O wpływie mikroserwisów na proces wytwarzania oprogramowania

Od 5 lat można obserwować coraz większe zainteresowanie wielu firm architekturą opartą na mikroserwisach – trend ten nabiera coraz większego rozpędu. Od 50 lat znane jest nam prawo Conway'a, które jest trafną obserwacją na temat relacji pomiędzy wytworzonym oprogramowaniem a organizacją, która to oprogramowanie dostarcza. Jak w takim razie obserwacje sprzed 50 lat mają się do tego nowoczesnego sposobu tworzenia systemów IT?

MIKROSERWISY

Konferencja GeeCon 2013. Pamiętam, jakby to było wczoraj, kiedy Sam Newman, stojąc przed wielkim kinowym ekranem, opowiadał o „koncepcji" mikroserwisów. Słowo „koncepcja" dobrze oddaje wymiar tego, co wtedy się działo. Większość firm pracowała na monolitach, które były uruchomione we własnych serwerowniach. Oczywiście SOA – czyli architektura zorientowana na usługi – nie była czymś nowym. Można stwierdzić, że mikrousługi to kolejna wersja SOA, jednak cele i wyzwania, które rozwiązują te dwa podejścia, są zupełnie inne.

Mikroserwisy są jednym z popularniejszych sposobów do wytwarzania oprogramowania. Zaczynając budować rozproszony system dzisiaj, wiele elementów składowych infrastruktury lub narzędzi koniecznych do efektywnego uruchomienia takiego systemu jest dostępnych, co więcej, dostępnych jako rozwiązanie w chmurze, na wyciągnięcie ręki. Takich narzędzi nie było kilka lat temu, co pokazuje, że rozwiązanie jest popularne.

Osoby techniczne – w tym autor tego tekstu – uwielbiają rozwiązywać problemy techniczne. Przez ostatnie pięć lat wiele firm, wielu programistów pracowało właśnie nad różnymi problemami, jakie niosą ze sobą mikrousługi, np. problemy związane z uruchamianiem tysięcy instancji naszych usług, odkrywaniem usług wzajemnie, czyli Service Discovery, lub rozproszonym logowaniem, korelacją logów przez Trace-Id i wiele, wiele innych. Dzięki temu mamy do dyspozycji naprawdę dobre i specjalistyczne narzędzia. Co więcej, jestem przekonany, że za kilka lat będziemy używali zupełnie innych narzędzi do budowania mikrousług, niż dzisiaj.

Czy skupiając się tak bardzo na technologii, my, programiści, zauważyliśmy, jak zmieniło się nasze podejście do tworzenia architektury?

Wiele obietnic mikrousług sprzed kilku lat zostało zrealizowanych. Na przykład dzięki podzieleniu odpowiedzialności na mniejsze możemy przeprowadzać częściej wdrożenia, a awarie poszczególnych elementów nie skutkują awarią całości systemu. Podział na mniejsze fragmenty umożliwia eksperymentowanie tylko w części systemu, a same eksperymenty mogą dotyczyć nowych technologii, języków programowania, jak i funkcjonalności biznesowych.

PRAWO CONWAY'A

Artykuł doktora Melvin'a Conway'a pod tytułem „How do committees invent?"[1] został opublikowany ponad pół wieku temu, w kwietniowym wydaniu magazynu DataMotion z 1968 roku. Główna teza tego artykułu jest taka, że organizacje tworzące systemy (w szerokim znaczeniu tego słowa) są ograniczone przez swoją własną strukturę.

Doktor Conway zauważa, że im większa jest organizacja, tym łatwiej zaobserwować to zjawisko. Jest to ważne i kluczowe w naszym porównaniu z mikroserwisami, ponieważ mikroserwisy znajdują zastosowanie głównie w większych organizacjach.

Moje doświadczenia zawodowe potwierdzają obserwacje opisane w tym artykule. Jeżeli jakaś biznesowa funkcjonalność w systemie jest implementowana przez trzy zespoły developerskie, występuje ogromne prawdopodobieństwo, że powstaną trzy usługi realizujące tę funkcjonalność. Jeżeli realizacja nowej funkcjonalności jest powierzona jednemu zespołowi, to ten zespół będzie budował jeden „większy" mikroserwis.

Z czego to wynika? Istotne jest zrozumienie faktu, że pierwsze decyzje na temat architektury nowego rozwiązania zostają podjęte w momencie, w którym powstają zespoły projektowe lub jeżeli wytworzenie rozwiązania jest powierzone istniejącym już zespołom. Za każdym razem, kiedy następuje podział pracy na kilka zespołów, dane zadanie również skutkuje podziałem na kilka zespołów. Dzięki temu każdy z zespołów realizujących swoją część zadania jest w stanie realizować ją efektywnie. Następnie każdy z zespołów będzie dążył do ustalenia interfejsów komunikacji z innymi zespołami, co w konsekwencji przełoży się na interfejsy komunikacji poszczególnych modułów architektury, przez co organizacja zespołów zostanie trwale odzwierciedlona w architekturze danego rozwiązania.

Założenia doktora Conway'a były poparte jedynie obserwacjami związanymi z projektami militarnymi. Nie zostały przeprowadzone dokładne badania, które mogłyby w statystycznie istnotny sposób potwierdzić tę teorię. Kolejne, bardziej precyzyjne badania ukazały się dopiero w roku 2008. „Exploring the Duality between Product

1. http://www.melconway.com/Home/pdf/committees.pdf

and Organizational Architectures: A Test of the «Mirroring» Hypothesis"[2] porównuje zespoły projektowe:

» Pracujące ściśle – gdzie programiści pracują w dedykowanych zespołach, najczęściej w jednej lokalizacji i zatrudnionych przez jednego pracodawcę.
» Pracujące w sposób rozproszony – gdzie programistami są najczęściej wolontariusze z różnych firm, niekomunikujących się ze sobą bezpośrednio oraz są z różnych lokalizacji. Projekty to najczęściej produkty typu open-source.

Badacze zmierzyli kod źródłowy, wytworzony w tych dwóch różnych stylach i podejściach do wytwarzania oprogramowania.

Zespoły pracujące ściśle tworzą produkty, zoptymalizowane pod względem funkcjonalności i szybkości działania. Takie produkty w swojej architekturze mają znamiona rozwiązań monolitycznych, w których występuje dużo powiązań między poszczególnymi fragmentami systemu, a same pliki są większe i zawierają więcej funkcji.

Zespoły pracujące w sposób rozproszony muszą tworzyć produkty, których architektura jest modułowa i łatwo rozszerzalna. To stanowi o podstawie sukcesu takich projektów, ponieważ jest to skuteczny sposób, żeby zachęcić programistów do pracy nad takimi produktami. Dzięki temu poszczególne fragmenty systemu są ze sobą mniej związane, a same pliki są mniejsze i zawierają mniej funkcji.

Takie badania są silnym dowodem potwierdzającym hipotezę, że architektura produktu dąży do odzwierciedlenia struktury organizacji, w której jest wytwarzana.

MIKROSERWISY KONTRA PRAWO CONWAY'A

Dlaczego ważne jest mówienie o architekturze mikrousług w kontekście prawa Conway'a? Przyjmując za prawdziwe stwierdzenie, że architektura naszego systemu zależy od podziału na zespoły, musimy sobie szczerze odpowiedzieć na pytanie, kto tak naprawdę jest architektem naszego systemu?

Zapewne jest to jakaś osoba, która decyduje o kształcie zespołu, ale kto to może być? Być może jest to CTO firmy, być może są to menedżerowie, być może jest to ktoś z działu HR, być może są to liderzy poszczególnych zespołów, być może jest to grono starszych inżynierów oprogramowania, mających duży wpływ na organizację. Na to pytanie każda organizacja musi odpowiedzieć sobie sama.

Zobaczmy na poszczególnych przykładach, jak te dwie koncepcje przeplatają się ze sobą.

PRZYKŁAD: BUDOWANIE SYSTEMU OD ZERA

Zaczynasz budowę nowego systemu w pięcioosobowym zespole. Praca idzie świetnie, zespół szybko dostarcza funkcjonalności, co pozwala wybić się firmie na rynku. System zarabia na siebie, więc jest zapotrzebowanie na nowe funkcjonalności, obsługę większego ruchu itp.

Nagle zespół liczy już 15 osób. W tym momencie nie da się już efektywnie pracować. Musi nastąpić podział na dwa zespoły. Jakie firma ma możliwości:

» Nie zmieniamy architektury i budujemy dalej monolityczny system. Liczymy się z faktem konieczności wytworzenia procesów związanych z obsługą wspólnego kodu źródłowego. Zasady te dotyczą przeglądania kodu, wdrażania, testowania itp.
» Zmieniamy architekturę, dzieląc jeden system monolityczny na dwa. Dzięki temu każdy zespół ma własny kod źródłowy, który może niezależnie wdrażać, testować. Stosując dalej to podejście, staniemy w przyszłości w obliczu wielu wyzwań związanych z infrastrukturą i utrzymaniem wielu usług. Na starcie, kiedy mamy tylko dwie usługi, możemy nie dostrzegać problemów wynikających z posiadania setek usług.

Problemy obydwu rozwiązań będą się nasilały wraz z rosnącą liczbą osób pracujących nad produktem. Nie rekomenduję nikomu jednego czy drugiego podejścia. Budowanie monolitu czy mikrousług ma swoje plusy i minusy. Z pewnością w wielu miejscach dobrze wykonany monolit będzie lepszym i łatwiejszym w utrzymaniu rozwiązaniem. Jednak budowanie monolitu na siłę, kiedy nie ma ku temu przesłanek (np. wydajnościowych lub związanych z zasobami sprzętowymi), ma taką wadę, że nie będzie dostosowany do stylu pracy wielu zespołów, co nieustannie będzie generowało problemy logistyczne związane z pracą wielu osób nad jednym kodem źródłowym. Automatycznie pojawiają się problemy typu: „kto zepsuł builda?", lub „kto teraz robi wdrożenie na produkcję?". Wynika to z faktu, że model organizacji (czyli podział na kilka zespołów) nie jest dostosowany do architektury rozwiązania.

Będąc świadomym tego, jak może wyglądać praca z systemem w przyszłości, możemy planowo podejmować decyzje, jak powinniśmy kierować naszymi pracami rozwojowymi, tak by firma skutecznie mogła realizować wymagania biznesowe.

PRZYKŁAD: TWORZENIE ARCHITEKTURY

Jeżeli miałbym wskazać jedno najważniejsze prawo lub podejście w programowaniu, które jest istotne przy budowaniu systemów składających się z mikrousług, byłoby to „High in cohesion, low in coupling".

High in cohesion

Dlaczego ważne jest budowanie usług, które są wysoce spójne? Dzięki temu inżynierowie oprogramowania mogą szybko zrozumieć, co jest treścią, kluczową odpowiedzialnością danej usługi. Chodzi o zamykanie jednej i wyodrębnionej odpowiedzialności w konkretnej usłudze. Koncepcja prosta, stosowana praktycznie od początków programowania. Dziesiątki lat temu programiści zauważyli, że zamiast wielokrotnie używać słowa kluczowego GOTO, można wydzielić pewne fragmenty kodu do funkcji. Wraz z rozwojem technologii zasada ta pozostaje niezmienna, przekłada się tylko na inne określenia w różnych stylach programowania.

Low in coupling

Druga zasada dotyczy małego powiązania między usługami. Każdy z nas zapewne wielokrotnie doświadczył frustracji, kiedy musiał

2. https://www.hbs.edu/faculty/Publication%20Files/08-039_1861e507-1dc1-4602-85b8-90d71559d85b.pdf

zmienić jedną funkcjonalność, a zmiana ciągnęła za sobą zmianę w kilku miejscach. Jeżeli tak się dzieje, mówimy, że mamy w kodzie spaghetti, z tym że nie mamy tego na poziomie kodu źródłowego, a na poziomie mikroserwisów.

Te podstawowe zasady programowania przewijają się przez wiele języków programowania i są podstawą programowania obiektowego. Specjalista od DDD bez wahania stwierdzi, że te zasady są podstawą do wydzielania Bounded Contextów, czyli rozdzielnych domen w naszym systemie. Architektura oparta na tych założeniach będzie łatwo utrzymywalna i modyfikowalna w przyszłości.

Co ciekawe, bardzo łatwo jest sprawdzić, czy usługi są wysoce spójne i luźno powiązane. Wystarczy popatrzeć na częstotliwość i intensywność zmian w poszczególnych plikach. Jeżeli zmiany pojawiają się w dużej mierze w środku usługi, tam gdzie jest najwięcej logiki biznesowej, a zmiany są symboliczne na interfejsach między usługami – uwzględniając fakt, że mamy precyzyjne typy w interfejsach – prawdopodobnie mamy do czynienia z systemem, który jest silnie spójny i mało luźno powiązany między sobą.

Mówiąc o dzieleniu dużego systemu na małe konteksty, powinniśmy wiedzieć, jakiej wielkości powinna być nasza mikrousługa. Zapewne jeżeli masz w głowie jakąś definicję wielkości mikroserwisów, jest ona poprawna, przynajmniej częściowo. Kluczowe pytanie, jakie zadaje sobie wiele osób, brzmi: jakiej wielkości powinien być mikroserwis? Każdy przecież wie, że nie powinna być ani za duża, ani za mała. W jaki sposób można definiować poprawną wielkość mikroserwisów?

- Jeżeli bez problemu szybko da się zrozumieć wszystkie jej funkcjonalności.
- Jeżeli usługa nie ma więcej niż ileś linii kodu źródłowego (w zależności od języka programowania).
- Jeżeli da się stwierdzić, że łatwiej jest przepisać usługę niż ją refaktoryzować.

Wszystkie te stwierdzenia są poprawne, jednak żadna z nich nie daje nam pewności, że dobrze budujemy nasze usługi. Stosując zasadę „High in cohesion, low in coupling", jedne usługi mogą być mniejsze, inne trochę większe. Natomiast podział na usługi powinien wynikać z wymagań funkcjonalnych i niefunkcjonalnych naszego systemu.

Powyższe powinno nam uświadomić, że połączenia i komunikacja między usługami są bardzo ważnym aspektem tworzenia systemów rozproszonych. Z drugiej strony wiemy, że usługi będą powstawały w zależności od tego, jak zostaliśmy podzieleni na zespoły. Wiedząc, że kształt naszej architektury zależy od kształtu naszych zespołów, czy powinniśmy pozostawiać przypadkowi to, w jakich zespołach powinniśmy pracować? Już na etapie analizy wymagań osoby z doświadczeniem technicznym powinny mieć wpływ na kształt zespołów, który wynika z wymagań i założeń systemu.

Oczywiście może to nie być idealny kształt zespołów i będzie trzeba potem to zmieniać, żeby dostosować się do zmian w wymaganiach. Dzięki wstępnemu planowaniu będziemy w stanie uniknąć ogromnych błędów. W zaimplementowanym systemie zmiana powiązań i architektury mikroserwisów, kiedy system już powstał, nie należy do szybkich procesów, ponieważ będzie to się wiązało ze zmianami w wielu miejscach, a sam proces zmian kilku lub kilkunastu usług jest trudny, ponieważ wymaga najczęściej koordynacji kilku zespołów.

Ważne jest zrozumienie, że to właśnie poszczególne zespoły pracujące nad zadaniem będą tworzyły oddzielne regiony funkcjonalności. Niewłaściwy podział zadań między zespołami, lub niewłaściwy kształt zespołów, może doprowadzić do tego, że nasza architektura na tym ucierpi i będzie męką w rozwoju i utrzymaniu.

PRZYKŁAD: ROZWÓJ NOWYCH FUNKCJONALNOŚCI BIZNESOWYCH W ŚWIECIE MIKROSERWISÓW

Mając na uwadze fakt, że architektura zależy od kształtu naszych zespołów, jak w takim razie mogą wyglądać zmiany architektury mikrousług, które wynikają ze zmian organizacyjnych? Wyobraźmy sobie organizację, która tworzy system do obiegu dokumentów, a w dziale technologii pracuje kilka zespołów. Każdy zespół utrzymuje do 10 różnych usług.

Zespół A utrzymuje kilka mikrousług, które realizują następujące zadania:

- 4 usługi realizują proces importu dokumentów od partnerów biznesowych.
- 2 usługi realizują proces związany z drukowaniem dokumentów.
- 1 usługa realizuje proces związany z raportowaniem dokumentów.

Organizacja jest nastawiona na rozwój funkcjonalności związanych z drukowaniem dokumentów. Zespół A nie jest w stanie zwiększyć swoje składu, ponieważ pracowałby nieefektywnie. Powstaje zatem nowy zespół – B. Zespół B może składać się częściowo ze starych członków zespołu A lub może to być zupełnie nowy zespół.

Co w takiej sytuacji się dzieje? Zespół A przekazuje odpowiedzialność za istniejące dwie usługi, dotyczące drukowania dokumentów, zespołowi B. Ma to następujące zalety:

- Ponieważ te usługi są relatywnie małe i realizują konkretne funkcjonalności, nowy zespół szybko może wdrożyć się w temat.
- Nie następuje fizyczne przecięcie lub rozdzielenie kodu źródłowego. Ponieważ jest to tylko umowa w organizacji, kto za jakie usługi odpowiada, nowy zespół od następnego dnia może być odpowiedzialny operacyjnie za nowe usługi.
- Zespół B może rozwijać dalej istniejące usługi, jednak może też podjąć decyzje, że w pierwszej kolejności napisze je od nowa, po czym zacznie dalej rozwijać usługę w celu realizacji kolejnych funkcjonalności biznesowych.

Ten przykład pokazuje nam, że świat mikroserwisów pasuje do świata szybko zmieniających się wymagań biznesowych, a zarazem budowanie nowych zespołów czy zmiana struktury organizacji nie stanowi większego problemu, jeżeli system jest dobrze podzielony na małe fragmenty.

Można jedynie się zastanawiać, czy to dobrze, że jedno zagadnienie biznesowe realizuje kilka usług? Nie widzę w tym większych przeszkód. Zespół mógł podzielić funkcjonalność na kilka usług, ponieważ jedna usługa musi być wysoce wydajna – w związku z tym skalowana, inna usługa może być dostępna publicznie, więc musi być odpowiednio zabezpieczona, a jeszcze inna zawiera głęboką domenę, musi być więc dobrze zaprojektowana i napisana. Wszystkie te trzy usługi mogą być w jednej domenie.

Firma Spotify[3] przedstawiła wielkości swoich zespołów i usług. Było to 810 mikrousług, 600 developerów oraz 90 zespołów. To znaczy, że na jednego inżyniera oprogramowania przypada więcej niż jedna usługa, a średnio 7-osobowy zespół developerski utrzymuje lub pracuje na 9 usługach! Z mojego doświadczenia wynika, że przy podobnych proporcjach pracuje się najlepiej. Więcej niż 10 usług do utrzymania dla 7-osobowego zespołu stanowi spore wyzwanie.

PRZYKŁAD: WSPÓŁPRACA POMIĘDZY DOMENAMI

W moim przekonaniu zespoły nie powinny być właścicielami usług w pojęciu kodu źródłowego i zmian wprowadzanych do tego kodu. Każdy zespół powinien ponosić jedynie odpowiedzialność operacyjną za swoje usługi.

Jakie negatywne konsekwencje może mieć stosowanie zasady, że tylko zespół będący właścicielem usługi może edytować jej kod źródłowy?

Wyobraźmy sobie, że musimy dodać nowe funkcjonalności do kodu źródłowego usługi, którą opiekuje się inny zespół. Co robimy? Komunikujemy się z tym zespołem, uzgadniamy, co powinno być zrobione oraz na kiedy zmiana powinna być gotowa. Następnie zespół opiekujący się tą usługą musi to zaplanować, zrealizować, przetestować i wdrożyć. Może się tak zdarzyć, że implementacja będzie niewłaściwa. Prawie na pewno zespół realizujący zmianę nie będzie mógł jej wykonać, kiedy będzie to potrzebne. Niestety boleśnie czuje to zespół potrzebujący rozszerzenia funkcjonalności tej usług.

Stwierdzając, że zespół ponosi jedynie odpowiedzialność operacyjną za swoje usługi, pozwalamy innemu zespołowi modyfikować kod źródłowy danej usługi. Bez najmniejszych problemów zespół odpowiedzialny operacyjnie wykona przegląd kodu przed wdrożeniem tej usługi, ale dzięki takiemu podejściu zespół implementujący zmianę będzie mógł wykonać ją szybciej, a zespół sprawujący odpowiedzialność operacyjną nie będzie obciążony realizacją celów biznesowych innego zespołu.

Mowa tutaj o zmianach, które są raczej drobnymi rozszerzeniami istniejącej funkcjonalności, a nie zmianach polegających na modyfikacji koncepcyjnych działania danej usługi.

Oznacza to duże zmiany dla zespołu. Wiele mówi się o tym, by członkowie zespołu byli wielofunkcyjni. Doceniamy członków zespołu, którzy są dobrymi specjalistami w zadaniach frontendowych, ale potrafią też wprowadzić zmiany na backendzie – tylko po to, by ich frontend był lepszy. Oczywiście działa to w obie strony.

Przy takim podejściu organizacyjnym do odpowiedzialności za system potrzebne jest coś więcej. Istotne jest to, jak szybko członkowie zespołu są w stanie zrozumieć i wprowadzić zmiany – najczęściej niewielkie – do kodu źródłowego innego zespołu. Mowa tutaj o szybkości zrozumienia cudzego kodu źródłowego, który może być napisany w innym języku programowania lub z użyciem innych frameworków. Dzięki temu zespoły nie są silosami, oddzielonymi barierami od innych zespołów, tylko w razie potrzeby potrafią współpracować ze sobą, również na poziomie kodu źródłowego.

Zespół

Jak w takim razie tworzyć skuteczne zespoły, które efektywnie dostarczają wartość w środowisku mikroserwisów? Taki zespół, oprócz odpowiedzialności za pewne usługi, powinien mieć własny zestaw celów oraz zakres autonomii do ich realizacji. Wybór narzędzi czy technologii nie powinien być niczym ograniczony, a taką swobodę wyboru od strony technicznej dają nam tylko mikroserwisy. Ważne jest również, żeby organizacja nie krępowała tych wyborów. Zespół taki powinien też potrafić pracować niezależnie od innych zespołów. To znaczy powinien oczywiście samodzielnie realizować zmiany w swoich usługach, ale powinien być otwarty na zmiany w innych usługach. Niemniej najważniejszą cechą zespołu powinno być skupienie swojego obszaru działania na jednej lub maksymalnie kilku drobniejszych funkcjonalnościach biznesowych, tak by usługi, które powstają, realizowały te, i tylko te, zadania możliwie najlepiej.

PODSUMOWANIE

Budując monolity, tworzeniem architektury danego systemu zajmowali się najczęściej starsi inżynierowie oprogramowania, a sama architektura była spójna. Budując systemy w większych organizacjach, które będą oparte na mikroserwisach, odpowiedzialność za budowanie takiej architektury przejmuje organizacja i formowanie jej może nie być trywialne.

Możemy t[illegible]yć nowe rozwiązania bez zmian organizacyjnych w zespołach przez co struktura zespołów będzie nam w pewnym sensie narzucała architekturę lub może być w sprzeczności z tym, co chcemy zbudować. Oczywiście jest możliwe budowanie innej architektury systemu niż architektura organizacji, ale to musi być robione świadomie i może być bardzo trudne.

W większości sytuacji lepszym podejściem może być nakreślenie najpierw architektury rozwiązania IT. Pozwalając, by struktura zespołów dopasowała się do architektury, zyskamy komfort pracy programistów, co przełoży się na jakość oprogramowania, a w rezultacie szybkość wprowadzania zmian i niskie koszty utrzymania takiego systemu.

3. https://gotocon.com/dl/goto-berlin-2015/slides/KevinGoldsmith_MicroservicesSpotify.pdf

MICHAŁ LEWANDOWSKI

Software developer. Przez ostatnie kilka lat zajmował się różnymi rzeczami związanymi z wytwarzaniem oprogramowania, począwszy od analizy wymagań, przez prowadzenie zespołu, samo kodowanie, skończywszy na utrzymaniu i odpalaniu systemu. Jest przekonany, że każdy profesjonalista, oprócz twardych umiejętności związanych z kodowaniem, powinien mieć rozwinięte umiejętności miękkie.