Cykl wytwarzania (życia) oprogramowania

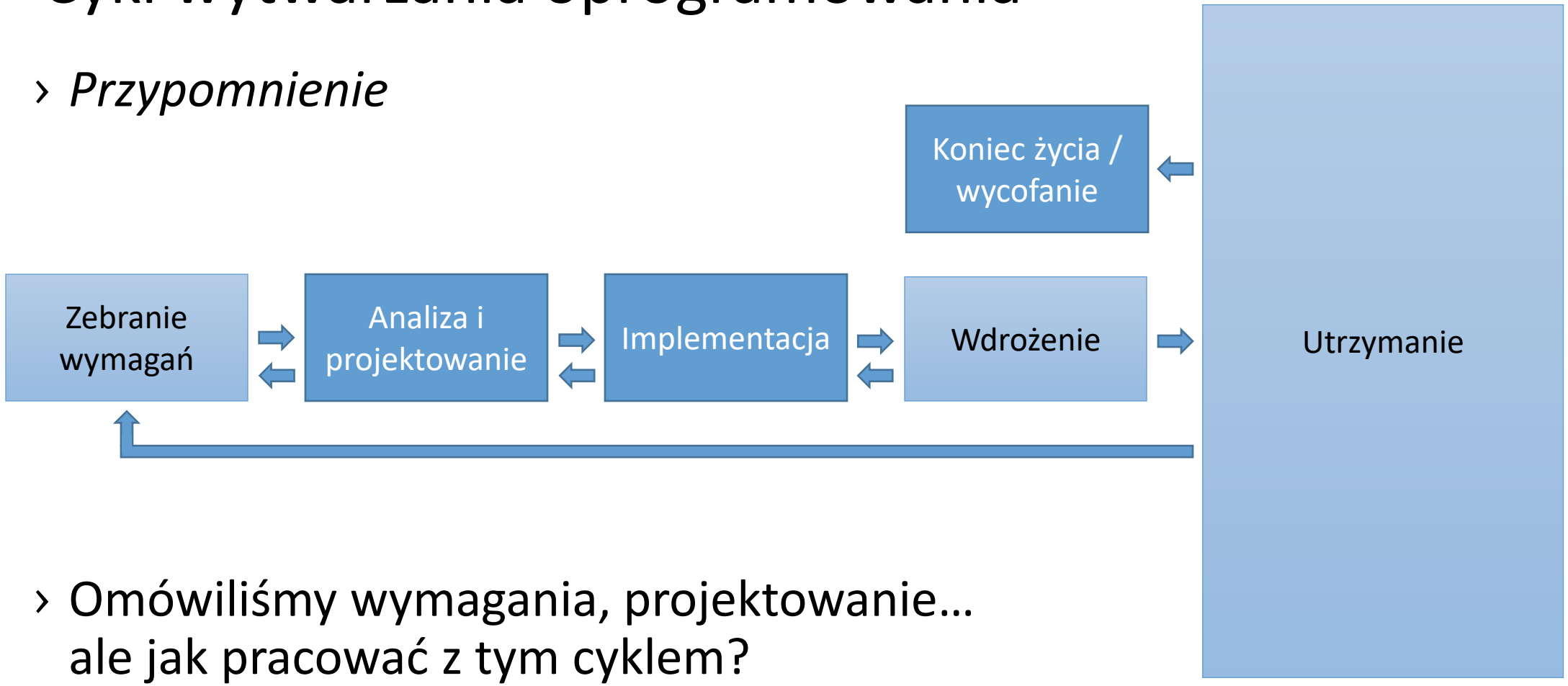
Sztuka Wytwarzania Oprogramowania, w. 5

Konrad Grochowski

Instytut Informatyki, Politechnika Warszawska, 2025 ©

Cykl wytwarzania oprogramowania

› *Przypomnienie*

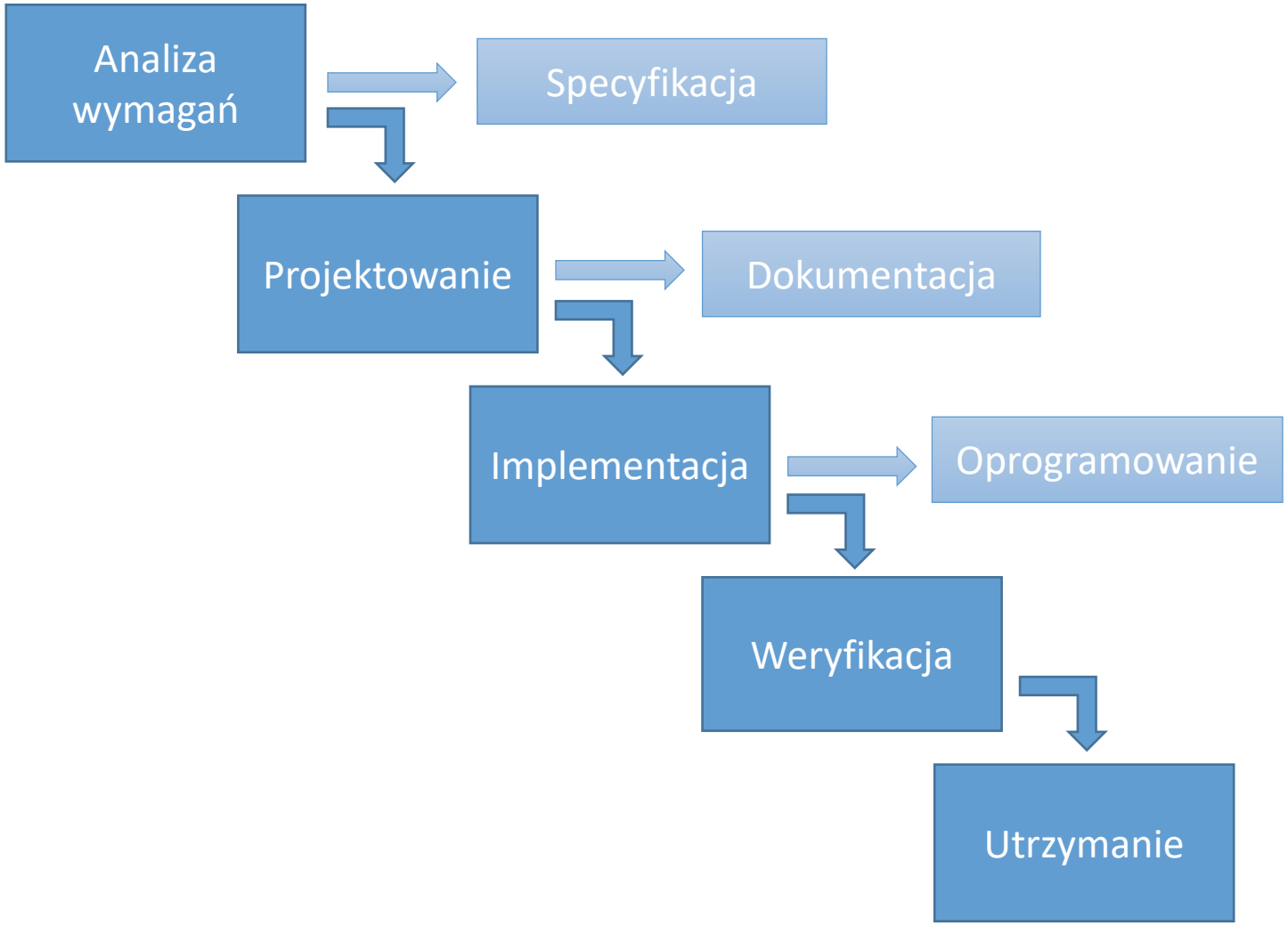


› Omówiliśmy wymagania, projektowanie…
ale jak pracować z tym cyklem?

Co dalej?

› Istnieją różne *modele* realizacji cyklu życia oprogramowania

Model wodospadowy (waterfall)



Waterfall - cechy

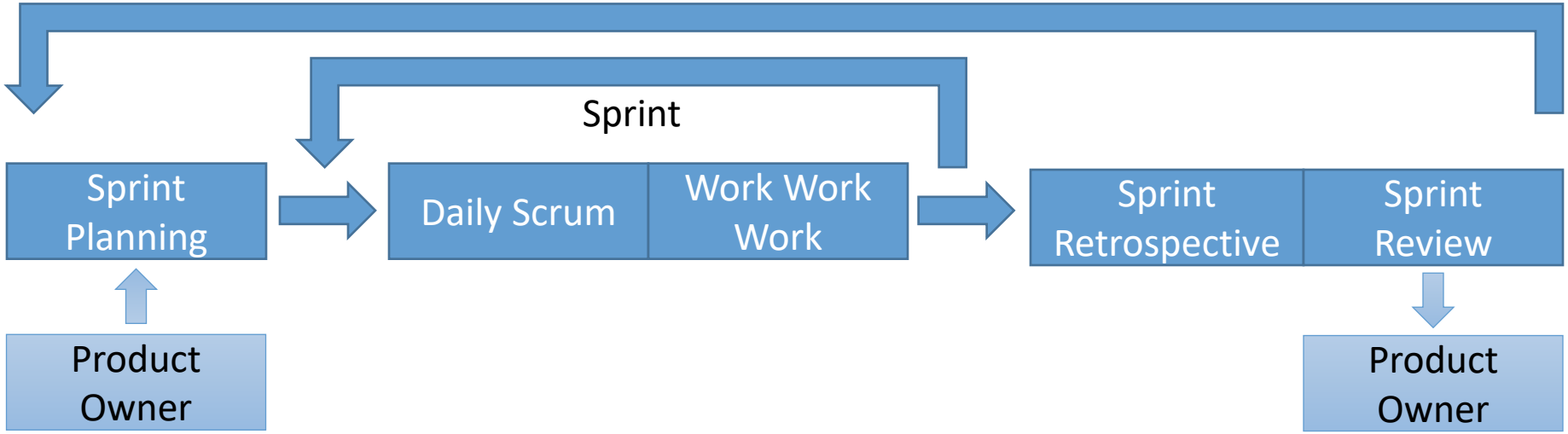
- Dzieli proces na łatwe do zrozumienia i rozdzielenia etapy
- Dostarcza pośrednie efekty pracy
- ... i zdaniem wielu - nie działa (otwarta pętla sterowania)
  - czy też – istnieje, by udowadniać, że inne metody są lepsze
- Ale jest domyślnym sposobem postępowania przy kontraktach o „ustalonej cenie” (*fixed price*)
  - szczególnie w szeroko rozumianej tzw. sferze budżetowej
  - duże doświadczenie wytwórcy w konkretnych problemach *może* działać
- Dlatego bardzo często występuje w formach *zmodyfikowanych*
  - Jakieś formy iteracji, podzielenia na pod-problemy, nachodzenia faz etc.

Metodyki zwinne (Agile)

- W pewnym sensie – drugie ekstremum
- Krótkie cykle „od wymagań do wdrożenia”
  - liczone w dniach, nie miesiącach
- Interakcja z klientem na każdym etapie
- Oprogramowanie praktycznie non-stop „gotowe do wdrożenia”
  - To powiązane z CI/CD (ale CI to narzędzie, a nie metodyka)
- Hipotetycznie – *zwinna* reakcja na zmiany / elastyczność
- ... ale wymaga tej elastyczności także od klienta
- ... także w kosztach

SCRUM

- Krótkie cykle – Sprint (zazwyczaj 2 – 4 tyg.)
- Klient / reprezentant interesów klienta – Product Owner
- Opiekun i „kapitan” zespołu – Scrum Master
- *Codzienna* obserwacja procesu



Tak dla odmiany ...

- ... nie ma procesu z definicji *najlepszego*
- Agile wydaje się „naturalniejszy”
  - Sprawniej radzi sobie z krótkoterminowymi problemami
- Ale jak np. *zacząć* projekt sprintami?
  - To problem wielu przyrostowych modeli
- Co z projektami będącymi częścią większego przedsięwzięcia?
  - Terminy, konieczna wymiana dokumentacji etc.
- Co z projektami o ustalonym budżecie?
  - Zarówno Waterfall jak i Agile twierdzą, że obniżają koszty tego drugiego
- Często występuje w efekcie jakaś mieszanka
  - Osobiście *marzy mi się* prawdziwy SCRUM ze *świadomym* klientem
- Dostosowywać proces do potrzeb, a nie potrzeby do procesu

Agile Manifesto

We are uncovering better ways of developing
software by doing it and helping others do it.
Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while *there is value* in the items on
the right, we value the items on the left more.

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