

# Sztuka Wytwarzania Oprogramowania

## Wykład 1 - wstęp

Robert Nowak

24Z

## Cel i zakres przedmiotu

Przedmiot dotyczący technik i praktyk stosowanych w wytwarzaniu i utrzymaniu oprogramowania. Nacisk położony jest na pracę projektanta kodu i programisty, ukazując kodowanie jako element większej całości.

### Zakres:

- ▶ Potok wytwarzania oprogramowania, translator, repozytorium kodu, debugger, profiler, CI/CD.
- ▶ Styl kodowania, czysty kod, ocena jakości kodu, metryki.
- ▶ Testowanie.
- ▶ Obiektowe wzorce projektowe.
- ▶ Współbieżne wzorce projektowe.
- ▶ Praca z kodem zastanym, refaktoring.
- ▶ Podstawy UML, cykl wytwarzania oprogramowania (wstęp).

## Cel i zakres przedmiotu (2)

Zakładamy znajomość:

- ▶ programowania strukturalnego i obiektowego.
- ▶ algorytmów i struktur danych.
- ▶ języka Python, C++ i C.

Przedmioty powiązane:

- ▶ programowanie strukturalne, obiektowe, funkcyjne, zdarzeniowe, ...
- ▶ **inżynieria oprogramowania** - od analizy wymagań przez projektowanie i wdrażanie. Przedmiot niezbędny dla inżyniera informatyka.

Na przedmiocie SWO poruszamy niskopoziomowe elementy inżynierii oprogramowania.

# Literatura

- ▶ Hunt, Thomas. 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.

# Podstawowe dane o przedmiocie

**Miejsce spotkań:** (poniedziałek 10<sup>15</sup> – 12<sup>00</sup>), sala 170

**Strona przedmiotu:**

<https://staff.elka.pw.edu.pl/~rnowak2/dyd/swo>

**Prowadzący:**

dr hab. inż. Robert Nowak, prof. uczelni,  
[robert.nowak@pw.edu.pl](mailto:robert.nowak@pw.edu.pl)  
mgr inż. Konrad Grochowski,  
[konrad.grochowski@pw.edu.pl](mailto:konrad.grochowski@pw.edu.pl)

**Konsultacje:** stacjonarnie lub online, patrz <http://repo.pw.edu.pl>

# Zaliczenie przedmiotu

kolokwium 1	0 – 10pkt
kolokwium 2	0 – 10pkt
laboratorium	0 – 20pkt
suma	40 pkt

37 – 40 pkt.	ocena 5
33 – 36	4.5
29 – 32	4
25 – 28	3.5
21 – 24	3

**Do zaliczenia SWO konieczne jest uzyskanie min. 10 pkt z laboratorium**

# Laboratorium

- ▶ Realizowane w zespołach 2-osobowych.
- ▶ 4 spotkania, każde trwa 4 godziny.
- ▶ Szczegóły będą przedstawione na wykładzie nr 2 przez Konrada Grochowskiego.

Tematy ćwiczeń:

1. Edytor, repozytorium git, praca w parach, rewizja kodu.
2. Przygotowanie środowiska, zestawienie potoku CI/CD, testy, walidacje.
3. Monitorowanie aplikacji, debuggowanie, profilowanie.
4. Praca z kodem zastanym, refaktoring.

# Narzędzia niezbędne do zaliczenia laboratorium

- ▶ uruchomienie maszyny wirtualnej (np. virtualbox, kvm)
- ▶ konteneryzacja (docker)
- ▶ edytor tekstu (vim)
- ▶ środowiska (clang, visual studio code)
- ▶ kompilator (gcc, g++)
- ▶ narzędzie do budowania aplikacji (make, cmake, scons)
- ▶ debugger (gdb)
- ▶ języki programowania (C++, Python)
- ▶ narzędzia do badania oprogramowania (valgrind, cppcheck, iperf)

# *Podstawowe zasady dobrego stylu programowania*

# Reprezentacja cyfrowa



Komputer może przetwarzać tylko informację dostępną w formie cyfrowej

Poprawna reprezentacja informacji jest bardzo istotna.

**1974 × 2024**

zadanie dla ucznia szkoły podstawowej, ale:

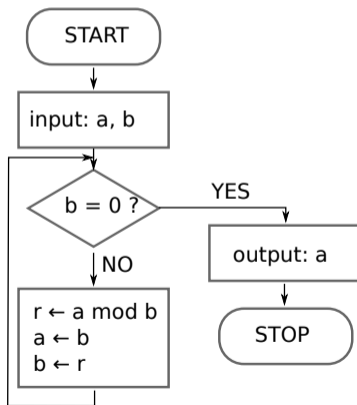
*MCMLXXIV × MMXXIV*

Reprezentacja liczb w kodzie U2:

	0	0	0	0	0	1	0	0	(+4)
+	1	1	1	1	1	1	0	1	(-3)
	0	0	0	0	0	0	0	1	(+1)

# Algorytm

Musimy dostarczyć algorytm, aby komputer mógł rozwiązać problem.



```
Euclid(a, b)                                     // argumenty
begin
    while b > 0 do
        r ← a % b                                // reszta
        a ← b
        b ← r
    end
    return b                                       // result
end
```

# Tworzenia algorytmów i programów

- ▶ Komputer potrzebuje programów (instrukcji)
- ▶ Tylko człowiek może stworzyć program, **silna Sztuczna Inteligencja (strong AI) nie istnieje!**

Programowanie:

- ▶ bardzo kosztowne, wieloletnia praca zespołowa,
- ▶ większość projektów wytwarzania kończy się porażką!

Wiele wysiłku wkłada się w wielokrotne wykorzystanie (reusability).

# Czytelny kod

## KISS (Keep It Simple Software)

- ▶ prosty projekt
- ▶ pojedyncza odpowiedzialność
- ▶ przedkłada się czytelność kodu nad jego optymalność

## Standardy kodowania:

- ▶ pliki źródłowe,
- ▶ nazewnictwo,
- ▶ formatowanie kodu źródłowego,
- ▶ stosować konsekwentnie ten sam styl kodowania.

# Stosowanie narzędzi

## Wykrywanie błędów w programach

- ▶ Wykorzystywać kompilator.
- ▶ **Zawsze** doprowadzić do czystej kompilacji (bez żadnych ostrzeżeń) na najwyższym poziomie.
- ▶ Rozumieć każde ostrzeżenie.

## Zarządzanie kodami źródłowymi (stosowanie repozytorium)

## Systemy do automatycznej kompilacji

## Optymalizacja oprogramowania

- ▶ nie optymalizować jeżeli nie ma takiej potrzeby
- ▶ skupić się na złożoności obliczeniowej (  $O(\cdot)$  )
- ▶ używać narzędzi (profiler) do wykrywania czasochłonnych fragmentów kodu

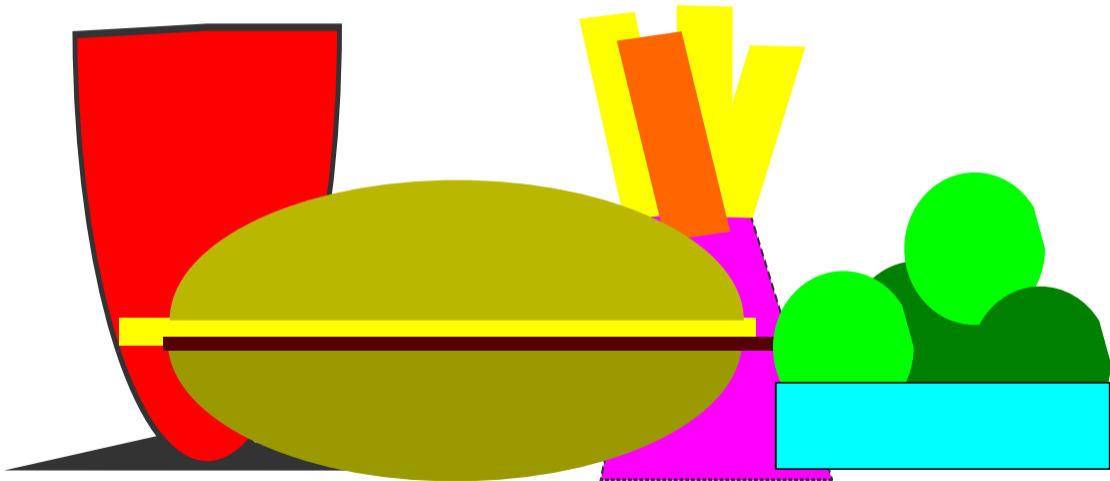
# Pewne fakty związane z tworzeniem oprogramowania

Najczęstsze przyczyny niepowodzenia projektu:

- ▶ niestabilne wymagania
- ▶ optymistyczna estymacja kosztów (głównie czasu) realizacji projektów
- ▶ niska jakość oprogramowania

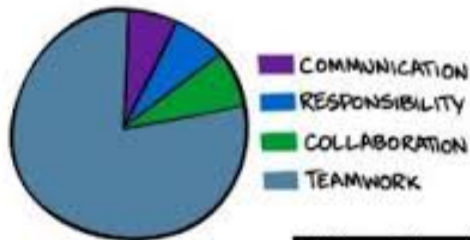
\*R. Glass, Frequently Forgotten Facts about Software Engineering, 2001

# Jakość w tworzeniu programowania



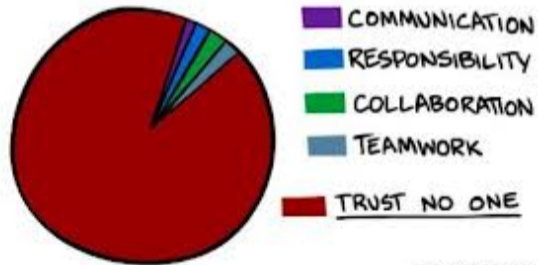
# Po co prowadzi się projekty zespołowe na studiach

WHAT GROUP PROJECTS ARE  
SUPPOSED TO TEACH YOU



Endless Origami

WHAT GROUP PROJECTS TAUGHT ME



endlessorigami.com

\*<http://endlessorigami.com>

## Typowe warunki pracy programisty

- ▶ praca z kodem zastanym - nie zdarza się tworzenie aplikacji od początku
- ▶ posługiwanie się paletą narzędzi, w tym różnymi językami programowania
- ▶ wirtualizacja, konteneryzacja

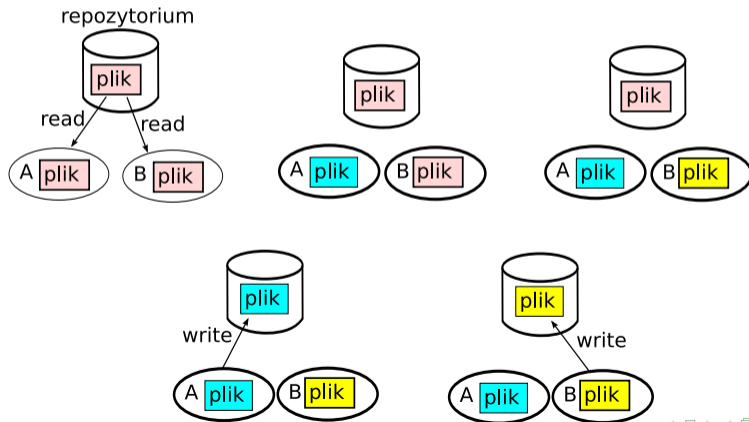
Dlaczego C++ na laboratorium:

- ▶ standard języka jest tworzony przez organizację standaryzacyjną ANSI/ISO (nie przez firmę)
- ▶ wspiera wiele paradygmatów: programowanie strukturalne, obiektywne, generyczne
- ▶ łatwo pokazać testowanie, analizę kodu (bo język statyczny), znajdowanie błędów (bo język kompilowany do kodu binarnego)
- ▶ można pokazać wpływ kodu na wydajność

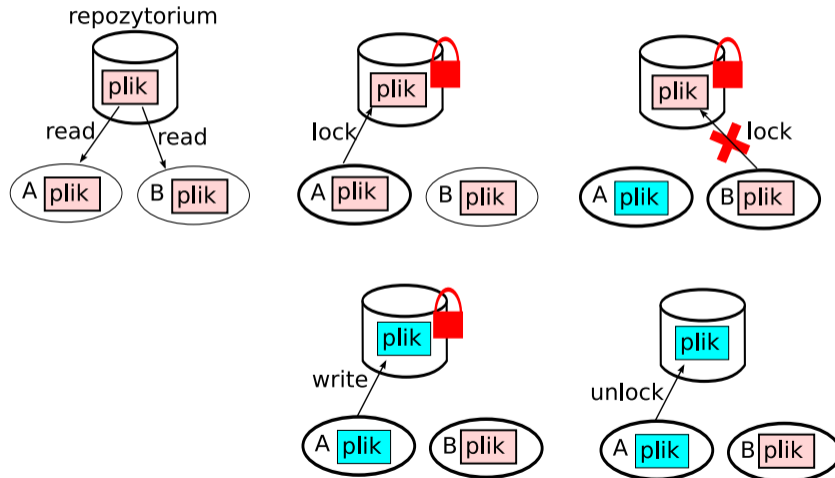
# *Repozytorium kodu*

# System zarządzania wersjami, repozytorium

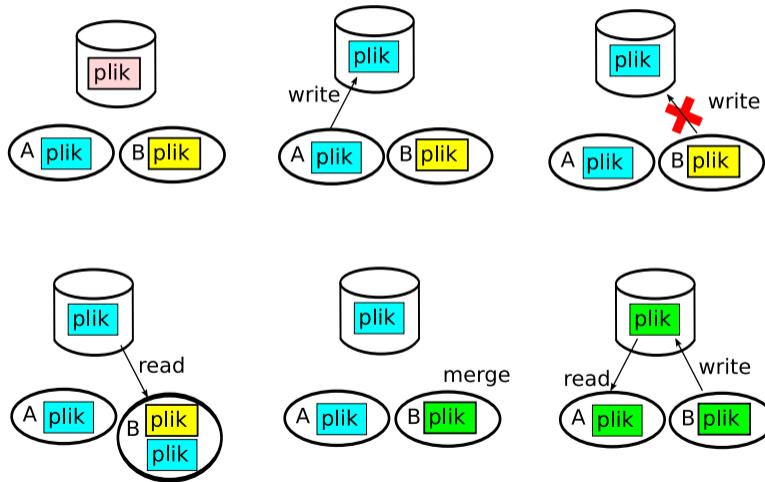
serwer plików, przechowuje kolejne wersje plików i katalogów, umożliwia wycofywanie zmian i pracę grupową



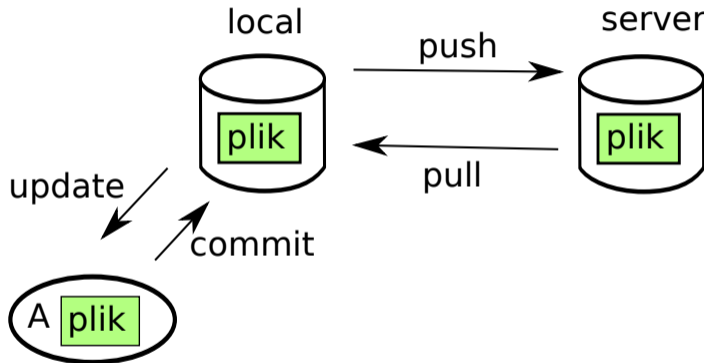
# Repozytorium - blokowanie



# Repozytorium - rozstrzyganie konfliktów



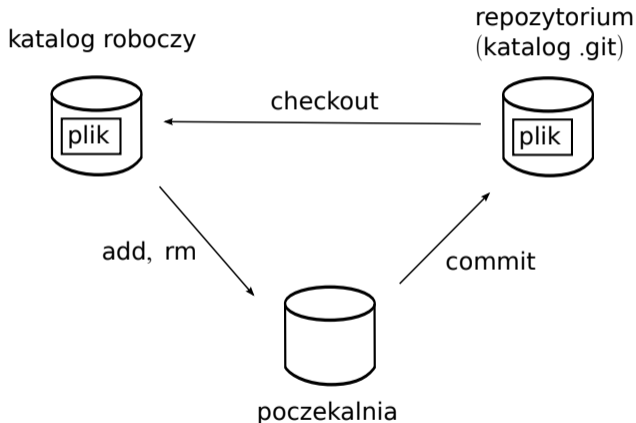
# Repozytorium - serwer lokalny i zdalny (publikowanie)



- ▶ git - commits are snapshots
- ▶ mercurial - commits are diffs

# Git - podstawy (1)

Git posługuje się trzema bazami: katalogiem roboczym, poczekalnią i repozytorium lokalnym.



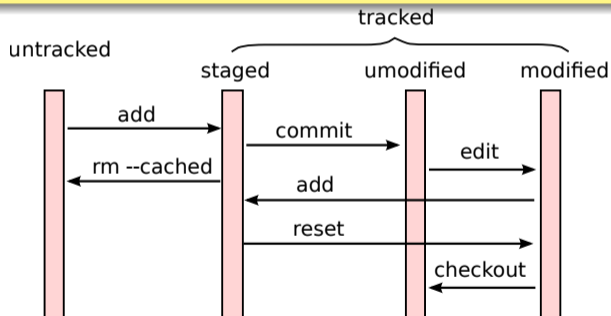
`git status` # drukuje aktualny stan

## Git - podstawy (2), pliki

Git dodaje dane, nie modyfikuje.

Każdy plik jest:

- ▶ nieśledzony
- ▶ śledzony
  - ▶ bez-zmian
  - ▶ zmodyfikowany
  - ▶ w poczekalni



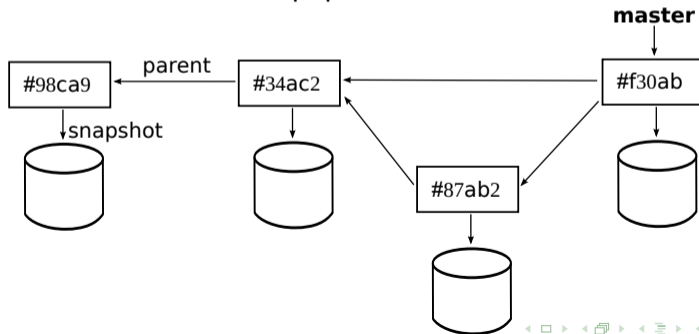
## Git - podstawy (3), commit

Git przechowuje rzuty bazy (migawki), a nie różnice.

Zatwierdzenie (commit) tworzy:

- ▶ rzut bazy (migawkę)
- ▶ meta-dane (klucz, autor, ..., wskaźniki na poprzedników, wskaźnik na rzut)

Gałąź (branch) to wskaźnik na meta-dane.



# Opis narzędzia SCons

- ▶ `www.scons.org`
- ▶ opis projektu to skrypt Pythona
- ▶ bardzo łatwo rozszerzalny
  - ▶ automatyczna analiza zależności
  - ▶ wsparcie dla: C++, Java, Fortran, Yacc, Lex, Qt, LaTeX, Subversion, msvc (zastępuje .vcproj, .sln) i in.
  - ▶ Linux, AIX, Windows NT, Mac OS X, Solaris i in.
  - ▶ wsparcie dla kompilacji równoległej
  - ▶ detekcja zmiany zawartości plików przy użyciu sygnatury MD5 (lub tradycyjnie timestamps)

# SCons - tworzenie aplikacji

```
# plik SConstruct  
Program(source=['calc.cpp', 'main.cpp'], target = prog)
```

Windows (Visual Studio Command Prompt):

```
cl /Fo calc.obj /c calc.cpp /nologo  
cl /Fo main.obj /c main.cpp /nologo  
link /nologo /OUT:prog.exe calc.obj main.obj
```

Linux (gnu g++):

```
g++ -o calc.o -c calc.cpp  
g++ -o main.o -c main.cpp  
g++ -o prog calc.o main.o
```

# SCons - tworzenie biblioteki

```
# plik SConstruct
SharedLibrary(source=['x.cpp', y.cpp'], target = bib)
```

Windows (Visual Studio Command Prompt):

```
cl /Fo x.obj /c x.cpp /nologo
cl /Fo y.obj /c y.cpp /nologo
link /nologo /dll /out:bib.dll /implib:bib.lib x.obj y.obj
```

Linux (gnu g++):

```
g++ -o x.os -c -fPIC x.cpp
g++ -o y.os -c -fPIC y.cpp
g++ -o libbib.so -shared x.os y.os
```

# Debugger

program służący do uruchamiania innych programów

- ▶ zatrzymywanie działania badanego programu w określonym miejscu
- ▶ wykonywanie pojedynczych instrukcji
- ▶ wyświetlanie zawartości zmiennych

**Przykłady:** GNU debugger (gdb), Microsoft Visual Studio, inne

## Narzędzie do wysyłania komunikatów (np. Boost.Log)

- ▶ badaniu stanu aplikacji
- ▶ często jedyny dostępny mechanizm do śledzenia błędów (gdy nie ma możliwości użycia debuggera)

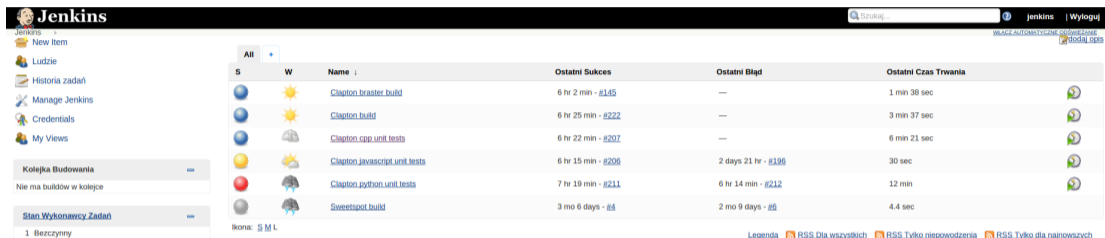
```
#include <boost/log/trivial.hpp>

int main(int, char*[]) {
    BOOST_LOG_TRIVIAL(debug) << "A debug severity message";
    BOOST_LOG_TRIVIAL(info) << "An informational severity message";
    BOOST_LOG_TRIVIAL(warning) << "A warning severity message";
    BOOST_LOG_TRIVIAL(error) << "An error severity message";
    return 0;
}
```

obserwatorzy: konsola, plik, NT Event Loggers, UNIX Syslog

alternatywne rozwiązania: pahteios, glog, log4cxx, log4cpp

# Ciągła integracja (Continuous integration) - jenkins



The screenshot shows the Jenkins web interface. On the left is a sidebar with navigation links: New Item, Ludzie, Historia zadań, Manage Jenkins, Credentials, and My Views. Below these are sections for 'Kolejka Budowania' (Build Queue) and 'Stan Wykonawcy Zadań' (Agent Status). The main area displays a table of recent builds. The table has columns for status (S), weather icon (W), name, last success, last failure, and last duration. The builds listed are Clapton braster build, Clapton build, Clapton.cpp unit tests, Clapton javascript unit tests, Clapton.python unit tests, and Sweetspot build. Each row includes a link to the build details and a small icon representing the build's status.

S	W	Name	Ostatni Sukces	Ostatni Błąd	Ostatni Czas Trwania
		<a href="#">Clapton braster build</a>	6 hr 2 min - <a href="#">#145</a>	—	1 min 38 sec
		<a href="#">Clapton build</a>	6 hr 25 min - <a href="#">#222</a>	—	3 min 37 sec
		<a href="#">Clapton.cpp unit tests</a>	6 hr 22 min - <a href="#">#207</a>	—	6 min 21 sec
		<a href="#">Clapton javascript unit tests</a>	6 hr 15 min - <a href="#">#206</a>	2 days 21 hr - <a href="#">#196</a>	30 sec
		<a href="#">Clapton.python unit tests</a>	7 hr 19 min - <a href="#">#211</a>	6 hr 14 min - <a href="#">#212</a>	12 min
		<a href="#">Sweetspot build</a>	3 mo 6 days - <a href="#">#4</a>	2 mo 9 days - <a href="#">#6</a>	4.4 sec

ikona: S M L

Legenda RSS Dla wszystkich RSS Tylko niepowodzenia RSS Tylko dla najnowszych

obserwacja zmian w repozytorium, uruchamianie budowania, testów jednostkowych, testów funkcjonalnych

- ▶ wiele obsługiwanych repozytoriów
- ▶ narzędzia do budowania dla różnych środowisk
- ▶ rozbudowany zbiór wtyczek, mechanizmy ułatwiające ich instalację
- ▶ interfejs użytkownika - przeglądarka www

# *Obiektowe wzorce projektowe*

- ▶ standardowe rozwiązania często pojawiających się problemów projektowych
- ▶ sprawdzone w praktyce
- ▶ najczęściej dotyczą programowania obiektowego
- ▶ znajomość wzorców projektowych pozwala lepiej zrozumieć obiektowe podejście do programowania

Termin 'wzorce projektowe' upowszechnił się po wydaniu książki „Design Patterns: Elements of Reusable Object-Oriented Software” autorstwa Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides” (1994), zawierająca 23 wzorce.

# Tworzenie nowych klas

Budowa klas na podstawie już istniejących (wielokrotne wykorzystanie kodu):

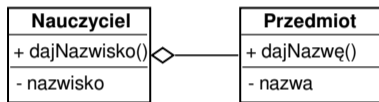
- ▶ agregacja,
- ▶ dziedziczenie.

**Agregacja: prostsza kontrola - należy ją faworyzować**

Język UML (Unified Modeling Language) - diagram klas.

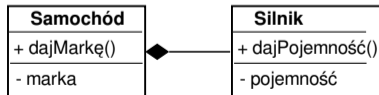
# Agregacja - relacja „składa się z” lub „posiada”

Agregacja- gdy budowa z mniejszych kawałków większej całości.



```
class Przedmiot { /* ... */ };
class Nauczyciel {
private:
    std::vector<Przedmiot*> przedm_;
};
```

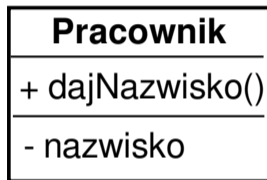
Kompozycja - agregacja, gdy obiekt składowy nie może istnieć bez obiektu głównego



```
class Silnik { /* ... */ };
class Samochod {
private:
    Silnik silnik_;
};
```

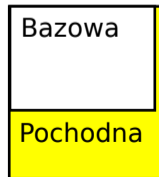
# Dziedziczenie - relacja „może być traktowany jako”

Dziedziczenie wprowadza powiązanie pomiędzy typami.



```
//klasa bazowa
class Pracownik {
    //...
};

//klasy pochodna
class Kierownik : public Pracownik {
    //...
};
```



# Zalecenia końcowe

- ▶ Czytać kod innych.
- ▶ Programować. Programowanie to zajęcie praktyczne.

**Polecana lektura: Chacon, Straub. Pro Git. 2014.**

# *Dziękuję*

robert.nowak@pw.edu.pl