Programowanie zespołowe

Marcin Ochab
21.02.2018

Wstęp

Zasady ogólne

- Tok indywidualny:
  - wszystkie zasady pozostają takie same
  - **łącznie z obecnościami!**
- Obecności:
  - dopuszczalne 2 nieusprawiedliwione nieobecności
  - **3 nieobecność = skreślenie z listy studentów**
  - nie ma możliwości odrabiania zajęć
  - **nieprzygotowanie się do zajęć = nieobecność!**

Zaliczenie

- Ukończony projekt oddany w nieodwołalnym terminie:
  * ostatnie zajęcia: **06.06.2017**
  * **niedotrzymanie daty = powtarzanie przedmiotu!**

- 5 kolokwiów:
  * podstawy Javy
  * GIT z linii komend
  * standardy kodowania
  * SQL
  * zaawansowana Java

- Jedna poprawka każdego - hurtowo dopiero w sesji

Zaliczenie

- prezentacja* przez grupę jednego z referatów:
  - standardy kodowania
  - budowanie złożonych kwerend SQL
  - dokumentowanie kodu źródłowego - javadoc
  - dzienniki logów w Javie (Log4J)

  *prezentacja następuje po uprzednim zatwierdzeniu jej treści i jakości podczas konsultacji! (ok 30 min)

- Spełnienie zasad SCRUM!
- Pozytywne zaliczenie **każdego** z wymagań
- Zaliczenie wykładu **tylko** po zaliczeniu tego co powyżej!

Kryteria oceny

Ocena końcowa jest wyznaczana na podstawie:

- 50% - średnia arytmetyczna ocen z zaliczonych pisemnych kolokwiów- student musi zaliczyć na ocenę pozytywną wszystkie kolokwia

- 50% - ocena projektu wraz z dokumentacją

Przedsięwzięcie

Temat: System zarządzania zadaniami

Technologie:

- JavaFX - NetBeans
- MySQL (MariaDB)
- GitHub
- Jira

Efekt po 6 Sprintach:

- gotowy produkt w opakowaniu
- instalator
- instrukcja obsługi, dokumentacja, UML

Zasady SCRUM

- Samoorganizacja wewnątrz zespołu 4-5 os.
- Odpowiedzialność zbiorowa w ramach grupy
- 2 tygodniowe Sprinty
- Utrzymywanie aktualnych diagramów w Jira:
  - przesuwanie Tasków przez osoby za nie odpowiedzialne
  - obowiązkowy Daily Sprint w środku tygodnia pomiędzy zajęciami
  - razie potrzeby komentarze, etykiety

Zasady SCRUM

- Każdy Sprint ma być działającą nową wersją spełniającą przede wszystkim Cel Sprintu, ale i jego wszystkie Story

- Możliwość nie wypełnienia tylko jednego celu Sprintu z wszystkich 6 !

- Zmiany w Product Backlogu robimy tylko w uzgodnieniu z Product Ownerem (prowadzącym)

- Pozostałe tablice są do dyspozycji zespołu:

Sprint Backlog, In process , Ready for Testing, Testing, Done

Daily Sprint

- Krótkie spotkanie wszystkich członków grupy oko w oko
  - na żywo, Skype, Google Hangouts, Appear.in, ...
  - minimum 3 razy na Sprint w tym raz na zajęciach w środku Sprintu
  - nie może to być tylko offlineowa wymiana maili/wiadomości ma messengerze!

- Każdy odpowiada bardo zwięźle na 3 pytania:
  - Co zrobiłem ostatnio, co pomogło Zespołowi Deweloperskiemu przybliżyć się do osiągnięcia Celu Sprintu?
  - Co zrobię w najbliższym czasie, co pomoże Zespołowi Deweloperskiemu przybliżyć się do osiągnięcia Celu Sprintu?
  - Czy widzę jakiekolwiek przeszkody mogące uniemożliwić mi lub Zespołowi Deweloperskiemu osiągnięcie Celu Sprintu?

- Sprawozdanie ze spotkania trafia niezwłocznie do prowadzącego mailem

Definition of Done

- task ma być działający tak, aby pozwalał spełnić cel sprintu
- kod musi być:
  - udokumentowany (okomentowany - javadoc)
  - zgodny ze standardami kodowania
  - przetestowany - Unit
  - commity zlinkowane z Jira mają być na GitHubie

- tylko task zgodny z Definition of Done może być przesunięty do Done

Definition of Done wyjaśnienia

- jeżeli tematu nie wyjaśniono wcześniej kod musi być:
  - okomentowany/udokumentowany:
    - nieznajomość javadoc nie zwalnia z komentowania kodu stosownie do aktualnego stanu wiedzy zespołu
  - zgodny ze standardami kodowania:
    - nieznajomość standardów kodowania nie zwalnia ze uporządkowania i schludności (NetBeans format, nazewnictwo klas i zmiennych, itd.)
  - przetestowany:
    - nieznajomość JUnit nie zwalnia z testów: kod musi zostać przetestowany ręcznie przez inną osobę niż autor, co musi być widoczne w Jira. Testy musza później zostać uzupełnione.

Lista obecności

i

Podział na grupy

Przydział referatów

Przygotowanie na kolejne zajęcia

- Kolokwium z podstaw Javy:
  - zakres: roz. 3-10, 11-13 bez metod, 14, 17, 19-27

http://www.tutorialspoint.com/java/java_tutorial.pdf

  - Bruce Eckel „Thinking in Java" - edycja polska

- Dopracowanie specyfikacji wymagań i pomysłu na aplikację
- Przypomnienie diagramów UML (diagramy przypadków użycia, stanów, sekwencji, klas)

Specyfikacja systemu

- Aplikacja powinna mieć kilka modułów: administracja użytkownikami (role), moduł raportów, moduł konfiguracji
- Użytkownik zwykły widzi swoje zadania
- Administrator zarządza wszystkim
- Kierownik widzi zadania swojej grupy
- Raporty w pdf

Cele i zakres systemu

- dokładne zdefiniowanie problematyki projektu, czyli do czego ma służyć tworzony system komputerowy i jakie problemy mają być rozwiązywane za jego pomocą,

- ustalenie tzw. interesariuszy projektu, czyli osób lub grup osób, które są potencjalnymi użytkownikami wytworzonego oprogramowania,

- ustalenie jakie są gromadzone dane w organizacji i kto z użytkowników je wprowadza do systemu,

Cele i zakres systemu

- ustalenie rodzajów potrzebnych raportów (wydruków) które mają być generowane i udostępniane w projekcie oraz kto z użytkowników je generuje i z nich korzysta,
- ustalenie rodzajów potrzebnych dokumentów które mają być generowane w projekcie oraz kto z użytkowników je ma generować i z nich korzystać,

Cele i zakres systemu

- ustalenie przepływu informacji środowisku, w którym ma pracować system,
- ustalenie uprawnień (praw dostępu) poszczególnych grup użytkowników (np. ustawa o ochronie danych osobowych).

Określenie wymagań

Wymagania funkcjonalne - dotyczą tego co ma realizować system; jakie ma spełniać funkcje, jakich dostarczać usług, jak zachowywać się w określonych sytuacjach. Wymagania funkcjonalne powinny być kompletne (opisywać wszystkie usługi żądane od systemu) i spójne (nie zawierać stwierdzeń sprzecznych).

Określenie wymagań

Wymagania niefunkcjonalne - dotyczą tego jak system powinien realizować swoje zadania; np. wymagania dotyczące koniecznych zasobów, ograniczeń czasowych, niezawodności, bezpieczeństwa, przenośności, współpracy z określonymi narzędziami i środowiskami, zgodności z normami i standardami, a także przepisami prawnymi, w tym dotyczącymi tajności i prywatności, itp. Wymagania niefunkcjonalne dla wielu systemów są co najmniej tak ważne jak wymagania funkcjonalne (np. szybkość działania wyszukiwarki może być równie ważna jak precyzja wyszukiwania)