Zaawansowana inżynieria oprogramowania

Wykład:

"Programowanie ekstremalne
(ang. eXtreme Programming)"

Marek Krętowski
pokój 206
e-mail: m.kretowski@pb.edu.pl
http://aragorn.pb.bialystok.pl/~mkret

Wersja 1.11

Początki programowania ekstremalnego

- Powstało w wyniku obserwacji tradycyjnych procesów poprzez eliminację elementów spowalniających tworzenie oprogramowania i położenie szczególnego nacisku na elementy zwiększające wydajność, efektywność i jakość oprogramowania
  - syndrom LOOP (**L**ate, **O**ver budget, **O**vertime, **P**oor quality)
- Relatywnie nowa tzw. lekka (lub zwinna) metodyka opisująca proces tworzenia oprogramowania
- XP narodziło się w środowisku programistów Smalltalk-a; Kent Beck i Ward Cunningham opracowali zestaw dobrych praktyk programistycznych, które miały zwiększyć efektywność tworzenia oprogramowania.
- Wg K. Beck-a XP jest to "skuteczny, bezpieczny, elastyczny, naukowy i przyjemny sposób programowania"
- Pierwszy projekt (C3 - Chrysler Comprehesive Compensation) oparty na wypracowanych zasadach zrealizowano w drugiej połowie lat 90-tych

Programowanie ekstremalne

- Programowanie ekstremalne jest to realna koncepcja określająca metodologię tworzenia wysokiej jakości oprogramowania.
- Zorientowana na czynnik ludzki, starająca się maksymalnie zaadaptować do naturalnych cech ludzi a nie im przeciwdziałać.
- Fundamentalne założenia XP:
  - praca z rzeczywistymi klientami zamiast ze specyfikacją,
  - wyprzedzające tworzenie testów,
  - wstępne opinie użytkowników (historie, opowieści),
  - wyeliminowanie niepotrzebnych działań
- Założenia te, jako fundamentalne, pozostają niezmienne dla każdego projektu XP. Ponieważ jednak jest to metodologia otwarta, a przy tym młoda, reszta czynników wciąż ewoluuje.
- Ekstremalność ma nawiązywać do sportów ekstremalnych (intensywność, odwaga, ...)

Czym XP rożni się od innych metodyk?

- Wczesne, konkretne i ciągłe informacje zwrotne w krótkich cyklach programowania
- Planowanie przyrostowe
- Zdolność do elastycznego projektowania i implementacji funkcjonalności w zależności od dynamicznie zmieniających się wymagań
- Opieranie się na automatycznych testach napisanych wspólnie z klientem
- Opieranie się na komunikacji słownej i kodzie źródłowym oprogramowania w celu przekazania intencji programistów i klienta
- Zapewnienie bliskiej współpracy programistów
- Promowanie prostoty rozwiązań i iteracyjnego procesu poprawy zaimplementowanych rozwiązań
- ...

Pięć wartości XP

- **Komunikacja -** ścisła, bezpośrednia współpraca pomiędzy programistami i klientem
- **Prostota** - poszukiwanie najprostszego rozwiązania spełniającego wyznaczone założenia, skupienie się na tym co w danej chwili jest potrzebne
- **Informacja zwrotna** - szybkie iteracje i bezpośredni kontakt umożliwiają uzyskiwanie niemalże natychmiastowych informacji na temat stanu projektu i aktualnie istniejących zagrożeń
- **Odwaga** - szybkie podejmowanie decyzji; agresywność i brak oporów przed zmianami mogącymi wpłynąć na jakość, termin i koszt oprogramowania
- **Szacunek** – wzajemny szacunek członków zespołu (zarówno klientów jak i informatyków); szanowanie czasu i pracy innych

Zespół klientów

- **Opowiadacze** - (*storyteller*), osoby merytorycznie znające dziedzinę problemu
  - przekazują szczegółowe wymagania odnośnie rozwiązywanych problemów w postaci historii użytkowników (*user stories*) - elementarnych scenariuszy podstawowych funkcji systemu
  - do nich kierowane są pytania dotyczące niejasności
  - czuwają nad właściwym kierunkiem prac programistów (zbieżność implementacji z koncepcją)
  - określa się ich mianem dawców złota
- **Akceptanci** - weryfikuje zgodność tego co powiedzieli opowiadacze z tym, co stworzyli programiści
  - przeprowadzają testy akceptacyjne
- **Posiadacze złota** - dostarczają zasobów do tworzenia systemu
  - do ich zadań należy zatrudnianie ludzi, zapewnianie funduszy oraz odpowiedniego wyposażenia
- **Planiści** - synchronizuje cykl wdrażania systemu z normalną pracą przedsiębiorstwa
  - odpowiednikiem w zespole programistów jest tropiciel
- **Wielki szef** - człowiek sprawujący generalny nadzór nad wszystkimi pracami związanymi z powstawaniem systemu
  - pracuje na potrzeby obydwu zespołów
  - odpowiada za klarowną organizację całego przedsięwzięcia

Prawa i obowiązki klientów

- Obowiązek stworzenia wspólnie z programistami jasnej czytelnej specyfikacji wymagań
- Prawo do informacji jaki jest koszt, czas implementacji poszczególnych funkcjonalności i jaki nakład pracy potrzebny jest do osiągnięcia poszczególnych celów
- Prawo do obserwacji postępu prac poprzez ewaluację dostarczanych w kolejnych iteracjach działających części systemu
- Prawo do potwierdzenia jakości systemu poprzez wykonanie serii testów
- Prawo do zmiany zdania, podmiany wymagań i zmiany priorytetów
- Prawo do informacji o zmianach planu, ewentualnych opóźnieniach, zmianach estymacji tak aby możliwa była zmiana zakresu w celu uzyskania interesującej funkcjonalności w założonym czasie i budżecie

Zespół programistów

- **Trener** - służy programistom doświadczeniem zawodowym,
  - odpowiedzialny za płynny przebieg projektu,
  - decyduje w sytuacjach kryzysowych
- **Tropiciel** - stanowi sprzężenie zwrotne pomiędzy *planistą* a programistami
  - jego zadaniem jest zweryfikowanie realności przyjętych założeń (głównie pod kątem szybkości prac)
  - posiada wizję całości systemu, intuicyjnie wyczuwając jak ryzykowne są poszczególne etapy
  - ma również informować planistę o postępach
- **Negocjator** - zapewnia należytą komunikację pomiędzy zespołem programistów i zespołem klientów
  - poza tym odpowiedzialny jest za komunikację wewnątrz zespołów; do jego zadań należy organizowanie większych zebrań
  - rozsądzanie sporów, osiąganie kompromisów
- **Architekt** - buduje elastyczną strukturę programu, która łatwo poddaje się refaktoryzacji; jego zadania:
  - opracowanie minimalnej liczby testów potrzebnej do weryfikacji poprawności architektury w jej aktualnej wersji
  - wykrycie wszystkie defekty w kodzie

Prawa i obowiązki programistów

- Prawo do informacji co należy zrobić, dzięki prostej, czytelnej specyfikacji wymagań z jasno określonymi priorytetami
- Obowiązek określenia ile realizacja danego zadania zajmie czasu i prawo do zmiany estymacji
- Prawo akceptacji odpowiedzialności za zadanie w miejsce obowiązku akceptacji przydzielonego odgórnie zadania i dotyczącej go estymacji
- Obowiązek tworzenia wysokiej jakości oprogramowania spełniającego wyznaczone wymagania (ale nic wykraczającego poza wymagania!)
- Prawo do spokojnej, dającej poczucie dobrej zabawy i spełnienia pracy

Reguły XP

- Praktyki wspierające wartości XP podzielone zostały na następujące obszary:
  - Planowanie (planning)
  - Projektowanie (designing)
  - Kodowanie (coding)
  - Testowanie (testing)
  - Zarządzanie (managing)
- W odróżnieniu od innych metodologii w XP nie ma wyraźnego podziału na fazy, w których realizowane są zadania z poszczególnych obszarów tzn. planujemy przez cały czas, projektujemy przez cały czas, implementujemy przez cały czas i testujemy przez cały czas

Reguły i praktyki XP
(www.eXtremeProgramming.org)

Planning

- User stories are written.
- Release planning creates the release schedule.
- Make frequent small releases.
- The project is divided into iterations.
- Iteration planning starts each iteration.

Managing

- Give the team a dedicated open work space.
- Set a sustainable pace.
- A stand up meeting starts each day.
- The Project Velocity is measured.
- Move people around.
- Fix XP when it breaks.

Coding

- The customer is always available.
- Code must be written to agreed standards.
- Code the unit test first.
- All production code is pair programmed.
- Only one pair integrates code at a time.
- Integrate often.
- Set up a dedicated integration computer.
- Use collective ownership.

Designing

- Simplicity.
- Choose a system metaphor.
- Use CRC cards for design sessions.
- Create spike solutions to reduce risk.
- No functionality is added early.
- Refactor whenever and wherever possible.

Testing

- All code must have unit tests.
- All code must pass all unit tests before it can be released.
- When a bug is found tests are created.
- Acceptance tests are run often and the score is published.

Historie użytkowników

- Tworzenie oprogramowania w XP odbywa się przyrostowo przez wdrażanie kolejnych wydań produktu. Planowanie wydania odbywa się przed rozpoczęciem każdej nowej iteracji. Podstawowym celem jest oszacowanie każdej pojedynczej historii użytkownika, powstałej w wyniku gry planistycznej z klientem.
- Do szacowania używa się jednostek zwanych idealnymi osobo-tygodniami. Idealny osobo-tydzień to tydzień pracy wyłącznie nad programem, bez dodatkowych zajęć, ale wliczający czas testowania programu.
- Spisywane historie (scenariusze) użytkowników (ang. user stories)
  - podobne trochę do przypadków użycia (UML), ale zdecydowanie krótsze i pisane językiem zrozumiałym dla klienta, stanowią formalny dokument wymagań
  - stanowią podstawę dla planowania i podziału zadań oraz dla testów akceptacyjnych
  - klient określa, które historie są dla niego najważniejsze i które powinny być realizowane w pierwszej kolejności
  - projektanci mogą stwierdzić, ze przedstawiona historia jest zbyt duża i poprosić użytkownika o podzielenie jej na mniejsze części.

Stały kontakt z klientem

- Potrzebny jest jak najszerszy kontakt z klientem. Jeżeli jest on jednak nieosiągalny, może to prowadzić do opóźnień w realizacji systemu (czy wręcz fiaska)
- Najlepiej jeśli klient dostępny jest przez cały projekt, w idealnym przypadku pracuje w tym samym pomieszczeniu co zespół projektowy (ang. *on-site customer*)
- Aby zadowolić wymagania klienta należy bezwzględnie podążać za jego życzeniami
- Niektórzy twierdzą, że klient nie jest poważny, jeżeli nie może poświęcić wystarczająco dużo czasu dla nowego systemu.

Prostota rozwiązań

- "Nigdy nie projektuj funkcjonalności, która może nigdy nie zostać wykorzystana"
- XP zakłada, że wymagania klienta, rynku i sytuacja w branży ciągle się zmieniają. Nie ma więc sensu planować rozwiązań, o których nie wiadomo, czy zostaną wykorzystane w przyszłości
- Celem XP jest jak najszybsze i najprostsze osiągniecie satysfakcji klienta przez dostarczenie oprogramowania, spełniającego postawione wymagania
- Programista implementując stara się osiągnąć sukces przy minimalnych nakładach

  "Always do the simplest thing that could possibly work"
- Jeśli jednak okaże się, że projektowane rozwiązanie jest zbyt uproszczone może zostać szybko zmienione i rozszerzone

Prototypowanie

- Minimalizacja ryzyka poprzez budowę prototypów (ang. *spike solutions*)
- Prototypowanie dla zagadnień stanowiących potencjalne ryzyko wystąpienia problemów z implementacją
- Prototyp powinien tylko obejmować zagadnienia związane z ryzykiem i nic poza tym
- Prototypowanie powinno się zakończyć w momencie uzyskania odpowiedzi, że dany problem można rozwiązać
- Nigdy nie powinno się wykorzystywać kodu prototypu w produkcji!

"Build one to throw away"

Refaktoryzacja

- *"Refactoring is the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure"* (Martin Fowler):
  - upraszczanie, restrukturalizacja kodu;
  - zmiana istniejącej implementacji w celu umożliwienia rozbudowy o nową funkcjonalność;
  - dostosowanie kodu do przyjętych standardów;
- Sesje refaktoringowe powinny być przeprowadzane regularnie w trakcie trwania projektu
- Korzystanie z narzędzi wspomagających wprowadzanie jednoczesnych zmian w całym kodzie
  - np. zmian sygnatur metod w deklaracji klasy i klasach wykorzystujących te metody
- Nigdy nie należy refaktoryzować kodu, dla którego nie istnieją testy

Refaktoryzacja (2)

- Sednem jest stosowanie niewielkich transformacji (nazywanych refaktoryzacjami) niezmieniających zachowania (funkcjonalności) po których następuje kontrolne uruchomienie testów jednostkowych
  - seria refaktoryzacji może zmieniać istotnie (poprawić) strukturę kodu
  - przykładowe transformacje: ekstrakcja metody (lub stałej), wprowadzenie zmiennej objaśniającej
- Typowe (przykładowe) cele refaktoryzacji:
  - usunięcie powtórzeń kodu, poprawa czytelności, skrócenie kodu metod, usunięcie zakodowanych „na sztywno" stałych, ...
- **Zapaszek** (brzydki) **kodu** (ang. code smells) – określenie (metafora) używane w celu opisania kodu wymagającego (zapewne) refaktoryzacji; czasami może też być **smrodek kodu** (ang. code stench) – kod nieodwołalnie wymaga poprawy
  - przykłady zapaszków:powtórzony kod; duża metoda;uderzająco podobne klasy; klasa z ogromną ilością kodu....

Programowanie w parach

- Podczas gdy jedna osoba pisze kod, druga na bieżąco go sprawdza, sugeruje możliwe rozwiązania, może służyć pomocą i zwraca uwagę na błędy syntaktyczne
- PP jest dość trudne, wymaga dobrego zgrania zespołu, ale przynosi wymierne korzyści w postaci lepszego kodu
- PP pomaga również w dokonywaniu poprawek; druga osoba może bowiem wiedzieć więcej o danym fragmencie kodu
- programiści wewnątrz pary powinni co jakiś czas zamieniać się miejscami; w ten sposób wysiłek jest rozłożony równomiernie
- Ponadto pary powinny się między sobą mieszać (brak stałych partnerów - dotyczy tylko programowania!)
- Generalnie PP pomaga propagować wiedzę o różnych fragmentach programu na cała grupę osób, pracujących przy systemie

Standard kodowania

- XP narzuca wszystkim programistom wspólny standard kodowania i dokumentowania - eliminuje to nieporozumienia i ułatwia rozumienie kodu zwłaszcza przy PP
- Standard taki powinien być ustalony i zaakceptowany przez cała grupę
  - powinien jednoznacznie określać wygląd kodu, ale nie powinien być zbyt długi i szczegółowy (polecane są opracowania jednostronicowe)
- Standard dokumentowania zakłada, że samych komentarzy w kodzie jest jak najmniej:
  - klasy powinny być tak zaprojektowane by przeznaczenie poszczególnych metod było jasne, a samo działanie oczywiste.
  - dokumentowanie całych klas (poleca się zwięzłe opisanie przeznaczenia klasy i jej działania)
- XP stara się podtrzymać naturalne skłonności programistów do upiększania i ulepszania kodu

Stabilne i przewidywalne tempo pracy

- Swego rodzaju symbolem, znakiem rozpoznawczym XP, stało się wymaganie odpowiedniego rytmu pracy (np. w postaci 40-to godzinnego tygodnia faktycznej pracy)
- Zespoły programistów powinny być przyzwyczajone do stałej wydajności i stałego obciążenia.
- Częsta praca w nadgodzinach bardzo źle wpływa na morale i efektywność zespołu
  - może przytrafić się czasem jeden tydzień nieco większego obciążenia, ale dwa tygodnie mogą już oznaczać kłopoty z harmonogramem prac
  - ważne jest to, by ustalić konkretną, nienaruszalną granicę obciążenia grupy
- Rzeczywiste tempo pracy powinno być stale weryfikowane (mierzone na podstawie efektów pracy), co pozwala na uwiarygodnienie szacunków kolejnych iteracji i zakończenia projektu

Wspólna odpowiedzialność

- Zbiorowa praca nad kodem, to jednak nie tylko wspólne pisanie go, ale i wspólna odpowiedzialność za niego=> kolektywne kontrolowanie kodu - brak podziału kompetencji co do kontroli kodu
- Dzięki standardom kodowania każdy programista jest zaznajomiony z całym systemem, nawet jeśli go nie pisał
- W XP wszyscy są odpowiedzialni za dokonywanie poprawek kodu. Jeśli trzeba coś zmodyfikować nie ma problemu, bo poprawki może zrobić każdy. Częste przeorganizowywanie doprowadza kod do stanu dobrej przejrzystości, a gotowe procedury testujące zapewniają, ze poprawki nie doprowadza do katastrofy.
- XP preferuje umieszczenie całej grupy programistów w jednym pomieszczeniu, co ma pomagać w komunikacji i rozwijaniu życia grupy. Zostawia jednak dla każdego jego prywatną przestrzeń. Do pracy w parach powinny być wyznaczone oddzielne komputery.

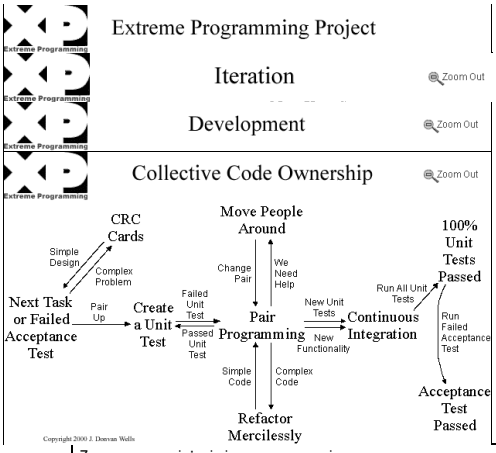
Małe wydania i możliwie częsta integracja

- Małe kroki - osiąga się je poprzez podział zadania na małe historie użytkownika
- Pojedynczy fragment kodu może być szybko przetestowany i złączony z reszta systemu; programista po wykonaniu każdego nowego fragmentu programu integruje go z systemem
- Najczęściej stosuje się jedna maszynę, na której w danej chwili może pracować jedna osoba zajmującą się łączeniem kodu
- Małe wydania to także akceptacje powstałego systemu przez klienta; dzięki ciągłym łączeniu zawsze istnieje sprawnie działająca wersja => informacja zwrotna, która ocenia czy zespół realizuje to, o co chodziło
- Ciągłe łączenie jest ułatwione w XP dzięki prostym projektom, ciągłym testom i wspólnej odpowiedzialności za kod

Stałe testowanie

- Ciągłe testowanie to podstawowe działanie podczas pisania programu
- Programista jeszcze przed napisaniem danej procedury tworzy kod (test jednostkowy), który ma ją testować (**test-first** programming):
  - dzięki temu podczas pisania właściwego kodu procedury lepiej rozumie rozwiązywany problem i zabezpieczy ją przed tymi możliwościami
- Pisanie procedury testowej nie powinno jednak trwać zbyt długo i nie powinna być ona zbyt rozbudowana.
- Zespół dąży do automatyzowania procedur testowych, które są uruchamiane po każdorazowym łączeniu kodu oraz po refaktoringu
  - gwarantuje to, że zmiany w innych częściach oraz integracja programu nie spowodowały nowych problemów
- Ponadto dla każdej historii, powstałej w grze planistycznej, klient określa test akceptacyjny. Kolejne wydanie nie jest zakończone dopóki test ten nie zostanie zdany pomyślnie.

Model procesu



Typowy dzień

- Poranne spotkania - zespół zaczyna prace od krótkiego (np. 15 minut) spotkania na stojąco obok stanowisk (możliwość dostępu do kodu)
  - ma na celu zakomunikowanie problemów, które się ostatnio pojawiły i rozważenie możliwych rozwiązań
- Programowanie - po ustaleniu zadań zespół przystępuje do pracy
  - zarówno zakres prac jak i skład par jest dobierany autonomicznie przez programistów a nie narzucany
  - wszyscy programują w parach, tworzą testy wyprzedzające i refaktoryzują w celu uzyskania dopracowanego kodu
  - programista może zwracać się do klienta o pomoc i dodatkowe wyjaśnienia
  - zaleca się kończenie realizacji zadań w ramach jednego dnia
- Integracja - po stworzeniu i przetestowaniu nowego elementu integruje się kod z właściwym systemem w repozytorium (na wydzielonym stanowisku) oraz uruchamia testy
  - jeżeli w wyniku błędów nie uda się zintegrować powstałego kodu do końca dnia zaleca się porzucenie stworzonego kodu i rozpoczęcie od nowa w kolejnym dniu
- Koniec dnia - po przepracowaniu ustalonego czasu, uśmiechnięty zespół idzie do domu

Kiedy nie warto/można stosować XP?

- Jeżeli wykorzystywany proces umożliwia produkcję oprogramowania w sposób satysfakcjonujący klientów i członków grupy projektowej
- W dużych projektach lub wymagających dużego stopnia formalności
  - niezbędny duży zespół - więcej niż kilkanaście osób i czas rzędu kilkunastu miesięcy; rozbudowana dokumentacja
- Brak dostępu do klienta - możliwość uzyskania szybko informacji zwrotnej na temat dostarczanych rozwiązań jest bardzo ograniczona
  - zespół projektowy i/lub klient pracuje w różnych, odległych lokalizacjach
- Zarządzanie częstymi zmianami nie jest możliwe a ich koszty są bardzo wysokie (np. ze względu na wykorzystane technologie)
- W innych przypadkach warto rozważyć zastosowanie XP, zwłaszcza gdy:
  - wymagania nie są ustabilizowane i mogą podlegać częstym zmianom
  - istnieje grupa projektowa składa się z deweloperów o wysokich umiejętnościach
  - wymagana jest wysoka efektywność pracy zespołu projektowego

Przygotowano na podstawie:

- *Extreme Programming: A gentle introduction* Don Wells, *[www.eXtremeProgramming.org]*, 2009.
- G. Młynarczyk, J. Kołodziej – *Inżynieria oprogramowania, wykład 11*, WSZiB w Krakowie
- *Extreme Programming - Zapewnienie skutecznej i wydajnej pracy programistów*, Krzysztof Kaczmarski, 2002