W7 [WZORCE PROJEKTOWE]

1. Pojęcie wzorca projektowego

Uniwersalne, sprawdzone w praktyce rozwiązanie często pojawiających się, powtarzalnych problemów projektowych. Określa zasadniczą część jego rozwiązania tak, aby można było je zastosować wiele razy, za każdym razem w nieco inny sposób Pokazuje powiązania i zależności pomiędzy klasami oraz obiektami. Wzorzec projektowy nie jest gotową implementacją rozwiązania, lecz jego opisem. Po co studiować wzorce projektowe? By uzyskać dobre rozwiązanie; żeby nie zaczynać wszystkiego od początku; pozwalają na komunikację między fachowcami.

2. Geneza wzorców projektowych, Go4

Wzorce projektowe w informatyce wywodzą się z wzorców projektowych w architekturze, które zostały zaproponowane przez amerykańskiego architekta Christophera Alexandra i miały ułatwić konstruowanie mieszkań i pomieszczeń biurowych. Pomysł ten nie został jednak przyjęty.
Termin wzorca projektowego został wprowadzony do inżynierii oprogramowania przez Kenta Becka oraz Warda Cunninghama w 1987 roku, kiedy, na konferencji OOPSLA, przedstawili wyniki swojego eksperymentu dotyczącego ich zastosowania w programowaniu. Wzorzec projektowy został spopularyzowany w 1995 przez Bandę Czterech (**Erich Gamma**, **Richard Helm**, **Ralph Johnson** oraz **John Vlissides**) dzięki książce *Inżynieria oprogramowania: Wzorce projektowe*.

3. Metody opisu wzorców

Nazwa, Intencja (Po co?), Problem (Co rozwiązuje?), Rozwiązanie (Opis rozwiązania), Uczestnicy, Konsekwencje, Implementacja (Sposób użycia).

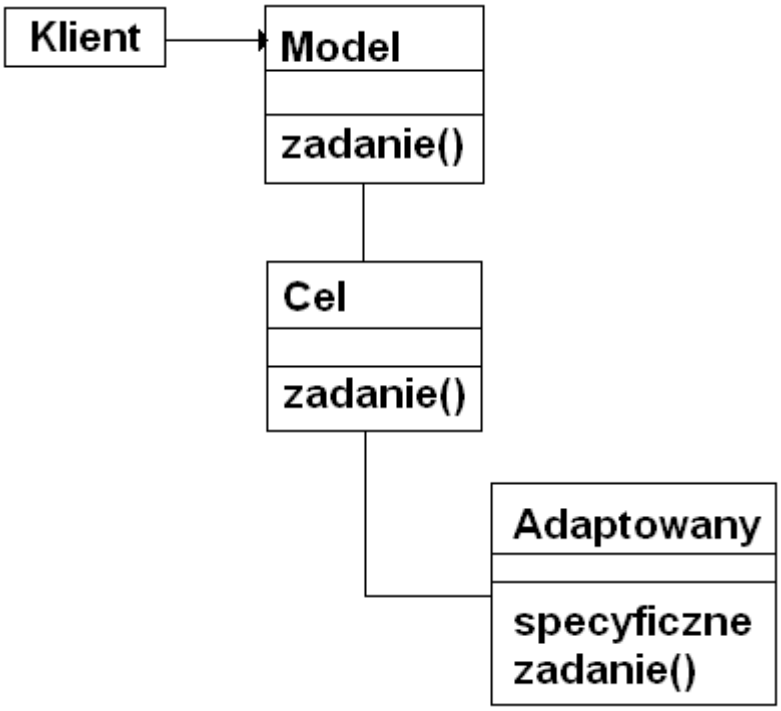
4. Zasada identyfikacji zmienności i hermetyzacji

Odkryj zmienności i hermetyzuj je.

5. Wzorce:

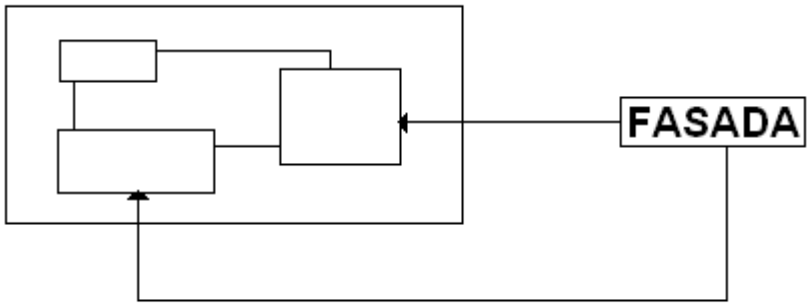
a. Adapter

Nazwa: Adapter
Intencja: Dopasowanie istniejącego interfejsu do innego interfejsu (lub obiektu do interfejsu).
Problem: Niezgodność interfejsu (obiekt zachowuje się tak jak trzeba, ale ma nieodpowiedni interfejs).
Rozwiązanie: Obudowanie obiektu pożądanym interfejsem.
Uczestnicy: Adapter, Adaptowany, Cel, Użytkownik.
Konsekwencje: Dopasowanie istniejących obiektów do tworzonych struktur i uniknięcie ograniczeń.
Implementacja:



b. Fasada

Nazwa: Fasada
Intencja: Uproszczenie sposobu korzystania z istniejącego systemu.
Problem: Potrzeba wykorzystania tylko części możliwości systemu.
Rozwiązanie: Nowy interfejs do istniejącego systemu:



Uczestnicy: Interfejs dobudowywany (specjalizowany)
Konsekwencje: Uproszczenie wykorzystania systemu
Implementacja: Definicja nowej klasy (o pożądanym interfejsie). Wykorzystanie istniejących funkcji systemu.

c. Most

Pozwala oddzielić abstrakcję obiektu od jego implementacji. Zaleca się stosowanie tego wzorca aby: odseparować implementację od interfejsu; poprawić możliwości rozbudowy klas, zarówno implementacji, jak i interfejsu (m.in. przez dziedziczenie); ukryć implementację od klienta, co umożliwia zmianę implementacji bez zmian interfejsu.
Przykład: Wyobraźmy sobie abstrakcję, jaką jest figura. Można ją wyszczególnić na np. kwadraty, czy trójkąty, jednak są pewne metody dla każdej figury jak np. rysowanie. Jednak rysowanie może być różne dla różnych bibliotek graficznych czy systemów operacyjnych. Wzorzec mostu pozwala na stworzenie nowych klas, które dostarczają konkretnych implementacji do rysowania. Klasa abstrakcyjna figury dostarcza informacji o figurze (np. wielkość), podczas gdy implementacja dostarcza interfejs do rysowania.

W8 [DOKUMENTACJA WYMAGAŃ]

1. Pojęcie aktora i przypadku użycia

Aktor to podmiot (np. użytkownik lub inny system), który wchodzi w interakcję z systemem. Przypadek użycia – specyfikacja gwarantująca aktorowi wykonanie przez system usługi realizującej cel aktora. Uczestnik – podmiot biorący udział w procesach biznesowych wspieranych przez system. Uczestnik może być aktorem systemu lub nie (jeśli nie wchodzi w intencję z systemem).

2. Metody opisu przypadków użycia (UC)

Każdy przypadek użycia może być opisany scenariuszem, diagramem oraz w języku naturalnym.
1) Nazwa
2)Opis
3) Aktorzy
4) Aktor główny
5) Warunki wstępne
6) Stan przed i po

3. Klasyfikacja przypadków użycia wg A. Cockburna

- nieformalny opis - kilka luźnych zdań ogólnie opisujących przypadek
- formalny opis - kilka paragrafów, podsumowanie
- pełen opis - formalny dokument

1) Słońce - opis ogólny
2) Plaża - to, co user chce widzieć

3) Głębiny - to, co po kolei się wykonuje

4. Notacja UML do przedstawiania diagramów UC

W notacji UML używamy prostokątów dla systemu, elips do zobrazowania przypadków użycia oraz „ludzików" do zobrazowania aktorów. Łączą ich strzałki (tylko aktor → przypadek użycia lub aktor ← przypadek
użycia, nie aktor → aktor). Strzałek może nie być wcale w momencie, gdy zarówno aktor jak i przypadek użycia przekazuje jakąś informację lub polecenie.

5. Dobre i złe przypadki użycia, metody oceny

Dobry - nie jest szczegółowy, aktor ma wyraźny cel w wykonaniu przypadku użycia. Złe to takie, jak jest za szczegółowe (np wprowadza liczbę), lub jest bez celu (np. logowanie się do systemu).

6. Zastosowania UC, zalety i wady

Zalety i zastosowania:
Jest to forma identyfikacji wymagań
Można w przyszłości na tym testy opierać
Można na tym monitorować postęp pracy nad systemem
Naturalność - czytasz i to rozumiesz
Wady:
Jak mamy system, który się nie komunikuje z innymi systemami i aktorami, to UC jest gówno warte (Nie wymyśliliśmy nic lepszego)

7. Rodzaje wymagań

Wymagania funkcjonalne: opisują funkcje wykonywane przez system (np. logowanie użytkownika).
Wymagania niefunkcjonalne: opisują ograniczenia przy których system musi realizować swoje funkcje (np. ograniczenia prawne, standardy określone w dokumencie XXX etc.)

8. Dokument specyfikacji wymagań

Częściowo ustrukturalizowany zapis wykorzystujący zarówno język naturalny, jak i proste, częściowo przynajmniej sformalizowane notacje. Opisywana w: języku naturalnym (wada: niejednoznaczność), formalizm matematyczny, język naturalny strukturalny (ograniczone słownictwo i składnia), tablice/formularze, diagramy blokowe, kontekstowe, przypadków użycia.

W9 [FAZA PROJEKTOWANIA]

1. Czym jest faza projektowania

Celem fazy projektowania jest opracowanie modelu projektowego, czyli szczegółowego opisu implementacji systemu. W odróżnieniu od analizy, w projektowaniu dużą rolę odgrywa środowisko implementacji. Projektanci muszą więc posiadać dobrą znajomość języków, bibliotek i narzędzi stosowanych w trakcie implementacji.

2. Czynności fazy projektowania

W fazie projektowania następuje dalsze uszczegółowienie wyników analizy. Projekt musi być wystarczająco szczegółowy, aby mógł być podstawą implementacji. Jego stopień szczegółowości zależy od poziomu zaawansowania programistów. W czasie projektowania dokonuje się również projektowania składowych systemu nie związanych z dziedziną problemu oraz proponuje się optymalizację systemu. Zadaniem projektanta jest też dostosowanie do ograniczeń i możliwości środowiska implementacji oraz określenie fizycznej struktury systemu.

- Szczegółowy opis implementacji systemu
- Uszczegółowienie wyników analizy
- Określenie fizycznej struktyry systemu
- Dostosowanie do środowiska implementacji

3. Projektowanie składowej bazy danych

Niemal każda aplikacja musi w sposób trwały przechowywać dane. Projekt składowej zarządzania danymi jest więc praktycznie nieodzownym elementem projektu systemu. Dane w sposób trwały

mogą być przechowywane na dwa podstawowe sposoby: w pliku lub w bazie danych (relacyjnej, obiektowej lub innej). Poszczególne elementy danych mogą być przechowywane w następującej postaci: w jednej relacji lub pliku lub w odrębnym pliku dla każdego rodzaju obiektów lub krotek.
Jak wiadomo, dane są przetwarzane w pamięci operacyjnej. Dlatego też w czasie pracy oprogramowania muszą być sprowadzone z pamięci trwałej do pamięci operacyjnej. Sprowadzenie danych do pamięci operacyjnej oraz zapisanie do trwałej pamięci może odbywać się na bieżąco, kiedy program zażąda dostępu i gdy następuje zapełnienie bufora lub na zlecenie użytkownika.

4. Projektowanie składowej interfejsu użytkownika

Z punktu widzenia użytkownika interfejs aplikacji jest niezwykle ważny, a jego dobre zaprojektowanie i funkcjonalność nieraz przesądza o satysfakcji użytkownika. Dlatego też projektant powinien poświęcić wystarczająco dużo uwagi i wysiłku w celu opracowania interfejsu jak najlepiej odpowiadającego potrzebom i przyzwyczajeniom użytkownika. Przed przystąpieniem do prac projektowych nad interfejsem warto jest zbadać dotychczasowe przyzwyczajenia użytkownika. Można to zrobić na przykład przyglądając się oprogramowaniu, z którym użytkownik pracuje na co dzień i zbierając jego opinie o dobrych i złych stronach interfejsu tego oprogramowania. Interfejs powinien być intuicyjny w obsłudze.
Reguła Millera 7 +/-2 (człowiek może się jednocześnie skupić na 5 -9 elementach), tekst do okienek powinien być po lewej lub na gorze od okienka, guziki OK i Anuluj na prawej na dole, prosta obsługa błędów (np. przy wprowadzaniu danych), grupowanie podobnych funkcji, pytanie się, czy na pewno chcesz to zrobić, paski postępu.

5. Rezultaty fazy projektowania

Poprawiony dokument opisujący wymagania, poprawiony model, uszczegółowiona specyfikacja projektu zawarta w słowniku danych, dokument opisujący stworzony projekt, zasoby interfejsu użytkownika, projekt bazy danych, projekt fizycznej struktury systemu, poprawiony plan testów, harmonogram fazy implementacji.

W10 [TESTOWANIE]

1. Weryfikacja i atestowanie oprogramowania

Weryfikacja (verification) to testowanie zgodności systemu z wymaganiami zdefiniowanymi w fazie określenia wymagań. Atestowanie (validation) oznacza ocenę systemu lub komponentu podczas lub na końcu procesu jego rozwoju na podstawie zgodności z wyspecyfikowanymi wymaganiami. Atestowanie jest więc weryfikacją końcową; nazywane jest czasem zatwierdzeniem.

2. Pojęcie błędu i błędnego wykonania

Błąd to niepoprawna konstrukcja znajdująca się w programie, która może doprowadzić do jego niewłaściwego działania. Błędne wykonania oznacza niepoprawne działanie systemu w trakcie jego pracy.

3. Klasyfikacja testów, metody wykrywania błędów

Klasyfikacja testów:

- testy dynamiczne, które polegają na wykonywaniu programu i porównywaniu uzyskanych wyników z wynikami poprawnymi
- testy statyczne, oparte jedynie na analizie kodu tworzonego programu. Takie testy są zwykle wykonywane przez programistę systemu

Wykrywanie błędów:

- testy funkcjonalne – zakładają znajomość wymagań wobec testowanych funkcji. System traktowany jest więc jako tzn. czarna skrzynka, która w nieznany sposób realizuje wymagane funkcje.
- testy strukturalne – zakładają znajomość sposobu implementacji testowanych funkcji.

- Biała skrzynka - patrzenie w kod i dobieranie danych testowych
- Czarna skrzynka - bez patrzenia w kod, klepanie aż program się wysypie.

Można zasiać x błędów w programie i dać testerom. Na podstawie wykrytych przez testerów błędów mam pogląd na to ile błędów jest naprawdę w systemie.

4. Metoda weryfikacji, przeglądu i audytu

Metody weryfikacji: przeglądy, inspekcje, testowanie, sprawdzanie, audytowanie lub inna działalność ustalająca i dokumentująca, czy składowe, procesy, usługi lub dokumenty zgadzają się z wyspecyfikowanymi wymaganiami

Przegląd jest procesem lub spotkaniem, podczas którego produkt roboczy lub pewien zbiór produktów roboczych jest prezentowany dla personelu projektu, kierownictwa, użytkowników, klientów lub innych zainteresowanych stron w celu uzyskania komentarzy, opinii i akceptacji.

Audytem nazywany jest niezależny przegląd i ocena jakości oprogramowania, która zapewnia zgodność z wymaganiami na oprogramowanie, a także ze specyfikacją, generalnymi założeniami, standardami, procedurami, instrukcjami, kodami oraz kontraktowymi i licencyjnymi wymaganiami.

Inspekcja - technika oceny prowadzona przez osoby biorące udział w projekcie, (ale nie będące autorami danego kodu). Wynikiem jest jedna informacja zbiorcza, która nie daje informacji o tym kto się obija).

W11 [ZARZĄDZANIE KONFIGURACJĄ]

1. Pojęcie konfiguracji oprogramowania

Konfiguracja oprogramowania mówi nam w jakim statusie jest dane oprogramowanie. Tak jak konfiguracja pulpitu, jakichś pasków zadań itp.

2. Zarządzanie konfiguracją oprogramowania (ZKO) – organizacja i zadania

Celem zarządzania konfiguracją oprogramowania jest planowanie, organizowanie, sterowanie i koordynowanie działań mających na celu identyfikację, przechowywanie i zmiany oprogramowania w trakcie jego rozwoju, integracji i przekazania do użycia.

3. Pojęcie pozycji konfiguracji (PK), produktu bazowego, wydania, wersji i statusu konfiguracji

PK - Wszystkie elementy projektu mają swoje pozycje konfiguracji (dokumentacje, jakieś moduły). PK posiada projekt oraz coraz to jego drobniejsze komponenty.

PB Produkt Bazowy - PK (ta główna) zaakceptowana, stanowiąca podstawę do dalszego rozwoju programu.

Status konfiguracji - przedstawiany w postaci tabelki, mowi nam o tym jakie PK ze sobą będą grały.

Wydanie - oficjalne przekazanie na zewnątrz firmy produktu (robione z jakiegoś produktu bazowego)

Wersje (warianty) - PK, które ma identyczną logikę ale rożni się w pewnych aspektach (platforma, jezyk, dodatki testowe itp)

4. Metody identyfikacji i opisu PK

Są PK niskiego poziomu (dokumenty, moduły itp), oraz wyższego poziomu (konfiguracje). Do opisu powinno służyć :

Nazwa

Opis

Wersja xxx.xxx.xxx

5. Pojęcie jakości – różne ujęcia

Azja – bliskie ideałowi

USA – ważne, żeby się sprzedawało

Europa – żeby spełniało oczekiwania (definicja przyjęta również u nas)

Jakość zależy od osoby, która dany produkt ocenia.

6. Jakość oprogramowania wg norm ISO i IEEE

ISO 9000 - norma ogólna

ISO 9126 - norma programowania

7. Definiowania jakości oprogramowani i dobór metryk

Def: Spełnia oczekiwania te co są dzisiaj i te co mogą być za kilka lat -> łatwość taniość zmian.

Metryki:
Kryteria jakości
- łatwe w obsłudze
Miara:
- czas w jakim kobitka z biura wklepie coś do systemu.

8. Jakość produktów i jakość procesów wytwórczych

Szybsze oddanie produktu wiąże się z tym, że będzie miał więcej błędów.
Jakość procesów wytwórczych:
Jeżeli programista pracował maksymalnie po 8h to jest szansa, że program będzie zawierał mniej błędów -> będzie lepszej jakości.

Wersja 1.0

Autorzy: Jakub Dudkowski, Sebastian Maciejewski, Michał Ogrodowski, Magdalena Przybyło, Michał Lewandowski, Dominik Przybysz.