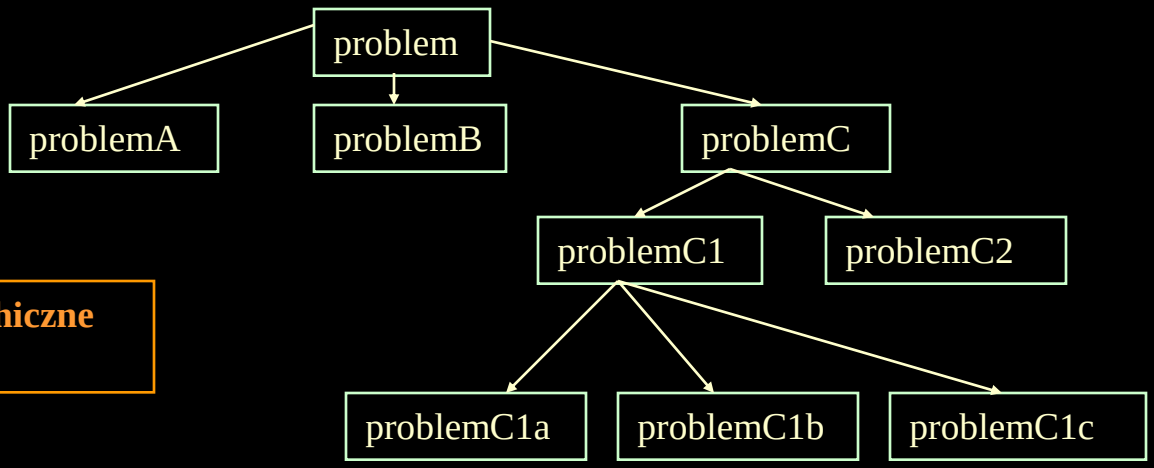
ZASADY TWORZENIA OPROGRAMOWANIA

1. Tylko złożone oprogramowanie wymaga inżynierii (cykl życia składający się z modelowania i testowania oraz sprzężenia zwrotnego – prosty problem, zajęcia z programowania)
2. Złożoność przedsięwzięcia programistycznego stanowi trzon problemu (*niepowodzenie uwarunkowane utratą kontroli intelektualnej*)
3. Ogólną metodą rozwiązywania złożoności jest dekompozycja zagadnienia na części (sztuką jest określenie, jakich? – pułapki!)
   - rozwiązanie poszczególnych części może nie rozwiązać problemu – plan musi przewidywać ich złożenie
   - czasem części mogą być trudniejsze do rozwiązania niż całość?

TWORZENIA OPROGRAMOWANIA

Dziel i zwyciężaj (ogólna metoda rozwiązywania problemów)

1. **Podstawa dekompozycji – części niezależne**
2. **Ograniczenie elementów niezależnych (części –7+-2).**
   Dekompozycja problemu przedstawiona w postaci drzewa hierarchii



Potęga intelektu – hierarchiczne rozwiązywanie problemu

Rozwiązywanie problemów miliony razy trudniejszych niż te które można rozwiązać bezpośrednio.

TWORZENIA OPROGRAMOWANIA
Odbiorcy

3. Każdy etap jest realizowany dla określonego odbiorcy (odbiorca zadań wykonanych na danym etapie znajduje się na etapie następnym – wymagania – twórca – projektant – kodujący – testujący (odbiorca na każdym etapie)
4. Zdolność do abstrakcyjnego myślenia jest bez wątpienia największym talentem ludzkiego rozumu (największe niebezpieczeństwo) z kolei uogólnienie prowadzi do stereotypów
5. Zasada „od rozmycia do skupienia” (wstępny ogólny diagram ma prowadzić do spójnych, jasnych i jednoznacznych specyfikacji – rola inżyniera oprogramowania – system płacowy)
6. Dokumentacja – błędne jest twierdzenie „dobrze wykonana praca dokumentuje się sama” (kodowanie – program bez komentarzy jest prawie nie do zrozumienia), („najbardziej oczywiste informacje raz wykorzystane, nie zapisane, będą utracone”; notuj założenia i o nich pamiętaj, nie powielaj informacji – powinna być zapisana i ewentualnie modyfikowana w jednym miejscu)

TWORZENIA OPROGRAMOWANIA
Dokumentacja

6a. W dokumentacji nie stosuj synonimów i nie rozpraszaj pojęć

7. Wejście/wyjście (we-wy_ jest podstawą oprogramowania

- **określenie wszystkich wejść i wyjść (widocznych i niewidocznych (data czas systemowy)**
- **określenie zakresu dopuszczalnego we-wy**
- **określenie szczególnych wartości we-wy (czek 0)**

TWORZENIA OPROGRAMOWANIA
Nie przesadzaj!!

1. Ma powstawać dobrze funkcjonujący program spełniający oczekiwania odbiorcy.
2. Jeżeli system nie jest uszkodzony nie naprawiaj go („każdy system można poprawiać... doprowadzając go do „śmierci”)
3. Użytkownik docelowy ma decydujący głos w dyskusji na temat funkcji i użyteczności produktu
4. Wykorzystaj poprzednie prace i doświadczenie – nie uszczęśliwiaj użytkownika (pulpit, share documents, dokumentacja).

   Nie wynajduj koła!! Twórz oprogramowanie z myślą jego ponownego wykorzystania.
5. Bądź odpowiedzialny za oprogramowanie które tworzysz.

ZARZADZANIE INŻYNIERIĄ
Rola menedżera

1. Dobry menedżer kierujący pracą inżyniera oprogramowania to osoba zapewniająca odpowiedni sprzęt i izolująca inżynierów od problemów nietechnicznych.

2. Techniczne składniki pracy menedżera to harmonogram, budżet i jakość.

3. Po rozpoczęciu projektu zadanie menedżera zmienia się ze zgadywania tego, co będzie potrzebne na monitorowanie i utrzymywanie kierunku prac

ZARZADZANIE INŻYNIERIĄ
Miary

1. Zarejestrowana historia (podobny projekt realizowaliśmy przez 1 rok) w inżynierii oprogramowania nosi znamiona miary.

2. Powiązanie ilości kodu KDSI (thousand of delivered source code instruction), LOC (line of code) z elementami potrzebnymi do napisania tego kodu:

   Ludzie – ilu i w jakim czasie brało udział w realizacji projektu,

   Czas – jak długi był harmonogram i jego poszczególne etapy

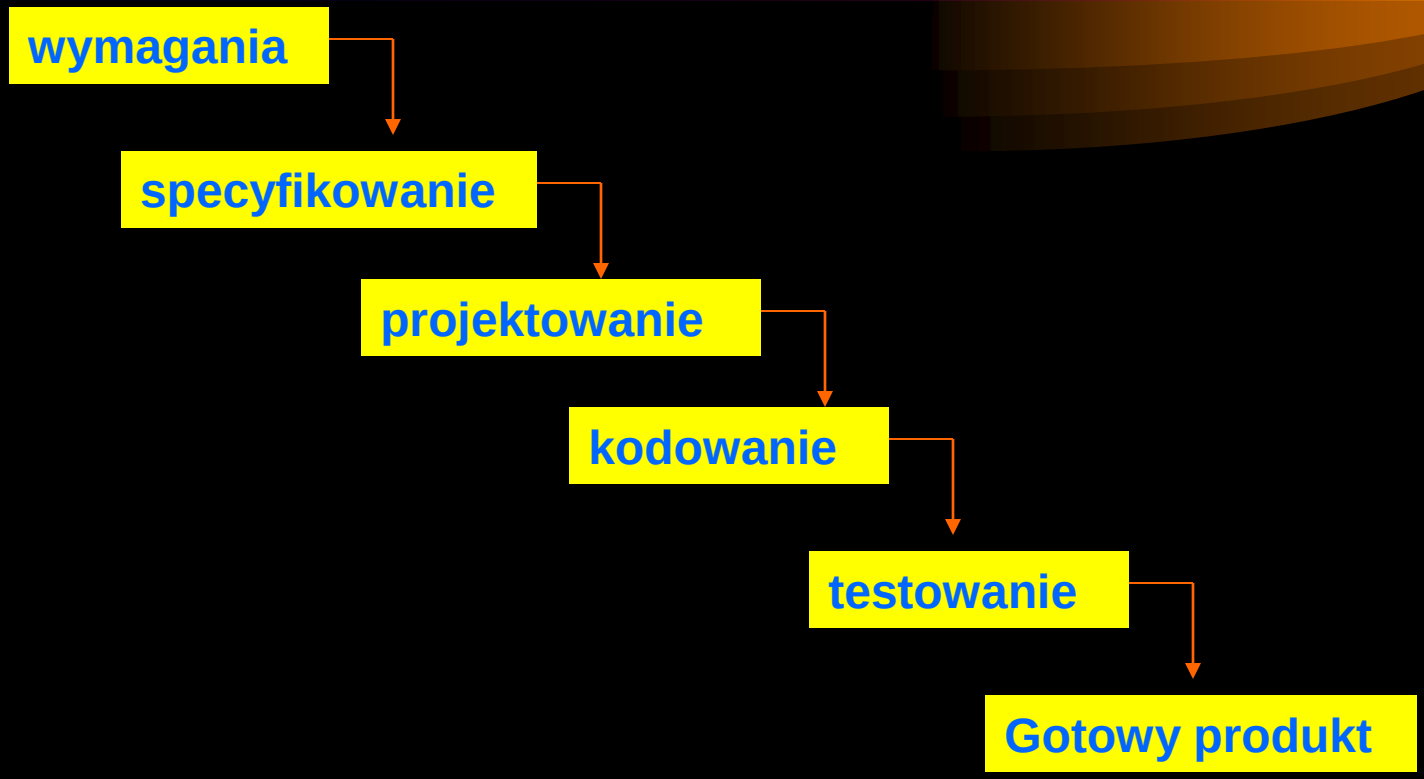
   Sprzęt – jaki zasób rzeczy (komputery, biurka, pokoje, kawa był potrzebny

3. Brak własnych danych historycznych – modele estymacyjne – algorytmiczne (na podstawie wzoru)

„Każda miara może być krytykowana jako niedokładna, ale w porównaniu z zgadywaniem – niemal każdy pomiar jest lepszy”

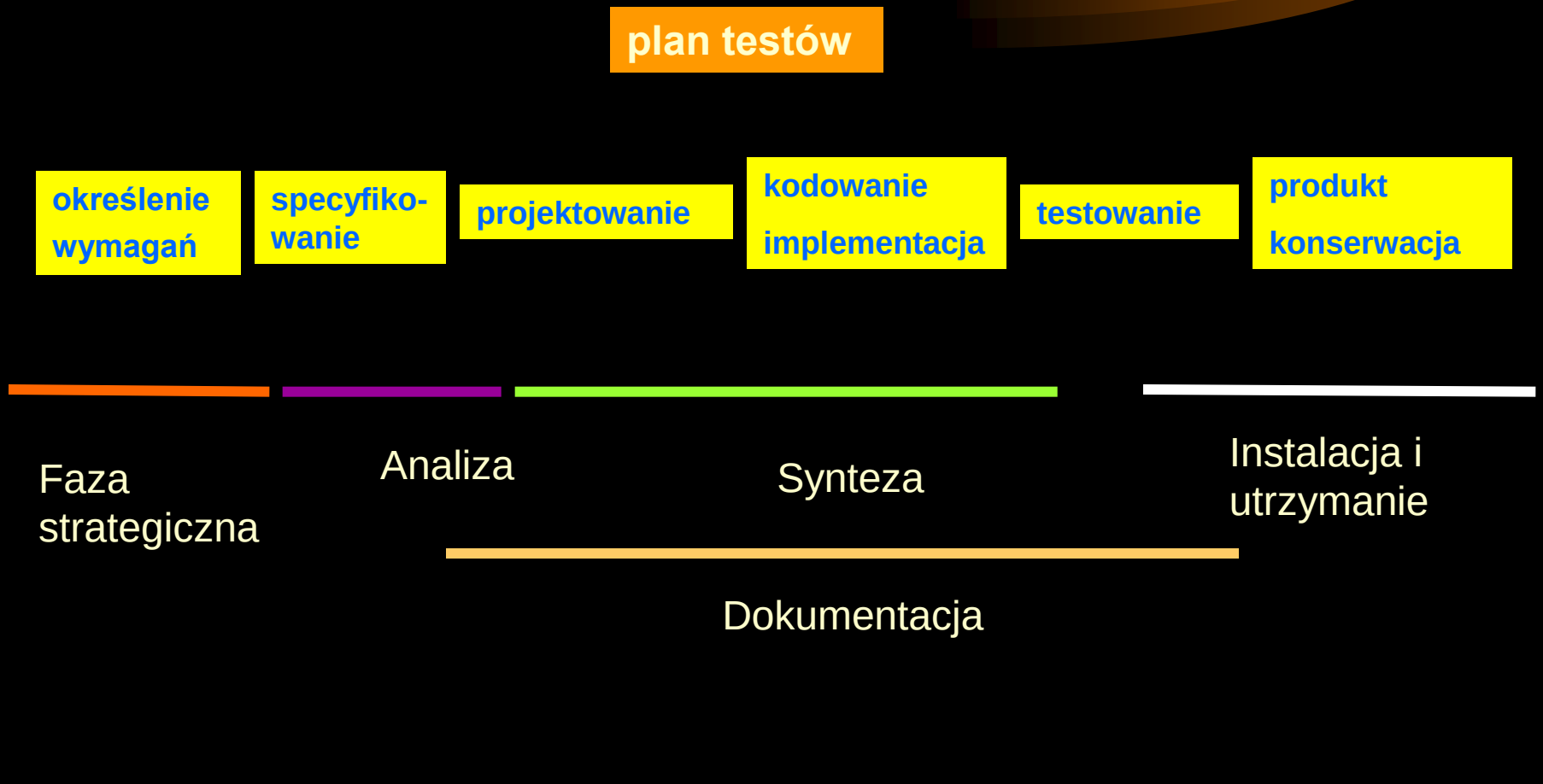
MODELE CYKLU ŻYCIA OPROGRAMOWANIA (1)

Model kaskadowy



MODELE CYKLU ŻYCIA OPROGRAMOWANIA (2)

Fazy w modelu kaskadowym



MODELE CYKLU ŻYCIA OPROGRAMOWANIA (3)

Wady i zalety m. kaskadowego

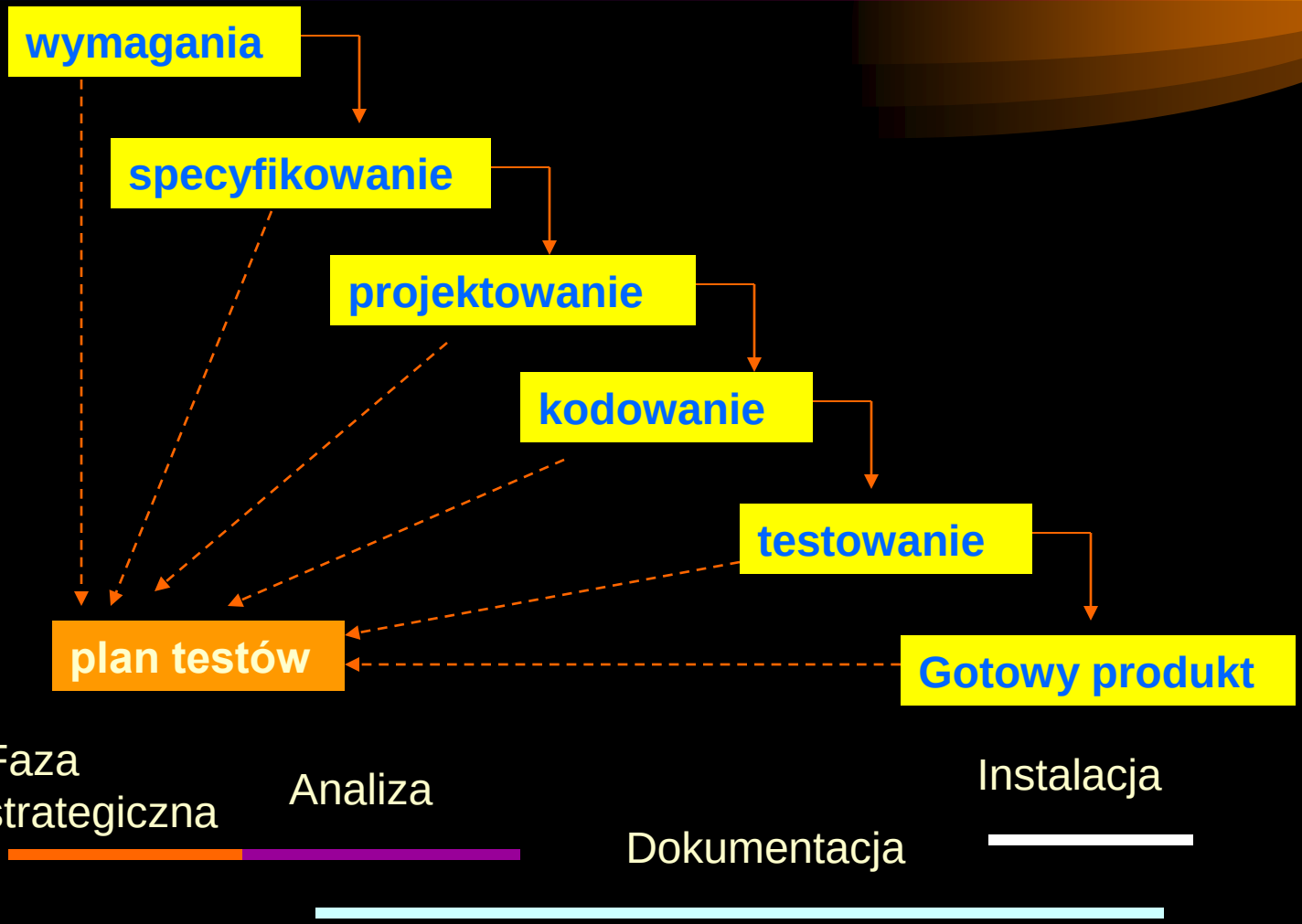
1. **Narzucona twórcom oprogramowania ścisła kontrola kolejności realizacji prac.**

2. **Wysoki koszt błędów popełnionych we wstępnych fazach (błędy popełnione w fazie wymagań i specyfikacji najpewniej zostana odkryte dopiero w fazie testowania)**

3. **Klient (inwestor – użytkownik) bierze udział jedynie we wstępnej fazie projektu (jest mało angażowany) więc jego zainteresowanie produktem słabnie, co ma oczywisty negatywny wpływ na wartościowanie produktu przez Klienta** ***„Najbardziej cenimy to, co sami wykonamy, lub jest realizowane z naszym udziałem”***

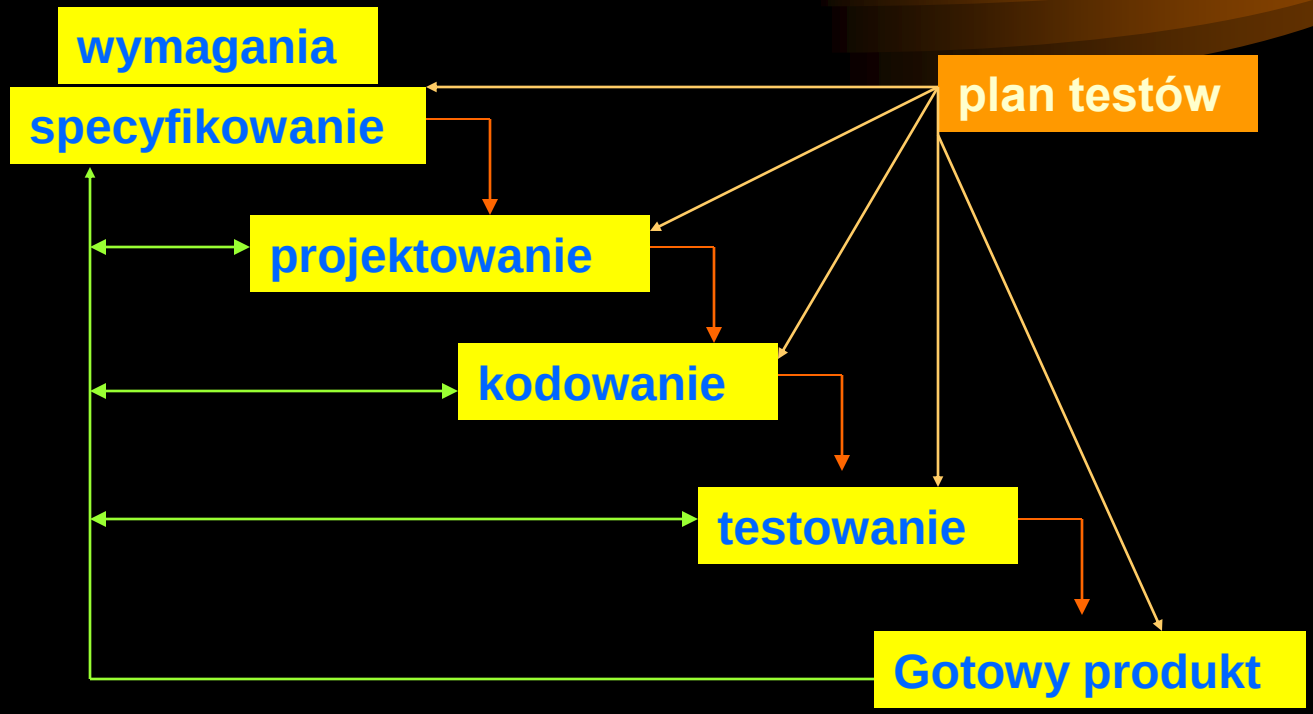
MODELE CYKLU ŻYCIA OPROGRAMOWANIA (4)

Model kaskadowy – rozbudowa testowania



MODELE CYKLU ŻYCIA OPROGRAMOWANIA (5)

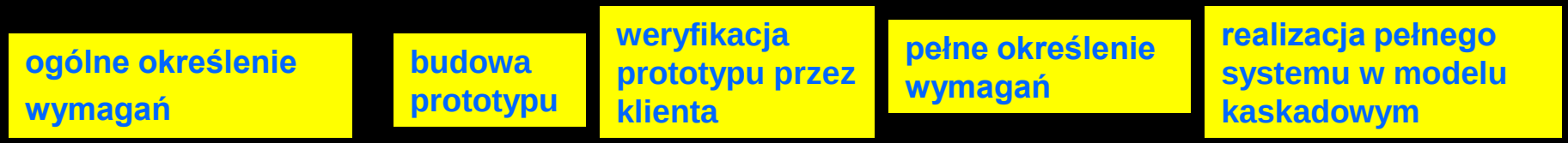
Iteracje w modelu kaskadowym



MODELE CYKLU ŻYCIA OPROGRAMOWANIA (6)

Inne modele

1. **Realizacja kierowana dokumentami – (model armii USA*) – kaskadowy z warunkiem kompletnej dokumentacji każdej fazy (szereg dokumentów weryfikowanych przez klienta) pozwalającej nawet na zmianę firmy programistycznej** ***(dodatkowa praca, przerwy na weryfikację).***
2. **Prototypowanie *(prototyping)* –model pozwalający na minimalizację ryzyka związanego z błędami w fazie wymagań.**



Pozwala na wyeliminowanie nieporozumień klienta i twórców oprogramowania. Prowadzi do wykrycia brakujących funkcji, trudnych usług, braków w specyfikacji wymagań oraz punktów krytycznych systemu. (szybka demonstracja systemu, wstępne szkolenia użytkowników

MODELE CYKLU ŻYCIA OPROGRAMOWANIA (7)

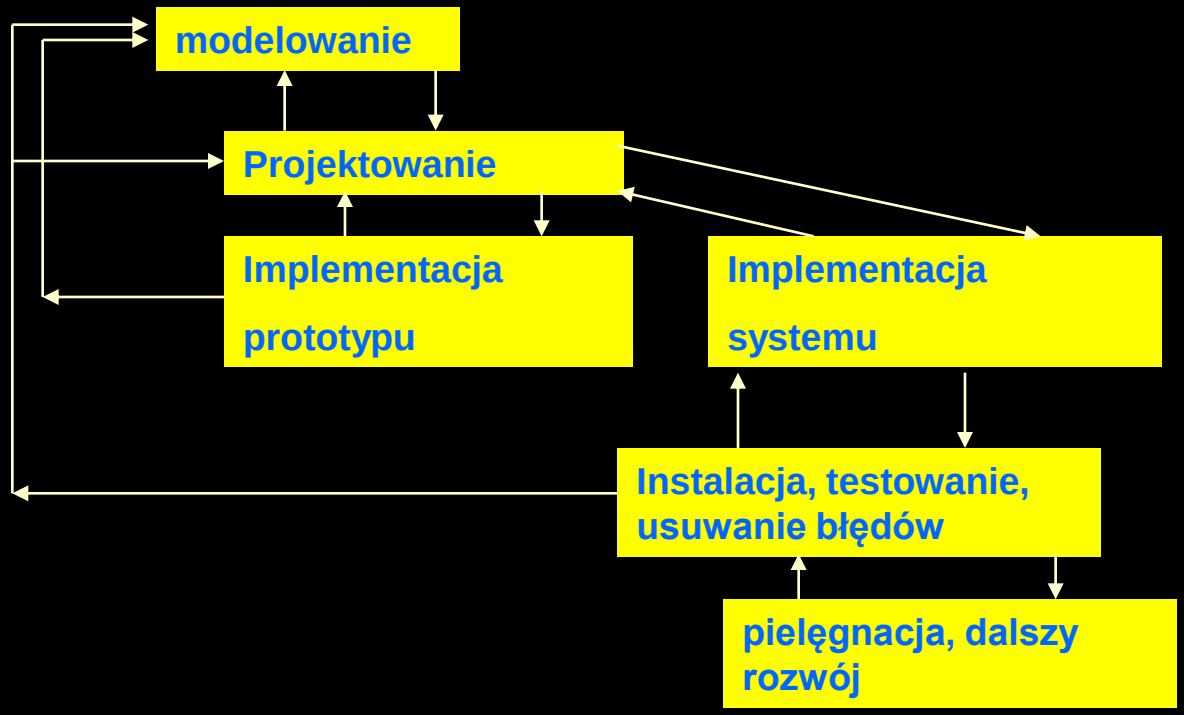
Prototypowanie - techniki

- **Niepełna realizacja – pominięcie elementów oczywistych i skupienie się na wymaganiach trudnych.**
- **Opis systemu w językach wysokiego poziomu: Smalltalk. LISP, wykonywalne języki specyfikacji formalnych.**
- **Wykorzystanie gotowych komponentów**
- **Szybkie programowanie (bez testów i finezji)**
- **Automatyczne generowanie interfejsu użytkownika**
- **Papier i ołówek – prototypowanie interfejsu uzytkownika**
- **Programowanie odkrywcze** *(exploratory programming)* **iteracyjna weryfikacja ogólnego projektu przez klienta – systemy złożone**

MODELE CYKLU ŻYCIA OPROGRAMOWANIA (7)

Inne modele

Model konstrukcji prototypów – złożone systemy o wymaganiach niejasnych lub wieloznacznych – cykliczna realizacja systemu poprzez prototypy weryfikowane przez klienta



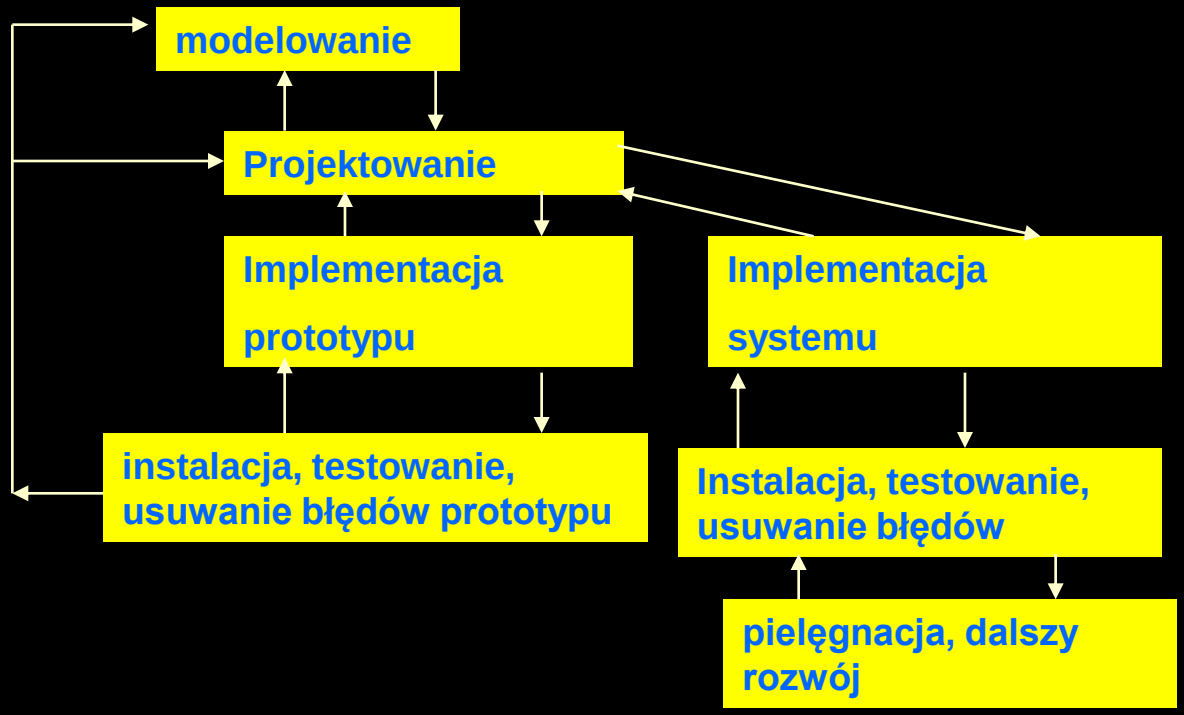
Prototypy prowizoryczne mogą być konstruowane bardzo szybko w małych kosztach

metodologia pozwala na weryfikację wymagań

MODELE CYKLU ŻYCIA OPROGRAMOWANIA (8)

Inne modele

Model ewolucyjnej konstrukcji prototypów – złożone systemy o wymaganiach niejasnych lub wieloznacznych – cykliczna realizacja systemu poprzez prototypy weryfikowany przez klienta poprawiany, testowany i instalowany jako zrealizowany przyrost



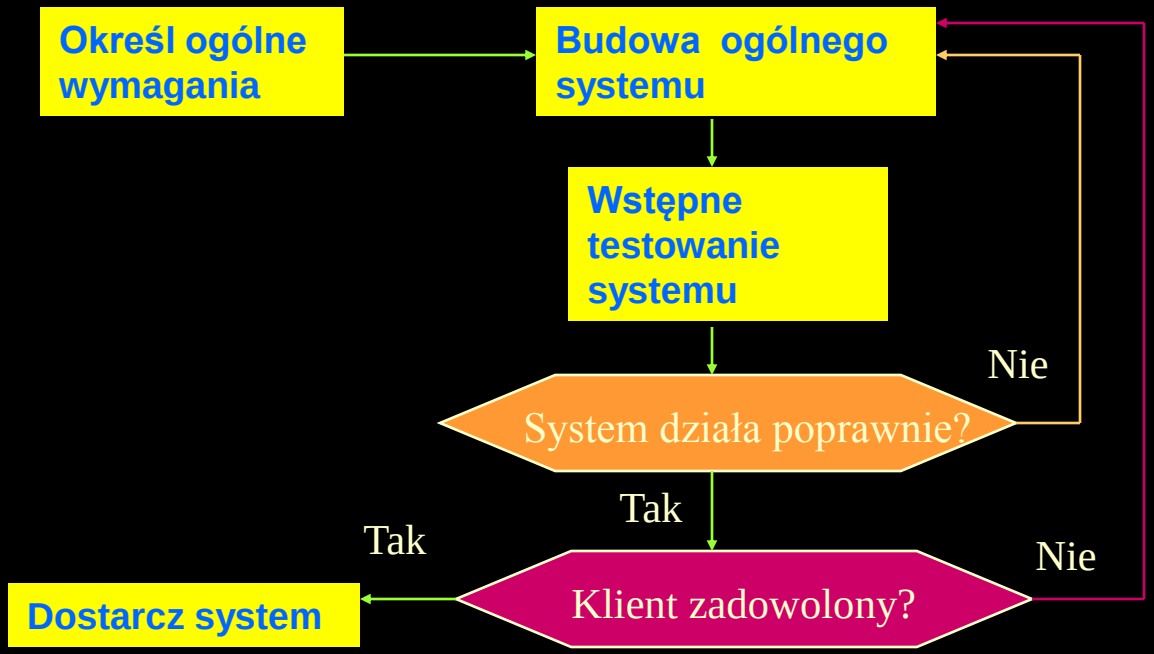
Połączenie podejścia przyrostowego i iteracyjnego

Szybkie dostarczanie niepełnej wersji systemu

MODELE CYKLU ŻYCIA OPROGRAMOWANIA (8)

Inne modele

3. Programowanie odkrywcze – złożone systemy o trudnych do sprecyzowania wymaganiach – cykliczna realizacja systemu ogólnego do wymagań weryfikowanych przez klienta



Model może być stosowany jako „sposób” tworzenia systemu (amatorski).

Profesjonalnie stosuje się go w prototypowaniu

MODELE CYKLU ŻYCIA OPROGRAMOWANIA (9)

Inne modele

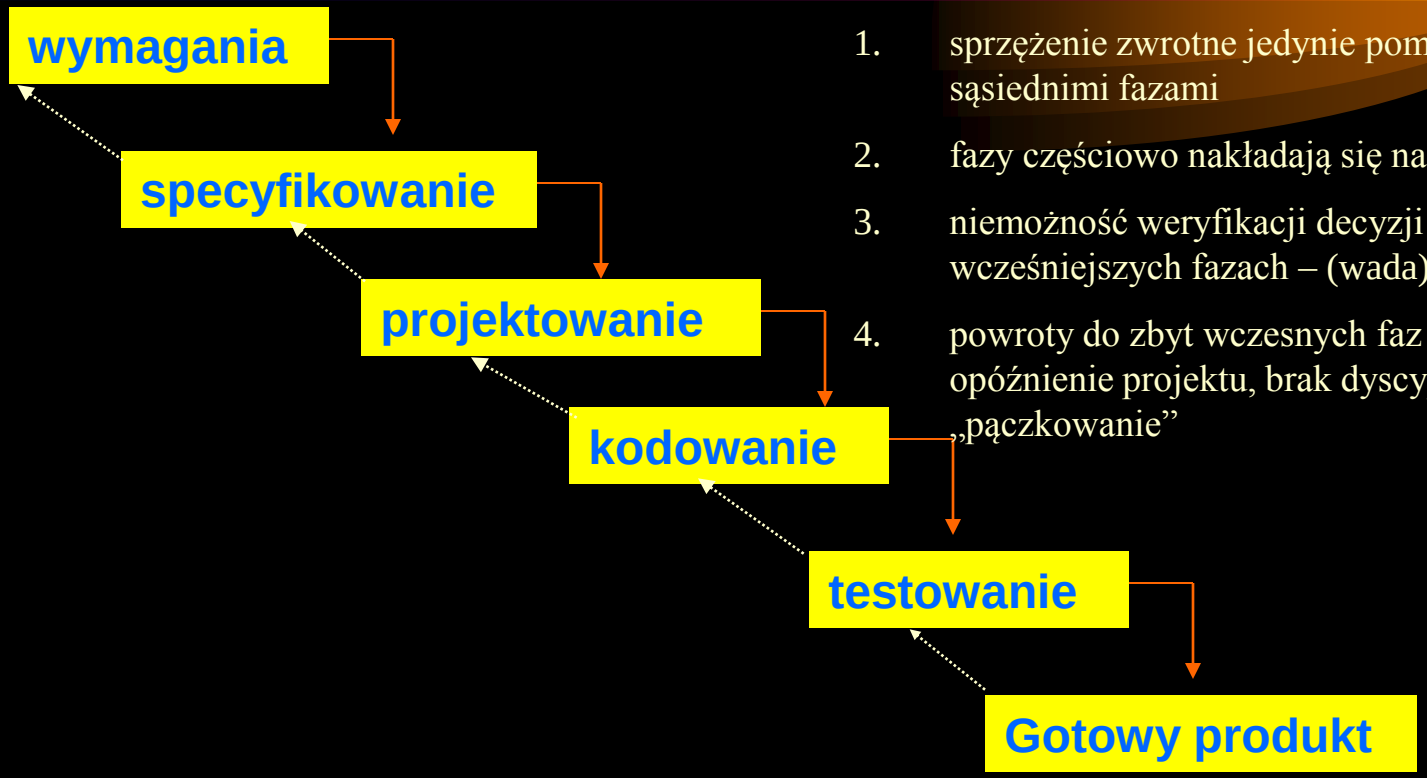
4. **Realizacja przyrostowa** – po określeniu projektu wstępnego następuje iteracyjne testowanie podzbioru funkcji systemu. Następnie jest realizowany projekt szczegółowy tej części i implementacja części systemu realizującej te funkcje.

5. **Montaż z gotowych elementów** - wykorzystywane na różnych etapach realizacji projektu (najczęściej implementacji) : biblioteki, pełne aplikacje języki symboliczne itp. – narzędzia CASE (Computer Assisted/Aided Software/System Engineering)

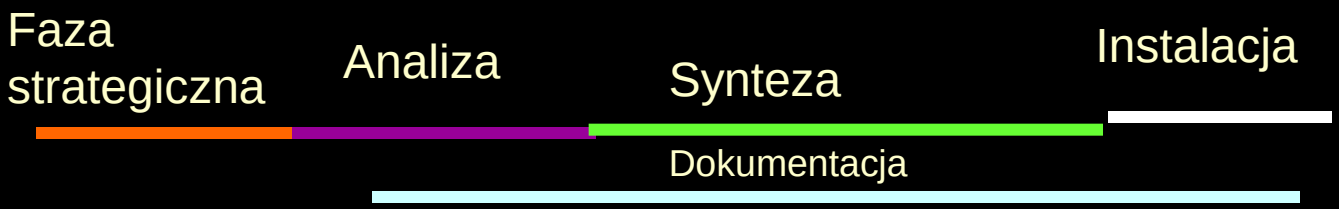
6. **Model spiralny** – (Boehma – 1988) – najbardziej ogólny model cyklu życia oprogramowania

MODELE CYKLU ŻYCIA OPROGRAMOWANIA (1)

Model kaskadowy (często stosowany w praktyce do projektów o niewielkiej złożoności)

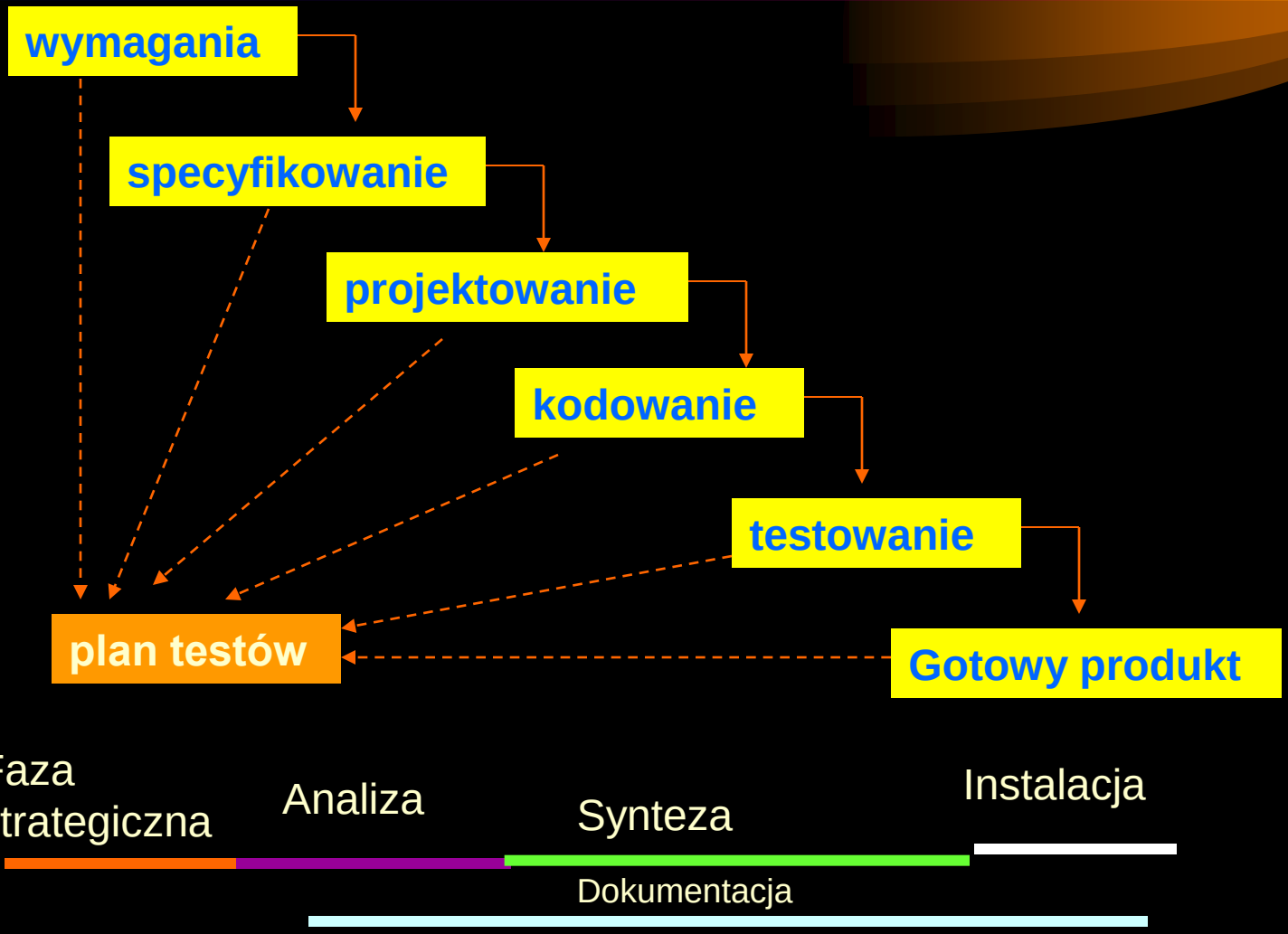


1. sprzężenie zwrotne jedynie pomiędzy sąsiednimi fazami
2. fazy częściowo nakładają się na siebie
3. niemożność weryfikacji decyzji podjętych we wcześniejszych fazach – (wada)
4. powroty do zbyt wczesnych faz powodują opóźnienie projektu, brak dyscypliny i „pączkowanie”



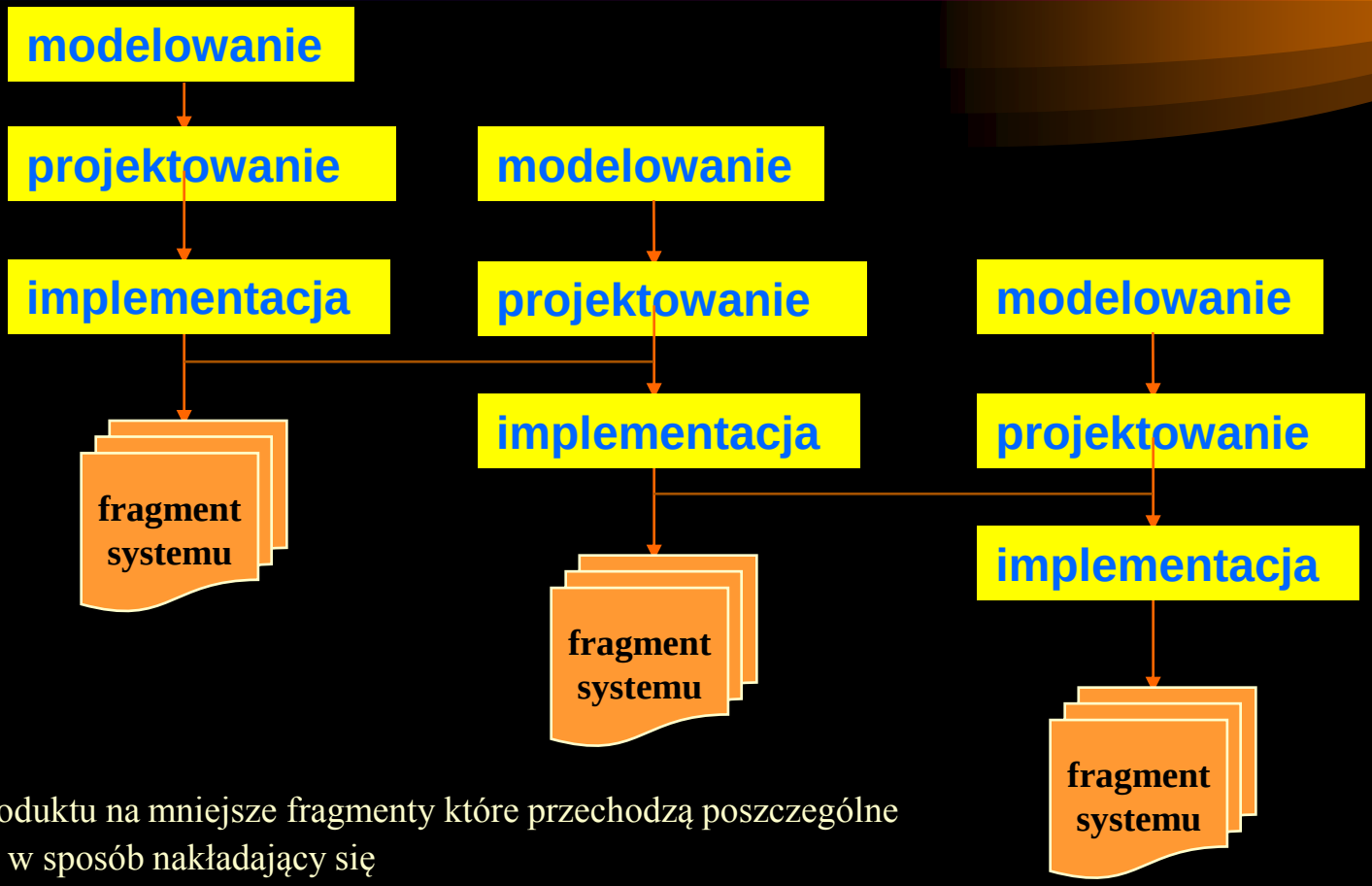
MODELE CYKLU ŻYCIA OPROGRAMOWANIA (4)

Model kaskadowy – rozbudowa testowania



MODELE CYKLU ŻYCIA OPROGRAMOWANIA (2)

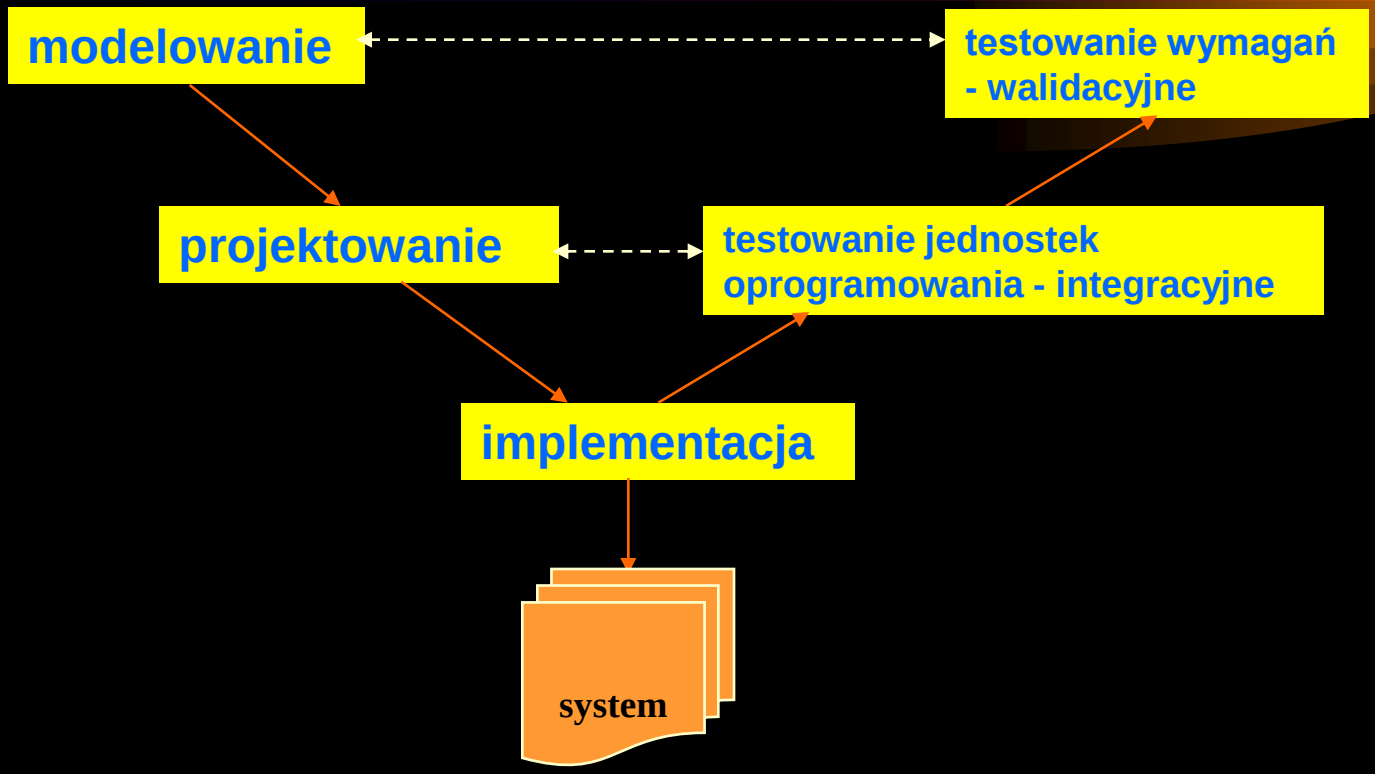
Model przyrostowy (często stosowany w praktyce do modeli iteracyjnych i metodyk)



1. podział produktu na mniejsze fragmenty które przechodzą poszczególne fragmenty w sposób nakładający się
2. konieczność dokładnej definicji interfejsów pomiędzy fragmentami
3. łatwość implementacji w modelach kaskadowych i iteracyjnych

MODELE CYKLU ŻYCIA OPROGRAMOWANIA (3)

Model V (eliminacja niemożności testowania produktu danej fazy)

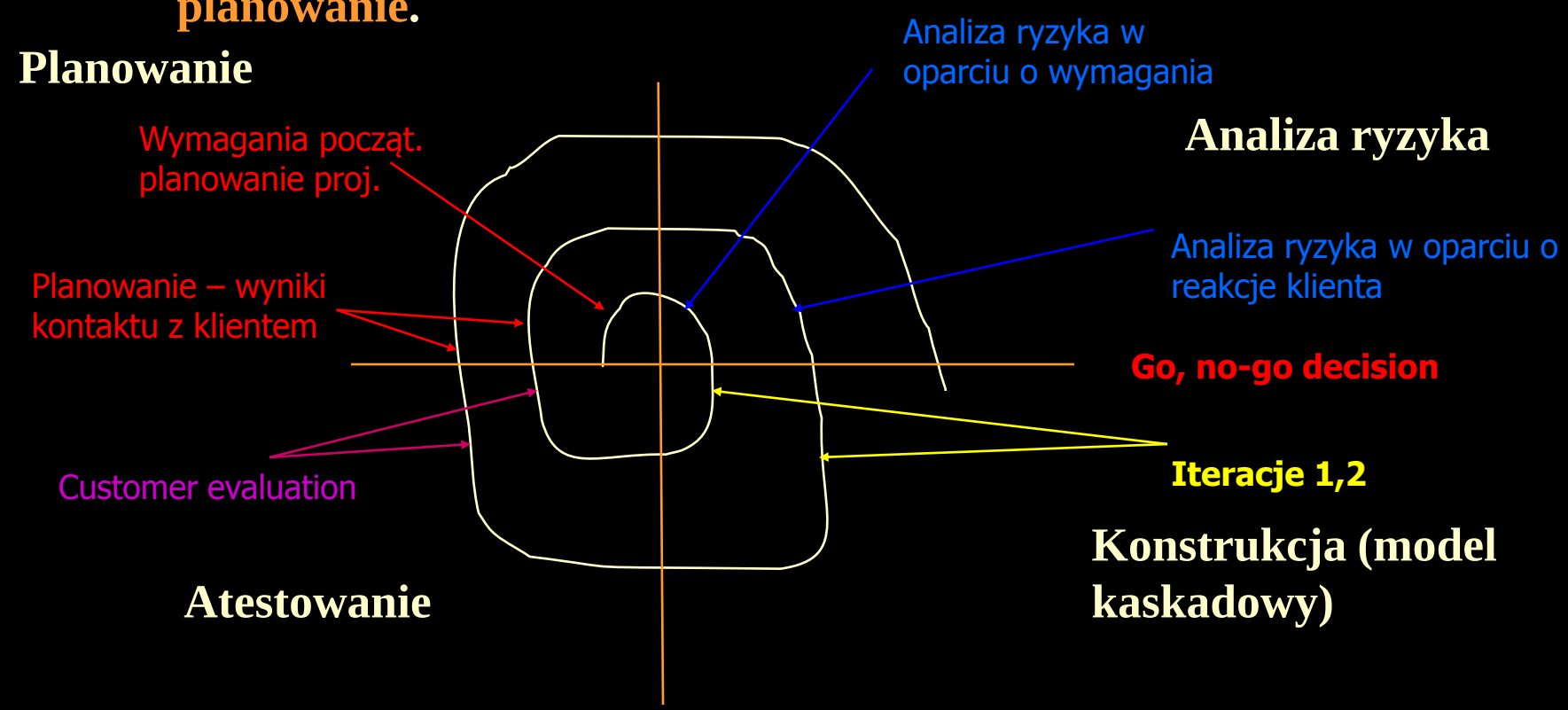


1. udział dwóch zespołów: projektowego i testującego
2. zespół projektowy opracowuje produkty poszczególnych faz – zespół testujący testuje powstające produkty
3. testowanie jest związane z fazami produkcyjnymi

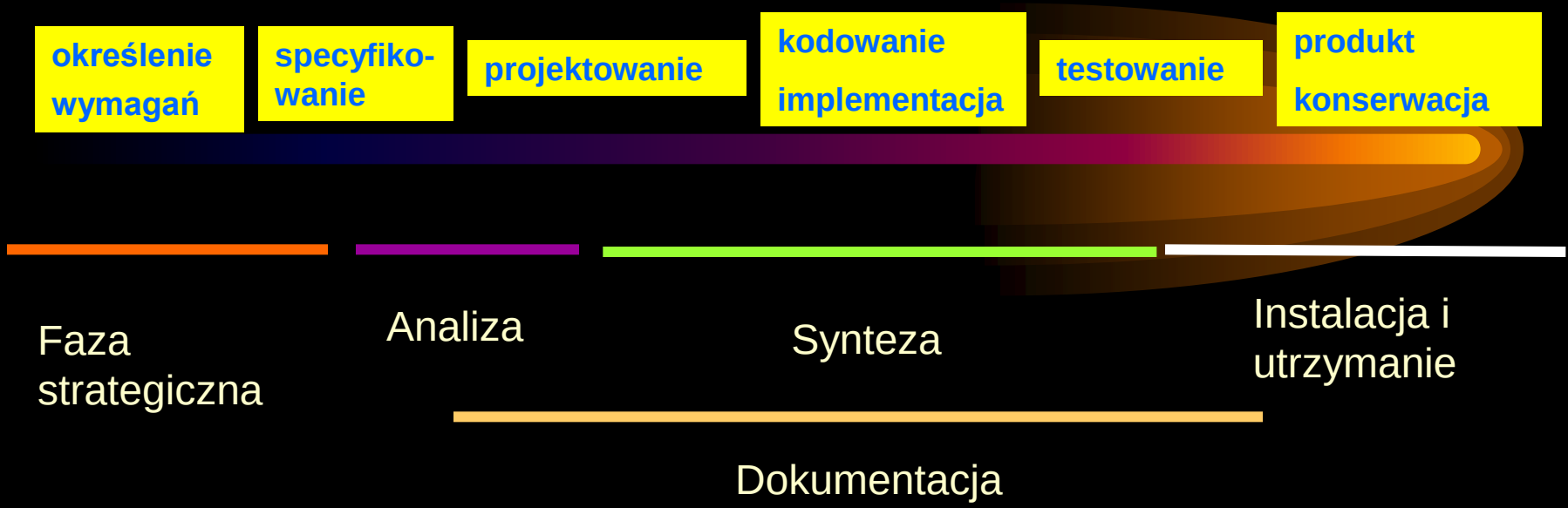
MODELE CYKLU ŻYCIA OPROGRAMOWANIA (10)

Model spiralny

(Boehma – 1988) – cztery główne fazy cyklu życia oprogramowania wykonywane cyklicznie: analiza ryzyka, konstrukcja, atestowanie, planowanie.



FAZA STRATEGICZNA



- rozmowy z klientami (przedstawicielami)
- cele przedsięwzięcia z pkt. widzenia klienta
- zakres i kontekst przedsięwzięcia
-ogólne określenie wymagań – wykonanie wstępnej analizy i projektu systemu
-propozycja kilku sposobów realizacji systemu
- szacowanie kosztów oprogramowania
-analiza rozwiązań
-prezentacja wyników analizy klientowi – korekta
-budowa wstępnego harmonogramu
-określenie standardów wg których będzie realizowane przedsięwzięcie