Inżynieria oprogramowania II

Wykład 8:

"UPEDU: Implementacja
(ang. *Implementation discipline*)"

Marek Krętowski
e-mail: mkret@wi.pb.edu.pl
http://aragorn.pb.bialystok.pl/~mkret

Na podstawie podręcznika: „Software Engineering Process with the UPEDU" P. Robillard, P. Kruchten, P. d'Astous, Addison-Wesley, 2003

Wprowadzenie

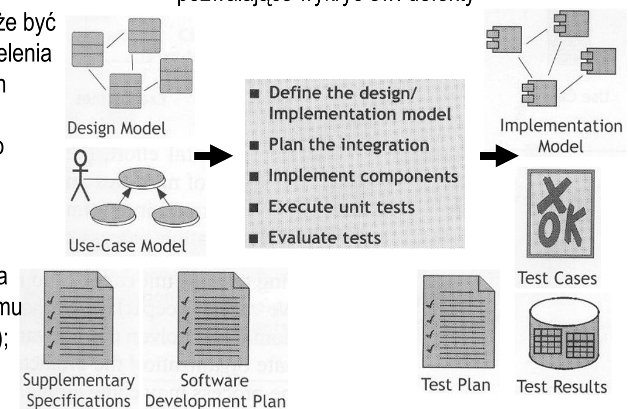
- Implementacja - szczegółowe rozwiązania projektowe są transfomowane w sekwencje wyrażeń (kod) języka(-ów) programowania
- Oznacza zakończenie procesu "krystalizacji informacji" w sposób zachowujący ciągłość czynności z innych dyscyplin
- Podstawowe cele:
  - organizacja systemu w ramach struktury komponentów
  - implementacja klas i obiektów (kod i programy wykonywalne)
  - walidacja jakości implementacji poprzez testowanie pojedynczych komponentów
  - integracja komponentów w działający system (zwany często w języku ang. *build*)
- *Build* - sprawna (funkcjonująca) wersja pełnego systemu lub jego części; odpowiada podzbiorowi funkcjonalności dostarczanemu w finalnym produkcie, który jest wynikiem wszystkich iteracji

Wprowadzenie (2)

- Podstawowe role:
  - integrator - odpowiedzialny jest za zaplanowanie i zrealizowanie połączenia w jedną całość (integrację) poszczególnych komponentów tworząc działającą wersję systemu (*build*); w przypadku niewielkich systemów, z małą liczbą komponentów, rola integratora może być praktycznie wyeliminowana
  - programista (ang. implementer) - odpowiedzialny jest za wytworzenie kodu, który prowadzi do wykonywalnych komponentów
  - recenzent (ang. reviewer) - odpowiada za przeglądy i inspekcje kodu źródłowego; kluczowe rola z punktu widzenia jakości kodu oraz podporządkowania się przyjętym wskazówkom programistycznym (ang. *guidelines*)
- Końcowym produktem implementacji są programy (komponenty) wykonywalne, które wyprowadzone (wywodzą się) z pakietów i ich komponentów, które z kolei bazują na klasach i diagramach klas, które z kolei powstają na podstawie diagramów przypadków użycia (i innych diagramów jak stanów czy współpracy)

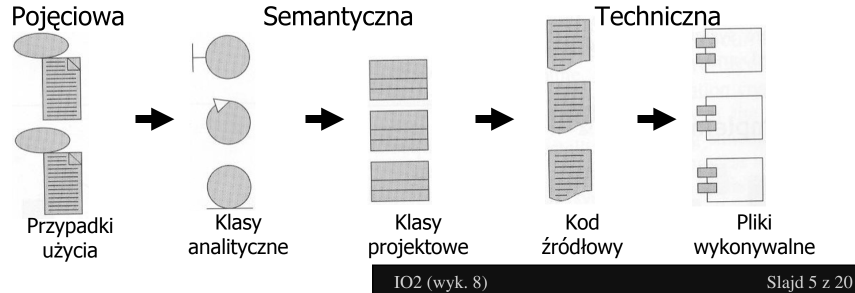
Zadania związane z implementacją

- W **dużych i skomplikowanych systemach** implementacja musi być zaplanowana, zorganizowana i dokonana jej walidacja
- Model implementacji może być wykorzystany do przydzielenia zadań programistycznych osobom oraz do oceny zasobów niezbędnych do wygenerowania kodu
- Plan integracji określa kolejność powstawania komponentów oraz ustala kolejność łączenia systemu w całość (kolejne *build*-y); zwykle integracja nie stanowi problemu
- Na podstawie klas projektowych tworzone są poszczególne komponenty oraz przeprowadzane są testy jednostkowe pozwalające wykryć ew. defekty

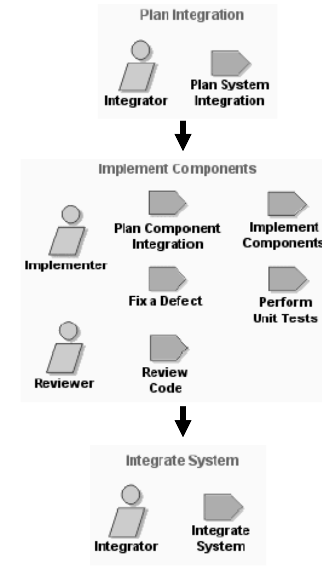


Strukturalizacja wiedzy

- Implementacja systemu wymaga przetworzenia dużych ilości informacji pochodzących z różnych dziedzin wiedzy:
  - zawarta w przypadkach użycia jest pojęciowa i specyficzna dla aplikacji
  - zawarta w klasach jest semantyczna i związana z algorytmami i strukturą modelu
  - zawarta w kodzie jest syntaktyczna i związana z konkretnym j. programowania
- Narzuca to konieczność odpowiedniej organizacji procesu (zwłaszcza przy pracy zespołowej), w przeciwnym razie narażamy się na niepowodzenie



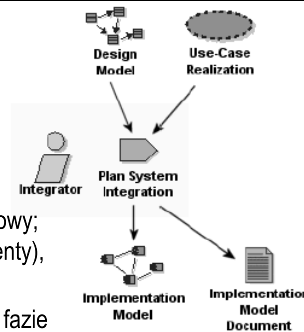
Czynności procesu



- Planowanie integracji (integrator):
  - Planowanie integracji systemu (ang. Plan System Integration)
- Implementacja komponentów (programista, recenzent):
  - Planowanie integracji komponentów (ang. Plan Component Integration)
  - Implementacja komponentów (ang. Implement Components)
  - Naprawa defektów (ang. Fix a Defect)
  - Przeprowadzenie testów jednostkowych (ang. Perform Unit Tests)
  - Przeglądanie kodu (ang. Review Code)
- Przeprowadzenie integracji systemu (integrator):
  - Integracja systemu (ang. Integrate System)

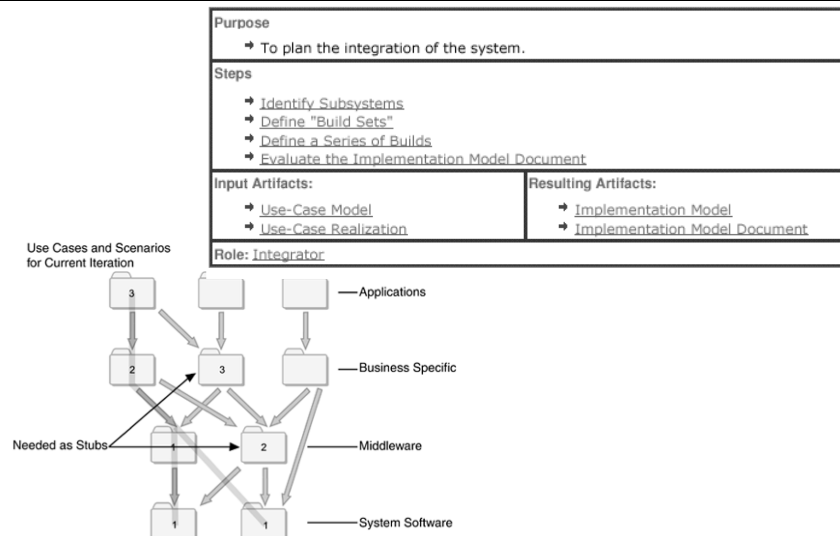
Planowanie integracji

- Organizując zadania integrator powinien brać pod uwagę różnorakie kryteria: od złożoności poszczególnych komponentów po dostępność zasobów
- Plan integracji wskazuje programistom kolejność zadań oraz przedstawia jak komponenty modelu implementacyjnego będą tworzone
- Najczęściej integracja jest wykonywana w sposób przyrostowy; kod jest pisany i testowany w niedużych partiach (komponenty), które następnie są kolejno łączone z całością
- Oznacza to, że integracja odbywa się nie tylko w końcowej fazie implementacji, ale raczej jest rozłożona w czasie; dzięki przyrostowemu łączeniu błędy związane z komponentami mogą być wykrywane wcześnie
- Powodzenie planowania i realizacja integracji zależy w dużej mierze od doświadczenia integratora i umiejętności kojarzenia rozwiązań projektowych z aktualnymi (lub wcześniej testowanymi) planami. Przyjmuje się, że brak niezbędnego doświadczenia ze strony integratora może prowadzić do nadmiernego wzrostu złożoności systemu



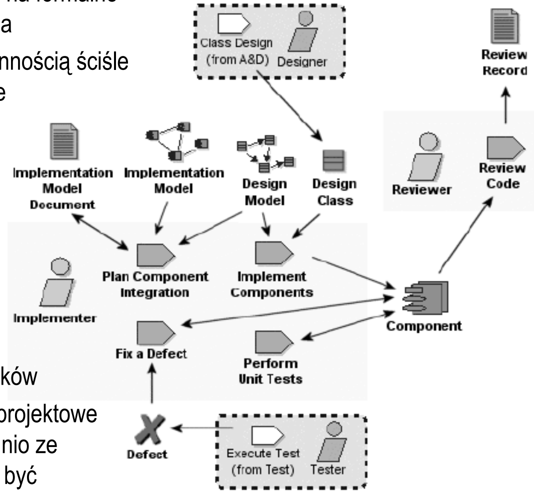
Planowanie integracji systemu
(ang. Plan System Integration)

Purpose	
➔ To plan the integration of the system.	
Steps	
➔ Identify Subsystems ➔ Define "Build Sets" ➔ Define a Series of Builds ➔ Evaluate the Implementation Model Document	
Input Artifacts: ➔ Use-Case Model ➔ Use-Case Realization	Resulting Artifacts: ➔ Implementation Model ➔ Implementation Model Document
Role: Integrator	



Implementacja komponentów

- Przełożenie rozumienia projektu na formalne wyrażenia języka programowania
- Implementacja powinna być czynnością ściśle techniczną wymagającą niewiele lub wcale inwencji twórczej
- Kreatywność powinna się wyrażać poprzez czynności projektowe, natomiast kodowanie wymaga przede wszystkim dużej fachowości i powinno być wykonywane przez wykwalifikowanych i najlepiej doświadczonych techników
- Pewne szczegółowe czynności projektowe (zwłaszcza związane bezpośrednio ze sposobem implementacji) mogą być finalizowane dopiero podczas programowania



Planowanie integracji komponentów
(ang. Plan Component Integration)

Purpose
- To plan the order in which the components contained in an implementation subsystem should be integrated.

Steps
- Define the Builds
- Identify the Classes
- Update the Subsystem's Imports

Input Artifacts:	Resulting Artifacts:
Design Model Implementation Model Implementation Model Document	Implementation Model Document

Role: Implementer

- Identyfikowane są przypadki użycia i scenariusze, które realizowane będą w ramach podsystemu (może to oznaczać tylko częściową realizację)
- Bazując na realizacjach przypadków użycia (diagramy przebiegu, współpracy, ...) identyfikowane są klasy, które należy zaimplementować w danej iteracji oraz te, które już zostały zaimplementowane; ponadto może pojawi się konieczność implementacji dodatkowych klas, nie występujących jawnie, ale niezbędnych do kompilacji

Implementacja komponentów
(ang. Implement Components)

Purpose
- To produce source code in accordance with the design model.

Steps
- Implement Operations
- Implement States
- Use Delegation to Reuse Implementation
- Implement Associations
- Implement Attributes
- Provide Feedback to Design
- Evaluate the Code

There is no strict order between the steps. Start implementing the operations, and implement associations and attributes as they are needed to be able to compile and run the operations.

Input Artifacts:	Resulting Artifacts:
Design Class Design Model	Component

Role: Implementer

- Syndrom VIP (ang. variation in practice) - pojawia się gdy poszczególni programiści w zespole stosują własne techniki nazewnictwa (skróty w nazwach zmiennych), definiowania klas, strukturalizacji danych itd.; może wpływać niekorzystnie na niezawodność a na pewno na czytelność wynikowego kodu

Naprawa defektów
(ang. Fix a Defect)

Purpose
- To fix a defect

Steps
- Stabilize the Defect
- Locate the Fault
- Fix the Fault

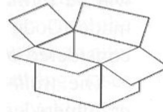
Input Artifacts:	Resulting Artifacts:
Defect	Component

Role: Implementer

- Debugowanie jest najbardziej prymitywną formą testowania
- Wykrywane defekty związane są z samą implementacją i związane są najczęściej z poziomem umiejętności osoby piszącej kod; często są efektem wadliwej struktury kodu, złego zrozumienia cech i konstrukcji języka programowania, niespójnego nazewnictwa lub błędnej pisowni wyrażeń
- Potrzeba szczegółowego debugowania zwykle oznacza niską fachowość
- Tworzenie programów wolnych od usterek (ang. bug-free) od samego początku jest najbardziej pożądanym rozwiązaniem

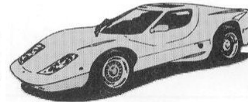
Testowanie jednostkowe

- Testowanie jednostkowe (ang. unit testing) zawiera się w dyscyplinie implementacja a nie testowania, gdyż to programiści są odpowiedzialni za stworzenie niezawodnych komponentów
- Sprawdzany jest przede wszystkim funkcjonowanie komponentu z punktu widzenia przypadku użycia, który jest realizowany
- Przypadki testowe przygotowywane są w taki sposób aby kontrolować możliwe wyczerpująco przepływy sterowania i danych
- Testy strukturalne (biała skrzynka)
- Testy wydajnościowe śledzą wykonania programu; identyfikują wąskie gardła i braki efektywności
- Testy solidności - identyfikują błędy wykonania skutkujące upadkiem lub obniżoną wydajnością (wycieki pamięci, niezainicjowane zmienne, błędy związane ze stosem)
- Testy funkcjonalne (czarna skrzynka) - zwane specyfikacyjnymi


White box

Reliability

Black box


Performance

Przeprowadzenie testów jednostkowych
(ang. Perform Unit Tests)

Purpose
- To verify the specification of a unit.
- To verify the internal structure of a unit.

Steps
- Execute Unit Tests
- Evaluate the Execution of Test
- Verify Test Results
- Recover from Halted Tests

Input Artifacts:	Resulting Artifacts:
Component	Component

Role: Implementer

More Information:
- Guidelines: Test Case

Przeglądanie kodu źródłowego
(ang. Review Code)

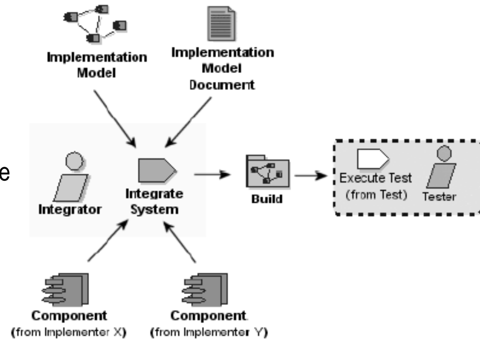
- Ważny mechanizm zapewniający i umożliwiający kontrolę jakości kodu
- Wspólny styl kodowania jest wspierany poprzez wykorzystywanie **wskazówek programistycznych** przez wszystkich członków zespołu oraz proces przeglądania kodu
- Wskazówki programistyczne (ang. *guidelines*) - lista dobrych praktycznych zwyczajów kodowania, np. konwencja nazewnictwa klas czy zmiennych; układ definicji klas; stosowane wcięcia w definicjach metod; ograniczenia w stosowaniu pewnych cech języków progr. (rekurencja, polimorfizm, ...)
- Odpowiednie przeglądy pozwalają często na wykrycie błędów, które nie zostały wykryte podczas wcześniejszego testowania i dlatego przeglądy są traktowane jako czynność komplementarna do testowania
- Protokoły przeglądów powinny zawierać wszystkie modyfikacje zalecane do wprowadzenia w analizowanym kodzie

Techniki przeglądania kodu

- **Inspekcje kodu** (ang. code inspection) - formalna technika oceny kodu, gdzie poszczególne wyrażenie (linie kodu i ich komentarze) są analizowane; zgłaszane uwagi są dokumentowane i przypisywane im są priorytety; w przypadku wymaganych poważnych zmian inspekcja jest powtarzana po ich wprowadzeniu; niezbędny odpowiedni trening i przygotowanie sesji, ale postrzegana jako bardzo efektywna
- **Przejścia** (ang. walkthrough) - programista prowadzi recenzenta(-ów) przez kod; recenzent zadaje pytania i formułuje uwagi dotyczące techniki, stylu, możliwych błędów, naruszenia standardów kodowania, itd.
- **Czytanie kodu** (ang. code reading) - kod jest przekazywany recenzentom (pracują samodzielnie i niezależnie); następnie może odbyć się spotkanie na którym zadawane są pytania i prezentowane uwagi; w przypadku braku możliwości spotkania (np. praca na odległość, ograniczenia czasowe) przekazywane programistom w formie pisemnej
- Kombinacje poprzednio wymienionych
- Świetna okazja do nauki i wymiany doświadczeń - tzw. synchronizacja pojęciowa (ang. *cognitive synchronization*)

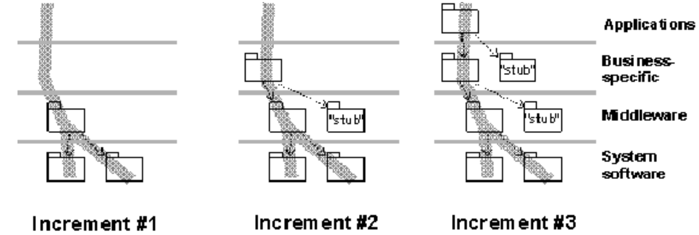
Integracja systemu

- Plan integracji definiuje implementowane w poszczególnych iteracjach komponenty, które są łączone w celu stworzenia funkcjonalnej wersji systemu (*build*)
- Niezbędne może być opracowywanie szkieletowych komponentów (pełnomocnicy i driver-y) pozwalających na uruchomienie i testowanie



Integracja systemu
(ang. Integrate System)

Purpose	
➜ To integrate the implementation subsystems piecewise into a build.	
Steps	
➜ Accept Subsystems and Produce Intermediate Builds ➜ Promote Baselines	
Input Artifacts: ➜ Component ➜ Implementation Model ➜ Implementation Model Document	**Resulting Artifacts:** ➜ Build
Role: Integrator	



Artefakty implementacji

Component	A **component** represents a piece of software code (source, binary or executable), or a file containing information (for example, a startup file or a ReadMe file). A component can also be an aggregate of other components; for example, an application consisting of several executables.
UML representation:	Components, possibly stereotyped as, for example, «application», «document», «executable», «file», «library», «page», «table» or «test component»
Role:	Implementer
Optionality:	Use of any of the stereotypes is optional.

Implementation Model	The **implementation model** is a collection of components, and the implementation subsystems that contain them. Components include both deliverable components, such as executables, and components from which the deliverables are produced, such as source code files.
UML representation:	Model, stereotyped as «implementation model».
Related Artifact:	Implementation Model Document
Role:	Integrator
More information:	➜ Checkpoints:Implementation Model

Artefakty implementacji (2)

Implementation Model Document	The **implementation model document** provides a detailed plan for integration within an iteration.
Roles:	Implementer / Integrator

Build	A build is an operational version of a system or part of a system that demonstrates a subset of the capabilities to be provided in the final product. A build comprises one or more components (often executable), each constructed from other components, usually by a process of compilation and linking of source code.
UML representation:	Package in the implementation model (either its top-level package or an implementation subsystem), stereotyped as «build».
Role:	Integrator