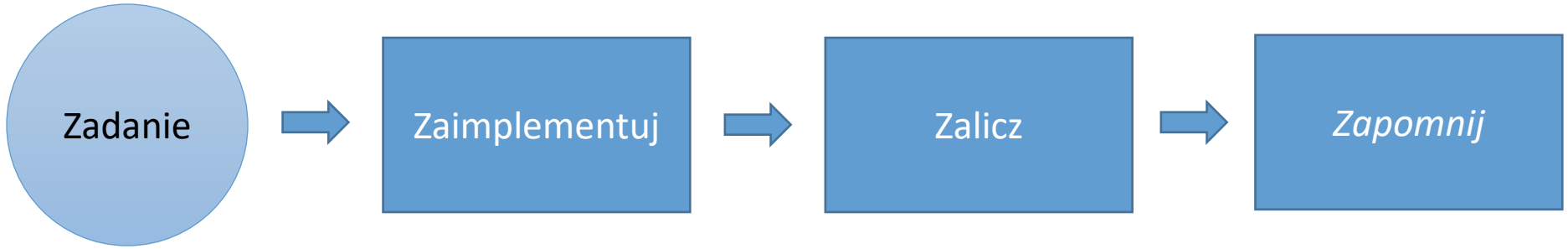
Cykl wytwarzania (życia) oprogramowania

Sztuka Wytwarzania Oprogramowania, w. 6

Konrad Grochowski

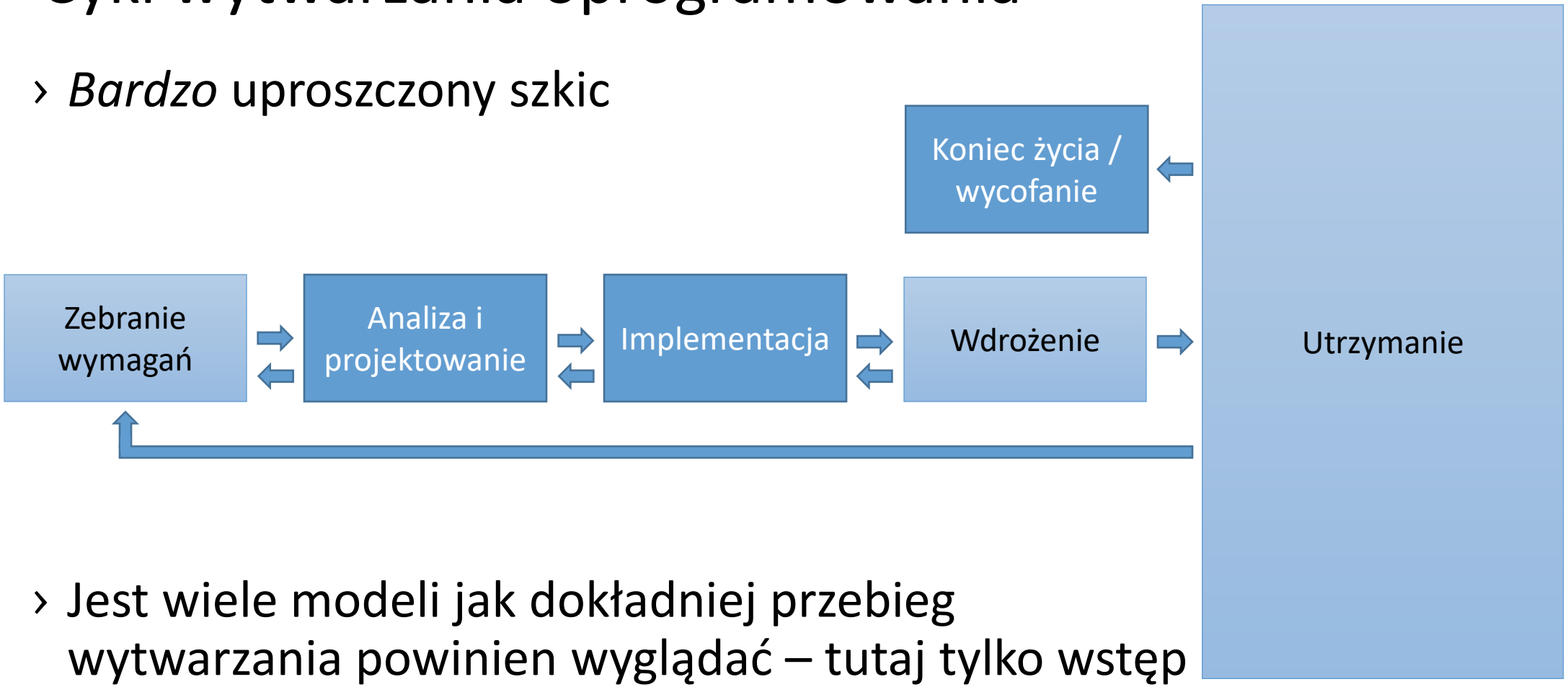
Cykl życia oprogramowania na studiach

- 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

Automatyzacja cyklu

- Teraz powinniśmy porozmawiać, jakie są różne podejścia do wytwarzania oprogramowania i różne konstrukcje cyklów
- ... ale zrobimy mały przeskok, do tematów bliższych następnemu laboratorium
- ... i porozmawiamy o narzędziach wspomagających pracę z jakimś cyklem wytwarzania

Automatyzacja cyklu

- Mamy zdefiniowany jakiś proces wytwarzania oprogramowania
- Jak tylko pojawia się proces to każdy ~~leniwy~~ porządny inżynier zadaje sobie pytanie:
  jak mogę zmusić maszyny, by mi w tym procesie pomogły?
- Czyli *co można zautomatyzować?*
- Ale czy to faktycznie tylko *lenistwo*?

Automatyzacja procesu a jakość

- *Rozsądny* poziom formalizacji procesu zazwyczaj pomaga osiągnąć zakładane cele (a dokładniej – bez formalizacji nie ma procesu, bez procesu nie ma powtarzalności i przewidywalności)
- Najlepszą formalizacją jest automatyzacja, szczególnie jeśli zawiera obiektywną i jednoznaczną ocenę zgodności z procesem
- We współczesnych procesach wytwarzania oprogramowania stara się automatyzować jak największą część procesu
  (choć *code review* nic nie zastąpi jak długo kod będzie robiony przez ludzi)

Continuous Integration

- Proces ciągłej integracji i weryfikacji zmian wprowadzanych do projektu (często stosowana z *feature branch*)
- Ma zapobiegać „zepsuciu” głównej gałęzi kodu
- Dostarcza informacji zwrotnej autorom zmian – wcześniejsze wykrywanie błędów
- Może przeprowadzać walidacje niedostępne programistom „na co dzień” (kompilacja kilkunastoma kompilatorami etc.)
- Nie jest złotym środkiem na wszystkie problemy –
  jest tak dobry, jak dobre są testy – napisane przez zespół...

Continuous Integration

- Co może robić?
  - kompilować kod
  - analizować kod statycznie
  - uruchamiać testy jednostkowe i zbierać pokrycie
  - uruchamiać testy walidacyjne / akceptacyjne
  - generować dokumentację
  - przygotowywać pakiety instalacyjne
  - czekać na krok wykonany przez człowieka
  - co tylko się chce, jak długo jest to realizowalne przez narzędzie, które da się uruchomić „automatycznie”
- Sekwencję tych operacji nazywa się *potokiem CI (CI pipeline)*

Continuous Delivery / Continuous Deployment

- Takie wykorzystanie *potoku,* które gwarantuje, że zawsze mamy produkt, który możemy dostarczyć klientowi
- W szczególnych przypadkach „dostarczenie” może być nawet częścią samego potoku
- Raczej niespotykane w oprogramowaniu krytycznym, ale sama idea „zawsze dobrego stanu repozytorium” jest bardzo dobra

Continuous Integration - realizacja

- Najczęściej narzędzie „sprzęgnięte” z systemem kontroli wersji
  - Na przykład – każdy *git push* uruchamia proces CI
- Może wymagać dużo zasobów (więcej niż maszyna programisty)
- Stabilne/odtwarzalne środowisko
  - Pomocna bywa wirtualizacja
  - Albo konteneryzacja (*Docker* – uruchamianie aplikacji Linuxowych w odseparowanym środowisku w ramach tego samego systemu)
  - Często „w chmurze” (to też dotyczy sytuacji z dużą ilością zasobów)
- Proces dzieli się na samodzielne „małe” kroki, dla przejrzystości
- Niezależne zadania można zrównoleglić

Continuous Integration - narzędzia

- Współczesne systemy „opakowujące” kontrolę wersji często oferują wbudowany system do CI/CD
  - GitLab CI/CD
  - GitHub Actions
  - Bitbucket Pipelines
- Istnieją dedykowane narzędzia
  - Jenkins
  - Travis CI
  - Circle CI
  - AppVeyor
  - …

Dziękuję za uwagę

Konrad.Grochowski@pw.edu.pl