



**POLSKO-JAPOŃSKA WYŻSZA SZKOŁA
TECHNIK KOMPUTEROWYCH**

Wydział Informatyki

Katedra Inżynierii Oprogramowania
Inżynieria Oprogramowania i Baz Danych

Aleksander Pszczółkowski

Nr albumu s3250

Otwarta platforma zarządzania wymaganiami

Praca magisterska napisana
pod kierunkiem:
dr inż. Mariusz Trzaska

Warszawa 18 czerwca 2012

Spis treści

1	Wstęp	5
1.1	Cel pracy	6
1.2	Zarys koncepcji	7
1.3	Struktura pracy	8
2	Istniejące rozwiązania	9
2.1	Analiza rynku systemów wsparcia zbierania wymagań	9
2.1.1	Model SaaS (Software as a Service)	10
2.1.2	Aplikacje desktopowe	18
2.1.3	Platforma IBM. jazz	21
2.2	Problemy z istniejącymi rozwiązaniami	22
3	Podstawy teoretyczne	
	i omówienie koncepcji proponowanego rozwiązania	26
3.1	Stan faktyczny w inżynierii wymagań	26
3.2	Inżynieria wymagań	28
3.3	Motywacja	34
3.4	Opis proponowanego rozwiązania	35
3.5	System Reqmanager — opis funkcjonalności	37
3.5.1	Obsługa wielu projektów	37
3.5.2	Ogólny przegląd stanu projektu	37
3.5.3	Wsparcie gromadzenia wymagań	38
3.5.4	Uniwersalność formatu danych	40
3.5.5	Zarządzanie zmianą	41
3.5.6	Centralne repozytorium	41

3.5.7	Kolaboracja	41
3.5.8	Generator specyfikacji	41
3.5.9	Praktyczne zastosowanie	42
4	Technologie wykorzystane w implementacji	43
4.1	Zarys technologii	43
4.2	Framework Grails	44
4.3	Spring Framework i Hibernate ORM	52
4.4	Pozostałe technologie	53
4.4.1	Postgresql	53
4.4.2	JavaScript, jQuery i biblioteka jsUML2	53
4.4.3	System kontroli wersji git i serwer heroku	58
4.4.4	Środowisko programistyczne (Linux, vim)	59
5	Rozwiązania implementacyjne	60
5.1	Architektura systemu	60
5.2	Funkcjonalności systemu	61
5.2.1	Tworzenie diagramów	64
5.2.2	Współdzielenie diagramów	70
5.2.3	Mapowanie przypadków użycia	70
5.2.4	Przypadki użycia a wymagania	72
5.2.5	Edytor tekstu	74
5.2.6	Generowanie dokumentacji	74
5.3	Przykład zastosowania	76
5.3.1	Ekran główny	76
5.3.2	Nowy projekt	76
5.3.3	Pierwsze wymaganie	77
5.3.4	Przegląd projektu	77
5.3.5	Współdzielenie wymagań	78
5.3.6	Łączenie przypadków użycia z wymaganiem	78
5.3.7	Generowanie specyfikacji	79
5.4	Zalety i wady proponowanego rozwiązania	80

6	Podsumowanie	84
6.1	Plan rozwoju	84
6.2	Zakończenie	85

Streszczenie

Praca dotyczy zagadnienia zbierania i dokumentowania wymagań systemów informatycznych w małych i średnich przedsiębiorstwach oraz narzędzi związanych z tym procesem. Procesy inżynierii wymagań zdają się być zaniedbanym aspektem zarządzania projektami na rynku nowoczesnych aplikacji internetowych a istniejące rozwiązania są drogie i często przystosowane do zastosowań o dużej skali. W niniejszej pracy podjęto próbę wypracowania rozwiązania mającego zastosowanie w niewielkich, dynamicznych projektach. Proponowana koncepcja pozwala skupić się na najważniejszych aspektach formułowania i dokumentowania wymagań oraz umożliwia wprowadzanie i kategoryzowanie zebranej wiedzy. Na podstawie zgromadzonych danych, aplikacja umożliwia automatyczną generację dokumentu specyfikacji wymagań. Technologia HTML5 w połączeniu z językiem JavaScript umożliwiła stworzenie narzędzia wspomagającego graficzne modelowanie przypadków użycia, które może zostać uruchomione w wielu różnych środowiskach.

Rozdział 1

Wstęp

Rozwój technologii internetowych w ostatnich latach umożliwił programistom tworzenie wysoce interaktywnych narzędzi przy jednoczesnym ograniczeniu wymagań systemowych jakie muszą spełnić klienci, chcący skorzystać z wytworzonego oprogramowania. W szczególności, niedawno wprowadzona na rynek, technologia HTML5 pozwala na tworzenie rozwiązań, które były dotychczas zarezerwowane głównie dla technologii Flash, MS Silverlight czy Java-FX, bez konieczności instalacji dodatków do przeglądarek.

Z inżynierią wymagań wykonawcy spotykają się praktycznie na samym początku pracy nad projektem informatycznym. Niezbędne są skuteczne narzędzia opisu i przetwarzania gromadzonej wiedzy oraz komunikowania jej wszystkim członkom zespołu. Dzięki wyżej wspomnianej technologii HTML5, istnieje możliwość usprawnienia procesów fazy analizy przy jednoczesnym zachowaniu wysokiej dostępności aplikacji, wprowadzając nowe narzędzia oparte na metodach graficznych.

Na rynku nowych technologii istnieje wiele aplikacji rozwiązujących rozmaite problemy, od prostych list zakupów, po zaawansowane narzędzia wspierające zarządzanie projektami. Pomimo istnienia wielu aplikacji skupiających się na dziedzinie inżynierii wymagań, bardzo ciężko znaleźć rozwiązanie dostosowane do potrzeb i realiów prowadzenia niewielkich projektów.

W niniejszej pracy zaprezentowano koncepcję aplikacji wspomagającej gromadzenie i zarządzanie wymaganiami. Rozwiązanie stworzono z myślą

o rzeczywistych problemach jakie wiążą się z fazą analizy w projekcie informatycznym. Zaproponowany system ma za zadanie ułatwić zespołom projektowym komunikację i dostęp do wiedzy przy jednoczesnym skupieniu uwagi na najistotniejszych aspektach definiowania i przetwarzania zebranej wiedzy.

1.1 Cel pracy

W małych i średnich przedsiębiorstwach, procesy inżynierii wymagań są najczęściej słabo sformalizowane. Informacje często są przekazywane werbalnie lub trafiają do zespołu z różnych, heterogenicznych źródeł i w wielu, zwykle niekompatybilnych, formatach. Pomimo faktu, iż na rynku nie brakuje oprogramowania wspierającego zarządzanie wymaganiami, istniejące rozwiązania, często nie są przystosowane do realiów prowadzenia projektów w małych, dynamicznych przedsiębiorstwach. Skomplikowane i trudno dostępne systemy często narzucają potrzebę instalacji oprogramowania po stronie klienta a rozwiązania dostępne online, w formule SaaS (Software As A Service), często powielają tylko funkcjonalności potężnych (i drogich) systemów takich jak IBM DOORS czy IBM RequisitePro. Ponadto niezwykle trudno jest znaleźć rozwiązanie darmowe lub posiadające ogólnodostępną wersję demonstracyjną.

Celem niniejszej pracy jest prezentacja koncepcji usprawniającej zbieranie i przetwarzanie artefaktów fazy analizy, dostarczając narzędzi edycji tekstu oraz graficznego modelowania przypadków użycia. W efekcie, użytkownik ma możliwość automatycznego wygenerowania dokumentu SRS (Software Requirement Specification) [16] na podstawie wprowadzonych do systemu danych.

Podejście zaprezentowane w tej pracy, opiera się na założeniach, że nowoczesne oprogramowanie konkurujące w segmencie wspierania procesów powstawania projektów, powinno m.in.: stanowić centralne repozytorium wiedzy o wymaganiach; być łatwo dostępne w jak największej ilości różnych środowisk; umożliwiać łatwą kolaborację oraz być łatwe w obsłudze, prowokując do kreatywności, zamiast stawiać bariery w postaci skomplikowanych formularzy i tabel oraz niejasnych funkcjonalności.

1.2 Zarys koncepcji

Proces analizy wymagań jest trudny, ponieważ często jest procesem niesformalizowanym, rozproszonym i udokumentowanym na wiele sposobów (różne nośniki danych, niekompatybilne oprogramowanie). Ponadto wymagania zwykle pochodzą od wielu interesariuszy: od sponsora projektu, po użytkowników końcowych. Niezależnie od poziomu dojrzałości organizacji i stosowanej metodyki zarządzania projektami, źródła pochodzenia wiedzy na temat systemu pozostają silnie rozproszone i niekompatybilne.

Na potrzeby niniejszej pracy, został stworzony prototyp systemu pozwalający w łatwy sposób dokumentować gromadzoną na etapie analizy wiedzę. W powstałym systemie, głównymi narzędziami definiowania wymagań są pliki tekstowe oraz diagramy przypadków użycia. Projektanci systemu, otrzymują proste w obsłudze narzędzia, pozwalające na skupienie się nad sednem problemu, który budowany system ma za zadanie rozwiązać. W przedstawionej w niniejszej pracy koncepcji narzędzia, użytkownik ma do dyspozycji edytor tekstowy, w którym dokumentuje zidentyfikowane wymagania. Dzięki wykorzystaniu prostego języka znaczników (Markdown [17]), może skupić się na logicznej strukturze opisywanego zagadnienia. Brak narzędzi typu „WYSIWYG” znanych ze standardowych edytorów tekstu jest zabiegiem celowym — ma za zadanie sprawiać, że korzystający z systemu nie są rozpraszeni potrzebą myślenia o graficznej reprezentacji tekstu opisującego problem, przy jednoczesnym zachowaniu czytelnej struktury dokumentu. System załączników i komentarzy, umożliwia iteracyjną pracę nad wymaganiami przy wykorzystaniu zewnętrznych źródeł informacji.

Dzięki graficznemu edytorowi przypadków użycia standardu UML, użytkownik ma możliwość definiowania kluczowych funkcjonalności systemu i łączenia ich z wybranymi wymaganiami. Tworzone diagramy, są dostępne do ponownego wykorzystania w innych miejscach w systemie, dzięki czemu wspierana jest kolaboracja i korzystanie z już istniejących rozwiązań. Zarówno opisy wymagań jak i przypisane im przypadki użycia, są integralną częścią dokumentu specyfikacji systemu informatycznego. Dlatego na każdym etapie fazy analizy istnieje możliwość automatycznego wygenerowania specyfikacji

wymagań na podstawie danych wprowadzonych do aplikacji. Takie podejście sprzyja iteracyjnej konstrukcji oprogramowania oraz umożliwia prezentację prac nad dokumentacją już na wczesnym etapie projektu.

Koncepcja systemu bazuje na silnym przekonaniu, że brak ograniczeń w postaci skomplikowanych i przytłaczających funkcjonalności pozwala skupić się na procesie twórczym w fazie definiowania wymagań, umożliwiając rozwiązywanie rzeczywistych problemów.

1.3 Struktura pracy

Rozdział 2 przedstawia dostępne na rynku narzędzia wspierające zarządzanie wymaganiami. Rozdział 3 jest dokładnym opisem dziedziny problemu oraz koncepcji zaimplementowanego prototypu. W rozdziale 4 zaprezentowano najistotniejsze technologie, jakie wykorzystano podczas pracy nad prototypem oraz krótkie omówienie środowiska programistycznego w jakim powstał przedmiot pracy. Rozdział 5 stanowi opis zaimplementowanego prototypu. W rozdziale 6 zawarto podsumowanie wyników pracy oraz propozycje kierunków dalszego rozwoju.

Rozdział 2

Istniejące rozwiązania

W poprzednim rozdziale nakreślono temat przewodni niniejszej pracy. Poruszono problem dostępności i użyteczności istniejących narzędzi wspomagających zarządzanie wymaganiami. Zaprezentowano także autorską koncepcję rozwiązania, mającego na celu usprawnienie procesu tworzenia specyfikacji wymagań użytkownika.

W tym rozdziale przeprowadzono analizę rynku oprogramowania wspierającego fazę analizy. W drugiej części rozdziału poruszono temat najistotniejszych problemów jakie posiadają istniejące narzędzia.

2.1 Analiza rynku systemów wsparcia zbierania wymagań

Ilość aplikacji związanych z procesami inżynierii wymagań dostępnych na rynku jest ogromna. Zestawienie INCOSE Requirements Management Tools Survey [14] zawiera podzbiór tych narzędzi, jednak opublikowana lista nie jest od dawna aktualizowana, oraz nie wprowadza żadnego rodzaju kategoryzacji.

Dostępne rozwiązania oferowane są w zasadzie w trzech różnych modelach. Najpopularniejszą ostatnio architekturą jest Software As A Service, czyli aplikacja internetowa zainstalowana na serwerach twórcy oprogramowania lub w infrastrukturze zewnętrznej firmy hostingowej.

Również klasyczne aplikacje okienkowe, które należy zainstalować na kom-

puterze klienta nadal cieszą się dużą popularnością. Należy jednak zaznaczyć, iż często są to starsze systemy, nierzadko implementowane jeszcze przed popularyzacją zaawansowanych aplikacji webowych. Niektóre firmy oferują także platformy w architekturze klient-serwer, wymagające istnienia zarówno centralnego serwera — repozytorium w sieci jak i desktopowych aplikacji klienckich. W tej sekcji zostaną przedstawione i porównane wybrane aplikacje z każdej z trzech kategorii.

2.1.1 Model SaaS (Software as a Service)

W ostatnim czasie, wraz z popularyzacją i rozwojem technologii internetowych znacznie wzrosły techniczne możliwości aplikacji dostępnych z poziomu przeglądarki internetowej. Tendencja ta sprzyja powstawaniu licznych rozwiązań, adresujących specyficzne problemy, często w obrębie ściśle określonego segmentu rynku. Jednym z przykładów takiego wąskiego segmentu mogą być np. aplikacje do zarządzania projektami online. Wśród popularnych rozwiązań znajdują się takie produkty jak `basecamp.com`, `unfuddle.com` czy polski `nozbe.com`. Innym przykładem mogą być projekty wspierające proces rekrutacji (`humanway.com`, `recruiterbox.com`, `jobvite.com`). Na uwagę zasługują także systemy rezerwacji hoteli, pensjonatów i niezależnych pokoi i mieszkań jak `airbnb.com`, `booking.com` czy `b&b.com`.

Naturalnie powyższe przykłady dotyczą tylko skrawka możliwych do pokrycia segmentów. W rzeczywistości, bardzo ciężko znaleźć wertykalny rynek, który jest niezagospodarowany serwisami, oferującymi różne podejścia do rozwiązania problemów z danej dziedziny.

Powodem takiego stanu rzeczy jest szeroko pojęta popularyzacja internetu oraz sukcesywne przenoszenie się do sieci firm tworzących oprogramowanie. Dystrybucja software'u w formule SaaS ma wiele zalet w stosunku do klasycznych aplikacji „desktopowych”. W szczególności pominięty jest w tym przypadku proces rozprowadzania aplikacji do klientów za pomocą sieci stacjonarnych sklepów z oprogramowaniem, jak miało to miejsce w ciągu ostatniej dekady. Zaimplementowane rozwiązanie, jest gotowe do użycia z chwilą udostępnienia w internecie. Udostępnienie aplikacji na serwerach twórcy oprogra-

Welcome michael.reiff@vertive.com | Your Account | Help | Report a Bug | Logout

Select Project: Orca User Account Registr...

Requirements | Projects | Reporting | Quality | Search | My Home | Getting Started

Packages | Business Requirements | Software Requirements | Iterations | Actors | Use Cases | Glossary

Business Requirements

Key	Rank	Name	Package	Priority	Iteration	Status
BR21	1	Internal User can add promotions to the site...	Promos and Coupo.	Low	None Selected	Approved
BR20	2	User can browse retail promotions		High	None Selected	Submitted
BR19	27	User can configure a receive only email acco...	Email	High	Release Ca...	In Progress
BR18	6	System will support email newsletters	Email	High	Release Ca...	In Progress
BR17	28	System will send automated emails	Email	High	None Selected	In Progress
BR16	5	System will include multiple secondary page ...	Page Template Fu.	High	Release Ca...	In Progress
BR15	26	Users can forward an offer via email and hav...	Offer Saving & B.	High	Release 1....	Submitted
BR14	4	User can create a My Offers page	Sub-affiliate Pu.	High	Release 1....	Submitted
BR13	29	user can invite out of network friends and i...	Social Networkin.	High	Release 1....	Submitted
BR12	7	User can create a public profile for social ...	Social Networkin.	High	Release 1....	Submitted
BR11	3	User can upload an offer	Sub-affiliate Pu.	High	Release 1....	Submitted
BR10	30	Users can sort search results and browse hie...	Search & Sort	High	Release Ca...	Submitted
BR9	31	Users can search via faceted nav and/or site...	Search & Sort	High	Release Ca...	Submitted
BR8	23	Users will receive automated offers based on...	Offer Recommenda.	High	Release 1....	Submitted
BR7	24	Users can review their view and transaction ...	Transaction Hist.	High	Release Ca...	In Progress

New BR: Add Record

First | Prev | Next | Last

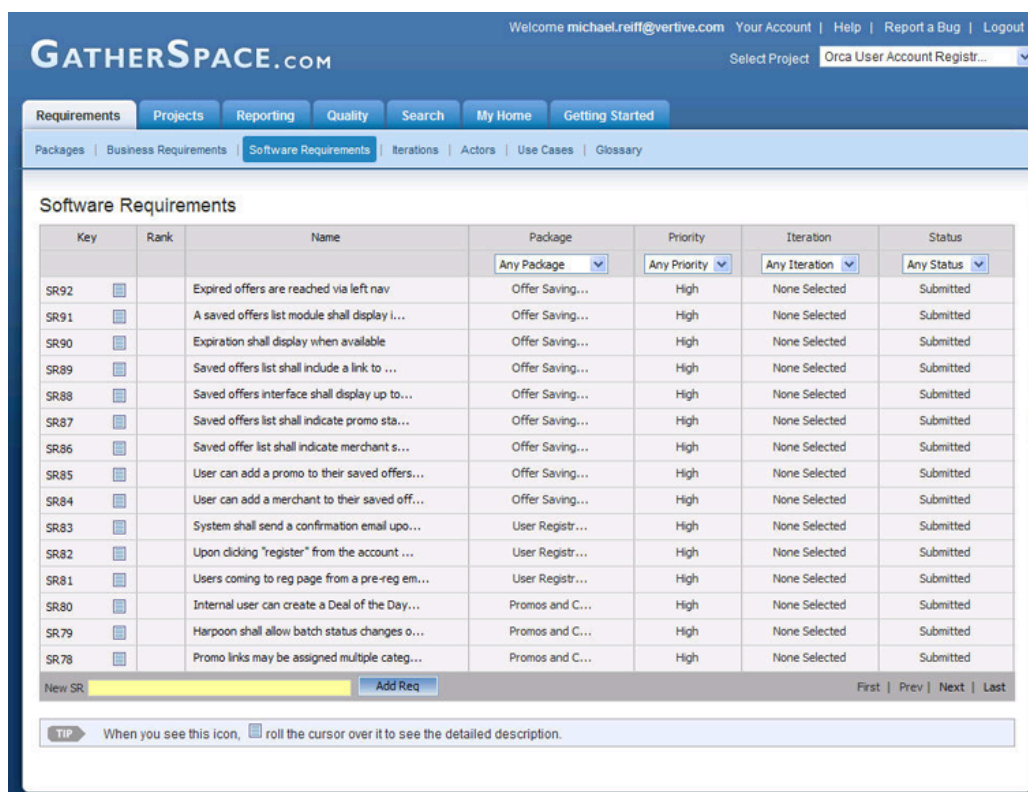
TIP: When you see this icon, roll the cursor over it to see the detailed description.

Rysunek 2.1: Gatherspace.com - business requirements

mowania lub takich dostawców usług jak Amazon Web Services czy Heroku, daje nieograniczone możliwości wprowadzania nowych funkcjonalności, poprawek i skalowania systemu. Z kolei monitorowanie zachowań klientów pozwala niemalże w czasie rzeczywistym odpowiadać na potrzeby użytkowników poprzez modyfikację i wdrażanie nowych funkcjonalności.

W związku z licznymi zaletami modelu SaaS, w internecie powstało wiele aplikacji próbujących zaadresować problemy związane z procesem zbierania i dokumentowania wymagań użytkownika. Do najciekawszych rozwiązań można zaliczyć gatherspace.com, accompa.com, requirementone.com, tracecloud.com czy Jama Contour (<http://www.jamasoftware.com/contour/>)

Na potrzeby tej pracy zostały przedstawione funkcjonalności systemów Gatherspace oraz Tracecloud.



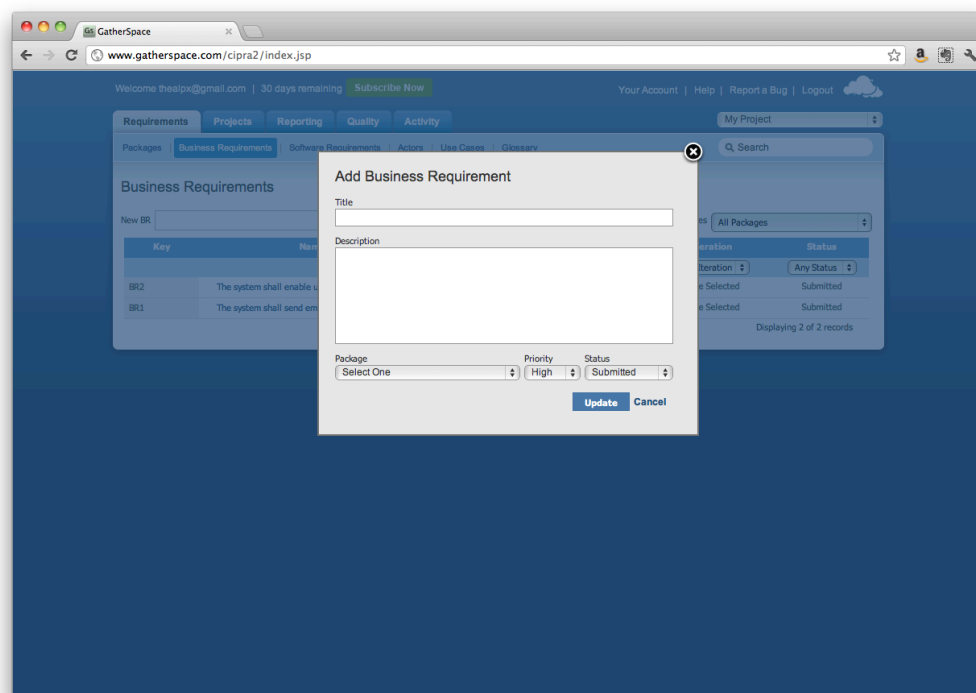
Rysunek 2.2: Gatherspace.com - software requirements

Gatherspace

Gatherspace dzieli wymagania na biznesowe (business requirements) oraz funkcjonalne (software requirements). Dostępne są osobne widoki dla obu typów wymagań (Rys. 2.1, 2.2). Wymagania można katalogować w definiowane przez użytkownika pakiety, podobnie jak pliki w katalogach, w systemach operacyjnych.

Na Rysunku 2.2 przedstawiono widok listy wymagań funkcjonalnych. Dodawanie wymagań przebiega w prosty sposób. Po wpisaniu tytułu wymagania i zatwierdzeniu pola formularza, element zostaje dodany do listy bez przeładowania strony.

Wymaganie może zostać dodane na dwa sposoby — podając jedynie tytuł lub wybierając opcję „Add with detail”. Wówczas pojawia się „popup” umożliwiający dodanie zarówno treści jak i opisu wymagania (patrz Rys.



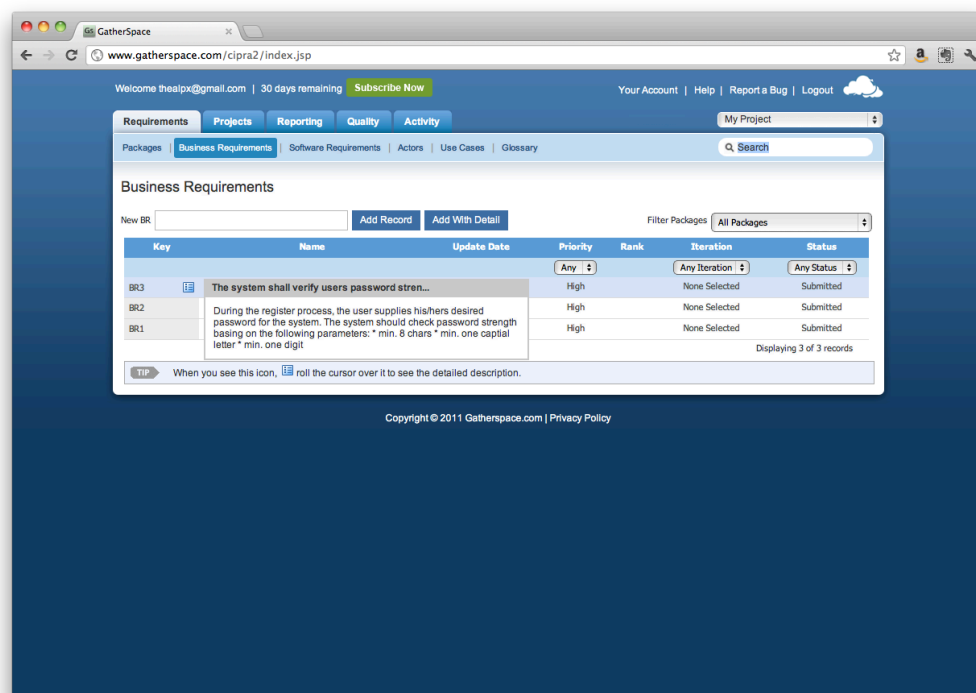
Rysunek 2.3: Gatherspace.com - dodawanie wymagania ze szczegółami

2.3).

Wymaganie posiadające opis, zaopatrzone jest w specjalną ikonkę po lewej stronie. Użytkownik wskazujący w to miejsce myszką może podejrzeć treść wymagania, bez konieczności otwierania jego szczegółów. Jest to dość przydatna funkcjonalność, pozwalająca szybko zapoznać się pobieżnie z wybranym elementem. Rysunek 2.4 zawiera zrzut ekranu prezentujący tę funkcjonalność.

Widok szczegółów wymagania jest dość prosty i przejrzysty (Rys. 2.5). Do dyspozycji użytkownika jest formularz zawierający pola z parametrami, takimi jak m.in.: priorytet, status, poziom złożoności, pakiet do którego należy, przez kogo wymaganie zostało dodane oraz kto lub co jest źródłem wymagania.

Dodawanie przypadków użycia jest wydzielone do osobnego widoku i funkcjonuje jako osobny zestaw formularzy (Rys. 2.6).

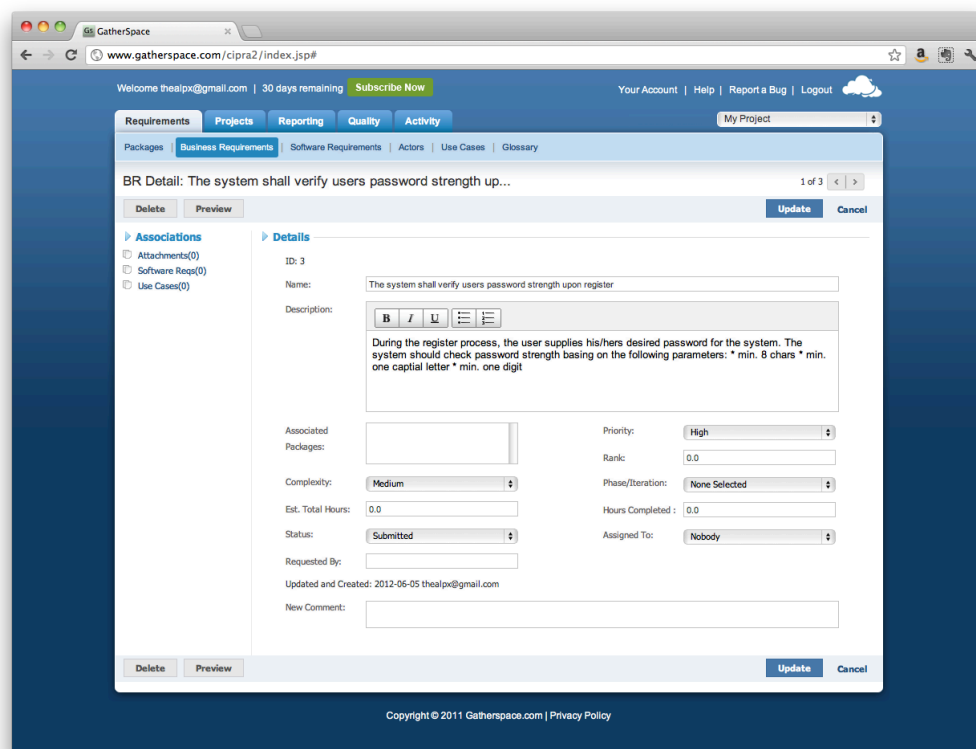


Rysunek 2.4: GatherSpace.com - tooltip ze szczegółami wymagania

Istnieje możliwość łączenia przypadków użycia poprzez wskazanie konkretnego wymagania w osobnym widoku, jednak system nie umożliwia łączenia tych artefaktów z poziomu edytora graficznego.

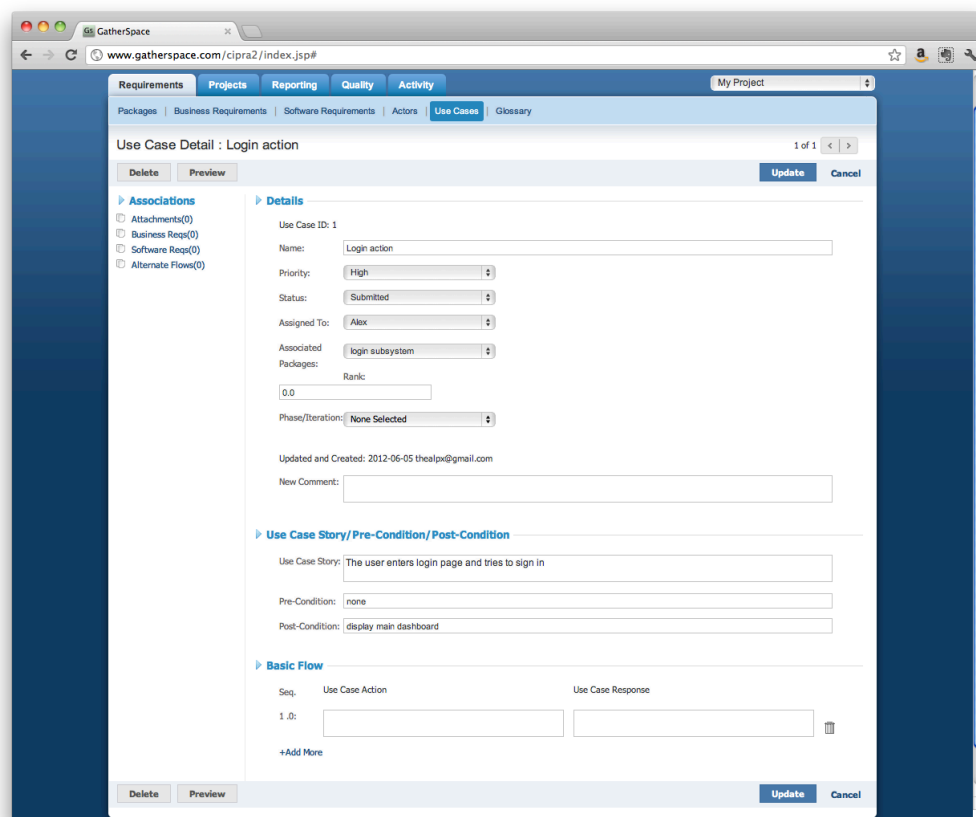
Zarówno przypadki użycia jak i wymagania mogą zostać wyświetlone w formie przyjaznej do druku. Dla przykładu zamieszczono zrzut ekranu obrazujący jeden przypadek użycia przygotowany w tym formacie na Rysunku 2.7

Dostępne w systemie raporty umożliwiają wyświetlenie i wydruk raportów dotyczących wymagań funkcjonalnych i нефункциональных, raport przypadków użycia oraz raport identyfikowalności (traceability report), umożliwiające przeanalizowanie związków pomiędzy wymaganiami a przypadkami użycia (patrz Rysunek 2.8).



Rysunek 2.5: Gatherspace.com - widok szczegółów wymagania

Koszt zakupu licencji na oprogramowanie GatherSpace to minimum 19\$ za miesiąc użytkowania. Co ciekawe, istnieje możliwość zakupu wersji oprogramowania, które zostanie zainstalowane w infrastrukturze klienta. Jest to interesujące rozwiązanie dla klientów korporacyjnych, dla których przechowywanie danych na zewnętrznych serwerach jest wykluczone z powodów bezpieczeństwa (np. banki). Jednak koszt zakupu takiej licencji, jak można przeczytać na stronie GatherSpace.com, to 4900\$ za instalację systemu. Dodatkowo firma żąda 295\$ miesięcznie za obsługę systemu. Biorąc pod uwagę zakres funkcjonalności oraz użyteczność systemu, są to ogromne kwoty, przewyższające, w ocenie autora, wartość dodaną wniesioną do organizacji wraz z wdrożeniem systemu GatherSpace.



Rysunek 2.6: Gatherspace.com - widok szczegółów wymagania

Tracecloud

Jak widać na Rys. 2.9 Tracecloud jest znacznie gorzej przygotowany od strony interfejsu użytkownika. Mimo tego jest bardziej skomplikowanym systemem, udostępniającym więcej funkcjonalności niż Gatherspace.

Główny przegląd projektu zawiera tabelę z wyświetlonymi metrykami projektu. Można z niej odczytać ilość zaimplementowanych wymagań, ilościowe wyniki testów, przegląd zgłoszonych błędów i wartości liczbowe wymagań pogrupowane w poszczególne statusy. Po dokładnym przeanalizowaniu tabeli, można odczytać interesujące informacje na temat projektu, jednak użyteczność tego sposobu prezentacji danych jest bardzo niska.

Oprócz ilościowego przeglądu stanu projektu, użytkownik ma do dyspozycji menu i odnośniki do najważniejszych modułów systemu, takich jak wy-

Printable Use Case Detail View

www.gatherspace.com/cipra2/d_usecase/usecase_view.jsp

USE CASE REPORT

Use Case # 1 : Login action

Use Case Description:

The user enters login page and tries to sign in

Use Case Properties:

Actor(s):	None Selected
PreCondition:	none
PostCondition:	display main dashboard
Priority:	High
Status:	Submitted
Project:	My Project
Footprint:	Created by thealpx@gmail.com on 2012-06-05

Basic Flow:

Num.	User Action	System Response
1		

Alternate Flows:

Associated Business Requirements:

BR 2	The system shall enable user to log in
------	--

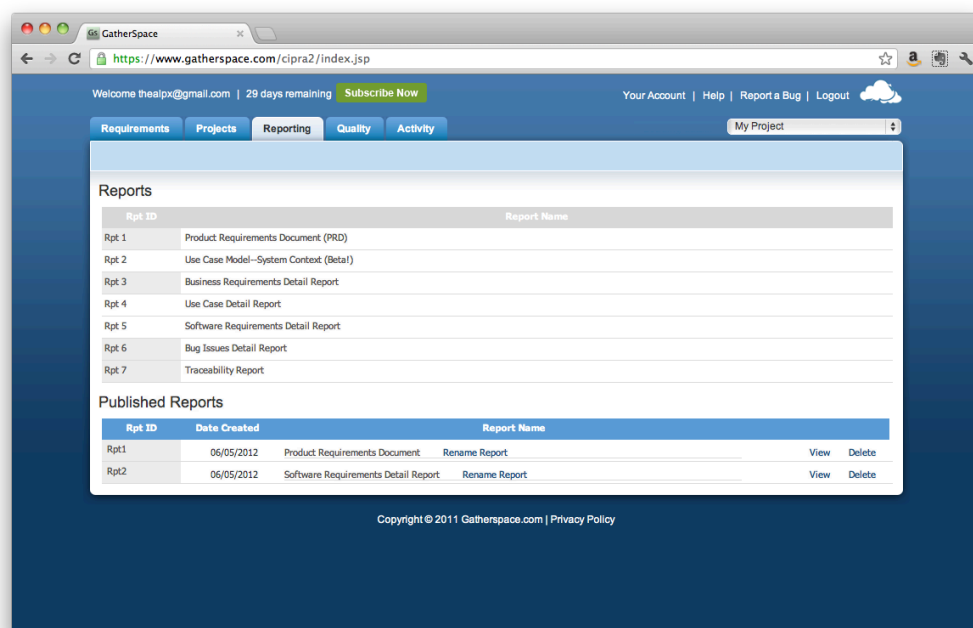
Rysunek 2.7: Gatherspace.com - wymaganie przygotowane do druku

magania biznesowe, funkcjonalne, scenariusze testowe i błędy.

Przegląd wymagań opiera się o wykres ze statystykami oraz prostą listę (Rys. 2.11), z kolei szczegóły wymagania zawierają informację o jego nazwie, priorytecie, właścicielu oraz opis. Dodatkowo twórcy umieścili moduł śledzenia umożliwiający połączenie wymagań biznesowych z funkcjonalnymi (Rys. 2.12).

Oprócz licznych raportów wyświetlanych w obrębie aplikacji, nie znaleziono funkcjonalności generowania dokumentu specyfikacji wymagań.

Tracecloud dostępny jest jako wersja „trial” przez 6 miesięcy. Koszt wdrożenia waha się od 350\$ do 1000\$ za miesiąc użytkowania systemu w zależności od maksymalnej ilości obsługiwanych wymagań. Najtańsza licencja przewi-



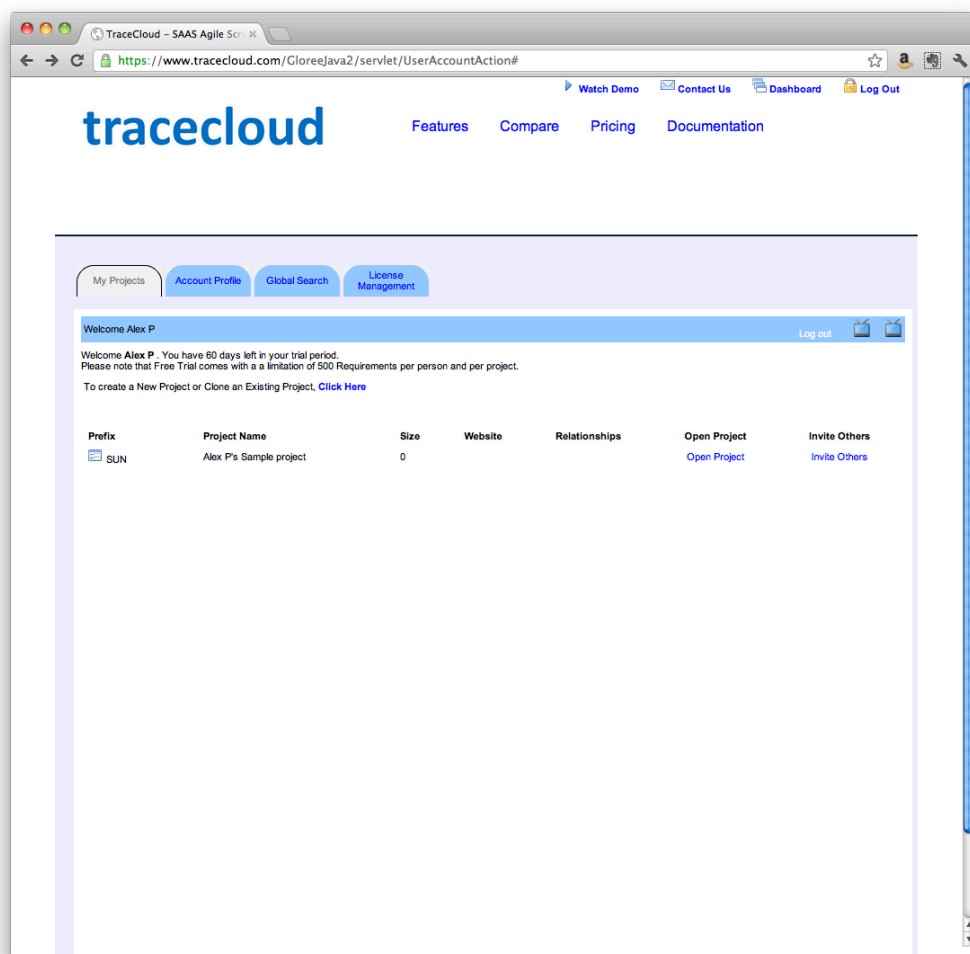
Rysunek 2.8: Gatherspace.com - dostępne raporty

duże obsługę do 5000 wymagań, podczas gdy wersja najdroższa — do 10tyś.

2.1.2 Aplikacje desktopowe

Model SaaS, mimo wszystkich swoich zalet, posiada również wady. Głównym powodem kontrowersji jest konieczność powierzenia danych biznesowych firmie udostępniającej narzędzie na swoich serwerach. Dla dużych korporacji, pilnie strzegących swoich tajemnic handlowych, takie rozwiązanie, może być nie do zaakceptowania ze względów bezpieczeństwa. Mimo potencjalnej możliwości uniknięcia kosztów budowy i utrzymania infrastruktury sprzętowo-software'owej ryzyko związane z brakiem kontroli nad powierzonymi danymi jest często zbyt wielkie.

W klasycznych aplikacjach okienkowych, odpowiedzialność w zakresie zabezpieczenia danych spoczywa na samej korporacji i jej pracowniku. Dzięki temu, nadal istnieje zapotrzebowanie na oprogramowanie instalowane na lokalnym dysku użytkownika. Przykładami takich aplikacji są m.in. DOORS

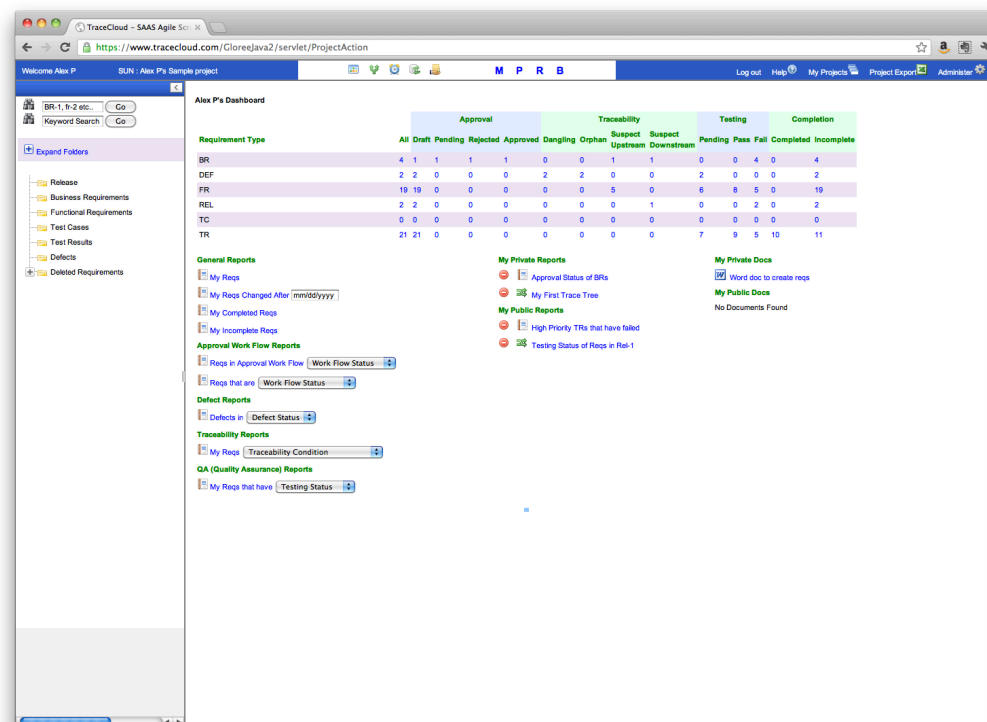


Rysunek 2.9: Tracecloud - widok listy projektów

czy Rational Requirements Composer.

Rational Requirements Composer jest aplikacją dostarczaną przez firmę IBM. Dostępna jest wersja zbudowana na bazie platformy *eclipse* jak również klient webowy, uruchamiany w przeglądarce internetowej na platformie IBM Jazz. Jest to potężne narzędzie wspierające kolaborację, śledzenie wymagań, audyty, raportowanie, graficzne modelowanie procesów biznesowych oraz zaawansowaną edycję dokumentów tekstowych.

Definiowanie wymagań w Rational Requirements Composer jest możliwe na wiele sposobów, m.in.: szablonowanie interfejsów, edycja tekstu, wykorzystanie story board'ów czy użycie wbudowanych w system kontrolek i goto-



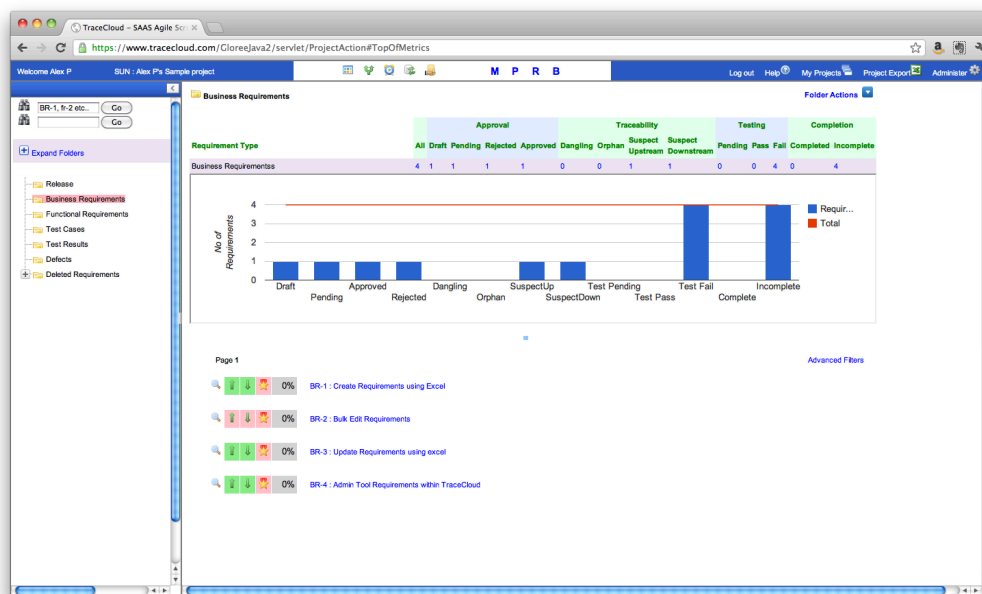
Rysunek 2.10: Tracecloud - ekran główny

wych szablonów.

Przykładowy ekran definiowania wymagania umieszczono na Rysunku 2.13.

Platforma umożliwia również modelowanie procesów biznesowych w języku BPML (Business Process Modeling Language) (Rysunek 2.14).

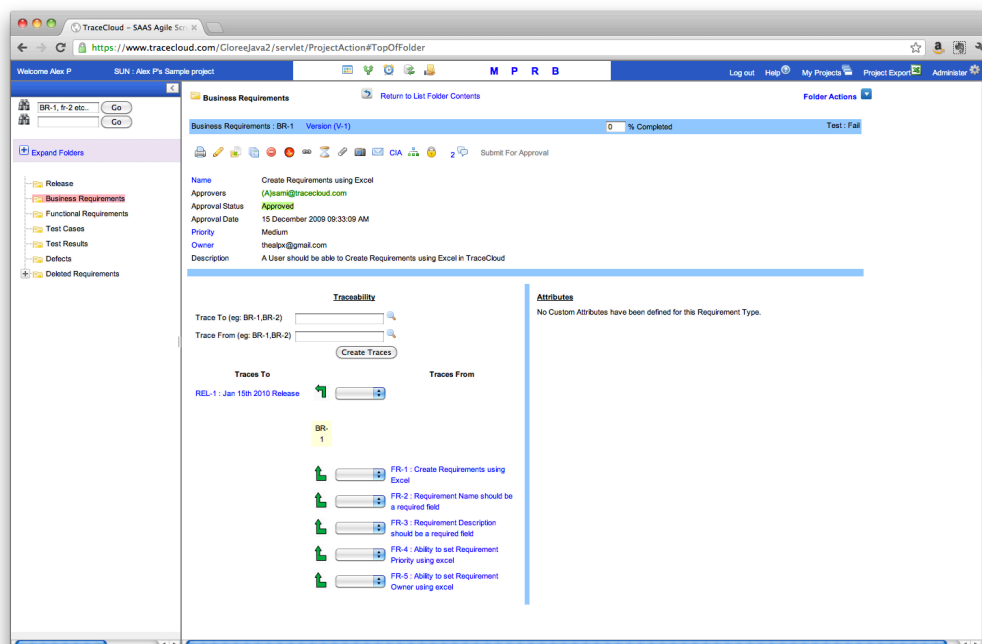
Zaawansowane narzędzia wspierające kolaborację, umożliwiają komunikację drogą elektroniczną, rewidowanie artefaktów, aktualizację statusów zadań i wiele innych funkcjonalności zaprezentowanych na Rysunku 2.15. Z kolei audyty i historia pozwalają na wspólne tworzenie i modyfikację dokumentów tekstowych na zasadzie śledzenia zmian podobnego do tego, znanego z pakietu MS Office (Rysunek 2.16).



Rysunek 2.11: Tracecloud - przegląd wymagań

2.1.3 Platforma IBM. jazz

Platforma IBM Jazz zasługuje na osobną sekcję, ponieważ jest zintegrowanym, kompleksowym zestawem narzędzi i aplikacji dla przedsiębiorstw zajmujących się tworzeniem oprogramowania. Centrum zarządzania platformą Jazz jest serwer Jazz Team Server stanowiący repozytorium danych i ośrodek dowodzenia. Platforma składa się zarówno udostępnionych na serwerze usług sieciowych oraz aplikacji instalowanych lokalnie, łączących się ze zdalnym serwerem (tzw. rich-client applications). Ogólna architektura platformy została zaprezentowana na Rysunku 2.17. Interesującym rozwiązaniem w tym przypadku jest fakt, iż jazz. net stanowi swojego rodzaju szynę integracyjną, pozwalającą na korzystanie z istniejącego oprogramowania zainstalowanego na licznych komputerach klienckich. Komunikacja z aplikacjami klienckimi przebiega na bazie protokołu HTTP i webservice'ów RESTowych.

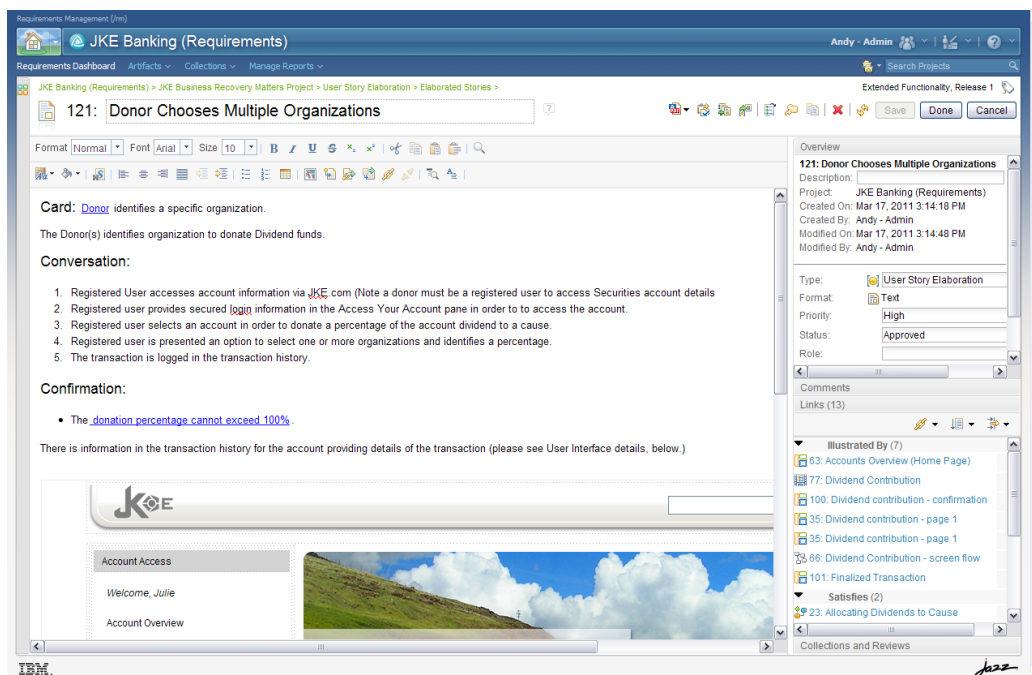


Rysunek 2.12: Tracecloud - szczegóły wymagania

2.2 Problemy z istniejącymi rozwiązaniami

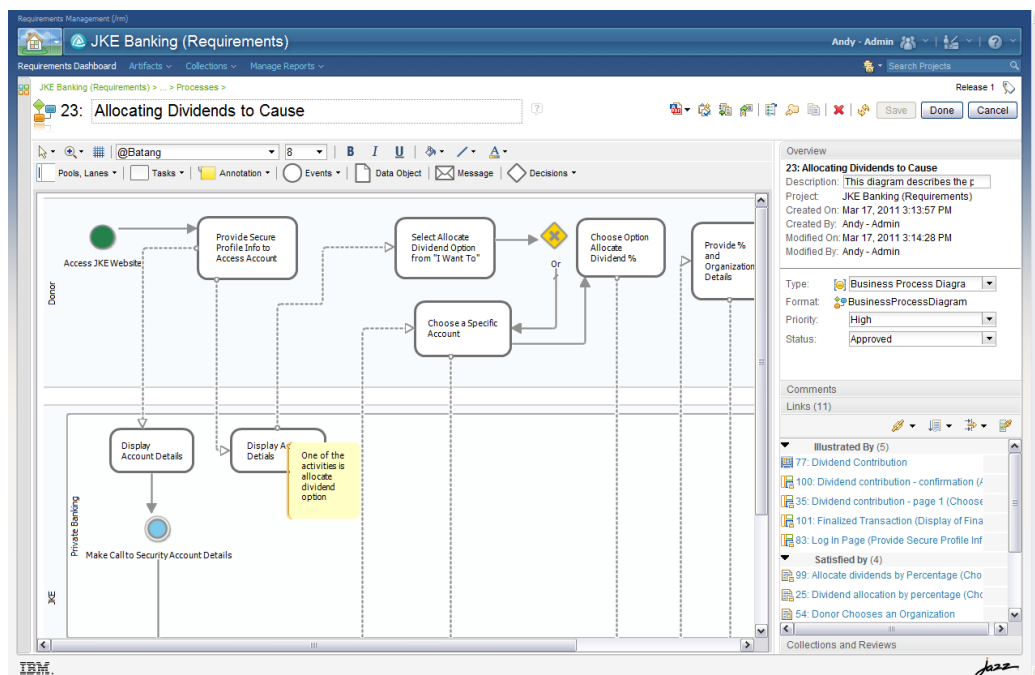
Analizując istniejące rozwiązania, nie można oprzeć się wrażeniu, że wszystkie przystosowane są do dużych, korporacyjnych projektów. Większość dostępnych aplikacji online stara się powielać funkcjonalności znane z produktów firmy IBM. W konsekwencji trudnym zadaniem jest znalezienie innowacyjnych elementów wśród dostępnych narzędzi. Pozytywnie wyróżniają się szczególnie ciężkie aplikacje desktopowe IBM wraz z platformą Jazz. net, jednak są to rozwiązania przystosowane do bardzo dużych projektów. Małe i średnie przedsiębiorstwa potrzebują prostszych, „lżejszych” i bardziej dostępnych narzędzi.

Proces definiowania wymagań w niewielkich projektach jest w dużej mierze procesem twórczym. Często pojawienie się nowej potrzeby lub zmiany jest inicjowane z kilku heterogenicznych źródeł. W niewielkich projektach, gdzie często jedna osoba, lub mały zespół odpowiedzialny jest za przygotowanie specyfikacji, wymagania przybierają postać wiadomości email, rozmów tele-



Rysunek 2.13: Rational Requisite PRO - definiowanie wymagania. Źródło: jazz.net

fonicznych, notatek ze spotkań i formalnych dokumentów. Zadaniem osoby lub zespołu odpowiedzialnego za budowę dokumentu specyfikacji, jest prze-filtrowanie częściowych informacji oraz ekstrakcja i udokumentowanie niezbędnej wiedzy. Niestety żadne narzędzie (przynajmniej na razie) nie zastąpi skutecznie pracy człowieka w tym procesie, dlatego, należy pracować nad wyposażeniem analityków i projektantów systemów informatycznych w odpowiednie narzędzia, ułatwiające zadanie definiowania wymagań.



Rysunek 2.14: Rational Requisite PRO - definiowanie wymagania. Źródło: jazz.net.

Overall Review: Draft → In progress 42% completed → Pause Review → Reviewed → Finalized

Due: Mar 24, 2011

Instructions to reviewers:

Please review the additional glossary terms that we are using in this project

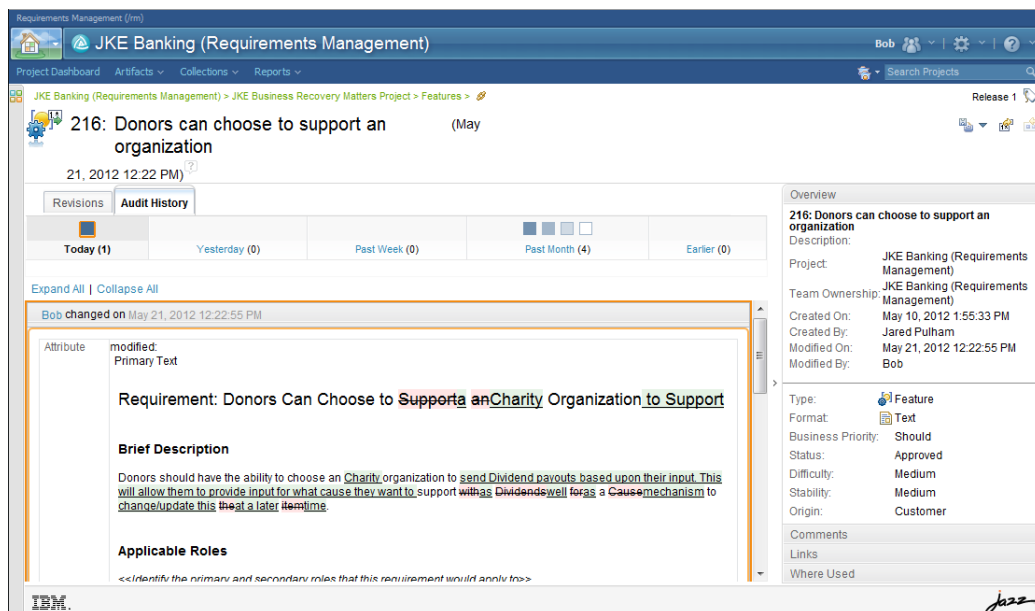
Participant	Type of Participant	Review results
☐ Al	Approver	Done - 4 Approved
☐ Bob	Approver	
☐ Curtis	Optional Reviewer	Done - 4 Abstained
☐ Tanuj	Optional Reviewer	25% - 1 Reviewed
☐ Ursula	Approver	25% - 1 Approved

Created from: Project Glossary

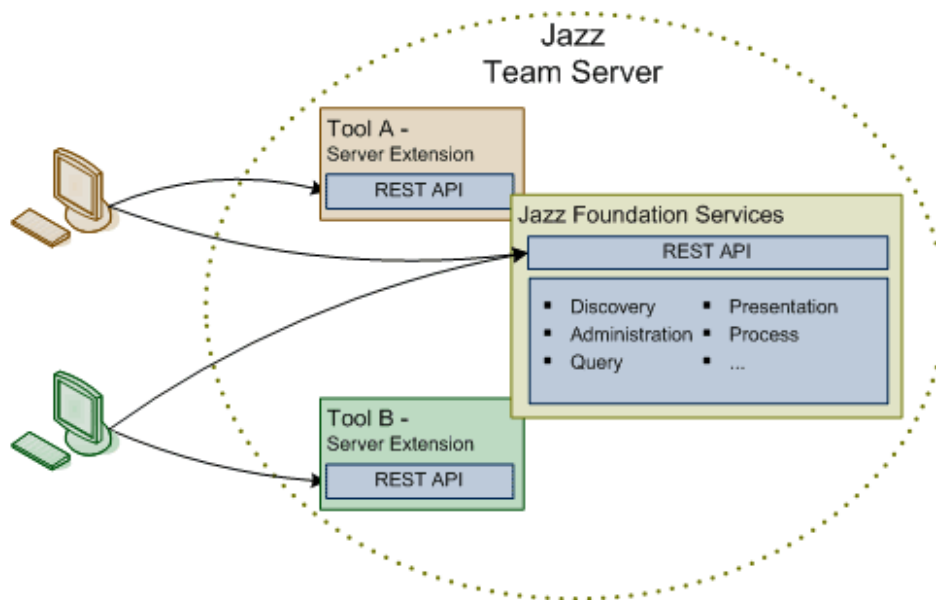
Artifact versions are live

ID	Artifact	Version	Status
75	Organization	Mar 17, 2011 3:14 PM	1 Approved, 1 Abstained
113	Donor	Mar 17, 2011 3:14 PM	1 Approved, 1 Reviewed, 1 Abstained
17	Dividends for a Cause	Mar 17, 2011 3:14 PM	1 Approved, 1 Abstained
64	Business Recovery Matters	Mar 17, 2011 3:14 PM	2 Approved, 1 Reviewed, 1 Abstained

Rysunek 2.15: Rational Requisite PRO - kolaboracja. Źródło: jazz.net.



Rysunek 2.16: Rational Requisite PRO - historia zmian. Źródło: jazz.net.



Rysunek 2.17: Platforma jazz.net - Jazz Team Server. Źródło: jazz.net

Rozdział 3

Podstawy teoretyczne i omówienie koncepcji proponowanego rozwiązania

W poprzednim rozdziale opisano przykładowe narzędzia funkcjonujące na rynku oprogramowania do zarządzania wymaganiami. Opisano ich główne wady i zalety oraz przedstawiono najważniejsze problemy związane ze stosowaniem ich w praktyce.

W tym rozdziale, opisano procesy inżynierii wymagań od strony teoretycznej. Została również zwrócona uwaga na niezbędne elementy nowoczesnego systemu wspierającego pracę z wymaganiami w małych i średnich przedsiębiorstwach. Na końcu, przedstawiono koncepcję nowego narzędzia, będącego przedmiotem tej pracy.

3.1 Stan faktyczny w inżynierii wymagań

Inżynieria wymagań stanowi fundament wiedzy pozwalającej na realizację wysokiej jakości projektów informatycznych w założonym budżecie i czasie.

Istnieje obszerna literatura podejmująca tematy stricte związane z inżynierią wymagań (Gause and Weinberg 1989; Jackson 1995; Kovitz 1999; Robertson and Robertson 1999a; Sommerville and Sawyer 1997), jednak

praktyczne doświadczenia oraz liczne badania wskazują na istotne problemy z wprowadzaniem teorii w życie [5, 20, 26].

Znany raport opublikowany przez Standish Group International [26] prezentuje dramatyczną rzeczywistość, gdzie tylko niewielka część projektów kończy się częściowym lub całkowitym sukcesem. Wyniki raportu wskazują na najważniejsze powody niepowodzeń projektów jak brak włączenia użytkowników w proces budowy oprogramowania, niekompletne i nieudokumentowane wymagania, wiecznie zmieniający się zakres projektów czy brak wsparcia kadry zarządzającej.

W badaniu Nikula et al. (2000) przeprowadzonym wśród małych i średnich przedsiębiorstw w Finlandii, autorzy wskazują na alarmujący brak wiedzy dotyczącej wdrażania procesów inżynierii wymagań. Jednocześnie, wnioski z badania wskazują na konieczność zajęcia się tymi problemami na poziomie korporacyjnym, poprzez wdrażanie, optymalizację i automatyzację procesów IW w małych i średnich przedsiębiorstwach [20].

Z kolei badanie Quispe et al. (2010), również dotyczące małych i średnich przedsiębiorstw, zwraca szczególną uwagę na brak lub nienależytą komunikację z klientem, adresowanie nieodpowiednich problemów klienta, wadliwe specyfikacje oraz stosowanie praktyk inżynierii wymagań „ad hoc” [22].

Powyższe badania zajmują się głównie czynnikami niepowodzeń projektów. W badaniu Fernández et al. (2012) przeanalizowano 12 projektów zakończonych sukcesem. Wnioski z tego badania sugerują, że wiele ze zidentyfikowanych czynników wpływających na niepowodzenia projektów można rozwiązać zanim staną się realnym zagrożeniem. Ponadto zauważono, że klienci często mają znaczny wpływ na parametry związane z niepowodzeniem projektu.

Analiza wyników powyższych badań w kontekście tej pracy sugeruje przede wszystkim konieczność zwrócenia szczególnej uwagi na poziom dojrzałości organizacji w aspekcie inżynierii wymagań. Nie wszystkie projekty kończą się porażką, a doświadczenia pokazują, że im bardziej dojrzałe procesy wdrażane są w organizacji i budowana jest odpowiednia „kultura” firmy, tym większe są szanse na realizację projektu zwieńczoną sukcesem.

Podsumowując, żaden system informatyczny ani narzędzie nie rozwiąże wszystkich problemów związanych z inżynierią oprogramowania. To na orga-

nizacji ciąży odpowiedzialność usprawniania procesów fazy analizy, edukacji pracowników i zasięgania wiedzy ekspertów. Sukcesy w branży oprogramowania można osiągnąć tylko poprzez ciągłą naukę, konsekwentną realizację procesów i ich usprawnianie na poziomie korporacyjnym.

3.2 Inżynieria wymagań

Inżynieria wymagań (IW) jest dziedziną, na którą składają się wszelkie działania związane z odkrywaniem (elicitation), analizą (analysis), weryfikacją (validation) oraz zarządzaniem wymaganiami [25]. Dotyczy ona każdego systemu informatycznego, niezależnie od jego rozmiarów i jest jednocześnie jednym z najistotniejszych procesów w cyklu życia projektu.

IW dostarcza metodologii i narzędzi, służących do precyzyjnego określenia celu i zakresu projektu, poprzez identyfikację interesariuszy i ich potrzeb oraz udokumentowanie wyników w formie umożliwiającej ich analizę, komunikację oraz finalnie — implementację [2].

Zarys historyczny

Kryzys oprogramowania na przełomie lat '60 i '70 XX wieku był katalizatorem do podjęcia prób usystematyzowania młodej wówczas dziedziny tworzenia systemów informatycznych. Był to okres popularyzacji sprzętu komputerowego, powszechnego, dramatycznie rosnącego zapotrzebowania na programistów oraz nagłośnionych, wielkich porażek oprogramowania, jak katastrofa rakiety kosmicznej Mariner 1 [7]. Niedobór wykwalifikowanych programistów, zmusił korporacje do intensywnych poszukiwań nowych rozwiązań. Literatura branżowa szybko przepełniła się głosami nawołującymi do poprawy metod zarządzania w projektach informatycznych. Pojawiły się takie tytuły jak "Controlling Computer Programming"; "New Power for Management"; "Managing the Programming Effort"; "The Management of Computer Programming Efforts". W roku 1968, w drodze ustaleń na konferencji NATO Software Engineering (tu po raz pierwszy w branży zaczęto mówić o „kryzysie oprogramowania”), "zarządzanie oprogramowaniem" stało się fundamentem i jednym

z głównych dyskursów w dziedzinie inżynierii oprogramowania [11].

Pionierzy inżynierii oprogramowania podejmowali próby zaczerpnięcia wiedzy z zakresu inżynierii systemów i wykorzystania jej w projektach informatycznych. Ten kierunek rozwoju opierał się na paradygmacie klasycznej inżynierii, zakładającym budowę systemu zgodną z określonym procesem: od identyfikacji problemu, opracowaniu specyfikacji, konstrukcji systemu i jego utrzymania. Takie sekwencyjne podejście do tworzenia systemów informatycznych było fundamentem powstania pierwszych procesów wytwórczych oprogramowania.

Klasycznym podejściem był przedstawiony w pracy Winstona Royce’a (1970), model kaskadowy (sekwencyjny model liniowy)¹. W swojej pracy, Royce proponując rozwiązanie sekwencyjne (w którym żaden kolejny etap nie mógł zostać rozpoczęty przed zakończeniem etapu poprzedniego) zastrzegł, że „wierzy w zaproponowaną koncepcję, jednak jej implementacja jest ryzykowna i naraża projekty na niepowodzenie” [27]. Zatem już na samym początku swojego istnienia, model kaskadowy wraz z analogią inżynierii oprogramowania do inżynierii klasycznej, był uważany w branży za nieprzystosowany do dziedziny problemu. Mimo tego, model kaskadowy, z powodzeniem, znalazł zastosowanie w dużych projektach m.in. rządowych, wojskowych i kosmicznych, gdzie zarówno klient jak i wykonawcy bardzo dobrze rozumieli wymagania systemu [19]. W kolejnych latach pojawiały się modyfikacje modelu kaskadowego, a rola podejścia iteracyjnego przybierała na znaczeniu. W roku 1986 Barry Boehm opisał model spiralny, będący połączeniem ustrukturalizowanego procesu kaskadowego z rozwojem przyrostowym, opartym na prototypowaniu [4]. Kolejnym kamieniem milowym w dziedzinie procesów wytwórczych oprogramowania było wydanie książki Kent’a Beck’a „Extreme Programming Explained” [3] w roku 1999 oraz publikacja „Agile Manifesto” w roku 2001, gdzie po raz pierwszy, formalnie zaproponowano termin „zwinnego programowania” [15].

¹Royce podał w swoim modelu wariant liniowego procesu, gdzie kolejne fazy były zwrotnie sprzężone z fazami poprzednimi [27], jednak w większości organizacji stosujących model kaskadowy, jest to ściśle sekwencyjny proces liniowy [21].

Podobieństwa w procesach wytwórczych

Pomimo znaczących różnic w dostępnych i udokumentowanych metodykach projektowych, dla wszystkich istnieje część wspólna, w postaci nieuniknionej fazy analizy na etapie rozpoczęcia projektu. Nie ma możliwości rozpoczęcia modelowania ani implementacji systemu, bez wcześniejszej wiedzy w zakresie jego przeznaczenia i oczekiwanych funkcji. Zarówno w metodykach zwinnych, jak i w tradycyjnych procesach wytwórczych, w początkowej fazie projektu, kluczową rolę gra przygotowanie systemu od strony wymagań. Zatem inżynieria wymagań dotyczy każdego procesu wytwórczego.

Procesy w inżynierii wymagań

Jak podaje Sommerville [25], do dziedziny inżynierii wymagań możemy zaliczyć następujące pod-procesy:

- studium wykonalności
- gromadzenie i analiza wymagań
- weryfikacja wymagań
- zarządzanie wymaganiami

Studium wykonalności (feasibility study)

Wg. Sommerville'a (2006), przy dużych projektach, w początkowym stadium powinno zostać przeprowadzone studium wykonalności. Podstawowym celem jego przeprowadzania jest udzielenie odpowiedzi na pytanie, czy należy kontynuować projekt. W trakcie tego etapu zwiększa się wiedza o dziedzinie problemu, a koncepcja systemu jest weryfikowana pod kątem celowości zastosowania w organizacji, wszelkich ograniczeń systemowych, a także sprzętowych. W modelu Rational Unified Process, studium wykonalności powinno zostać włączone do fazy rozpoczęcia projektu (inception phase) [18].

W niektórych przypadkach (szczególnie w dużych projektach krytycznych) studium wykonalności może zostać rozbudowane do pełnej analizy

oceny ryzyka. Wówczas powinny zostać uwzględnione aspekty nietechniczne i administracyjne, np.: czy projekt ma przydzielony wystarczający budżet, jakie aspekty polityczne należy wziąć pod uwagę w trakcie definiowania wymagań, czy projekt może być wykonany w przyjętych ramach czasowych.

Gromadzenie i analiza wymagań

Gromadzenie wymagań jest procesem, na który składają się działania zmierzające do zgłębienia celu i motywacji przyświecających budowie analizowanego systemu. Etap ten wiąże się z identyfikacją wszystkich wymagań, jakie musi spełnić system, aby osiągnąć sukces.

Proces gromadzenia wymagań w języku angielskim nosi nazwę „requirements elicitation”. „Elicitation” w dosłownym tłumaczeniu, oznacza „wywołanie”, „wydobycie”, „ujawnienie”. W szczególności termin „wydobycie” lepiej oddaje naturę tych działań, niż popularne w języku polskim „gromadzenie” czy „definiowanie” wymagań. Innymi słowy jest to proces komunikacji analityków z użytkownikami w celu zdobycia jak największej ilości informacji o budowanym systemie.

Głównymi technikami związanymi z pozyskiwaniem wymagań są: identyfikacja interesariuszy, gromadzenie faktów i informacji przy pomocy wywiadów, kwestionariuszy, list kontrolnych, warsztatów kreatywnych, burzy mózgów, map myśli (mindmapping), modelowania i prototypowania. Znaczna część wymagań stanowi wiedzę zdobytą w procesie przeprowadzania wywiadów z interesariuszami projektu. W związku z powyższym, można wprowadzić uniwersalne uogólnienie, stwierdzając, że głównym narzędziem wykorzystywanym w procesie gromadzenia wymagań jest szeroko rozumiana komunikacja. Wywiady przeprowadzane z interesariuszami mogą mieć formalny lub nieformalny przebieg, a stosowane w tym procesie narzędzia w dużej mierze zależą od poziomu skomplikowania projektu, wypracowanych metod, dostępnych narzędzi i preferencji samych analityków.

Wymagania zwykle pochodzą z wielu heterogenicznych źródeł. Najwyższy priorytet zawsze powinien przysługiwać wymaganiom pochodzącym od klienta i użytkowników systemu. Jednak wpływ na specyfikację systemu mogą

mieć również źródła zewnętrzne, takie jak niezależni eksperci z dziedzin obejmowanych przez projekt, uwarunkowania prawne i ekonomiczne, czy ogólnie przyjęte standardy i procedury związane ze szczególnymi aspektami tworzonego systemu.

Dokumentem stanowiącym rezultat pracy nad gromadzeniem wymagań jest specyfikacja wymagań systemu. Standardem określającym proces tworzenia i strukturę tego dokumentu jest IEEE Guide for Developing System Requirements Specifications [10]. Wiele organizacji definiuje również własne standardy i procesy, lepiej odpowiadające specyfice realizowanych projektów.

Do procesu gromadzenia wymagań i narzędzi z nim związanych, autor powraca przy okazji opisu koncepcji proponowanego rozwiązania, w następnym rozdziale.

Weryfikacja wymagań

Celem procesu weryfikacji wymagań jest dowiedzenie, że koncepcja budowanego systemu odpowiada rzeczywistym potrzebom użytkowników. Jest to kluczowy proces inżynierii wymagań, ponieważ poprawa zidentyfikowanych na tym etapie problemów jest o rząd wielkości mniej kosztowna, niż późniejsze zmiany systemowe. Zdecydowanie łatwiej jest bowiem poprawiać projekt architektury oprogramowania niż wprowadzać modyfikacje koncepcyjne w istniejącym już systemie.

Weryfikacja wymagań powinna być przeprowadzana na podstawie dokumentu specyfikacji wymagań systemu. Działania podejmowane w trakcie weryfikacji wymagań powinny uwzględniać przede wszystkim zasadność, bezkonfliktowość, kompletność, implementowalność i weryfikowalność wszystkich wymagań. Zasadność pozwala odpowiedzieć na pytanie, czy dane wymaganie rozwiązuje realny problem w systemie, potwierdzając tym samym celowość swojego istnienia. Ponieważ w trakcie gromadzenia wymagań przeprowadzane są wywiady z wieloma interesariuszami projektu, mającymi różne, często sprzeczne oczekiwania, konieczna jest weryfikacja spójności specyfikacji pod względem konfliktów. Zapewnienie bezkonfliktowości polega na takim sformułowaniu wymagań w specyfikacji, aby żadne nie były ze sobą w sprzecz-

ności. Kompletność jest weryfikacją dokumentu wymagań z perspektywy realizacji wszystkich pożądaných funkcji. Wszystkie zdefiniowane wymagania muszą być możliwe do zrealizowania w określonym czasie, przy użyciu określonego sprzętu i oprogramowania. W trakcie badania implementowalności, analityk skupia się na odpowiedzi na pytanie czy dane wymaganie jest w pełni wykonalne w rzeczywistych warunkach. Z kolei weryfikowalność ma na celu zapewnienie narzędzi, umożliwiających sprawdzenie i demonstrację poprawności działania zrealizowanego wymagania za pomocą metod empirycznych.

Istnieje wiele narzędzi i metod, jakie można wykorzystać w procesie weryfikacji wymagań. Wyniki badania Boehem et al „Prototyping vs. Specifying” [6] dowodzą, że jedną ze skuteczniejszych metod prewencji przed definiowaniem błędnych wymagań jest metoda prototypowania. Badanie to polegało na podziale grupy studentów inżynierii oprogramowania na dwa zespoły. Celem obu zespołów była implementacja systemu o określonym zakresie. Jedna grupa korzystała z metody prototypowania. Ich konkurenci, natomiast, korzystali z podejścia zorientowanego na specyfikację. W rezultacie, grupa stosująca metodę prototypowania osiągnęła lepsze wyniki, w szczególności, w zakresie użyteczności interfejsu użytkownika i prostoty obsługi dostarczonego systemu. W przeciwieństwie do podejścia opartego na samej specyfikacji, prototypowanie umożliwia wizualizację wymagań systemu, uniezależniając interpretację treści wymagania od wyobraźni odbiorcy.

Jednak, jak każda metoda, prototypowanie nie jest wolne od wad. Istnieje ryzyko, że prototyp, kładąc zbyt duży nacisk na pewne szczegóły, sprawi, że ogólna koncepcja systemu przestanie być wyraźnie dostrzegalna. Ponadto, wyzwaniem w przypadku prototypowania jest odpowiednia komunikacja z klientem. Musi on dokładnie rozumieć specyfikę i cel tworzenia prototypu. Istnieje bowiem pokusa, aby pod naciskami klienta wykorzystać prototyp jako finalny produkt. W związku z powyższym, wśród wszystkich członków zespołu, niezbędna jest pełna świadomość faktu, iż prototyp, po spełnieniu swojej funkcji zostanie odstawiony, a doświadczenia z jego budowy posłużą jako informacje wejściowe do implementacji finalnego rozwiązania. Powstało wiele prac dotyczących tej metody, jednak szczegółowe aspekty tego podejścia wykraczają poza zakres tej pracy. Osoby zainteresowane metodami

prototypowania znajdują kilka ciekawych pozycji w bibliografii [1, 8].

Zarządzanie wymaganiami

Specyfika większości projektów informatycznych, wiąże się z tym, że raz udokumentowane wymagania rzadko pozostają aktualne do momentu zakończenia prac. Wymagania są przedmiotem ciągłych zmian i niezbędne są narzędzia, których zadaniem będzie zarządzanie tymi zmianami. Proces zarządzania wymaganiami należy rozpocząć w momencie powstania pierwszej wersji specyfikacji wymagań i kontynuować aż do zakończenia projektu.

3.3 Motywacja

Mając na uwadze teoretyczne tło dziedziny jaką jest inżynieria wymagań oraz praktyczne problemy z jakimi borykają się dzisiaj małe i średnie przedsiębiorstwa w zakresie zarządzania wymaganiami, powstała koncepcja rozwiązania, adresującego najpilniejsze problemy w rzeczywistych projektach małej i średniej skali.

Jak wykazano w rozdziale 2, pomimo rozwoju technologii oraz świadomości firm, bardzo duża część małych i średnich przedsiębiorstw nadal ignoruje zagadnienia z zakresu inżynierii wymagań. Dojrzałość tych organizacji pod kątem inżynierii oprogramowania jest bardzo niska. Niewielką część harmonogramów projektów przeznacza się na fazę analizy, a narzędzia stosowane w trakcie tego procesu najczęściej ograniczają się do pakietów biurowych, klientów poczty elektronicznej i odręcznych notatek. Mikro-firmy osiągające na tym polu sukcesy najczęściej charakteryzują się posiadaniem bardzo małych zespołów ekspertów z ogromnym doświadczeniem w branży, posiadającymi techniki wypracowane metodami prób i błędów, nad którymi pracowali przez wiele lat w swojej karierze. Jednak zdecydowana większość firm nadal eksperymentuje z autorskimi podejściami, często integrując różne istniejące narzędzia.

Część winy za taki stan rzeczy leży po stronie twórców oprogramowania. Istnieje niewiele systemów umożliwiających skuteczne zarządzanie wymaga-

niami w środowisku dynamicznych, zorientowanych na rynek internetowy, projektów, jakie najczęściej realizują firmy z sektora MŚP. Kolejną ważną obserwacją jest fakt, iż bardzo często mamy do czynienia z projektami, których wymagania wymyślane są przez samych twórców, przez co proces dokumentowania wymagań i tworzenia specyfikacji przebiega ad hoc lub nie istnieje w ogóle. Dodatkowo, brak umiejętności zarządzania zmianą, powoduje, że projekty oddawane są po terminie, z przekroczonym budżetem, często nie dostarczając klientowi pożądanego efektu. Jest to problem całej branży — zarówno kadry zarządzającej, analityków, programistów, klientów i użytkowników.

Małe i średnie przedsiębiorstwa są motorem napędowym dzisiejszej gospodarki opartej w znacznym stopniu o techniki informacyjne. Obowiązkiem inżynierii oprogramowania jest stałe kwestionowanie istniejących rozwiązań i katalizowanie zmian w kierunku usprawniania procesów związanych z konstrukcją systemów informatycznych. Jednak inżynieria wymagań zdaje się być niedocenianym i nadal słabo rozumianym zagadnieniem związanym z wytwarzaniem oprogramowania. Jest to jednak kluczowa, przyszłościowa dziedzina — w miarę postępu technologicznego, zapotrzebowanie na oprogramowanie nadal będzie wzrastało, a wraz z nim poziom skomplikowania systemów wymagających jasno zdefiniowanych wymagań.

3.4 Opis proponowanego rozwiązania

Proces pozyskiwania i przetwarzania wymagań powinien być przede wszystkim łatwy w implementacji dla organizacji oraz dający dużą dozę elastyczności. W rozdziale 'Problemy z istniejącymi rozwiązaniami' opisano stan sztuki i zaadresowano niedociągnięcia dzisiejszych narzędzi.

Motywy przewodnim proponowanego rozwiązania jest przystępność i prostota obsługi oraz wniesienie rzeczywistej wartości dodanej poprzez dostarczenie funkcji, mających realne zastosowanie w praktyce.

W trakcie prac nad koncepcją narzędzia przyjęto następujące założenia:

- Silna koncentracja na praktycznym zastosowaniu systemu wsparcia za-

rzządzania wymaganiami — system musi rozwiązywać rzeczywiste problemy zespołów projektowych w zakresie gromadzenia i przetwarzania wymagań. Wdrożenie systemu musi wiązać się z wniesieniem wartości dodanej do procesów inżynierii wymagań w organizacji;

- Obsługa wielu projektów w obrębie aplikacji — system musi umożliwiać obsługę i zarządzanie wieloma projektami w obrębie tej samej aplikacji;
- Ogólny przegląd stanu projektu — najistotniejsze informacje z perspektywy wymagań systemu powinny być dostępne na jednej stronie podsumowującej projekt;
- Wsparcie gromadzenia wymagań — system powinien wspierać pozyskiwanie, definiowanie, katalogowanie (grupowanie) nieprzetworzonych wymagań oraz nadawanie im odpowiednich priorytetów przy pomocy intuicyjnych narzędzi;
- Uniwersalność formatu przechowywanych danych — system powinien umożliwiać łatwe przetwarzanie treści wymagań przez zewnętrzne systemy;
- Dostarczenie rozwiązań zarządzania zmianą — aplikacja powinna dostarczać wsparcia zarządzania wymaganiami na późniejszych etapach projektu;
- Stworzenie centralnego repozytorium wiedzy na temat wymagań — system powinien stanowić główne źródło wiedzy i komunikacji w zakresie wymagań;
- Wsparcie kolaboracji — system powinien umożliwiać współpracę nad wymaganiami poprzez obsługę wielu użytkowników i dostarczenie im modułu komentarzy;
- Automatyzacja tworzenia specyfikacji — należy zaimplementować algorytm generujący dokument specyfikacji wymagań na podstawie informacji dostępnych w systemie;

W efekcie powstał prototyp systemu, otwierający drogę do dyskusji i rozwoju w kierunku poprawienia procesów inżynierii wymagań w małych i średnich przedsiębiorstwach, skupiający się na praktycznych rozwiązaniach.

3.5 System Reqmanager — opis funkcjonalności

Poniżej opisano funkcjonalności systemu i przedstawiono przyjęte rozwiązania, które posłużyły do realizacji założeń systemu.

3.5.1 Obsługa wielu projektów

System umożliwia tworzenie wielu projektów. Formularz dodawania projektu składa się z pól: nazwy, krótkiego opisu oraz głównej notatki projektu, będącej jego szczegółowym opisem w formie tzw. briefu. Ostatnią niezbędną informacją, którą należy podać jest planowana data zakończenia prac nad projektem. W założeniu opis i notatka projektu ulegają zmianom w miarę upływu czasu, dlatego na późniejszych etapach projektu istnieje możliwość bardzo łatwej edycji zmian treści tych atrybutów.

3.5.2 Ogólny przegląd stanu projektu

Ekranem głównym w obrębie wybranego projektu jest przegląd jego stanu z perspektywy wymagań. Użytkownik ma możliwość natychmiastowej edycji głównej notatki opisującej założenia całego projektu oraz jego krótki opis. Ponadto, dostępna jest lista wymagań zawierająca podstawowe informacje takie jak kod, opis, priorytet oraz ilość przypadków użycia. Wszystkie te elementy umieszczone na jednej stronie, umożliwiają użytkownikowi ogólną orientację w zakresie i celach projektu pod kątem wymagań.

3.5.3 Wsparcie gromadzenia wymagań

Praktyczne narzędzie do zastosowania w procesie analizy musi wspierać najwcześniejszy etap — powstawania wymagań. Narzędzie musi umożliwiać prostą w obsłudze opcję dodawania nowych informacji w postaci krótkich notek, dłuższych artykułów lub opisów całych procesów. Jednocześnie nie może obarczać użytkownika obowiązkiem nadawania identyfikatorów oraz innych meta-informacji. Powinno również ograniczać potrzebę formatowania treści. Celem użytkownika, jest spisanie jak najszybciej wszystkich obserwacji i zachowanie ich w formie możliwej do późniejszego przetworzenia.

Proponowane rozwiązanie wspiera ten proces dzięki zintegrowanemu edytorowi tekstu, edytorowi diagramów przypadków użycia oraz systemowi załączników.

Edytor tekstowy

Formatowanie tekstu w języku znaczników Markdown [17] umożliwia nadanie dokumentowi logicznej struktury wykorzystując zwykły tekst (plain text) zaopatrzony w odpowiednie oznaczenia. Dzięki temu użytkownik otrzymuje czytelny dokument, przy jednoczesnym ograniczeniu formy graficznej, która jest w tym kontekście drugoplanowa. Przykład składni dokumentu Markdown zaprezentowano na Listingu ??.

```
1      # Brief projektu
3
5      Ten dokument zawiera tzw "brief" projektu - wstępny
        opis
7
9      ## System w musi uwzględnić
        następujące funkcjonalności:
11
        * logowanie użytkownika
```

```

13      * dodawanie i zarządzanie projektami
      * dodawanie i zarządzanie wymaganiami
      * _tworzenie diagramow przypadkow uzycia_

```

Listing 3.1: przykład składni języka Markdown

Po konwersji przez Markdown do języka HTML, otrzymamy wynik przedstawiony na Listingu 3.2:

```

2      <h1>Brief projektu</h1>

4      <p>Ten dokument zawiera tzw "brief" projektu -
      wstepny opis systemu, jaki mamy zamiar zbudowac.</
      p>

6      <h2>System musi uwzgledniac nastepujace
      funkcjonalnosci:</h2>

8

10     <ul>
12         <li>logowanie uzytkownika</li>
13         <li>dodawanie i zarzadzanie projektami</li>
14         <li>dodawanie i zarzadzanie wymaganiami</li>
15         <li><em>tworzenie diagramow przypadkow uzycia</em>
16         </li>
17     </ul>

```

Listing 3.2: wynik konwersji powyższego kodu do HTML

Dzięki temu istnieje możliwość tworzenia spójnych dokumentów o logicznej strukturze, przy niewielkim nakładzie pracy.

Edytor diagramów UML

Edytor diagramów UML umożliwia wzbogacenie treści wymagania o graficzną reprezentację powiązanych przypadków użycia. Moduł ten wspiera współdzielenie diagramów pomiędzy wieloma wymaganiami, umożliwiając

oznaczanie odpowiednich elementów jako odpowiedzialnych za realizację aktualnie opracowywanego wymagania.

Szczególnie istotnym elementem systemu jest generowanie obiektów aplikacji na podstawie danych odczytywanych bezpośrednio ze struktury diagramu. W efekcie, po utworzeniu elementu na diagramie i odpowiednim powiązaniu go z danym wymaganiem, powstanie reprezentacja tego elementu w postaci obiektu klasy Groovy, który następnie zostanie zapisany do bazy danych dzięki mapowaniu obiektowo — relacyjnemu będącemu integralną częścią frameworka Grails. Jednocześnie zachowana zostaje odpowiednia relacja między utworzonym obiektem klasy UseCase, a wymaganiem. Mapowanie elementów utworzonych na diagramie przebiega dzięki możliwości serializacji diagramu do formatu XML. Szczegółowo ten mechanizm opisano w rozdziale dotyczącym rozwiązań implementacyjnych (rozdz. 5).

W przyszłości, istnieje możliwość rozszerzenia funkcjonalności edytora o kolejne rodzaje diagramów UML. Ponadto, w trakcie ewentualnej rozbudowy systemu, wygodnym usprawnieniem byłoby automatyczne przypisywanie nowo utworzonych obiektów do aktualnie edytowanego wymagania.

System załączników

System załączników dodatkowo wzbogaca możliwości ekspresji — do wymagania użytkownik może dodać dokumenty lub treści multimedialne powiązane z danym wymaganiem, mające na celu lepsze zrozumienie istoty problemu.

3.5.4 Uniwersalność formatu danych

Narzędzie do zarządzania wymaganiami powinno ułatwiać późniejsze procesowanie wprowadzonych do niego danych. Proponowany system realizuje tę potrzebę przechowując wszystkie wymagania oraz powiązane diagramy w bazie danych. Treść opisu wymagań jest przechowywana w postaci zwykłego tekstu, zaopatrzonego w znaczniki Markdown. Dzięki temu, treść wymagań jest znacznie łatwiejsza w przetwarzaniu, niż dokumenty tworzone przez klasyczne edytory tekstu.

Struktura diagramów powiązanych z wymaganiami jest przechowywana

w bazie danych w formacie xml. Takie podejście również w znacznym stopniu ułatwia późniejsze przetwarzanie informacji zawartych na diagramach.

System aktywnie wykorzystuje powyższe właściwości podczas automatycznego generowania specyfikacji wymagań.

3.5.5 Zarządzanie zmianą

Zarządzanie zmianą jest istotne z punktu widzenia ewolucji wymagań i stopniowego uszczegóławiania dokumentacji systemu. Proponowane narzędzie wspiera ten proces zachowując poprzedni stan wymagania w trakcie jego zapisu. Dzięki temu możliwe jest prześledzenie zmian dokonywanych w wymaganiach.

3.5.6 Centralne repozytorium

Proponowany system jest dostępny przez przeglądarkę, dzięki czemu stanowi centralne repozytorium wiedzy na temat wymagań w projekcie. Interfejs www z elementami html5 powoduje, że narzędzie wymaga jedynie nowoczesnej przeglądarki internetowej, bez żadnych dodatkowych wtyczek. Dzięki temu aplikacja cechuje się wysokim poziomem dostępności.

3.5.7 Kolaboracja

Możliwość komentowania wymagań jest istotnym elementem nowoczesnego systemu, umożliwiającym pracę grupową nad wybranymi elementami.

3.5.8 Generator specyfikacji

Jedną z kluczowych funkcjonalności proponowanego rozwiązania jest automatyczne generowanie dokumentu specyfikacji wymagań na podstawie aktualnych danych w systemie. Użytkownik w każdej chwili jest w stanie wygenerować bieżącą wersję specyfikacji i poddać ją całościowej analizie. Częścią specyfikacji zostają zarówno wymagania jak i powiązane z nimi przypadki użycia.

3.5.9 Praktyczne zastosowanie

Proponowane funkcjonalności są oparte na rzeczywistych doświadczeniach, w rzeczywistych projektach w małych i średnich przedsiębiorstwach. Bazują na silnym przekonaniu, że narzędzie, aby było wykorzystywane, musi być wygodne i proste w użyciu oraz wносить wartość dodaną do istniejącej infrastruktury i metodyki pracy. Jednocześnie nie adresuje całego procesu zarządzania projektem — realizuje tylko swoje zadanie związane z przetwarzaniem i zarządzaniem wymaganiami. W dojrzałym przedsiębiorstwie, narzędzie służące do zarządzania wymaganiami powinno być dopełnione pełnoprawnym systemem zarządzania projektami, systemem kontroli wersji oraz systemem zgłaszania i monitorowania błędów (trac, redmine).

Rozdział 4

Technologie wykorzystane w implementacji

W poprzednim rozdziale opisano główne założenia proponowanego systemu i jego poszczególne funkcjonalności. W tym rozdziale opisano narzędzia, technologie i biblioteki, które zastosowano podczas budowy prototypu.

4.1 Zarys technologii

Implementację systemu oparto o technologie Java/JEE i pochodne. Jako podstawę szkieletu aplikacji wykorzystano framework Grails umożliwiający szybkie prototypowanie aplikacji internetowych. W warstwie bazy danych wykorzystano bazę postgresql oraz technologię mapowania obiektowo-relacyjnego Hibernate ORM, dostarczaną wraz z frameworkiem Grails.

Interfejs użytkownika oparto o Twitter Bootstrap — pre-definiowany zestaw stylów oraz komponentów do natychmiastowego wykorzystania w aplikacjach internetowych.

Moduł graficznej edycji diagramów UML został stworzony wykorzystując „jsUML2” — bibliotekę przygotowaną przez zespół prof. José Raúl’a Romero, na uniwersytecie w Kordobie.

Edytor tekstu oparty jest o bibliotekę JavaScript Ace Editor. Jest to wiodąca biblioteka, wykorzystywana m.in. w eksperymencie Cloud9 IDE —

webowym, zintegrowanym środowisku programistycznym zapoczątkowanym przez zespół Mozilli, pod nazwą Mozilla Skywrite.

Pierwotnie docelowym środowiskiem testowym dla proponowanego systemu był Google App Engine, jednak ze względu na ograniczenia tej usługi, prototyp umieszczono na darmowej infrastrukturze firmy Heroku (heroku.com) znacznie lepiej obsługującej nowoczesne aplikacje oparte o Javę.

4.2 Framework Grails

Framework Grails jest stosunkowo młodą technologią opartą na sprawdzonych rozwiązaniach. Firma która zajmuje się jej rozwojem jest tą samą organizacją, odpowiedzialną za stworzenie Spring Framework — jednej z najpopularniejszych i jednocześnie najbardziej rozbudowanych platform programistycznych dla języka Java dostępnych na rynku. To właśnie Spring Framework leży u podstaw Grails i wraz z Hibernate ORM stanowi trzon technologiczny tego narzędzia.

Grails integruje najlepsze praktyki i narzędzia zarówno ze Spring Framework jak i z Hibernate ORM. Jest poniekąd odpowiedzią środowiska Javy, na rosnącą konkurencyjność frameworków opartych o dynamicznie typowane, skryptowe języki programowania jak Django (język Python) czy Ruby On Rails (Ruby). Jednocześnie, ze względu na fakt iż jest swoistą nakładką na technologie oparte na Javie, dostarcza znacznie większych możliwości niż konkurenci, wynikających z rozbudowanego ekosystemu Javy.

Język Groovy

Podstawowym językiem programowania w platformie Grails, jest język Groovy. Groovy jest dynamicznie kompilowanym językiem o składni i filozofii zbliżonej do takich języków jak Python lub Ruby. Kod bajtowy będący wynikiem kompilacji uruchamiany jest na wirtualnej maszynie Javy, przez co język integruje się niemal w sposób przeźroczysty z technologiami opartymi o JVM. Ponadto, kod programu napisanego w Javie jest poprawnym programem Groovy zarówno pod względem syntaktycznym jak i semantycznym.

Bardziej zwięzła i przyjazna składnia Groovy wraz z szerokimi możliwościami Javy sprawia, że ten język skryptowy jest solidnym rozwiązaniem o szerokim spektrum zastosowań.

W celach porównawczych, na Listingu 4.1 zamieszczono znany, trywialny problem programistyczny „FizzBuzz”, zarówno w języku Groovy, jak i Java. Program „FizzBuzz” otrzymując na wejściu ciąg liczb, wypisuje na konsolę słowo „Fizz” jeśli dana liczba jest podzielna przez 3, słowo „Buzz”, jeśli dana liczba jest podzielna przez 5, natomiast słowo „FizzBuzz” w przypadku podzielności przez 15.

```
1
3 // Groovy:
4 for (i in 1..100) {
5     println "${i%3?'': 'Fizz '}${i%5?'': 'Buzz '}" ?: i
6 }
7
9 //Java:
10 public class FizzBuzz{
11     public static void main(String[] args){
12         for(int i= 1; i <= 100; i++){
13             if(i % 15 == 0){
14                 System.out.println("FizzBuzz");
15             } else if(i % 3 == 0){
16                 System.out.println("Fizz");
17             } else if(i % 5 == 0){
18                 System.out.println("Buzz");
19             } else{
20                 System.out.println(i);
21             }
22         }
23     }
24 }
```

Listing 4.1: Program FizzBuzz

Grails

W szczególności filozofia Grails jest bardzo zbliżona do popularnego frameworka Ruby On Rails (RoR) umożliwiającego ekspresowe prototypowanie aplikacji obsługujących komunikację z bazą danych w zakresie (1) tworzenia, (2) odczytu, (3) aktualizacji i (4) usuwania danych za pomocą formularzy. Tego typu aplikacje popularnie nazywane są aplikacjami „CRUD” — create, retrieve, update, delete). Wiele przyjętych rozwiązań w Grails zostało zaczerpniętych z RoR, jednak oba frameworki znacznie różnią się pod względem technologicznym. Szczegółowe porównanie tych technologii leży poza zakresem tej pracy, jednak osoby zainteresowane, powinny zapoznać się z doskonałym wątkiem w serwisie Stackoverflow, przytoczonym w bibliografii [24].

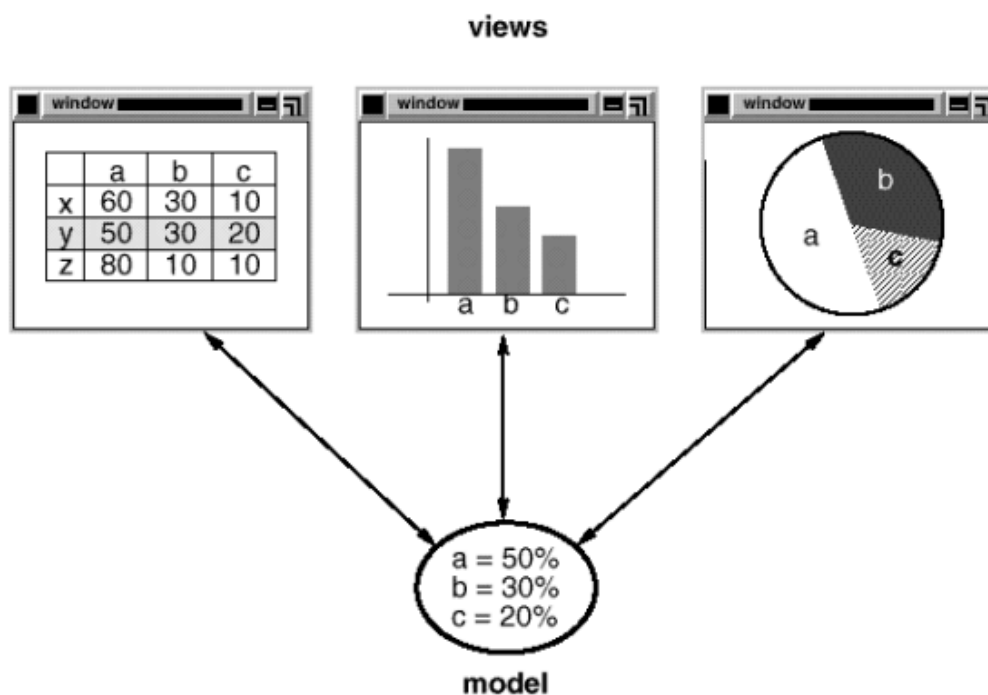
Grails jest silnie oparte o paradygmat „convention over configuration” — architekturze zakładającej minimalizację potrzeby konfiguracji aplikacji przez programistę na rzecz przyjętych konwencji, do których należy się stosować, aby wydajnie dostarczać działające rozwiązania. Podejście to dotyczy wszelkich aspektów związanych z tworzeniem aplikacji w Grails — od struktury katalogów, przez umiejscowienie i zawartość plików konfiguracyjnych, po nazewnictwo klas, zmiennych i zastosowanie odpowiednich struktur danych.

Model — View — Controller

Oparcie warstwy webowej w Grails na wzorcu Model-View-Controller (MVC) jest naturalnym podejściem w nowoczesnych frameworkach ułatwiających tworzenie aplikacji internetowych. Wzorzec MVC [13] pierwotnie dotyczył tworzenia aplikacji okienkowych w Smalltalk’u-80 [9]. Podstawowym założeniem MVC jest separacja danych i logiki od ich reprezentacji graficznej. MVC składa się z trzech rodzajów obiektów.

Model jest reprezentacją danych, wiedzy. Model może być odpowiednikiem jakiegoś obiektu lub strukturą wielu obiektów.

Widok jest wizualną reprezentacją modelu, „pobiera” dane na temat obiektu i wyświetla je w odpowiedni sposób, często uwydatniając pewne elementy modelu, inne z kolei odpowiednio ukrywając. Funkcja ta czyni z widoku pewnego



Rysunek 4.1: Model-View-Controller - różna reprezentacja graficzna tych samych danych. Źródło: [13].

rodzaju filtr prezentacji danych.

Kontroler stanowi łącznik pomiędzy użytkownikiem a systemem, z którym użytkownik wchodzi w interakcję. Dostarcza użytkownikowi odpowiednich widoków, adekwatnych do stanu systemu [23]. Diagram umieszczony na Rysunku 4.1 przedstawia poglądową strukturę wzorca MVC.

Praca z Grails

Struktura katalogów nowo utworzonej aplikacji Grails została załączona na Listingu 4.2:

```
2
4  + grails-app
   + conf          ---> lokalizacja artefaktow
       konfiguracji
   + hibernate     ---> opcjonalne pliki konfiguracyjne
       hibernate
6  + spring        ---> opcjonalne pliki konfiguracyjne
       spring
   + controllers   ---> kontrolery aplikacji
   + domain        ---> klasy reprezentujace modele
   + i18n          ---> zlokalizowane komunikaty i18n
10  + services      ---> warstwa serwisowa
   + taglib        ---> biblioteki znacznikow
12  + util          ---> klasy pomocnicze
   + views         ---> lokalizacja widokow
14  + layouts      ---> lokalizacja
   + lib
16  + scripts      ---> skrypty gant
   + src
18  + groovy        ---> opcjonalnie, zrodla groovy
                       inne niz w grails-app/*
20  + java         ---> opcjonalnie, zrodla java
   + test         ---> generowane testy aplikacji
22  + web-app
       + WEB-INF
```

Listing 4.2: Struktura katalogów Grails

GORM — Grails Object Relational Mapping

Po wstępnej analizie, mając wiedzę na temat dziedziny problemu, w prosty sposób można utworzyć podstawową funkcjonalność przykładowej aplikacji. Proces tworzenia elementów budowanego systemu rozpoczyna się od stworze-

nia klasy Groovy reprezentującej model (Listing 4.3 przedstawia rozwiązanie na przykładzie implementacji z systemu Reqmanager):

```

1      class Project {
2          String name
3          String description
4          String mainNote
5          java.sql.Timestamp timestamp
6          java.util.Date deadline
7          static hasMany = [requirements:Requirement]
8      }

```

Listing 4.3: Klasa modelu Project.groovy

Jeśli nie skonfigurujemy aplikacji w niestandardowy sposób, po stworzeniu powyższej klasy Groovy w katalogu grails-app/domain, Framework Grails przy pomocy Hibernate utworzy strukturę w bazie danych odpowiadającą strukturze modeli. Wynik odzwierciedlenia obiektu *Project* w relacyjnej bazie danych, zamieszczono na Listingu 4.4:

Table "public.project"		
Column	Type	Modifiers
id	bigint	not null
version	bigint	not null
deadline	timestamp without time zone	
description	character varying(255)	not null
name	character varying(255)	not null
timestamp	timestamp without time zone	not null
main_note	character varying(255)	
Indexes:		
"project_pkey" PRIMARY KEY, btree (id)		
Referenced by:		
TABLE "requirement" CONSTRAINT "fk15a8dc43ffd118f2" FOREIGN KEY (project_id) REFERENCES project(id)		

Listing 4.4: Struktura bazy danych dla modelu Project

Powyższy przykład przedstawia, jak w prosty sposób, przy niskich nakładach pracy, programista jest w stanie zbudować szkielet aplikacji zasilanej

bazą danych. Jak widać, pola klasy Groovy, zostały odpowiednio odwzorowane w strukturze danych wraz z odpowiednimi typami po stronie bazy.

Na szczególną uwagę zasługuje konstrukcja w klasie *Project* przedstawiona na Listingu 4.5.

```
1 static hasmany = [requirements: requirement]
```

Listing 4.5: Relacja jeden-do-wielu

W języku Groovy konstrukcja „[:]” jest reprezentacją pustej tablicy asocjacyjnej — struktury danych umożliwiającej przechowywanie kolekcji par „klucz — wartość”. W Groovy, domyślną kolekcją inicjalizowaną przy pomocy literału „[:]” jest obiekt klasy `java.util.LinkedHashMap`. Zatem powyższa zmienna statyczna jest referencją do obiektu klasy `LinkedHashMap`, zawierającego jedną parę, gdzie kluczem jest ciąg znaków „requirements” a wartością, jest klasa *Requirement*. W ten sposób Grails obsługuje relacje jeden-do-wielu. W analogiczny sposób, istnieje możliwość definiowania pozostałych rodzajów relacji, tj. jeden-do-jeden, wiele-do-wielu, etc). Konstrukcja taka, umożliwia uzyskanie listy wymagań w projekcie, jak zaprezentowano na Listingu 4.6:

```
1 //pobiera z bazy projekt o id = 1
   def project = Project.get(1)
3
5 //zwraca liste wymagan podlaczonych do projektu
   def reqs = project.requirements
```

Listing 4.6: Wymagania przypisane do projektu

Kontrolery i widoki

W celu połączenia konkretnych widoków z akcjami kontrolerów, konwencja Grails wymaga odpowiedniego nazewnictwa metod w klasach kontrolerów. Generalizując, należy stosować się do zasady, aby nazwa metody odpowiedzialnej za realizację wybranej akcji kontrolera, posiadała plik o tej samej nazwie, z rozszerzeniem. gsp, w odpowiednim katalogu w ścieżce `grails-app/views`. Poniżej zamieszczono przykład realizacji akcji „index” w kontrolerze

„ProjectController”. Wbudowany w Grails mechanizm odwzorowywania adresów URL i przekierowywania do odpowiednich akcji kontrolerów automatycznie wykrywa którą metodę należy wykonać. Zatem, po wskazaniu przeglądarce internetowej adresu `http://reqmanager.herokuapp.com/project/index` system, przez konwencję, wykryje konieczność wywołania metody „index” w kontrolerze „ProjectController”. Z kolei po wykonaniu tej metody, (o ile programista nie zdecyduje inaczej) automatycznie wyświetlany jest widok `index.gsp` umieszczony w katalogu `grails-app/views/project`. W widoku, do wykorzystania dostępne są zmienne ustawione w tabeli asocjacyjnej, zwracanej przez kontroler (w tym przypadku, zmienna „project” zawierająca referencję do listy wszystkich projektów w systemie). Listing 4.7 przedstawia klasę kontrolera z akcją *index* oraz odpowiadający mu widok *index.gsp*.

```
2 // klasa kontrolera:
package pl.edu.pjwstk.reqmanager

4

6 class ProjectController {
    def index = {
        return [projects : Project.list()]
8    }
}

10

12 //widok w pliku o nazwie index.gsp
//umieszczony w katalogu grails-app/views/project/
14 <!DOCTYPE html>
<html>
16     <div>
        <h1>Project list</h1>
18     <g:each in='${projects}' var='project'>
        Nazwa: ${project.name}<br />
20         Deadline: ${project.deadline.format("dd-MM yy")}<br
            />
        Opis: ${project.description}
22     </g:each>
    </div>
24 </html>
```

Listing 4.7: Widok i kontroler

Należy jeszcze zwrócić uwagę na pewien detal w strukturze kodu źródłowego kontrolera. Grails dopuszcza dwa podejścia do deklarowania akcji — (1) za pomocą klasycznych metod oraz (2) przy użyciu domknięć (closures). Formalnie domknięcia są obiektami wiążącymi funkcję oraz środowisko w jakim ta funkcja ma działać. Jest to specyficzna konstrukcja, będącą zdefiniowanym blokiem kodu, który można przekazać jako parametr innej metodzie lub funkcji. Przykład zamieszczony na Listingu 4.8 wyjaśnia zasadę na jakiej działają domknięcia w Groovy:

```
2      def closeAllProjects() {  
3          def projects = Project.list()  
4          projects.collect { it.open = false }  
      }
```

Listing 4.8: Domknięcie w Groovy

Powyższy kod pobiera do zmiennej „projects” listę wszystkich projektów w systemie. Następnie, na liście projektów wywoływana jest metoda „collect”. Metoda collect (zdefiniowana w interfejsie Collection) przyjmuje jako parametr blok kodu, domknięcie. Metoda collect iteruje po wszystkich elementach zbioru na rzecz którego została wykonana i wykonuje blok kodu przekazany jej jako parametr na kolejnych elementach kolekcji, następnie zwracając listę zmodyfikowanych przez domknięcie elementów.

4.3 Spring Framework i Hibernate ORM

Sercem Grails jest Spring Framework oraz Hibernate ORM. Obie technologie są wiodące na rynku i szeroko stosowane w komercyjnych projektach dowolnych rozmiarów. Spring jest de facto zestawem narzędzi, wzorców projektowych i bibliotek realizujących ogromną ilość funkcjonalności, mających za zadanie przyspieszyć proces wytwarzania oprogramowania oraz w pewnym zakresie zapewnić wysoką jakość tworzonych rozwiązań. Przede wszystkim jednak, jest tzw. kontenerem Inversion Of Control (odwrócenie sterowania)

[12]. Prócz IoC Spring Framework to potężna platforma integrująca wiele rozwiązań, takich jak webowy framework Spring MVC, biblioteki i wzorce dotyczące bezpieczeństwa (Spring Security), wrapper jdbc, transakcje (Spring JDBC Templates), etc.

Hibernate ORM jest technologią pozwalającą na mapowanie rekordów z relacyjnych baz danych, na obiekty w systemie przy pomocy plików konfiguracyjnych xml. W Grails, potrzeba konfiguracji warstwy ORM ogranicza się do minimum, a większość standardowych zadań można zrealizować konfigurując odpowiednio blok *mapping* w klasie modelu.

4.4 Pozostałe technologie

W poniższych sekcjach opisano komplementarne technologie jakie zastosowano w trakcie implementacji prototypu.

4.4.1 Postgresql

Trwałość danych w aplikacjach internetowych standardowo realizowana jest przez bazę danych. Na potrzeby prezentowanego prototypu, wykorzystano Postgresql. Jest to obiektowo-relacyjna baza danych udostępniona na licencji Wolnego i Otwartego Oprogramowania. PostgreSQL implementuje znaczną część standardu SQL:2008.

4.4.2 JavaScript, jQuery i biblioteka jsUML2

Wiele funkcjonalności zrealizowano po stronie klienta, wykorzystując język JavaScript. W niewielkim zakresie wykorzystano popularną bibliotekę jQuery, wspierającą realizację najczęściej pojawiających się problemów w trakcie prac nad aplikacjami internetowymi. Jednak wykorzystanie jQuery w prototypie systemu jest marginalne i ogranicza się jedynie do zastosowania metody *ajax()* ułatwiającej wykonywanie asynchronicznych wywołań serwera.

jsUML2

jsUML2 jest biblioteką stworzoną przez środowisko naukowe, na uniwersytecie w Kordobie. Jest to zaimplementowany w całości w języku JavaScript, rozbudowany zestaw klas umożliwiający tworzenie wielu rodzajów diagramów UML w przeglądarce internetowej. jsUML wykorzystuje najnowszą specyfikację języka znaczników HTML5. Obiekt *canvas* oferuje szerokie spektrum możliwości programistom aplikacji internetowych. Element *canvas* umożliwia programistyczne tworzenie kształtów i obrazów bitmapowych w przeglądarce internetowej. Canvas udostępnia programistom interfejs *HTMLCanvasElement* umożliwiający modyfikację zawartości elementów canvas w htmlu. Metoda interfejsu *getContext()* zwraca tzw. „drawing context” — w zależności od parametru przekazanego metodzie *getContext()*, zwracany jest obiekt klasy (1) *CanvasRenderingContext2D* w przypadku przekazania parametru '2d' lub (2) *WebGLRenderingContext* w przypadku przekazania parametru 'experimental-webgl'.

Poniżej zamieszczono przykładowy kod, umożliwiający rozpoczęcie pracy obiektem klasy *CanvasRenderingContext2D*. Wykonanie kodu z Listingu 4.9 spowoduje narysowanie prostokąta:

```
2      var canvas = document.getElementById('example');
      var drawingContext = canvas.getContext('2d')

4

      drawingContext.fillStyle = "rgb(200,0,0)";
6      drawingContext.fillRect(10, 10, 55, 50);
```

Listing 4.9: Przykład HTML5 canvas

Biblioteka jsUML2 składa się z dwóch pakietów klas: (1) *UDCore* — zawierający bazowe klasy umożliwiające tworzenie i obsługę graficzną diagramów i komponentów pomocniczych; (2) *UDModules* — zawierający faktyczną implementację diagramów standard UML, zbudowaną na bazie klas pierwotnych z pakietu *core*.

Głównymi elementami w poszczególnych pakietach są:

- *UDCore*

- Diagram
- Node (węzeł)
- Relation (relacja)
- Modules
 - Use case diagrams
 - Class diagrams
 - Component diagrams
 - Message sequence diagrams
 - State charts

Na listingu 4.10 zawarto wykaz klas dostępnych w pakiecie *core*:

```
2      AttributeFields.js
      AttributeItem.js
      CircleSymbol.js
4      CollapsibleFields.js
      Component.js
6      ComponentSymbol.js
      ConnectorItem.js
8      DataStoreItem.js
      Diagram.js
10     Dialog.js
      Element.js
12     Elliptical.js
      GuardItem.js
14     JSFun.js
      JSGraphic.js
16     LoopItem.js
      NodeFigure.js
18     Node.js
      ObjectItem.js
20     OperationFields.js
      OperationItem.js
22     Point.js
      Rectangular.js
```

```

24      RegionItem.js
      Region.js
26      RegionLine.js
      RelationEnd.js
28      Relation.js
      RelationLine.js
30      Rhombus.js
      RoleItem.js
32      Rombo.js
      Separator.js
34      Space.js
      SpecificationItem.js
36      StereotypeFields.js
      StereotypeItem.js
38      SuperComponent.js
      SuperNode.js
40      Tab.js
      TextArea.js
42      TextBox.js
      TextFields.js
44      Text.js
      TransitionItem.js

```

Listing 4.10: Klasy bazowe jsUML2

W pakiecie *modules*, do dyspozycji programisty znajdują się pod-pakiety klas przedstawione na Listingu 4.11:

```

1      activity
      class
3      component
      generic
5      profile
      sequence
7      stateMachine
      usecase

```

Listing 4.11: Pakiety modułów jsUML2

Listing 4.12 zawiera strukturę dokumentu html, jaki należy stworzyć w celu inicjalizacji biblioteki jsUML2.

```
2      <div id="ud_diagram_div" style="position: relative">
      <canvas id="c1" class="ud_diagram_canvas" style="
          position: absolute;" width="500")
      <canvas id="c2" class="ud_diagram_canvas" style="
          position: absolute;" width="500")
4  </div>
```

Listing 4.12: HTML canvas element

Niezbędna jest również inicjalizacja odpowiedniego diagramu (na przykładzie diagramu przypadków użycia, na Listingu 4.13):

```
2      var mainEl = document.getElementById('c1');
3      var motionEl = document.getElementById('c2');
4
5
6      var mainCtx = mainEl.getContext('2d');
7      var motionCtx = motionEl.getContext('2d');
8
9
10     var div = document.getElementById('ud_diagram_div')
11
12     var diagram = new UMLUseCaseDiagram()
13
14
15     //metoda initialize() przyjmuje kolejno parametry:
16     //(1) unikalny identyfikator diagramu
17     //(2) element blokowy html bedacy kontenerem diagramu
18     //(3) obiekt 'context' stanowiacy 'gorna' warstwe
19     //     diagramu
20     //(4,5) rozmiary w pikslach
21
22     diagram.initialize(1, div, motionCtx, 600, 600)
```

Listing 4.13: HTML canvas element - JS

4.4.3 System kontroli wersji git i serwer heroku

W trakcie pracy wykorzystano system kontroli wersji *git* oraz serwer heroku.

Git jest systemem kontroli wersji udostępnionym na licencji GPL, stworzonym przez Linusa Torvaldsa w 2005 roku. Głównymi założeniami projektu były szybkość działania, bezpieczeństwo oraz rozwiązanie problemów jakie twórca uważał są najbardziej rażące w systemach CSV oraz SVN.

Heroku udostępnia platformę hostingową obsługującą wiele języków programowania. Zasada działa serwerów Heroku, oparta jest na koncepcji utrzy-

mania aplikacji w „chmurze” serwerów — infrastruktura sprzętowa jest niewidoczna dla klientów. Użytkownicy tej usługi wykupują moc obliczeniową serwerów odpowiadającą potrzebom utrzymywanych aplikacji, płacąc za czas dostępu do sprzętu o wybranej charakterystyce. Główną zaletą tego rozwiązania jest możliwość natychmiastowego skalowania aplikacji. W przypadku znacznego wzrostu ruchu na serwerze, użytkownik może wykupić dodatkową moc obliczeniową, bez konieczności przerywania ciągłości działania usługi w sieci. Heroku udostępnia swoją infrastrukturę w podstawowym zakresie, bez konieczności uiszczania opłat, dlatego jest to rewelacyjne rozwiązanie na potrzeby testów i tworzenia rozwiązań „proof-of-concept”. Dzięki natywnemu wsparciu systemu kontroli wersji Git, instalacja aplikacji w chmurze, polega na przesłaniu na serwer aktualizacji kodu źródłowego i wykonania polecenia inicjalizującego instalację kodu na serwerze heroku. Listing 4.14 przedstawia tę procedurę.

```
1 ~$ git add .  
~$ git commit -m 'update'  
3 ~$ git push origin master
```

Listing 4.14: Commit i wysłanie kodu na serwer Heroku

4.4.4 Środowisko programistyczne (Linux, vim)

Do stworzenia zarówno prototypu aplikacji, jak i do edycji i składu części tekstowej niniejszej pracy, wykorzystano zasadniczo narzędzia dostępne w systemie operacyjnym Linux (Ubuntu Linux v11.10).

Do edycji kodu źródłowego wykorzystano klasyczny edytor tekstowy vim z zainstalowanym dodatkiem ułatwiającym poruszanie się w strukturze katalogów grails.

Część opisowa niniejszej pracy również powstała w edytorze tekstu vim. Weryfikacja pisowni została wykonana przy użyciu narzędzia aspell z zainstalowanym słownikiem języka polskiego. Do składu tekstu zastosowano popularne oprogramowanie L^AT_EX.

Rozdział 5

Rozwiązania implementacyjne

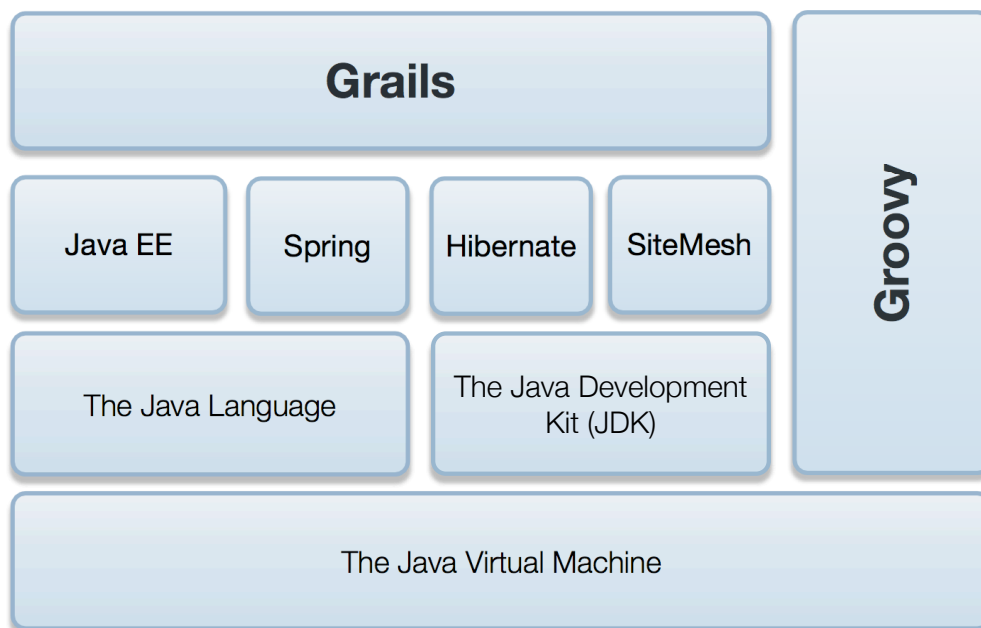
W poprzednim rozdziale opisano narzędzia jakie wykorzystano w trakcie budowy prototypu. W niniejszym rozdziale, przedstawiono zastosowane konkretne rozwiązania implementacyjne. Szczególną uwagę zwrócono na moduł edycji diagramów przypadków użycia oraz generator specyfikacji wymagań.

5.1 Architektura systemu

System Reqmanager zbudowany jest w oparciu o, opisaną w rozdziale 4, technologię Grails, integrującą wiele istniejących rozwiązań zbudowanych na bazie wirtualnej maszyny Javy. Na Rysunku 5.1 przedstawiono ogólną architekturę Grails.

Definiowanie wymagań w systemie Reqmanager polega na jednoczesnym dostępie do edytora tekstu oraz edytora diagramów przypadków użycia. Użytkownik, mając otwartą aplikację, oprócz tekstowego opisu wymagania, ma możliwość dodawania oraz edycji przypadków użycia w formie graficznej.

Aplikacja Reqmanager jest typową aplikacją internetową typu klient-serwer, gdzie klientem jest przeglądarka użytkownika korzystającego z aplikacji zainstalowanej na zdalnym serwerze, odpowiednio reagującej na żądania klienta na zasadzie żądanie — odpowiedź (request — response). Przeglądarka internetowa jest tak zwanym „cienkim klientem” z racji silnego uzależnienia działania od serwera aplikacji. Na Rysunku 5.2 przedstawiono poszczególne



Rysunek 5.1: Architektura Grails

moduły proponowanego systemu wraz z podziałem na warstwy.

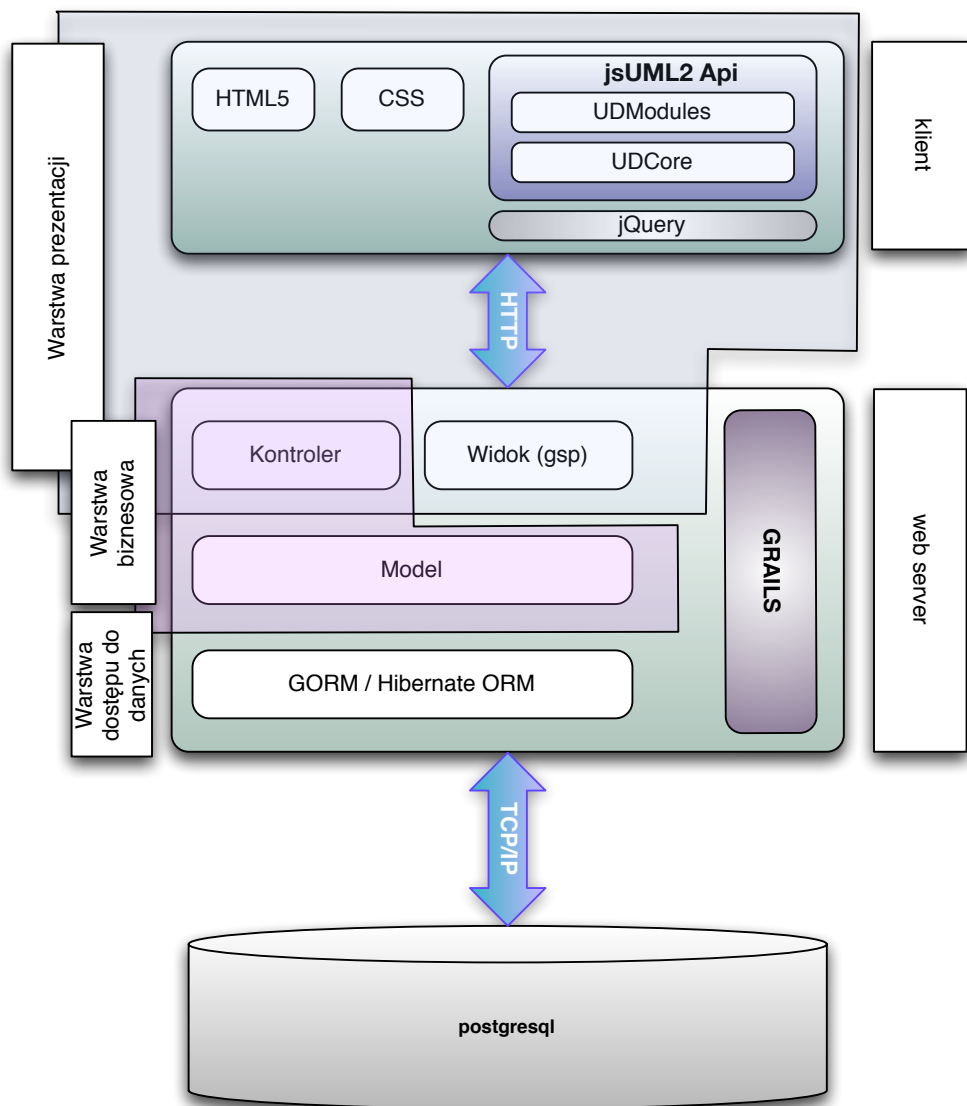
Na potrzeby aplikacji Reqmanager, utworzono następującą strukturę modeli aplikacji:

```
Diagram
Project
Requirement
UseCase
```

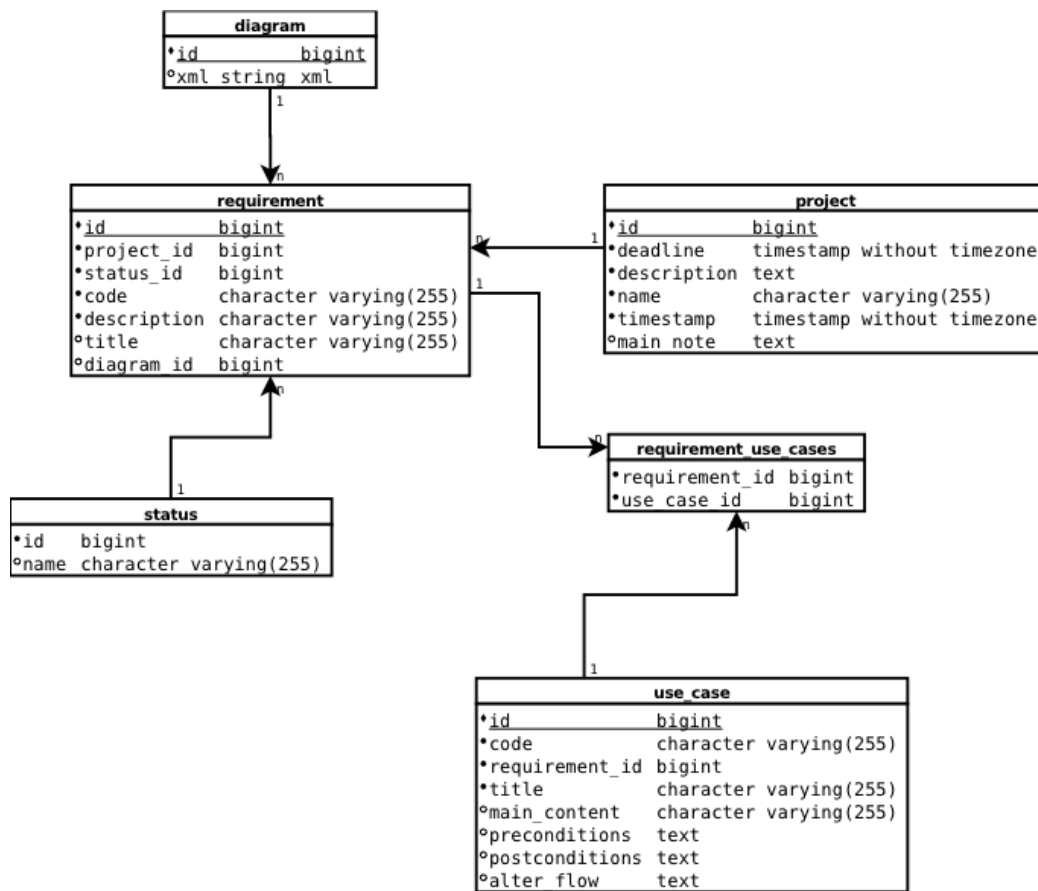
Struktura bazy danych jest generowana automatycznie, na podstawie analizy klas modeli. Diagram utworzonej bazy, przedstawiono na rysunku 5.3

5.2 Funkcjonalności systemu

Poniżej opisano poszczególne funkcjonalności proponowanego rozwiązania wraz z przykładami realizacji w kodzie.



Rysunek 5.2: Ogólna architektura systemu



Rysunek 5.3: Diagram struktury bazy danych systemu Reqmanager

5.2.1 Tworzenie diagramów

Diagramy przypadków użycia w systemie edytowane są po stronie klienta, dzięki wykorzystaniu elementu *canvas* wprowadzonego niedawno do specyfikacji języka HTML5. Obsługa tej funkcjonalności po stronie przeglądarki, stanowi problem związany z zapisem stanu diagramu. W celu rozwiązania tego problemu, zaimplementowano mechanizm serializacji stanu elementów diagramu w formacie xml do bazy danych. Cel ten osiągnięto, wykorzystując metodę *getXMLString()* udostępnioną przez wszystkie obiekty dziedziczące po klasie *Diagram* w bibliotece jsUML2. Przykładową strukturę zserializowanego diagramu, umieszczono na Listingu 5.3. Dodanie nowego diagramu wiąże się zatem z inicjalizacją i wyświetleniem użytkownikowi obszaru roboczego diagramu bez komunikacji z serwerem. Diagram tworzony jest po załadowaniu widoku edycji wymagania, jak pokazano na Listingu 5.1:

```
1      window.onload = function() {  
2          var app = new AppUseCase("ud_diagram_div");  
3      }
```

Listing 5.1: Widok requirement/edit.gsp

AppUseCase jest obiektem JavaScript typu singleton, którego skrócona implementacja została przedstawiona na Listingu 5.2:

```
1      var AppUseCase = function(elementId) {  
2          var xmlstr = document.getElementById('diagramXml').  
3              value  
4          var ucDiag = new UMLUseCaseDiagram()  
5          if(xmlstr) {  
6              ucDiag.setXMLString(xmlstr);  
7          }  
8  
9          var c1 = document.getElementById('c1');  
10         var c = c1.getContext('2d');  
11         var c2 = document.getElementById('c2');  
12         c2.onmousedown = function() { return false; };  
13         var mc = c2.getContext('2d')  
14  
15  
16         var div = document.getElementById('ud_diagram_div');  
17  
18  
19         ucDiag.initialize(11, div, c, mc, 600, 1000);  
20         ucDiag.draw();  
21     }
```

Listing 5.2: Implementacja obiektu JS AppUseCase

Powyższy kod JavaScript, pobiera wartość z elementu formularza i identyfikatorze „diagramXml” oraz tworzy nowy obiekt UMLUseCaseDiagram, zdefiniowany w pakiecie *modules* biblioteki jsUML2. Następnie weryfikuje czy pole *diagramXml* jest niepuste. Jeśli warunek jest prawdziwy, wywołuje metodę *setXMLString()* na obiekcie diagramu, dzięki czemu diagram inicjalizowany jest z wartościami wcześniej zserializowanymi do formatu xml i zapisanymi w bazie danych. Kolejne wiersze zajmują się utworzeniem kontekstów na bazie elementów *canvas*, aby na końcu zainicjować i narysować odpowiedni diagram w oknie przeglądarki użytkownika. Należy jednocześnie zaznaczyć, iż w przypadku, gdy nie istnieje zserializowana wersja diagramu w bazie (pole *diagramXml* jest puste), wówczas zostanie zainicjalizowany

nowy, pusty diagram, nieposiadający żadnych elementów.

Zapis danych xml

Dopiero gdy użytkownik wybierze opcję zapisu aktualnego wymagania, zserializowany stan diagramu w formacie xml, zostanie przesłany metodą POST do serwera. Następnie otrzymany po stronie serwera ciąg znaków xml zostanie przetworzony, utworzony zostanie obiekt klasy Diagram i na końcu nastąpi zapis obiektu do bazy danych. Za ten proces odpowiedzialna jest akcja *save* kontrolera *Requirement*. Jest to dość ograniczone rozwiązanie, które można byłoby usprawnić wprowadzając asynchroniczny, automatyczny zapis stanu diagramu co kilka sekund lub po każdej aktualizacji po stronie klienta.

```
1
3
5
7
9
11
13
15
<UMLUseCaseDiagram name="Use case diagram">
  <UMLSystem id="UMLSystem_4" x="165" y="97" ... >
    <superitem id="stereotypes" visibleSubComponents="
      true"/>
    <item id="name" value="System name"/>
    <UMLUseCase id="UMLUseCase_1" x="232" y="142" ... >
      <superitem id="stereotypes" visibleSubComponents="
        true"/>
      <item id="name" value="test test"/>
    </UMLUseCase>
    <UMLUseCase id="UMLUseCase_3" x="222" y="209" ... >
      <superitem id="stereotypes" visibleSubComponents="
        true"/>
      <item id="name" value="for req 74"/>
    </UMLUseCase>
  </UMLSystem>
</UMLUseCaseDiagram>
```

Listing 5.3: Struktura xml diagramu

Baza danych PostgreSQL obsługuje specjalny typ danych „xml”. Dzięki temu, po stronie bazy dokonywana jest weryfikacja poprawności dokumentu jaki programista ma zamiar zapisać. Niestety Framework Grails nie współpracuje dobrze z taką konstrukcją, dlatego niezbędna była implementacja rozwią-

zania rzutującego typ *String* na typ *SQLXMLType*. W tym celu została stworzona specjalna klasa w Javie, implementująca interfejs *org.hibernate.usertype.UserType* (patrz Listing 5.4). Hibernate udostępnia ten interfejs umożliwiając programistom tworzenie własnych typów danych i zdefiniowanie ich zachowania w stosunku do bazy. Dodatkowo, niezbędne było odpowiednie przygotowanie modelu, podając typ pola przechowującego xml jako *String*, a następnie stworzyć odpowiedni blok „mapping” w modelu. Na Listingu 5.5, dla przykładu zaprezentowano implementację klasy *Diagram* z systemu Reqmanager:

```
package pl.edu.pjwstk.reqmanager

import java.io.Serializable;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Types;
import org.hibernate.HibernateException;

/**
 * Store and retrieve a PostgreSQL "xml" column as a
 * Java string.
 */
public class SQLXMLType implements org.hibernate.
    usertype.UserType {

    private final int[] sqlTypesSupported = new int[] {
        Types.VARCHAR };

    @Override
    public int[] sqlTypes() {
        return sqlTypesSupported;
    }

    @Override
    public Class returnedClass() {
        return String.class;
    }
}
```

```

28      @Override
      public boolean equals(Object x, Object y) throws
          HibernateException {
          if (x == null) {
30              return y == null;
          } else {
32              return x.equals(y);
          }
34      }

36      @Override
      public int hashCode(Object x) throws
          HibernateException {
38          return x == null ? null : x.hashCode();
      }

40

42      @Override
      public Object nullSafeGet(ResultSet rs, String[]
          names, Object owner) throws HibernateException,
          SQLException {
          assert(names.length == 1);
44          String xmldoc = rs.getString( names[0] );
          return rs.isNull() ? null : xmldoc;
46      }

48      @Override
      public void nullSafeSet(PreparedStatement st,
          Object value, int index) throws
          HibernateException, SQLException {
50          if (value == null) {
              st.setNull(index, Types.OTHER);
          } else {
52              st.setObject(index, value, Types.OTHER);
          }
54      }

56

58      @Override
      public Object deepCopy(Object value) throws
          HibernateException {

```

```

        if (value == null)
            return null;
        return new String( (String)value );
    }

    @Override
    public boolean isMutable() {
        return false;
    }

    @Override
    public Serializable disassemble(Object value)
        throws HibernateException {
        return (String) value;
    }

    @Override
    public Object assemble(Serializable cached, Object
        owner) throws HibernateException {
        return (String) cached;
    }

    @Override
    public Object replace(Object original, Object
        target, Object owner) throws HibernateException
    {
        return original;
    }
}

```

Listing 5.4: Implementacja niestandardowego typu SQLXMLType

```

1      package pl.edu.pjwstk.reqmanager
2      public class Diagram {
3          String name
4          String xmlString
5
6
7          static belongsTo = Requirement
8          static mapping = {
9              xmlString type: SQLXMLType
10             }
11         static constraints = {
12             name(nullable:true)
13         }
14     }

```

Listing 5.5: Implementacja modelu Diagram

5.2.2 Współdzielenie diagramów

Istotnym elementem systemu jest współdzielenie diagramów między wieloma wymaganiami. Każdy diagram może zostać przypisany do kilku wymagań. Wówczas modyfikacja przypadków użycia w trakcie edycji konkretnego wymagania, wiąże się z aktualizacją stanu diagramu we wszystkich korzystających z niego miejscach w systemie.

5.2.3 Mapowanie przypadków użycia

W związku z możliwością współdzielenia diagramów pomiędzy wieloma wymaganiami, w aktualnej wersji prototypu, przypadki użycia są łączone z wymaganiem, tylko po wykonaniu przez użytkownika akcji wskazania przypadku użycia jako realizującego aktualnie modyfikowane wymaganie.

Na etapie edycji diagramu, użytkownik ma możliwość wskazania konkretnych use case'ów znajdujących się na diagramie, jako odpowiedzialnych za realizację wybranego wymagania. Powiązanie przypadku użycia z wymaganiem polega na wyborze opcji *mark for requirement* z paska narzędzi diagramu, a następnie kliknięciu w dany przypadek użycia na diagramie. Po

zatwierdzeniu operacji, zostaje wysłane asynchroniczne żądanie do serwera, pod adres `http://reqmanager.herokuapp.com/requirement/addUseCase/id_wymagania`. W żądaniu przekazywane są do serwera parametry niezbędne dla zestawienia przypadku użycia z wymaganiem. Poniżej zamieszczono listę tych parametrów:

- *id* — identyfikator wymagania do którego dodawany jest przypadek użycia
- *clickedUCName* — nazwa przypadku użycia wprowadzona przez użytkownika
- *clickedUCId* — identyfikator przypadku użycia wygenerowany automatycznie przez bibliotekę jsUML2

Asynchronicznie wysłane do serwera żądanie zostaje przekazane wraz z parametrami do akcji *addUseCase* kontrolera *Requirement*. Kod odpowiedzialny za połączenie przypadku użycia z wymaganiem, został załączony na Listingu 5.6.

```
def addUseCase = {  
  2   def requirement = Requirement.get(params.id)  
      def useCase = UseCase.findByTitle(params.clickedUCName)  
  4   if(useCase == null) {  
        useCase = new UseCase(title: params.clickedUCName,  
                               code: params.clickedUCId)  
  6     requirement.addToUseCases(useCase)  
        render(text: "ok")  
  8   } else {  
        if(!requirement.useCases.contains(useCase)) {  
 10     requirement.addToUseCases(useCase)  
        render(text: "ok")  
 12   }  
      }  
 14 }
```

Listing 5.6: Kod łączący przypadek użycia z wymaganiem

Jak można odczytać z powyższego kodu źródłowego, akcja ta, pobiera odpowiednie wymaganie oraz podejmuje próbę znalezienia przypadku użycia po nazwie. Jeśli dany przypadek użycia nie istnieje jeszcze w bazie danych (nowo utworzony przypadek użycia), zostaje stworzony nowy obiekt klasy UseCase z odpowiednimi parametrami (nazwa nadana przez użytkownika oraz identyfikator wygenerowany po stronie klienta). W przeciwnym razie, zostaje on dodany do listy przypadków użycia połączonych asocjacją z danym wymaganiem, o ile nie istnieje już na tej liście.

5.2.4 Przypadki użycia a wymagania

W celu odpowiedniego oznaczenia przypadków użycia przypisanych do wybranego wymagania, zaimplementowano algorytm dwustopniowej inicjalizacji diagramu. Pierwszym krokiem jest ekstrakcja z bazy danych odpowiedniego wymagania, weryfikacja czy jest do niego przypisany diagram oraz czy istnieją dla niego jakieś przypadki użycia. Na podstawie tych informacji, ładowany jest widok *requirement/show.gsp*. Następnie, po załadowaniu strony uruchamiany jest skrypt po stronie klienta, odwołujący się asynchronicznie z powrotem do serwera z „pytaniem” o nazwy przypadków użycia przypisanych do danego wymagania. Następnie, do pamięci ładowana jest struktura xml diagramu. Program iteruje po wszystkich węzłach w poszukiwaniu pokrywających się przypadków użycia. Jeśli znajdzie choć jeden, zmienia kolor jego tła na żółty. Listing 5.7 przedstawia implementację tego algorytmu.

```
var reqUseCases = $.ajax({  
2   type: "GET",  
   url: "http://reqmanager.herokuapp.com/requirement/  
       getUseCases/" + reqId,  
4   cache: false  
}).done(function(xht){  
6   var dom = (new DOMParser()).parseFromString(diag.  
       diagram.getXMLString(), "text/xml");  
   var documentEle = dom.documentElement;  
8   var usecaseArray = documentEle.getElementsByTagName(''  
       UMLUseCase'');
```

```

10      for(var i = 0; i < usecaseArray.length; i++) {
12          for(var j = 0; j < usecaseArray[i].childNodes.length;
              j++) {
              var itemValue = usecaseArray[i].childNodes[j].
                  attributes.value;
14          if(itemValue != undefined) {
              if(xht.indexOf(itemValue.value.toString()) != -1)
              {
16                  var divv = document.getElementById("
                      ud_diagram_div");
                  var foundUseCase = null;

18
20                  for(var k = 0; k < diag.diagram._nodes.length;
                      k++) {
                      if(diag.diagram._nodes[k].getName().toString
                          ()
22                          === xht[xht.indexOf(itemValue.value.
                              toString())]) {
                              foundUseCase = diag.diagram._nodes[k];
24                          }
                      };
26                  if(foundUseCase) foundUseCase.
                      setBackgroundColor("#F3F5BC");
                      diag.diagram.draw();
28              }
              }
30          }
32      });

```

Listing 5.7: Mechanizm oznaczania przypadku użycia przypisanego do aktualnie otwartego wymagania

5.2.5 Edytor tekstu

W celu ułatwienia użytkownikowi pracy z tekstem, zaimplementowano edytor, oparty na bibliotece Ace Editor. Kod inicjujący edytor tekstu, znajduje się na Listingu 5.8.

```
var editor = ace.edit("description");  
2 var MarkdownMode = require("ace/mode/markdown").Mode;  
editor.setTheme("ace/theme/clouds_midnight");  
4 editor.getSession().setMode(new MarkdownMode());  
editor.getSession().setUseWrapMode(true);
```

Listing 5.8: Inicjalizacja edytora tekstu

Kod z Listingu 5.8 inicjuje edytor, załączając go do wskazanego węzła w dokumencie html (w tym przypadku jest to warstwa „description”). Następnie ustawia wygląd graficzny edytora oraz przestawia go w tryb pracy z formatem Markdown. Zawartość edytora tekstu, przekazywana jest do serwera po wysłaniu formularza przez użytkownika. Ponieważ zawartość formularza znajduje się w elemencie *div* w strukturze DOM strony, w celu przekazania jego wartości, jest ona kopiowana do specjalnego pola typu textarea. Proces ten zaprezentowano na Listingu 5.9.

```
1 $("#submitbtn").click(function() {  
    textarea.val(editor.getSession().getValue());  
3    $("#diagramXml").val(app.diagram.getXMLString());  
});
```

Listing 5.9: Kopiowanie zawartości edytora tekstu do pola textarea

5.2.6 Generowanie dokumentacji

W prototypie zaimplementowano mechanizm generowania specyfikacji wymagań przy użyciu bibliotek MarkdownJ oraz iText.

MarkdownJ jest napisaną w Javie implementacją konwertera tekstu do formatu HTML. Jest to bardzo prosta biblioteka, w której główną klasą jest *MarkdownProcessor*. Metoda *markdown()* przyjmuje jako parametr ciąg znaków (sformatowany znacznikami Markdown). W odpowiedzi zwracana jest ta

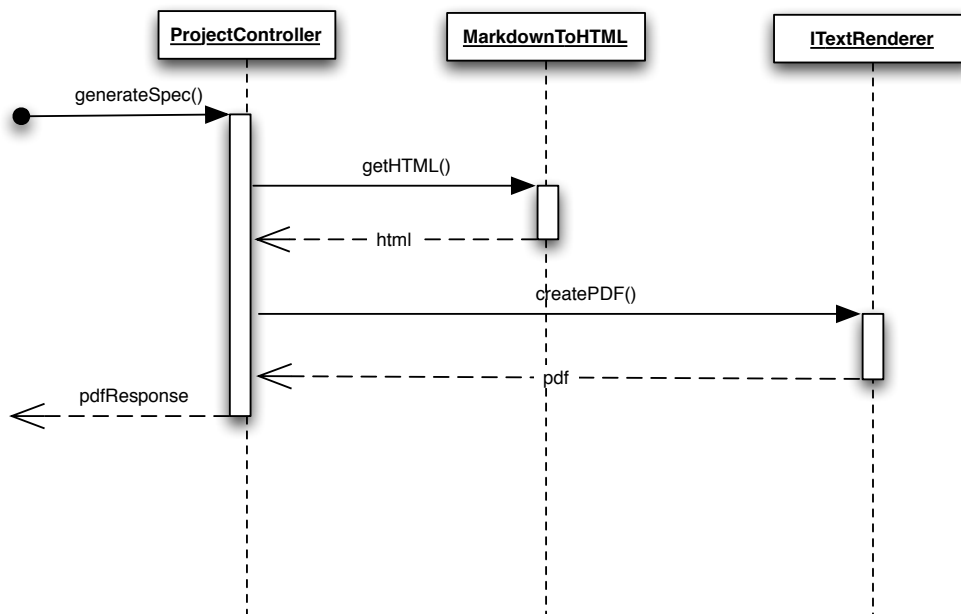
sama treść, osadzona w odpowiedniej strukturze HTML. Przykładowy kod konwertujący przedstawiono na Listingu 5.10.

```
2      def m = new MarkdownProcessor();
      StringBuffer sb = new StringBuffer()
4      sb.append("# Example header in Markdown")
      sb.append("\n")
      sb.append("This will render as a regular '<p>'
6      paragraph")
      String html = m.markdown(sb.toString());
```

Listing 5.10: Wykorzystanie klasy MarkdownProcessor

iText z kolei, jest zaawansowaną biblioteką wykorzystywaną do generowania wszelkiego rodzaju dokumentów PDF bezpośrednio z kodu Javy. W pracy wykorzystano klasę *ITextRenderer* dostępną w pakiecie org. xhtmlrenderer.pdf. Pozwala ona na generowanie dokumentów PDF w oparciu o strukturę XHTML. Połączenie MarkdownJ z biblioteką iText idealnie spełnia wymagania systemu w zakresie automatycznego tworzenia specyfikacji.

Generowanie specyfikacji w systemie przebiega w dwóch krokach. W pierwszej kolejności zostają pobrane niezbędne informacje z bazy danych dotyczące wybranego projektu, a zawartość poszczególnych pól, zostaje przekazana do klasy MarkdownToHTML konwertującej tekst do HTML. Następnie, zwrócona struktura jest przekazywana klasie ITextRenderer w celu wygenerowania dokumentu PDF. Zwrócony strumień danych zostaje wysłany do klienta jako odpowiedź serwera, co skutkuje możliwością zapisu pliku PDF na komputerze klienckim. Proces ten zobrazowany został na diagramie sekwencji, załączonym na Rysunku 5.4.



Rysunek 5.4: Generowanie dokumentu pdf specyfikacji

5.3 Przykład zastosowania

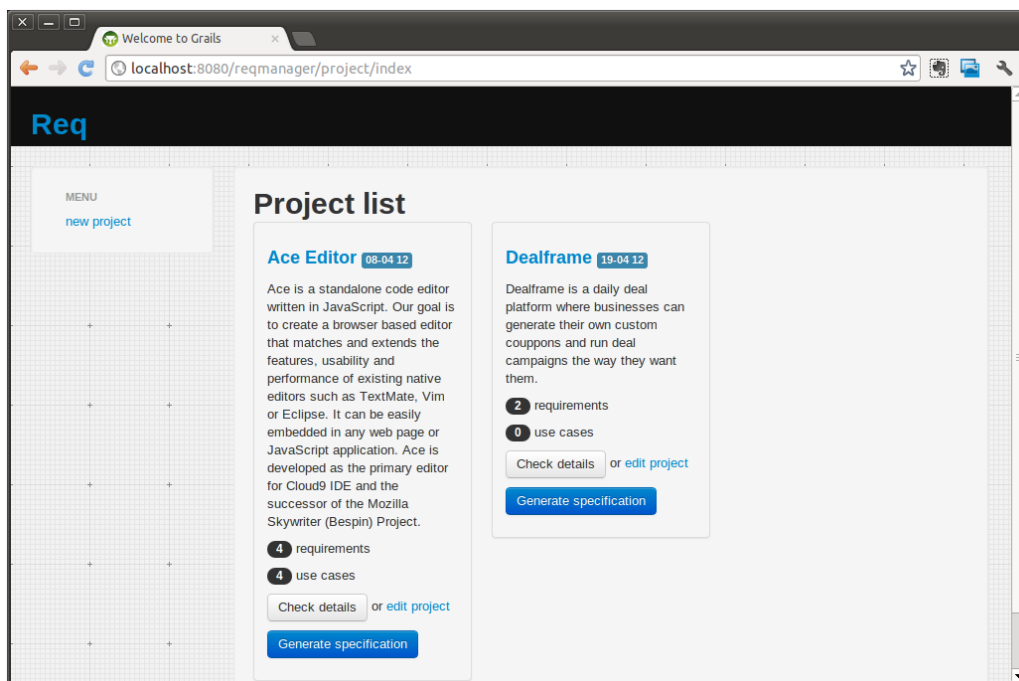
W tej sekcji opisano przykład zastosowania proponowanego narzędzia.

5.3.1 Ekran główny

Ekranem głównym systemu jest przegląd wszystkich projektów, przedstawiony na Rysunku 5.5. Przy każdym projekcie zawarte są podstawowe informacje na jego temat, takie jak nazwa, data zakończenia, krótki opis, ilość wymagań oraz przypadków użycia. Ponadto dostępne są trzy przyciski: (1) odnośnik do szczegółów projektu, (2) odnośnik do edycji projektu, (3) przycisk generujący aktualną wersję specyfikacji wymagań.

5.3.2 Nowy projekt

Dodanie nowego projektu polega na wypełnieniu prostego formularza zawierającego informacje o nazwie, dacie zakończenia prac, krótkim opisie oraz



Rysunek 5.5: Ekran główny projektu

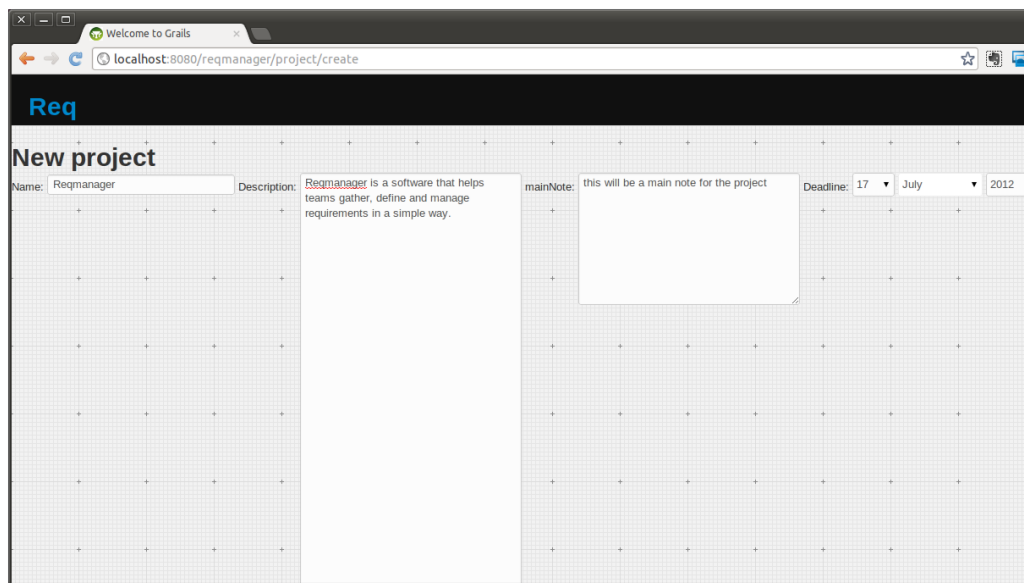
głównej notatce. Rysunek 5.6 przedstawia ekran zakładania projektu.

5.3.3 Pierwsze wymaganie

Do nowo utworzonego projektu, można natychmiast rozpocząć dodawanie wymagań. Widok dodawania pierwszego wymagania załączono na Rysunku 5.7. W tym miejscu, użytkownik ma możliwość podjęcia decyzji dotyczącej ponownego wykorzystania już istniejącego diagramu lub stworzenia nowego. Rysunek 5.7 przedstawia stan aplikacji po stworzeniu nowego diagramu, dodaniu pierwszego aktora, elementu granicy systemu i jednego przypadku użycia. W kolejnych widokach zaprezentowano współdzielenie diagramów.

5.3.4 Przegląd projektu

Rysunek 5.8 przedstawia stan systemu po dodaniu jednego wymagania. Za pośrednictwem tego ekranu, użytkownik ma możliwość kontynuacji pracy nad główną notką projektową (i łatwy zapis tego pola za pomocą przyci-



Rysunek 5.6: Dodawanie nowego projektu

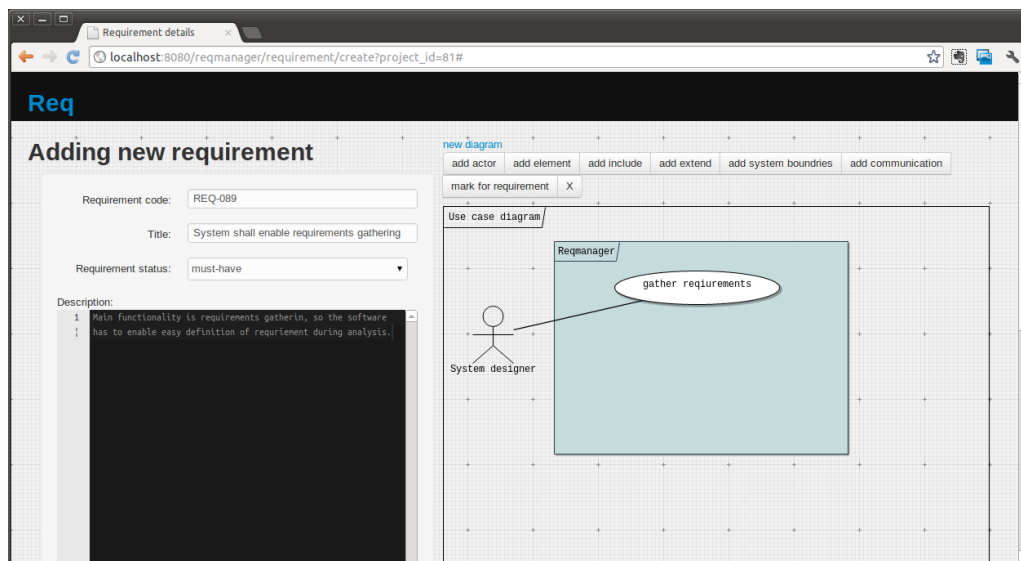
sku „Update” umieszczonego w widocznym, dostępnym miejscu) lub dodania kolejnego wymagania.

5.3.5 Współdzielenie wymagań

Podczas dodawania kolejnych wymagań, użytkownik ma możliwość skorzystania z już istniejącego diagramu, co zaprezentowano na Rysunku 5.9. Zamiast tworzenia nowej instancji diagramu, użytkownik ma możliwość wskazania istniejącego obiektu, który zostanie przypisany do aktualnie definiowanego elementu. Rysunek 5.9 zawiera również przykład zastosowania podstawowej składni Markdown w opisie wymagania.

5.3.6 Łączenie przypadków użycia z wymaganiem

Wracając do widoku przedstawionego na Rysunku 5.7, należy zwrócić uwagę na przycisk „mark for requirement” w pasku narzędzi edytora UML. Wybierając tę opcję, a następnie wskazując wybrany przypadek użycia na diagramie, zostaje wykonana asynchronicznie operacja połączenia use case’u z wymaganiem. Po przejściu do podglądu wymagania, przypisane do niego przypadki

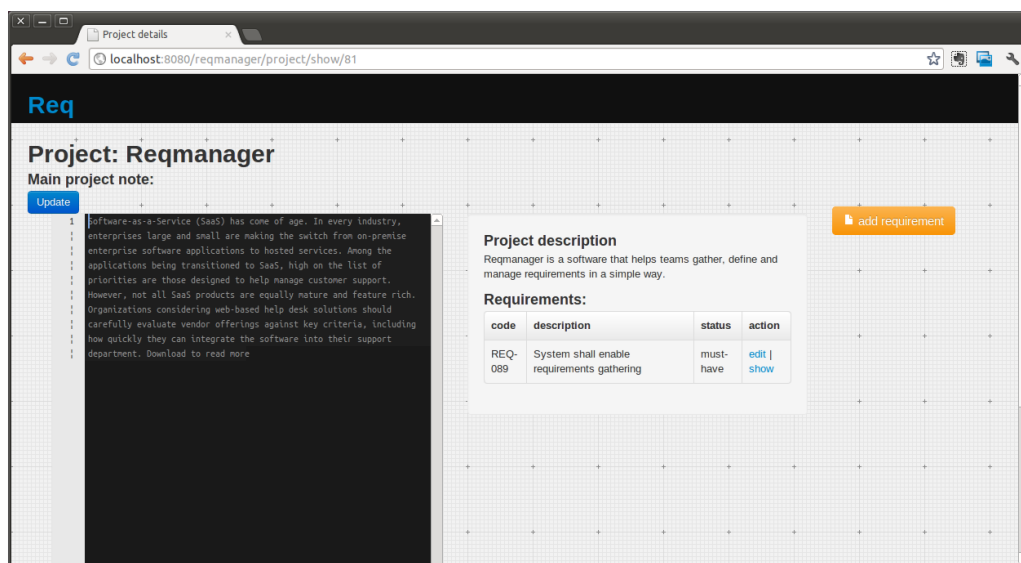


Rysunek 5.7: Definicja wymagania

użycia są zaznaczane kolorem żółtym. Właściwość ta, została zaprezentowana odpowiednio na Rysunkach 5.10 oraz 5.11.

5.3.7 Generowanie specyfikacji

W każdym momencie prac nad analizą, użytkownicy mają możliwość wygenerowania aktualnej specyfikacji wymagań z ekranu głównego aplikacji. Proces polega wyłącznie na wybraniu opcji „Generate specification”. System analizuje dane wprowadzone do bazy i na ich podstawie generuje dokument PDF, który jest przekazywany do przeglądarki klienckiej. Wynik wygenerowania specyfikacji dla projektu powstałego na potrzeby powyższych przykładów pokazano na Rysunku 5.12.

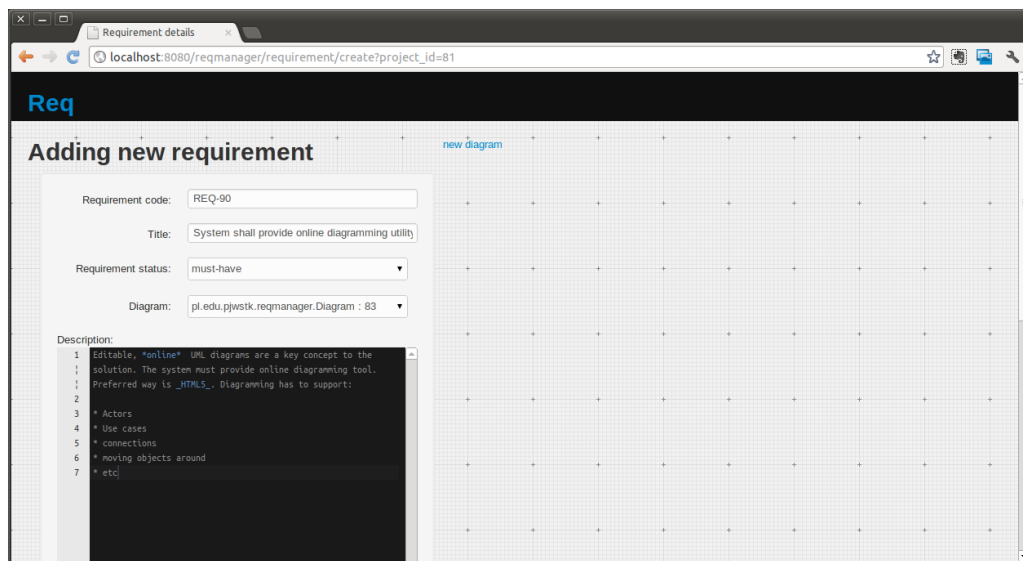


Rysunek 5.8: Stan systemu po dodaniu pierwszego wymagania

5.4 Zalety i wady proponowanego rozwiązania

Proponowane rozwiązanie jest podjęciem próby uproszczenia procesów inżynierii wymagań. Głównym celem stworzenia prototypu było dostarczenie narzędzia umożliwiającego efektywne zbieranie i przetwarzanie wymagań przy jednoczesnym uproszczeniu poziomu jego skomplikowania. Powstał prototyp realizujący te cele, jednak siłą rzeczy, jest to narzędzie przystosowane jedynie do małych projektów. W przeciwieństwie do istniejących rozwiązań, filozofia proponowanego systemu inspirowana była filozofią systemów Unix — programy i aplikacje powinny wykonywać jedną czynność i powinny ją wykonywać możliwie najlepiej. Jednocześnie należy tworzyć aplikacje tak, aby mogły w łatwy sposób współpracować z innymi systemami. Dlatego proponowane rozwiązanie, aby rozwiązywało skutecznie problemy ogólnie pojętej inżynierii oprogramowania, powinno być stosowane komplementarnie z innymi narzędziami, w szczególności systemem zarządzania projektami i zespołem. Od świadomości i dojrzałości organizacji zależy, jak taka infrastruktura zostanie zbudowana oraz jak wybrane narzędzia będą funkcjonowały w organizacji.

Dużą zaletą systemu jest dostępność przez przeglądarkę internetową przy jednoczesnej realizacji funkcjonalności manipulacji obiektami graficznymi.

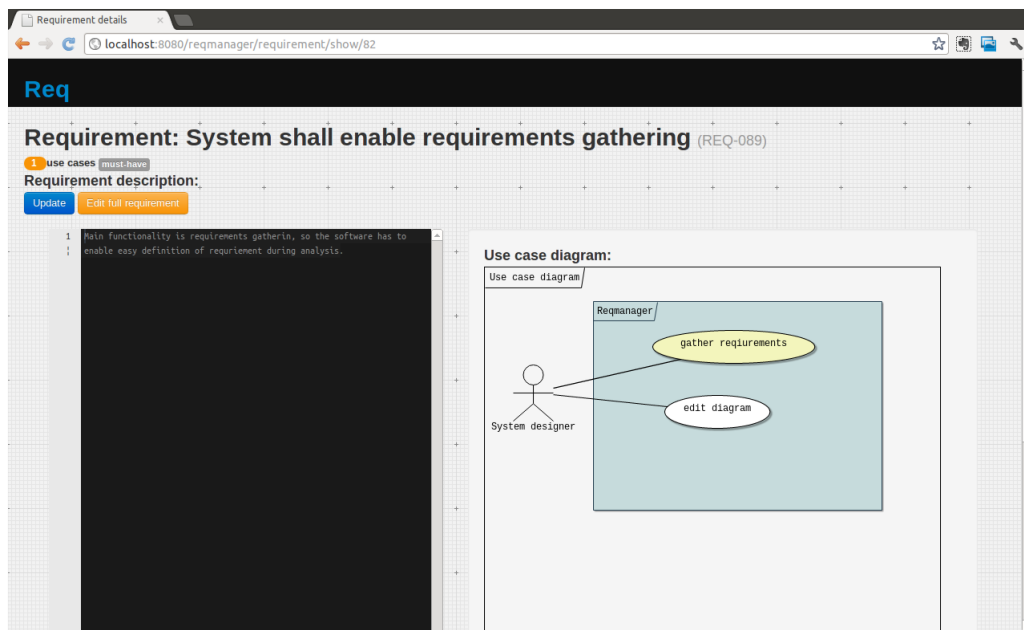


Rysunek 5.9: Współdzielenie diagramu

W szczególności zastosowanie technologii HTML5 daje przewagę w dostępności nad istniejącymi rozwiązaniami, najczęściej implementowanymi w technologii Flash. Rzadko też można spotkać podobny poziom elastyczności aplikacji do zarządzania wymaganiami. Koncepcja systemu celowo nie narzuca wielu ograniczeń związanych ze strukturą wymagań w wierze, że brak restrykcji projektanta wpłynie pozytywnie na proces definiowania i formułowania wymagań.

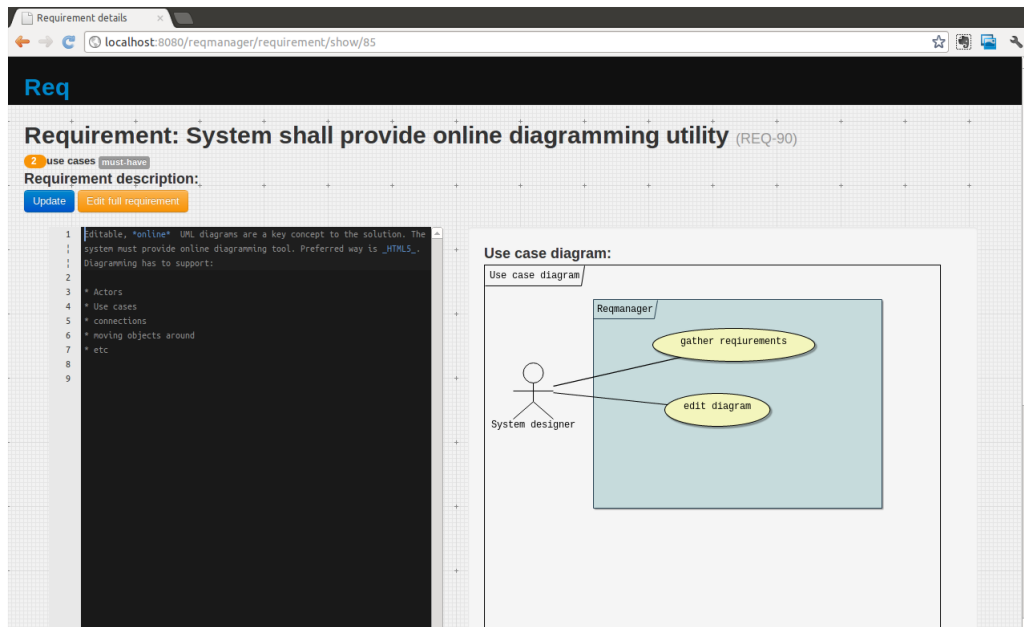
Sam wybór technologii jaką jest Grails, jest zaletą stworzonego prototypu. Zastosowanie tej technologii umożliwia łatwą rozbudowę i skalowanie systemu. System wtyczek platformy Grails sprzyja modułowej naturze aplikacji stworzonych w tej technologii. Z kolei zarządzający całą aplikacją „silnik” oparty na String Framework, Hibernate ORM oraz filozofii „convention over configuration” sprzyjają stosowaniu najlepszych praktyk programistycznych i ponowne wykorzystanie kodu.

Z pewnością istotną wadą jest brak obsługi „śledzenia” wymagań (traceability). Jednak w istniejących rozwiązaniach śledzenie wymagań nie rozwiązuje w pełni problemu. Nadal należy dbać o zgodność kodów i identyfikatorów wymagań w komunikacji z wieloma systemami. W dobrze zorganizowa-

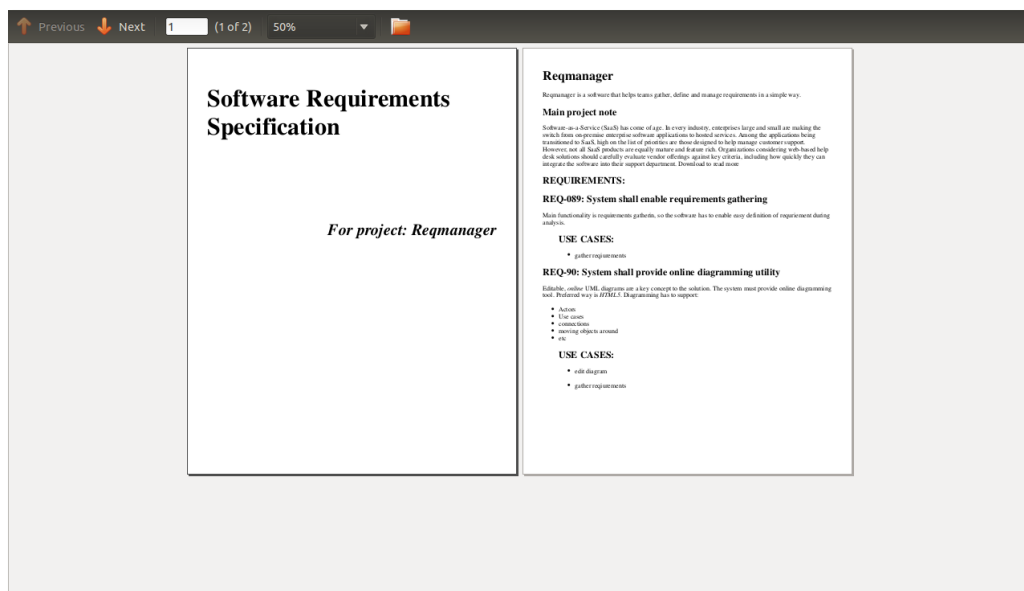


Rysunek 5.10: Jeden przypadek połączony z wymaganiem

nej firmie, te problemy można wyeliminować przez narzucenie odpowiedniej kultury pracy, dbałość o szczegóły i skrupulatność w raportowaniu implementowanych rozwiązań do różnych systemów, jednak dopóki istnieje wiele narzędzi wspomagających różne etapy w procesie wytwórczym, odpowiednia komunikacja i wymiana danych jest jedyną metodą umożliwiającą skuteczne śledzenie wymagań.



Rysunek 5.11: Dwa przypadki użycia połączone z wymaganiem



Rysunek 5.12: Wygenerowana specyfikacja wymagań w PDF

Rozdział 6

Podsumowanie

W poprzednim rozdziale omówiono szczegóły implementacyjne przyjętego rozwiązania oraz krótko omówiono jego wady i zalety z perspektywy autora. Zaprezentowano również przykłady wykorzystania aplikacji poparte odpowiednimi zrzutami ekranu. W tym rozdziale zawarto proponowane ścieżki rozwoju stworzonego oprogramowania oraz podsumowanie całości pracy.

6.1 Plan rozwoju

Proponowane narzędzie wsparcia gromadzenia wymagań, jest jedynie próbą nadania kierunku rozwojowi oprogramowania z tego segmentu. W pełni funkcjonalne narzędzie powinno wspierać obsługę wielu kont użytkowników, śledzenie wymagań, pełną obsługę diagramów (aktorzy, połączenia przypadków użycia, dziedziczenie), wsparcie importu/eksportu danych oraz automatyczny, asynchroniczny zapis obszaru roboczego diagramu.

Na szczególną uwagę zasługuje koncepcja integracji narzędzia z systemem kontroli wersji. Programista pracujący nad kodem źródłowym dotyczącym wybranego wymagania, mógłby w komentarzu zamykającym zadanie umieszczać identyfikator zrealizowanego wymagania. W systemie Reqmanager należałoby zaimplementować podsystem monitorujący wszystkie operacje zapisu do repozytorium, ekstrahować z komentarzy identyfikatory wymagań i odpowiednio oznaczać status ich realizacji na podstawie treści komentarza,

umieszczając przy wymaganiu informacje z systemu kontroli wersji (programista, komentarz, data, status, etc).

Kolejną możliwością rozwoju jest lepsze wsparcie dokumentowania przypadków użycia poprzez wzbogacenie możliwości edycji warunków zajścia a priori i a posteriori. Przykładowo warunki takie mogłyby być zapisywane w postaci kodu źródłowego Groovy lub Java i wykonywane po stronie serwera w odpowiednim momencie. Taka funkcjonalność umożliwiałaby nowatorskie podejście do testowania systemów oraz mogłaby dawać początek kierunku rozwoju bardziej zaawansowanego systemu umożliwiającego dynamiczne generowanie kodu poprzez definicję samych wymagań.

Interesującą koncepcją jest również procesowanie języka naturalnego i automatyczna ekstrakcja wymagań z dokumentów tekstowych. Takie rozwiązanie musiałoby opierać się o zaawansowane biblioteki analizujące tekst oraz implementację inteligentnych algorytmów radzących sobie z ekstrakcją najważniejszych fragmentów tekstu w celu sformułowania logicznego wymagania.

6.2 Zakończenie

Analiza istniejących rozwiązań, doświadczenia wyniesione z licznych, małych projektów komercyjnych, realizowanych przez sektor MŚP oraz badania naukowe w tej dziedzinie, pokazują, że inżynieria wymagań jest w większości organizacji bardzo słabo rozwinięta. Jednak poziom skomplikowania procesów i sprzętu komputerowego będzie rozwijał się w bardzo szybkim tempie jeszcze przez wiele lat, a wraz z nimi, potrzeba tworzenia coraz bardziej złożonego, niezawodnego oprogramowania. Dlatego organizacje chcące przetrwać na tak konkurencyjnym rynku jakim jest wytwarzanie oprogramowania, muszą pracować nad ciągłym usprawnianiem procesów, budową odpowiedniej infrastruktury i świadomości wśród pracowników oraz pracować blisko ze środowiskiem naukowym w poszukiwaniu coraz lepszych rozwiązań inżynierskich. Jedno narzędzie czy koncepcja nie rozwiąże problemów z jakimi mamy do czynienia w inżynierii wymagań. Jest to dziedzina integrująca bardzo zróżnicowane środowiska — świat naukowy i techniczny, ze światem biz-

nesowym, opartym często na bardzo ciężko mierzalnych parametrach. Z tego względu niezwykle trudno jest wyznaczyć jasne granice inżynierii wymagań i odpowiednio sformalizować procesy. Być może w przyszłości, zostaną rozwinięte zaawansowane metody formalne pozyskiwania wymagań lub powstaną systemy procesowania języka naturalnego, które dadzą początek znacznym przemianom i zniwelują zależność od czynnika ludzkiego w procesach twórczych.

Od formalnego wprowadzenia dziedziny inżynierii oprogramowania, jako dziedziny nauki, w roku 1968, na konferencji NATO, wypracowano wiele skutecznych metod wspierających procesy związane z pozyskiwaniem, przetwarzaniem i zarządzaniem wymaganiami. Dojrzałość organizacji tworzących oprogramowanie zależy między innymi od poziomu wiedzy związanej z procesami inżynierii wymagań oraz umiejętności ich skutecznego wdrożenia. Narzędzia zastosowane w tych procesach grają istotną, jednak nie pierwszoplanową rolę. Nadal na dzień dzisiejszy kluczem do sukcesu projektu, oprócz odpowiedniego przygotowania merytorycznego, jest umiejętność zbudowania odpowiedniego zespołu specjalistów z doświadczeniem i wyposażenie ich w odpowiednie mechanizmy pozwalające na swobodne i efektywne działanie. Organizacje nadal starające się przełożyć bezpośrednio dziedzinę wytwarzania oprogramowania na klasyczne zasady inżynierii systemów zdają się nie dostrzegać lub ignorować drastyczne różnice jakie dzielą te dziedziny. Kluczem do sukcesu projektów informatycznych, w pierwszej kolejności, są doskonali, doświadczeni projektanci.



Bibliografia

- [1] J. Arnowitz, M. Arent, and N. Berger. *Effective Prototyping for Software Makers*. Morgan Kaufmann series in interactive technologies. Elsevier Science, 2006. ISBN: 9780080468969.
- [2] S. Easterbrook B. Nuseibeh. *Requirements Engineering: A Roadmap*. 2000.
- [3] K. Beck. *Extreme Programming Explained: Embrace Change*. The XP Series. Addison-Wesley, 2000. ISBN: 9780201616415.
- [4] B Boehm. “A spiral model of software development and enhancement”. In: *SIGSOFT Softw. Eng. Notes* 11.4 (Aug. 1986), pp. 14–24. ISSN: 0163-5948. DOI: 10.1145/12944.12948.
- [5] Barry Boehm. “A view of 20th and 21st century software engineering”. In: *Proceedings of the 28th international conference on Software engineering*. ICSE '06. Shanghai, China: ACM, 2006, pp. 12–29. ISBN: 1-59593-375-1.
- [6] Barry W. Boehm, Terence E. Gray, and Thomas Seewaldt. “Prototyping vs. specifying: A multi-project experiment”. In: *Proceedings of the 7th international conference on Software engineering*. ICSE '84. Orlando, Florida, United States: IEEE Press, 1984, pp. 473–484. ISBN: 0-8186-0528-6.
- [7] F.P. Brooks. *The mythical man-month: essays on software engineering*. Addison-Wesley Pub. Co., 1995. ISBN: 9780201835953. URL: <http://books.google.pl/books?id=fUYPAQAAMAAJ>.

- [8] R. Budde and P. Bacon. *Prototyping: an approach to evolutionary system development*. Springer-Verlag, 1992. ISBN: 9783540543527.
- [9] P. Coad and J. Nicola. *Object-Oriented Programming*. Yourdon Press Computing Series v. 1. Yourdon Press, 1993. URL: <http://books.google.pl/books?id=nd2WQgAACAAJ>.
- [10] Institute of Electrical et al. *IEEE Guide to Software Requirements Specifications*. American national standard. IEEE, 1984. URL: <http://books.google.pl/books?id=F0A1RAAACAAJ>.
- [11] Nathan L. Ensmenger. “Letting the “Computer Boys“ Take Over: Technology and the Politics of Organizational Transformation”. In: *International Review of Social History* 48 (2003), pp. 153–180.
- [12] M. Fowler. *Inversion of Control Containers and the Dependency Injection pattern*. 2004.
- [13] Erich Gamma et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. 1st ed. Addison-Wesley Professional, 1994. ISBN: 0201633612.
- [14] Tools Database Working Group. *INCOSE Requirements Management Tools Survey*. International Council on Systems Engineering. 2010. URL: <http://www.incose.org/productspubs/products/rmsurvey.aspx>.
- [15] Jim Highsmith and Martin Fowler. “The Agile Manifesto”. In: *Software Development Magazine* 9.8 (2001), pp. 29–30.
- [16] USA) IEEE (IEEE. “IEEE Recommended Practice for Software Requirements Specifications: Std 830”. In: (1998). internal report.
- [17] A. Swartz J. Gruber. *Markdown*. 2004. URL: <http://daringfireball.net/projects/markdown/>.
- [18] Philippe Kruchten. *The Rational Unified Process: An Introduction*. 3rd ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003. ISBN: 0321197704.

- [19] NASA. *NASA Software Engineering Requirements*. 2009. URL: http://nodis3.gsfc.nasa.gov/npg_img/N_PR_7150_002A_/N_PR_7150_002A_.pdf.
- [20] U. Nikkula, J. Sajaniemi, and H. Kälviäinen. “A State-of-the-Practice Survey on Requirements Engineering in Small- and Medium-Sized Enterprises”. In: (2000).
- [21] R.S. Pressman. *Software engineering: a practitioner’s approach*. 5th ed. McGraw-Hill Higher Education, 2010. ISBN: 9780073375977. URL: http://books.google.pl/books?id=y4k_AQAAIAAJ.
- [22] Alcides Quispe et al. “Requirements Engineering Practices in Very Small Software Enterprises: A Diagnostic Study”. In: *Proceedings of the 2010 XXIX International Conference of the Chilean Computer Science Society*. SCCC ’10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 81–87.
- [23] Trygve Reenskaug. *Models - Views - Controllers*. 1979. URL: <http://heim.ifi.uio.no/~trygver/1979/mvc-2/1979-12-MVC.pdf>.
- [24] *Ruby on Rails vs Grails vs. Spring ROO vs. Spring App*. Stackoverflow. 2010. URL: <http://stackoverflow.com/questions/2840890/ruby-on-rails-vs-grails-vs-spring-roo-vs-spring-app>.
- [25] I. Sommerville. *Software Engineering*. 8th ed. Pearson, 2006.
- [26] *The CHAOS Report*. Tech. rep. Standish Group International, 1994.
- [27] R. Winston. “Managing the Development of Large Software Systems”. In: *Proceedings ()*, pp. 1–9.