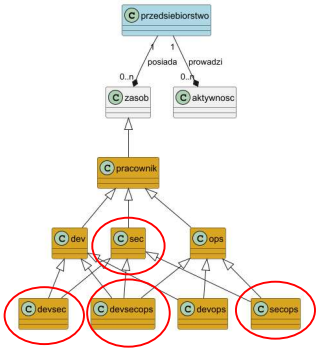
Inżynieria Oprogramowania

materiały powtórkowe
2025/26

prof. dr hab. inż. Piotr Cofta
CISSP

1

Kim będziecie



2

Co to zatem jest

- Inżynieria oprogramowania to
  - dziedzina **inżynierii**
  - dotycząca **produkcji** oprogramowania,
  - której celem jest
    - **planowe** uzyskanie
    - wysokiej **jakości** produktu
    - (w tym **bezpieczeństwa**)
    - **na czas**
    - i w ramach **budżetu**.

3

Zapewnienie jakości

- Jakość jest wymaganiem **niefunkcjonalnym** dotyczącym **całego systemu**
  - Poprawność, niezawodność, wydajność
  - **Bezpieczeństwo**, ...
- Wymagania niefunkcjonalne
  - są bardzo trudne do spełnienia
- Wymagania dotyczące całego systemu
  - są bardzo trudne do weryfikacji

No to mamy problem

4

Przedsiębiorstwo

- **Przedsiębiorstwo** działa poprzez dwa typy **aktywności**
  - **procesy**: powtarzalne, nie zmieniające stanu firmy,
    - np. produkcja, księgowość, sprzedaż, ...
  - **programy**: unikalne działania biznesowe, zmieniają stan firmy
    - np. wejście na nowy rynek
    - np. wytworzenie nowego produktu
    - ...

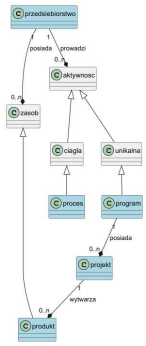
5

Program i projekt

- **Program** jest aktywnością mającą cel biznesowy
  - np. wzrost sprzedaży
- **Projekt** jest aktywnością mającą na cel wytworzenie **produktu**-rezultatu (deliverable)
  - np. wytworzenie nowego oprogramowania

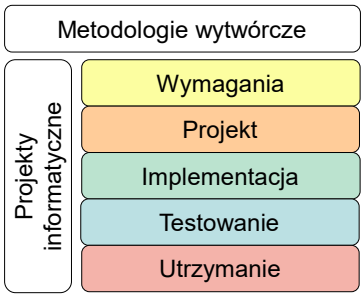
6

5 x P



7

Fazy wytwarzania oprogramowania



8

Podsumowanie

- inżynieria oprogramowania
- wymaganie niefunkcjonalne
- fazy:
  - wymagania
  - projektowanie
  - implementacja
  - testowanie
  - utrzymanie
- Dev / Sec / Ops
- 5 x P
  - przedsiębiorstwo
  - proces
  - program
  - projekt
  - produkt (deliverable)

9

Metodologie

Inżynieria Oprogramowania
2025/26

prof. dr hab. inż. Piotr Cofta

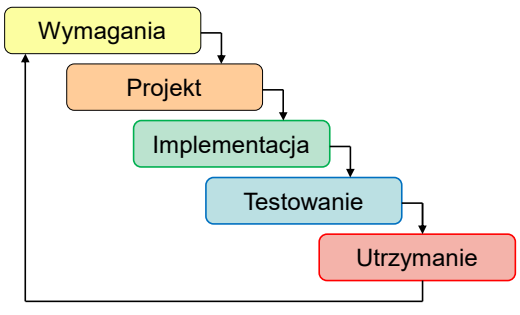
10

Metodologie wytwórcze

- Każdy sposób ułożenia 'klocków' to inna metodologia wytwórcza, czyli
  - standard
  - ciągu operacji
  - w czasie, zakresie itp.
  - obejmującego wszystkie fazy
- Te 'klocki' można ułożyć na wiele sposobów
  - historycznie próbowano około 10 układów

11

Metodologia waterfall



12

Współczesne metodologie

Zwinny (agile)
Open source
Formalny

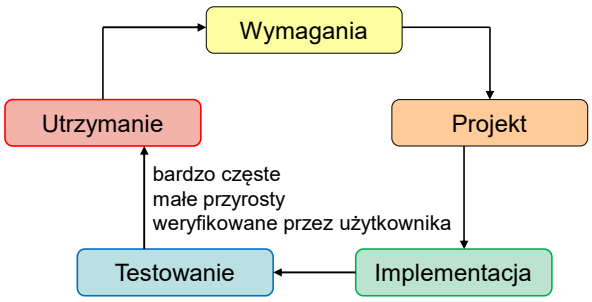
- Firma zazwyczaj wykorzystuje
  - jedną metodologię jako dominującą
  - kilka metodologii, zależnie od wymagań klienta
  - szkodliwą mieszankę metodologii

13

Agile

14

Agile (zwinne)

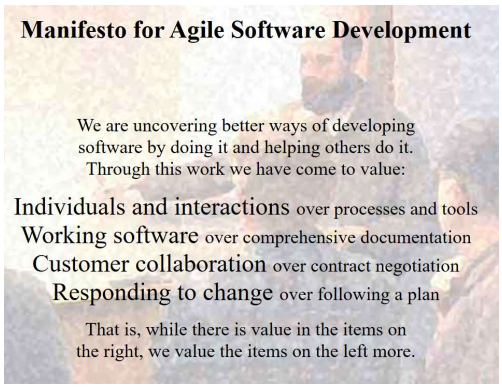


15

Krótki horyzont planowania

- Klienci zmieniają swoje wymagania
  - zmienia się rynek
  - jest to niewygodne, ale niestety prawdziwe
- Oprogramowanie musi nadążać za zmianami
  - Amazon 11.6s
- Krótki horyzont planowania
  - pomaga wymaganiom funkcjonalnym
  - szkodzi wymaganiom niefunkcjonalnym
    - wymogi bezpieczeństwa nie są lubiane

16



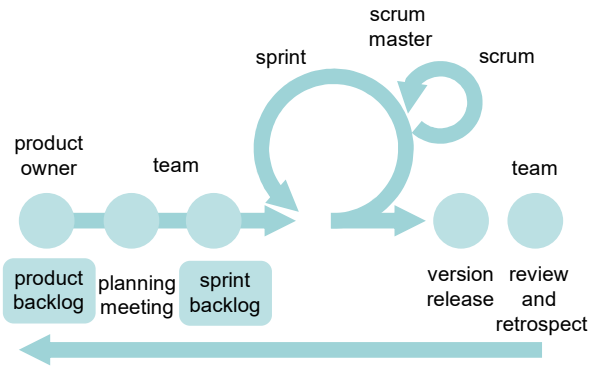
http://agilemanifesto.org/ (2001)

17

Scrum it all..

https://explore.versionone.com/state-of-agile (2017)

18



19

Podstawowy cykl Sprint

- Projekt jest podzielony na krótkie regularne odcinki czasu ('sprinty')
  - 5 dni - 1 miesiąca
- W ramach projektu zespół przypisany do Sprintu jest mały
  - 7-12 osób
- Oraz stabilny
  - zmiany tylko w uzasadnionych przypadkach

20

- Każdy odcinek czasu zaczyna się od uzupełnienia 'backlogu'
  - planowane rzeczy do zrobienia
  - nie zrobione, choć powinny być zrobione
  - modyfikacje, wyniki testowania itp.
  - uszczegółowienie tego, co trzeba zrobić
- Backlog jest uzupełniany przez
  - product ownera (reprezentuje użytkownika)
  - członków zespołu
- Wpisom nadawane są priorytety

21

- Do backlogu trafiają zadania różnego typu
  - techniczne (np. napisać kod)
  - organizacyjne (np. zamówić pizzę)
  - precyzujące (np. rozpisać wymaganie)
  - porządkujące (np. uporządkować biurka)
  - ...
- .. jak długo zespół uważa, że powinny być one zrealizowane

22

- Planowanie sprintu polega na doborze z backlogu elementów które będą wykonane w tym cyklu
  - przez zespół !!
- Wynikiem planowania jest backlog dla tego sprintu

23

- Od tej chwili członkowie zespołu realizują sprint
  - wybierają zadania które mogą najlepiej wykonać
  - pobierają te zadania z backlogu sprintu
  - realizują te zadania w kolejności
  - ewentualne nowe zadania są dopisywane do backlogu produktu

24

- Codziennie odbywa się scrum
  - krótkie spotkanie (kwadrans)
  - cały zespół
  - każdy raportuje
    - nad czym pracuje
    - co zrobił
    - czy są kłopoty
  - w odniesieniu do celów sprintu
- Scrum jest koordynowany przez scrum master

25

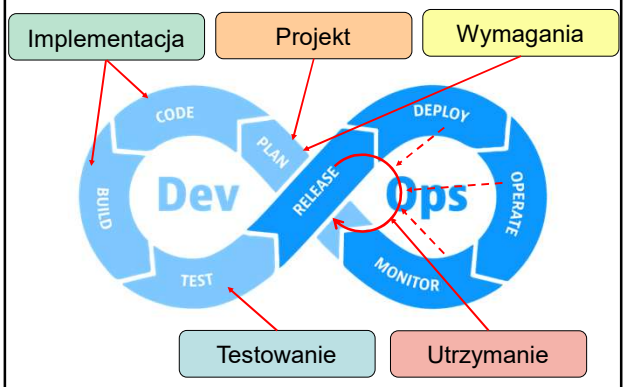
- Na koniec sprint
  - powstaje nowa wersja którą można pokazać użytkownikowi (lub product owner)
  - spotkanie dotyczące tego co było
    - co poszło dobrze
    - co poszło źle
    - co może być usprawnione

26

DevOps

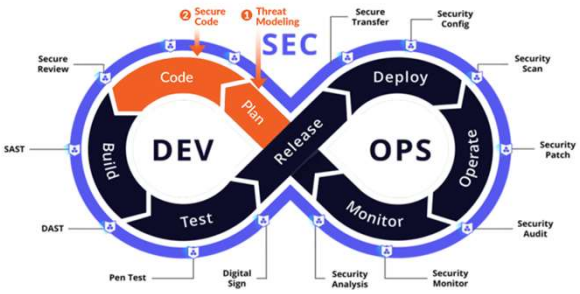
- DevOps jest praktyką agile
  - zastosowanie agile do bardzo szybkich cykli
- Podkreśla ciągłość współpracy 'dev' i 'ops'
  - wspólna odpowiedzialność
- Osobna specjalność zawodowa
- Tutaj omawiamy tylko 'dev'

27



28

DevSecOps



29

CI/CD

- DevOps naciska na automatyzację
  - wszystko, co może zostać zautomatyzowane, powinno być zautomatyzowane
- CI/CD jest platformą narzędziową DevOps
  - zestaw narzędzi dobrany do projektu
  - być może kiedyś jednolite środowisko

30

Gdzie jest bezpieczeństwo

- Agile
  - jest to jedno z wymagań w backlog
  - stopniowo realizowane tak jak inne
- DevOps
  - rozszerza praktykę na DevSecOps
  - na każdym etapie pętli
- CI/CD
  - specjalistyczny zestaw narzędzi
  - automatyzacja

31

Open Source

32

"Szansa napotyka podatny grunt"

- 80'-90'
  - doświadczeni deweloperzy
  - poczucie wspólnoty
  - docenianie technicznej doskonałości
  - nuda pracy dla korporacji
- Narzędzia
  - komputery osobiste
  - Internet
  - rozproszone repozytorium (Git)
  - mail

33

Charakterystyka

- Cel projektu jest zrozumiany przez uczestników
- Kod jest dostępny dla wszystkich zainteresowanych
- Każdy uczestnik może pracować niezależnie
- Łatwo do łączyć się, łatwo odejść
- Praca nad projektem zazwyczaj nie jest głównym zajęciem
- Każdy sensowny wkład pracy jest doceniany

34

Istotne

- Dokumentacja
  - wszystko jest na piśmie
- Zarządzanie wersjami
  - wszystko jest pamiętane
- Merytokracja
  - szefem jest ten, kto wie i pracuje

35

Formalne

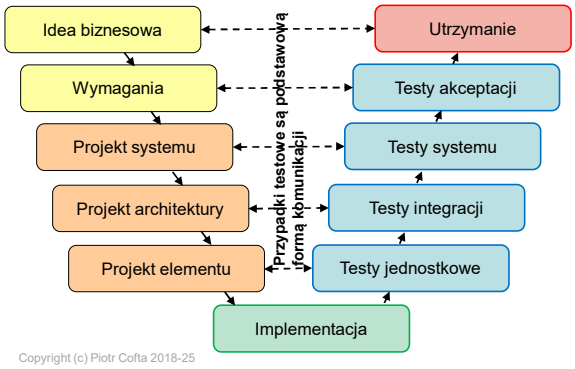
36

Metoda formalna

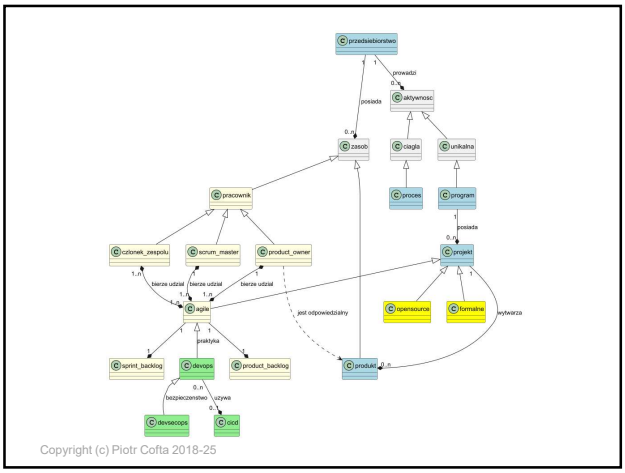
- Wykorzystywana dla oprogramowania wysokiego ryzyka którego błędne działanie może spowodować znaczne lub trwałe szkody
  - uszkodzenie ciała lub śmierć
  - uszkodzenie krytycznej infrastruktury kraju
  - destabilizacja rynków lub instytucji
  - zmniejszenie potencjału obronnego
  - ...

37

V model (wersja współczesna)



38



39

Terminologia

- metodologia
- waterfall
- zwinne (agile)
- sprint
- scrum
- backlog
- DevOps
- DevSecOps
- CI/CD
- open source
- formalne
- V

40

Wymagania

Inżynieria Oprogramowania
2025/26

prof. dr hab. inż. Piotr Cofta

41

Wymaganie

- Wymaganie określa cechy produktu które są ważne dla klienta / użytkownika / interesariusza
  - klient będzie niezadowolony jeżeli nie spełnimy wymagań
- Wymaganie określa także konieczne zgodności ze standardami, kontraktami itp.
  - bez spełnienia wymagań nie będzie zapłaty

42

Interesariusze (stakeholders)

- Osoba, grupa lub organizacja która jest zainteresowana produktem
  - użytkownicy, w tym potencjalni użytkownicy
  - dostawcy sprzętu, bibliotek itp.
  - deweloperzy, również potencjalni
  - zespół managerski
  - księgowość
  - właściciele firmy
  - ...

43

Wymagania są trudne

1. **Unrealistic expectations of stakeholders**
2. **Scope creep**
3. Stakeholders do not express what they consider obvious
4. Not enough time to process requirements
5. Stakeholders not available or not interested

Jarzębowicz A. et al. (2018) What is Troubling IT Analysis?

44

Cechy dobrego wymagania

- Kryteria
  - **Testowalne** - czy można to wytestować
  - **Mierzalne** - czy jest mierzalne
  - **Kompletne** - czy zawiera wszystko co potrzebne
  - **Zrozumiałe** - czy język jest poprawny
  - **Jednoznaczne** - czy nie można go źle zrozumieć
  - **Niesprzeczne** - czy jest zgodny z pozostałymi
- Dobre wymaganie jest procedurą testowania

45

Wymagania bezpieczeństwa

- "System ma być bezpieczny"
- To nie jest dobry przykład
  - jakie są granice systemu?
  - musi, może czy powinien być bezpieczny?
  - co to znaczy 'bezpieczny'?
  - **jak to testować?**
  - czy musi być absolutnie bezpieczny?
  - czy musi być ciągle bezpieczny?

46

Typy wymagań

- Funkcjonalne
  - 'co': oczekiwane wyniki lub elementy systemu
  - np. MUSI mieć interfejs użytkownika
- Niefunkcjonalne - 'jakości'
  - 'jak' : w jaki sposób spełniane są funkcjonalne
  - np. interfejs użytkownika MUSI być użyteczny
- Ograniczenia
  - dodatkowe ograniczenia na wybór rozwiązania
  - np. interfejs użytkownika MUSI wspierać Chrome

47

Istotne niefunkcjonalne

- Bezpieczeństwo (i prywatność)
  - odporność na ataki z zewnątrz
- Użyteczność
  - łatwość użycia systemu przez użytkownika
- Utrzymywalność
  - łatwość dbania o system przez administratora
- Testowalność
  - gotowość do testowania

48

Wymagania w metodologiach

49

Metodologie przed agile

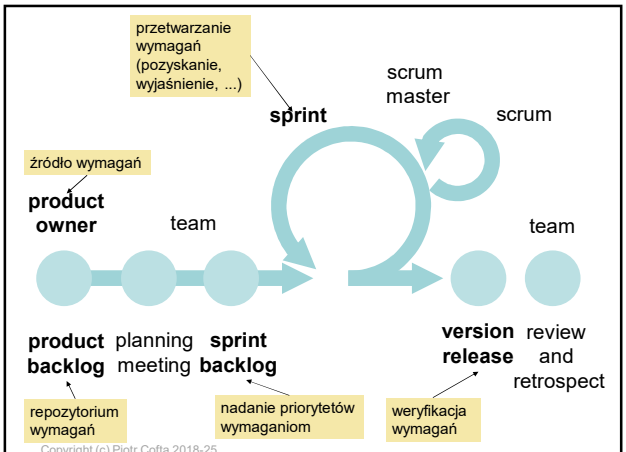
- Stosowana jest inżynieria wymagań
  - Podejście systematyczne i zdyscyplinowane
  - Zestandaryzowane (ISO9126)
  - Z własnym certyfikatem (IREB CPRE)
  - I stanowiskiem (requirement engineer)
- W metodologii formalnej jest to nadal wymagane!

50

'W agile nie mamy wymagań'

- Jest to popularne ale nie jest prawdziwe
  - w agile też potrzebne są wymagania
- Agile manifesto jest często źle interpretowane
  - Product owner - źródło wymagań
  - Sprint planning - sterowne przez wymagania
  - Release - weryfikacja względem wymagań
  - Sprint - process opracowania i modyfikacji wymagań

51



52

Wymagania w agile

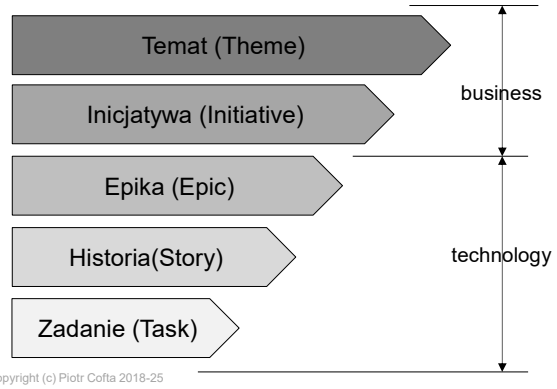
- Podstawowe cechy
  - określanie wymagań tak późno, jak to możliwe
  - gotowość na zmiany wymagań
  - skoordynowanie z praktyką np. Sprint
  - późne dokumentowanie wymagań
- Sposób integracji z praktyką Sprint
  - proces 'epicki'
  - czyli tzw. historie użytkownika

53

Proces 'epicki'

54

Proces 'epicki'



55

Poziomy procesu

- Temat - ogólny kierunek organizacji
- Inicjatywa - biznesowa potrzeba zmiany
- Epika - dobrze zdefiniowana fragment pracy
- Historia - perspektywa użytkownika
- Zadanie - jednostka pracy

- Przykład!

56

Epika

- Spójny opis prac dla projektu

- Epika: 'Musimy spełniać polskie normy bezpieczeństwa'
- Czyli: musimy zmodyfikować nasz produkt

57

Historia ('story', 'user story')

- Luźno sformułowane oczekiwania różnych interesariuszy, zebrane bez specjalnej edycji
  - nie ma gwarancji, iż wszystkie będą spełnione
  - lub nawet zrozumiane poprawnie

- Historia: jako product manager chciałbym poznać wymagania dotyczące certyfikacji

58

Zadanie z wymaganiami

- Zadanie i wymagania są generowane gdy są potrzebne
  - Zadanie to fragment pracy który przyczynia się do spełnienia historii
  - Wymaganie związane z zadaniem mówi nam, kiedy jest ono gotowe

- Zadanie: Zidentyfikować wymagania dotyczące certyfikacji w Polsce

59

RFC 2119

- **MUSI**: cecha jest absolutnie konieczne, brak spełnienia skutkuje odrzuceniem produktu
- **NIE MOŻE**: cecha absolutnie nie może wystąpić w produkcie; wystąpienie oznacza odrzucenie produktu
- **POWINIEN**: produkt powinien posiadać daną cechę, chyba iż istnieją ważne powody aby jej nie posiadał; brak wymaga wyjaśnienia
- **NIE POWINIEN**: produkt nie powinien posiadać danej cechy, chyba iż istnieją ważne powody aby ta cecha istniała, istnienie wymaga wyjaśnienia
- **MOŻE**: cecha jest całkowicie opcjonalna, jej obecność jest mile widziana ale nie wpływa na odbiór produktu

60

MoSCoW

- **M - Must have** - krytycznie ważne
- **S - Should have** - ważne, ale może być pominięte
- **C - Could have** - pożądane, ale obecnie nie jest to najlepsze wykorzystanie ludzi
- **W - Won't have** (this time) / Wish - nie będzie uwzględnione w tym Sprint

61

Posumowanie

- Interesariusz (stakeholder)
- Scope creep
- Dobre wymaganie jest testowalne
- Wymagania
  - funkcjonalne
  - niefunkcjonalne
  - ograniczenia
- Epic proces
  - temat
  - inicjatywa
  - epika
  - historia (użytkownika)
  - zadanie
- Język
  - RFC2119
  - MoSCoW

62

Projektowanie

Inżynieria Oprogramowania
2025/26

prof. dr hab. inż. Piotr Cofta

63

Co to znaczy 'projektować'?

- Projektowanie to systematyczna metoda
- rozłożenia problemu na prostsze elementy
- poprzez zastosowanie ograniczonej liczby wzorców projektowych
- aż do momentu kiedy jest możliwe programowanie ('kodowanie')
- poprzez podejmowanie uzasadnionych decyzji projektowych

Projekt to udokumentowany ciąg uzasadnionych decyzji

64

Decyzje projektowe

- Uzasadnione,
- udokumentowane,
- odwracalne

- Czyli
  - musi być powód aby konieczne było podjęcie decyzji
  - powód ten musi być udokumentowany
  - każda decyzja jest odwracalna
  - ale niektóre są bardzo kosztowne

65

ADR - Architecture Decision Record

- Struktura
  - In the context of .. <sytuacja>
  - .. while facing .. <problem>
  - .. we/I decided .. <decyzja>
  - .. neglecting .. <inne opcje>
  - .. to achieve .. <co było decydujące>
  - .. accepting .. <problemy które pozostały>

- Albo .. notatka w kodzie.. :)

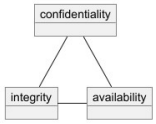
66

CIA

- Confidentiality (poufność)
  - właściwy odczyt danych jest możliwy tylko dla autoryzowanych jednostek
- Integrity (spójność)
  - manipulacje danymi oraz ich usuwanie są dozwolone niezaprzeczalnie tylko dla autoryzowanych jednostek
- Availability (dostępność)
  - autoryzowane jednostki mają szybki i wiarygodny dostęp do danych

67

The CIA impossibility



- Nie jest możliwe kompletne spełnienie wszystkich trzech wymagań
  - zatem: nie ma systemu kompletnie bezpiecznego

68

Jak długo projektujemy w agile

- Projektowanie w agile jest przyrostowe
  - zaczynamy gdy tylko jest to możliwe
  - projektujemy tyle, ile jest potrzebne
  - dokumentujemy tyle, ile jest niezbędne
- W Sprint, projektowanie jest ciągiem zadań
  - zadania pobierane z backlog, wyniki oddawane do backlog
- **Projektowane kończy się z końcem projektu**

69

Konflikt

- Projektowanie w agile jest
  - fragmentaryczne i przyrostowe,
  - podatne na zmiany
  - krótkoterminowe
- Bezpieczeństwo wymaga projektowania
  - wczesnego i długoterminowego
  - struktury całego systemu
  - z zachowaniem niezmienności rozwiązań
- **Jest to istotny konflikt**

70

Wzorce projektowe (design patterns)

- problem dekomponujemy używając gotowych **wzorców projektowych**
- Wzorzec projektowy to nie kawałek gotowego kodu, ale sposób **podejścia do problemu**
- Wzorzec projektowy jest znanym **powtarzalnym rozwiązaniem sytuacji**
- Jest on **najlepszą praktyką** (best practice), skumulowaną wiedzą praktyków

71

Wzorce projektowe bezpieczeństwa

- Nie istnieje ustalony zbiór wzorców projektowych bezpieczeństwa, ale jest ich sporo
  - Authentication Patterns
  - Authorization Patterns
  - Secure Communication Patterns
  - Input Validation Patterns
  - Session Management Patterns
  - Audit and Logging Patterns
  - Error Handling Patterns
  - Data Protection Patterns
  - Defense in Depth

72

Input Validation Pattern

- Input validation
  - prosty wzorzec
  - dobra praktyka
  - wszechstronne zastosowanie
  - dobre zabezpieczenie
  - prowadzi do programowania defensywnego
  - oraz do 'defense in depth'

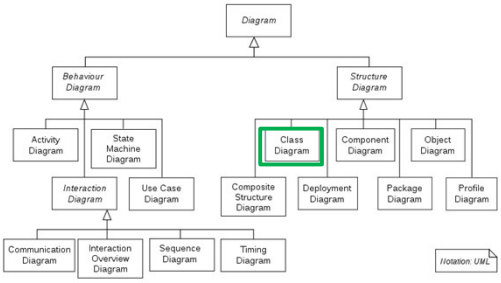
73

Input validator

- W danym fragmencie systemu mamy to co chcemy ochronić i to, co dostarcza dane / podaje polecenia
- To, co chronimy nie powinno ufać temu, kto go wywołuje
- Co robić?
- **Weryfikować to, czego żąda wywołujący**

74

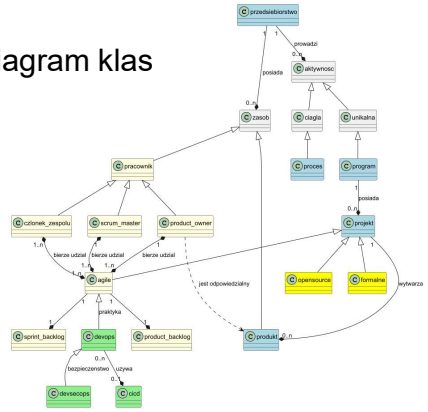
Typy diagramów UML



https://en.wikipedia.org/wiki/Unified_Modeling_Language

75

Diagram klas



76

Przypadki użycia

- Use case (przypadek użycia)
  - opisuje jeden, wybrany przebieg interakcji z systemem
  - nie stara się generalizować, ale uchwycić aktorów
- Mis-use cases (przypadki nadużycia)
  - opisuje potencjalne ataki, zagrożenia itp.
  - Metoda ta sama co w wypadku nominalnych przypadków użycia, ale inni aktorzy

77

AI jako wektor ataku

1. Atakujący opracowuje użyteczny i nietypowy kod, celowo ze złośliwym fragmentem
2. Atakujący umieszcza ten kod w Internecie, dostęp publiczny
3. Ops_AI pobiera kod z Internetu razem z innymi danymi
4. Ops_AI trenuje AI z wykorzystaniem programu
5. Dev potrzebuje użytecznego programu
6. Dev zapytuje AI o taki kod i otrzymuje go ze złośliwym fragmentem
7. Dev nie sprawdza tego przed dodaniem do produktu
8. Dev wypuszcza na rynek produkt ze złośliwym fragmentem
9. Użytkownik nabywa produkt z takim fragmentem
10. Uruchomiony fragment kodu w produkcie informuje atakującego
11. Atakujący wykorzystuje fragment aby zaatakować użytkownika

78

Podsumowanie

- Projektowanie to ciąg decyzji projektowych
  - odwracalnych
- ADR: Architecture Decision Record
- Wzorce projektowe (design patterns)
  - Input validator
- UML: Unified Modeling Language
  - diagram klas
- Przypadki użycia
  - przypadki nadużycia

79

Implementacja

Inżynieria Oprogramowania
2025/26

prof. dr hab. inż. Piotr Cofta

80

Zarządzanie wersjami

- Śledzi zmiany
- Pozwala na powrót do poprzednich wersji
- Wprowadza kontrolę dostępu
- Pamięta kto co zrobił
- Utrzymuje dodatkowe informacje
- Dostarcza statystyk

81

Rodzaje zarządzania wersjami

- Scentralizowany
  - centralne repozytorium plików
  - 'pożyczane' do edycji (checked out)
  - 'zwracane' po wykonanej edycji (checked in)
- Zdecentralizowany (rozproszony)
  - każdy użytkownik ma kompletna kopię repozytorium
  - wszystkie zmiany są dopuszczalne i są lokalne
  - scalanie zachowanych lokalnych wersji

82

Git ma swoje opinie

- Świat składa się z rozproszonych **repozytoriów** (repositories) opcjonalnie połączonych zgodnie z pewnymi **konwencjami** (conventions)
- Każde repozytorium jest **pełnym opisem** historii projektu i może istnieć **niezależnie**
- Repozytorium to rozchodzące i schodzące się **gałęzie** (branches) złożone z **zatwierdzeń** (commits)
- **Nie ma centralnego repozytorium** i nie ma absolutnej prawdy o projekcie; zmiany są **scalane** (merged) jeśli jest to potrzebne

83

Repozytorium (repository)

- Repozytorium to zbiór plików projektu
  - dowolna kartoteka i jej podkartoteki
  - ta kartoteka jest kartoteką roboczą
  - pliki specjalne Git są ukryte w podkartotece .git
- Na komputerze może istnieć wiele repozytoriów
  - ich pliki są od siebie niezależne
  - operacje na różnych repozytoriach są niezależne
- Każdy może utworzyć nowe repozytorium
  - wystarczy zainstalować git na komputerze
  - open source

84

Zatwierdzenie (commit)

- Zmiany dokonane w katalogu roboczym i jego podkatalogach są zachowane w lokalnym repozytorium dopiero po **zatwierdzeniu**
- Zatwierdzenie
  - tworzy w repozytorium kopię wszystkich znanych plików z katalogu roboczego i podkatalogów
  - (w rzeczywistości zachowuje tylko zmiany)
- Zatwierdzenie jest nazwane
  - można do niego wrócić
  - wszystkie pliki będą odtworzone do stanu z zatwierdzenia

85

Gałąź (branch)

- Szereg kolejnych zatwierdzeń tworzy gałąź
  - każda gałąź to inna historia zmian w repozytorium
  - każda gałąź ma swoją nazwę
  - w repozytorium może istnieć dowolna liczba gałęzi
- W danym momencie tylko jedna gałąź jest wybrana (bieżąca)
  - zmieniamy ją instrukcją 'checkout'
- Gałęzie tworzą charakterystyczne 'drzewo'

86

Scalanie (merge)

- Dowolna gałąź może być scalona z inną
  - przestaje wtedy istnieć
- Scalenie powoduje iż pliki z jednej i drugiej, w najnowszych wersjach, są dostępne
- Może to spowodować konflikty
  - zazwyczaj rozwiązywane automatycznie
  - niekiedy wymagają ingerencji człowieka

87

Resetowanie

- Daną gałąź można zresetować do dowolnego zatwierdzenia które do niej należy
  - 'cofnięcie historii'
- Istnieją różne sposoby resetowania
  - niektóre powodują zapomnienie wycofanej historii

88

Repozytorium zdalne

- Repozytorium zdalne umożliwia współpracę
  - każdy użytkownik ma swoje repozytorium lokalne
  - na dowolnym serwerze istnieje repozytorium zdalne
  - każde lokalne może pobrać zawartość zdalnego
  - każde lokalne może zaktualizować zdalne
    - to niekiedy jest ograniczane
- Jest to tylko konwencja, nie wymóg !

89

Bezpieczeństwo kodu

90

Sławne strcpy

What is the strcpy() Function?

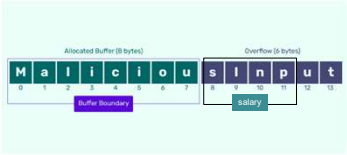
The `strcpy()` function is a standard library function in the C programming language, designed to copy strings from one memory location to another. It is included in the **string.h** header file and stands for "string copy". The primary objective of this function is to replicate a source string into a destination buffer while ensuring both strings are null-terminated.

The `strcpy()` function works by taking two arguments: a pointer to the destination buffer (called `dest`) and a pointer to the source string (called `src`). The function iterates through the characters in the source string, copying each character to the destination buffer, and finally appending a null character `\0` to terminate the destination string. Here's a short code example:

```
#include  <stdio.h>
#include  <string.h>

int main() {
    char src[] = "Hello, World!";
    char dest[20];

    strcpy(dest, src);
    printf("Copied string: %s\n", dest);

    return 0;
}
```


91

'Buffer overflow'




```
char buffer[8];
int salary;

input [] = read();

strcpy(buffer,input);
```


92

Zapobieganie buffer overflow

- Uświadomienie programistów
- Używanie strncpy
- Nie korzystanie z C / C++ jeżeli nie jest to konieczne
  - używać Rust, Go, Python, Java, C#
- Wykorzystanie analizatorów kodu
- Izolowanie danych i kodu w pamięci
  - nie działa dla ataku ze stosem
- Izolowanie programów aby ograniczyć zniszczenia

93

Podsumowanie

- Zarządzanie wersjami (version control)
- Git
  - repozytorium (repository)
  - gałąź (branch)
  - zatwierdzenie (commit)
  - scalenie (merge)
  - conflict
  - reset
  - clone / fork
  - pull / push
- Bezpieczeństwo kodu
  - buffer overflow
  - Git Copilot, ChatGPT
  - upstream
  - downstream

94

Testowanie

Inżynieria Oprogramowania
2025/26

prof. dr hab. inż. Piotr Cofta

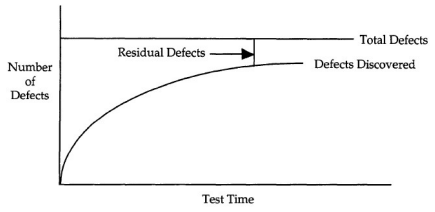
95

Dlaczego testujemy

- aby zmniejszyć niepewność
- aby wykryć błędy
- aby zademonstrować staranność
- aby uzyskać certyfikację
- ...
- **nie testujemy aby wykazać poprawność**

96

Im dłużej tym lepiej



McConnell, S: Code Complete

97

Testowanie w agile

- Tak, jak wszystko, w agile testowanie jest częścią działalności zespołu
- Zadania (task) mogą dotyczyć testowania
- Na końcu Sprint jest faza testowania
- Testerzy mogą pracować w ramach zespołu

98

Statyczne - dynamiczne

- Wykorzystuje tekst (dokumenty, kod)
  - nie uruchamia systemu
  - bazuje na przeczytaniu (skanowaniu) tekstu
- Wykorzystuje działający system
  - wymaga uruchomienia systemu
  - wykonuje zaplanowane przypadki testowe

99

Inspekcja kodu

(testowanie statyczne)

100

Inspekcja kodu

- Inspekcja kodu to systematyczne sprawdzenie kodu
  - przez innego dewelopera lub testera
  - celem wykrycia błędów i nieprawidłowości
- W wypadku bezpieczeństwa
  - celem wykrycia potencjalnych podatności
- Potencjalna podatność jest to konstrukcja która potencjalnie może zostać podatnością

101

Ograniczenia statycznego

- Użyteczność testowania statycznego jest ograniczona przez problem stopu
  - dla nietrywialnego kodu,
  - nie jest możliwe automatyczne określenie,
  - na podstawie statycznej analizy,
  - czy kod się kiedykolwiek zatrzyma
  - (A. Turing, 1936)
- Nie może istnieć analiza statyczna która udowodniłaby poprawność kodu

102

Testy penetracji

(testowanie dynamiczne)

103

Przypadki testowe

- W testowaniu dynamicznym wykonujemy **przypadki testowe** (test cases)
- Przypadek testowy to minimalna jednostka testowania dynamicznego
  - jest spełniony ('pass') lub nie ('fail')
- Nie ma 'częściowo spełnionego' przypadku testowego
  - choć niespełniony przypadek testowy może dostarczyć dodatkowych informacji

104

Ograniczenia

- Liczba możliwych kombinacji wejściowych jest na tyle duża iż nie jest praktycznie możliwe przejście przez wszystkie
- Użyteczność testowania dynamicznego jest ograniczona przez praktycznie osiągalne pokrycie

105

Black and white

- Testy 'white box'
  - wewnętrzna struktura systemu (kodu) jest znana
  - można zaprojektować przypadek wykorzystując tą wiedzę
- Testy 'black box'
  - wewnętrzna struktura nie jest znana
  - można obserwować tylko wejście i wyjście
- Testy 'grey box'
  - struktura jest częściowo znana

106

Testy penetracji

- 'Ethical (white) hacking'
- Test całego systemu
- Black box, niekiedy grey box
- Dokonywane 'z zewnątrz'
- Mają na celu wykrycie podatności

107

Podsumowanie

- Statyczne
  - inspekcja kodu (code inspection)
  - problem stopu (halting problem)
- Dynamiczne
  - przypadek testowy
  - pokrycie
  - white-black-grey
  - testy penetracji (pen testing)
  - white hacking

108

Utrzymanie

Inżynieria Oprogramowania
2025/26

prof. dr hab. inż. Piotr Cofta

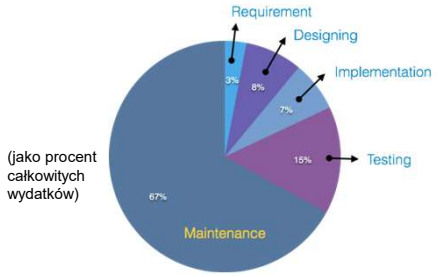
109

Utrzymanie oprogramowania

- Gdy oprogramowanie jest gotowe, rozpoczyna się faza utrzymania
  - gotowe: po testach, zaakceptowane
- Utrzymanie to nie operacja (ops)
  - ops dba o ciągłość działania oprogramowania w takiej wersji jaka jest
- Utrzymanie to dopasowanie oprogramowania do zmian (wytworzenie nowych wersji)
  - zmiany w potrzebach rynku (klienta) - nowe wersje
  - zmiany konieczne dla korekcji błędów - patching
  - ..

110

Koszt utrzymania



dla typowego oprogramowania 67% kosztów to utrzymanie

https://www.tutorialspoint.com/software_engineering/software_maintenance_overview.htm

111

'Wypuszczenie' (Release)

- Oznaczamy (tag) bieżące zatwierdzenie jako kandydata ('release candidate')
  - .. i praca może toczyć się dalej
  - .. Git pamięta dokładnie to, co oznaczyliśmy
- Budujemy artefakty
  - zdatne do uruchomienia
- Tworzymy dystrybucję
  - dostarczamy ją do ops
- Z zachowaniem reguł bezpieczeństwa !

112

Budowa (build)

- Budowa to przekształcenie zawartości repozytorium (i innych zasobów) w artefakt gotowy do dystrybucji
- Budowa jest wysoce zautomatyzowana
- W CI/CD budowa jest uruchamiana po każdej zaaprobowanej poprawce do repozytorium

113

Artefakt

- "Artefakt to sztucznie wytworzony przedmiot lub element, który nie występuje naturalnie w przyrodzie, a jest wynikiem działalności człowieka."
- Każdy element produktu informatycznego jest zwany artefaktem
- Artefakty są zazwyczaj przechowywane w osobnym repozytorium

114

Dystrybucja (distro)

- Zestaw artefaktów stanowiący razem kompletny produkt jest zwany dystrybucją
  - distribution, distro
- Ten sam produkt może posiadać wiele dystrybucji
  - jedna wersja - min. jedna dystrybucja
    - lokalizacja - inna dystrybucja tej samej wersji
  - 'duże' wersje vs. 'małe' wersje
    - Windows 10, 11, ...
    - 'wersja z ostatniego wtorku' (build)

115

Obraz

- Wykorzystanie obrazów jest związane z
  - konteneryzacją
  - wirtualizacją
- Obraz to kompletny zestaw wszystkich plików potrzebnych do uruchomienia aplikacji
  - pliki wykonywalne
  - biblioteki
  - pliki konfiguracyjne
  - parametry
  - dane
  - ...

116

Bezpieczeństwo dystrybucji

- Ataki na dystrybucję są rzadkie, ale bardzo efektywne
  - można zainfekować wszystkie komputery na których działa zmodyfikowana dystrybucja
- Minimalne reguły bezpieczeństwa
  - podpis elektroniczny
  - bezpieczny transfer
  - kontrola dostępu
  - wydzielenie czułych danych

117

Podpis (signature)

- Dystrybucja gotowa na rynek powinna być podpisana elektronicznie
  - zapewnia to iż nie będzie ona zmodyfikowana
- Zazwyczaj jest to podpisywane hierarchicznie
  - jeden podpis na dystrybucję
- Możliwe jest sprawdzenie podpisu
  - kryptografia asymetryczna
- To samo dla każdej aktualizacji

118

Co z hasłami?

- Czułe dane: hasła, klucze, certyfikaty itp.
  - nie mogą być zakodowane jawnie w repozytorium
    - w jakiejkolwiek postaci
  - nie mogą pojawić się w postaci jawnej w artefaktach
    - np. w binarnym kodzie
  - powinny być osobno przechowywane
    - Azure Key Vault, AWS Vault, ...
  - i bezpiecznie przesyłane
    - zakodowane
    - lub zabezpieczone sprzętowo

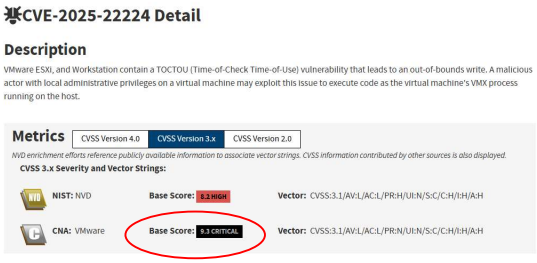
119

Jak szybko korygować błędy

- Natychmiast jeżeli jest to możliwe
  - Windows: tydzień
- 'zero day vulnerability' (ZDV)
  - podatność znana, ale jeszcze nie skorygowana
  - często nie wyłapywana przez ochronę
  - otwiera pole do ataków
- Im dłużej czekamy, tym mniej bezpiecznie

120

ZDV, VMWare, 2025



121

Koniec życia (EoL - End of Life)

- Wszystko przemija .. nawet najlepsze oprogramowanie
  - Lotus 123
  - Word Perfect
- Bezpieczny koniec życia wymaga
  - planowej migracji lub zaniechania użytkowania
  - zakończenie wsparcia
  - odinstalowania oprogramowania
  - skutecznego zniszczenia danych (w tym haseł!)

122

Podsumowanie

- Nowe wersje
  - zaznaczenie (tag)
  - budowanie (build)
  - artefakt (artefact)
  - podpisanie (sign)
  - distrybucja (distribution, distro)
  - obraz (image)
- Bezpieczeństwo
  - podpis elektroniczny
  - ochrona haseł
- Korekcja błędów
  - Zero Day Vulnerability
  - aktualizacje
- Koniec życia (end of life, EoL)

123