Inżynieria oprogramowania 1 — Wprowadzenie

Arkadiusz Chrobot

Katedra Systemów Informatycznych, Politechnika Świętokrzyska

Kielce, 2 października 2024

Notatki

Plan

Notatki

Bibliografia

- Ian Sommerville
  Inżynieria oprogramowania,
  PWN, Warszawa, 2020,
  strona `WWW autora`: https://iansommerville.com.
- Krzysztof Sacha
  Inżynieria oprogramowania
  PWN, Warszawa, 2010.
- David Farley
  Nowoczesna inżynieria oprogramowania,
  Helion, Gliwice, 2023.
- Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides
  Wzorce projektowe: Elementy oprogramowania obiektowego wielokrotnego użytku
  Helion, Gliwice, 2017.

…i wiele innych wymienionych na stronie przedmiotu oraz na slajdach!

Notatki

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 on top of each other, with no structural integrity, but just done by brute force and thousand of slaves.”
Alan Kay

Notatki

Wprowadzenie

Definicja IEEE 610.2 (tłumaczenie własne)

„Inżynieria oprogramowania jest zastosowaniem systematycznego, zdyscyplinowanego i mierzalnego podejścia do wytwarzania, wdrażania i utrzymywania (ang. *maintenance*) oprogramowania; tzn. zastosowaniem inżynierii do oprogramowania i badaniem takiego podejścia."

Oprogramowanie

Oprogramowanie — programy komputerowe i inne *artefakty* powstające w procesie wytwarzania oprogramowania.

Robocza definicja inżynierii autorstwa Davida Farley'a

Inżynieria jest zastosowaniem empirycznego, naukowego podejścia w celu znajdowania wydajnych i ekonomicznych rozwiązań praktycznych problemów.

Notatki

Kryzys oprogramowania

W latach 60. poprzedniego wieku, pojawiła się na rynku trzecia generacja systemów komputerowych, która otwierała możliwość tworzenia złożonego oprogramowania. Wkrótce jednak okazało się, że jest to bardzo skomplikowane. Nie znano wówczas sposobu na wytwarzanie złożonego, wydajnego i niezawodnego oprogramowania. Większość projektów informatycznych, które miały taki cel, kończyła się niepowodzeniem, wręcz klęską. Sytuację tę zaczęto określać mianem *kryzysu oprogramowania*. Aby zaradzić tym problemom zorganizowano w 1968 roku, pod auspicjami NATO, konferencję, która miała miejsce w Garmisch, w Niemczech. W ramach tego właśnie wydarzenia formalnie powstała idea inżynierii oprogramowania.

Notatki

Rezultaty

Zasady wytwarzania oprogramowania, które gwarantowałby pełne powodzenie takiego przedsięwzięcia, nie są jeszcze znane. Sytuacja w branży, zdaniem wielu zaangażowanych osób, nie uległa znaczącej zmianie:

- Joel Mathis, „Starliner: What went wrong?" (2024),
- Gregory Travis, „How The Boeing 737 MAX Disaster Looks To a Software Developer (2024),
- „Potężna awaria systemów Ministerstwa Sprawiedliwości. Utrudnienia trwają już ponad 20 godzin" (2024),
- Simon Sharwood, „CrowdStrike file update bricks Windows machines around the world" (2024),
- *The Risks Digest*,
- The Standish Group Chaos Report: "19% projektów związanych z oprogramowaniem zakończyło się niepowodzeniem, 50% ukończono z problemami, 31% zakończono z powodzeniem" (2020).

Notatki

Przyczyny

Wybrane przyczyny problemów z oprogramowaniem:

- projekty informatyczne dotyczą zazwyczaj zagadnień innowacyjnych,
- produkt, jakim jest oprogramowanie, jest niematerialny,
- wymagania względem oprogramowania ulegają częstym zmianom,
- złe zarządzanie projektami,
- „wojna wykładników".

Sources:

Terence Parr
Why Writing Software Is Not Like Engineering
http://www.cs.usfca.edu/~parrt/doc/software-not-engineering.html

Chuck Allison
Code Quality
[Źródło nie jest już dostępne]

Notatki

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? (zarządzanie projektem, złożone projekty, podział prac, projektowanie, architektura oprogramowania, dokumentacja, koszty produkcji, testowanie, niezawodność, utrzymanie).

Notatki

Pożądane cechy oprogramowania

Wybrane cechy oprogramowania wysokiej jakości:

- łatwość utrzymania (ang. *maintainability*),
- niezawodność,
- dostępność,
- efektywność,
- funkcjonalność (ang. *usability*).

Notatki

Paradygmaty programowania

Niektóre z głównych paradygmatów programowania:

1. imperatywne
   1.1 proceduralne/strukturalne (Pascal, C, Perl),
   1.2 obiektowe (C++, Java),
2. declarative
   2.1 funkcyjne (Erlang, Elixir, LISP, JavaScript),
   2.2 logiczne (Prolog).

Notatki

Paradygmaty tworzenia oprogramowania

Istnieje wiele modeli wytwarzania oprogramowania, do których należą:

- model kaskadowy (ang. *waterfall*),
- tworzenie iteracyjne,
- tworzenie z użyciem metod formalnych,
- integracja i konfiguracja,
- metody zwinne.

Notatki

Rozwój oprogramowania

Rozwój oprogramowania to proces wytwórczy, którego produktem jest program lub programy komputerowe. Przebieg tego procesu jest zależny od specyfiki tworzonego oprogramowania, ale w każdym przypadku można wyróżnić pięć głównych faz:

1. specyfikacja — prostsza w przypadku oprogramowania *powszechnego użytku* i dużo trudniejsza dla *oprogramowania specjalizowanego*,
2. projektowanie i implementacja,
3. zatwierdzanie i weryfikacja,
4. utrzymanie,
5. wycofanie z użycia.

Notatki

Model procesu rozwoju oprogramowania

Model (lub paradygmat) procesu przedstawia go w uproszczony sposób z określonej perspektywy. Modele zazwyczaj należą do jednej z trzech kategorii:

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

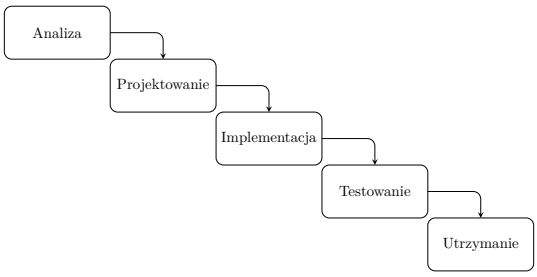
Notatki

Model kaskadowy

Model kaskadowy (ang. *Waterfall Paradigm*) jest najstarszym modelem rozwoju oprogramowania. Został on zaproponowany w 1970 przez dra Winstona W. Royce'a i jest adaptacją podobnych modeli stosowanych w innych dziedzinach inżynierii. W oryginalnej koncepcji tego modelu wyróżniono pięć czynności, które są wykonywane tylko raz w całym procesie: *określanie i analiza wymagań*, *projektowanie oprogramowania i systemu*, *implementacja i testowanie jednostkowe*, *integracja i testowanie systemowe*, *wdrożenie i utrzymanie*. Zaletą tego modelu jest uporządkowanie prac nad oprogramowaniem, ale dowolna zmiana w wymaganiach generuje znaczące koszty — proces należy zacząć od początku. Jest on odpowiedni dla przedsięwzięć informatycznych realizowanych w ramach większego projektu inżynierskiego, np. tworzenia nowego modelu samochodu. W takim projekcie zazwyczaj wymagania są dobrze znane i rzadko ulegają zmianom.

Notatki

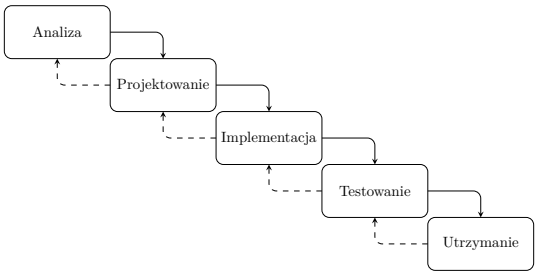
Model kaskadowy



Rysunek: Uproszczony model kaskadowy

Notatki

Model kaskadowy



Rysunek: Uproszczony model kaskadowy

Notatki

Tworzenie iteracyjne

W tworzeniu iteracyjnym (ang. *iterative development*), produkt (oprogramowanie) jest rozwijany stopniowo. Najpierw przygotowywana jest początkowa wersja, oparta na wczesnej specyfikacji i dostarczana użytkownikom. Po uzyskaniu informacji zwrotnych zespół wytwórczy poprawia/ulepsza produkt i odsyła go do ponownej ewaluacji. Proces ten powtarza się, aż do uzyskania pożądanej wersji oprogramowania. W tym podejściu fazy, które są wspólne dla wszystkich modeli wytwarzania oprogramowania, przeplatają się, zamiast następować po sobie. Dodatkowo, ten model można połączyć z prototypowaniem Prototyp może zostać przygotowany jako eksperymentalna implementacja nowych funkcji i włączony do tworzonego produktu lub porzucony. W tym ostatnim przypadku jest on tworzony po to, aby odkryć nowe wymagania lub uściślić istniejące. Tworzenie iteracyjne jest odpowiednie dla projektów, w których wymagania się często zmieniają. Wynikowy produkt zazwyczaj lepiej odpowiada potrzebom użytkowników, ale jego wewnętrzna struktura może być gorzej zaprojektowana i wykonana. Proces wytwórczy może być trudny do śledzenia i zarządzania.

Notatki

Metody formalne

Metody formalne wprowadzają aparat matematyczny do procesu wytwarzania oprogramowania. Wymagania są zapisywane w określonych *językach specyfikacji*. Najczęściej są to *języki formalne*, o ścisłej składni i semantyce. Oprogramowanie jest *wyprowadzane* (ang. *derived*) ze specyfikacji, a jego poprawność weryfikowana przy pomocy matematycznych dowodów. Produkt końcowy jest wysokiej jakości i niezawodny. Niestety, nie każdy rodzaj oprogramowania może być tworzony z użyciem metod formalnych. Dodatkowo, ich użycie w projekcie generuje dodatkowe, nie zawsze uzasadnione koszty. Współcześnie są one stosowane głównie w projektach dotyczących oprogramowania, którego krytyczną cechą jest bezpieczeństwo (ang. *safety-critical*).

Dalsza lektura:

Gerald O'Regan
Concise Guide to Formal Methods
Springer International Publishing AG, Cham, Switzerland, 2017

Notatki

Metody formalne

„Ten years ago, researchers into formal methods (and I was the most mistaken among them) predicted that the programming world would embrace with gratitude every assistance promised by formalisation to solve the problems of reliability that arise when programs get large and more safety-critical. Programs have now got very large and very critical — well beyond the scale which can be comfortably tackled by formal methods. There have been many problems and failures, but these have nearly always been attributable to inadequate analysis of requirements or inadequate management control. It has turned out that the world just does not suffer significantly from the kind of problem that our research was originally intended to solve."

Tony Hoare w 1995 roku

Notatki

Integracja i konfiguracja

To podejście do wytwarzania oprogramowania opiera się na wielokrotnym wykorzystaniu komponentów, które zostały opracowane w poprzednich projektach, zakupione lub są dostępne jako oprogramowanie otwartoźródłowe. W tej metodzie prace nad oprogramowaniem mogą postępować szybko i tanio, ale z koniecznością wprowadzenia pewnych kompromisów dotyczących funkcji udostępnianych przez produkt. Wynikowe oprogramowanie może nie odpowiadać w pełni potrzebom klientów.

Notatki

Metody zwinne

W 2001 roku kilku znanych w branży programistów stworzyło i podpisało Manifest programowania zwinnego (ang. *Manifesto for Agile Software Development*):
„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 dla nas mają te, które wypisano po lewej."*
Źródło:
https://agilemanifesto.org/iso/pl/manifesto.html
Manifest ten uzupełnia *Dwanaście zasad programowania zwinnego*.

Notatki

Metody zwinne

Opracowanie Manifestu programowania zwinnego i Zasad programowania zwinnego doprowadziło do powstania szeregu *metod* tworzenia oprogramowania, nazywanych krótko *metodami zwinnymi*. Również część metod odkrytych przez powstaniem manifestu i zasad zwinności jest zaliczanych do tego grona, do którego należą: *Lean Software Development*, PDCA (Plan — Do — Check — Act), *Kanban*, *programowanie ekstremalne* (ang. *Extreme Programming* — XP), *Feature-Driven Development*, *Dynamic System Development Method*, *Crystal*, *Scrum*. Te metody mogą być całkowicie lub częściowo łączone z innymi, niezaliczanymi do zwinnych. Głównym celem metody zwinnych jest dostarczenie klientowi działającego oprogramowania i zmniejszenie dodatkowej pracy wykonywanej w projekcie. Te metody najlepiej pasują do małych projektów, z niewielkimi zespołami wytwórczymi i niesprecyzowanymi wymaganiami. Nie skalują się dobrze na większe projekty.

Notatki

Ewolucja metod zwinnych

W wyniku rozwoju metod zwinnych powstały trzy inne modele tworzenia oprogramowania:

CI/CD Continuous Integration/Continuous Delivery — ciągła integracja (ang. *Continuous Integration*) polega na tworzeniu oprogramowania *małymi przyrostami*, które są często, nawet klika razy dziennie, integrowane, kompilowane i testowane w zautomatyzowany sposób. Jeśli na końcu tego procesu oprogramowanie jest wdrażane, lub uzyskiwana jest jego wersja gotowa do wdrożenia, to ten proces nazywamy ciągłym wdrażaniem (ang. *Continuous Delivery*).

DevOps integruje rozwój oprogramowania z jego wdrażaniem, co pozwala inżynierom oprogramowania łatwiej uzyskać informacje zwrotne odnośnie do stanu i problemów związanych z ich produktem, po jego wdrożeniu. Wymaga większej automatyzacji niż CI/CD w zakresie wdrażania, konfigurowania i monitorowania oprogramowania. DevOps jest szczególnie przydatne w środowiskach chmurowych.

DevSecOps uzupełnia DevOps o kwestie bezpieczeństwa podczas rozwoju i utrzymania oprogramowania, a także o automatyzację procesów związanych z jego bezpieczeństwem.

Notatki

Narzędzia i metody inżynierii oprogramowania

Inżynieria oprogramowania nie tylko tworzy modele procesu rozwoju oprogramowania, ale również dostarcza środki (nazywane *metodami inżynierii oprogramowania*) i narzędzia pomagające je stosować. Metody te pozwalają inżynierom oprogramowania tworzyć modele programów komputerowych, które stanowią ich specyfikację. Jako przykłady można podać UML (*Unified Modeling Language*), analizę zorientowaną obiektowo, analizę strukturalną.
Narzędzia inżynierii oprogramowania początkowo określano mianem CASE (ang. *Computer-Aided Software Engineering*. Współcześnie ten akronim jest rzadko używany. Ten narzędzia stosowane są w procesie zbierania i analizowania wymagań oraz projektowania (tzw. narzędzia Upper CASE) lub w implementacji i testowaniu (narzędzia Lower CASE). Współczesnymi przykładami są GitLab, GitHub, JUnit.

Notatki

Koszty tworzenia oprogramowania

Całkowity koszt tworzenia zależy od typu oprogramowania i zastosowanego zastosowanego modelu rozwoju. Ogólnie, najbardziej kosztowną fazą jest weryfikacja (testowanie), które może pochłonąć do 40% budżetu, a w przypadku oprogramowania, w którym bezpieczeństwo jest krytyczną cechą, nawet do 50%. W całym cyklu życia oprogramowania, obejmującym opracowanie, utrzymanie i wycofanie, największe koszty przypadają na tę środkową fazę.

Notatki

Wyzwania

Do najważniejszych wyzwań współczesnej inżynierii oprogramowania można zaliczyć:

- przestarzałe oprogramowanie (ang. *legacy software*),
- terminy dostaw,
- jakość oprogramowania, w tym bezpieczeństwo i niezawodność,
- koszt społeczny wdrożenia oprogramowania.

Notatki

Podsumowanie

Oprogramowanie staje się coraz bardziej skomplikowane, a od jego poprawnego działania zależy coraz więcej aspektów ludzkiego życia. Nie wiemy jeszcze dokładnie jak tworzyć niezawodne, wydajne i tanie oprogramowanie. W stosunkowo krótkiej historii inżynierii oprogramowania odnotowano wiele katastrof (patrz slajd nr 7), ale również wiele sukcesów (choćby sondy kosmiczne Voyager). Ostatnia opublikowana wersja dokumentu pt. *Chaos Report* (rok 2020) sugeruje, że w przyszłości oprogramowanie będzie wytwarzane *potokowo* (ang. *pipelined*), czyli podobny sposób jak w metodach DevOps/DevSecOps. Wydaje się, że inżynieria oprogramowania ostatecznie zmierza we właściwym kierunku. Jako wniosek końcowy można stwierdzić, że stosowanie się do zasad inżynierii oprogramowania nie gwarantuje powodzenia projektu informatycznego, ale ich ignorowanie jest przepisem na katastrofę.

Dla rozrywki

Aaron Cummings,
Uptime 15,364 days - The Computers of Voyager,
https://www.youtube.com/watch?v=H62hZJVqs2o

Notatki

?

Notatki

Dziękuję Państwu za uwagę!

Notatki

Notatki