Zwinne wprowadzenie do TDD, BDD

Krzysztof Manuszewski

Agenda

- Kilka oczywistych obserwacji nt. komplikacji systemów
- Problem z wymaganiami
- Metodologie zwinne
- Degradacja oprogramowania
- Żeby być zwinnym trzeba ~~ćwiczyć~~ testować
- Jak wygląda test jednostkowy?
- Dlaczego pisanie testów jednostkowych jest trudne?
- Oprogramowanie sterowane "czymś tam" czyli modne skróty: TDD, BDD, ATDD
- Co dalej?

Arthur Charles Clarke:

KAŻDA WYSTARCZAJĄCO ZAAWANSOWANA TECHNOLOGIA JEST NIEROZRÓŻNIALNA OD MAGII.

Krzysztof Manuszewski:

TECHNOLOGIE NIE REDUKUJĄ LICZBY PROBLEMÓW - TECHNOLOGIE JE MNOŻĄ.

Postęp = i^5

Stale doskonalone technologie i narzędzia pozwalają budować coraz bardziej skomplikowane systemy

- Internet
- Interfejs użytkownika
- Integracja danych i usług
- Istniejące (przestarzałe) rozwiązania
- Itd.

Problemy z wytwarzaniem oprogramowania

- Nie istnieją działające, duże systemy*, które zaprojektowano i zrealizowano jako duże systemy. Wszystkie duże systemy powstały jako małe i ewoluowały do stanu obecnego

** Dotyczy projektów komercyjnych, nie dotyczy np. projektów rządowych ...*

Tradycyjne podejście

- Zbieranie i analiza wymagań
- Projektowanie
- Implementacja
- Integracja
- Testowanie+poprawianie błędów
- Wdrożenie

- Konserwacja

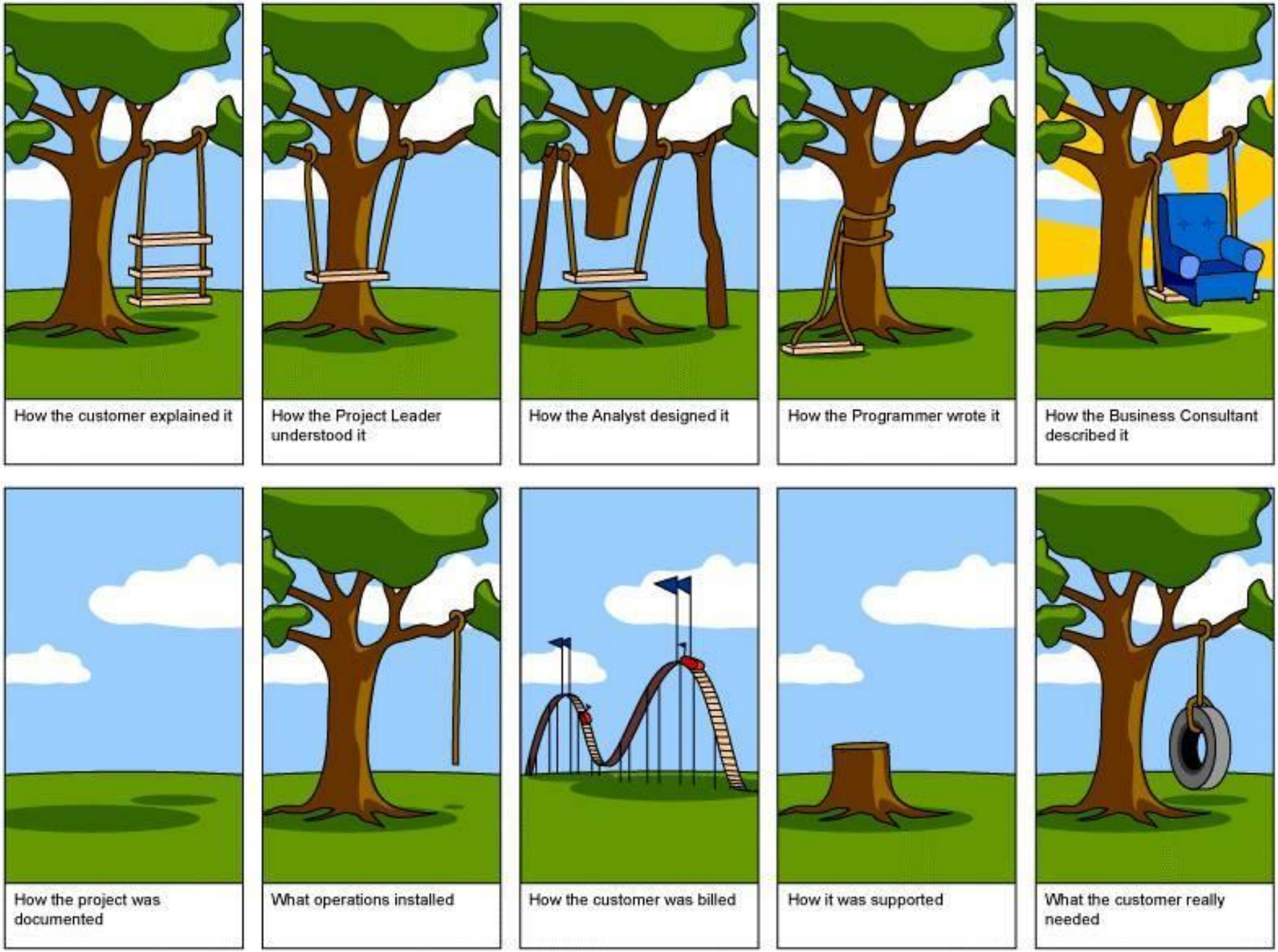
- Iterujemy, proszę Państwa! Iterujemy!
- Zasada 10

Quo vadis … programisto?

- Skąd wiadomo co robić:

W teorii:

- Kontrakt z klientem
- Architektura systemu
- Projekt klas/bazy danych


How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
What the customer really needed

Jak to jest w rzeczywistości

- Klient nigdy nie wie czego potrzebuje
- Nawet jeśli klient sądzi, że wie to na pewno się myli
- Jeśli klient przypadkiem wie czego potrzebuje to nie ma pojęcia jak to osiągnąć, mimo że mu sie może wydawać inaczej
- Klient wymyśla bezsensowne wymagania na poczekaniu
- Klient zawsze zapomina o kluczowych wymaganiach
- Klient nie rozumie przełożenia wymagań na cenę

Druga strona też nie jest bez winy…

- Analitycy notorycznie nie rozumieją czego chce klient czyli tzw. biznes*
- Projektanci nie są nieomylni!
- Zawsze pojawiają się drobne, nieprzewidziane modyfikacje - zwykle katastrofalne w skutkach!
- Dokumentacja ** nie jest aktualna. Nigdy!
- Jeśli dokumentacja jest aktualna to z pewnością znaczy..., że kod jest nieaktualny!

*W normalnych sytuacjach płaci ten kto z system będzie korzystał/potrzebuje go

**Diagram klas to też dokumentacja

Pomysł na to jak sobie z tym poradzić to z grubsza...

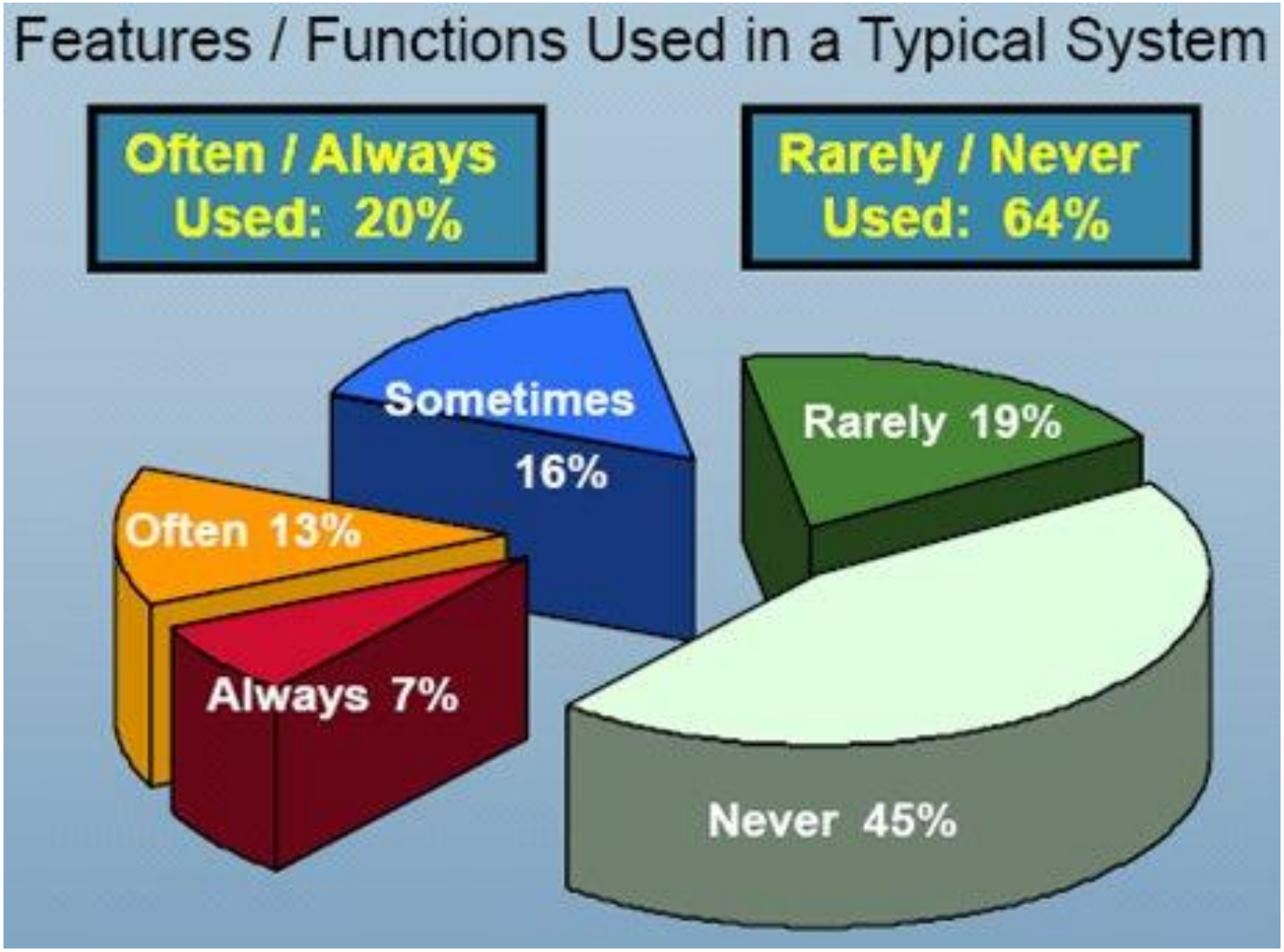
Manifest programowania zwinnego

Odkrywamy nowe metody programowania dzięki praktyce w programowaniu i wspieraniu w nim innych.
W wyniku naszej pracy, zaczęliśmy bardziej cenić:

Ludzi i interakcje od procesów i narzędzi
Działające oprogramowanie od szczegółowej dokumentacji
Współpracę z klientem od negocjacji umów
Reagowanie na zmiany od realizacji założonego planu.

Oznacza to, że elementy wypisane po prawej są wartościowe, ale większą wartość mają dla nas te, które wypisano po lewej..

Ciekawostka



na podstawie Standish Group Chaos Report

Ciekawostka

- "56% of defects caused by requirements" (*Source: James Martin, An Information Systems Manifesto*)
- "82% of the effort required to correct defects caused by requirements" (*Source: Martin & Leffinwell*)
- "44% of cancelled projects caused by issues related to requirements" (*Source: The Standish Group Chaos Report*)

Co z tego wynika

- Klient musi widzieć postępy prac. Jak najczęściej. Np. co 2 tygodnie. Najlepiej dać mu co jakiś czas coś użytecznego
- Nie powinnismy zbyt długo zbierać wymagań – szkoda czasu. Klient i tak zmieni zdanie, albo zapomni o czymś, albo skończą mu sie pieniądze.
- Nie powinnismy przejmować sie wszystkimi znanymi wymaganiami i tak nie da rady ogarnąć wszystkiego* (LRM**).
- Nie da się przewidzieć/zaprojektować kształtu finalnego systemu na początku prac

* *AFAIK Superman jest zajęty*

** *the Last Responsible Moment*

Working software, customer collaboration

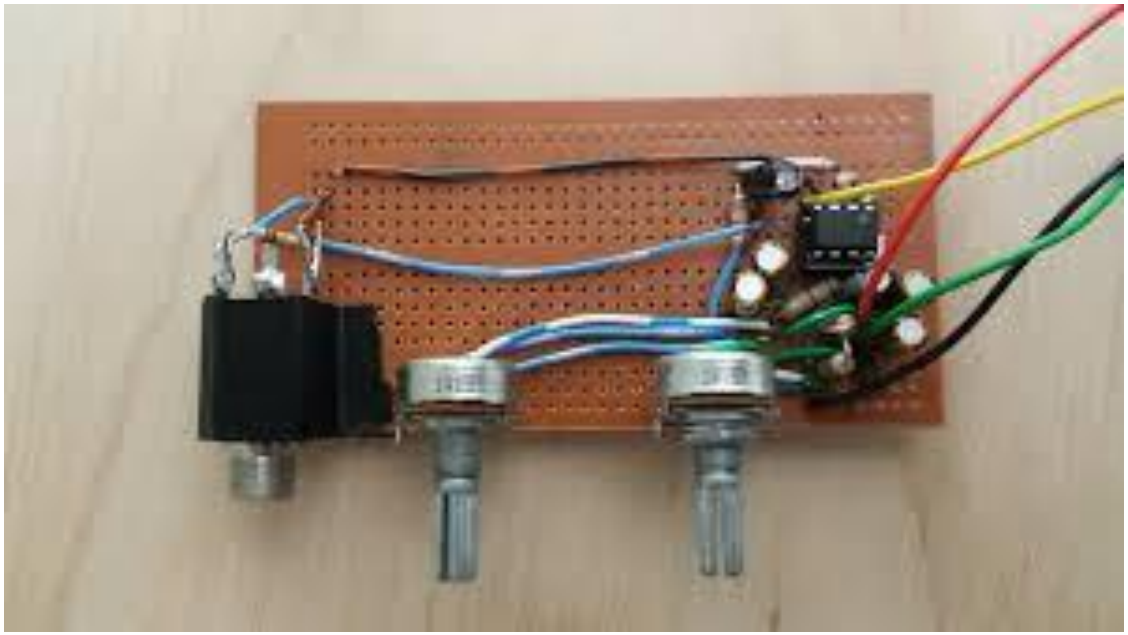
- Klient musi widzieć za co płaci
- Programista musi wiedzieć co robić

Czyli ...

... Wymagany jest PROTOTYP

2 oblicza prototypu

- budujemy prototyp jak najtaniej (i wyrzucamy)
- prototyp wejdzie do produkcji (być może po pewnych rozszerzeniach :-)



Ale skąd wiedzieć co dalej

- Pokazujemy
- Słuchamy
- Obserwujemy
- Mierzymy
- Ekperymentujemy

Agile w miniaturze

1. Zbieramy wymagania (User Stories) przez rozsądny czas
2. Wybieramy (np. tyle żeby wyrobic sie np. w 2 tygodnie):
   - User Stories, które maja dla klienta dużą wartość (pewnie i tak nie wszystko uda się zrealizować wiec niech ma choc tyle)
   - User Stories krytyczne dla projektu/architektury (niektóre założenia bardzo trudno zmienić. Inne US mogą być niewykonalne a zawsze lepiej anulować projekt wcześniej)
3. Robimy nową wersję – unikając niepotrzebnych decyzji (LRM)
4. Pokazujemy klientowi i słuchamy (tj. dodajemy i odrzucamy wymagania, zmieniamy priorytety itd.)
5. Jeśli klient chce płacić za więcej GOTO 2

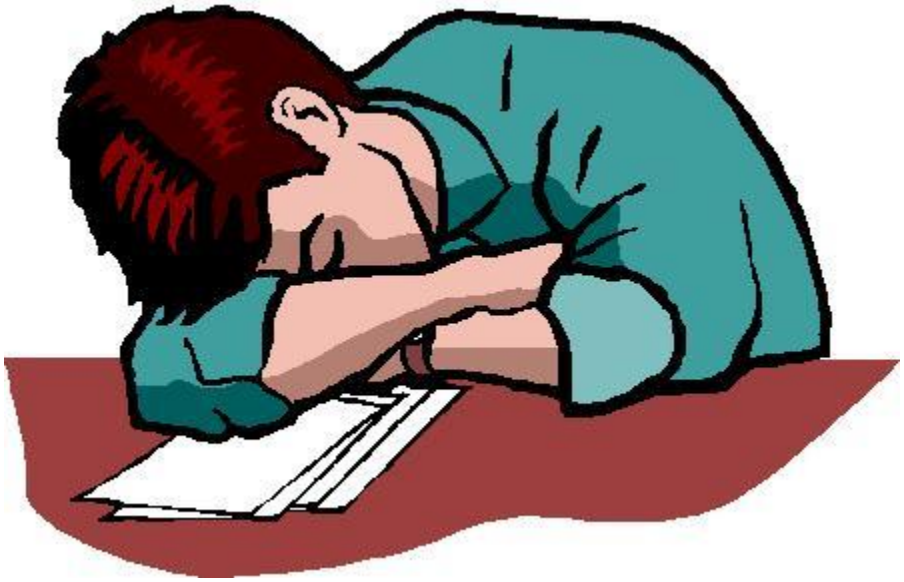
Agile w miniaturze

Gdzie jest haczyk ?

Problemem jest koszt krótkich iteracji

- Zbieranie i analiza wymagań tu
- Projektowanie tu
- Implementacja
- Integracja tu
- Testowanie+poprawianie błędów tu
- Wdrożenie tu

Kogo to obchodzi...



... miało być o programowaniu ...

Zagadka

Jaki system nie wymaga zmian?

Zagadka

Jaki system nie wymaga zmian?

ZALICZONY

Zmiany są

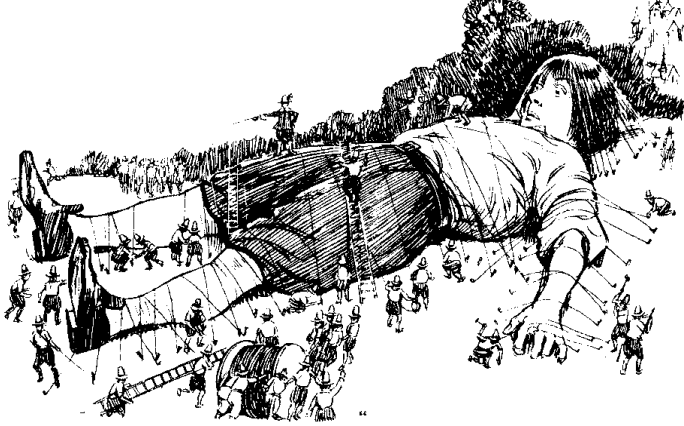
Zawsze ... .

Niejako z definicji system, który odniósł sukces (jest używany, przynosi dochody) rozwija się czyli wymaga zmian, poprawek, rozszerzeń itd.

Można nazwać to pielęgnacją, wersją nr. X.Y albo po prostu następną iteracją ...

ale oznacza to **zmiany**, **zmiany**.

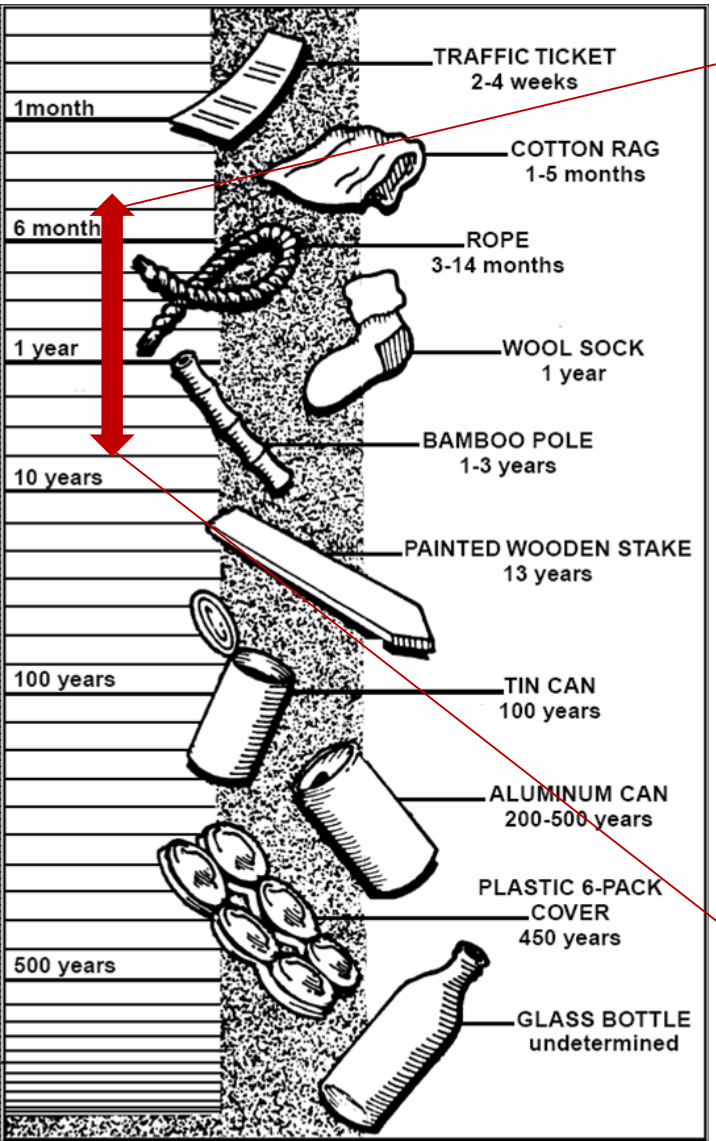
Kod się degraduje

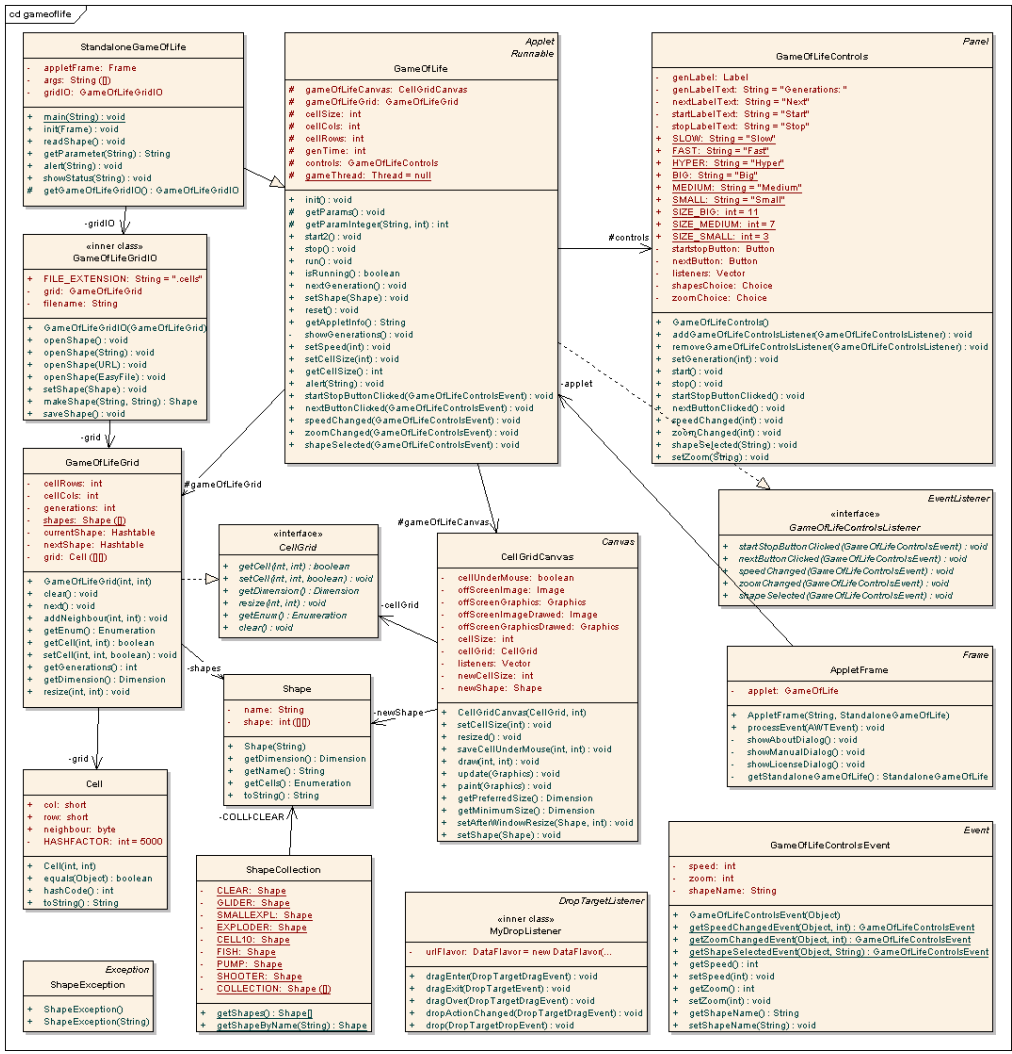


- Coraz trudniej zrozumieć kod*
- Coraz trudniej wprowadzać zmiany
- Coraz łatwiej popełnić błąd
- Skutki zmian są coraz trudniejsze do przewidzenia
- Testowanie wymaga coraz więcej czasu (nawet przy panowanych drobnych zmianach)
- Planowany czas zmian jest coraz dłuższy i mniej wiarygodny
- Coraz mniej osób chce dotykać kodu
- Klient ma coraz wiekszą ochotę sprawić sobie nowy system (albo przynajmniej nowy zespół programistów)

**Dla przypomnienia: dokumentacja jest nieaktualna*

Okres połowicznego rozpadu kodu





Warunkiem sukcesu (nie tylko w Agile)

jest kod DOBREJ jakości

Zasada skauta

zostaw kod w lepszym stanie niż go zastałeś

Refaktoryzacja: zmiany mające na celu poprawę jakości kodu bez zmian funkcjonalnych.

Refaktoryzacja

1. Nie należy łączyć zmian funkcjonalnych i refaktoryzacji
2. Refaktoryzacja powinna być realizowana małymi krokami

Problem: zmiany w działającym kodzie często powodują, że ten przestaje działać...

- *Refaktoryzacja może implikować dodatkowe testowanie (koszt, czas), aby tego uniknąc trzeba zautomatyzować testy*

TESTING

I FIND YOUR LACK OF TESTS DISTURBING.

Test jednostkowy

- Test jednostkowy jest to automatyczny fragment kodu uruchamiający i weryfikujący poprawność wykonania pewnego aspektu kodu produkcyjnego
- Zestaw testów jednostkowych powinien być utrzymywany (i uruchamiany) przez cały czas rozwoju systemu
- Testy mogą być uruchamiane pojedynczo lub masowo przez każdego członka zespołu, w czasie automatycznych buildów itd. Są częścią projektu, ale nie są dostarczane do klienta.

Po co pisać testy jednostkowe

- Ułatwiają znajdowanie błędów
- Ułatwiają zrozumienie kodu (dokumentacja)
- Ułatwiają utrzymanie kodu (modyfikacje)
- Ułatwiają pisanie kodu (TDD)

Jaki powinien być test jednostkowy?

- Powinien być łatwy w pisaniu
- Powinien być łatwy w uruchamianiu
- Powinien być łatwy w diagnostyce
- Powinien być szybki
- Powinien być niezależny
- Powinien być jednostkowy
- Powinien być łatwy w utrzymaniu

Narzędzia

- Frameworki: XUnit (Nunit, MSTest, MBUnit), Mspec
- Narzedzia uruchomienowe: VS, R#, TestDiven.Net, NCrunch
- Narzędzia do produkcji namiastek: RhinoMock, Moq
- Frameworki DI
- Narzedzia do oceny pokrycia kodu (VS, DotCover, NCover)

Co w tym trudnego?

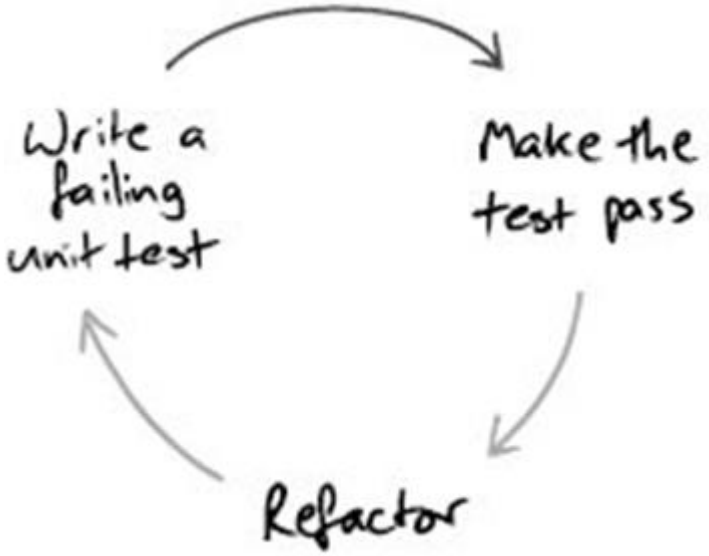
- Test: test powinien być dobrej jakości (krótki, przejrzysty)
- Skala: testów może być tyle co kodu - pisanie, a przedewszystkim utrzymanie testów
  - rak testów
  - spektakularne problemy
- Kod: sam w sobie jest trudny
- Jednostkowość czyli czy można … za-mock-ować wszechświat

Podejścia do pisania testów

- Trudno pisać testy do istniejącego kodu!
- Testy piszemy po (zaraz po) napisaniu kodu – w ten sposób możemy kod stosunkowo łatwo zmienić, zawsze należy sprawdzić czy test upada
- Testy piszemy przed kodem (TDD/BDD)

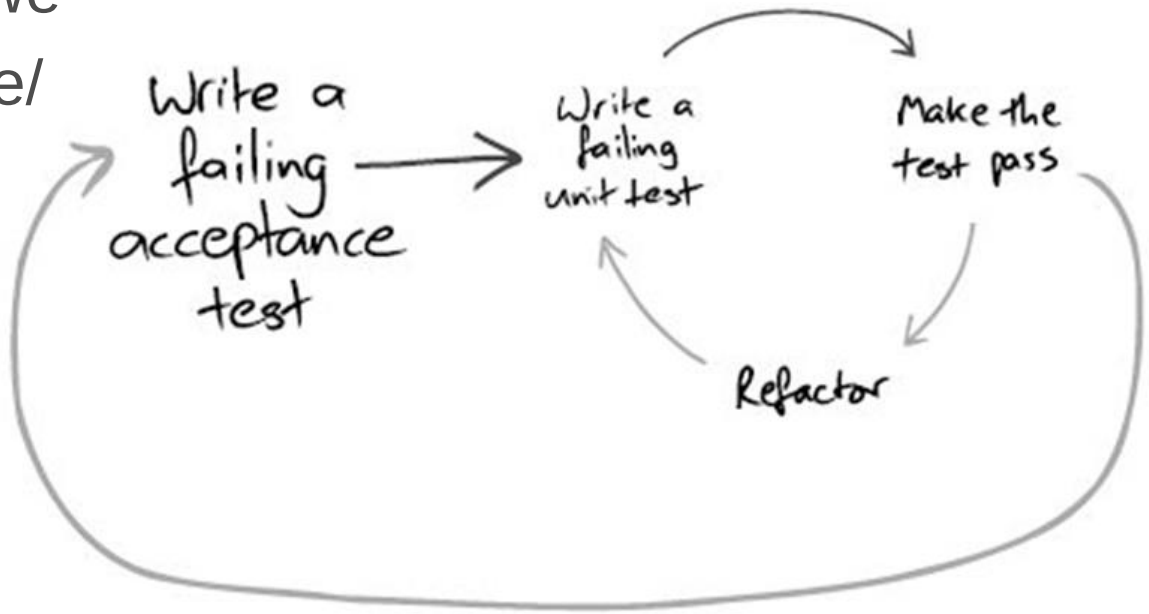
Test Driven Development (Design)

- Oznacza, że najpierw piszemy testy a potem kod
- Red -> Green ->Refactor



Behaviour Driven Development (Design)

- Główny nacisk jest kładziony na to by test odpowiadał konkretnemu zachowaniu biznesowemu.
- Wyznawcy (☺) starają się podkreślać, że nie tyle piszą test co specyfikują zachowanie.
  - Testy jednostkowe
  - Testy systemowe/ integracyjne/ akceptacyjne

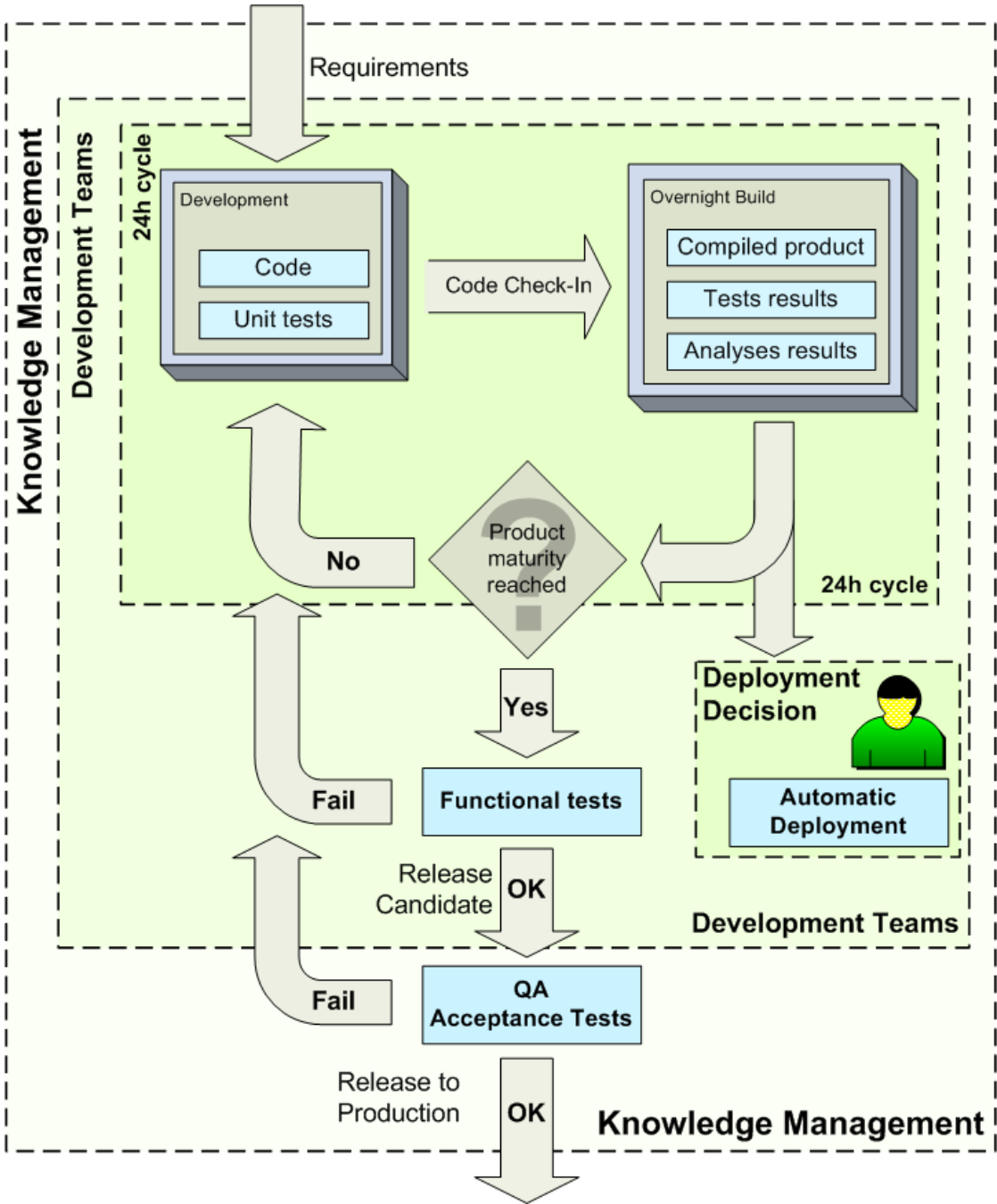


Test testowi nie równy

- Testy jednostkowe
- Testy integracyjne
- Testy systemowe
- Testy end-to-end
- Testy akceptacyjne

Co dalej?

- Continious Integration
- Continois Delivery



Co dalej?

- Automatyczne testy akceptacyjne

User Story ?

As a potential customer

I want to search for books by a simple string

So that I can easily allocate books by something I remember from them

Kryteria ?

As result should be displayed items matched by title or author

- Co dokładnie znaczy "match"?
- Co z wieloma wyrazami w zapytaniu?
- Co wyświetlić jeśli książka została znaleziona po tytule i po autorze?

Communication Gap?

- *Wikipedia: „**Abstraction** is a process by which higher concepts are derived from the usage and classification of literal ("real" or "concrete") concepts, first principles, or other methods…”*

Przykłady są precyzyjne?

Given the following books

Author	Title	Price
Martin Fowler	Analysis Patterns	50.20
Eric Evans	Domain Driven Design	46.34
Ted Pattison	Inside Windows SharePoint Services	31.49
Gojko Adzic	Bridging the Communication Gap	24.75

Scenario: Title should be matched

When I perform a simple search on 'Domain'

Then the book list should exactly contain book 'Domain Driven Design'

Scenario: Space should be treated as multiple OR search

When I perform a simple search on 'Windows Communication'

Then the book list should exactly contain books 'Inside Windows SharePoint Services', 'Bridging the Communication Gap'

Agile nie jedno ma imię

- **Lean software development**
- Scrum
- Canban

- XP

I niejednakowo wygląda:

- Stand Up, Planning Poker, Retrospective Meeting
- Pair Programming, Weak Code Ownership

To działa!!!

- Raczej dla małych projektów?
- Ale również w dużych firmach
  - **Amazon**: http://theagileexecutive.com/2009/07/20/scrum-at-amazon-guest-post-by-alan-atlas/
  - **Microsoft**: http://www.eweek.com/c/a/IT-Management/Microsoft-Lauds-Scrum-Method-for-Software-Projects/
  - **Yahoo**:http://agilesoftwaredevelopment.com/blog/artem/lessons-yahoos-scrum-adoption
  - **Salesforce.com**:http://www.slideshare.net/sgreene/salesforcecom-agile-transformation-agile-2007-conference
  - **IHS**

Źródła nt. Agile

- Poppendieck M&T
  - *Leading Lean Software Development: Results Are not the Point*
  - *Implementing Lean Software Development: From Concept to Cash*
  - *Lean Software Development: An Agile Toolkit*
- Highsmith, J. - *Agile Project Management*
- Beck K., Andres C. - *Extreme Programming Explained: Embrace Change*
- Cohn M.
  - *User Stories Applied: For Agile Software Development*
  - *Agile Estimating and Planning*

Screencasty nt. Scrum/Agile NDC 2009:

- Mike Cohn
  - *1 - Getting Agile With Scrum*
  - *2 - User Stories*
  - *3 - Agile Estimating*
  - *4 - Agile Planning*
  - *5 - Leading a Self-Organizing Team*
  - *6 - Prioritizing Your Product Backlog*
- Craig Larman
  - *The Toyota House Way for Large-Scale Lean Development*
  - *Agile Architecting*

Agile nie jedno ma imię

- **Lean software development**
- Scrum
- Canban

- XP

I niejednakowo wygląda:

- Stand Up, Planning Poker, Retrospective Meeting
- Pair Programming, Weak Code Ownership

Screencasty nt. Scrum/Agile/Solid

- Mike Cohn
  - NDC 2009:*1 - Getting Agile With Scrum, 2 - User Stories, 3 - Agile Estimating, 4 - Agile Planning ,5 - Leading a Self-Organizing Team*
  - *6 - Prioritizing Your Product Backlog*
- Craig Larman
  - NDC 2009: *The Toyota House Way for Large-Scale Lean Development*
- *Robert Martin*
  - NDC 2009: *S.O.L.I.D Principles of OO class design, Clean Code III – Functions*

Ksiazki

- *Robert Martin*
  - *Clean Code*
- Kent Beck
  - Test Driven Development: By Example
- *Roy Osherove*
  - The Art of Unit Testing
- Steve Freeman, Nat Pryce
  - Growing Object-Oriented Software Guided by Tests
- *Mary & Tom Poppendieck*
  - Lean Software Development
  - Leading Lean Software Development

Credits

- http://www.projectcartoon.com/#sthash.vmO9QDjo.dpuf
- https://omowizard.files.wordpress.com/2011/04/elephant_team.jpg
- https://www.standishgroup.com/
- http://django-notes.blogspot.com/2012/06/growing-object-oriented-software-guided.html
- http://brodzinski.com/2009/04/when-unit-testing-doesnt-work.html
- http://www.kidney-bingos.demon.co.uk/LPW2009/
- https://www.flickr.com/photos/kino/429412983
- http://www.thedeliverersseries.com/2014_04_01_archive.html
- http://www.onlyimage.com/stock-photo/dollars-under-zoom-inflation-has-reduced-cost-of-dollar-1049764
- https://pelhamnicholas.wordpress.com/2015/05/28/pre-amp-equilizer/