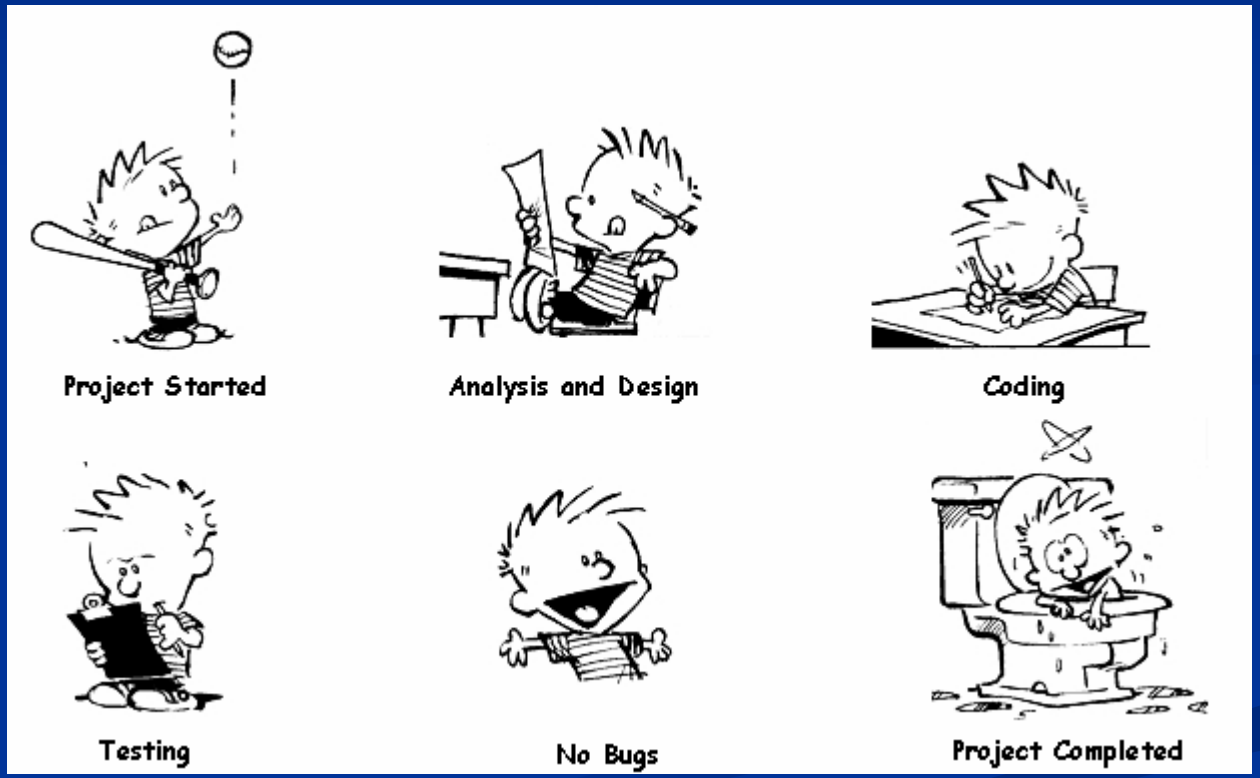
INŻYNIERIA OPROGRAMOWANIA

Proces tworzenia oprogramowania



http://www.projectportfolio.pl/fun/Cykl%20zycia%20projektu.jpg

Wykorzystane materiały:

- prezentacje J.E. Sienkiewicza
- I. Sommerville, *Inżynieria oprogramowania*, WNT 2003

Czym jest proces tworzenia oprogramowania?

- Zbiorem czynności i związanych z nimi wyników, które prowadzą do powstania produktu programowego
- Może obejmować tworzenie oprogramowania od podstaw, jednak coraz częściej nowe oprogramowanie powstaje przez rozszerzanie i modyfikowanie istniejących systemów bądź powtórne wykorzystanie istniejących komponentów
- Automatyzacja procesu tworzenia oprogramowania jest do tej pory niewielka

Zasadnicze czynności w procesie tworzenia oprogramowania

- Specyfikowanie. Funkcjonalność oprogramowania i ograniczenia jego działania muszą być dobrze zdefiniowane.
- Projektowanie i implementowanie. Stworzenie oprogramowania spełniającego specyfikację.
- Testowanie i zatwierdzanie. Wykazanie, że wytworzone oprogramowanie spełnia specyfikację i oczekiwania klienta.
- Ewolucja. Oprogramowanie ewoluuje, aby spełniać zmieniające się potrzeby użytkowników.

Specyfikowanie

- Zasadniczym etapem tworzenia specyfikacji jest tzw. inżynieria wymagań:
  - proces szukania, analizowania, dokumentowania, sprawdzania i zatwierdzania usług i ograniczeń projektowanego systemu
  - innymi słowy – proces ustalenia, co system powinien robić, przy uwzględnieniu istniejących ograniczeń
  - opisy usług i ograniczeń są wymaganiami stawianymi systemowi
- W wyniku przeprowadzonych analiz dostajemy kompletną Specyfikację Wymagań Systemowych

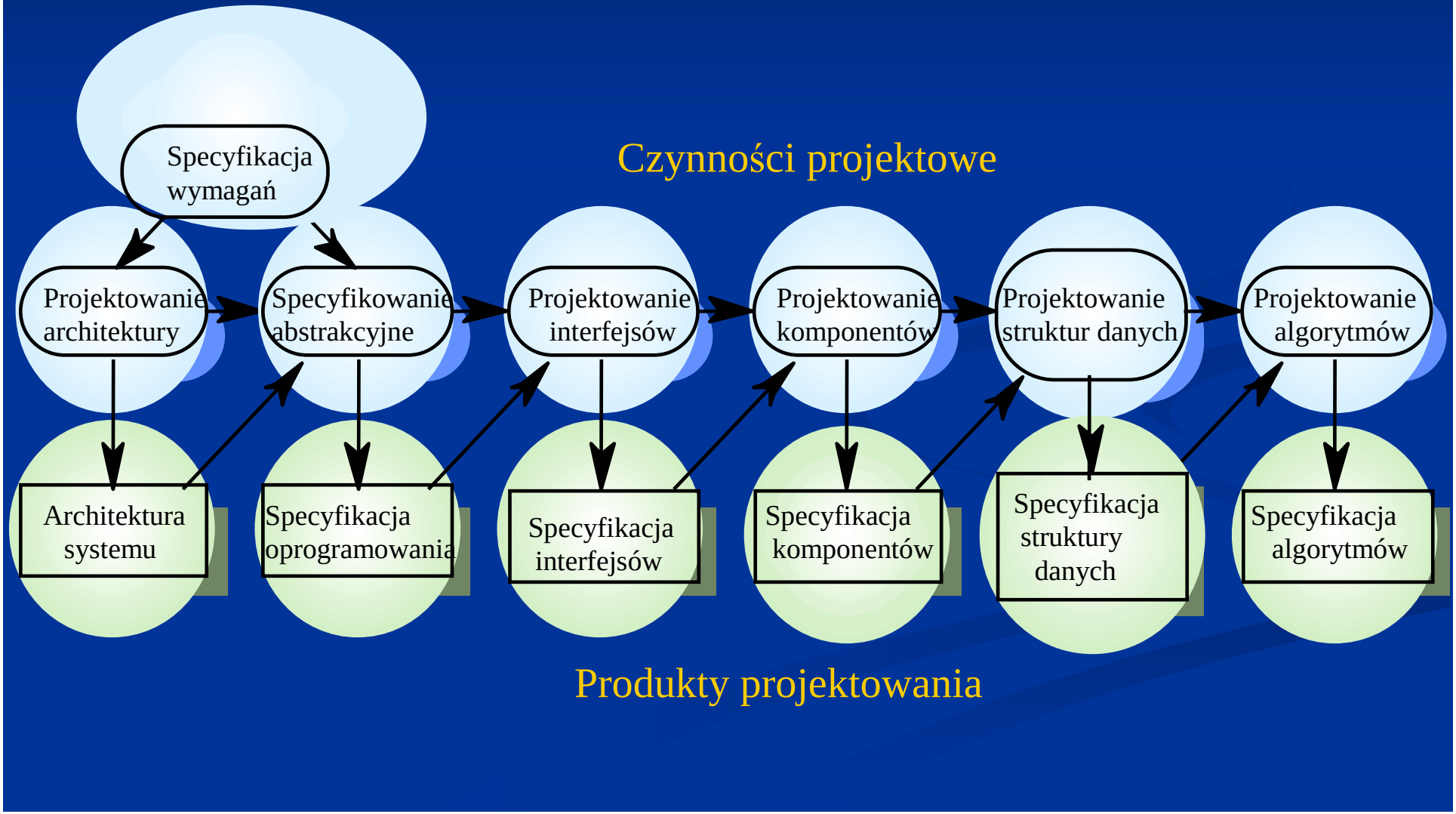
Inżynierii wymagań poświęcone będą osobne wykłady

Projektowanie

- Projekt oprogramowania to opis struktury oprogramowania, które ma być zaimplementowane; danych, które są częścią systemu; interfejsów między komponentami systemu i użytych algorytmów.
- Proces projektowania może obejmować opracowanie kilku modeli systemu na różnych poziomach abstrakcji.

- Czynności:
  - Projektowanie architektury
  - Specyfikowanie abstrakcyjne
  - Projektowanie interfejsów
  - Projektowanie komponentów
  - Projektowanie struktur danych
  - Projektowanie algorytmów

Ogólny model procesu projektowania



Metody projektowania

- Do chwili obecnej projektowanie jest często działaniem *ad hoc*, bez formalnej kontroli zmian ani zarządzania projektowaniem.
- Lepsze podejście polega na użyciu metod strukturalnych, które są zbiorami notacji i porad dla projektantów oprogramowania. Ich użycie polega zwykle na opracowaniu graficznych modeli systemu i prowadzi do dużej ilości dokumentacji projektowej.
- Metody strukturalne mogą obejmować:
  - modele przepływu danych,
  - modele encja-związek,
  - modele strukturalne,
  - modele obiektowe.

Metodom projektowania poświęcone będą osobne wykłady

Implementowanie

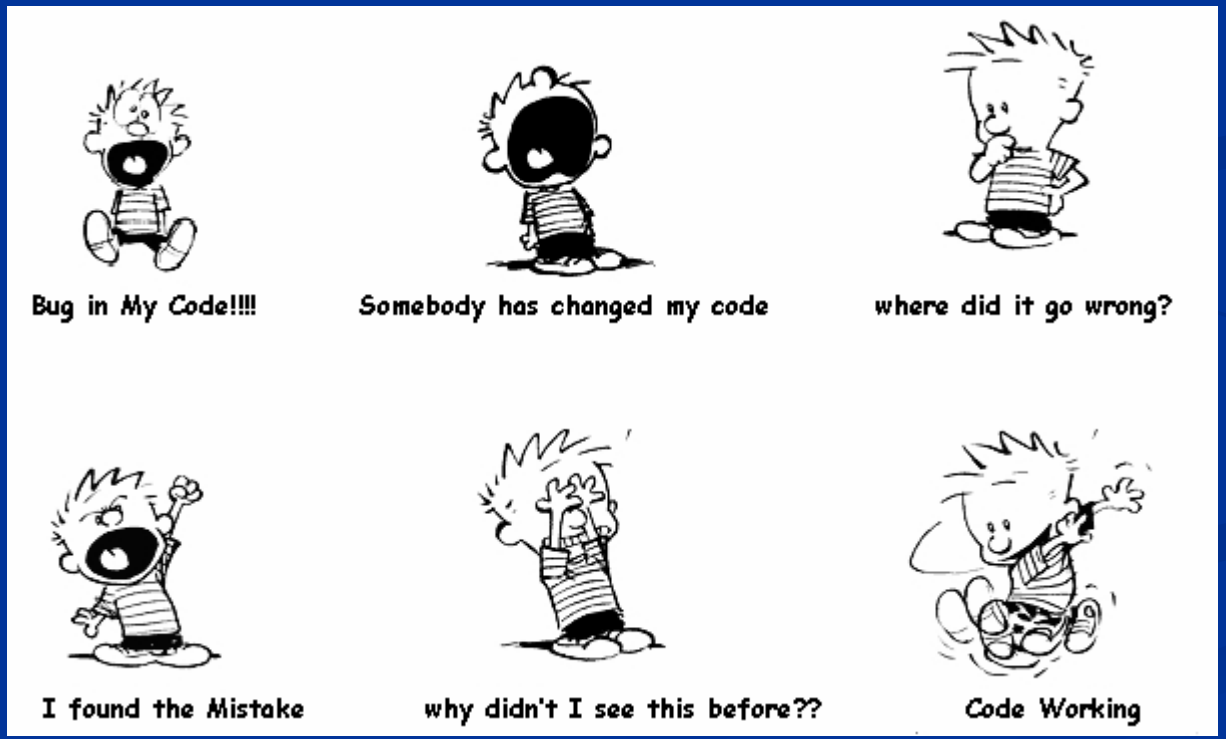
- Faza implementowania w tworzeniu oprogramowania to proces przekształcania specyfikacji systemu w działający system, na postawie projektu.

Zatwierdzanie oprogramowania

- Weryfikacja i zatwierdzenie mają wykazać, że system jest zgodny ze swoją specyfikacją i spełnia oczekiwania klienta.
- Obejmuje proces sprawdzania, m.in. kontrole, inspekcje i recenzje w każdym kroku procesu tworzenia, od definicji wymagań użytkownika aż do implementacji.
- Z wyjątkiem małych programów, nie należy testować systemu jako pojedynczej całości – najpierw należy wykonywać testy jednostkowe, przechodząc do coraz większych struktur.

Testowanie

- Programiści wykonują pewne testy kodu, który napisali. Zwykle prowadzi to do wykrycia błędów, które należy usunąć z programu.
- Testowanie programu i usuwanie błędów to dwa różne procesy.
- Znalezienie lokalizacji błędu może wymagać nowych przypadków testowych.



Fazy procesu testowania

- Testowanie poszczególnych jednostek (klas, komponentów)
  - Testuje się poszczególne jednostki, aby zapewnić, że działają poprawnie.
- Testowanie modułów i podsystemów
  - Moduł jest kolekcją niezależnych jednostek takich jak klasy obiektów, abstrakcyjne typy danych, albo bardziej luźną kolekcją procedur i funkcji.
  - Kolekcję modułów można z kolei zintegrować w podsystemie (większej całości).
- Testowanie systemu
  - Proces ma wykryć błędy wynikające z nieprzewidzianych interakcji między podsystemami zintegrowanymi w pełen system oraz problemy z interfejsami podsystemów.
- Testowanie odbiorcze (akceptacyjne)
  - Jest to końcowa faza procesu testowania przed przyjęciem systemu do użytkowania.

Testowaniu poświęcone będą osobne wykłady

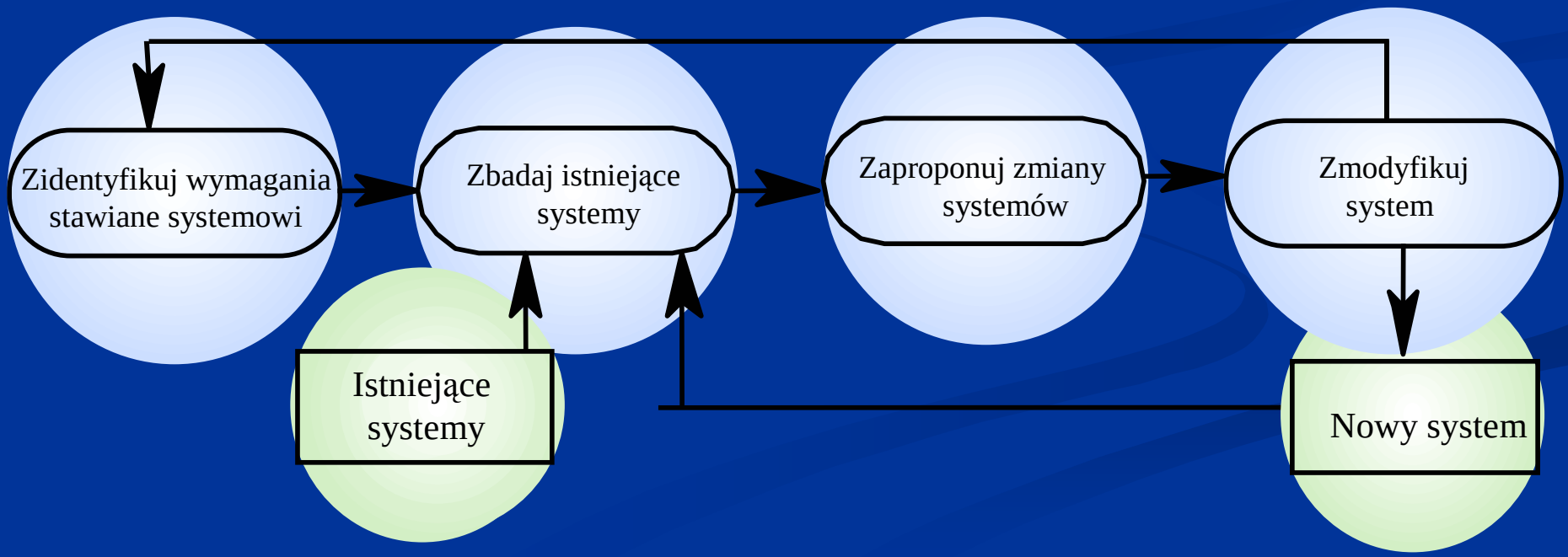
Testowanie alfa i beta

- Testowanie alfa jest testowaniem odbiorczym, stosowanym kiedy system jest tworzony dla konkretnego klienta. Proces testowania trwa do momentu osiągnięcia zgody pomiędzy wytwórcą systemu i klientem co do tego, że dostarczony system jest możliwą do przyjęcia implementacją wymagań.

- Testowanie beta stosuje się, kiedy system sprzedawany jako produkt programowy. Testowanie polega na dostarczeniu systemu pewnej liczbie potencjalnych klientów, którzy zgodzili się z niego korzystać. Klienci informują wytwórców o pojawiających się problemach.

Ewolucja oprogramowania

- Elastyczność systemów oprogramowania jest jedną z głównych przyczyn, że coraz więcej i więcej oprogramowania włącza się do wielkich, złożonych systemów.
- W przypadku oprogramowania, w przeciwieństwie do sprzętu, zmiany mogą być wprowadzone w każdej chwili tworzenia systemu, lub nawet po jego zakończeniu.
- Koszt tych zmian może być bardzo wysoki, ale wciąż będzie niższy niż odpowiednie zmiany w sprzęcie systemu.



Praca, błędy i koszty w tworzeniu systemu

- Rozkład pracy w cyklu tworzenia systemu:
  - Specyfikacja 6%
  - Projektowanie 5%
  - Kodowanie 7%
  - Testowanie 15%
  - Eksploatacja / ewolucja 67%
- Źródła błędów:
  - Specyfikacja 56%
  - Projektowanie 27%
  - Kodowanie 7%
  - Inne 10%
- Koszty poprawienia błędów:
  - Specyfikacja 82%
  - Projektowanie 13%
  - Kodowanie 1%
  - Inne 4%

na podstawie: konspekt ISPI – R. Tadeusiewicz

Podstawowe modele procesu tworzenia oprogramowania

- Model kaskadowy (wodospad, ang. waterfall)

  Podstawowe czynności specyfikowania, tworzenia, zatwierdzania i ewolucji są odrębnymi fazami procesu.
- Model przyrostowy (ang. incremental)

  Specyfikowanie, projektowanie, implementowanie i zatwierdzanie przeplatają się.
- Model ewolucyjny (ang. evolutionary)

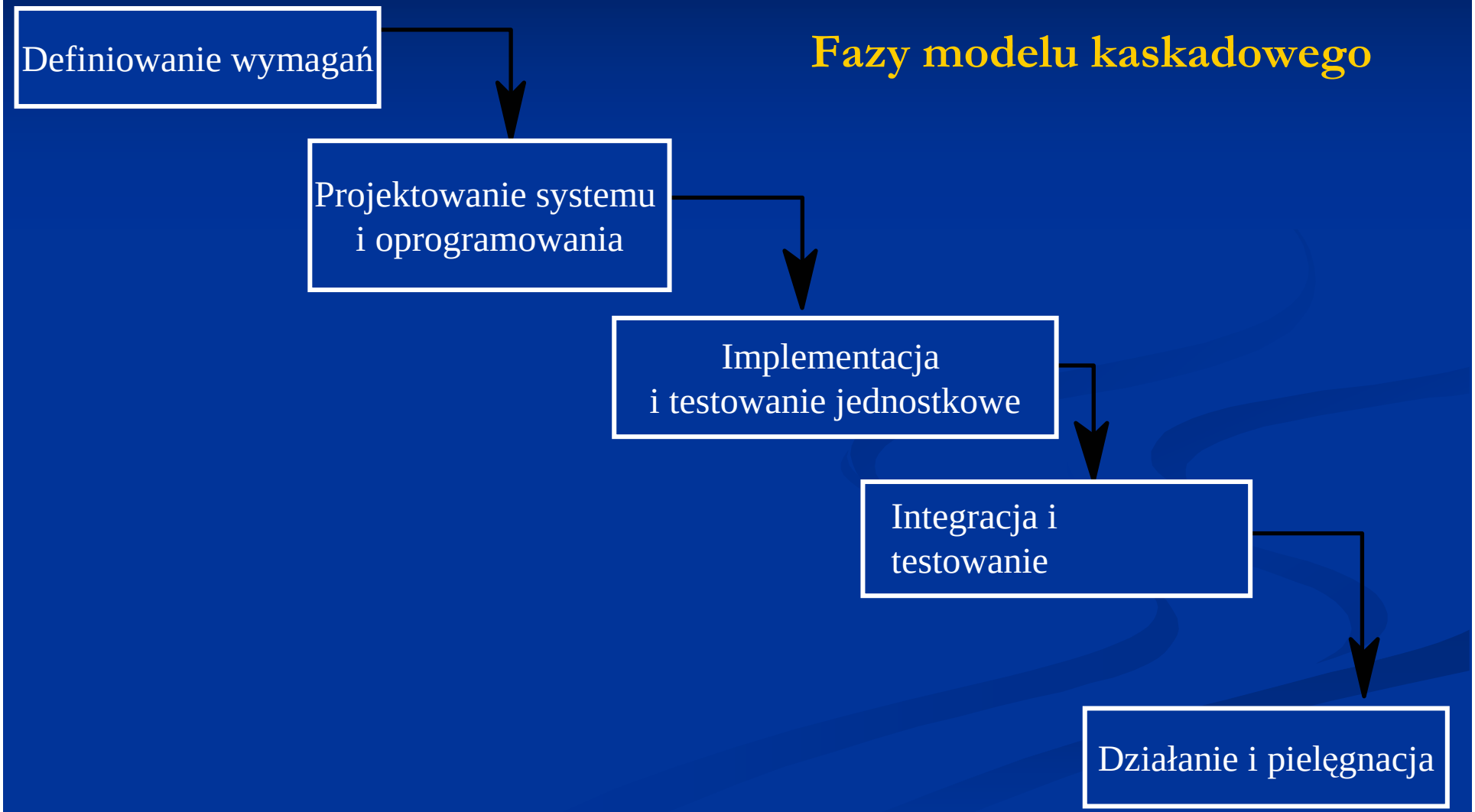
  Tworzy się prototypy w celu przedyskutowania z klientem oraz uzyskania akceptacji.
- Model spiralny (ang. spiral)

  Proces tworzenia ma postać spirali, której każda pętla reprezentuje jedną fazę procesu.
- Tworzenie formalne systemu (anf. formal)

  Oparte na budowaniu formalnych matematycznych specyfikacji systemu i przekształcaniu tych specyfikacji w program za pomocą metod matematycznych.
- Tworzenie z użyciem wielokrotnym

  W tym podejściu zakłada się istnienie dużej liczby komponentów zdatnych do ponownego użycia.

Model kaskadowy



Opis poszczególnych faz modelu kaskadowego

- Definiowanie wymagań

  Zebranie i zdokumentowanie wymagań na system. Powstaje specyfikacja systemu.

- Projekt systemu i oprogramowania

  Zdefiniowanie architektury systemu (elementów sprzętowych i programowych). Zdefiniowanie interfejsów i interakcji między poszczególnymi elementami. Powstaje projekt systemu.

- Implementacja i testowanie jednostkowe

  Kodowanie poszczególnych jednostek programowych (np. klas) oraz ich testowanie.

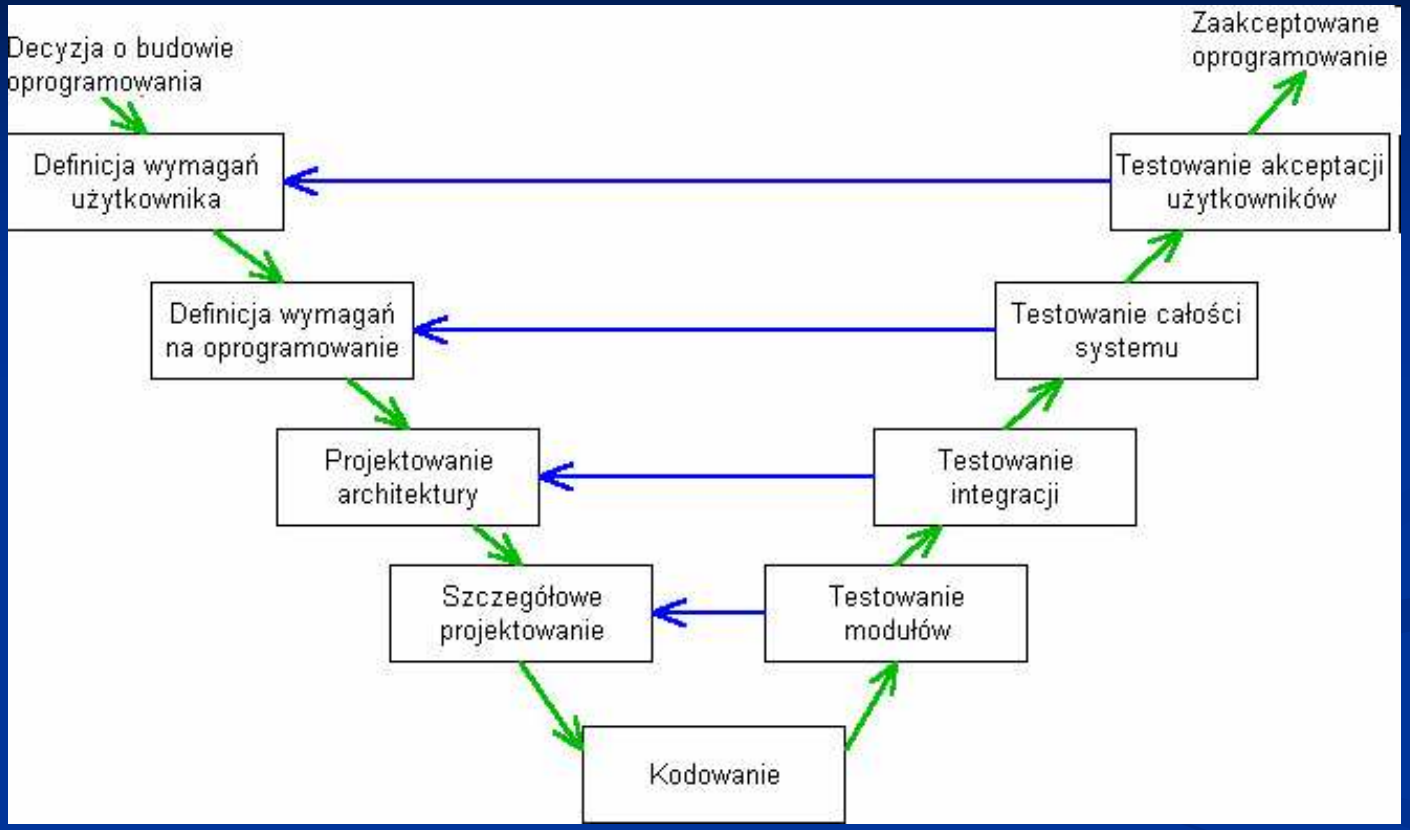
- Integracja i testowanie

  Zintegrowanie jednostek programowych do kompletnego systemu. Sprawdzenie, czy kompletny system spełnia specyfikację.

- Działanie i pielęgnacja

  Przekazanie systemu do użytkowania. Modyfikacja systemu zgodnie ze zmianą wymagań.

Wariant modelu kaskadowego – model V



Po sprawdzeniu po prawej stronie, jeżeli wynik danego testu jest niepomyślny, to po następuje odesłanie problemu z powrotem na lewą stronę. Tam problem zostaje naprawiony i jeszcze raz podlega sprawdzeniu.

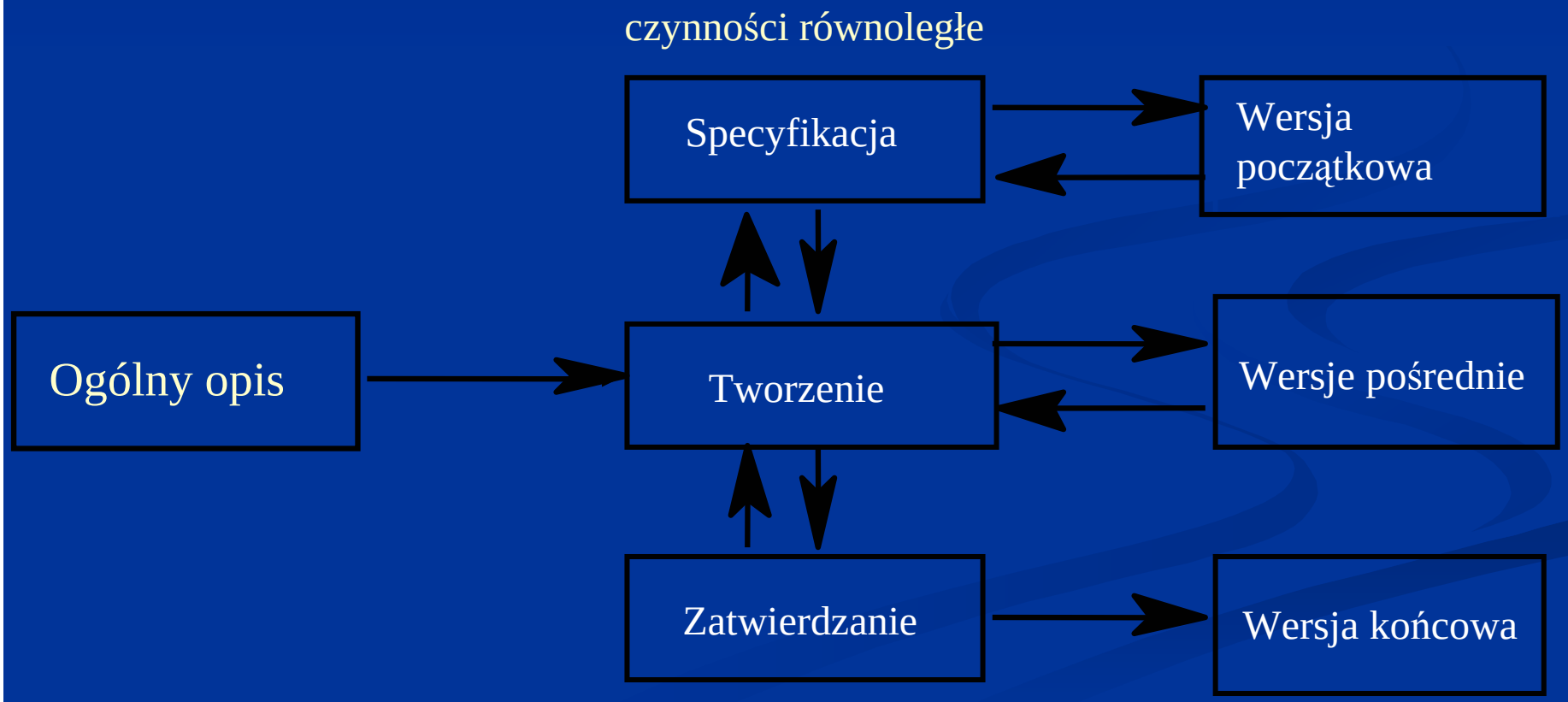
źródło: konspekt ISPI – R. Tadeusiewicz

Wady modelu kaskadowego

- Powodzenie projektu zależy głownie od poprawnej i kompletnej specyfikacji systemu – model kaskadowy powinien być używany jedynie wówczas, gdy wymagania są jasne i zrozumiałe.
- Nieelastyczny podział na rozłączne etapy – następnej fazy nie powinno się rozpoczynać, jeśli poprzednia się nie zakończy.
- Ewentualne iteracje są kosztowne oraz wymagają powtarzania wielu prac – koszty opracowania i akceptacji dokumentów są wysokie.
- Ostatni etap może się rozrosnąć w sposób niekontrolowany.

Model przyrostowy

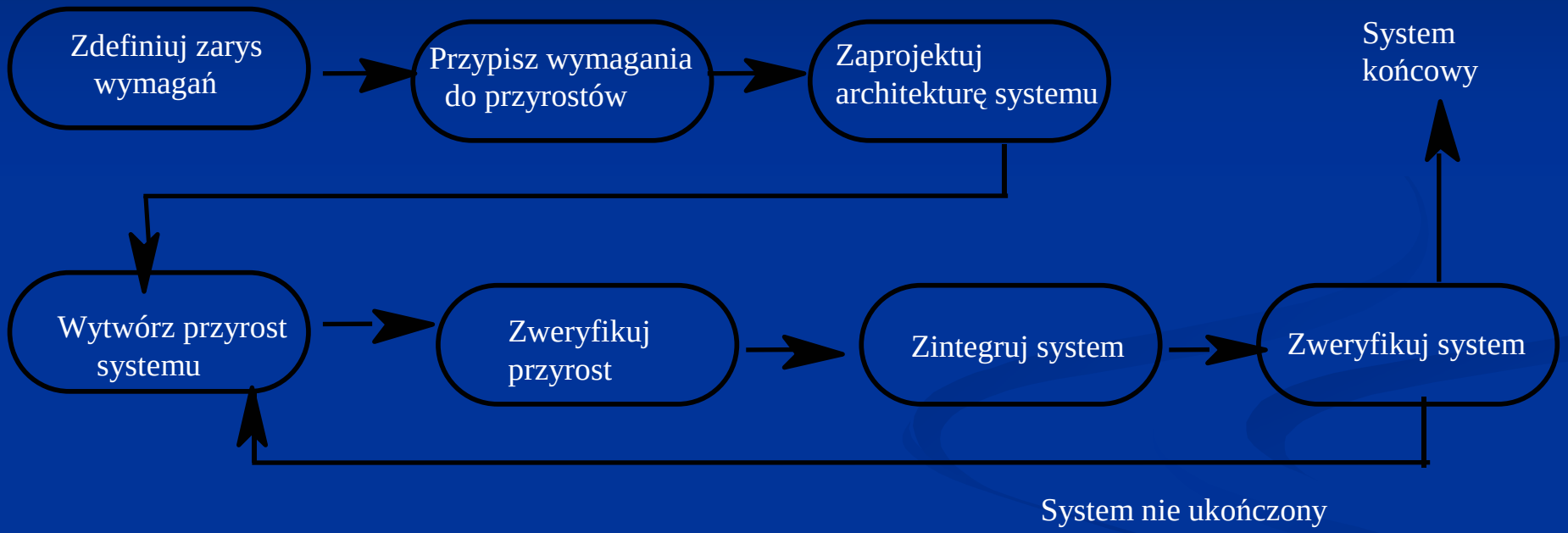
Zasada: opracowanie wstępnej implementacji, pokazanie jej użytkownikowi z prośbą o komentarze i udoskonalanie jej w wielu wersjach aż do powstania odpowiedniego systemu.



Tworzenie badawcze w modelach przyrostowych

- Praca z klientem, polegająca na badaniu wymagań. Tworzenie rozpoczyna się od tych części systemu, które są dobrze rozpoznane. System ewoluuje przez dodawanie nowych cech, które proponuje klient, aż do ostatecznej wersji systemu. Pojawiają się iteracje, polegające na powtarzaniu pewnych etapów tworzenia systemu.
- Podejście przyrostowe zaproponowano jako sposób na ograniczenie powtarzania prac w procesie tworzenia oraz danie klientom pewnych możliwości odkładania decyzji o szczegółowych wymaganiach do czasu, aż zdobędą pewne doświadczenia w pracy z systemem.
- Klienci identyfikują w zarysie usługi, które system ma oferować. Wskazują, które z nich są dla nich najważniejsze, a które najmniej ważne. Definiuje się następnie pewną liczbę przyrostów, które maja być dostarczone.
- Gdy przyrost jest już gotowy i dostarczony, klienci mogą go uruchomić. Oznacza to, że szybko otrzymują część funkcjonalności systemu

Schemat tworzenia przyrostowego



Warianty modelu przyrostowego

- Programowanie zwinne (ang. Agile software development)
  - Ludzie i interakcje pomiędzy nimi – zamiast procesów i narzędzi
  - Działające oprogramowanie – zamiast szczegółowej dokumentacji
  - Współpraca z klientem – zamiast negocjowania kontraktów
  - Reagowanie na zmiany – zamiast realizowania planu

- Programowanie ekstremalne (ang. eXtreme Programming, XP)
  - określenie metafory tworzonego systemu
  - gra planistyczna
  - prosty projekt systemu
  - programowanie sterowane testami (Test Driven Development, TDD)
  - ciągła integracja, częste wydania systemu
  - udział klienta w zespole
  - programowanie w parach

Programowaniu zwinnemu i ekstremalnemu poświęcone będą osobne wykłady

Zalety modelu przyrostowego

- Klienci nie muszą czekać na dostarczenie całego systemu, zanim zaczną czerpać z niego korzyść.
- Klienci mogą używać wstępnych przyrostów jako rodzaju prototypu i zdobywać doświadczenia, które inspirują wymagania wobec późniejszych przyrostów.
- Ryzyko całkowitej porażki przedsięwzięcia jest mniejsze.
- Usługi o najwyższym priorytecie będą dostarczane jako pierwsze.

Bardzo dobrze sprawdza się
w przypadku systemów małych lub średnich
oraz tych z krótkim czasem życia.

Wady modelu przyrostowego

- Proces nie jest widoczny
- System ma często złą strukturę
- Brak dokładnej specyfikacji
- Dokumentacja czasami nie jest aktualna
- Konieczne mogą być specjalne narzędzia i techniki
- Konieczna stała dostępność przedstawiciela klienta.

Nie sprawdza się w przypadku dużych systemów o długim czasie życia

Model ewolucyjny

- W tym modelu, podczas projektowania tworzy się prototyp w celu jego przedyskutowania z klientem oraz uzyskania jego akceptacji. Po akceptacji prototypu przechodzi się do kolejnych etapów tworzenia oprogramowania.
- Prototypowanie zapobiega błędnemu zrozumieniu wymagań systemu, które może powodować wzrost kosztów – pozwala programistom lepiej zrozumieć potrzeby klienta.
- Pozwala klientowi zobaczyć jak mniej więcej system będzie wyglądał (można np. rozpocząć wcześnie szkolenia).
- Prototyp jest zwykle porzucany (nie jest dalej rozwijany) – jest to tzw. prototypowanie z porzuceniem. W przeciwnym przypadku mamy do czynienia z tworzeniem badawczym.
- Prototypowanie z porzuceniem ma na celu eksperymentowanie z tymi wymaganiami użytkownika które są niejasne, aby lepiej zrozumieć wymagania klienta i wypracować lepszą definicję wymagań stawianych systemowi.
- Główne wady: konieczne specjalnie narzędzia; prototyp kosztuje.

Model spiralny

- Zaproponowany w 1998 r.
- Proces nie jest przedstawiany jako ciąg czynności z pewnymi nawrotami między nimi, ale ma postać spirali.
- Każda pętla spirali reprezentuje jedną fazę procesu.
- Najbardziej wewnętrzna pętla może być poświęcona wykonalności systemu, następna definicji wymagań stawianych systemowi, kolejna projektowaniu itd.
- Jawne potraktowanie zagrożeń.

Model spiralny – schemat

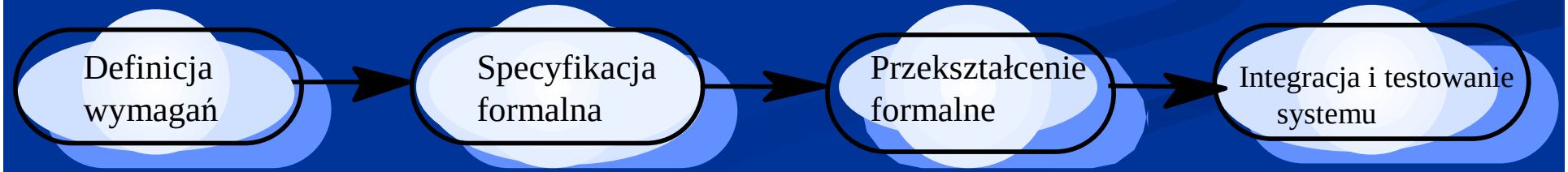
INŻYNIERIA OPROGRAMOWANIA

Sektory spirali w modelu spiralnym

- Ustalanie celów
  - Definiuje się konkretne cele tej fazy przedsięwzięcia. Identyfikuje się ograniczenia, którym podlega proces i produkt.
- Rozpoznanie i redukcja zagrożeń
  - Przeprowadza się szczegółową analizę każdego z rozpoznanych zagrożeń przedsięwzięcia.
- Tworzenie i zatwierdzanie
  - Po ocenie zagrożeń wybiera się model tworzenia systemu.
- Planowanie
  - Recenzuje się przedsięwzięcie i podejmuje decyzję, czy rozpoczynać następną pętlę spirali.

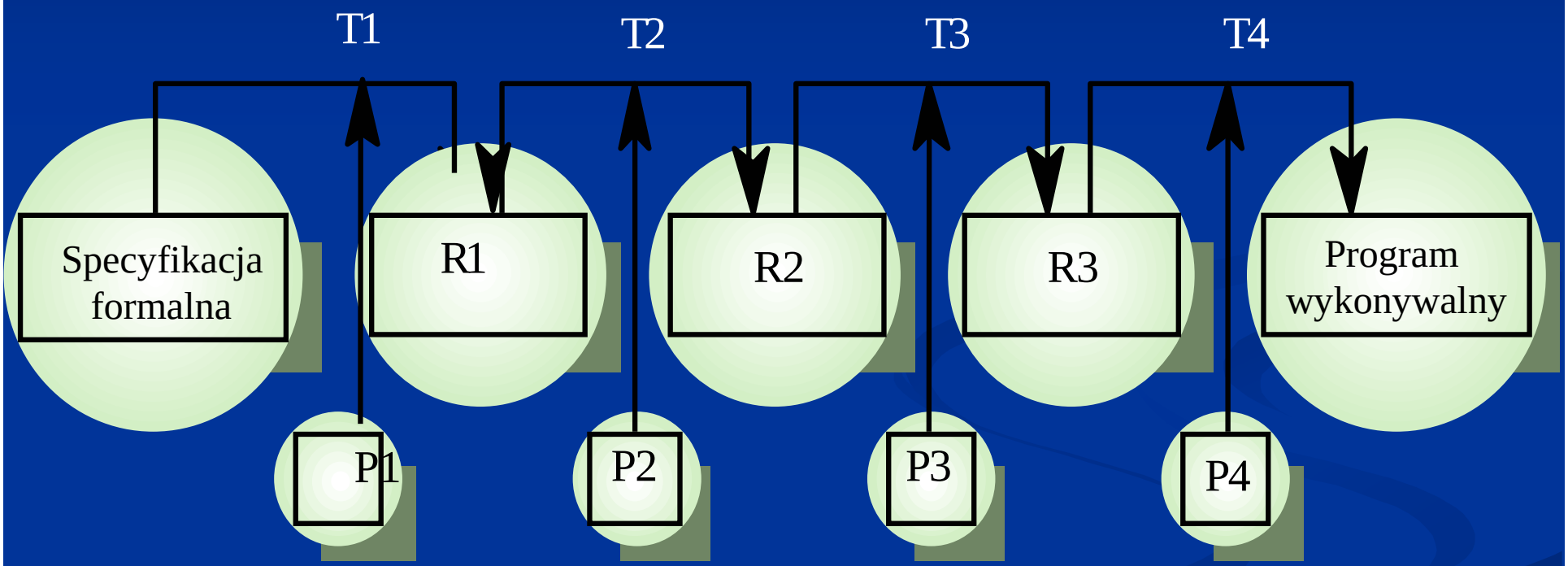
Tworzenie formalne systemów

- Proces tworzenia jest oparty na matematycznych przekształceniach specyfikacji systemu w program wykonywalny.
- Podejście to ma wiele wspólnego z modelem kaskadowym.
- W procesie przekształcania, formalna matematyczna reprezentacja systemu jest metodycznie przekształcana w bardziej szczegółowe, ale wciąż matematycznie poprawne reprezentacje systemu.



Przekształcenia formalne

INŻYNIERIA OPROGRAMOWANIA



Tworzenie formalne – przykład

INŻYNIERIA OPROGRAMOWANIA

- Zapisujemy wymagania w wybranej notacji, np. Backus–Naura (BNF):

```
<postal-address> ::= <name-part> <street-address> <zip-part>
<name-part> ::= <personal-part> <last-name> <opt-jr-part> <EOL> |
   <personal-part>
<name-part> <personal-part> ::= <first-name> | <initial> ".„
<street-address> ::= <house-num> <street-name> <opt-apt-num> <EOL>
<zip-part> ::= <town-name> "," <state-code> <ZIP-code> <EOL>
<opt-jr-part> ::= "Sr." | "Jr." | <roman-numeral> | "„
```

- Przekształcamy formalnie zapisane wymagania do postaci np. gotowego formularza, kodu C++ czy zapytań SQL ręcznie, bądź przy pomocy specjalistycznego oprogramowania.
- Generujemy parser (np. programem YACC), który sprawdzi utworzony kod pod względem jego zgodności z wyspecyfikowanymi wymaganiami.

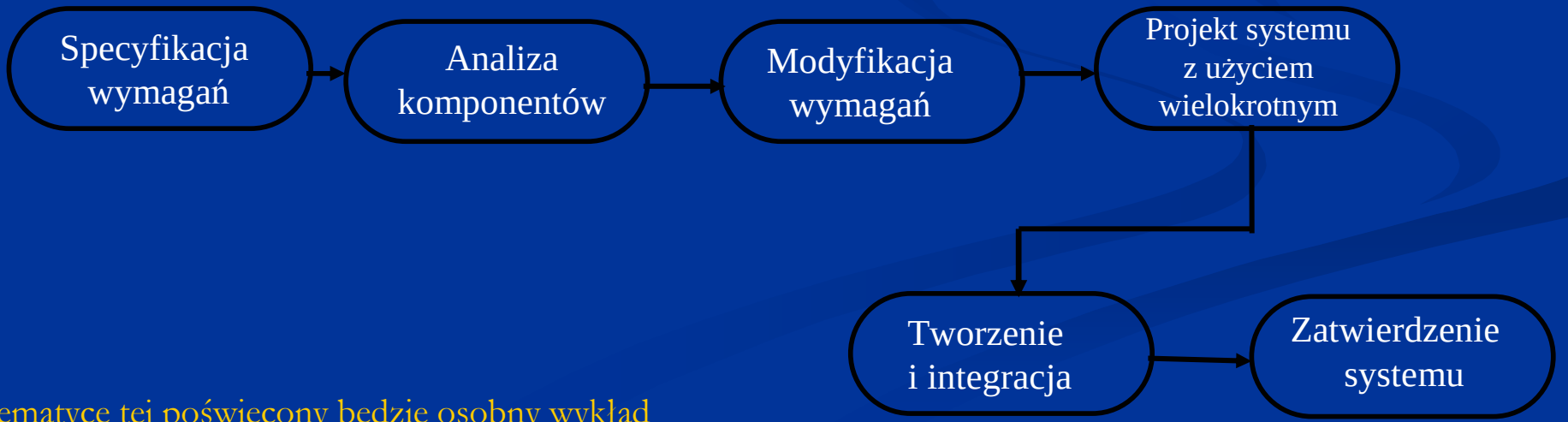
Tworzenie formalne – zastosowania i problemy

- Oprócz specjalistycznych dziedzin procesy oparte na przekształceniach formalnych są używane rzadko.
- Najlepiej znanym przykładem takiego formalnego procesu tworzenia jest Cleanroom, pierwotnie opracowany przez IBM. Cleanroom jest oparty na przyrostowym tworzeniu oprogramowania, gdy formalnie wykonuje się każdy krok i dowodzi jego poprawności.
- Wymagają specjalistycznej wiedzy i w praktyce okazuje się, że w wypadku większości systemów nie powodują zmniejszenia kosztów lub polepszenia jakości w porównaniu z innymi podejściami.
- Interakcje systemów nie poddają się łatwo specyfikowaniu formalnemu.

Więcej informacji: http://www.fmeurope.org

Tworzenie z użyciem wielokrotnym

- Zakłada się istnienie dużego zbioru dostępnych komponentów programowych do użycia wielokrotnego oraz integrującej je struktury
- Etapy procesu:
  - analiza komponentów
  - modyfikacja wymagań
  - projektowanie systemu z użyciem wielokrotnym
  - tworzenie i integracja



Tematyce tej poświęcony będzie osobny wykład

Zautomatyzowane wspomaganie procesu tworzenia oprogramowania

- Komputerowo wspomagana inżynieria oprogramowania (CASE) korzysta z różnego oprogramowania do wspomagania czynności procesu tworzenia oprogramowania, takich jak inżynieria wymagań, projektowanie, programowanie i testowanie.
- Czynności, które można zautomatyzować za pomocą CASE:
  - oprogramowanie do tworzenia graficznych modeli systemu jako części specyfikacji wymagań i projektu oprogramowania
  - czytanie projektu za pomocą słownika danych, który przechowuje informacje o encjach i związkach w projekcie
  - generowanie graficznego interfejsu użytkownika na podstawie opisu interfejsu, opracowanego interaktywnie przez użytkownika
  - śledzenie błędów przez udostępnienie informacji o wykonującym się programie (debugging)
  - automatyczne tłumaczenie programów ze starych wersji języków programowania

Technologia CASE

- Technologia CASE jest obecnie dostępna dla większości rutynowych czynności procesu tworzenia oprogramowania.
- Inżynieria oprogramowania jest czynnością projektową opartą głównie na kreatywnym myśleniu. Istniejące systemy CASE automatyzują pewne rutynowe czynności, ale próby zastosowania sztucznej inteligencji do wspomagania programowania nie powiodły się dotychczas.
- W większości firm procesy związane z inżynierią oprogramowania są czynnością zespołową. Inżynierowie spędzają więc sporo czasu na interakcji z innymi członkami zespołu. Technologia CASE nie daje tu zbyt dużego wsparcia.

Klasyfikacja narzędzi CASE

- Narzędzia do planowania
  - Narzędzia do szacowania kosztów, arkusze kalkulacyjne
- Narzędzia do zarządzania zmianami
  - Narzędzia do zapisu i śledzenia wymagań, systemy kontroli zmian, systemy zarządzania wersjami
- Narzędzia do projektowania
  - Edytory i procesory tekstów, edytory diagramów
- Narzędzia do zarządzania konfiguracjami
  - Narzędzia do budowania systemów, systemy zarządzania wersjami
- Narzędzia do prototypowania
  - Języki bardzo wysokiego poziomu, generatory interfejsu użytkownika
- Narzędzia wspomagające
  - Edytory projektów, słowniki danych

Project Completed

Time for a Party

PARTY !!!