

SYSTEM ZARZĄDZANIA PROJEKTAMI, DEDYKOWANY METODY

PRZEPISACZ Z KARTY DYPLOMOWEJ

UNIWERSYTET ZIELONOGÓRSKI

Wydział Informatyki, Elektrotechniki i Automatyki

Praca dyplomowa

Kierunek: Informatyka

SYSTEM ZARZĄDZANIA PROJEKTAMI DEDYKOWANY
METODYCE SCRUM

Piotr Joński

Promotor:
dr inż. Andrzej Marciniak

Zielona Góra, 02 2016

Piotr Joński

Zielona Góra 10/02/2016

431IDZ

Wydział Informatyki,

Elektrotechniki i Automatyki UZ

OŚWIADCZENIE

Świadomy odpowiedzialności karnej oświadczam, że przedkładana praca dyplomowa pt.

System zarządzania projektami dedykowany metodyce Scrum

została napisana przeze mnie samodzielnie i nie była wcześniej podstawą żadnej innej urzędowej procedury związanej z nadaniem dyplomu wyższej uczelni lub tytułów zawodowych. Jednocześnie oświadczam, że w/w praca nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994r. o prawie autorskim i prawach pokrewnych innych osób (DZ. U. z roku 2000 Nr 80 poz. 904) oraz dóbr osobistych chronionych prawem cywilnym.

Oświadczam również, że egzemplarz pracy dyplomowej w formie wydruku komputerowego jest zgodny z egzemplarzem pracy dyplomowej w formie elektronicznej.

.....

podpis

Streszczenie

CO TU DAĆ?

słowa kluczowe: praca dyplomowa, skład komputerowy, formatowanie dokumentu.

Spis treści

1. Wstęp	1
1.1. Wprowadzenie i przegląd literatury	1
1.2. Cel i zakres pracy	1
1.3. Struktura pracy	1
2. Wprowadzenie	3
2.1. Szkic problemu	3
2.2. Czym właściwie jest scrum	3
2.3. Przegląd dostępnych narzędzi	5
2.4. Rozwiązanie problemu	6
3. Analiza biznesowa problemu i założenia projektowe	7
3.1. Role i użytkownicy systemu	7
3.2. Wymagania funkcjonalne	7
3.3. Wymagania нефunkcjonalne	10

Spis rysunków

Spis tabel

2.1. Porównanie wybranych systemów zarządzania projektami	5
---	---

Rozdział 1

Wstęp

1.1. Wprowadzenie i przegląd literatury

Co tu dać?

1.2. Cel i zakres pracy

Celem pracy było wytworzenie systemu do zarządzania projektami dedykowanego metody Scrum. Obecnie na rynku jest wiele zarówno darmowych jak i płatnych narzędzi, które oferują możliwość prowadzenia projektów różnymi metodami. W zakres pracy wchodzi:

1. zapoznanie się z literaturą tematu,
2. opracowanie założeń projektu,
3. spis wymagań funkcjonalnych i нефункциональных systemu,
4. implementacja wszystkich funkcjonalności oraz usunięcie powstałych błędów,
5. przetestowanie systemu ,
6. wytworzenie części opisowej pracy.

1.3. Struktura pracy

Opis rozdziałów zostanie uzupełniony na końcu pracyitemize

Rozdział 2

Wprowadzenie

2.1. Szkic problemu

Szybki rozwój w dziedzinie informatyki spowodował wzrost zapotrzebowania na systemy, które pomogłyby rozwiązywać problemy kwestii organizacyjnej projektów. W dzisiejszych czasach zarządzanie projektami jest szerokim zagadnieniem, a na jego temat powstało wiele publikacji. Autorzy proponują takie rozwiązania dot. zarządzania projektami jak:

1. scrum,
2. lean Software Development,
3. feature Driven Development,
4. test Driven Development.

Nie jest to wyczerpująca lista metodyk - istnieje wiele innych metod tworzenia oprogramowania, jednak opis wszystkich wykracza poza zakres tej pracy. W mojej pracy dyplomowej poruszam temat systemu do zarządzania projektami, który jest dedykowany metodyce Scrum. Nie bez powodu wybrana została właśnie ta zwinna metodyka - jest to na chwilę obecną najbardziej popularna forma wytwarzania oprogramowania, a umiejętności pracy wraz z nią są pożądane przez większość pracodawców na całym świecie.

2.2. Czym właściwie jest scrum

Scrum jest zwinną metodyką wytwarzania oprogramowania, której początki sięgają połowy lat 80. Polega ona na przyrostowym wytwarzaniu produktu oraz stałej komunika-

cji z klientem bądź też jego reprezentantem - właścicielem produktu (*ang. product owner*). Zespół deweloperski *team* podejmuje się przygotowania wybranych przez siebie funkcjonalności, których wybór może być zależny od właściciela produktu - to on ustala priorytety. Jednak w celu zminimalizowania ryzyka "wyższej władzy", czyli narzucaniu zbyt wysokich wymagań i nieosiągalnych terminów, co często czynią *product ownerzy*, została wprowadzona kolejna rola - *scrum master*, którą ciężko jest przetłumaczyć na język polski (z tego względu w dalszej części pracy będę posługiwał się właśnie tym terminem). To on "chroni" zespół przed przepracowaniem i rozwiązuje napotymane po drodze problemy. Aby prawidłowo przedstawić omawiany temat należy zapoznać się z kilkoma pojęciami, które są nieodłącznym elementem tej metody:

1. **właściciel produktu** (*ang. product owner*) - osoba odpowiedzialna za projekt,
2. **scrum master** - osoba, która pomaga zespołowi w organizacji czasu pracy a także rozwiązuje powstałe problemy,
3. **zespół deweloperski** (*ang. team*) - zespół programistów wspólnie pracujący nad wytworzeniem produktu,
4. **rejestr produktu** (*ang. backlog*) - jest to zbiór zadań / funkcjonalności wchodzące w skład wytwarzanego projektu,
5. **interwał** (*ang. sprint*) - interwały czasowe, w których realizowane są zadania. Bezpośrednio przed każdym sprintem występuje planowanie, czyli wybieranie funkcjonalności do zrealizowania w danym sprincie. Bezpośrednio po sprincie powinna wystąpić retrospektywa oraz demo produktu, które jest często pomijane przez wiele zespołów,
6. **historyjka** (*ang. story*) - najmniejsza jednostka podlegająca ocenie (wg. wybranej przez zespół skali).
7. **zadanie** (*ang. task*) - zadania, które mogą być powiązane bezpośrednio ze story lub projektem.
8. **retrospektywa** (*ang. retrospective*) - wydarzenie, które ma miejsce na koniec sprintu. W tym momencie zespół deweloperski spisuje wszystkie wady i zalety minionego sprintu oraz wspólnie z *scrum masterem* starają się rozwiązać zaistniałe problemy. Jest to jeden z ważniejszych elementów Scruma.

2.3. Przegląd dostępnych narzędzi

Wytwarzając ten system skupiłem się, aby wpasował się on w wybraną przeze mnie metodykę oraz rozwiązywał szereg mankamentów, które napotkałem podczas korzystania z obecnych już rozwiązań. W celu lepszego zrozumienia problemów postanowiłem opisać kilka z nich.

Pierwszym i niewątpliwie największym problemem tego typu systemów jest to, że są one płatne, przez co nie wszystkich na nie stać. Niektóre są darmowe, lecz tylko dla projektów typu open source, co nie sprawdza się podczas pracy nad wspólnymi projektami w firmie lub np. pracą dyplomową.

Kolejną wadą jest to, że nie są one proste w instalacji. Często występują jako potężne programy, przez co nie można ich bezpłatnie hostować w chmurze gdyż potrzebują dużo płatnych zasobów.

Ostatnim, jednak nie mniej ważnym problemem jest ich czytelność. Oferują one szereg zbędnej zwykłemu użytkownikowi - programiście - funkcjonalności, która jest przydatna tylko w określonych warunkach. Taki ogrom zakładek i okienek powoduje często zamęt i niezrozumienie wśród programistów.

Aby zobrazować wagę powyższych problemów zostały one zestawione w poniższej tabeli porównującej wybrane systemy wraz z systemem, który został wytworzony w ramach pracy dyplomowej.

Tab. 2.1. Porównanie wybranych systemów zarządzania projektami

Cecha systemu	Wybrane systemy			
	JIRA	Bitbucket	GitHub	Autorski system
Darmowy	do 10 użyt.	do 5 użyt.	open source	tak
Hosting	wew. / zew.	zew.	zew.	wew. / zew.
Czas instalacji	ok. 5 minut	nd.	nd.	ok. 2 minuty
Wymagania	baza danych	nd.	nd.	docker
Cechy	issue tracker wiki	issue tracker wiki	issue tracker wiki	issue tracker

2.4. Rozwiązanie problemu

Celem pracy było wytworzenie systemu do zarządzania projektami dedykowanego metody Scrum. Obecnie na rynku jest wiele zarówno darmowych jak i płatnych narzędzi przedstawionych w tabeli 2.1, które oferują możliwość prowadzenia projektów różnymi metodami. Wcześniej wymienione wady skłoniły mnie do opracowania własnego systemu, który ma za zadanie rozwiązać wspomniane problemy w następujący sposób:

1. łatwość instalacji - do tego celu zostało wykorzystane oprogramowanie docker, które wspiera błyskawiczne stawianie systemów,
2. przejrzysty interfejs użytkownika - został osiągnięty dzięki open source'owej bibliotece PrimeFaces,
3. bez zbędnej funkcjonalności - zostały zaimplementowane tylko obligatoryjne funkcje tego typu systemu oraz niewielka ilość udogodnień.

Rozdział 3

Analiza biznesowa problemu i założenia projektowe

3.1. Role i użytkownicy systemu

W projektowanym systemie występują cztery rodzaje użytkowników:

1. administrator - posiada on największe uprawnienia w systemie,
2. właściciel produktu - odzwierciela rolę właściciela produktu w Scrumie,
3. scrum master - odzwierciela rolę scrum mastera w Scrumie,
4. deweloper - posiada najmniejsze uprawnienia, jest częścią zespołu deweloperskiego.

Na tym etapie warto wspomnieć, że każdy użytkownik jest deweloperem. Każdy projekt może mieć tylko jednego właściciela produktu, a każdy z nich może być product ownerem tylko raz. Dodatkowo zespół deweloperski ma przydzielonego tylko jednego scrum mastera, przy czym scrum master może być przypisany do wielu zespołów jednocześnie.

3.2. Wymagania funkcjonalne

Prezentowany przeze mnie system ma pewne założenia oraz wymagania. W tym rozdziale zajmę się opisem wymagań funkcjonalnych.

We wcześniejszym akapicie zostały omówione role w systemie. Każda z tych ról ma pewne uprawnienia lub restrykcje. Każda taka cecha zostanie przedstawiona jako wymaganie funkcjonalne prezentowanego systemu.

Jednym ze sposobów na prowadzenie dokumentacji projektu, jak i zbioru wymagań jest utrzymywanie rejestru historyjek użytkownika. Historyjki użytkownika są częścią zwinnych metody prowadzenia projektu. Jako że wytwarzanemu systemowi towarzyszy metodyka Scrum, nie mogło tutaj zabraknąć tego elementu.

Historyjka jest jednostką funkcjonalności w projektach XP. Pokazujemy postęp prac, dostarczając przetestowany i zintegrowany kod, który składa się na implementację danej historyjki. Historyjka powinna być zrozumiała i wartościowa dla klientów, testowalna przez programistów i na tyle mała, żeby programiści mogli zaimplementować sześć historyjek w takcie jednej iteracji¹.

Historyjki mogą mieć wiele wzorców. W tej pracy są stosowane dwa z nich:

- Jako *<typ użytkownika>* mogę *<nazwa zadania>*.
- Jako *<typ użytkownika>* mogę *<nazwa zadania>* w celu *<cel>*[6]

Administrator

1. jako administrator mogę tworzyć nowych użytkowników w celu dodania ich do systemu,
2. jako administrator mogę usuwać użytkowników z systemu z wyjątkiem siebie samego,
3. jako administrator mogę dodawać użytkowników do zespołu w celu modyfikacji zespołu,
4. jako administrator mogę usuwać użytkowników z zespołu w celu modyfikacji zespołu,
5. jako administrator mogę nadać uprawnienia administratora dowolnemu użytkownikowi,
6. jako administrator mogę odebrać uprawnienia administratora dowolnemu administratorowi,
7. jako administrator mogę przypisać właściciela produktu do projektu,
8. jako administrator mogę usunąć właściciela produktu z projektu,
9. jako administrator mogę utworzyć projekt w celu dodania go do systemu,

¹K. Beck, M. Fowler, *Planning Extreme Programming*, Addison-Wesley, Boston 2000, s.42

10. jako administrator mogę usunąć projekt w celu usunięcia z systemu oraz powiązanych z nim elementów t.j. sprint, story, backlog oraz inne. Operacja usuwania odbywa się kaskadowo,
11. jako administrator mogę modyfikować projekt,
12. jako administrator mogę tworzyć nowe zespoły w celu dodania ich do systemu,
13. jako administrator mogę dodawać zespoły do projektów w celu przydzielenia uprawnień,
14. jako administrator mogę usuwać zespoły z projektów w celu odebrania uprawnień,
15. jako administrator mogę przypisać scrum mastera do zespołu,
16. jako administrator mogę usunąć scrum mastera z zespołu,
17. jako administrator mogę tworzyć, usuwać oraz edytować statusy zadań,
18. jako administrator mogę tworzyć, usuwać oraz edytować priorytety zadań,
19. jako administrator mogę tworzyć, usuwać oraz edytować typy zadań.

Product owner

1. TODO

Scrum master

1. TODO

Developer

1. jako deweloper mogę tworzyć zadania,
2. jako deweloper mogę usuwać zadania,
3. jako deweloper mogę dodawać komentarze do zadań,
4. jako deweloper mogę dodawać komentarze do retrospektyw,
5. jako deweloper mogę edytować swój profil,
6. jako deweloper mogę zmienić swoje hasło,
7. jako deweloper mogę przeglądać profile innych użytkowników,
8. jako deweloper mogę przeglądać wszystkie projekty, do których jestem przypisany,
9. jako deweloper mogę przypisać zadanie do dowolnego użytkownika,

3.3. Wymagania niefunkcjonalne

Kolejnym zagadnieniem, które zostanie poruszone w tym rozdziale będą wymagania niefunkcjonalne. Do analizy zostanie wykorzystana metoda FURPS. Oto pełny spis wymagań niefunkcjonalnych systemu:

1. **Functionality** - funkcjonalność, system powinien:

- spełniać wszystkie wymagania funkcjonalne,
- mieć możliwość administracji poprzez panel administracyjny,
- posiadać audyt w postaci logów systemu,
- być łatwo rozszerzalny.

2. **Usability** - używalność, system powinien posiadać następujące cechy:

- ergonomia - łatwość używania,
- look & feel, czyli estetyczność oraz możliwość modyfikacji wyglądu,
- internacjonalizacja - wiele wersji językowych.

3. **Reliability** - niezawodność, w której skład wchodzi:

- dostępność, czyli czas pomiędzy awariami,
- odzyskiwalność - ile czasu zajmie ponowne uruchomienie systemu.

4. **Performance** - wydajność, system powinien:

- być przepustowy ,
- mieć dużą responsywność,
- działać szybko.

5. **Supportability** - wsparcie, do którego należy:

- prostota w instalacji,
- łatwość konfiguracji,
- adaptowalność, czyli możliwość zaadoptowania systemu do innych warunków,
- testowalność, czyli testy jednostkowe oraz integracyjne.

Rozdział 4

Dokumentacja projektowa

Bibliografia

- [1] JavaServer Faces i Eclipse Galileo. Tworzenie aplikacji WWW *Andrzej Marciniak* Helion 2010
- [2] Core JavaServer Faces. Wydanie II *David Geary, Cay S. Horstmann* Helion 2008
- [3] Enterprise JavaBeans 3.0 *Bill Burke & Richard Monson-Haefel* Helion 2007
- [4] JBoss AS 7. Tworzenie aplikacji *Francesco Marchioni* Helion 2014
- [5] Core J2EE. Wzorce projektowe *Deepak Alur, John Crupi, Dan Malks* Helion 2004
- [6] Scrum. O zwinnym zarządzaniu projektami. Wydanie II rozszerzone *Mariusz Chrapko* Helion 2015
- [7] Java. Kompendium programisty. Wydanie VIII *Herbert Schildt* Helion 2012
- [8] Zwinne wytwarzanie oprogramowania. Najlepsze zasady, wzorce i praktyki *Robert C. Martin* Helion 2015
- [9] Git. Rozproszony system kontroli wersji *Włodzimierz Gajda* Helion 2013
- [10] Mistrz czystego kodu. Kodeks postępowania profesjonalnych programistów *Robert C. Martin* Helion 2013
- [11] Czysty kod. Podręcznik dobrego programisty *Robert C. Martin* Helion 2010
- [12] Rusz głową! Wzorce projektowe *Eric Freeman, Elisabeth Freeman, Bert Bates, Kathy Sierra* Helion 2011
- [13] Refaktoryzacja do wzorców projektowych *Joshua Kerievsky* Helion 2005