w formie uporządkowanych atrybutów. Aby zdefiniować strukturę atrybutów wymagań, stosuje się odpowiednie zestawienia, które przedstawia się albo w postaci tabeli, albo w postaci modelu informacji.

W praktyce projektowej liczba wymagań i zależności pomiędzy nimi nieustannie wzrasta. Żeby skutecznie zarządzać złożonymi wymaganiami, konieczna jest selekcja dostępu, co pozwala na dobór wymagań potrzebnych tylko do bieżącego zadania.

Bardzo istotne w kontekście określanie harmonogramu prac i istotności wymagań zamawiających, jest ustalenie priorytetów wymagań. Nadawanie priorytetów wymaganiom odbywa się w różnym czasie, podczas różnych za-

dań w projekcie i na podstawie różnych kryteriów. Przygotowanie do nadawania priorytetów odbywa się według następujących zasad: (1) określenie celów i ograniczeń procesu nadawania priorytetów, (2) określenie kryteriów nadawania priorytetów, (3) określenie istotnych interesariuszy, (4) wybranie wymagań lub innych wytworów, których priorytety mają być ustalone.

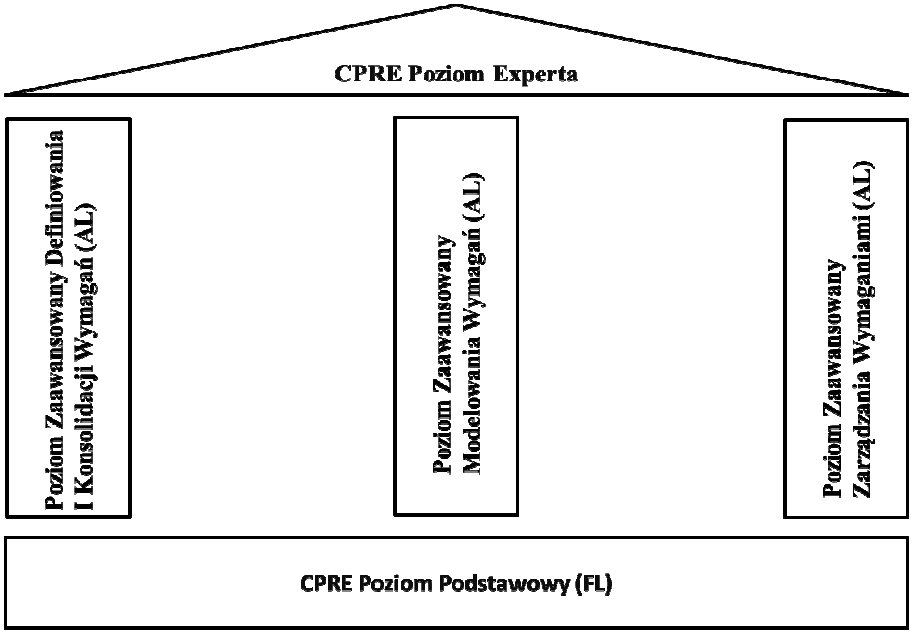
Wymagania zmieniają się podczas całego cyklu życia oprogramowania. Zmianami wymagań zarządza się i realizuje je w systematycznym, zdefiniowanym procesie zarządzania zmianami. W tym procesie rada kontroli zmian odpowiada za przetwarzanie przychodzących żądań zmian. Do zadań rady kontrolowania zmian należą: (1) klasyfikacja przychodzących żądań zmian, (2) określenie wysiłku potrzebnego do przeprowadzenia zmiany, (3) ocena relacji kosztu dla żądanej zmiany, (4) stworzenie nowych wymagań zgodnie z żądanie zmiany, (5) decyzja czy zaakceptować, czy odrzucić zgłoszenie zmiany, (6) określenie priorytetu zaakceptowanego zgłoszenia zmiany, (7) przydzielenie zaakceptowanej zmiany do konkretnego projektu.

Członkowie rady kontrolowania zmian to zwykle kierownik zarządzania zmianami, zleceniodawca, architekt oprogramowania, przedstawiciel użytkowników, kierownik do spraw jakości oraz inżynier wymagań.

7.2.5. Certyfikacja IREB

IREB stworzył i nadzoruje międzynarodowy system akredytacji i certyfikacji w dziedzinie inżynierii wymagań, zwany CPRE (Certyfikowany Specjalista Inżynierii Wymagań – Certified Professional in Requirements Engineering) [14]. Celem IREB jest stworzenie i dalsza budowa jednolitego, powszechnego, międzynarodowego systemu kwalifikacji zawodowych dla inżynierów wymagań.

Stworzony przez IREB system składa się z trzech poziomów: podstawowego, zaawansowanego i eksperckiego. W skład każdego poziomu wchodzi kompendium wiedzy (tzw. sylabus), zasady akredytacji oraz egzaminów certyfikacyjnych i pytania egzaminacyjne.



Rys. 7.2. Schemat certyfikacji IREB, w oparciu o [14]

Poziom podstawowy dostępny jest w tej chwili po angielsku, francusku, niemiecku, hiszpańsku, portugalsku, po polsku i po szwedzku. IREB jest kierowana przez 12-osobowy Komitet Sterujący, w skład którego wchodzą przedstawiciele firm informatycznych, wyższych uczelni oraz innych organizacji. Inicjatywy IREB na forum międzynarodowym realizuje 35-osobowa Grupa Wspierająca.

Od 2007 roku ponad 11 000 fachowców pozytywnie przeszło egzaminy CPRE Poziomu Podstawowego – z tendencją wzrostową. Jak dotąd, większość egzaminów certyfikacyjnych odbywała się w Niemczech, Austrii i Szwajcarii, ale certyfikaty CPRE zdobyli już także fachowcy z Holandii, Hiszpanii, Bułgarii, Egiptu, USA, Kolumbii, Brazylii, Korei Południowej, Polski, Malezji oraz Indii.

W Polsce inicjatywa certyfikacji i szkoleń IREB rozwijana jest od roku 2011. Obecnie w Polsce działa dziewięcioosobowa Rada IREB złożona ze specjalistów zajmujących się inżynierią wymagań z uczelni i przemysłu informatycznego. Do zadań Rady należy: przetłumaczenie słownika terminologii oraz kompendium podstawowego CPRE na język polski, przetłumaczenie

i wydanie podręcznika do kursu podstawowego CPRE na język polski, tłumaczenie na język polski kompendiów zaawansowanych CPRE.

Podstawowy egzamin CPRE jest oferowany również studentom kończącym kursy inżynierii oprogramowania w dwu polskich uczelniach: Politechnice Warszawskiej oraz PJWSTK jako nieobowiązkowy element kształcenia kursu inżynierskiego.

7.3. Podsumowanie

Kształcenie inżynierów w obszarze inżynierii wymagań jest istotnym elementem wykształcenia współczesnego inżyniera informatyka. Choć w większości wypadków uczelnie nie dedykują tym zagadnieniom odrębnego kursu, to jednak inżynieria wymagań jest istotnym elementem kształcenia we wszystkich programach nauczania dla kierunku informatyka. W większości wypadków kształcenie to odbywa się metodami projektowymi, co należy podkreślić z satysfakcją. W odróżnieniu od programów kształcenia na uczelniach szwedzkich w Polsce zdecydowanie rzadziej wykonywane są „rzeczywiste" projekty przy współpracy z przemysłem. Z pewnością wynika to z ogólnych ułomności polskiego systemu kształcenia, który w niewystarczającym stopniu powiązany jest jeszcze z problemami i potrzebami przemysłu informatycznego.

W badaniach rynku widoczne jest też zainteresowanie przemysłu szkoleniami z tego zakresu. Pojawienie się takich organizacji jak IREB zdaje się potwierdzać potrzebę kształcenia i certyfikacji w tym obszarze. W polskich realiach jednak certyfikacja ta jest jeszcze na początku drogi. Zainteresowanie certyfikacją taką jak CPRE jest jak na razie ograniczone, ale biorąc pod uwagę trendy światowe powinno w najbliższym czasie wzrastać.

7.4. Bibliografia

1. The Standish Group, New Standish Group report shows more projects failing and less successful projects, http://www.standishgroup.com/newsroom/chaos_2009.php , 2009

2. KPMG, KPMG New Zealand Project Management Survey, 2010 http://www.kpmg.com/NZ/en/IssuesAndInsights/ArticlesPublications/Documents/Project-Management-Survey-report.pdf , 2010
3. Info-Tech Research Group, Flawed Requirements Trigger 70% of Project Failures: http://www.infotech.com/research/flawed-requirements-trigger-70-of-project-failures, 2006
4. Galorath, D.: *Software Project Failure Costs Billions – Better Estimation & Planning Can Help*, http://www.galorath.com/wp/software-project-failure-costs-billions-better-estimation-planning-can-help.php
5. Leffingwell, D.: *Zarządzanie Wymaganiami*, WNT, Warszawa, 2003
6. Royce, W.: *Managing the Development of Large Software Systems*, Proceedings of the IEEE Wescon Conference, 1970
7. Sommerville, Y, Sawyer, P.: *Requirements Engineering, Good Practices*, Pearson Education, 2003
8. IEEE Standards Coordinating Committee, I*EEE Standard Glossary of Software Engineering Terminology, IEEE Std 610*, IEEE, 1990
9. Strona Polskiego oddziału IREB dostępna jest pod adresem, www.ireb.org.pl, 2013
10. Strona Amerykańskiego Stowarzyszenia Kolejnictwa I infrastruktury znajduje się pod adresem, http://www.arema.org/publications/ , 2013
11. Strona kursy inżynierii wymagań na uniwersytecie w Lund dostępna jest pod adresem, http://cs.lth.se/ets170, 2013
12. Strona narzędzia ReqT dostępna jest pod adresem http://www.reqt.org/, 2013
13. Strona narzędzia Release Planner dostępna jest pod adresem, www.expertdecisions.com/, 2013
14. Strona organizacji IREB dostępna jest pod adresem, www.ireb.org, 2013
15. Opis metodologii SCRUM znajduje się na stronie, http://www.scrum.org/ 2013
16. Opis narzędzia Rational Doors znajduje się na stronie, http://www-01.ibm.com/software/rational/ 2013
17. IEEE, *IEEE Recommended Practice for Software Requirements Specifications, IEEE 830, 1998*
18. Słownik języka polskiego dostępny jest na stronie http://sjp.pwn.pl/

19. Informacje o Politechnice Warszawskiej znajdują się na stronie http://www.isep.pw.edu.pl/
20. Informacje o Wojskowej Akademii Technicznej znajdują się na stronie http://www.wat.edu.pl/
21. Informacje o Polsko-Japońskiej Wyższej Szkole Technik Informacyjnych znajdują się na stronie http://www.pjwstk.edu.pl/
22. Informacje o Wyższej Szkole Informatyki i Zarządzania znajdują się na stronie http://www.wit.edu.pl/

Rozdział 8

Uniwersyteckie Centrum Kompetencyjne Technologii Oprogramowania

Poniższy rozdział stanowi rodzaj 'case study', gdzie na przykładzie Uniwersyteckiego Centrum Kompetencyjnego Technologii Oprogramowania funkcjonującego przy Politechnice Gdańskiej przedstawiona została koncepcja powoływania i funkcjonowania centrów rozwijania technologii informatycznych tworzonych przy uczelniach wyższych przy współpracy z partnerami biznesowymi. Przedstawione zostały potrzeby powoływania takich jednostek organizacyjnych, możliwe obszary współpracy ale i bariery najczęściej występujące na ścieżce współpracy środowisk uczelni i biznesu. Rozdział wpisuje się w tematykę współpracy biznesu i środowisk uniwersyteckich w zakresie inżynierii oprogramowania.

Coraz więcej znaczących na rynku firm z różnych branż wydziela w ramach swoich organizacji tak zwane Centra Kompetencji (CK).

Sama idea centrum kompetencji jest ideą dość nową, o której wzmianki w literaturze biznesowej zaczęły pojawiać się w latach 2002 – 2003. Brak jest jednocześnie jednolitej definicji centrum kompetencji. Sama nazwa stosowana jest zamiennie w różnych formach, poczynając od sformułowań odnoszących się do tzw. Centrów Doskonałości (ang. Centrum of Exellence) poprzez Centra Kompetencji (ang. Competency Center) do Centrów Zdolności (ang. Capability Center).

W obrębie organizacji CK może odnosić się do grupy osób, departamentu lub wydzielonego obiektu/pomieszczenia. Termin ten może odnosić się również do sieci instytucji współpracujących ze sobą w celu podnoszenia jakości w zakresie wiedzy, usług czy produktów (np. The Rochester Area Colleges Center for Excellence in Math and Science) [3].

W instytucjach uniwersyteckich CK często odnosi się do zespołu pracowników skoncentrowanych na określonym obszarze badań; centrum może również obejmować pracowników różnych wydziałów z różnych dziedzin naukowych w celu rozwiązania interdyscyplinarnego problemu naukowego i wspierać ich pracę poprzez dostarczenie odpowiedniego zaplecza badawczego [6].

Bez względu na przyjętą nazwę funkcjonowanie Centrum Kompetencji odnosi się, w swojej najbardziej podstawowej formie, do powołania zespołu ludzi, którzy zawiązują współpracę i wykorzystują w jej ramach najlepsze praktyki w obszarze określonej dziedziny przedmiotu (nauki, biznesu) w celu osiągnięcia założonych wcześniej rezultatów (naukowych, biznesowych). Działalność tego zespołu powinna wspierać i pokrywać obszar głównych potrzeb:

- ***wsparcie*** - dla określonej dziedziny przedmiotu zespół CK powinien oferować wsparcie na płaszczyźnie biznesowej/naukowej, które może być dostarczane poprzez realizację usług lub też poprzez zapewnianie wiedzy eksperckiej z określonego obszaru;
- ***wiedza*** - zespół CK określa standardy, dobre praktyki, metodologie, narzędzia i wiedzę z obszaru dziedziny przedmiotu;
- ***obopólne uczenie się*** - dzielenie się wiedzą i dobrymi praktykami przez partnerów centrum, realizacja cyklicznych szkoleń, ustalenie ścieżek certyfikacji oraz rozwoju kompetencji, wzmacnianie spójności zespołu, definiowanie i standaryzowanie ról w zespole;
- ***efektywność*** - przygotowanie zbioru kryteriów oceny oraz metod ich pomiaru, umożliwiających wykazanie efektywności podjętej współpracy, osiągania przyjętych celów strategicznych, a przez to zasadność powołania CK;
- ***zarządzanie*** - dysponowanie ograniczonych zasobów (pieniędzy, ludzi, itp.) do wsparcia wszystkich niezbędnych obszarów podejmowanej współpracy w sposób zapewniający partnerom centrum efektywną realizację kluczowych inwestycji i projektów jak również tworzenie skali/rankingu ekonomicznej korzyści dla przygotowywanych alternatywnych projektów.

Należy zauważyć, że przedmiot współpracy może dotyczyć zarówno samej wiedzy, dostarczenia dobrych praktyk, szkoleń, jak również może

obejmować wsparcie techniczne czy technologiczne (serwis określonych rozwiązań technologicznych, testowanie lub wytwarzanie oprogramowania). Dodatkowo Centrum Kompetencji może być powołane w celu ożywienia i dokończenia podjętych i zawieszonych inicjatyw [2].

W praktyce najczęściej spotykane są, jednorodne pod względem prowadzonej działalności, partnerskie centra kompetencyjne, które przejmują rolę pomostu między klientami a przedsiębiorstwem. W takich centrach realizowane są zwykle usługi serwisowe i doradcze wspierające odbiorców końcowych usług czy produktów, a w odniesieniu do firm matek świadczone są usługi testowe, wdrożeniowe czy szkoleniowe.

Największe zainteresowanie projektowaniem i uruchamianiem tego typu centrów wykazują środowiska przedsiębiorstw, które szybciej poddają się przemianom rynkowym, gospodarczym i informatycznym niż jednostki naukowe. Przedsiębiorcy sprawniej nawiązują współpracę między sobą, przystępując do różnego rodzaju klastrów przemysłowych stanowiących przestrzennie skoncentrowaną grupę przedsiębiorstw, instytucji i organizacji powiązanych siecią pionowych i poziomych zależności, często o charakterze nieformalnym, która poprzez skupienie szczególnych zasobów pozwala osiągnąć tym przedsiębiorstwom trwałą przewagę konkurencyjną.

Nie spotyka się prawie wcale mieszanych struktur organizacyjnych, obejmujących partnerów z różnych środowisk branżowych, a już do wyjątków można zaliczyć centra łączące partnerów biznesowych i akademickich. Spowodowane jest to faktem, iż w większości przypadków środowiska naukowe (szczególnie jednostki państwowe) ze względu na rozbudowaną strukturę administracyjną, częsty brak procedur regulujących warunki i sposoby nawiązywania współpracy ze środowiskiem biznesowym oraz niechęć do inwestowania własnych środków finansowych w projekty badawczo-rozwojowe same przyczyniają się do wykluczania z procesu rozwoju gospodarki opartej na wiedzy i nie stanowią atrakcyjnego partnera dla środowisk zewnętrznych. Dodatkowo do najczęściej wskazywanych barier utrudniających nawiązanie współpracy przedsiębiorstw ze środowiskiem uczelni zaliczana jest nieznajomość w środowiskach biznesowych oferty strefy badawczo-rozwojowej (B+R) oraz trudność w uzyskaniu wyczerpującej informacji na temat potencjału badawczego uczelni [1].

Biorąc pod uwagę powyższe fakty należy tym bardziej przedstawiać pozytywne działania podejmowane w ramach inicjatyw akademickich nakierowanych na rozszerzenie współpracy nauki z biznesem i wskazywać obopólne korzyści takiej współpracy. Przykładem takim jest na pewno Uniwersyteckie Centrum Kompetencyjne Technologii Oprogramowania działające od 2011 roku przy Wydziale Zarządzania i Ekonomii Politechniki Gdańskiej.

Uniwersyteckie Centrum Kompetencyjne Technologii Oprogramowania (ang. University Competence Center - UCC) przy Zakładzie Zarządzania Technologiami Informatycznymi (ZZTI) funkcjonuje oficjalnie od 12 stycznia 2011 r. W tym dniu w Warszawie IBM Polska i Politechnika Gdańska podpisały umowę o utworzeniu UCC na Wydziale Zarządzania i Ekonomii.

Sama współpraca z IBM rozpoczęła się w roku 2007 i zainicjowana została na płaszczyźnie współpracy między IBM a Zakładem Zarządzania Technologiami Informatycznymi PG. W ramach programu Academic Initiative firma IBM przekazała uczelni licencje na wykorzystanie oprogramowania do celów dydaktycznych, a polscy partnerzy IBM zorganizowali dla pracowników zakładu merytoryczne szkolenia z dostarczonych metod i narzędzi. W ten sposób do zajęć ze studentami włączono aplikacje z rodziny Rational wspomagające zarządzanie przedsięwzięciami informatycznymi.

Kolejnym krokiem była organizacja praktyk studenckich. Studenci profilu *zarządzanie technologiami informatycznymi w przedsiębiorstwie* otrzymali ofertę udziału w przedsięwzięciach realizowanych przez IBM Polska, czy to w jego warszawskiej siedzibie, czy to poprzez zdalny dostęp do nieprzebranych materiałów szkoleniowych. Powodzenie pierwszej edycji praktyk spowodowało, że powołane w tym samym czasie Międzywydziałowe Koło Naukowe Badania Technologii Informatycznych znaczną część swojej aktywności zaczęło poświęcać na realizację projektów współinicjowanych przez IBM. Widocznym znakiem tego zaangażowania była m.in. wizyta na PG ciężarówki IBM prezentującej nowoczesne technologie.

8.1. Podstawowe zadania UCC

W ramach podpisanej umowy jako główne z zadań UCC wskazano wykorzystanie rodziny produktów IBM Rational - narzędzi do projektowania

i konstruowania aplikacji w wyznaczaniu kierunków dydaktycznych i zarządzaniu projektami informatycznymi [8].

Na mocy podpisanej umowy IBM zapewniło do celów edukacyjnych i badawczych zarówno oprogramowanie jak i ekspertów z dziedziny wykorzystywanych rozwiązań do merytorycznej opieki nad UCC oraz zobligowało się do przeprowadzenia szkoleń w zakresie kompetencji rozwijanych w Centrum.

Wydział Zarządzania i Ekonomii Politechniki Gdańskiej wprowadził do zajęć dydaktycznych oprogramowanie i materiały edukacyjne IBM, a także umożliwił wybranym pracownikom naukowym zdobycie certyfikatów w obszarach, w jakich rozwijane jest UCC. Centrum współpracuje z innymi ośrodkami akademickimi, a w jego działania zaangażowanych jest kilkudziesięciu studentów i doktorantów.

Korzyści z nawiązanej współpracy czerpią od początku obie strony współpracy jak również otoczenie biznesowe uczelni.

Dla studentów Politechniki Gdańskiej współpraca ta oznacza możliwość zapoznania się z nowoczesnymi technologiami i szansę na odbycie stażu w międzynarodowej korporacji.

IBM Polska otrzymuje od zespołu roboczego ZZTI szczegółowe analizy dotyczące badanych narzędzi informatycznych, jak również zleca część badań do realizacji w ramach tematów prac magisterskich i doktorskich realizowanych w zakładzie.

Firmy z otoczenia biznesowego PG, wykorzystujące oprogramowanie IBM, mogą bezpośrednio zwrócić się do ZZTI z pytaniami związanymi z wdrożonymi przez nich rozwiązaniami [7].

8.2. Rozszerzenie współpracy w zakresie badania technologii oprogramowania

Realizowane prace w ramach współpracy Centrum i firmy IBM miały do tej pory charakter o silnym nastawieniu na wsparcie dydaktyki i możliwość pracy na profesjonalnych narzędziach firmy IBM, mniej natomiast nastawione były na współpracę biznesową.

Zmianę w tym zakresie wprowadza rozszerzenie współpracy zawarte na podstawie umowy CAS, czyli „IBM Center for Advanced Studies (CAS)".

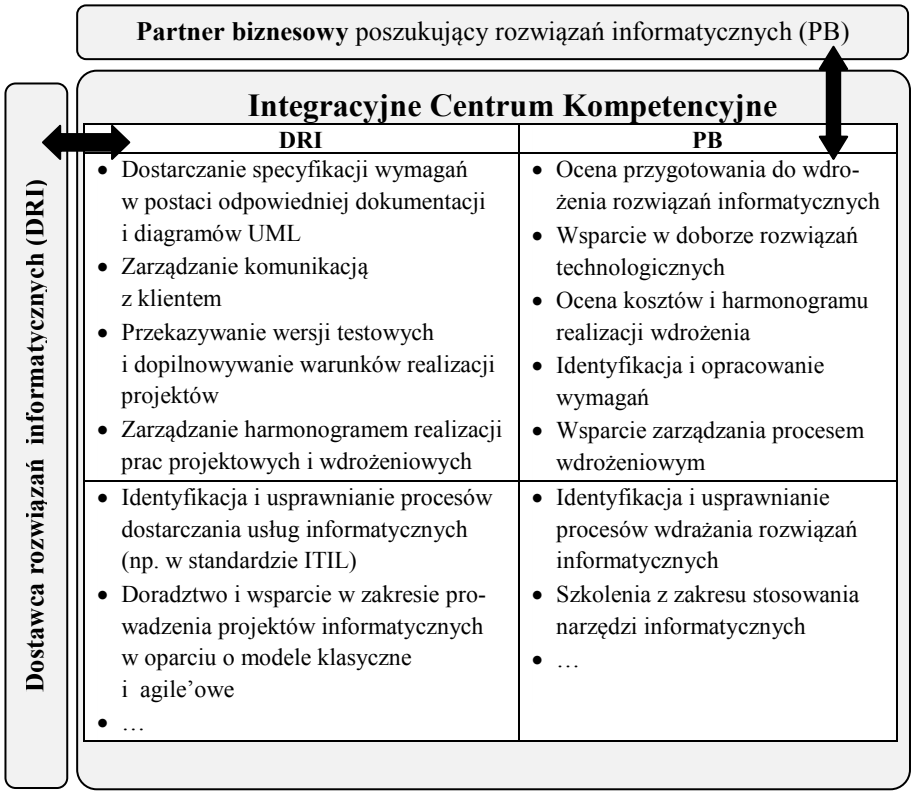
W ramach tej umowy obecnie funkcjonujące UCC przekształci się w „Center for Advanced Studies on Campus (CAS on Campus)", którego celem jest nawiązanie współpracy z IBM CAS na płaszczyznach naukowo-badawczej i biznesowej. W ramach CAS on Campus powołana zostanie „Grupa R&D" czyli grupa naukowo-badawcza w składzie co najmniej trzech pracowników naukowych i 5 studentów działająca na Uczelni w ramach CAS on Campus oraz „Mieszany Zespół Badawczy" – grupa naukowo-badawcza, w skład której wchodzą członkowie Grupy R&D oraz pracownicy IBM.

W ramach rozszerzonej współpracy PG oraz IBM wspólnie utworzą Centrum Badań Zaawansowanych na Politechnice Gdańskiej (Center for Advanced Studies on Campus „CAS PG"), wyspecjalizowany ośrodek, którego celem będzie rozwijanie współpracy naukowo-badawczej i biznesowej pomiędzy IBM i PG. W szczególności poszerzeniu ulegnie współpraca w obszarach Development, Bussines oraz Services, w ramach której Centrum CAS współpracować będzie z IBM w zakresie rozwijania nowych wersji, ew. rozszerzeń oprogramowania IBM, prowadzenia działającej przy Uczelni firmy zarejestrowanej jako IBM Business Partner, świadczącej odpłatne usługi na rzecz firmy IBM i innych partnerów biznesowych z rynku IT oraz opracowania i zatwierdzenia portfolio innowacyjnych usług, opartych na technologii IBM, oferowanych przez działające przy CAS PG grupy R&D.

8.3. Kierunki rozwoju UCC

Rozpoczęta współpraca z IBM jest w założeniu bazą do dalszego rozwoju centrum w kierunku nastawionym przede wszystkim na świadczenie usług zarządczych na rzecz firm z obszaru ICT.

Planowany kierunek rozwoju to przekształcenie Centrum Kompetencyjnego w formę Integracyjnego Centrum Kompetencji (ICK) obejmującego w sobie również grupę CAS, które stanowić będzie swoistego rodzaju organizację-platformę wspierającą firmy zewnętrzne w procesie planowania, projektowania, wdrażania i utrzymywania rozwiązań informatycznych. Przykładowe obszary wsparcia w zakresie usług świadczonych przez ICK dla strony dostawcy IT jak i klientów wdrażających u siebie rozwiązania technologii informatycznych przedstawia rysunek 8.1.



Rys. 8.1. Zadania ICK (ICC)
Źródło: opracowanie własne

W najprostszym opisie ICK stanowić ma odzwierciedlenie najlepszych praktyk zarządzania IT w zakresie dostarczania zintegrowanych usług i rozwiązań informatycznych.

Głównym celem funkcjonowania ICK będzie wsparcie partnerów biznesowych w upraszczaniu i udoskonalaniu procesów zarządzania przedsiębiorstwem w obszarze rozwiązań IT, w szczególności wsparcie procesu doboru i wdrażania aplikowanych rozwiązań informatycznych dopasowanych do poziomu analizowanych procesów biznesowych oraz stanu informatyzacji. Realizacja tych zadań wymagać będzie dostarczenia przez ICK usług związanych m.in. z analizą istniejących w przedsiębiorstwie rozwiązań IT, oceną przydatności IT dostępnych na rynku, oraz oceną możliwości ich wymiany lub integracji. Jednocześnie ICK stanowić będzie centrum zarządzania realizacją pro-

jektów zarówno dla organizacji dostawcy technologii jak i dla organizacji klienta stanowiąc niejako wyodrębnioną jednostkę organizacyjną mającą za zadanie koordynowanie prac nad projektem oraz ocenę możliwość zrealizowania projektu przed danego dostawcę u danego odbiorcy.

Centrum takie funkcjonować powinno jako samodzielna jednostka, realizująca zlecenia w odrębnym, własnym zespole wykonawczym, lub w zespole mieszanym zbudowanym z przedstawicieli ICK jak i przedstawicieli organizacji, dla której świadczone będą usługi.

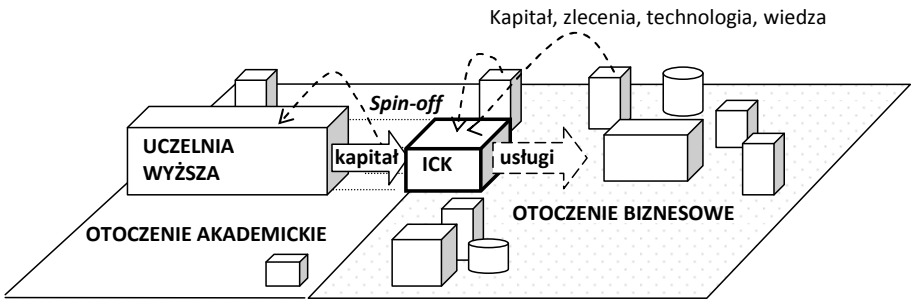
Jednym z koniecznych i istotnych warunków rentownego funkcjonowania ICK jest wyodrębnienie go jako samodzielnego podmiotu gospodarczego funkcjonującego przy uczelni wyższej, przyjmującego charakter firmy *spin-off*[1]. Takie rozwiązanie pozwoli pozyskiwać zewnętrzne źródła finansowania, a jednocześnie otrzymywać określone wsparcie kapitałowe ze strony placówki naukowej.

Posiadając charakter podmiotu gospodarczego ICK przynależeć będzie bezpośrednio do otoczenia biznesowego i jako takie będzie mogło prowadzić działalność zarobkową na warunkach rynkowych (Rysunek 8.2).

Prowadzenie działalności biznesowej umożliwi realizację zleceń pozyskiwanych od przedsiębiorstw, a w efekcie działalności usługowej pozyskiwany będzie kapitał zewnętrzny na funkcjonowanie i rozwój centrum, dostęp do technologii oraz wiedzy praktycznej.

Jednocześnie wsparcie kapitałowe od jednostki macierzystej pozwalające m.in. na doposażenie laboratorium badawczego centrum, rekompensowane będzie dostarczaniem do uczelni aktualnej wiedzy praktycznej, realizacją szkoleń dla studentów przy współpracy z partnerami biznesowymi centrum, certyfikowanie w określonych technologiach i umiejętnościach, podnoszeniem kompetencji pracowników uczelni w obszarze współpracy z biznesem, pozyskiwaniem kapitału z funduszy zewnętrznych na rozwój uczelni, skutecznym poszukiwaniem miejsc praktyk dla studentów.

[1] Firma typu spin-off to przedsiębiorstwo powstałe przez wydzielenie się z jednostki macierzystej, którego celem jest komercjalizacja wiedzy naukowej i technologii.



Rys. 8.2. Otoczenie Integracyjnego Centrum Kompetencji (ICK)
Źródło: opracowanie własne

Realizacja tych założeń wymaga jednak czasu, a przede wszystkim wypracowywania kultury współpracy między uczelnią a biznesem, gdyż kluczowym problemem pojawiającym się nagminnie na styku funkcjonowania tych dwóch ogniw jest całkowicie różna forma administracyjna i prawna funkcjonowania jednostek naukowych i biznesowych oraz inne, często sprzeczne, cele ekonomiczne i biznesowe.

8.4. Podsumowanie

Poruszony w rozdziale temat jest jedynie zarysowaniem idei, wskazaniem potrzeb jak i możliwości powoływania w ramach uczelni Centrów Kompetencyjnych w zakresie rozwijania technologii informatycznych, które w odróżnieniu od dużych z założenia parków technologicznych czy inkubatorów przedsiębiorczości stanowią alternatywę nawiązania skutecznej współpracy ze środowiskiem biznesowym. Poprzez zastosowanie idei małego zespołu roboczego oraz zwinnych metodyk zarządzania projektami podejmowanymi w ramach centrum, zwiększana jest skuteczność i efektywność dotarcia do potencjalnych klientów i partnerów centrum. Warunkiem koniecznym jest umożliwienie wydzielania się takich centrów na zasadzie firm typu spin-off działających na rynku jako niezależne administracyjnie i strukturalnie podmioty gospodarcze.

8.5. Bibliografia

1. Bach-Dąbrowska I., Szczygielski Ł.: *Semantyczny portal wiedzy środowiskiem wsparcia procesów integracji uczelni i biznesu*, Informatyka „4" przyszłości, Tom II, Wydawnictwo Politechniki Warszawskiej, 2011, str. 402-217
2. George O. Mark: *The lean six sigma guide to doing more with less.* John Wiley and Sons., 2010, str. 261.
3. Khalil Tarek M.; Lefebvre L. A.; McSpadden Mason Robert: *Management of technology: the key to prosperity in the third millennium.* Selected papers from the ninth International Conference on Management of Technology. Emerald Group Publishing,13 Listopad 2001, str. 164
4. Marks Eric A.: *Service-oriented architecture governance for the services driven enterprise*. John Wiley & Sons, 2008, str. 271
5. Obrien James A.: *Management Information Systems* (Special Indian Edition ed.) McGraw-Hill Education (India), str. 315
6. National Research Council (U.S.). Committee on Materials Science and Engineering: Forging Stronger Links to Users. National Academies Press, 2000, str.139
7. Wroniak M.: *Gdzie nauka łączy się z biznesem*. Live&Travel, Czerwiec 2012, str.42-43
8. Maciąg M., Janik K.: *IBM otwiera Uniwersyteckie Centrum Kompetencji na Politechnice Gdańskiej*. Archiwum wiadomości IBM Polska. http://www.ibm.com/news/pl/pl/2011/01/12/t500415t09815e47.html

Rozdział 9

Współpraca między przemysłem IT oraz uczelniami - szwedzkie doświadczenia

Udana współpraca uczelni z przemysłem to trochę jak pogodzenie ognia i wody. Uczelnie dogłębnie ale pomału analizują problemy, proponują ich rozwiązania testując przez rok lub dwa i w końcu publikując przez lat kilka. Przemysł chce, aby rozwiązania były gotowe jak najszybciej. Nie jest łatwo pogodzić ze sobą te dwa sprzeczne cele, jednak nasze doświadczenia sugerują, że jest to możliwe.

Uczelnie i firmy mają naturalnie skonfliktowane cele. Uczelnie chcą aby młodzi ludzie studiując przez kolejne pięć lat zgłębiali tajniki dziedziny którą wybrali podążając za programem nauczania. Firmy natomiast chcą pozyskać młodych ludzi najwcześniej jak tylko się da i „wdrożyć" ich w klimat korporacji oraz przekazać jak najwięcej praktycznej wiedzy. Obie strony często z pogardą patrzą na siebie, próbując od czasu do czasu udowodnić sobie, że to „tamci" się na niczym nie znają.

Zdezorientowany młody człowiek często wybiera jedną z dwóch opcji porzucając kompletnie tę drugą. Często słyszy się od ludzi, którzy zaczęli pracować na studiach, że studia to strata czasu i najlepiej zacząć zbierać doświadczenie najwcześniej jak tylko się da, zamiast studiować. Ci którzy pozostają wierni systemowy kształcenia przeżywają często rozczarowanie podczas poszukiwania pracy, a jeszcze częściej spotykają się z brakiem szacunku dla ich studenckich osiągnięć.

Autorzy uważają, że te dwa światy nie tylko powinno się lepiej połączyć, ale też, że rezultaty połączenia mogą być bardzo pozytywne dla obu stron. W tym rozdziale przedstawimy modele współpracy przemysłu i uczelni oraz omówimy przykłady takiej współpracy razem z korzyściami płynącymi

dla obu stron i zagrożeniami, na które należy zwrócić uwagę. Jako ostatnie zagadnienie w tym rozdziale, przedyskutujemy postawy i modele zachowań, które ułatwiają współpracę między przemysłem i uczelniami. W tym rozdziale rozwiniemy naszą poprzednią publikacje o dodatkową analizę modeli naukowych oraz dokładniejsze podsumowanie naszych doświadczeń [3].

Ponieważ badania naukowe przypominają często spacer w nieznane, naukowcy nie mogą obiecać gdzie i kiedy się on zakończy. Ludzie z przemysłu natomiast oczekują dokładnych dat i gwarancji, że spacer zakończy się tam, gdzie oni by chcieli. Pogodzenie tych dwóch celów wymaga odpowiednio przemyślanego i ewaluującego w czasie rzeczywistym procesu, który z dużym prawdopodobieństwem przyniesie zamierzone korzyści.

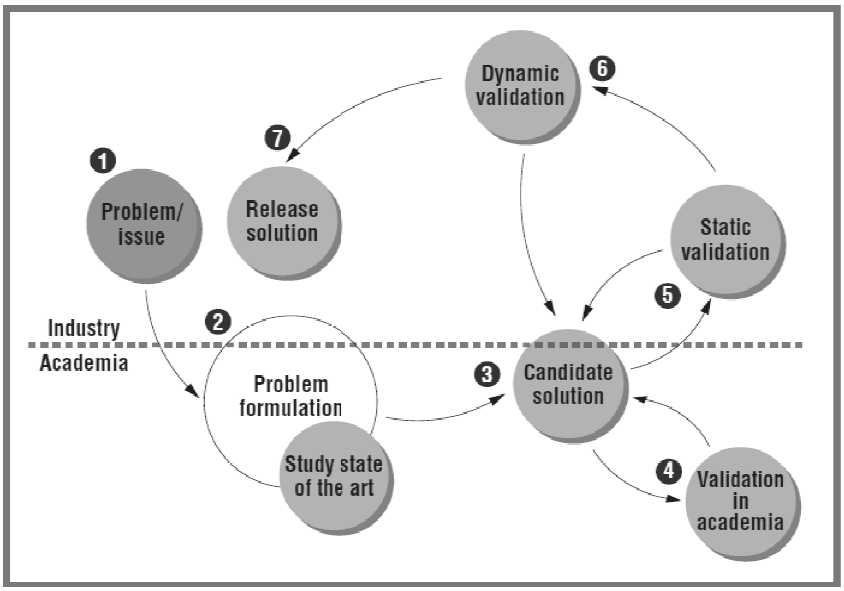
9.1. Modele współpracy przemysłu i uczelni

Modele współpracy pomiędzy uczelnią a przemysłem oraz czynniki umożliwiające udaną współpracę to tematy obecne w literaturze naukowej inżynierii oprogramowania i inżynierii wymagań. Dla przykładu model transferu rezultatów badań przedstawiony przez Gorschek'a i innych [1] podkreśla, że selekcja problemu do rozwiązania powinna odbywać się w oparciu o potrzeby przemysłu, a nie opinie naukowców. Gorschek i inni [1] radzą, aby testowanie rozwiązania odbywało się w dwóch fazach – statycznej i dynamicznej. W fazie statycznej pozyskiwane są opinie na temat propozycji rozwiązania problemu. W fazie dynamicznej natomiast, rozwiązanie poddawane jest prawdziwej próbie działania na pilotażowym projekcie w firmie. Wnioski i doświadczenia z fazy dynamicznej dają nieocenione informacje na temat stosowalności i jakości rozwiązania.

Po zidentyfikowaniu problemu (zobacz 1 na rysunku 9.1.), należy przeprowadzić analizę potencjalnego zakresu poprawy które może przynieść rozwiązanie problemu. W następnym kroku (2) następuje zdefiniowanie celów badań, identyfikacja problemów badawczych oraz wstępna analiza charakterystyki kontekstu badań. Następnie, naukowcy proponują rozwiązanie problemu przy ścisłej współpracy z przemysłem (krok 3 na rysunku 9.1.). W kroku czwartym, naukowcy przeprowadzają weryfikacje proponowanego rozwiązania w warunkach laboratoryjnych. W tym momencie wielu naukowców koń-

czy swoją pracę nad rozwiązaniem problemu i przechodzi do publikacji wyników. Według modelu Gorschek'a i innych, jest to dopiero połowa drogi w kierunku rozwiązania akceptowalnego w przemyśle. W kroku piątym Gorschek i inni sugerują weryfikacje rozwiązania w przemyśle poprzez wywiady oraz seminaria lub ankiety. W fazie statycznej rozwiązanie nie jest używane w kontekście.

Podczas tej weryfikacji mogą pojawić się nowe fakty, a naukowcy otrzymują komentarze i sugestie od przemysłu. Pod ich wpływem naukowcy mogą zostać zmuszeni do dalszego nakładu pracy nad proponowanym rozwiązaniem. W fazie szóstej, naukowcy dokonują weryfikacji dynamicznej, która oznacza wprowadzenie rozwiązania do użytku w przemyśle na pilotażowych projektach przez okres co najmniej kilku miesięcy. Jeżeli proponowane rozwiązanie spełnia oczekiwane kryteria, rozwiązanie jest podsumowywane w ostatnim kroku nazwanym „wypuszczeniem".



Rys. 9.1. Model współpracy miedzy przemysłem a uczelnią (na podstawie Gorschek i inni [1]).

Wohlin i inni [2] zaprezentowali czternaście czynników ułatwiających współprace przemysłu z uczelniami w oparciu o badania i doświadczenie z wieloletniej współpracy. Autorzy uważają, że udana współpraca wymaga do-

kładnego zaplanowania i przeprowadzenia. Pośród przedstawionych czternastu czynników znajdują się między innymi potrzeba posiadania w firmie tzw. "championa", czyli osoby, która będzie nie tylko zainteresowana rozwiązaniem danego problemu, lecz również skutecznie walczyć o pozyskanie środków i poświęcenie należytej uwagi danym badaniom. Do obowiązków takiego „championa" należą: prezentacja wyników badań w firmie, zadbanie o dostęp do innych pracowników oraz do potrzebnych materiałów a także promowanie rezultatów wśród współpracowników oraz analiza i ocena przedstawionych rezultatów. „Champion" musi mieć zwiększoną tolerancje na błędy pracowników oraz obniżone wymagania na dostarczenie idealnych wyników w terminie.

Pożądanymi czynnikami potrzebnymi do sukcesu współpracy przemysłu i uczelni okazują się również: doświadczenie naukowców w pracy z przemysłem, podejście naukowców do tej pracy i ich umiejętności interpersonalne, umiejętności organizacyjne naukowców oraz zaufanie pomiędzy stronami. Jeżeli chodzi o doświadczenie w pracy z przemysłem to jest to rzecz, której naukowiec musi po prostu doświadczyć. Jeżeli chodzi o organizacyjne umiejętności to każdy naukowiec jest powinien być dobrym „managerem" kursów które wykłada. Z jakiegoś dziwnego powodu, ciężko jest nam, naukowcom użyć dokładnie tych samych umiejętności we współpracy z przemysłem. Może głównymi problemami są brak wyraźnego autorytetu i prawa veta we wszystkim (tak jak na własnych wykładach). Na 14 podanych czynników aż siedem odnosi się do naukowców, sześć do firm i jeden (regularne spotkania) jest czynnikiem ogólnym.

9.2. Praktyczne doświadczenia ze współpracy uczelni i przemysłu

9.2.1. Sony Mobile Communications and Sony Mobile, Lund, Szwecja

Doświadczenia przedstawione są w oparciu o 7 lat współpracy, która nadal trwa. W przypadku Sony Mobile rozpoczęcie współpracy było ułatwione przez fakt, że Sony Mobile był fundatorem grantu doktoratu drugiego autora.

Dzięki temu ustanowiony został silny „champion" ; na poziomie CTO (Chief Technology Office), którego zadaniem było dbanie o projekt i o współpracę.

Patrząc z dzisiejszej perspektywy posiadanie takiego „championa" okazało się bardzo ważne i umożliwiło udane badania w zakresie inżynierii wymagań i zarządzaniu produktami. Wszystkie materiały, do których dostęp otrzymał drugi autor (czyli wymagania dla telefonów komórkowych) były materiałami niezwykle cennymi z punktu widzenia Sony Mobile, dlatego też bardzo wcześnie ustalone zostały standardy bezpieczeństwa danych oraz model współpracy zawierający duży kredyt zaufania.

Budowanie pełnego zaufania zajęło około 2 lat. Po kilku pierwszych publikacjach prześwietlonych dokładnie przez managerów firmy odkryli oni, że naukowiec gra w „ich drużynie" i że jego celem jest pomoc firmie i optymalizacja jej procesów, a nie szukanie taniej sensacji, którą można łatwo opublikować.

Bardzo ważnym elementem okazało się zbudowanie obustronnego zrozumienia granicy pomiędzy pracą naukową a doradczą oraz zrozumienia, które tematy nadają się na tematy badawcze, a które są po prostu pracą doradczą. Najważniejsze okazało się zdefiniowanie tematów, które będą ważne zarówno z naukowego punktu widzenia jak i dla przemysłu. Podzielenie się swoimi problemami z innymi (w formie publikacji) przynosi często niespodziewanie pozytywne efekty w postaci zgłaszających się z chęcią pomocy naukowców.

Bardzo ważnym elementem w tej „układance współpracy" była otwartość firmy i dzielenie się danymi, spostrzeżeniami i co najważniejsze czasem pracowników. Dla przykładu zorganizowanie 20 wywiadów z najważniejszymi pracownikami (wliczając w to szefa Android Development) nie stanowiło problemu. Oczywiście, pod warunkiem, że temat którym zajmował się drugi autor był ważny i ciekawy dla firmy. Ważnym elementem była umiejętność „sprzedaży" tematu badawczego managerom w Sony Mobile tak, aby zatwierdzili oni starania i pomogli w realizacji badań.

Z drugiej strony, dzięki zbudowanemu zaufaniu, wielokrotnie udawało się zdobyć dane i pomoc w tematach, które interesowały drugiego autora, ale niekoniecznie już Sony Mobile. W tym przypadku udało się wypracować kompromis i za cenę zajmowania się rzeczami ważnymi dla firmy dodatkowe tematy zostały również zbadane. Balans ten utrzymywany jest do dzisiaj a

każda ze stron stara się zawsze przedyskutować czy nowy pomysł na badania jest czymś interesującym dla firmy.

Kolejną bardzo ważną rzeczą z punktu widzenia naszego doświadczenia była różnica oczekiwań odnośnie szybkości i częstotliwości produkowania rezultatów. Na uczelniach naukowcy przyzwyczajeni są do kilkuletnich (studia doktoranckie) lub co najmniej półrocznych projektów (praca magisterska). Przemysł, jeżeli widzi potencjał w jakimś pomyśle, chce otrzymać wyniki za tydzień lub dwa, zmuszając nas, naukowców do wyboru pomiędzy podążaniem za przemysłem, albo do powrotu do „ciepłego" laboratorium gdzie czas mierzony jest w dekadach.

Dodatkową niewiadomą utrudniającą podjęcie decyzji o współpracy jest to jak dużo pracy naukowiec powinien zainwestować w firmie aby przeprowadzić badania a uniknąć darmowego doradztwa. Kolejnym dylematem naukowca jest wyczucie kiedy delikatnie zasugerować, że czas badań się kończy i pozostała praca to czysta implementacja i tworzenie produktu z pomysłu. Na te dwa pytania nie ma łatwej odpowiedzi, może poza regularnymi spotkaniami z managerami odpowiedzialnymi za kontakty i współpracę oraz umiejętnością znalezienia odpowiedniego balansu.

9.2.2. Siemens Corporate Research, Princeton USA

Współpraca z Siemens Corporate Research (SCR) USA różniła się pod wieloma względami od współpracy z Sony Mobile. Pomimo posiadania „championa", którego głos i opinie były respektowane w SCR, fakt, że centrala Siemens znajduje się w Niemczech powodował, że o pewne kontakty (szczególnie na poziomie dyrektorów) było raczej trudno. Mimo tego, drugiemu autorowi udało się przeprowadzić badania bez większych problemów.

W przypadku Siemensa doświadczenie w pracy z systemami wbudowanymi (ang. Embedded Systems) okazało się bardzo przydatne, gdyż umożliwiało lepsze zrozumienie badanego projektu. Wkład pracowników SCR w projekt badawczy był nieoceniony jednak najważniejszym czynnikiem okazała się umiejętność skupienia się na całości problemu (w tym przypadku problemu śledzenia związków między wymaganiami w bardzo dużych projektach z dziesiątkami tysięcy wymagań).

Dzięki podejściu całościowemu (zamiast wcześniej wspominanego upraszczania problemu do np. 100 wymagań dla jednego z podsystemów) wartość badań była kilkukrotnie większa w oczach firmy. Z drugiej strony, spojrzenie całościowe w naturalny sposób „zagrażało" lub minimalizowało prawdopodobieństwo spektakularnego sukcesu w badaniach, np. o 50% lepszej precyzji niż poprzednie metody. Dlatego też należy pamiętać, że współpracując z przemysłem i obierając całościowy pogląd na analizowany problem jednocyfrowe poprawy są faktycznie wielkim sukcesem. Nakład pracy na pozyskanie tych niewielkich ulepszeń jest bardzo często wielokrotnie większy niż w przypadku nakładu pracy poświęconego na wyniki uzyskiwane w warunkach laboratoryjnych.

9.3. Korzyści z prac dyplomowych sponsorowanych przez przemysł

Widziałem tę współpracę z drugiej strony niż Krzysztof - przez wiele lat pracując w szwedzkich firmach. Regułą było, kiedy tylko pojawiał się problem, braliśmy pod uwagę opcję, żeby rozwiązać go, zlecając napisanie pracy magisterskiej na ten temat. Oczywiście, dotyczyło to spraw o wiele skromniejszych niż opisane powyżej, ale takich jest - zwłaszcza w mniejszych firmach - bardzo dużo. Uczelnie i sami studenci przychylnie odnosili się do prac dyplomowych, do których pomysł przychodził z przemysłu. Dwukrotnie nadzorowałem ze strony firmy powstawanie takich prac, dbając, aby spełniały nasze potrzeby. Raz jako pracownik firmy telekomunikacyjnej Ericsson-Ellemtel, raz z Enea Realtime. Oba przedsięwzięcia zakończyły się sukcesem i dla studentów, i dla firmy.

W Polsce w ciągu dziesięciu lat pracy raz pełniłem taką rolę. Zagadnienie było szersze, miały go dotyczyć aż trzy równoległe prace. Odbyło się kilka spotkań, sfinansowałem zakup książek, a inna współpracująca firma dostęp do informacji. Mimo ciekawej, jak sądziliśmy, opcji wykonania części pracy w Stanach Zjednoczonych, w ciągu kliku miesięcy studenci zmienili zainteresowania i plany, a obie firmy dowiedziały się o tym po fakcie.

Nie obciążałbym jednak winą za to niepowodzenie tylko uczelni. Polskie firmy często obsadzają absolwentów kierunków informatycznych w ro-

lach techników, co sprawia, że przez pryzmat możliwości zatrudnienia i kariery często ważniejsze dla studenta są konkretne umiejętności narzędziowe niż wiedza inżynierska [5]. Późniejsza kariera to już raczej PMI, BPML oraz MBA niż informatyka. Komu chce się w tej sytuacji robić ambitną pracę magisterską?

9.3.1. Algorytm do testowania RAM

W firmie "Ellemtel" na początku lat 90-ych trwały prace, mające na celu stworzenie komputerów telekomunikacyjnych nowej (wówczas) generacji. Tak wtedy, jak i dzisiaj (w branży telekomunikacyjnej, systemów czasu rzeczywistego oraz wbudowanych) aspekty czasowe działania platformy sprzętowej miały kluczowe znaczenie.

Firma potrzebowała statystycznej analizy skuteczności różnych algorytmów testowania pamięci RAM pod kątem znajdowania błędów, w tym błędów parzystości. Było to typowe zadanie, do którego pełnych kompetencji nie musiał posiadać żaden z pracowników działu firmy, a z drugiej strony zadanie jednorazowe – tego typu kompetencje nie wymagały zatrudnienia nowego pracownika na stałe. W tej sytuacji, praca magisterska wykonana przez studenta Wydziału Matematyki Uniwersytetu w Sztokholmie, w zupełności spełniła potrzeby firmy.

9.3.2. Model do opisu rożnych narzędzi testowych

Firma doradcza "Enea" specjalizowała się w latach 1995-2000 w doradztwie oraz różnych usługach z dziedziny testowania oprogramowania. Często zadaniem firmy było doradzanie klientom przy wyborze stosownych narzędzi wspierających testowanie. Niestety, ten obszar był wówczas (i jest nadal [4]) bardzo źle, wręcz chaotycznie opisany. Aby sprawnie pomagać swoim klientom, Enea potrzebowała modelu, pozwalającego sprawnie klasyfikować i porównywać różne narzędzia testowe. Wykonanie takiego modelu było celem nadzorowanej przez Enea pracy dyplomowej na KTH (Politechnice Sztokholmskiej).

Praca spełniła potrzeby firmy, okazała się jednak bardzo pracochłonna, bardziej niż przewidywano, bowiem wymagała nie tylko umiejętności teore-

tycznej analizy (zadanie osoby piszącej pracę), ale także znacznego doświadczenia z narzędziami testowymi, które miała wyłącznie firma. To spowodowało większe niż zakładano wstępnie koszty realizacji takiej pracy.

9.4. Postawy i modele zachowań

Niestety pomimo naszych pozytywnych doświadczeń, nie jesteśmy w stanie obiecać łatwej drogi do sukcesu. Wręcz przeciwnie, nasze doświadczenia przypominają raczej długą i żmudną walkę z przeciwnościami, która okupiona jest o wiele większym wysiłkiem pracy niż można by się spodziewać. W naszej opinii pokora, otwartość i odrobina zrozumienia z każdej ze stron (otwartość ze strony firmy, pokora ze strony uczelni) jak i powstrzymanie się od typowej odpowiedzi „to nie są badania, to jest banalne" są jednymi z najważniejszych czynników sukcesu.

Tym, którzy uważają, że problemy przemysłu są problemami trywialnymi i nie powinny zaśmiecać głów akademickich polecamy poniższą łamigłówkę:

Jeżeli problem z którym próbuje sobie poradzić dana firma jest tak banalny to firma (zatrudniająca na pewno mądrych ludzi którym dobrze płaci) powinna już dawno sobie z nim poradzić.

9.5. Czynniki sukcesu i pokusa upraszczania

Pożądane jest, aby naukowcy mieli doświadczenie w pracy z przemysłem i dobrą motywację do współpracy. Ważne są umiejętności interpersonalne oraz zaufanie pomiędzy stronami.

Jedną z umiejętności, którą naukowcy rozwinęli w wyższym niż inni stopniu jest upraszczanie (modelowanie) realnych zjawisk do problemów, które mieszczą się w zakresie doktoratu lub pracy magisterskiej. Trzeba jednak pamiętać, że rzeczywistość to nie model. Dlatego właśnie naukowcy potrzebują pomocy osób z praktyką w firmach, aby zrozumieć, że uproszczenia i założenia często gubią z oczu sedno problemu.

9.6. Podsumowanie

Istnieje szereg możliwości współpracy przemysłu IT oraz uczelni bardzo korzystnych dla obu stron. Aby je wykorzystać, obie strony muszą nauczyć się obopólnie korzystnych zasad takiej współpracy, stworzyć jej model i wykorzystywać go w przyszłości. Niniejszy rozdział może stanowić punkt wyjścia do stworzenia takiego modelu.

9.7. Bibliografia

1. Gorschek, T., Garre, P., Larsson S.: *A Model for Technology Transfer in Practice*, IEEE Software, vol. 23, Nov. 2006
2. Wohlin, C., Aurum, A., Angelis, L., Phillips, L., Dittrich, Y., Gorschek, T., Grahn, H., Henningsson, K., Kagstrom, S., Low, G., Rovegard, P., Tomaszewski, P., van Toorn, C., Winter, J.: *The Success Factors Powering Industry-Academia Collaboration*, IEEE Software, vol. 29, Mar. 2012
3. B. Bereza i K. Wnuk *Współpraca uczelni z przemysłem*, http://www.computerworld.pl/artykuly/389008/Wspolpraca.uczelni.z.IT.html, 2013
4. Bereza, B.: *Test Tools Taxonomy*, http://victo.eu/ENG/Papers/test_tools_taxonomy.pdf, 2013
5. Bereza, B.: *Jak zostać świetnym informatykiem, Praktyki i staże w IT*, http://wwsi.edu.pl/files/Praktyki_i_staze_w_IT.pdf, 2013

Autorzy i afiliacje

WSTĘP
prof. dr hab. Zdzisław Szyjewski
Instytut Informatyki w Zarządzaniu, Wydział Nauk Ekonomicznych i Zarządzania, Uniwersytet Szczeciński
zszyjew@wneiz.pl
dr hab. Jakub Swacha
Instytut Informatyki w Zarządzaniu, Wydział Nauk Ekonomicznych i Zarządzania, Uniwersytet Szczeciński
jakubs@uoo.univ.szczecin.pl

ROZDZIAŁ 1
dr hab. inż. prof. PG Cezary Orłowski
Wydział Zarządzania i Ekonomii, Politechnika Gdańska
cezary.orlowski@zie.pg.gda.pl
mgr inż. Bartosz Chrabski
Wydział Zarządzania i Ekonomii, Politechnika Gdańska
bartosz.chrabski@pl.ibm.com
mgr inż. Kamil Dowgielewicz
Wydział Elektroniki i Informatyki, Politechnika Koszalińska
kamil.dowgielewicz@gmail.com

ROZDZIAŁ 2
dr Ilona Bluemke
Instytut Informatyki, Wydział Elektroniki i Technik Informacyjnych, Politechnika Warszawska
I.Bluemke@ii.pw.edu.pl
inż. Anna Stepień
Instytut Informatyki, Wydział Elektroniki i Technik Informacyjnych, Politechnika Warszawska

ROZDZIAŁ 3
Karol Kempa
Instytut Informatyki Teoretycznej i Stosowanej, Wydział Inżynierii Mechanicznej i Informatyki, Politechnika Częstochowska
mgr inż. Anna Wawszczak
Instytut Informatyki Teoretycznej i Stosowanej, Wydział Inżynierii Mechanicznej i Informatyki, Politechnika Częstochowska
anna.wawszczak@icis.pcz.pl

ROZDZIAŁ 4
mgr inż. Bartosz Chrabski
Wydział Zarządzania i Ekonomii, Politechnika Gdańska
bartosz.chrabski@pl.ibm.com
Robert Siara
Polsko-Japońska Wyższa Szkoła Technik Komputerowych
IBM Polska, dział oprogramowania
siara.robert@pl.ibm.com

ROZDZIAŁ 5
dr inż. Iwona Dubielewicz
Instytut Informatyki, Wydział Informatyki i Zarządzania, Politechnika Wrocławska
Iwona.Dubielewicz@pwr.wroc.pl
dr inż. Bogumiła Hnatkowska
Instytut Informatyki, Wydział Informatyki i Zarządzania, Politechnika Wrocławska
Bogumila.Hnatkowska@pwr.wroc.pl
prof. dr hab. inż. Zbigniew Huzar
Instytut Informatyki, Wydział Informatyki i Zarządzania, Politechnika Wrocławska
Zbigniew.Huzar@pwr.wroc.pl
dr inż. Lech Tuzinkiewicz
Instytut Informatyki, Wydział Informatyki i Zarządzania, Politechnika Wrocławska
Lech.Tuzinkiewicz@pwr.wroc.pl

ROZDZIAŁ 6
mgr Grzegorz Timoszuk
Instytut informatyki, Wydział Matematyki, Informatyki i Mechaniki, Uniwersytet Warszawski
gtimoszuk@mimuw.edu.pl

ROZDZIAŁ 7
dr inż. Włodzimierz Dąbrowski
Politechnika Warszawska, Instytut Sterowania i Elektroniki Przemysłowej
w.dabrowski@ee.pw.edu.pl
dr Andrzej Stasiak
Wydział Cybernetyki, Wojskowa Akademia Techniczna
astasiak@wat.edu.pl
dr Krzysztof Wnuk
Wydział Informatyki, Uniwersytet w Lund
wnuk@cs.lth.se

ROZDZIAŁ 8
dr inż. Irena Bach-Dąbrowska
Zakład Zarządzania Technologiami Informatycznymi, Wydział Zarządzania i Ekonomii, Politechnika Gdańska
ibach@zie.pg.gda.pl
dr inż. Artur Ziółkowski
Zakład Zarządzania Technologiami Informatycznymi, Wydział Zarządzania i Ekonomii, Politechnika Gdańska
aziolko@zie.pg.gda.pl

ROZDZIAŁ 9
Bogdan Bereza
Victo
bogdan.bereza@victo.eu
dr Krzysztof Wnuk
Wydział Informatyki, Uniwersytet w Lund
wnuk@cs.lth.se