Inżynieria oprogramowania II

Wykład 7:

"UPEDU: Analiza i projektowanie
(ang. *analysis and design 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

Analogia do powstawania kryształu

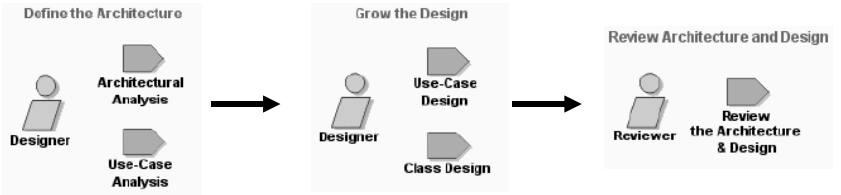
- Organizacja informacji podczas budowy oprogramowania a fizyczny proces krystalizacji, w którym płyn progresywnie ulega zmianie w kryształ
- Oprogramowanie widziane jako "krata" informacji (ang. lattice of information)
- W fizycznej kracie kryształu cząsteczki są powiązane poprzez różnego typu oddziaływania: fizyczne, chemiczne czy elektryczne
- W oprogramowaniu powiązania pomiędzy wyrażeniami dotyczą danych, przepływu sterowania, algorytmów, ...
- W obu przypadkach usunięcie jednego elementu ze struktury może skutkować diametralną zmianą własności struktury
- Sposób organizacji informacji jest kluczowy ze względu na niezawodność oprogramowania

Analogia do powstawania kryształu (2)

- Analiza i projektowanie składa się z czynności, które są niezbędne aby umożliwić krystalizację informacji w dobrze zdefiniowany program
- W procesie tym informacja jest stopniowo strukturalizowana w podstawowe elementy, pomiędzy którymi występują jasno zdefiniowane zależności
- Czynności analizy są wykorzystywane do zdefiniowania elementów informacji, natomiast czynności projektowania definiują powiązania, które wiążą elementy
- Proces krystalizacji informacji na który składa się "mieszanka" czynności analizy i projektowania nie jest prostoliniowy i jest zależny od wielu czynników (np. jakości wymagań, doświadczenia projektantów, stresu wynikającego z napiętego kalendarza, współpracy przedstawicieli klienta, wykorzystywanej metodologii, ...) => jakość rozwiązania
- 2 fazy krystalizacji: budowa szkieletu i rozrost kryształu wokół szkieletu

Czynności procesu

- 5 czynności:
  - analiza architektoniczna (ang. architectural analysis)
  - analiza przypadków użycia (ang. use-case analysis)
  - projektowanie przypadków użycia (ang. use-case design)
  - projektowanie klas (ang. class design)
  - przegląd elementów architektonicznych i projektowych (ang. review the architecture & design)
- Kolejność ich wykonania jest trudna do zaplanowania i poszczególne czynności przeplatają się

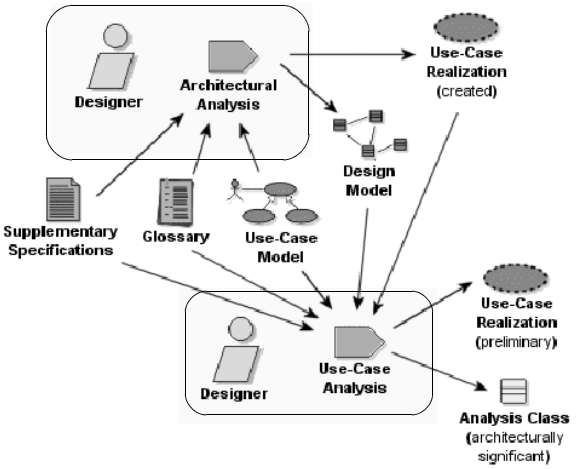


Czynności procesu (2)

- Schemat postępowania: zdefiniowanie architektury czy też szkieletu systemu a następnie na jego podstawie rozwijanie (przyrost) projektu systemu
- Poziom szczegółowości projektowania jest uzależniony od decyzji projektanta i musi uwzględniać poziom wiedzy i doświadczenia zespołu implementującego
- Poza elementami architektonicznymi istotne jest również "dopasowanie" systemu w kontekście operacyjnym (końcowi użytkownicy systemu) oraz wytwórczym (zespół projektowy i programistyczny)
- Planowanie i przydział ludzi w firmie wytwórczej opiera się na podstawowych komponentach architektury
- Architektura systemu jest najistotniejszym elementem, który może być wykorzystany do kontroli iteracyjnego i przyrostowego rozwoju systemu

Definiowanie architektury

- Zawiera zarówno czynności analityczne (zrozumienie wymagań) jak i projektowe (łączenie różnych elementów składających się na architekturę), które mogą przeplatać się nawet podczas pojedynczej sesji pracy
- Architektura systemu jest budowana w oparciu o informacje zebrane podczas rozpoznawania wymagań



Czynności architektoniczne

Definiowanie architektury (2)

- Czynności architektoniczne (analiza architektoniczna i analiza p.u.) są wykonywane równolegle z innymi czynnościami dyscypliny analizy i proj.
- Ponieważ architektura może być widziana jako szkielet, który pozwala na rozwój „kryształu informacji" oznacza to, że musi być zdefiniowana możliwie wcześnie
- Minimalizacja liczby zmiennych i parametrów ułatwia stworzenie wstępnej wersji architektury (np. pomija się wymagania niefunkcjonalne i ograniczenia implementacyjne)
- Szkielet oprogramowania jest podstawową strukturą produktu; zwykle jest to struktura logiczna, złożona jedynie z kilku elementów
- Architektura systemu obejmuje strukturalną organizację systemu, identyfikację komponentów oraz interakcje pomiędzy nimi. Ponadto dotyczy funkcjonalności, osiągów (ang. performance), wyboru spośród alternatyw, ...

Analiza architektoniczna
(ang. architectural analysis)

Purpose

- To define a candidate architecture for the system, based on experience gained from similar systems or in similar problem domains.
- To define the architectural patterns, key mechanisms and modeling conventions for the system.
- To define the reuse strategy
- To provide input to the planning process

Steps

- Develop Architecture Overview
- Survey Available Assets
- Identify Key Abstraction
- Identify Analysis Mechanisms
- Create Use-Case Realizations
- Identify Stereotypical Interactions
- Review the Results

Input Artifacts:	Resulting Artifacts:
Glossary Supplementary Specifications Use-Case Model	Design Model Use-Case Realization

Frequency: Once per iteration, with most work occurring in the early iterations; later iterations visit the activity primarily to validate and adjust the analytic aspects of the architecture.

Role: Designer

Analiza przypadków użycia
(ang. use-case analysis)

Purpose
- To identify the classes which perform a use case's flow of events.
- To distribute the use case behavior to those classes, using use-case realizations.
- To identify the responsibilities, attributes and associations of the classes.
- To note the usage of architectural mechanisms

Steps
- Supplement the Use-Case Descriptions
- For each use case realization
  - Find Analysis Classes from Use-Case Behavior
  - Distribute Behavior to Analysis Classes
- For each resulting analysis class
  - Describe Responsibilities
  - Describe Attributes and Associations
  - Define Attributes
  - Establish Associations between Analysis Classes
  - Describe Event Dependencies between Analysis Classes
  - Qualify Analysis Mechanisms
- Evaluate the Results of Use-Case Analysis

Input Artifacts:	Resulting Artifacts:
Design Model Glossary Supplementary Specifications Use-Case Model Use-Case Realization	Analysis Class Use-Case Realization

Frequency: Once per iteration, for a set of Artifact: Use-Case Realizations

Role: Designer

Korzyści z dobrej architektury

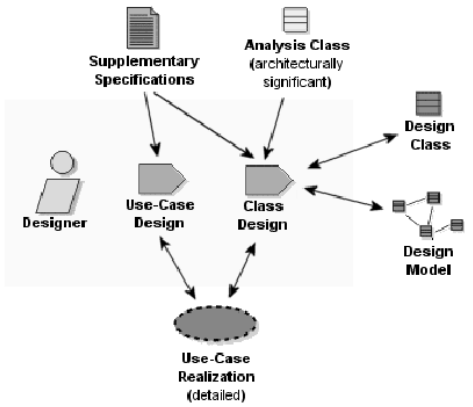
- Nie ma gwarancji, że wybrana architektura będzie idealna lub nawet odpowiednia, zwłaszcza, że nie istnieje miara jakości architektury; może się zdarzyć, że podczas późniejszych faz procesu są opracowywane alternatywne wersje architektury
- Dobra architektura pozwala utrzymywać intelektualną kontrolę nad projektem, dawać sobie radę z jego złożonością oraz utrzymywać integralność systemu
- Zapewnia efektywną bazę do ponownego wykorzystania oraz zarządzania projektem
- Dobrze przemyślna ułatwia oparcie się na komponentach:
  - komponent spełnia jasne, klarowne funkcje w kontekście architektury
  - komponent odpowiada i zapewnia fizyczną realizację zbioru interfejsów
  - komponent istnieje relatywnie do (w ramach) zadanej architektury

Ponowne wykorzystanie i elastyczność systemu

- Poprzez jasne wyrażenie podstawowych komponentów oraz krytycznych interfejsów, czynności architektoniczne pozwalają członkom zespołu na myślenie o ponownym wykorzystaniu i elastyczności
- Ponowne wykorzystanie (ang. reusability) jest definiowane jako zdolność włączenia w system pewnych zidentyfikowanych istniejących komponentów: dotyczy to zarówno wewnętrznych, np. wspólnych (standardowych) części systemu jak i zewnętrznych np. dostępnych z półki (ang. *off-the-shelf*); architektura może ułatwiać ponowne wykorzystanie na szerszą skalę (np. poprzez ponowne wykorzystanie samej architektury w ramach linii produktu - różna funkcjonalność, ta sama dziedzina)
- Elastyczna (ang. resilient) architektura:
  - poprawia łatwość utrzymania i rozszerzania
  - pozwala na jasne rozdzielenie pracy pomiędzy członków zespołu
  - umożliwia ukrywanie zależności sprzętowych i systemowych

Rozwijanie projektu
(ang. Growing the Design)

- Projektowanie architektoniczne jest szczególnie trudne gdyż obejmuje nie tylko zagadnienia techniczne ale także biznesowe i organizacyjne (np. przygotowanie przyszłego rozwoju systemu czy też potrzebę skrótów)
- Istnieje szereg mechanizmów wspomagających dobre projektowanie; większość jest obecna w metodykach i notacjach (należy się jednak ograniczać do kliku takich mechanizmów, które są dobrze zrozumiane przez cały zespół)
- Warto też opracowywać i wykorzystywać wskazówki (ang. guidelines), co pozwala na uniformizację procesu i redukcję różnorodności pomiędzy projektantami



Projektowanie klas
(ang. class design)

Purpose
- To ensure that the class provides the behavior the use-case realizations require.
- To ensure that sufficient information is provided to unambiguously implement the class.
- To handle non-functional requirements related to the class.
- To incorporate the design mechanisms used by the class.

Steps
- Create Initial Design Classes
- Identify Persistent Classes
- Define Class Visibility
- Define Operations
- Define Methods
- Define States
- Define Attributes
- Define Dependencies
- Define Associations
- Define Generalizations
- Handle Non-Functional Requirements in General
- Evaluate Your Results

Input Artifacts:	Resulting Artifacts:
Analysis Class Supplementary Specifications Use-Case Realization	Design Class Design Model Use-Case Realization

Role: Designer

Projektowanie przypadków użycia
(ang. use-case design)

Purpose
- To refine use-case realizations in terms of interactions.
- To refine requirements on the operations of design classes.
- To refine requirements on the operations of subsystems and/or their interfaces.
- To refine requirements on the operations of capsules

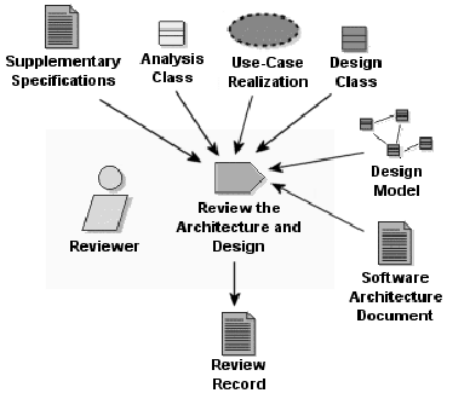
Steps
- Describe Interactions Between Design Objects
- Simplify Sequence Diagrams using Subsystems (optional)
- Describe Persistence-related Behavior
  - Writing Persistent Objects
  - Reading Persistent Objects
  - Deleting Persistent Objects
  - Modeling Transactions
  - Handling Error Conditions
  - Handling Concurrency Control
- Refine the Flow of Events Description
- Unify Classes and Subsystems
- Evaluate Your Results

Input Artifacts:	Resulting Artifacts:
Supplementary Specifications Use-Case Realization	Use-Case Realization

Role: Designer

Przegląd architektury i projektu
(ang. Reviewing Architecture and Design)

- Bardzo istotne z punktu widzenia jakości architektury i projektu
- Umożliwiają przekazywanie przez zaangażowane osoby ich rozumienia tworzonego produktu i jednocześnie stanowią ważny element walidacji wykonanej pracy
- Przeglądy mogą mieć różną formę i poziom szczegółowości (np. pobieżny - w celu sprawdzenia odpowiedniości stopnia szczegółowości)
- Komentarze i decyzje (oraz ich racjonalne uzasadnienia) dotyczące analizowanych elementów powinny być w taki sposób zorganizowane aby ułatwiały ewentualną modyfikację czy poprawienie architektury



- Ponadto rozpatrywanie takich aspektów jak styl czy estetyka pozwala ujednolicenie projektu, zwiększenie czytelności oraz unikanie niespodzianek

Przegląd architektury i projektu
(ang. Reviewing Architecture and Design)

Input Artifacts:	Resulting Artifacts:
Analysis Class Design Class Design Model Software Architecture Document Supplementary Specifications Use-Case Realization	Review Record

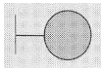
Role: Reviewer

Guidelines:
- Reviews

Checkpoints:
- Software Architecture Document
- Design Model

Workflow Details:
- Analysis & Design
  - Review Architecture and Design

Podstawowe artefakty

Boundary Class  Control Class  Entity Class 	Class Diagrams Iteration Diagrams Sequence Collaboration	Operation State Attributes	Mappings from Analysis Class Describe Realization of Use Case Mirror Implementation

Klasy analizy
(ang. analysis classes)

- Wykorzystywane aby wychwycić główne “grupy odpowiedzialności” w systemie; reprezentują prototypowe klasy systemu (pierwsze przybliżenie); definiują podstawowe pojęcia (na wysokim poziomie abstrakcji) systemu

Stereotypy klas:

- Klasy graniczne (ang. *boundary classes*) - modelują te części systemu, które zależą od elementów istniejących poza systemem, z którymi system musi współpracować; przykładowe interfejsy użytkownika, interfejsy systemu lub interfejsy urządzeń
- Klasy kontrolne (ang. *control classes*) - pokrywają kontrolę zachowania, które jest specyficzne dla jednej bądź kilku p.u.; modelują dynamikę systemu; stanowią pomost między klasami granicznymi a jednostkowymi
- Klasy “jednostkowe” (ang. *entity classes*) - modelują informacje wraz ze skojarzonym zachowaniem, które są przechowywane; nie są zwykle związane z żadnym konkretnym przypadkiem użycia oraz są niezależne od środowiska czy aktorów; reprezentują kluczowe pojęcia rozwijanego systemu

Realizacja przypadków użycia
(ang. use-case realization)

- Stanowi centralny obiekt zainteresowania w początkowej fazie analizy i projektowania; precyzuje jak poszczególne przypadki użycia będą realizowane w ramach modelu projektowego (jak obiekty współpracują w ramach p. u.)
- Celem jest rozdzielenie spraw dotyczących specyfikacji systemu (reprezentowanych przez model przypadków użycia i wymagania do systemu) od spraw projektowania systemu
- Realizacja p. u. jest perspektywą projektową p. u.; grupuje różne elementy związane z projektową stroną p. u. (np. diagram występujących klas, diagram przebiegu ilustrujący przepływ zdarzeń w otoczeniu instancji tych klas)
- Zwykle każda realizacja p. u. posiada co najmniej jeden diagram interakcji przedstawiający uczestniczące obiekty oraz ich interakcje

Klasy projektowe
(ang. design class)

- Klasa projektowa - opis zbioru obiektów mających taki sam zakres odpowiedzialności, powiązania, operacje, atrybuty i semantykę; czyli typowa klasa w rozumieniu programowania obiektowego
- Definicje klas powinny być na tym poziomie już wystarczająco szczegółowe, aby ich implementacja była jednoznaczna
- Uwzględnione powinny zostać wymagania niefunkcjonalne związane z klasami
- Pozostałe elementy jak podsystemy, pakiety i kolaboracje pokazują jak klasy są grupowane i jak współpracują
- Na kształt i organizację klas może mieć wpływ język implementacji

Model projektowy
(ang. design model)

- Model obiektów opisujący realizację p.u.; służy jako abstrakcja modelu implementacyjnego i jego kodu źródłowego
- Opisuje jak kod źródłowy będzie ustrukturalizowany i pisany
- Przyjmuje postać hierarchii pakietów (podsystemów projektowych i pakietów projektowych) z liśćmi reprezentującymi klasy bądź realizację przypadków użycia
- Dobry model projektowy jest zgodny z wymaganiami systemowymi, dający się stosunkowo łatwo przystosować do zmian oraz gotowy do wykorzystania podczas implementacji