Inżynieria Programowania - Wstęp

Arkadiusz Chrobot

Katedra Systemów Informatycznych, Politechnika Świętokrzyska w Kielcach

Kielce, 5 marca 2020

Plan wykładu

Bibliografia (część, reszta na stronie)

- Ian Sommerville, „Inżynieria oprogramowania”, WNT, Warszawa, 2003
- Krzysztof Sacha, „Inżynieria oprogramowania”, PWN, Warszawa, 2010
- Mariusz Flasiński, „Zarządzanie projektem informatycznym”, PWN, Warszawa, 2006
- Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, „Wzorce projektowe. Elementy oprogramowania obiektowego wielokrotnego użytku“, Helion, Gliwice 2017
- Adam Roman, „Testowanie i jakość oprogramowania”, PWN, Warszawa, 2015
- Daunerin' Aboot Ian Sommerville's writing and photography
- Inżynieria oprogramowania

Motto

"If you look at software today, through the lens of the history of engineering, it's certainly engineering of a sort — but it's the kind of engineering that people without the concept of the arch did. Most software today is very much like an Egyptian pyramid with millions of bricks piled of top of each other, with no structural integrity, but just done by brute force and thousand of slaves."
Alan Kay

Wstęp

Inżynieria Oprogramowania

Inżyniera Oprogramowania (ang. *software engineering*) - dziedzina inżynierii, która obejmuje wszystkie aspekty tworzenia oprogramowania od początkowej fazy specyfikacji systemu, aż do jego wycofania z użycia.

Oprogramowanie

Oprogramowanie w ujęciu Inżynierii Oprogramowania jest każdą postacią zapisu programu komputerowego (kod, dokumentacja, dane konfiguracyjne).

Kryzys rozwoju oprogramowania

W latach 60 ubiegłego wieku pojawiły się systemy komputerowe trzeciej generacji. To wydarzenie zapoczątkowało powstawanie programów o dużym stopniu skomplikowania. Wkrótce okazało się, że brakuje wytycznych jak takie oprogramowanie tworzyć. W efekcie wiele przedsięwzięć związanych z produkcją oprogramowania kończyło się klęską. W roku 1968 pod patronatem NATO zwołano w Niemczech konferencję, której celem było znalezienie wyjścia z zaistniałej sytuacji. To właśnie tam powstał pomysł stworzenia inżynierii programowania.

Efekty

Do chwili obecnej nie udało się znaleźć metod, które gwarantowałyby sukces każdego projektu programistycznego. Sytuacja nadal jest poważna:

- NIST szacuje roczne straty pieniężne w USA związane z projektami programistycznymi na około 60 miliardów dolarów,
- według Standish Group „30% projektów programistycznych jest porzucanych, około połowa z nich przekracza zamierzony budżet, 60% jest uważanych za katastrofę przez organizacje, które je zapoczątkowały, a 9 z 10 jest kończonych za późno”,
- Peter G. Neumann - ”Illustrative Risks to the Public in the Use of Computer Systems and Related Technology”

Przyczyny

Niektóre z przyczyn:

- przedsięwzięcia informatyczne mają często charakter innowacyjny,

Źródła:
Terence Parr - "Why writing software is not like engineering"
Chuck Allison - "Code Quality"

Przyczyny

Niektóre z przyczyn:

- przedsięwzięcia informatyczne mają często charakter innowacyjny,
- produkt inżynierii oprogramowania ma charakter niematerialny,

Źródła:
Terence Parr - "Why writing software is not like engineering"
Chuck Allison - "Code Quality"

Przyczyny

Niektóre z przyczyn:

- przedsięwzięcia informatyczne mają często charakter innowacyjny,
- produkt inżynierii oprogramowania ma charakter niematerialny,
- częste zmiany wymagań w trakcie realizacji projektu,

Źródła:
Terence Parr - "Why writing software is not like engineering"
Chuck Allison - "Code Quality"

Przyczyny

Niektóre z przyczyn:

- przedsięwzięcia informatyczne mają często charakter innowacyjny,
- produkt inżynierii oprogramowania ma charakter niematerialny,
- częste zmiany wymagań w trakcie realizacji projektu,
- złe zarządzanie projektem,

Źródła:
Terence Parr - "Why writing software is not like engineering"
Chuck Allison - "Code Quality"

Przyczyny

Niektóre z przyczyn:

- przedsięwzięcia informatyczne mają często charakter innowacyjny,
- produkt inżynierii oprogramowania ma charakter niematerialny,
- częste zmiany wymagań w trakcie realizacji projektu,
- złe zarządzanie projektem,
- „wojna wykładników".

Źródła:
Terence Parr - "Why writing software is not like engineering"
Chuck Allison - "Code Quality"

Informatyka a inżynieria oprogramowania

Informatyka (ang. *computer science*)

Jak tworzyć efektywne oprogramowanie? (algorytmy, struktury danych, złożoność obliczeniowa, języki programowania, paradygmaty programowania)

Inżynieria oprogramowania (ang. *software engineering*)

Jak efektywnie tworzyć oprogramowanie? (reguły zarządzania projektami, metody radzenia sobie ze złożonością projektów, architektura oprogramowania, dokumentacja, koszty produkcji, testowanie, niezawodność, pielęgnacja)

Cechy oprogramowania

Jakie właściwości powinno mieć oprogramowanie:

- zdolność do pielęgnacji (ang. *maintenance*),

Cechy oprogramowania

Jakie właściwości powinno mieć oprogramowanie:

- zdolność do pielęgnacji (ang. *maintenance*),
- niezawodność,

Cechy oprogramowania

Jakie właściwości powinno mieć oprogramowanie:

- zdolność do pielęgnacji (ang. *maintenance*),
- niezawodność,
- efektywność,

Cechy oprogramowania

Jakie właściwości powinno mieć oprogramowanie:

- zdolność do pielęgnacji (ang. *maintenance*),
- niezawodność,
- efektywność,
- użyteczność.

Paradygmaty programowania

Istnieją następujące podstawowe paradygmaty (programowania):

1. paradygmat imperatywny
   1. programowanie strukturalne (Pascal, C, Perl),
   2. programowanie obiektowe (C++, Java),
2. paradygmat deklaratywny
   1. programowanie funkcyjne (Erlang, LISP, JavaScript),
   2. programowanie logiczne (Prolog).

Paradygmaty tworzenia oprogramowania

Modeli tworzenia oprogramowania jest kilka, między innymi:

- model kaskadowy,
- tworzenie ewolucyjne,
- tworzenie z użyciem metod formalnych,
- tworzenie z użyciem wielokrotnym,
- metody zwinne.

Tworzenie oprogramowania

Tworzenie oprogramowania jest procesem zmierzającym do wytworzenia produktu, jakim są programy komputerowe. Przebieg tego procesu jest zależny od rodzaju tworzonego oprogramowania. Można jednak wyróżnić cztery zasadnicze czynności, które występują zawsze w tym procesie:

1. specyfikacja oprogramowania - prostsza w przypadku *oprogramowania powszechnego*, dużo trudniejsza w przypadku *oprogramowania specjalistycznego*,
2. implementowanie oprogramowania,
3. zatwierdzanie oprogramowania,
4. ewolucja oprogramowania.

Model procesu tworzenia oprogramowania

Model procesu (paradygmat) w sposób uproszczony opisuje przebieg procesu względem przyjętego punktu widzenia. Można je sklasyfikować jako:

- model przepływu prac,
- model przepływu danych,
- model rola-akcja.

Model kaskadowy

Paradygmat kaskadowy (ang. *waterfall*) jest pierwszym modelem procesu tworzenia oprogramowania jaki powstał. Podstawą przy jego tworzeniu były doświadczenia z innych dziedzin inżynierii. Obejmuje on pięć głównych czynności, które wykonywane są tylko raz w procesie tworzenia: *definiowanie i analizowanie wymagań*, *projektowanie oprogramowania*, *implementację i testowanie jednostek*, *integrację i testowanie systemu* oraz *wdrożenie i pielęgnację*. Model ten porządkuje prace nad oprogramowaniem, ale jest mało elastyczny i kosztowny jeśli zachodzą zmiany wymagań w trakcie projektu. Sprawdza się w projektach informatycznych realizowanych jako część projektu z innej dziedziny inżynierii.

Tworzenie ewolucyjne

Tworzenie ewolucyjne polega na stworzeniu prototypu oprogramowania na podstawie pierwszej wersji specyfikacji, następnie przekazaniu go użytkownikowi, zebraniu opinii i udoskonaleniu na ich podstawie wstępnej wersji oprogramowania. Ten proces powtarzany jest tak długo, aż powstanie ostateczna wersja systemu. Można wyróżnić dwie podklasy tego modelu:

1. tworzenie badawcze,
2. tworzenie z porzuceniem.

Ten paradygmat tworzenia oprogramowania jest bardziej efektywny niż model kaskadowy, a produkt wytworzony na jego bazie bardziej odpowiada potrzebom użytkowników. Niestety struktura oprogramowania tworzonego tą metodą może być zła. Proces wytwórczy nie jest przy takim podejściu widoczny. Wymagane są także specjalne narzędzia i techniki.

Tworzenie z użyciem metod formalnych

W modelu tworzenia z użyciem metod formalnych wymagania odnośnie opracowywanego oprogramowania zapisywane są w postaci formuł matematycznych. Przejście od wymagań do działającego programu dokonywane jest na zasadzie ściśle zdefiniowanych przekształceń matematycznych. Zaletą tego modelu jest to, że oprogramowanie opracowywane przy jego pomocy jest bardzo niezawodne i zawiera małą liczbę błędów. Jednakże nie każde oprogramowanie może być tworzone przy użyciu tego paradygmatu. Dodatkowo jest to proces dosyć kosztowny.

Tworzenie z użyciem wielokrotnym

W tym modelu zakłada się ponowne wykorzystanie komponentów oprogramowania, które zostały wcześniej wykonane lub zakupione. Zaletą tego podejścia jest względnie mały koszt tworzenia oprogramowania. Niestety niekiedy konieczne są kompromisy między funkcjonalnością dostępnych komponentów, a specyfikacją. Oznacza to, że oprogramowanie powstałe z użyciem tego modelu nie zawsze odpowiada wymaganiom użytkownika.

Metody zwinne

Główną wadą „klasycznych" metod tworzenia oprogramowania są bardzo sztywne procedury, które przy często zmieniających się wymaganiach powodują spowolnienie procesu twórczego i prowadzą do generowania rozbudowanej dokumentacji, zamiast działającego oprogramowania. W odpowiedzi na te problemy powstał szereg tak zwanych *metod zwinnych* (ang. *agile development*), w których nacisk kładziony jest na dostarczeniu użytkownikowi działającego oprogramowania, a nie bezużytecznej dokumentacji. Jedną z tych metod jest programowanie ekstremalne (ang. *extreme programming* - - XP). Główne postulaty tego modelu to ciągła rewizja kodu (programowanie w parach), ciągłe testowanie (zarówno po stronie programistów, jak i użytkowników), prostota projektu, ciągła praca nad definiowaniem i udoskonalaniem architektury systemu, integrowanie i testowanie komponentów kilkukrotnie w ciągu dnia, krótkie okresy wydań, intensywna współpraca z klientem. Metody zwinne sprawdzają się w przypadku małych projektów, o nieprecyzyjnie ustalonych wymaganiach wstępnych i realizowanych przez niewielkie zespoły. Wadą tych metod jest nieprecyzyjna dokumentacja procesu wytwórczego, która może utrudniać przyszłą konserwację oprogramowania.

Metody inżynierii oprogramowania

Celem inżynierii oprogramowania jest opracowanie metod porządkujących proces tworzenia programów komputerowych, tak aby produkt końcowy był wysokiej jakości, a wytwarzanie było ekonomiczne. Metody inżynierii oprogramowania pozwalają tworzyć modele (najczęściej graficzne) programów, które mogą być wykorzystywane jako specyfikacja i projekt systemu. Przykładem takich metod są metody strukturalne, metody obiektowe, Unified Modeling Language (UML).

Koszty tworzenia oprogramowania

Koszty budowy oprogramowania zależą od jego typu oraz od rodzaju zastosowanego modelu wytwórczego. Generalnie najbardziej kosztowną fazą (bez uwzględniania pielęgnacji) jest testowanie, które w typowych projektach pochłania około 40% całkowitych kosztów, a w przypadku systemów krytycznych nawet 50%. Całkowite koszty konserwacji oprogramowania (rozwoju po wdrożeniu) mogą być znacznie wyższe niż koszty tworzenia od podstaw.

Computer-Aided Software Engineering

Aby usprawnić tworzenie oprogramowania opracowano szereg narzędzi programowych, które wspomagają niektóre z czynności wykonywanych w trakcie tego procesu. Wszystkie te programy należą do kategorii *Computer-Aided Software Engineering* - CASE. Należą do nich programy wspomagające modelowanie (np. Rational Rose), generujące kod, zarządzające wersjami (np. Mercurial), testujące i inne.

Wyzwania

Najczęściej napotykanymi przez inżynierów oprogramowania wyzwaniami są:

- systemy odziedziczone,
- różnorodność systemów,
- terminy wdrożenia.

Podsumowanie

Oprogramowanie komputerowe staje się coraz bardziej skomplikowane, a od jego działania zależy coraz więcej aspektów ludzkiego życia. Nie znamy jeszcze metod, które pozwalałby tworzyć niezawodne, efektywne i pozbawione błędów oprogramowanie w sposób tani i skuteczny. Stosowanie metod inżynierii programowania nie gwarantuje sukcesu procesu wytwórczego, jednakże ich zignorowanie gwarantuje, że ten proces zakończy się katastrofą.

Pytania

?

KONIEC

Dziękuję Państwu za uwagę.