



Marcin Majer

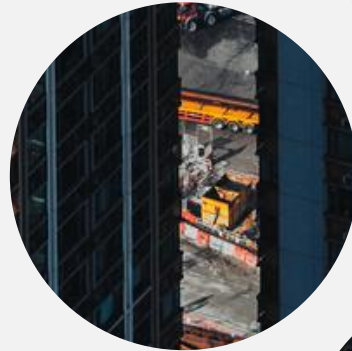
Architektury nowoczesnych systemów informatycznych

Politechnika opolska

2024

Plan wykładów

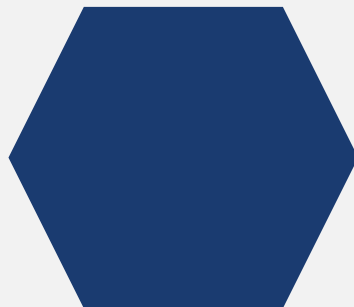
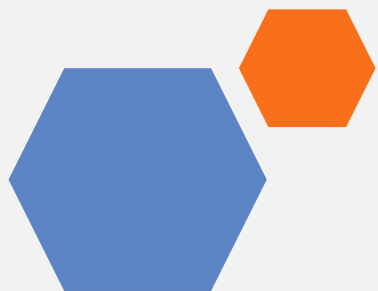
- Wprowadzenie do zagadnień, które będą omawiane na zajęciach, omówienie literatury i sposobów zaliczenia przedmiotu. Przetwarzanie wiedzy - komunikacja i użycie języka
- Związywanie modelu z implementacją
- Wyizolowanie dziedziny i wyrażenie modelu w programie
- Cykl życia obiektu dziedziny
- Użycie języka
- Moment przełomowy
- Odkrywanie pojęć niejawnych
- Projekt elastyczny
- Stosowanie wzorców analitycznych i projektowych oraz powiązanie ich z modelem
- Refaktoryzacja ku głębszemu zrozumieniu
- Utrzymywanie integralności modelu
- Destylacja
- Struktury dużej skali
- Łączenie strategii
- Kolokwium zaliczeniowe.



Wykład 2

Związanie modelu z implementacją





Modele mają wiele odmian i spełniają wiele ról, nawet te ograniczone kontekstem wyznaczonym przez projekt programistyczny. Projekt sterowany dziedziną wymaga modelu, który będzie nie tylko wspierał początkowy proces analizy, ale będzie stanowić podstawę całego procesu.

**Projektowanie sterowane dziedziną
wymaga również innego spojrzenia
na modelowanie**



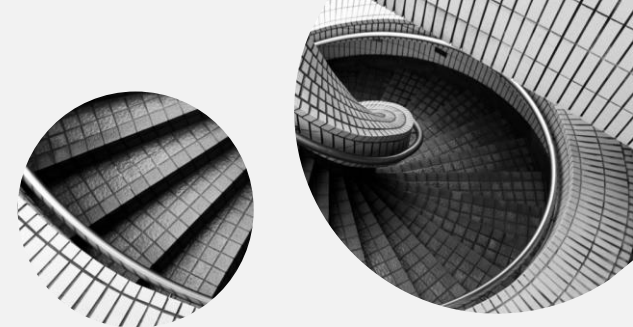
Projektowanie sterowane modelem



Astrolabium, używane do wyznaczania pozycji gwiazd, jest mechaniczną implementacją modelu nieba

Średniowieczny komputer astronomiczny Astrolabium zostało wynalezione przez starożytnych Greków, a następnie udoskonalone przez średniowiecznych arabskich naukowców. Obracające się koło (nazywane *rete*) stanowiło reprezentację położenia ustalonych gwiazd na sferze niebieskiej. Wymienne płyty, na których wryte były lokalne współrzędne astronomiczne, reprezentowały widoki z różnych szerokości geograficznych. Przesuwanie koła *rete* po płycie umożliwiało wykonanie obliczeń pozycji ciał niebieskich w dowolnym momencie roku. W przypadku posiadania pozycji gwiazdy lub Słońca można było też obliczyć czas astronomiczny. Astrolabium stanowiło mechaniczną implementację obiektowego modelu nieba.

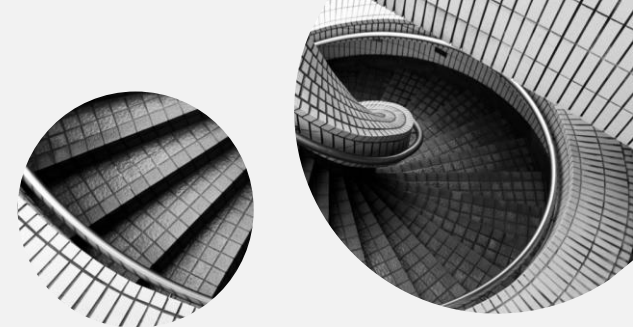
Projektowanie sterowane modelem



Astrolabium, używane do wyznaczania pozycji gwiazd, jest mechaniczną implementacją modelu nieba

Jeżeli projekt lub jego kluczowe części nie odwzorowują modelu dziedziny, wtedy model ten nie ma żadnej wartości, a cała poprawność programu staje pod znakiem zapytania. Jednocześnie złożone mapowania pomiędzy funkcjami modelu i projektu są trudne do zrozumienia i w praktyce utrzymanie zmian w projekcie staje się niemożliwe. Pomiędzy analizą a projektem tworzy się śmiertelna przepaść, uniemożliwiająca im wymianę między sobą zebranych przemyśleń.

Projektowanie sterowane modelem

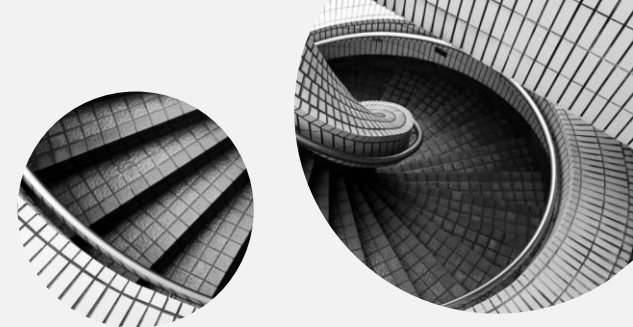


Język wszechobecny

Więc zdanie najpierw stwórz
By sedno zeń wykroić
Przesuwaj słowa, sortuj już
Daj szansę zniknąć im
Kolejność fraz — to pełny luz
Zupełnie bez znaczenia

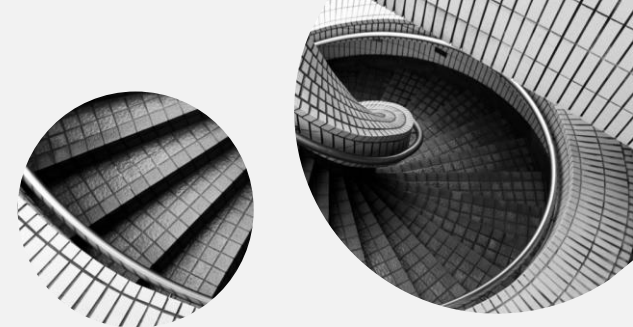
— *Lewis Carroll, „Poeta Fit, Non Nascitur”*

Projektowanie sterowane modelem



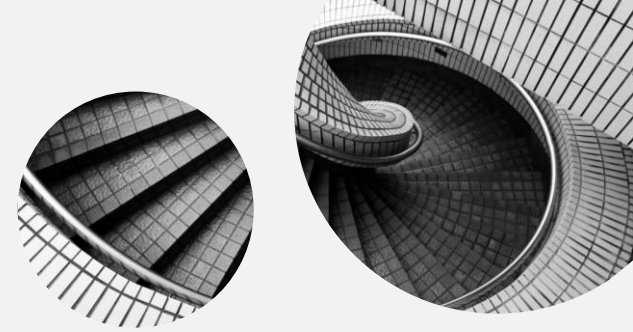
W celu odwzorowania modelu dziedziny projektuj system informatyczny stopniowo, tak aby odzwierciedlał go w bardzo dosłowny sposób, czyniąc mapowanie oczywistym. Wielokrotnie sprawdzaj model i modyfikuj go w taki sposób, aby w programie był implementowany bardziej naturalnie, nawet wtedy, gdy starasz się, aby lepiej odzwierciedlał założenia dziedziny. Domagaj się wspólnego modelu dobrze spełniającego obie te funkcje, a także wykorzystującego *JĘZYK WSZECHOBECNY*.

Projektowanie sterowane modelem



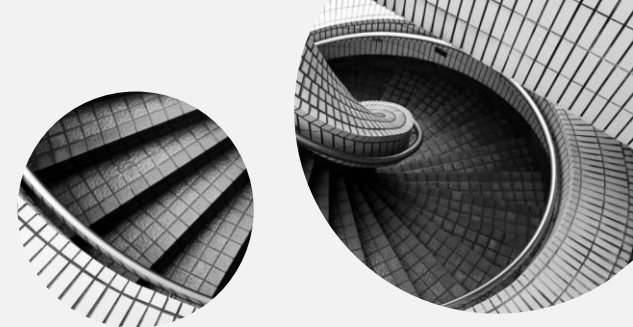
Inspiruj się terminologią użytą w modelu i wykorzystuj ją w projekcie, razem z podstawowym przypisaniem odpowiedzialności obiektów. Kod zacznie wyrażać model, dlatego dowolna zmiana wprowadzona do niego będzie mogła stanowić zmianę do modelu. Jej konsekwencje będą odczuwalne odpowiednio również w innych obszarach aktywności projektowej.

Projektowanie sterowane modelem



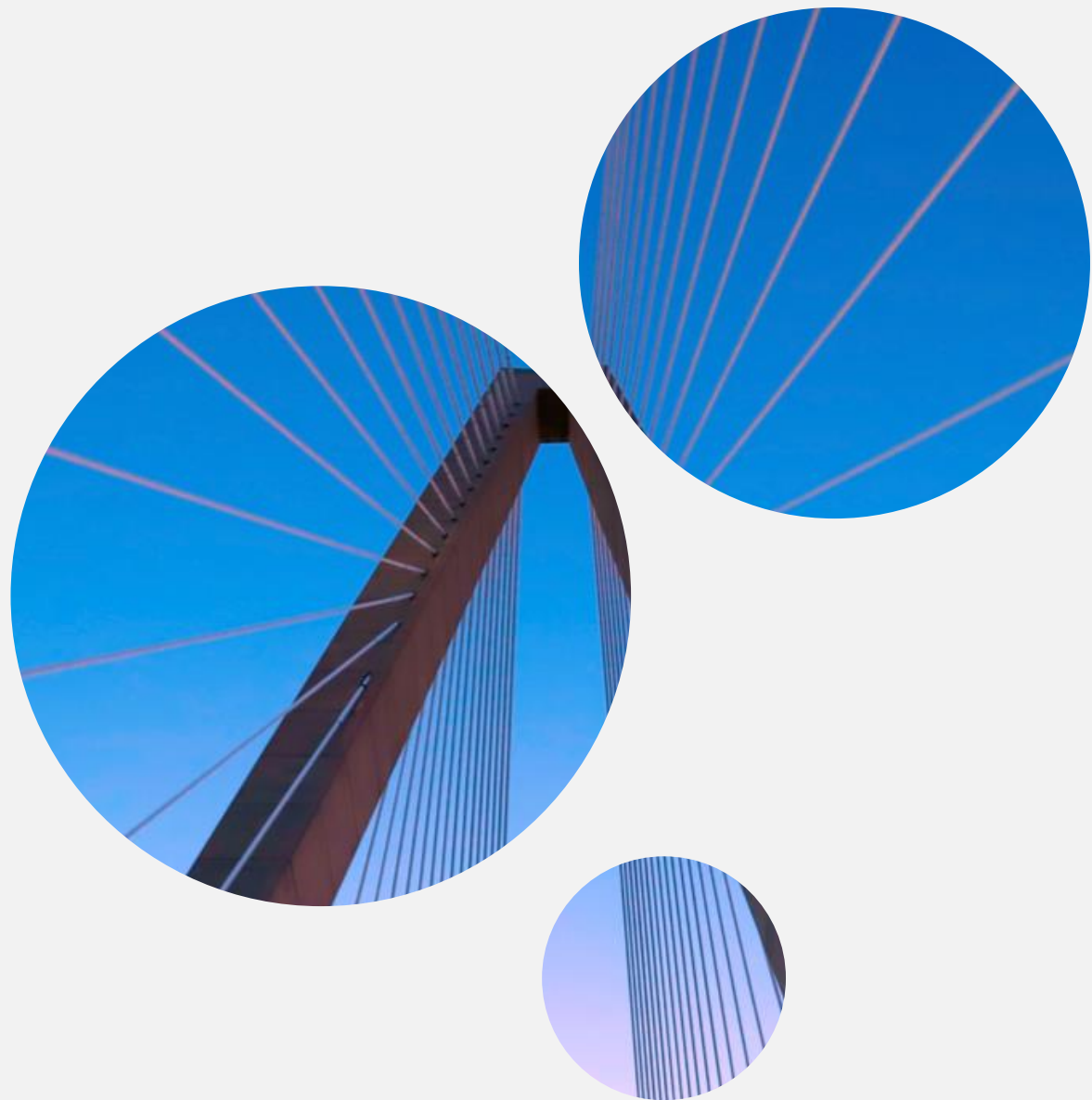
Niewolnicze powiązanie implementacji z modelem wymaga zazwyczaj użycia narzędzi oraz języków programistycznych wspierających paradygmat modelowania, jak chociażby programowania obiektowego.

Projektowanie sterowane modelem



Wspólnie tworzony model zmniejsza ryzyko popełnienia pomyłki, ponieważ projekt będzie bezpośrednim rezultatem szczegółowo przeanalizowanego modelu. Projekt, a nawet sam kod będzie przekazywać te same informacje, co model.

Modelowanie praktyczne



Modelowanie praktyczne



Jeżeli osoby tworzące kod nie czują odpowiedzialności za model lub też nie rozumieją, jak go zastosować w aplikacji, wtedy model ten nie może w żaden sposób wpłynąć na oprogramowanie. Jeżeli programiści nie uświadomią sobie, że zmiana kodu pociąga za sobą zmiany w modelu, wtedy każda przeróbka kodu osłabi model, zamiast go wzmocnić. Jeżeli jednocześnie osoba tworząca model jest oddzielona od procesu jego implementacji, wtedy nigdy nie zdobędzie wiedzy o jej ograniczeniach lub ją szybko utraci.



Modelowanie praktyczne



Jeżeli podział pracy nie będzie pozwalać na współpracę umożliwiającą przekazywanie subtelności *PROJEKTOWANIA STEROWANEGO MODELEM*, wtedy wiedza oraz umiejętności doświadczonych projektantów nie zostaną przekazane programistom.





Modelowanie praktyczne



W *PROJEKTOWANIU STEROWANYM MODELEM* fragment kodu stanowi wyrażenie modelu — zmiana kodu zmienia również model.

Czy im się to podoba, czy nie, programiści także tworzą model.

Lepiej więc od początku skonfigurować projekt w ten sposób, aby programiści mogli dobrze wykonać to zadanie modelowania.





Modelowanie praktyczne



Każda osoba techniczna mająca wkład do modelu musi spędzić pewną ilość czasu z kodem, jakkolwiek wysokopoziomowej roli w projekcie by ją pełniła.

Każdy, kto jest odpowiedzialny za zmianę kodu, powinien nauczyć się, w jaki sposób wyrażać model w kodzie.

Każdy programista musi być na pewnym poziomie zaangażowany w dyskusję na temat modelu i mieć kontakt z ekspertami dziedzinowymi.

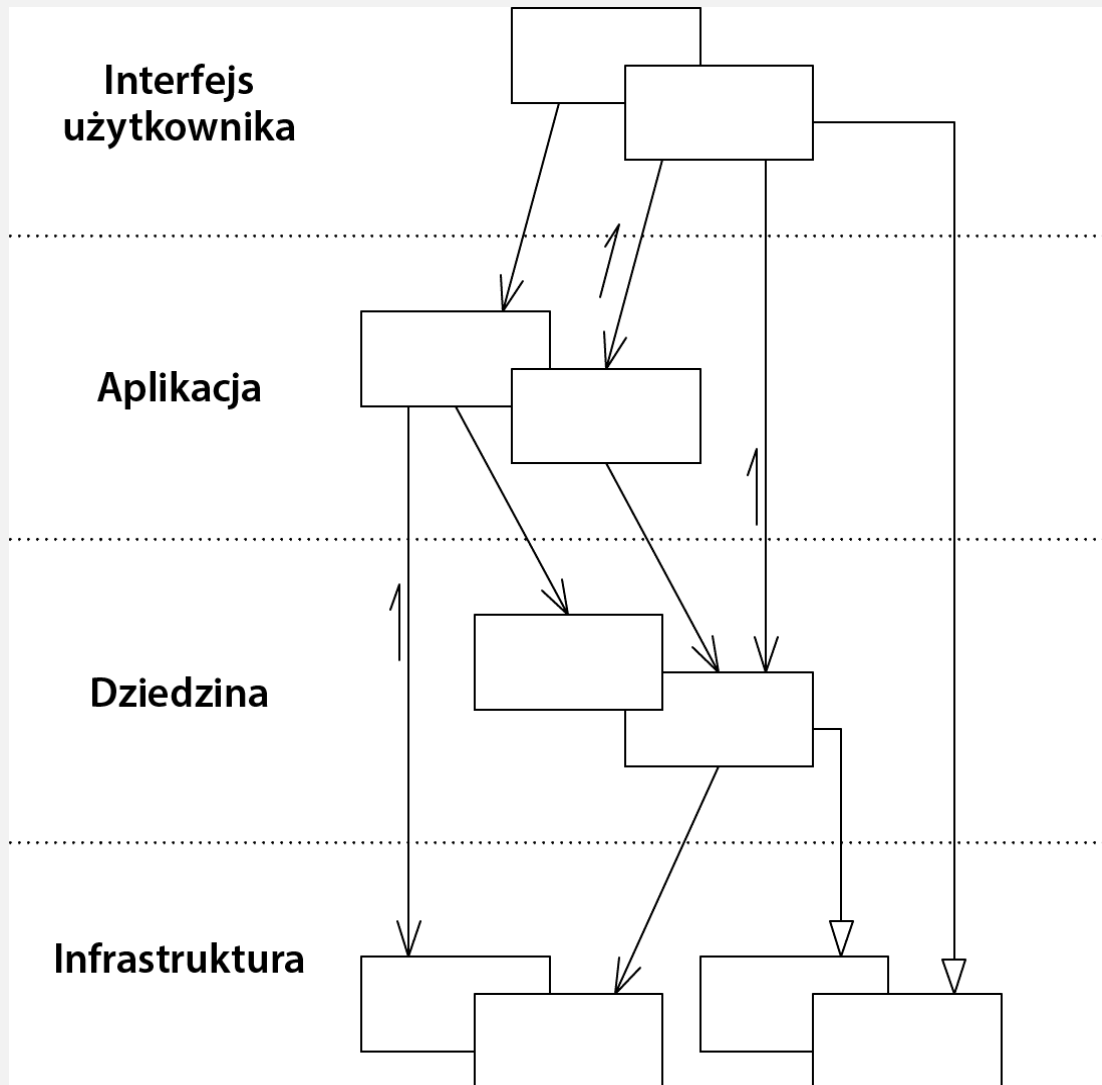
Wszystkie osoby wnoszące na różne sposoby wkład w projekt muszą świadomie zachęcać tych, którzy bezpośrednio dotyczą kodu, do dynamicznej wymiany zagadnień z modelem przy użyciu *JĘZYKA WSZECHOBECNEGO*.



Wykład 3

Wyizolowanie dziedziny i wyrażenie modelu w programie





Musimy oddzielić obiekty dziedziny od innych funkcji systemu, aby nie mieszać pojęć dziedzinowych z innymi, związanymi jedynie z technologią oprogramowania. W gąszczu całego systemu nie możemy też stracić z oczu samej dziedziny.

Architektura warstwowa

Warstwa interfejsu użytkownika (lub prezentacji)	Odpowiedzialna za wyświetlanie informacji dla użytkownika oraz interpretację jego poleceń. Zamiast użytkownika zewnętrzny aktor może być niekiedy innym systemem komputerowym.
Warstwa aplikacji	Definiuje zamierzone zadania programu i steruje obiektami dziedziny w celu rozwiązania postawionych przed nimi zadań. Zadania, za które jest odpowiedzialna ta warstwa, są znaczące dla biznesu i wymagane w celu poprawnego współdziałania z warstwami aplikacji innych systemów. Warstwa jest utrzymywana w zwartej postaci. Nie zawiera reguł ani wiedzy biznesowej, a jedynie zarządza zadaniami i deleguje je do rozwiązania przez obiekty dziedziny w kolejnej warstwie niżej. Warstwa nie musi odzwierciedlać konkretnej sytuacji biznesowej, lecz może posiadać stan, ukazujący postęp zadania użytkownikowi lub programowi.

Chociaż istnieje wiele typów warstw, to jednak najbardziej skuteczne architektury używają pewnych odmian następujących czterech warstw pojęciowych:

**Warstwa dziedziny
(lub modelu)**

Warstwa odpowiedzialna za reprezentację zagadnień biznesowych, informacji o sytuacji biznesowej oraz reguł biznesowych. W tym miejscu jest przechowywany oraz używany stan odzwierciedlający sytuację biznesową, chociaż szczegóły techniczne związane z przechowywaniem tego stanu są oddelegowane do warstwy infrastruktury. *Ta warstwa stanowi istotę programu od strony biznesowej.*

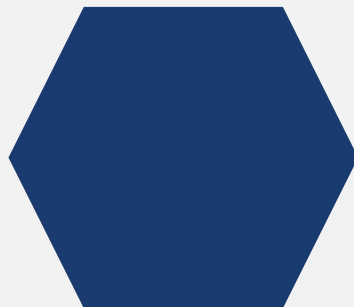
**Warstwa
infrastruktury**

Udostępnia podstawowe możliwości techniczne wspierające warstwy wyższe, tj. komunikaty wysyłane do aplikacji, zachowywanie dziedziny, rysowanie elementów interfejsu użytkownika itp. Warstwa infrastruktury może również poprzez szkielet architektury wspierać wzorzec interakcji pomiędzy wszystkimi czterema warstwami.

Chociaż istnieje wiele typów warstw, to jednak najbardziej skuteczne architektury używają pewnych odmian następujących czterech warstw pojęciowych:



W celu poprawnego wyizolowania dziedziny:



Podziel złożony program na warstwy.

Utwórz projekt każdej z nich, czyniąc ją zarazem spójną, jak i zależną jedynie od warstw położonych niżej.

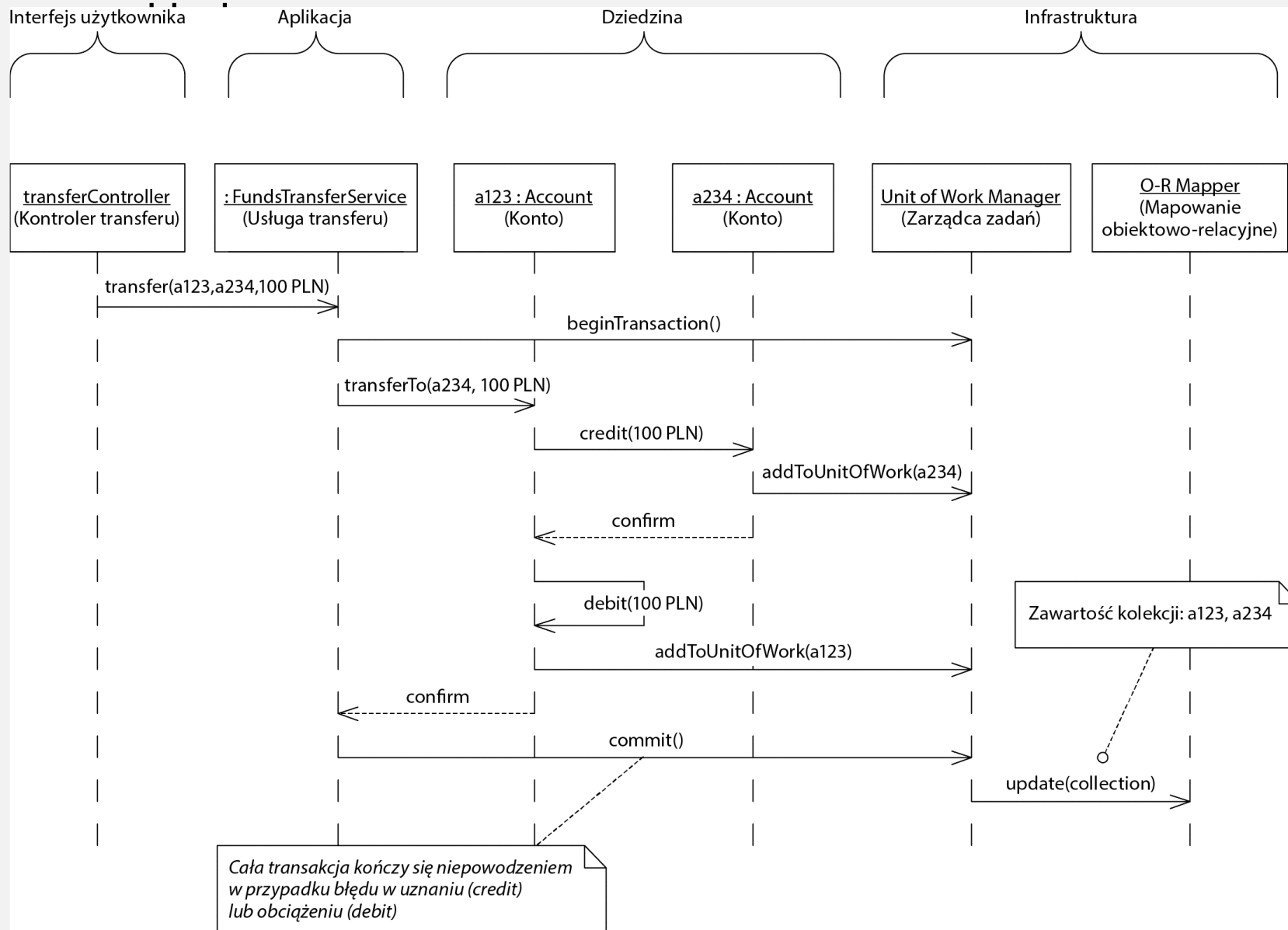
Aby utrzymać luźne powiązanie z wyższymi warstwami, stosuj standardowe wzorce architektoniczne.

Umieść kod związany z modelem dziedziny w pojedynczej warstwie i odizoluj go od kodu interfejsu użytkownika, aplikacji oraz infrastruktury.

Dzięki takiemu podejściu obiekty dziedziny, pozbawione konieczności obsługi wyświetlania, przechowywania, zarządzania zadaniami aplikacji itd., mogą skoncentrować się jedynie na wyrażeniu modelu dziedziny.

To umożliwia wzbogacenie i uproszczenie modelu na tyle, aby mógł on wychwycić wiedzę biznesową i wykorzystać ją w praktyce.

Podział na warstwy w przypadku funkcjonalności internetowej systemu bankowego -



Aplikacja udostępnia szereg funkcjonalności zarządzania kontami bankowymi.

Jedną z nich jest przelew środków, gdzie użytkownik podaje numery dwóch rachunków oraz kwotę, a następnie zatwierdza przelew.



Nawet jeżeli zespół będzie mieć więcej czasu, to bez pomocy eksperta członkowie zespołu prawdopodobnie nie dadzą rady opanować wymaganych technik. Jeżeli nawet na koniec pokonają te wszystkie przeciwności, to i tak w końcu utworzą prosty system, w którym nigdy nie będą wymagane bogate funkcjonalności.



W przypadku, gdy niedoświadczony zespół pracujący nad prostym projektem zdecyduje się wypróbować **PROJEKTOWANIE STEROWANE MODELEM Z ARCHITEKTURĄ WARSTWOWĄ**, stanie przed koniecznością podjęcia szybkiej i wymagającej nauki. Członkowie zespołu będą zmuszeni opanować zaawansowane nowe technologie oraz dać sobie radę z nauką modelowania obiektowego

Narzut spowodowany przez zarządzanie infrastrukturą oraz wszystkimi warstwami spowoduje wydłużenie prostych zadań.

Proste projekty mają zazwyczaj krótkie terminy i średnio skomplikowane oczekiwania.

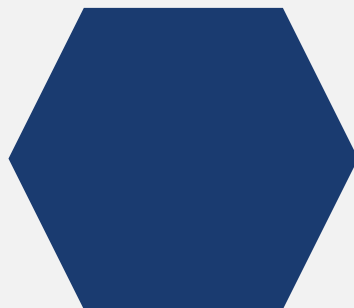
Projekt zostanie z pewnością przerwany, na długo zanim zespół dokończy przypisane doń zadania, nie mając szans na zaprezentowanie wspaniałych możliwości zastosowanego podejścia.



Wykład 4

Wyrażenie modelu
w programie



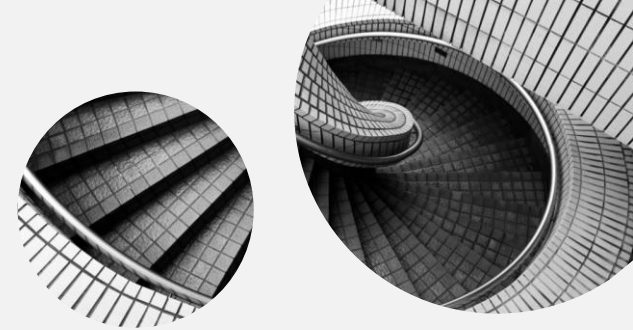


W sytuacji, gdy obiekt jest wyróżniany przez swoją tożsamość, a nie swoje atrybuty, umieść to jako podstawowe założenie jego definicji w modelu. Utrzymuj definicję klasy w prostej postaci, koncentrując się na ciągłości w jej cyklu życia oraz jej tożsamości. Określ sposoby rozróżnienia każdego z obiektów niezależnie od jego postaci lub historii. Uważaj na wymagania dotyczące porównywania obiektów przy użyciu ich atrybutów. Dla każdego z tych obiektów zdefiniuj operację, która gwarantuje zwrócenie niepowtarzalnego wyniku. Możesz to zapewnić, dołączając symbol o gwarantowanej niepowtarzalności. Oznacza to, że identyfikator obiektu może być nadawany zewnętrznemu lub też utworzony specjalnie dla systemu. W obu przypadkach musi on zapewniać rozróżnienie tożsamości w modelu. Model musi definiować, co oznacza, że dwie rzeczy są identyczne.


Śledzenie tożsamości ENCJI jest zasadnicze, jednak dołączanie jej do innych obiektów może znacząco obniżyć wydajność systemu, dodać pracy analitykom oraz zaciemnić model, sprawiając, że wszystkie obiekty będą wyglądać identycznie.

Projektowanie oprogramowania jest stałą walką ze złożonością. Musimy podejmować decyzje, aby takie specjalne traktowanie było stosowane jedynie tam, gdzie jest niezbędne.

Jeżeli jednak będziemy traktować tę nową kategorię obiektów jedynie poprzez pryzmat braku tożsamości, wtedy nie dodamy wiele do naszego słownika. W rzeczywistości WARTOŚCI (ang. value objects) mają w modelu swoją własną charakterystykę i swoje własne znaczenie. To właśnie te obiekty opisują rzeczy.



W sytuacji, gdy zależy Ci tylko na atrybutach danego elementu modelu, zakwalifikuj je do *WARTOŚCI*. Pozwól im wyrażać znaczenie atrybutów, które przechowują, i dodaj im wymaganą funkcjonalność. Traktuj *WARTOŚCI* jako niezmiennie. Nie nadawaj im tożsamości i unikaj złożoności projektowych wymaganych w *ENCJACH*.



Współdzielenie najbardziej przydaje się w wymienionych poniżej sytuacjach, w których ma ono największą wartość i sprawia najmniej problemów:

- **Gdy krytyczne staje się ograniczenie zajmowanej przestrzeni lub liczby obiektów w bazie danych.**
- **Gdy narzut komunikacyjny jest niewielki (na przykład w architekturze ze scentralizowanym serwerem).**
- **Gdy współdzielony obiekt jest niezmienny.**



Specjalne przypadki: Kiedy dopuszczać zmiany?

Niezmiennosc jest wspaniałym uproszczeniem implementacji, pozwalającym na bezpieczne współdzielenie oraz przekazywanie referencji. Jest również spójna ze znaczeniem wartości. Jeżeli wartość atrybutu ulegnie zmianie, wtedy używasz innej WARTOŚCI, zamiast zmieniać już istniejącą. Pomimo to istnieją przypadki, w których przez wzgląd na wydajność implementacji faworyzowane byłoby podejście umożliwiające zmianę WARTOŚCI. Poniższe czynniki przeważałyby na korzyść dopuszczalności zmian:


- Jeżeli WARTOŚĆ ulega częstym zmianom.
- Jeżeli tworzenie lub usuwanie obiektu jest kosztowne.
- Jeżeli zastąpienie (zamiast modyfikacji) obiektu naruszyłoby grupę obiektów
- Jeżeli WARTOŚCI nie są zbyt wiele razy współdzielone lub można z tego zrezygnować w celu usprawnienia grupowania lub z innych przyczyn



Wykład 5

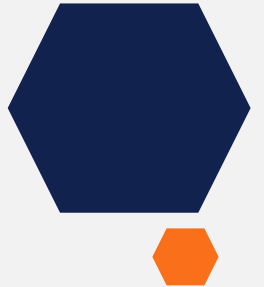
Wyrażenie modelu
w programie





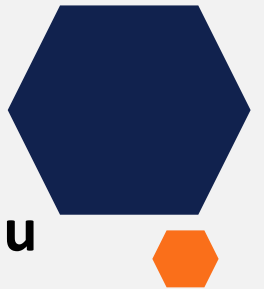
W przypadku, gdy niedoświadczony zespół pracujący nad prostym projektem zdecyduje się wypróbować *PROJEKTOWANIE STEROWANE MODELEM z ARCHITEKTURĄ WARSTWOWĄ*, stanie przed koniecznością podjęcia szybkiej i wymagającej nauki. Członkowie zespołu będą zmuszeni opanować zaawansowane nowe technologie oraz dać sobie radę z nauką modelowania obiektowego

Umieść całą logikę biznesową w interfejsie użytkownika. Podziel aplikację na małe funkcje i zaimplementuj je jako oddzielne moduły interfejsu, umieszczając w nich reguły biznesowe. Wykorzystaj bazę danych w charakterze współdzielonego repozytorium danych. Użyj najbardziej zautomatyzowanych narzędzi do tworzenia interfejsu użytkownika oraz programowania wizualnego, jakie tylko są dostępne na rynku.



Zalety:

- Dostępna od razu duża wydajność dla prostych aplikacji.
- Mniej doświadczeni programiści mogą używać tego wzorca po pobieżnym przeszkoleniu.
- Nawet braki w wymaganiach mogą zostać rozwiązane po udostępnieniu prototypu użytkownikom, a następnie szybkim zaadaptowaniu go do ich potrzeb.
- Aplikacje są od siebie oddzielone, więc harmonogram dostarczania małych modułów może być opracowany z dość dużą dokładnością. Rozszerzenie systemu o dodatkowe proste funkcjonalności może być łatwe.
- Z tym wzorcem dobrze współpracują bazy danych, które umożliwiają integrację na poziomie danych.
- Dobra współpraca z narzędziami 4GL.
- Po udostępnieniu aplikacji osoby ją utrzymujące będą w stanie szybko zmienić jej części, nawet te, których nie będą w stanie zrozumieć. Ewentualne efekty zmian będą mieć tylko lokalny wpływ na konkretny fragment interfejsu użytkownika.



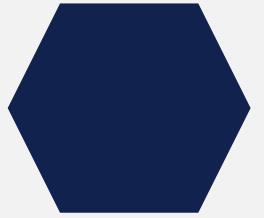
Wady:

- Integracja aplikacji jest trudna, o ile nie jest zrobiona poprzez bazę danych.
- Nie występuje współdzielenie kodu oraz nie ma żadnego modelu abstrakcyjnego logiki biznesowej. Reguły biznesowe muszą być duplikowane w każdej operacji, której dotyczą.
- Szybkie prototypowanie oraz przeróbki kodu mają swoje naturalne ograniczenia, ponieważ brak abstrakcji dziedziny ogranicza możliwości zmian.
- Złożoność szybko okaże się ograniczeniem nie do przezwyciężenia. Dlatego naturalną ścieżką rozwoju będzie tworzenie kolejnych prostych aplikacji. Nie ma prostej recepty na wzbogacanie istniejącej logiki.

Encje..encje..encje

Niektóre obiekty nie są określane jedynie przez swoje atrybuty. Reprezentują one tożsamość stałą w czasie, a bardzo często również w różnych sposobach ich reprezentacji. Niekiedy tego rodzaju obiekt musi być porównywany z innym, nawet jeżeli jego atrybuty różnią się między sobą. Co więcej, musi również istnieć sposób na odróżnienie obiektu spośród wielu obiektów o tych samych atrybutach. Pomyłka w tożsamości może prowadzić do uszkodzenia danych.

Encje...cd



W sytuacji, gdy zależy Ci tylko na atrybutach danego elementu modelu, zakwalifikuj je do WARTOŚCI. Pozwól im wyrażać znaczenie atrybutów, które przechowują, i dodaj im wymaganą funkcjonalność. Traktuj WARTOŚCI jako niezmiennie. Nie nadawaj im tożsamości i unikaj złożoności projektowych wymaganych w ENCJACH.

Współdzielenie..kiedy najlepiej

Współdzielenie najbardziej przydaje się w wymienionych poniżej sytuacjach, w których ma ono największą wartość i sprawia najmniej problemów:

- Gdy krytyczne staje się ograniczenie zajmowanej przestrzeni lub liczby obiektów w bazie danych.
- Gdy narzut komunikacyjny jest niewielki (na przykład w architekturze ze scentralizowanym serwerem).
- Gdy współdzielony obiekt jest niezmienny.

Modelowanie WARTOŚCI



Niektóre pojęcia z dziedziny nie dają się w sposób naturalny zamodelować jako obiekty. Usiłowanie zdefiniowania danej wymaganej funkcjonalności dziedzinowej w charakterze odpowiedzialności ENCJI lub WARTOŚCI może skutkować wypaczeniem definicji obiektów opartych na modelu lub też dodaniem pozbawionych znaczenia sztucznych obiektów.



Wykład 6

Moduły – budowa systemu



MODUŁY (zwane również PAKIETAMI)

- *MODUŁY* (ang. *modules*) stanowią stary i dobrze poznany element projektowy. Istnieje wiele technicznych powodów ich stosowania, jednak podstawową przyczyną ich użycia jest zazwyczaj przeciążenie rozmiarem i złożonością kodu, uniemożliwiające jego dobre zrozumienie. *MODUŁY* oferują dwa widoki modelu. Mogą być użyte do spojrzenia na szczegóły funkcjonalności zawartej w *MODULE* bez konieczności analizy reszty systemu lub też służyć do analizy relacji pomiędzy *MODUŁAMI* bez wnikania w ich szczegóły.
- W warstwie dziedziny *MODUŁY* powinny pojawiać się jako istotne części modelu, przekazujące pewną historię z dziedziny ujętą w większej skali.

Moduły...

Wszyscy stosują *MODUŁY*, lecz tylko nieliczni traktują je jako pełnoprawną część modelu. W trakcie trwania projektu kod jest zazwyczaj dzielony na różnego rodzaju kategorie, począwszy od wydzielenia różnych technicznych aspektów architektury, a skończywszy na zadaniach przypisanych do poszczególnych programistów. Fakt podziału kodu na *MODUŁY* we wczesnej fazie projektu jest nawet dobrze postrzegany przez samych programistów.

Moduły...

Stwierdzenie, że *MODUŁY* powinny być bardzo spójne i słabo ze sobą związane, jest naturalnie truizmem. Wprowadzie definicje związania i spójności elementów modelu wydają się brzmieć jak techniczne metryki, oceniane jedynie według podziału i rozmieszczenia asocjacji oraz interakcji pomiędzy obiektami, jednak należy mieć na uwadze, iż to nie sam kod jest dzielony na *MODUŁY*, a pojęcia dziedziny. Istnieje naturalna granica, definiująca liczbę rzeczy, o których człowiek może myśleć w danym momencie (stąd bierze się słabe związanie), a niespójne fragmenty różnych zagadnień są trudne do zrozumienia pośród innych (stąd duża spójność).

Moduły...

- Wybierz *MODUŁY*, które przekazują pewną historię o systemie i zawierają spójny zestaw pojęć. Takie podejście często skutkuje słabym związaniem pomiędzy *MODUŁAMI*. Jeżeli jednak tak się nie stanie, poszukaj sposobów na zmianę modelu w taki sposób, aby rozdzielić pojęcia, lub też poszukaj przeoczonego zagadnienia, które mogłoby stanowić podstawę *MODUŁU* w istotny sposób łączącego ze sobą jego elementy. Poszukaj słabego związania, biorąc pod uwagę pojęcia, które mogą być zrozumiane lub uzasadnione niezależnie od siebie. Modyfikuj model, dopóki nie będzie on, wraz z odpowiadającym mu kodem, podzielony w sposób zgodny z pojęciami dziedziny wysokiego poziomu.
- Nadaj *MODUŁOM* nazwy, które staną się częścią *JĘZYKA WSZECHOBECNEGO*. *MODUŁY* wraz ze swoimi nazwami powinny odzwierciedlać wiedzę dziedzinową.

Obiekty i nie obiekty w programowaniu

Podstawowe reguły łączenia nieobiektowych elementów z systemem obiekтовым:

- *Nie walcz z paradygmatem implementacji.* Zawsze istnieje inny sposób myślenia o dziedzinie. Spróbuj znaleźć pojęcia modelu pasujące do tego paradygmatu.
- *Pochyl się nad językiem wszechobecnym.* Nawet w sytuacji, gdy nie istnieje ścisłe połączenie pomiędzy narzędziami, spójne użycie języka może zapobiec rozejściu się elementów projektu.

Obiekty i nie obiekty w programowaniu

- *Nie polegaj jedynie na języku UML.* Niekiedy przywiązanie do jednego narzędzia, jak na przykład diagramy UML, skłania ludzi do zepsucia modelu tylko po to, aby pasował do postaci łatwej do narysowania. UML na przykład posiada funkcjonalności pozwalające na odwzorowanie ograniczeń, jednak nie zawsze są one wystarczające. Niekiedy lepszym wyjściem od adaptowania na siłę stylu rysowania zaprojektowanego do pewnego rodzaju obiektów będzie użycie innego stylu (być może bardziej pasującego do innych paradygmatów) lub nawet prostych opisów słownych w języku.
- *Bądź sceptyczny.* Czy narzędzie jest rzeczywiście warte uwagi? Na przykład fakt, że silnik reguł oferuje pewne reguły, nie oznacza automatycznie, że będą one warte nakładu pracy, który wprowadza jego implementacja. Reguły mogą być wyrażone w postaci obiektów. Być może będzie to nieco mniej schludne, lecz i tak użycie wielu paradygmatów niesamowicie komplikuje sprawy.

Podsumowanie

Był to przegląd wiedzy niezbędny do zaliczenia egzaminu z przedmiotu

M.Majer

