POLITECHNIKA WARSZAWSKA
Wydział Elektroniki i Technik Informacyjnych
Instytut Informatyki

Rok akademicki 2013/2014

PRACA DYPLOMOWA MAGISTERSKA

Michał Aniserowicz

[TYTUŁ] (Generator aplikacji opartych o architekturę CQRS?)

Praca wykonana pod kierunkiem
dra inż. Jakuba Koperwasa

Ocena: .....

.....

Podpis Przewodniczącego Komisji
Egzaminu Dyplomowego

Spis treści

Rozdział 1

Wstęp

TODO
- problem - w projektach często występuje duplikacja w wielu miejscach - w CQRS to w ogóle

Rozdział 2

Duplikacja

Zasada "DRY" (ang. *"Don't Repeat Yourself"*), sformułowana przez Andrew Hunt'a i David'a Thomas'a, mówi: "Każda porcja wiedzy powinna mieć pojedynczą, jednoznaczną, autorytatywną reprezentację w systemie" [1]. Poprawne jej stosowanie skutkuje osiągnięciem stanu, w którym pojedyncza zmiana zachowania systemu wymaga modyfikacji tylko jednego fragmentu reprezentacji wiedzy. Należy podkreślić, że autorzy tej zasady przez "reprezentację wiedzy" rozumieją nie tylko kod źródłowy tworzony przez programistów systemu. Zaliczają do niej również dokumentację systemu, schemat bazy danych przez niego używanej, i inne artefakty powstające w procesie wytwarzania oprogramowania, takie jak scenariusze testów akceptacyjnych.

Celem stosowania tej zasady jest unikanie duplikacji wiedzy zawartej w systemie.

2.1 Rodzaje duplikacji

TODO: Opisać co jest duplikacją, a co nie (użycie odnośnika (np. nazwy tabeli) w wielu miejscach nie jest). Odniesinie do Pragmatic Programmer.

Wyróżnia się cztery rodzaje duplikacji ze względu na przyczynę jej powstania [1]:

- duplikacja wymuszona (ang. *imposed duplication*) - pojawia się gdy programista świadomie duplikuje kod aplikacji uznając, że w danej sytuacji jest to nie do uniknięcia;
- duplikacja niezamierzona (ang. *inadvertent duplication*) - występuje, gry programista nie jest świadomy, że doprowadza do duplikacji;
- duplikacja niecierpliwa (ang. *impatient duplication*) - jest wynikiem niedbalstwa programisty; ma miejsce w sytuacji, gdy programista świadomie wybierze rozwiązanie najprostsze w danej sytuacji;
- duplikacja pomiędzy programistami (ang. *interdeveloper duplication*) - pojawia się, gdy kilku programistów tworzących tę samą aplikację wzajemnie duplikuje tworzony kod.

Duplikacja może występować w różnych postaciach:

2.1.1 Duplikacja czynności wykonywanych przez programistów

TODO: Ręczny update bazy (odpalanie po kolei skryptów sql), deploy na staging lub produkcję (wrzucanie plików przez ftp).

2.1.2 Duplikacja kodu źródłowego aplikacji

Duplikację kodu źródłowego aplikacji można podzielić na następujące kategorie [2]:

- duplikacja wyrażeń (ang. *literal duplication*) - najprostszy rodzaj duplikacji; obejmuje fragmenty kodu, których treść jest identyczna bądź różni się jedynie typami danych, na których te fragmenty operują (przykład: dwie identyczne metody umiejscowione w różnych modułach systemu);
- duplikacja strukturalna (ang. *structural duplication*) - obejmuje fragmenty kodu, które mają ten sam schemat działania, ale różnią się pojedynczymi instrukcjami (przykład: dwie pętle iterurąjce po tej samej kolekcji z tym samym warunkiem stopu, ale wykonujące różne operacje na elementach kolekcji);
- duplikacja czasowa (ang. *temporal duplication*) - określa fragment kodu, który jest niepotrzebnie wykonynway wiele razy (przykład: zliczenie elementów kolekcji podczas każdego sprawdzenia warunku stopu pętli iterurąjcej po tej kolekcji);
- duplikacja intencji (ang. *duplication of intent*) - występuje, gdy dwóch programistów umieści w różnych modułach aplikacji dwa fragmenty, których wynik działania (ale niekoniecznie treść) jest identyczny.

2.1.3 Duplikacja opisu dziedziny aplikacji

W każdej aplikacji obiektowej korzystającej z bazy danych występuje duplikacja opisu dziedziny aplikacji. Występuje ona w co najmniej dwóch miejsach:

1. schemacie bazy danych (DDL),
2. definicjach klas w kodzie źródłowym aplikacji.

Kolejnym typowym miejscem, w którym umieszcza się informacje o dziedzinie aplikacji jest jej dokumentacja.

W miarę jak rozrasta się projekt informatyczny, pojawia się tendencja do duplikowania fragmentów dziedziny poruszanego przez niego problemu. Duplikacja ta rozprzestrzenia się pomiędzy modułami aplikacji, których zadaniem obróbka tych samych danych, ale w różny sposób. Przykładowo, aplikacja może udostępniać przechowywane dane na następujące sposoby:

- wyświetlać je na stronie WWW,
- wystawiać jako API,
- eksportować do arkusza kalkulacyjnego.

Jeśli moduły te posiadają osobne implementacje dziedziny aplikacji, to każda zmiana dziedziny wymaga zmodyfikowania implementacji dziedziny w każdym z modułów - co wiąże się z dużymi kosztami.

Z drugiej strony, identyczność zestawu danych udostępnianego przez różne moduły może nie być pożądana. Przykładowo, na stronie WWW wyświetlane mogą być jedynie podstawowe dane danej encji, podczas gdy pełne dane dostępne są po wyeksportowaniu arkusza kalkulacyjnego. Takie wymaganie wymusza duplikację części dziedziny aplikacji pomiędzy różnymi implementacjami tej dziedziny.

TODO: Sformułować konkretny przykład aplikacji.

2.2 Skutki występowania duplikacji

Pojawinie się duplikacji w systemie ma zazwyczaj szkodliwe skutki. Rozprzestrzenia się ona tym szybciej, a wyrządzane szkody są tym dotkliwsze, im więszy jest rozmiar systemu.

Najbardziej oczywistym skutkiem występowania duplikacji jest wydłużenie się czasu poświęcanego przez programistów na nieskomplikowane lub powtarzalne zadania. Przykładowo, nieznajomość kodu współdzielonego przez wszystkie moduły systemu (tzw. rdzenia, ang. *core*) powoduje, że programiści niepotrzebnie spędzają czas na implementowaniu podstawowych funcji, które są już dostępne w rdzeniu systemu. Innym przykładem może być praktyka ręcznego wykonywania czynności, które mogą być łatwo zautomatyzowane. Procedura aktualizacji schematu bazy danych, na którą składa się wykonanie kilku skryptów DDL zajmnie dużo mniej czasu, jeśli wykona ją skrypt. Wrzucenie nowej wersji systemu na staging lub produkcje nie musi wiązać się z ręcznym wrzucaniem plików na ftp zdalnej maszyny - może dziać się automatycznie (ref continous delivery).

TODO: Nie podawać tu rozwiązań, skupić się na szkodach.

Po drugie, zmiana zachowania systemu wymaga modyfikacji wielu modułów systemu. Jeśli nie jest nigdzie zapisane, jakich - trzeba wszystkie miejsca zlokalizować ręcznie. Najbardziej jest to widoczne przy naprawie błędów: jeśli występuje duplikacja kodu, to programista naprawiający błąd powinien mieć swiadomość, że poprawiany przez niego fragment kodu może być powielony w wielu miejscach w systemie. Wszystkie te miejsca musi zlokalizować i poprawić.

Po trzecie, rozmiar systemu - czytelność kodu (jest rozwleczony) - rozmiar plików

Po czwarte, pojawienie się jednej dupliakcji to przyzwolenie na wprowadzenie innej

TODO: Czemu skupiam się na dziedzinie aplikacji - wydaje się bardziej kosztowna niż duplikacja czynności, a łatwiejsza do usunięcia niż duplikacja kodu. Ale o tym może juz po rozwiązaniach..?

2.3 Rozwiązanie: skrypty na wszystkie czynności

na update bazy, na deploy na stg / prod

2.4 Rozwiązanie: generyczna implementacja

2.4.1 Refleksja

TODO: Czy generatory to jedyne rozwiązanie? Duplikacji pozwala też uniknąć refleksja.

2.4.2 Serializacja (do JSONA)

TODO

2.4.3 Inne generyczne rozwiązania

TODO

Zauważmy jednak, że przedstawione propozycje rozwiązują jedynie problem duplikacji w logice biznesowej, podczas gdy najwięcej rodzajów duplikacji dotyczy dziedziny aplikacji.

2.5 Rozwiązanie: użycie licznych generatorów

2.5.1 Generator code-first / database-first

Rozwiązaniem problemu może być osiągnięcie implementacji, w której pełna dziedzina aplikacji definiowana jest w jednym miejscu, a jej implementacje są generowane automatycznie w odpowiednich modułach. Taka sytuacja sprawia, że duplikacja pomiędzy modułami przestaje być problemem - aby wprowadzić zmiany we wszystkich implementacjach, wystarczy wprowadzic pojedynczą modyfikację w definicji dziedziny aplikacji.

Najprostszym przykładem realizującym to rozwiązanie jest użycie generatora definicji klas na podstawie schematu bazy danych (tzw. podejście database-first) lub generatora schematu bazy danych na podstawie definicji klas (tzw. podejście code-first). Przykładami takich generatorów są:

- Hibernate,
- EntityFramework,
- LinqToSQL.

TODO: Wymienić więcej, dla różnych technolgii, dać przypisy.

Jednakże takie generatory eliminują tylko jeden rodzaj duplikacji, wymieniony na początku rozdziału.

2.5.2 Generator dokumentacji

Duplikacji w dokumentacji aplikacji można uniknąć poprzez zastosowanie narzędzi generujących tę dokumentację na podstawie kodu źródłowego aplikacji. Narzędzia takie pozwalają na wygenerowanie dokumentacji w kilku formatach, takich jak PDF czy HTML. Ich przykłady to:

- JavaDoc,
- ...

TODO: Wymienić więcej, dla różnych technolgii, dać przypisy.

W przypadku tych narzędzi występuje ten sam problem, co w przypadku generatorów schematu bazy danych lub definicji klas - eliminują one tylko jeden rodzaj duplikacji.

2.5.3 Inne generatory

Aby uniknąć pozostałych rodzajów duplikacji:

- duplikowania nazw i typów kolumn tabel w definicjach widoków bazy danych,
- duplikowania nazw i typów pól klas pomiędzy implementacjami dziedziny aplikacji w różnych jej modułach,

należałoby skorzystać z kolejnych generatorów.

Warto zauważyć, że różne typy generatorów za źródło danych obierają sobie różne definicje dziedziny: np. generator database-first bazuje na języku DDL, a generator dokumentacji - na kodzie źródłowym aplikacji. Używanie wielu różnych generatorów usuwających pojedyncze rozdzaje duplikacji w końcu doprowadziłoby do powstania "łańcucha" generatów, w którym wynik działania jedngo generatora byłby źródłem danych dla innego. Takie rozwiązanie może być bardzo trudne w utrzymaniu.

2.6 Rozwiązanie: jeden generator wszystkiego

Odpowiednim rozwiązaniem wydaje się być zastosowanie pojedynczego generatora potrafiącego wygenerować wszystkie potrzebne artefakty. Powstanie takiego generatora dającego się zastosować w każdym projekcie jest jednak bardzo mało prawdopodobne:

- każdy projekt ma inne wymagania odnośnie wygenerowanych artefaktów,
- prawdopodobne nie istnieje format, który pozwalałby zdefiniować każdą dziedzinę w najlepszy (najbardziej naturalny) dla niej sposób.

Rozdział 3

Implementacja

Celem niniejszej pracy jest próba stworzenia takiego generatora. Założenia funkcjonalne:

- generator ma być na tyle elastyczny, aby móc obsłużyć wiele sposobów zdefiniowania dziedziny aplikacji,
- generator powinien pozwalać na wygenerowanie dowolnych plików tekstowych (skryptów SQL, kodu źródłowego, dokumentancji HTML itd.),
- generator nie musi generować logiki biznesowej aplikacji - wystarczy dziedzina

Założenia niefunkcjonalne:

- generator zostanie stworzony w technolgii .NET Framework,
- ...

Szczególna uwaga zostanie poświęcona aplikacjom opartym o architekturę CQRS i wykorzystującym bazy danych typu NoSQL. Specyficzną cechą takich aplikacji jest to, że operują one na modelach o wysokim stopniu denormalizacji, co wiąże się z masowo występującą duplikacją danych.

TODO: Opisać CQRS. TODO: Opisać Event Sourcing. TODO: Opisac NoSQL. TODO: Opisać rodzaje baz NoSQL. TODO: Wybrać bazę NoSQL i dlaczego Cassandra. TODO: Opisać Cassandrę.

Bibliografia

[1] *The Pragmatic Programmer.* Hunt A., Thomas D. Addison-Wesley. Westford 2013. ISBN 0-201-61622-X.

[2] *Duplication in Software* [online], Just Software Solutions, http://www.justsoftwaresolutions.co.uk/design/duplication.html [dostęp: lipiec 2014].

Oświadczenie

Oświadczam, że Pracę Dyplomową pod tytułem *"[TYTUŁ]"*, którą kierował dr inż. Jakub Koperwas, wykonałem samodzielnie, co poświadczam własnoręcznym podpisem.

.....

Michał Aniserowicz