



Warsztat Inżyniera Oprogramowania (bardzo mały wstęp)

Sztuka Wytwarzania Oprogramowania, w. 1

Konrad Grochowski

Instytut Informatyki, Politechnika Warszawska, 2025 ©





Sztuka Wytwarzania Oprogramowania

- › *Software craftsmanship*
- › Osobiście wolę tłumaczenie „rzemiosło”
- › Talent + *warsztat* (pełen narzędzi)
- › Dadaizm, prymitywizm, czy ogólnie rozumiana sztuka abstrakcyjna *nie powinna* trafiać do oprogramowania
- › Przedmiot dotyczący technik i praktyk stosowanych w wytwarzaniu i *utrzymaniu* oprogramowania
- › Nacisk na pracę projektanta kodu / inżyniera oprogramowania
 - kodowanie jako element większej całości
- › Ten *warsztat* **nigdy** nie będzie kompletny
 - bądź *zamrożony*



Organizacja

- › Wykład: poniedziałek 10¹⁵-12⁰⁰ s. 170
- › Strona przedmiotu:
 - <https://staff.elka.pw.edu.pl/~rnowak2/dyd/swo>
- › Kanał Teams
- › Prowadzący:
 - dr hab. inż. Robert Nowak, prof. uczelni,
robert.nowak@pw.edu.pl
 - mgr inż. Konrad Grochowski,
konrad.grochowski@pw.edu.pl
- › Konsultacje: stacjonarnie lub online, patrz <http://repo.pw.edu.pl>



Regulamin

- › 2 x kolokwium po 10pkt (8 i 15 wykład - 24.11 i 26.01)
- › 4 x laboratorium po 5 pkt
- › Suma: 40pkt
- › Minimum 10pkt z laboratorium wymagane do zaliczenia
- › Oceny:
 - [37, 40] pkt. -> 5
 - [33, 37) pkt. -> 4,5
 - [29, 33) pkt. -> 4
 - [25, 29) pkt. -> 3,5
 - [21, 25) pkt. -> 3



Wymagania

- › Znajomość C++
- › Umiejętność obsługi konsoli w systemie Linux
- › Programowanie obiektowe
- › Znajomość algorytmów i struktur danych



Laboratorium

- › Termin: wtorek 8¹⁵-12⁰⁰ s. 011
- › 4 ćwiczenia (po 4h):
 1. Edytor, repozytorium git, recenzja kodu
mgr inż. Witold Wysota
Witold.Wysota@pw.edu.pl
 2. Zestawienie potoku CI/CD, testy, walidacje
mgr inż. Konrad Grochowski
Konrad.Grochowski@pw.edu.pl
 3. Monitorowanie aplikacji, debuggowanie, profilowanie
dr inż. Michał Chwesiuk
Michal.Chwesiuk@pw.edu.pl
 4. Praca z kodem zastanym, refaktoring
mgr inż. Katarzyna Nałęcz-Charkiewicz
Katarzyna.Nalecz-Charkiewicz@pw.edu.pl

N	P
28.10	21.10
25.11	02.12
09.12	16.12
20.01	13.01



Laboratorium

- › Zespoły 2 osobowe
- › Zapisy na Teams (ruszą w bieżącym tygodniu)
- › Termin zapisów – do 17.10
- › Dowolne „mieszanie grup”
- › W razie konfliktów – pierwszeństwo mają osoby z terminu zgodnego z przypisanym przez USOS



Laboratorium

- › Dokładna forma dostarczenia wykonanej pracy będzie podana na każdym zajęciach przez prowadzącego, ale opierać się będzie o repozytoria kodu na serwerze GitLab (<https://gitlab.com>).
- › Posiadanie konta na wyżej wymienionym serwerze jest niezbędnym warunkiem uczestniczenia w laboratorium.
- › Praca zespołu jest oceniana wspólnie – każdy z członków zespołu otrzymuje tę samą ocenę.
- › Zespół ma obowiązek dostarczyć efekty wykonanego ćwiczenia przed końcem laboratorium.
- › Efekty pracy będą oceniane po laboratorium.



Literatura

- › *Hunt*. Pragmatyczny programista. Od czeladnika do mistrza.
- › *Martin*. Czysty kod. Podręcznik dobrego programisty.
- › *Martin*. Mistrz czystego kodu. Kodeks postępowania profesjonalnych programistów.
- › *Gamma et al.* Wzorce projektowe.
- › *Fowler*. Refaktoryzacja. Ulepszanie struktury istniejącego kodu.

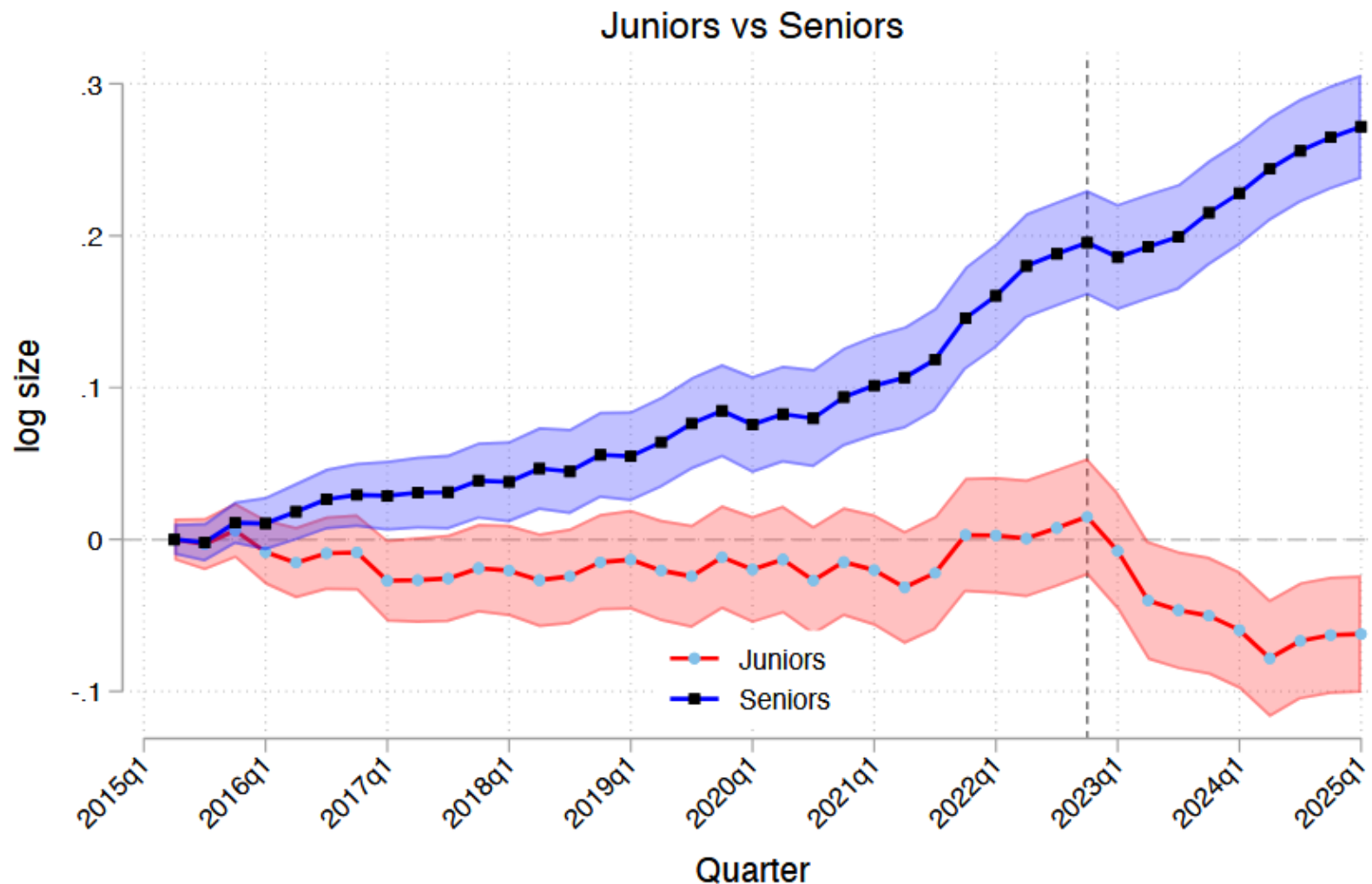


Sztuka Wytwarzania Oprogramowania

- › Talent + *warsztat*
 - + *rzetelność*
 - + *skrupulatność*
 - + ...
- › Kodowanie - element większej całości
- › Warsztat pełen narzędzi i ciągle rozwijany



Po co ten warsztat?



Źródło: Generative AI as Seniority-Biased Technological Change: Evidence from U.S. Résumé and Job Posting Data - August 2025

https://papers.ssrn.com/sol3/papers.cfm?abstract_id=5425555



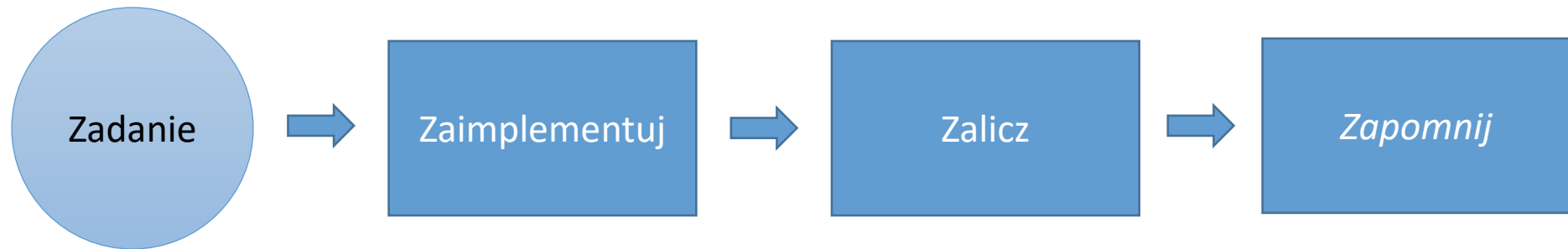
AI

- › To kolejne *narzędzie do warsztatu*
- › Poprawnie użyte *przyspiesza* a nie *zastępuje*
- › Czyli trzeba *umieć* zrobić to co się generuje
 - albo chociaż *rozumieć / ocenić*
- › Ostatnia szansa na *zrób to sam*
 - *ChatGPT już dwa lat temu dostałby 4 z SWO*



Cykl życia oprogramowania na studiach

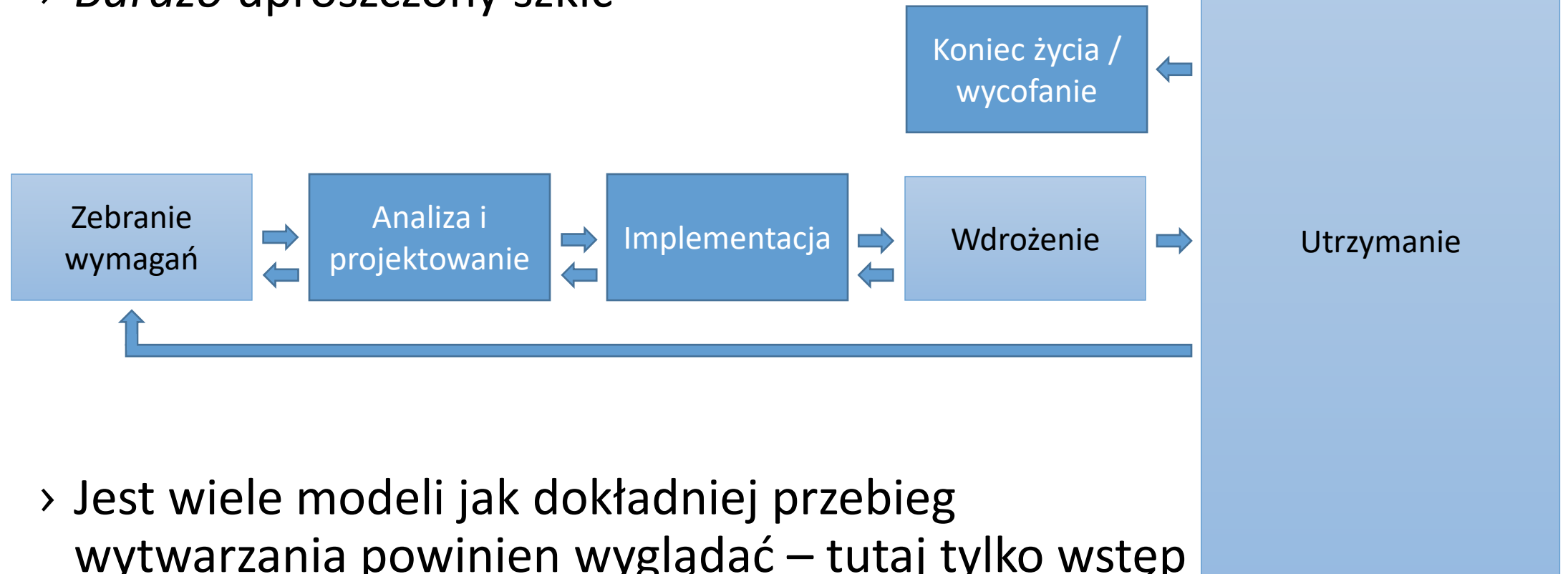
- › Każdy (mam nadzieję) zdaje sobie sprawę, że projekty na studiach są tylko pewnym przybliżeniem „prawdziwego” życia
- › Dodatkowo: proces dydaktyczny ma inne cele niż proces biznesowy
- › Niestety w efekcie cykl życia oprogramowania na studiach najczęściej wygląda tak:





Cykl wytwarzania oprogramowania

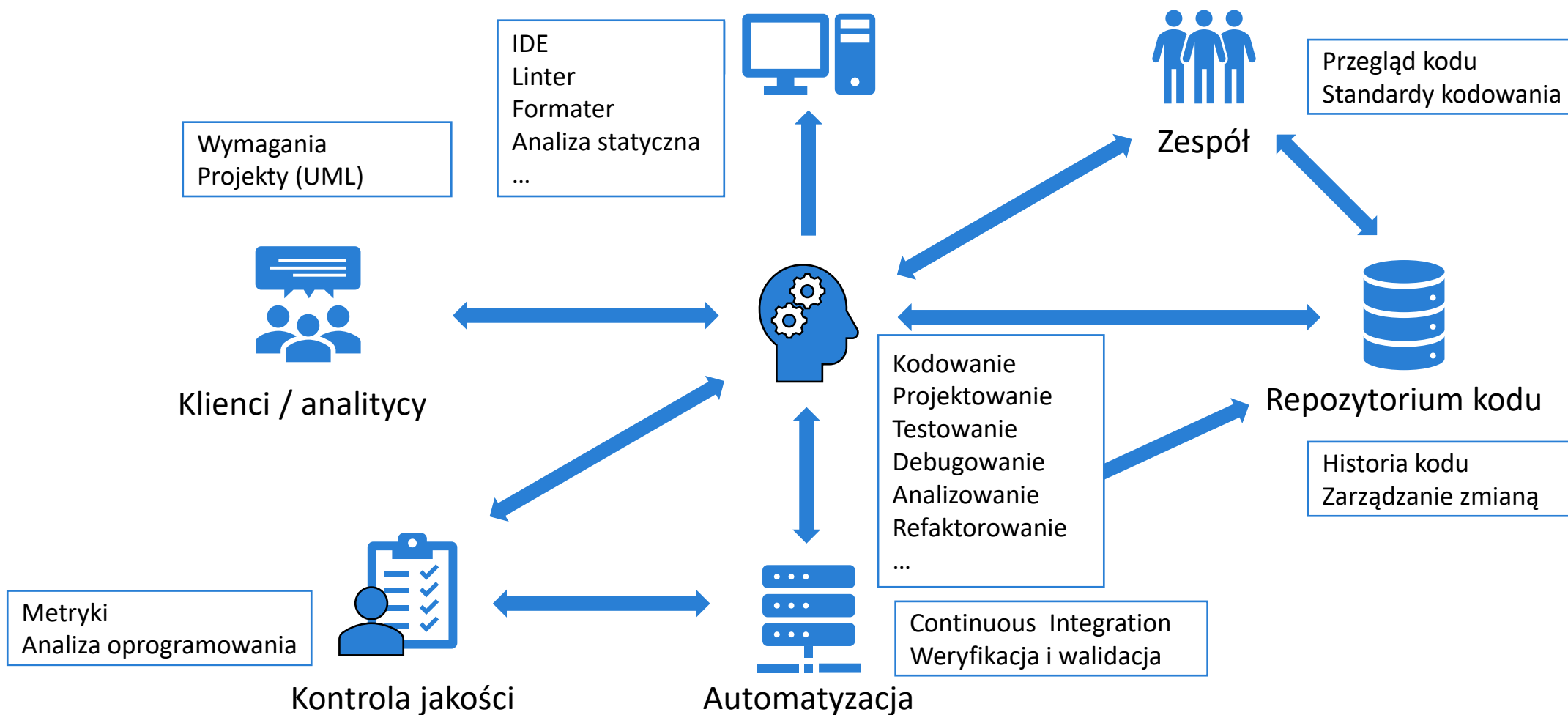
› *Bardzo* uproszczony szkic



› Jest wiele modeli jak dokładniej przebieg wytwarzania powinien wyglądać – tutaj tylko wstęp



Praca inżyniera oprogramowania





Warsztat

- › Mnogość narzędzi zaczyna mieć sens...
- › Standardy kodowania są jednym z narzędzi



Standardy kodowania

- › Cel: usystematyzowanie procesu wytwarzania kodu i samego kodu
- › Ujednolicenie – usprawnienie utrzymywalności (*maintainability*) i pracy w zespole
- › Zbiór reguł ma pomagać osiągnąć zakładaną jakość
- › *Coding conventions, coding standards, coding style* – stosowane często zamiennie, ale czasem ktoś rozróżnia *style* od *standard*
- › Bo *coding standard* to może być więcej niż głębokość wcięć...
 - Czy *tabs vs spaces*
- › Mini dygresja:
 - dostosowanie się do obowiązującego *coding standardu* stanowi test „dojrzałości”
 - usprawnianie standardów to cecha lidera i eksperta, ale konsensus jest kluczowy



Standard kodowania - Styl

- › Czyli: wcięcia, pozycje *klamerek*, białe znaki, białe linie etc.

```
int* a           vs  int *a
while( a==b )    vs  while (a == b)
if (x) {         vs  if (x)
                  {
```

- › Niektóre języki *proponują* style opracowane przez twórców
np. Python (PEP8), C# (C# Coding Conventions)
- › I na szczęście coraz częściej raczej się go *wybiera*, niż *wymyśla*
- › Wbrew pozorom *nie musi* być składnikiem standardu



Standard kodowania - Nazewnictwo

› Zarówno *notacja* jak i *semantyka*

Nazwa rzeczownik;	vs	LPSTR str1;
rzeczownik->czasownik();	vs	str1->m_len;
bool isManager;	vs	bool manager;
DługaNazwa	vs	długa_nazwa



Standard kodowania – co więcej?

- › Reguły komentowania kodu
- › Metryki kodu (długość funkcji, liczba parametrów etc.)
- › Wykorzystywane paradygmaty
(np. czy programujemy obiektowo w C)
- › Reguły bezpiecznego kodu
(np. nie używamy *gołych* wskaźników w C++,
wszystkie parametry funkcji powinny być *const*)
- › Reguły wynikające ze specyfiki przemysłu
(np. zakaz używania dynamicznej alokacji)
- › Wiele innych...



Standard kodowania – narzędzia

- › Zawsze lepiej oddelegować decyzję do narzędzia
 - Mniej konfliktów w zespole
 - Nie musi być idealne, byle konsekwentne
- › Sprawdzanie – **jedna z** funkcji *linterów* (np. flake8)
 - Na laboratorium 2 – clang-format do sprawdzania stylu
 - Na laboratorium 1+ – clang-tidy do „reguł innych niż styl”
- › Autoformatowanie (styl)
 - Wbudowane w IDE
 - Integrowane z IDE
 - Np. clang-format na laboratoriach (nie mylić z clang-tidy!) – zintegrowany z IDE



Standard kodowania – podsumowanie

- › Nie ma *jedynego słusznego poprawnego zawsze*
- › W idealnym świecie jest osiągany przez konsensus zespołu
- › Zespoły mogą używać różnych, a nawet ten sam zespół może używać wielu w różnych projektach
- › Rewolucje w inżynierii rzadko są lepsze od ewolucji (przestarzały standard trzeba powoli i konsekwentnie *przepychać* ku jasnej stronie mocy)
- › Istnieją *formalne* standardy przemysłowe, czasem wymagane w konkretnej branży (np. MISRA C), wtedy mniejszy wybór...
- › Istnieją standardy (reguły) przemysłowe *de-facto*, które można stosować do wielu zestawów: KISS, DRY, GRASP czy SOLID

Dziękuję za uwagę

Konrad.Grochowski@pw.edu.pl

