

Spis treści

1	Wstęp	1
1.1	Cel pracy	2
1.2	Zarys koncepcji	3
1.3	Struktura pracy	4
2	Stan faktyczny w inżynierii oprogramowania	5
2.1	Źródła wiedzy	5
3	Istniejące rozwiązania	8
3.1	Analiza rynku systemów wsparcia zbierania wymagań	8
3.1.1	Model SaaS (Software as a Service)	9
3.1.2	Aplikacje desktopowe	17
3.1.3	Platforma IBM .jazz	19
3.2	Problemy z istniejącymi rozwiązaniami	20
4	Podstawy teoretyczne i omówienie koncepcji proponowanego rozwiązania	22
4.1	Inżynieria wymagań	22
4.2	Motywacja podjęcia prac nad nowym rozwiązaniem	29
4.3	Opis proponowanego rozwiązania	30
4.4	System Reqmanager - opis funkcjonalności	31
4.4.1	Obsługa wielu projektów	32
4.4.2	Ogólny przegląd stanu projektu	32
4.4.3	Wsparcie gromadzenia wymagań	32
4.4.4	Uniwersalność formatu danych	35

4.4.5	Zarządzanie zmianą	35
4.4.6	Centralne repozytorium	36
4.4.7	Kolaboracja	36
4.4.8	Generator specyfikacji	36
4.4.9	Praktyczne zastosowanie	36
5	Technologie wykorzystane w implementacji	37
5.1	Zarys technologii	37
5.2	Framework Grails	38
5.2.1	Spring Framework i Hibernate ORM	46
5.3	Pozostałe technologie	47
5.3.1	Postgresql	47
5.3.2	Javascript, jQuery i biblioteka jsUML2	47
5.3.3	System kontroli wersji git i serwer heroku	48
5.3.4	Środowisko programistyczne (linux, vim, tmux)	48
6	Opis prototypu - System „Reqmanager”	49
6.1	Zbieranie wymagań w Reqmanager	50
6.2	Edycja diagramów UML	50
6.3	Kolaboracja	50
6.4	Implementacja	50
6.4.1	Diagramy przypadków użycia	50
6.4.2	Przetwarzanie wymagań (Mapowanie obiektów w JS na encje w bazie danych)	50
6.4.3	Generowanie dokumentacji	50
6.5	Problemy techniczne	50
6.6	Podsumowanie rozwiązania (zalety i wady)	50
7	Podsumowanie	51
7.1	Plan rozwoju	51
7.2	Zakończenie	51

Streszczenie

Brak ujednoliconego środowiska dokumentowania wymagań użytkownika to jeden z głównych problemów małych i średnich firm, w trakcie fazy analizy. Proces dokumentowania wymagań użytkownika zdaje się być zaniedbanym aspektem zarządzania projektami na rynku nowoczesnych aplikacji internetowych. Istniejące rozwiązania są drogie i często przystosowane do dużych projektów. W niniejszej pracy zaprezentowano narzędzie ułatwiające proces zbierania i przetwarzania wymagań użytkownika z wykorzystaniem technologii internetowych, w niewielkich projektach.

W niniejszej pracy zaprezentowano rozwiązanie dla małych i średnich projektów, pozwalające skupić się na najważniejszych aspektach formułowania i dokumentowania wymagań. Powstały system umożliwia wprowadzanie i kategoryzowanie wymagań zarówno w formie graficznej, jak i tekstowej. Na podstawie wprowadzonych danych, aplikacja umożliwia automatyczną generację dokumentu specyfikacji wymagań.

Dzięki zastosowaniu powszechnie dostępnych technologii internetowych, proponowane narzędzie może zostać uruchomione w wielu różnych środowiskach. Wykorzystanie technologii HTML5 oraz języka javascript umożliwiło stworzenie narzędzia wspomagającego graficzne modelowanie przypadków użycia.

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 daje nowe, szerokie możliwości przeglądarkom internetowym. Możliwości, które były dotychczas zarezerwowane głównie dla technologii Flash (oraz z trochę gorszymi wynikami MS Silverlight, JavaFX, itp.) stają się wspierane natywnie, w każdej popularnej przeglądarce internetowej.

Z wymaganiami użytkownika stykamy się praktycznie od samego początku pracy nad projektem informatycznym. Z tego powodu potrzebne są skuteczne narzędzia gromadzenia i przetwarzania wymagań użytkownika oraz komunikowania ich wszystkim członkom zespołu. Dzięki wyżej wspomnianej technologii HTML5, istnieje możliwość usprawnienia procesu dokumentowania wymagań przy jednoczesnym zachowaniu wysokiej dostępności aplikacji, włączając do niego nowe narzędzia oparte na metodach graficznych.

Rynek nowych technologii roi się dzisiaj od aplikacji rozwiązujących rozmaite problemy, od prostych list zakupów, po zaawansowane narzędzia wspierające zarządzanie projektami. Pomimo istnienia wielu aplikacji skupiających się na procesach fazy analizy, bardzo ciężko znaleźć rozwiązanie dostosowane do potrzeb i realiów prowadzenia projektów w małych i średnich

przedsiębiorstwach.

W niniejszej pracy zostanie zaprezentowana koncepcja aplikacji wspierającej zarządzanie wymaganiami użytkownika, stworzona z myślą o rzeczywistych problemach jakie wiążą się z tym etapem projektu informatycznego. Zaproponowany system ma za zadanie ułatwić zespołom projektowym komunikację i dostęp do wiedzy przy jednoczesnym skupieniu uwagi na najważniejszych aspektach definiowania i przetwarzania wymagań.

1.1 Cel pracy

W małych i średnich przedsiębiorstwach, proces gromadzenia i dokumentowania wymagań użytkowników jest najczęściej słabo sformalizowany. Wymagania 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 wymagają instalacji oprogramowania po stronie klienta [!ref] a rozwiązania dostępne online, w formule SaaS (Software As A Service) [!ref] często powielają tylko funkcjonalności potężnych (i drogich) systemów takich jak IBM DOORS [!ref] czy IBM RequisitePro [!ref]. Ponadto niezwykle trudno jest znaleźć rozwiązanie darmowe lub posiadające ogólną dostępną wersję demonstracyjną.

Celem niniejszej pracy jest konstrukcja prototypu systemu usprawniającego zbieranie i przetwarzanie wymagań użytkownika w postaci tekstowej oraz graficznej, dostarczając narzędzi edycji tekstu oraz graficznego modelowania przypadków użycia. W efekcie, użytkownik, po wprowadzeniu początkowych wymagań, ma możliwość automatycznego wygenerowania dokumentu SRS (Software Requirement Specification) [!ref].

Podejście zaprezentowane w tej pracy, opiera się na założeniach, że nowoczesne oprogramowanie wspierające zarządzanie wymaganiami, powinno m.in.: stanowić centralne repozytorium wiedzy o wymaganiach; być łatwo dostępne w jak największej ilości różnych środowisk; umożliwiać łatwą ko-

laborację 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 zbierania 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 wymagań pozostają rozproszone i niekompatybilne.

Na potrzeby niniejszej pracy, został stworzony prototyp systemu pozwalający w łatwy sposób dokumentować gromadzone wymagania użytkownika. W powstałym systemie, głównymi narzędziami definiowania wymagań są pliki tekstowe oraz diagramy przypadków użycia. Osoby odpowiedzialne w projekcie za zarządzanie wymaganiami, otrzymują proste w obsłudze narzędzia, pozwalające na skupienie się nad sednem problemu, który poszczególne wymagania ma na celu rozwiązać. Zaimplementowano system zarządzania projektem, w którym użytkownik ma możliwość zdefiniowania podstawowych parametrów takich jak nazwa projektu, planowany czas zakończenia oraz ogólny opis, zawierający kluczowe informacje o projekcie na etapie analizy. Korzystając z narzędzia definiowania wymagań w projekcie, użytkownik ma do dyspozycji edytor tekstowy, w którym dokumentuje zidentyfikowane wymagania użytkownika. Dzięki wykorzystaniu prostego języka znaczników (Markdown [!ref]), użytkownik skupia się na logicznej strukturze opisu wymagania. Brak narzędzi „WYSIWYG” znanych ze standardowych edytorów tekstu jest zabiegiem celowym - ma sprawiać, że użytkownik nie jest rozpraszany potrzebą myślenia o graficznej reprezentacji tekstu opisującego problem, przy jednoczesnym zachowaniu czytelnej struktury dokumentu. System załączników i komenatrzy, 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 przez użytkowników 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 wymagań użytkownika. Dlatego na każdym etapie fazy analizy istnieje możliwość automatycznego wygenerowania specyfikacji wymagań na podstawie danych wprowadzonych do systemu. Takie podejście sprzyja modelowi iteracyjnemu tworzenia specyfikacji i umożliwia prezentację prac nad dokumentem 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

W rozdziale 2 zostanie przedstawiony stan faktyczny związany z zastosowaniem metod pozyskiwania, gromadzenia i zarządzania wymaganiami w małych i średnich przedsiębiorstwach. Rozdział 3 przedstawia dostępne na rynku narzędzia wspierające zarządzanie wymaganiami. Rozdział 4 jest dokładnym opisem dziedziny problemu oraz koncepcji zaimplementowanego prototypu. W rozdziale 5 zaprezentowano najistotniejsze technologie, jakie wykorzystano podczas pracy nad prototypem oraz krótkie omówienie środowiska programistycznego w jakim powstał przedmiot pracy. Rozdział 6 stanowi opis zaimplementowanego prototypu. W rozdziale 7 zawarto podsumowanie wyników pracy oraz propozycje kierunków dalszego rozwoju.

Rozdział 2

Stan faktyczny w inżynierii oprogramowania

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.

Ten rozdział zadedykowany jest analizie stanu faktycznego w zakresie inżynierii oprogramowania w różnych organizacjach.

2.1 Źródła wiedzy

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 dostępne badania wskazują na istotne problemy z wprowadzaniem teorii w życie [5, 19, 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 problemami inżynierii wymagań na poziomie korporacyjnym, poprzez wdrażanie, optymalizację i automatyzację procesów IW w małych i średnich przedsiębiorstwach [19].

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” [21].

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 z powodzeniem 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 inżynierii wymagań w organizacji i 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. Na organizacji wytwarzającej oprogramowanie ciąży odpowiedzialność usprawniania procesów inżynierii wymagań, 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.

Rozdział 3

Istniejące rozwiązania

W tym rozdziale zostanie przeprowadzona analiza rynku oprogramowania wspierającego fazę analizy. Druga część rozdziału zajmie się identyfikacją najistotniejszych problemów jakie posiadają istniejące rozwiązania.

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.

3.1 Analiza rynku systemów wsparcia zbierania wymagań

Jak wspomniano we wstępie do rozdziału, na rynku nie brakuje systemów wsparcia procesu zbierania wymagań. 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 komputerze klienta nadal cieszą się dużą popularnością. Należy jednak zaznaczyć, iż w 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.

3.1.1 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 aplikacji, 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 aplikacje wspierające proces rekrutacji (humanway.com, recruiterbox.com, jobvite.com). Na uwagę zasługują także aplikacje wspierające rezerwacje hoteli, pensjonatów i niezależnych pokoi i mieszkań jak airbnb.com, booking.com czy b&b.com.

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

Powodem takiego stanu rzeczy jest szeroko pojęta popularyzacja internetu oraz sukcesywne przenoszenie się do sieci firm tworzących oprogramowanie. Dystrybucja oprogramowania 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 oprogramowania lub takich dostawców usług jak Amazon Web Services czy Heroku, daje nieograniczone możliwości wprowadzania nowych funkcjonalności, poprawek i skalowania systemu. Z kolei monitorowanie

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 l...	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 3.1: Gatherspace.com - business requirements

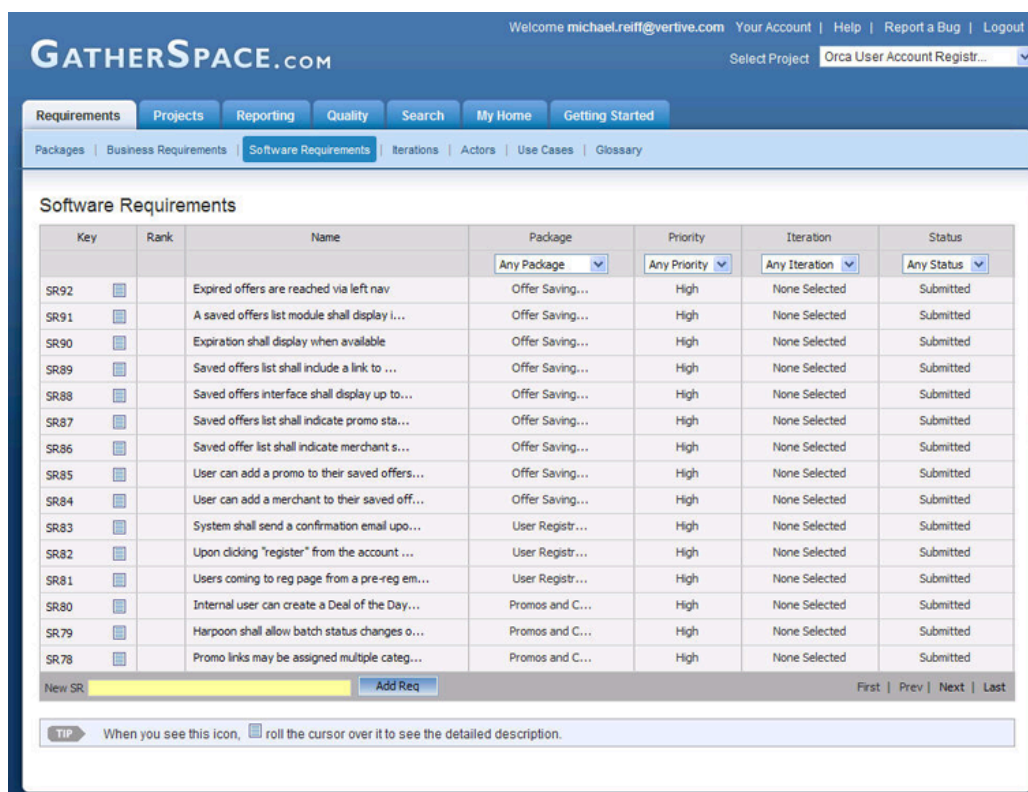
zachowań klientów pozwala niemalże w czasie rzeczywistym odpowiadać na potrzeby użytkowników poprzez modyfikację i wdrażanie nowych funkcjonalności.

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 zostaną przedstawione funkcjonalności systemów Gatherspace oraz Tracecloud.

Gatherspace

Gatherspace dzieli wymagania na biznesowe (business requirements) oraz funkcjonalne (software requirements). Dostępne są osobne widoki dla obu



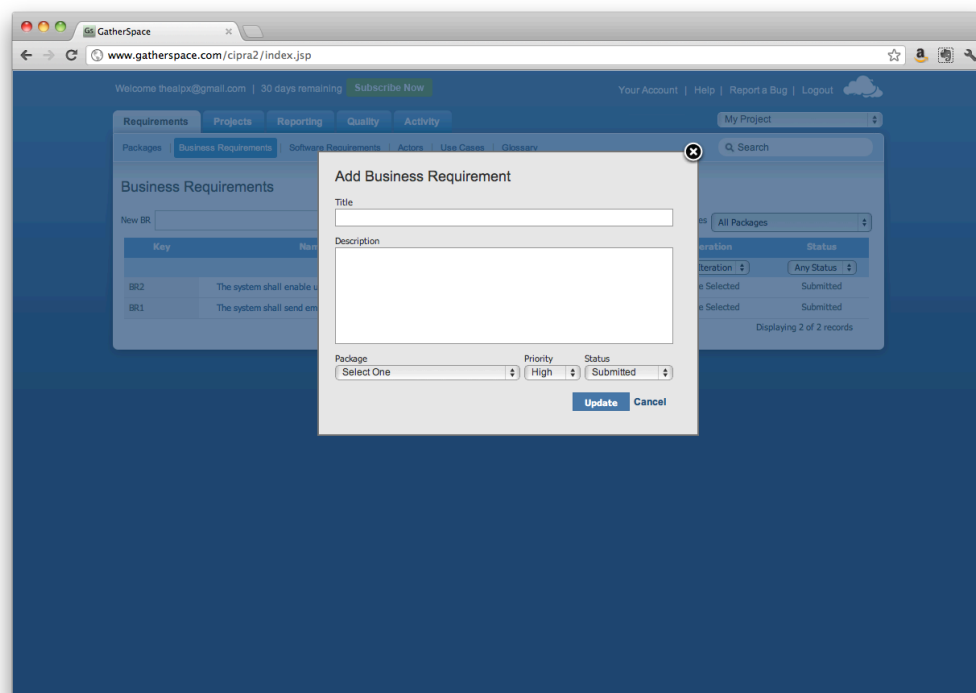
Rysunek 3.2: Gatherspace.com - software requirements

typów wymagań (rys. 3.1, 3.2). Wymagania można katalogować w definio-
wane przez użytkownika pakiety, podobnie jak pliki w katalogach, w syste-
mach operacyjnych.

Na Rysunku 3.2 przedstawiono widok wymagań funkcjonalnych. Doda-
wanie wymagań przebiega w prosty sposób. Po wpisaniu tytułu wymagania
i zatwierdzeniu pola formularza, element zostaje dodany do listy bez przeła-
dowania strony.

Wymaganie może zostać dodane na dwa sposoby - podając jedynie ty-
tuł lub wybierając opcję „Add with detail”. Wówczas pojawia się „popup”
umożliwiający dodanie zarówno treści jak i opisu wymagania (patrz Rys.
3.3).

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 przy-



Rysunek 3.3: Gatherspace.com - dodawanie wymagania ze szczegółami

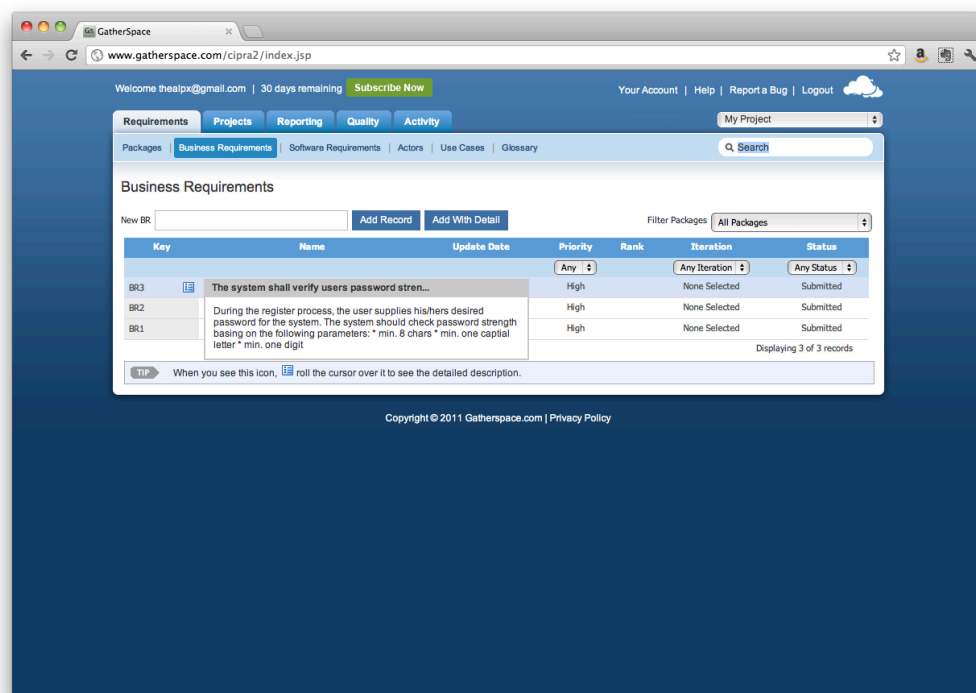
datna funkcjonalność, pozwalająca szybko zapoznać się pobieżnie z danym wymaganiem. Rysunek 3.4 zawiera zrzut ekranu prezentujący tę funkcjonalność.

Widok szczegółów wymagania jest dość prosty i przejrzysty (Rys. 3.5). Do dyspozycji użytkownika jest formularz zawierający pola z parametrami wymagania, 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. 3.6).

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

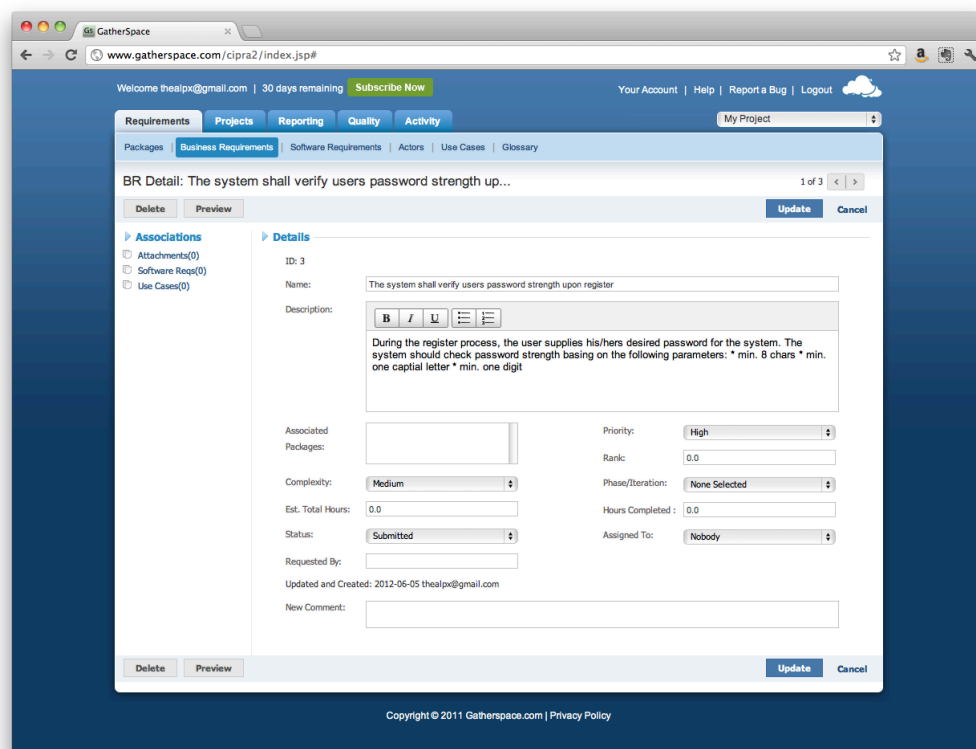
Zarówno przypadki użycia jak i wymagania mogą zostać wyświetlone w



Rysunek 3.4: GatherSpace.com - tooltip ze szczegółami wymagania

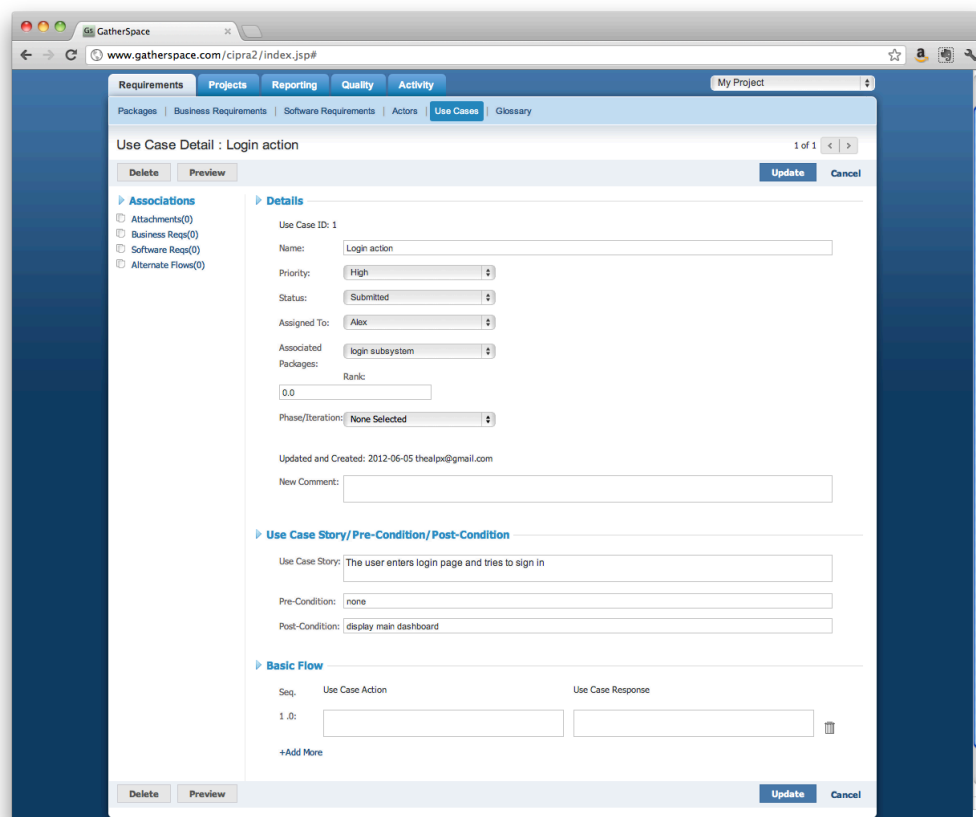
formie przyjaznej do druku. Dla przykładu zamieszczono zrzut ekranu obrazujący jeden przypadek użycia przygotowany do druku na Rys. 3.7

Dostępne w systemie raporty umożliwiają wyświetlenie i wydruk raportów dotyczących wymagań funkcjonalnych i niefunkcjonalnych, raport przypadków użycia oraz raport identyfikowalności (traceability report), umożliwiając przeanalizowanie związków pomiędzy wymaganiami a przypadkami użycia (patrz Rysunek 3.8).



Rysunek 3.5: 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, 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 3.6: Gatherspace.com - widok szczegółów wymagania

Tracecloud

Jak widać na Rys. 3.9 Tracecloud jest znacznie gorzej przygotowany od strony interfejsu użytkownika. Mimo tego jest znacznie bardziej skomplikowanym systemem, udostępniającym znacznie więcej funkcjonalności niż Gatherspace.

Główny przegląd projektu zawiera tabelę z wyświetlonymi metrykami projektu. Można z niej odczytać ilość zaimplementowanych wymagań, ilościowe wyniki testów, przedział 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 reprezentacji stanu projektu jest bardzo niska.

Oprócz ilościowego przeglądu stanu projektu, użytkownik ma do dyspo-

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
------	--

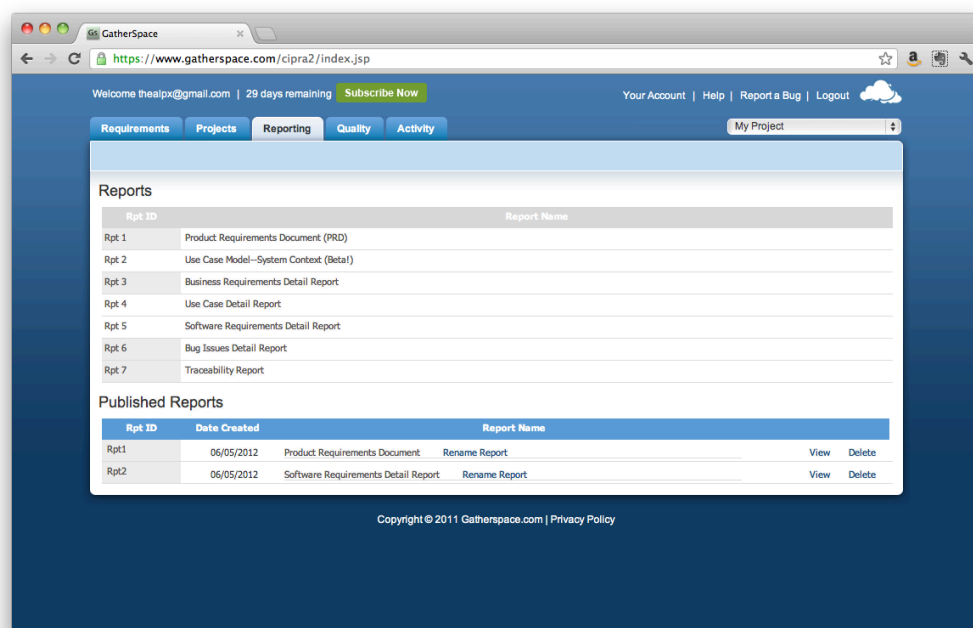
Rysunek 3.7: Gatherspace.com - wymaganie przygotowane do druku

zycji menu i odnośniki do najważniejszych modułów systemu, taki jak wymagania biznesowe, funkcjonalne, scenariusze testowe i błędy.

Przegląd wymagań opiera się o wykres ze statystykami oraz prostą listę (Rys. 3.11), 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. 3.12).

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 użytkownika systemu w za-



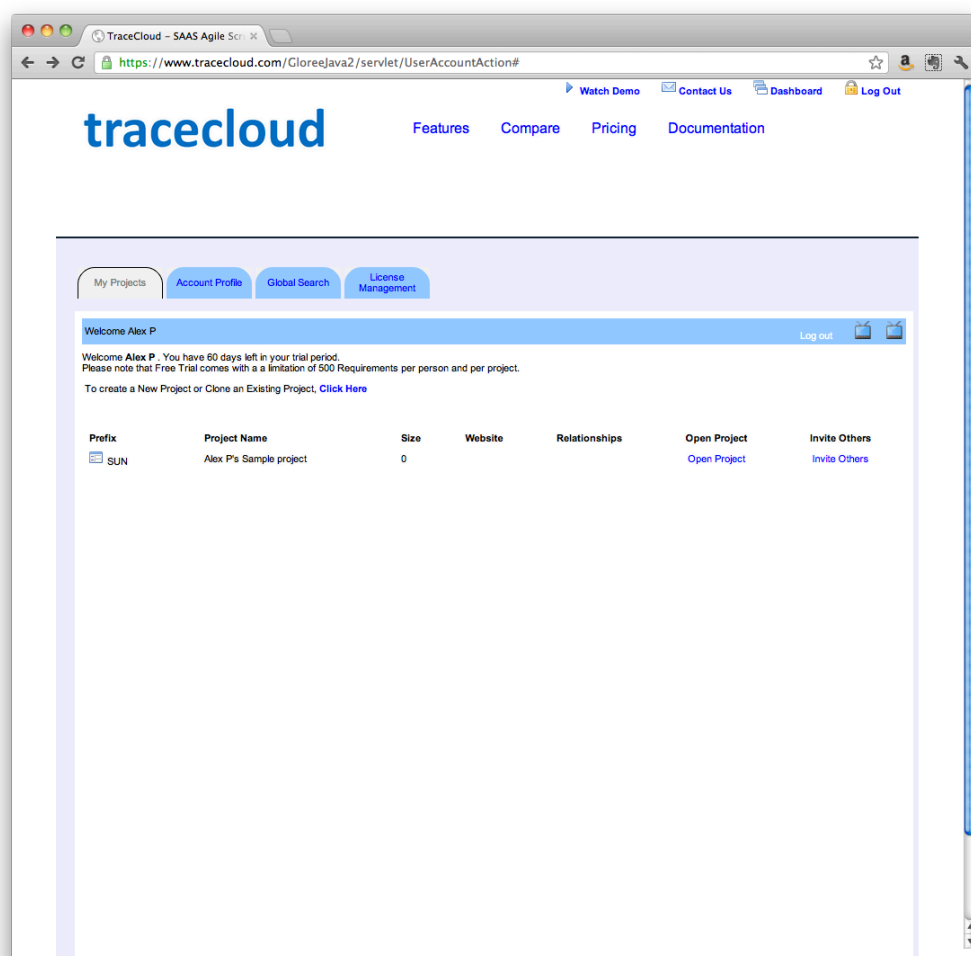
Rysunek 3.8: Gatherspace.com - dostępne raporty

leżności od maksymalnej ilości obsługiwanych wymagań. Najtańsza licencja przewiduje obsługę do 5000 wymagań, podczas gdy wersja najdroższa - do 10tyś.

3.1.2 Aplikacje desktopowe

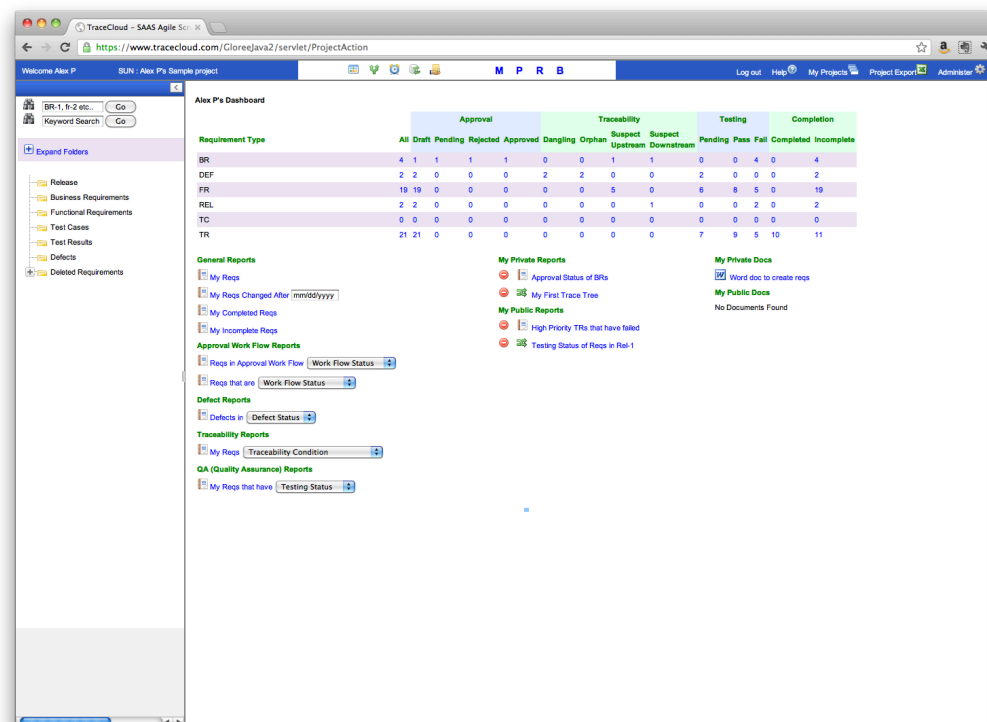
Model SaaS, mimo wszystkich swoich zalet, posiada również wady. Głównym powodem kontrowersji jest konieczność powierzenia danych biznesowych firmie udostępniającej narzędzie na swoich serwerach. Dla dużych korporacji, pilnie strzegących swoich tajemnic handlowych, takie rozwiązanie, może być nie do zaakceptowania 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



Rysunek 3.9: Tracecloud - widok listy projektów

temu, nadal istnieje zapotrzebowanie na oprogramowanie instalowane na lokalnym dysku użytkownika. Przykładami takich aplikacji są m.in. IBM DOORS oraz IBM Rational Requirements Composer.



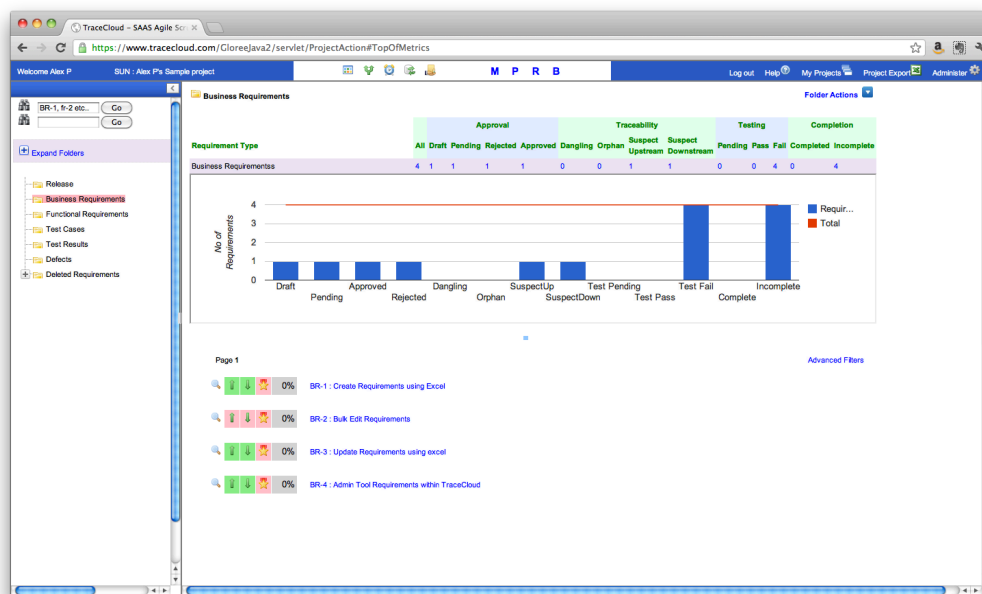
Rysunek 3.10: Tracecloud - ekran główny

RequisitePRO

DOORS

3.1.3 Platforma IBM .jazz

Platforma IBM jazz zasługuje na osobną sekcję, ponieważ jest zintegrowanym, kompleksowym zestawem narzędzi i aplikacji dla przedsiębiorstw tworzących oprogramowanie. Centrum zarządzania platformą jazz jest serwer 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).



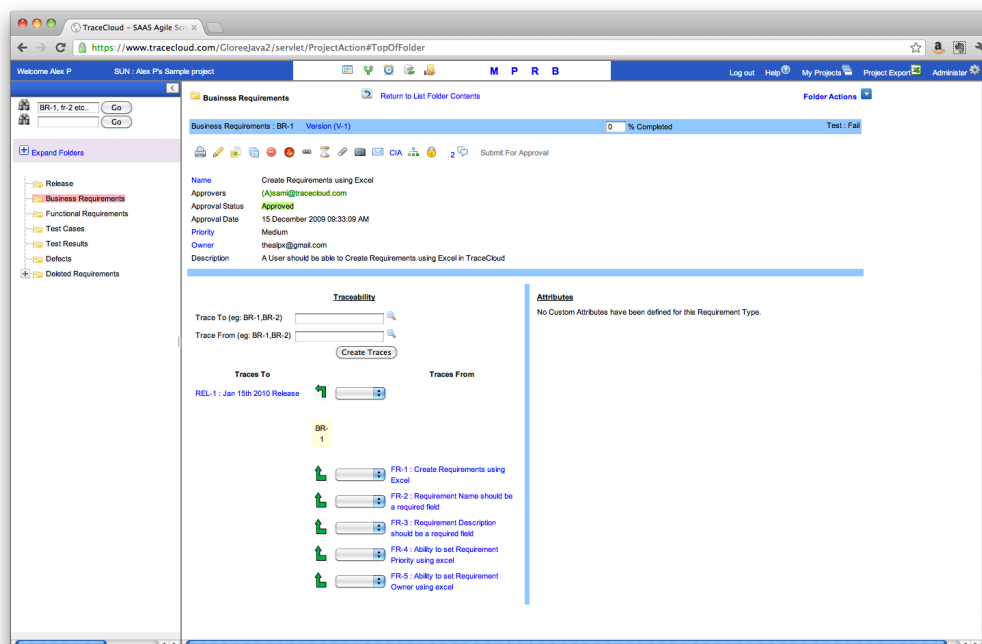
Rysunek 3.11: Tracecloud - przegląd wymagań

3.2 Problemy z istniejącymi rozwiązaniami

W poprzedniej sekcji omówiono wybrane narzędzia z wielu dostępnych na rynku aplikacji wspomagających zarządzanie wymaganiami. !! Analiza zastosowanych rozwiązań zdecydowanie prowokuje do przemyśleń i nasuwa na myśl serię pytań związanych motywacjami jakimi kierują się twórcy tego typu oprogramowania.

Analizując istniejące rozwiązania, nie można oprzeć się wrażeniu, że wszystkie przystosowane są do dużych, korporacyjnych projektów. Większość aplikacji powieli rozwiązania znane i sprawdzone funkcjonalności. W konsekwencji trudnym zadaniem jest znalezienie wyróżniających się elementów wśród oferowanych możliwości tych produktów.

Proces definiowania wymagań w niewielkich projektach jest w dużej mierze procesem twórczym. Często pojawienie się nowego wymagania jest inicjowane z kilku heterogenicznych źródeł. Nierzadko zdarza się również, że to samo wymaganie jest różnie komunikowane przez wiele źródeł. W niewielkich projektach, gdzie często jedna osoba, lub mały zespół odpowiedzialny jest



Rysunek 3.12: Tracecloud - szczegóły wymagania

za specyfikację wymagań, wymagania przybierają postać wiadomości email, rozmów telefonicznych, notatek ze spotkań i formalnych dokumentów. Zadaniem osoby lub zespołu odpowiedzialnego za specyfikację wymagań, jest przefiltrowanie częściowych informacji oraz ekstrakcja i udokumentowanie wymagań. Żadne narzędzie (przynajmniej na razie) nie zastąpi skutecznie pracy człowieka nad wyłuskaniem wszystkich wymagań.

!!!!Wymagania trzymane są w dokumentach procesorów tekstowych - trudno dostępne, ciężko analizować!!!

Rozdział 4

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, zostaną opisane procesy inżynierii wymagań od strony teoretycznej. Zostanie również zwrócona uwaga na niezbędne elementy nowoczesnego systemu wspierającego pracę z wymaganiami w małych i średnich przedsiębiorstwach. Na koniec rozdziału, przedstawiona zostanie koncepcja nowego narzędzia, będącego przedmiotem tej pracy.

4.1 Inżynieria wymagań

Inżynieria wymagań (IW) jest dziedziną, na którą składają się wszelkie działania związane z odkrywaniem (elicitation), analizą (analysis), weryfikacją (validation) oraz zarządzaniem wymaganiami [24]. 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 systemów informatycznych, 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 [18]. 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].

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.

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 [20].

Procesy w inżynierii wymagań

Jak podaje Sommerville [24], 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) [17].

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. W szczególności 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 powróci 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 wymagania, konieczna jest weryfikacja spójności specyfikacji pod względem konfliktów. Zapewnienie bezkonfliktowości polega na takim sformułowaniu wymagań w specyfikacji, aby żadne nie były ze sobą w sprzeczności. Kompletność jest weryfikacją dokumentu wymagań z perspektywy realizacji wszystkich porządkanych funkcji. Wszystkie zdefiniowane wymagania muszą być realizowalne 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 realizowalne 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 Boehm et al „Prototyping vs. Specifying” [6] dowodzą, że jedną ze skuteczniejszych metod prewencji przed definiowa-

niem 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. Klient musi 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 prototypowania, jednak szczegółowe aspekty tego podejścia wykraczają poza zakres tej pracy. Osoby zainteresowane metodami prototypowania znajdą 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.

4.2 Motywacja podjęcia prac nad nowym rozwiązaniem

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 w zakresie inżynierii oprogramowania, 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 analizę wymagań, 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 wymaganiami 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 porządanego efektu. Jest to problem całej branży - zarówno kadry zarządzającej, analityków, programistów, klientów i użytkowników. Problemy z zarządzaniem wymaganiami, powodują, spowolnienie in-

nowacji. Gdyby firmy zaczęły umiejętnie zarządzać wymaganiami i zmianą, więcej projektów kończyłoby się sukcesem, co pozytywnie wpłynęłoby na prędkość rozwoju oprogramowania. Ewidentnie istnieje niezagospodarowana luka na rynku w tym segmencie.

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 wytwarzaniem oprogramowania. 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.

4.3 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.

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

W trakcie prac nad koncepcją narzędzia postawiono następujące cele:

- Silna koncentracja na praktycznym zastosowaniu systemu wsparcia zarządzania 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 projekami 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.

4.4 System Reqmanager - opis funkcjonalności

Poniżej opisano poszczególne funkcjonalności systemu i przedstawiono mechanizmy, narzędzia oraz techniczne rozwiązania które posłużyły do imple-

mentacji poszczególnych założeń systemu.

4.4.1 Obsługa wielu projektów

System umożliwia tworzenie wielu projektów. Formularz dodawania projektu składa się 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.

4.4.2 Ogólny przegląd stanu projektu

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

4.4.3 Wsparcie gromadzenia wymagań

Praktyczne narzędzie do zastosowania w procesie gromadzenia wymagań musi wspierać najwcześniejszą fazę - powstawania wymagań. W dynamicznych projektach, wymagania pojawiają się niespodziewanie, w różnych sytuacjach i pochodzą z wielu różnych źródeł. Narzędzie musi wspierać prostą w obsłudze opcję dodawania wymagań 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. Ce-

lem 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 łączników.

Edytor tekstowy

Formatowanie tekstu w języku znaczników Markdown [16] 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 konsekwencie drugoplanowa. Przykład składni dokumentu Markdown zaprezentowano na Rys. X [!ref].

```
# Brief projektu
```

```
Ten dokument zawiera tzw "brief" projektu - wstępny opis systemu, jaki mamy zamiar zbudować.
```

```
## System w szczególności musi uwzględniać następujące funkcjonalności:
```

```
* logowanie użytkownika
* dodawanie i zarządzanie projektami
* dodawanie i zarządzanie wymaganiami
* _tworzenie diagramów przypadków użycia_
```

Po konwersji przez Markdown do języka HTML, otrzymamy następujący wynik:

```
<h1>Brief projektu</h1>
```

```
<p>Ten dokument zawiera tzw "brief" projektu - wstępny opis systemu, jak
```

<h2>System w szczególności musi uwzględniać następujące funkcjonalności:

- logowanie użytkownika
- dodawanie i zarządzanie projektami
- dodawanie i zarządzanie wymaganiami
- tworzenie diagramów przypadków użycia

Dzięki temu, w łatwy sposób 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 będącego integralną częścią frameworka Grails. Jednocześnie zostanie zachowana odpowiednia relacja między utworzonym obiektem klasy UseCase, a wymaganiem. Mapowanie elementów utworzonych na diagramie przebiega dzięki możliwości serializacji diagramu do formatu XML. Szczegółowo autor opisuje ten mechanizm w rozdziale dotyczącym rozwiązań implementacyjnych [!ref]

W przyszłości, istnieje możliwość rozszerzenia funkcjonalności edytora o kolejne rodzaje diagramów UML. Ponadto, w trakcie ewentualnej rozbu-

dowy 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.

4.4.4 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.

4.4.5 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.

4.4.6 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.

4.4.7 Kolaboracja

Możliwość komentowania wymagań została zaimplementowana z myślą o umożliwieniu kolaboracji nad wymaganiami członkom zespołu.

4.4.8 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 nią przypadki użycia.

4.4.9 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 5

Technologie wykorzystane w implementacji

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

5.1 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 - predefiniowany 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 [!ref].

Edytor tekstu oparty jest o bibliotekę javascript Ace Editor. Jest to wiodąca biblioteka, wykorzystywana m. in. w eksperymencie Cloud9 IDE [!ref] -

webowym, zintegrowanym środowisku programistycznym zapoczątkowanym przez zespół Mozilli, pod nazwą Mozilla Skywriter [!ref].

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.

5.2 Framework Grails

Framework Grails jest stosunkowo młodą technologią opartą na sprawdzonych rozwiązaniach. Firma która zajmuje się rozwojem tej technologii jest tą samą organizacją, odpowiedzialną za stworzenie Spring Framework - jednego 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, poniżej 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.

```
// Groovy:
for (i in 1..100) {
    println "${i%3?'':'Fizz'}${i%5?'':'Buzz'}" ?: i
}

//Java:
public class FizzBuzz{
    public static void main(String[] args){
        for(int i= 1; i <= 100; i++){
            if(i % 15 == 0){
                System.out.println("FizzBuzz");
            } else if(i % 3 == 0){
                System.out.println("Fizz");
            } else if(i % 5 == 0){
                System.out.println("Buzz");
            } else{
                System.out.println(i);
            }
        }
    }
}
```

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 [23].

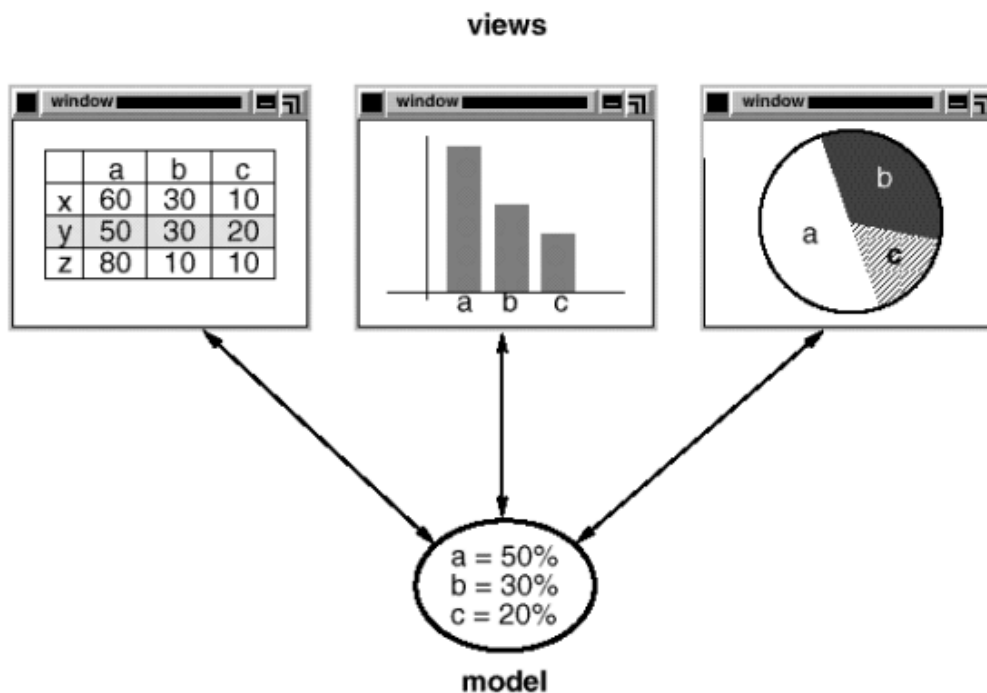
Grails jest silnie oparte o paradygmat „convention over configuration” - architekturze zakładającej minimalizację potrzeby konfiguracji aplikacji przez programistę na rzecz przyjętych konwencji, do których należy się stosować, aby wydajnie dostarczać działające rozwiązania. Podejście to dotyczy wszelkich 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



Rysunek 5.1: Model-View-Controller - różna reprezentacja graficzna tych samych danych. Źródło: [13].

i wyświetla je w odpowiedni sposób, często uwydatniając pewne elementy modelu, inne z kolei odpowiednio ukrywając. Funkcja ta czyni z widoku pewnego 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 [22].

Praca z Grails

Struktura katalogów nowo utworzonej aplikacji Grails wygląda następująco:

```
+ grails-app
+ conf          ---> lokalizacja artefaktów konfiguracji
+ hibernate     ---> opcjonalne pliki konfiguracyjne hibernate
```

- + spring ---> opcjonalne pliki konfiguracyjne spring
- + controllers ---> kontrolery aplikacji
- + domain ---> klasy reprezentujące modele
- + i18n ---> zlokalizowane komunikaty i18n
- + services ---> warstwa serwisowa
- + taglib ---> biblioteki znaczników
- + util ---> klasy pomocnicze special utility classes
- + views ---> lokalizacja widoków
 - + layouts ---> lokalizacja
- + lib
- + scripts ---> scripts
- + src
- + groovy ---> optional; location for Groovy source files
(of types other than those in grails-app/*)
- + java ---> optional; location for Java source files
- + test ---> generated test classes
- + web-app
- + WEB-INF

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 stworzenia klasy Groovy reprezentującej model (na przykładzie implementacji z systemu Reqmanager):

```
package pl.edu.pjwstk.reqmanager.examples
class Project {
    String name
    String description
    String mainNote
}
```

```

    java.sql.Timestamp timestamp
    java.util.Date deadline

    static hasMany = [requirements:Requirement]
}

```

Jeśli nie skonfigurujemy aplikacji w niestandardowy sposób, po stworzeniu powyższej klasy Groovy w katalogu `grails-app/domain`, Framework Grails przy pomocy Hibernate utworzy następującą strukturę w bazie danych:

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)
```

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 danych po stronie bazy.

Na szczególną uwagę zasługuje konstrukcja w klasie `Project`:

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

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:

```
//pobiera z bazy projekt o id = 1
def project = Project.get(1)

//zwraca listę wymagań podłączonych do projektu
def reqs = project.requirements
```

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 usatwione w tabeli asocjacyjnej, zwracanej przez kontroler (w tym przypadku, zmienna „project” zawierająca referencję do listy wszystkich projektów w systemie).

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

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

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

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, będąca zdefiniowanym blokiem kodu, który można przekazać jako parametr innej metodzie lub funkcji. Poniższy przykład wyjaśnia zasadę na jakiej działają domknięcia w Groovy:

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

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 jest wykonana i wykonuje blok kodu na kolejnych elementach kolekcji przekazany jej jako parametr, następnie zwracając listę zmodyfikowanych przez domknięcie elementów.

5.2.1 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]. ***** tu trochę o IoC ***** . 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. Jest to metoda w pewnym stopniu rozwiązująca problem niezgodności impedancji [25]

5.3 Pozostałe technologie

5.3.1 Postgresql

Trwałość danych w aplikacjach internetowych standardowo realizowana jest przez bazę danych. Na potrzeby prezentowanego prototypu, wykorzystano bazę danych Postgresql. Jest to obiektowo-relacyjna baza danych udostępniona na licencji Wolnego i Otwartego Oprogramowania.

Baza danych PostgreSQL implementuje znaczną część standardu SQL:2008 [!ref - <http://www.postgresql.org/docs/9.1/static/features.html>]

5.3.2 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 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”....

5.3.3 System kontroli wersji git i serwer heroku

5.3.4 Środowisko programistyczne (linux, vim, tmux)

Rozdział 6

Opis prototypu - System „Reqmanager”

krótki wstęp

- 6.1 Zbieranie wymagań w Reqmanager
- 6.2 Edycja diagramów UML
- 6.3 Kolaboracja
- 6.4 Implementacja
 - 6.4.1 Diagramy przypadków użycia
 - 6.4.2 Przetwarzanie wymagań (Mapowanie obiektów w JS na encje w bazie danych)
 - 6.4.3 Generowanie dokumentacji
- 6.5 Problemy techniczne
- 6.6 Podsumowanie rozwiązania (zalety i wady)

Rozdział 7

Podsumowanie

krótki wstęp

7.1 Plan rozwoju

7.2 Zakończenie

Bibliography

- [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.
- [15] Jim Highsmith and Martin Fowler. “The Agile Manifesto”. In: *Software Development Magazine* 9.8 (2001), pp. 29–30.
- [16] A. Swartz J. Gruber. *Markdown*. 2004. URL: <http://daringfireball.net/projects/markdown/>.
- [17] Philippe Kruchten. *The Rational Unified Process: An Introduction*. 3rd ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003. ISBN: 0321197704.
- [18] NASA. *NASA Software Engineering Requirements*. 2009. URL: http://nodis3.gsfc.nasa.gov/npg_img/N_PR_7150_002A_/N_PR_7150_002A_.pdf.

- [19] U. Nikkula, J. Sajaniemi, and H. Kälviäinen. “A State-of-the-Practice Survey on Requirements Engineering in Small- and Medium-Sized Enterprises”. In: (2000).
- [20] 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.
- [21] 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.
- [22] Trygve Reenskaug. *Models - Views - Controllers*. 1979. URL: <http://heim.ifi.uio.no/~trygver/1979/mvc-2/1979-12-MVC.pdf>.
- [23] *Ruby on Rails vs Grails vs. Spring ROO vs. Spring App*. Stackoverflow.
- [24] I. Sommerville. *Software Engineering*. 8th ed. Pearson, 2006.
- [25] K. Subieta. *Słownik terminów z zakresu obiektowości*. 1999.
- [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.