Spis treści

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 przetwazania wymagań użytkownika z wykorzystaniem technologii internetowych, w niewielkich projektach.

W niniejszej pracy zaprezentowano rozwiązanie dla małych i średnich projektów, pozwalające skupić się na najważniejszych akpektach 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 jednoczenym 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 najistotniejszych aspektach definiowania i przetwarzania wymagań. [13]

W niniejszej pracy zaprezentowano nowoczesne narzędzie wspierające proces zbierania wymagań użytkownika. Przyjęte rozwiązania mają za zadanie ułatwić zespołom projektowym komunikację, dostęp do wiedzy oraz wizualizację przypadków użycia w kontekście fazy analizy i zbierania wymagań. Głównymi założeniami przy tworzeniu koncepcji rozwiązania, były m.in. prostota obsługi oraz skupienie uwagi użytkownika na najważniejszych aspektach definiowaVnia wymagania.

1.1 Cel pracy

W niewielkich projektach, proces gromadzenia i dokumentowania wymagań użytkowników jest najczęściej słabo sformalizowany. Wymagania 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 (Sofware As A Service) [!ref] często powielają tylko funkcjonalnosći potężnych (i drogich) systemów takich jak IBM DOORS [!ref] czy IBM ReqiusitePro [!ref]. Ponadto niezwykle trudno jest znaleźć rozwiązanie darmowe lub posiadające ogólno 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 mode-

lowania 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.: stawnowić 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.

1.2 Zarys koncepcji

Proces zbierania wymagań jest trudny, ponieważ często jest procesem niesformalizowanym, rozproszonym i udokumentowanym na wiele sposobów (różne nośniki danych, niekomatybilne 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 ninejszej pracy, został stworzony prototyp systemu pozwalający w łatwy sposób dokumentować gromadzone wymagania użytkownika. W powstałym systemie, głownymi 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 wymaganie 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 "WYSIWYGźnanych ze standardowych edytorów tekstu sprawia, 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 zewnętrzych 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 iteracyjnemu podejściu do tworzenia specyfikacji wymagań i umożliwia prezentację specyfikacji już na wczesnym etapie projektu.

Pozyskiwanie wymagań jest procesem kreatywnym. Koncepcja systemu bazuje na silnym przekonaniu, że brak ograniczeń w postaci skomplikowanych i przytłaczających fukncjonalności pozwala skupić się na procesie twórczym w fazie definiowania wymagań, umożliwiając rozwiązywanie rzeczywiste problemy.

1.3 Struktura pracy

W rozdziale 2 zostaną przedstawione, dostępne na rynku, narzędzia wspierające zarządzanie wymaganiami użytkownika. Rodział 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 dlaszego 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 zostanie przeprowadzona analiza rynku oprogramowania wspierającego fazę analizy. Druga część rozdziału zajmie się identyfikacją najistotniejszych problemów jakie posiadają istniejące rozwiązania.

2.1 Analiza rynku systemów wsparcia zbierania wymagań

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. Również klasyczne aplikacje okienkowe, które należy zainstalować na komputerze klienta nadal cieszą się dużą popularnością. Należy jednak zaznaczyć, iż w przeważającej większości są to starsze systemy, nierzadko implementowane jeszcze przed popularyzacją zaawansowanych aplikacji webowych. Niektóre firmy oferują także platformy w architekturze klient-serwer, wymagające ist-

nienia 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 (Sowtware 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 bardzo specyficzne problemy, często w obrębie ściśle określonego segmentu rynku. Jednym z przykładów takiego wąskiego segmentu mogą być np. aplikacje do zarządzania projektami online. Wśród innych popularnych rozwiązań znajdują się takie produkty jak basecamp.com, polski nozbe.com czy unfuddle.com. Innym przykładem mogą być aplikacje wspierające proces rekrutacji (humanway.com, recruiterbox.com, jobvite.com). Na uwagę zasługują także aplikacje wspierające rezerwacje hoteli, 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 rozwiązania problemów danego segmentu.

Powodem takiego stanu rzeczy jest szeroko pojęta popularyzacja internetu oraz przenoszenie się do sieci firm tworzących opgrogramowanie. Dystrybucja oprogramowania w formule SaaS ma wiele zalet w stosunku do klasycznych aplikacji "desktopowych". W szczególności pomięty jest w tym przypadku proces rozprowadzania aplikacji do klientów za pomocą sieci stacjonarnych sklepów z oprogramowaniem. Zaimplementowane rozwiązanie, jest gotowe do użycia z chwilą udostępnienia w internecie. Udostępnienie aplikacji na serwerach twórcy oprogramowania daje nieograniczone możliwości wprowadzania nowych funkcjonalności i poprawek. Z kolei monitorowanie zachowań klientów pozwala niemalże w czasie rzeczywistym odpowiadać na potrzeby użytkowników.

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 (http://gatherspace.com/) [!ref] «««« krótki opis głównych osobliwości gatherspace'a »»»», tracecloud.com «««« krótki opis głównych osobliwości tracecloud'a »»»», accompa.com «««« krótki opis głównych osobliwości accompy »»»»

Gatherspace

Tracecloud

Accompa

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 możliwośći 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. IBM DOORS, IBM Rational RequisitePRO oraz seapine.com.

RequisitePRO

DOORS

seapine

2.1.3 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).

2.2 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 powiela rozwiązania znane i sprawdzone funkcjonalności. W konsekwencji trudnym zadaniem jest znalezienie wyróżniających się elementów wśród oferowanych możliwości tych produktów.

Proces definiowania wymagań w niewielkich (budżet max w granicach 100tyś euro - ?!) 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 za specyfikację wymagań, wymagania przybierają postać wiadomości email, rozmów telefonicznych, no-

tatek ze spotkań i formalnych dokumentów. Zadaniem osoby lub zespołu odpowiedzialnego za specyfikację wymagań, jest przefiltrowanie cząstkowych informacji oraz ekstrakcja i udokumentowanie wymagań. Żadne narzędzie (przynajmniej na razie) nie zastąpi skutecznie pracy człowieka nad wyłuskaniem wszystkich wymagań.

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

3.1 Teoria inżynierii wymagań

Inżynieria wymagań (IW) jest dziedziną, na którą składają się wszelkie działania związane z odkrywaniem (elicitation), analizą (analysis), weryfikacją (validation) oraz zarządzaniem wymaganiami [16]. Dotyczy ona każdego systemu informatycznego, niezależnie od jego rozmiarów i jest jednocześnie jednym z najistotniejszych procesów w cylku 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].

3.1.1 Proces wytwórczy oprogramowania a inżynieria wymagań

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 popularyzjacji sprzętu komputerowego, powszechnego, dramatycznie rosnącego zapotrzebowania na programistów oraz nagłośnionych, wielkich porażek systemów informatycznych, jak katastrofa rakiety kosmicznej Mariner 1 [6]. 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 [9].

Pionierzy inżynierii oprogramowania podejmowali próby zaczerpnięcia wiedzy z zakresu inżynierii systemów i wykorzystania jej przy projektowaniu oprogramowania. 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" [18]. 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 [14]. 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" [11].

Podobieństwo 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. Inżyniera wymagań dotyczy każdego procesu wytwórczego.

[1]Royce podał w swoim modelu wariant liniowego procesu, gdzie kolejne fazy były zwrotne sprzężone z fazami poprzednimi [18], jednak w większości organizacji stosujących model kaskadowy, jest to ściśle sekwencyjny proces liniowy [15].

Procesy w inżynierii wymagań

Jak podaje Sommerville [16], 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) [12].

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 elicitaion". "Elicitaion" w dosłownym tlumaczeniu, 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. Z związku w 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 [8]. 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 opgrogramowania 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 sfomułowaniu wymagań w specyfikacji, aby żadne nie były ze sobą w sprzeczności. Kompletność jest weryfikacją dokumentu wymagań z perspektywy realizacji wszystkich porządanych funkcji. Wszystkie zdefiniowane wymagania muszą być realizowalne w określonym czasie, przy użyciu określonego sprzętu i opgrogramowania. 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 Boehem et al “Prototyping vs. Specifying“ [5] 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 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 implementacj 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, 7].

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.

3.2 Czego należy oczekiwać od nowoczesnego systemu zarządzania wymaganiami?

W celu zdefiniowania ogólnych wymagań nowoczesnego systemu, przywołamy jeszcze jedną definicję inżynierii oprogramowania:

> "the systematic approach of developing requirements through an iterative cooperative process of analysing the problem, documenting the resulting observations in a variety of representation formats, and checking the accuracy of the understanding gained".

3.3 Ogólny opis rozwiązania

3.4 System Reqmanager - opis

Rozdział 4

Technologie wykorzystane w implementacji

krótki wstęp

4.1 Z czego korzystałem z lotu ptaka - grails, java, tomcat, heroku, git, postgresql, javascript, biblioteka jsuml2, jquery

4.2 Framework Grails

Framework Grails jest stosunkowo młodą technologią opartą na sprawdzonych rozwiązaniach. Firma która zajmuje się rozwojem Grails, to SpringSource - ta sama organizacja, któ¶a odpowiedzialna jest 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 stanowi trzon technologiczny dla tego młodego narzędzia wspomagającego tworzenie aplikacji opartych o technologie webowe.

4.2.1 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 sytnaktycznym 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.

4.2.2 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 przyśpieszyć proces wytwarzania oprogramowania oraz w pewnym zakresie zapewnić wysoką jakość tworzonych rozwiązań. Przede wszystkim jednak, jest tzw. kontenerem Inversion Of Control (odwrócenie sterowania) [10]. ********** 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 [17]

4.2.3 Framework Grails

Grails integruje wszystkie najlepsze praktyki i narzędzia zarówno ze Spring Framework jak i Hibernate ORM. Jest poniekąd odpowiedzią środowiska

Javy, na rosnącą konkurencyjność frameworków opartych o dynamicznie typowane, skryptowe języki programowania jak Django (python) czy Ruby On Rails. Jednocześnie, ze względu na fakt iż jest swoistą nakładką na technologie oparte na javie, dostarcza znacznie większych możliwości niż konkurenci, wynikających z ekosystemu javy.

4.3 Pozostałe technologie

4.3.1 Postgresql

4.3.2 Javascript, jQuery i biblioteka jsUml2

4.3.3 System kontroli wersji git i serwer heroku

4.3.4 Srodowisko programistyczne (linux, vim, tmux)

Rozdział 5

Opis prototypu - System Żeqmanager"

krótki wstęp

5.1 Filozofia i architektura systemu

5.2 Język UML w trakcie fazy analizy

5.3 Zbieranie wymagań w Reqmanager

5.4 Technologie webowe w kontekście RIA (zalety i wady przenoszenia sie do webu)

5.5 Implementacja

5.5.1 Diagramy przypadków użycia

5.5.2 Przetwarzanie wymagań (Mapowanie obiektów w JS na encje w bazie danych)

5.5.3 Generowanie dokumentacji

5.6 Problemy techniczne

5.7 Podsumowanie rozwiązania (zalety i wady)

Rozdział 6

Podsumowanie

krótki wstęp

6.1 Plan rozwoju

6.2 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 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.

[6] 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.

[7] R. Budde and P. Bacon. *Prototyping: an approach to evolutionary system development*. Springer-Verlag, 1992. ISBN: 9783540543527.

[8] Institute of Electrical et al. *IEEE Guide to Software Requirements Specifications*. American national standard. IEEE, 1984. URL: http://books.google.pl/books?id=F0A1RAAACAAJ.

[9] 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.

[10] M. Fowler. *Inversion of Control Containers and the Dependency Injection pattern*. 2004.

[11] Jim Highsmith and Martin Fowler. "The Agile Manifesto". In: *Software Development Magazine* 9.8 (2001), pp. 29–30.

[12] Philippe Kruchten. *The Rational Unified Process: An Introduction*. 3rd ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003. ISBN: 0321197704.

[13] Julio Cesar S P Leite. *A Survey on Requirements Analysis*. Tech. rep. 1987.

[14] NASA. *NASA Software Engineering Requirements*. 2009. URL: `http://nodis3.gsfc.nasa.gov/npg_img/N_PR_7150_002A_/N_PR_7150_002A_.pdf`.

[15] 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`.

[16] I. Sommerville. *Software Engineering*. 8th ed. Pearson, 2006.

[17] K. Subieta. *Słownik terminów z zakresu obiektowości*. 1999.

[18] R. Winston. "Managing the Development of Large Software Systems". In: *Proceedings* (), pp. 1–9.