IT Software Architecture, Cloud, Microservices and Processes

Release #90ef6c4, 2018-01-04

Matt Harasymczuk

2018-01-04

Introduction

Author

name Matt Harasymczuk

email matt@astrotech.io

www http://www.astrotech.io

facebook https://facebook.com/matt.harasymczuk

linkedin https://linkedin.com/in/mattharasymczuk

slideshare https://www.slideshare.net/astrotech/presentations

Tip: The most up-to-date version of this book is always at http://arch.astrotech.io

Other books from this series

Python and Machine Learning http://python.astrotech.io

DevOps and Development Tools Ecosystem http://devops.astrotech.io

GIT and GIT Flow http://git.astrotech.io

Agile, Scrum, Kanban, XP, Lean http://agile.astrotech.io

IT Software Architecture, Cloud, Microservices and Processes http://arch.astrotech.io

CHAPTER 1

Introduction

CHAPTER 2

Proces wytwarzania oprogramowania

2.1 Wstęp

2.1.1 Cel dokumentu

Dokument opisuje proces powstawania oprogramowania. Powodem stworzenia dokumentu jest konieczność uporządkowania procedur i procesów związanych z produkcją, przekazaniem oraz dystrybucją systemów informatycznych dla Klienta.

2.1.2 Odbiorcy dokumentu

Dokument kierowany jest do osób rozpatrujących innowacyjność projektu informatycznego. Ponadto może być wskazówką dla wszystkich zaangażowanych w proces powstawania oprogramowania systemów informatycznych, a w szczególności do programistów, kierowników projektów, architektów i liderów zespołów produkcji oprogramowania.

2.1.3 Zakres dokumentu

Niniejszy dokument przedstawia całościowy proces produkcji oprogramowania począwszy od przyjęcia zamówienia lub wyjścia z tzw. inicjatywą projektową na wdrożeniu i obsłudze powdrożeniowej skończywszy.

2.2 Technologie i standardy projektu

W ramach prowadzenia swojej działalności należy wybrać standardy technologii prowadzenia projektów. Wybór ten ma na celu uporządkowanie i zmniejszenie liczby wykorzystywanych technologii. Dzięki czemu wsparcie narzędziowe oraz operacyjne może zostać zapewnione w najlepszy możliwy sposób. Dla każdego projektu wybierane są stosowne technologie zapewniające jak najlepsze zrealizowanie założeń biznesowych. Wykaz technologii znajduje się w załączniku do dokumentu.

Firma jest organizacją dbającą o jakość wytworzonych rozwiązań. Dla poprawy kodu aplikacji stworzonych w ramach działań operacyjnych Firmy zdecydowano się na wprowadzenie nowoczesnych rozwiązań, w ramach czego wprowadzone i kultywowane zostały standardy wytwarzania oprogramowania, tj.:

- wykorzystanie systemu kontroli wersji,

- wdrożenie systemów i metodyk zwinnych w zarządzaniu projektami i oprogramowaniem,
- automatyzacja procesu budowania projektu,
- automatyczne zarządzanie zależnościami projektu,
- automatyzacja procesu testowego, zarówno części warstwy logiki biznesowej jak i graficznego interfejsu użytkownika,
- wprowadzenie warstwowej architektury rozwiązania,
- zastosowanie języków oprogramowania oraz technologii powszechnie stosowanych na świecie.

2.2.1 Języki programowania

Dostrzegając, iż na świecie Java de facto stała się standardem podjęliśmy decyzję aby projekty informatyczne były wytwarzane są w tym języku. Firma wytwarza oprogramowanie wykorzystując Javę po stronie serwerowej oraz w język Java Script po stronie klienta. Do pomniejszych zadań, skryptów lub narzędzi wykorzystywane są języki tj. Python, Ruby, Shell Script.

Dopuszcza się zwiększenie procentu wykorzystania innych języków lub zastosowanie technologii niewymienionych w powyższym zestawieniu jeżeli tylko jest to uzasadniona biznesowo lub informatycznie zmiana. W takim przypadku ma zastosowanie procedura wprowadzania nowych technologii w projektach.

Dokładną listę technologii precyzuje stosowny załącznik do dokumentu.

2.2.2 Procedury wprowadzania nowych technologii w projektach

W przypadku zapotrzebowania na zmianę technologii, architektury lub podziału aplikacji zgodnie z innym schematem przewidziana jest następująca procedura:

1. Wniosek zgłaszany jest do przełożonego liniowego lub lidera zespołu,
2. Lider lub przełożony wydaje opinię czy zmiana jest uzasadniona i czy należy ją rozważyć na szczeblu projektowym
3. Jeżeli zmiana jest uzasadniona lider przedstawia ją na cyklicznym spotkaniu projektowym pozostałym liderom lub kierownikom.
4. Jeżeli zmiana zostanie określona jako istotnie i pozytywnie wpływająca na jakość, szybkość wytwarzania lub działania aplikacji, ewentualnie znacząco przyczyni się do zwiększenia wartości biznesowych dopuszcza się możliwość jej wprowadzenia po uprzednim skonsultowaniu się z architektami aplikacji oraz poinformowaniu kierownika projektu i stosownego dyrektora działu. Stosowna zmiana wraz z uzasadnieniem powinna również znaleźć się w dokumentacji technicznej projektu.

2.2.3 System kontroli wersji

Firma jako standard systemu kontroli wersji wybrała rozwiązanie GIT. Aplikacja ta jest narzędziem pozwalającym na śledzenie zmian oraz ich autorów. GIT został wybrany, jako że na chwilę obecną jest najpopularniejszym oprogramowaniem tego typu na rynku i ma największe wsparcie wśród narzędzi deweloperskich. Ponadto w sieci Internet zgromadzone są duże zasoby wiedzy dotyczącej korzystania z tego oprogramowania, a społeczność chętnie pomaga rozwiązywać najczęściej spotykane problemy.

Zastosowanie GITa w projekcie wiąże się z przestrzeganiem odpowiednich konwencji nazewniczych oraz procesu wprowadzania funkcjonalności w aplikacji. Informacje te są zamieszczone w oddzielnym paragrafie “Konwencja nazewnicza w systemie kontroli wersji”.

2.2.4 Zarządzanie projektem oraz zadaniami

W Firmie oprogramowanie jest wytwarzane w duchu metodyk zwinnych (Agile). Główny nacisk jest położony na współpracę z klientem oraz jak najczęstsze udostępnianie aplikacji do testów. Dzięki takiemu rozwiązaniu ilość błędów jest mniejsza a klient jest w stanie swoje uwagi zgłosić na wczesnym etapie rozwoju systemu dzięki czemu koszt ich wprowadzenia jest mniejszy. Metodyki zarządzania projektami i zespołami są przedstawione w osobnym rozdziale.

Do systemów wspierających wytwarzanie oprogramowania w metodykach Scrum i Kanban wybraliśmy system Jira wraz z dodatkiem Jira Agile. W tym oprogramowaniu znajdują się tzw. rejestry produktów (backlogi) precyzyjnie opisujące zasady i kolejność tworzenia funkcjonalności i poprawek w systemie.

2.2.5 Budowanie projektu i zarządzanie zależnościami

Do automatyzacji budowania projektu oraz zarządzania jego zależnościami zostały wybrane rozwiązania Maven i Gradle.

Dla aplikacji zbudowanych przy użyciu innego języka programowania niż Java przewiduje się użycie odmiennych, specyficznych dla danej technologii rozwiązań.

2.2.6 Warstwowa architektura rozwiązań

Wśród aplikacji wytwarzanych w ramach projektów można wyróżnić co najmniej trzy główne warstwy, w ramach których zastosowanie mają różne technologie:

- warstwa widoku (frontendu),
- warstwa logiki biznesowej (backendu),
- warstwa persystencji (bazy danych).

Podział ten przyjął się jako standard na świecie. Dzięki wykorzystaniu takiego rozróżnienia Firma jest w stanie skutecznie tworzyć niezależne od siebie elementy aplikacji, które później stanowią całość oprogramowania.

2.3 Wersjonowanie aplikacji i kodu źródłowego

Projekty informatyczne, w których wytwarzanie zaangażowane jest wiele osób wymagają odpowiedniego podejścia do zarządzania zarówno wersjami jak i kodem źródłowym. W każdym takim oprogramowaniu wcześniej czy później przychodzi konieczność wprowadzenia systemu do kontroli wersji takiego jak np. *GIT*. Już samo to narzędzie pozwala w prosty i efektywny sposób na scalanie i śledzenie zmian wprowadzanych przez programistów. Największą jednak zaletą tego typu oprogramowania jest możliwość równoległej pracy nad systemem przez wiele osób. W XXI wieku gdzie projekty informatyczne stały się gigantyczne i długotrwałe, a w proces ich tworzenia zaangażowane są dziesiątki osób takie podejście jest jedynym skutecznym sposobem na wytwarzanie oprogramowania.

Firma jest organizacją dbającą o jakość wytworzonych rozwiązań. Dla poprawy kodu aplikacji stworzonego w ramach działań operacyjnych mają zastosowanie ogólnie przyjęte dobre praktyki wytwarzania oprogramowania oraz konwencje nazewnicze zgodne o ogólnoświatowym standardem dla danej technologii. Aby utrzymać przejrzystość oraz możliwość szybkiego śledzenia zmian, w systemie kontroli wersji został przyjęty standard nazewnictwa kolejnych przyrostów (ang. *commit*) oraz gałęzi (ang. *branch*) z nowymi funkcjonalnościami z poprawkami błędów. Przyjęta konwencja jest standardem opartym na uproszczonym schemacie *GIT Flow*, zwanym dalej *Lean GIT Flow*.

Dzięki zastosowaniu takiej konstrukcji system do przechowywania repozytorium może wymieniać informacje z aplikacją do zarządzania zadaniami oraz przyporządkowywać dany kod odpowiednim zadaniom. Umożliwia to także łatwą weryfikację oraz śledzenie postępu pracy nad konkretną funkcjonalnością.

2.3.1 Konwencja opisu zmian w systemie kontroli wersji

Każda zmiana w systemie kontroli wersji powinna zostać opisana według następującego przykładu:

```
ID-1337 Poprawka arkusza css formularza w module X
```

Powyższy przykład wymaga zastosowania odpowiedniego identyfikatora zadania w systemie do zarządzania projektem. W swoim opisie zestaw zmian (*commit*) powinien być zawierać znacznik konkretnego zlecenia na wykonanie zmian. Pozostała część opisu powinna jak najlepiej oddawać charakter wprowadzonej poprawki opisując dokładnie dokonaną zmianę. Zaleca się nieużywanie polskich znaków diakrytycznych oraz innych znaków specjalnych w opisie ze względu na możliwą niekompatybilność pomiędzy systemami. Polskie znaki specjalne należy zmienić na ich odpowiedniki. Długość pierwszej linii opisu wraz z identyfikatorem nie powinna przekraczać 80 znaków.

Dzięki zastosowaniu takiej konstrukcji system do hostowania repozytorium może wymieniać informacje z aplikacją do zarządzania zadaniami oraz przyporządkowywać dany kod odpowiednim zadaniom. Umożliwia to także łatwą weryfikację oraz śledzenie postępu pracy nad konkretną funkcjonalnością.

2.3.2 Konwencja nazewnicza wersji

W ramach projektów ma zastosowanie następująca konwencja nazewnicza wersji, tzw. *Semantic Versioning*:

```
X.Y.Z
```

Każda z kolejnych części rozdzielonych kropką jest liczbą naturalną (przykład `1.23.1`). Pierwszy segment oznacza tzw. wersję `major`, środkowy `minor` a ostatni `bugfix`.

Wszystkie narzędzia produkowane zewnętrznie oraz wewnętrznie powinny być opatrzone odpowiednią zależnością od konkretnej wersji. Nie przewiduje się wprowadzenia wersji "latest" ze względu ma możliwość niekompatybilności aplikacji ze zmianami.

Wersja major

Wersja `major` jest używana do określania zmian niekompatybilnych wstecznie lub przełomowych względem publicznego *API* aplikacji. Wszystkie narzędzia produkowane wewnętrznie lub zewnętrznie powinny precyzyjnie określać wersję `major` aplikacji, gdyż ma to krytyczny wpływ na ich działanie oraz kompatybilność.

Wersja `minor`

Wersja "minor" jest używana do określenia kolejnych przyrostów funkcjonalności aplikacji. Zgodnie z konwencją nazewniczą funkcjonalności w publicznym *API* dla danej wersji powinny wyłącznie przyrastać, chyba że jest to jasno określone i przeprowadzone zgodnie z polityką wyprowadzania zmian z użycia (ang. *deprecation*). Wprowadzone zmiany w wersji `minor` nie powinny powodować niekompatybilności pomiędzy oprogramowaniem zewnętrznym i wewnętrznym.

Wersja `bugfix`

Wersja `bugfix` jest przeznaczona do użytku wyłącznie dla poprawek bezpieczeństwa oraz funkcjonalności, wprowadzonych omyłkowo lub zauważonych podczas zwiększenia wersji `minor`.

2.3.3 Zarządzanie gałęziami

W ramach Firmy została wdrożona konwencja nazewnicza zwana *GIT Flow*. W ramach jej zastosowania wyróżnia się kilka specyficznych gałęzi rozwojowych oprogramowania. Każda z nich posiada unikalną rolę.

Branch produkcyjny `master`

W repozytorium główną gałęzią (ang. *branch*) jest `master`. Przechowywana jest w nim stabilna wersja kodu będąca odpowiednikiem wersji znajdującej się na środowisku produkcyjnym. Scalenie kodu do brancha `master` jest równoważne z wydaniem nowej wersji i jest dopuszczalne jedynie, gdy testy automatyczne, funkcjonalne, regresyjne i jednostkowe nie pozostawiają wątpliwości na temat stabilności oraz braku defektów we wprowadzonych zmianach. Branch ten odpowiada 1 do 1 sytuacji na produkcji.

Gdy kod pobierany jest z `Github.com` lub `Bitbucket.com` zwykle nie zmienia się domyślnego brancha (domyślnie jest `master`). Po ściągnięciu oczekujesz, że kod będzie stabilny i się uruchamiał. Tym samym przesłaniem kierujemy się w Firmie. Domyślny branch z repozytorium, które klonujesz musi być stabilny i zielony.

W wersji odchudzonej podejścia gałęzie z funkcjonalnościami bezpośrednio są scalane z `master` dzięki czemu integracja kodu przebiega szybko i często. Dzięki częstemu scalaniu kodu funkcjonalności są mniejsze a problemy integracyjne ujawniają się zdecydowanie szybciej. Rozwiązywanie małych konfliktów jest nie tylko łatwiejsze ale również nie wymaga dużej ingerencji w projekt.

Dopuszcza się możliwość niewykorzystywania gałęzi `develop` w projekcie, gdy wielkość projektu jest nieznaczna a wprowadzenie dodatkowego procesu przejściowego jest nadmierne. Nie zwalnia to z obowiązku utrzymywania stabilnego kodu w gałęzi głównej (`master`) i wymaga wprowadzenia podobnego procesu weryfikacji zmian dla każdej poprawki lub/i funkcjonalności, co w przypadku wdrożenia na środowisko produkcyjne.

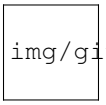

Fig.2.1.: Schemat scalania funkcjonalności z gałęzią `master`.

Gałąź integracyjna `develop`

W dużych repozytoriach, nad którymi pracuje wiele osób na raz (np. kilka 6 ± 3 osobowych zespołów) zachodzi konieczność wprowadzenie integracyjnej gałęzi rozwojowej (ang. *branch*). Zabieg ten ma na celu zabezpieczenie mastera przez scalaniem kodu, który mógłby go zdestabilizować. Dzięki takiemu podejściu proces staje się trochę bardziej skomplikowany ale za to pewniejszy i przewidywalny.

W takim przypadku w repozytorium główną gałęzią rozwojową staje się branch `develop`. Przechowywana jest w nim najnowsza wersja oprogramowania ze scalonymi ukończonymi funkcjonalnościami. Gałąź `develop` powinna przechowywać kod, co do którego poprawności nie ma zastrzeżeń. Kod powinien się budować oraz być odpowiednio przetestowany. Z gałęzi rozwojowej `develop` w każdym momencie można stworzyć tzw. kandydata do wdrożenia (ang. *release candidate*).

Stan powyżej opisany jest wysoce pożądany w przypadku każdego projektu bez względu na jego wielkość wraz z wprowadzeniem tzw. *Continuous Delivery*. Do czasu uzyskania odpowiedniej dojrzałości procesowej, zaleca się stosowanie pośredniczącej gałęzi `develop` w celu integrowania zmian.

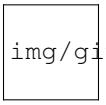

Fig.2.2.: Schemat scalania funkcjonalności z gałęzią `develop`.

Gałąź tymczasowa `release/X.Y`

Wprowadzenie brancha integracyjnego, który w standardzie *GIT Flow* nazywany jest `develop` nakłada konieczność wprowadzenia sposobu wdrażania kodu, tj. scalania go z branchem produkcyjnym (`master`). W tym celu tymczasowo powoływany jest branch `release/X.Y` (*X.Y.Z* oznaczają numer wersji zgodnie z wcześniejszym opisem, tzw. *semantic versioning*: `major.minor`), który jest tzw. kandydatem wydania (ang.

release candidate). Na tej gałęzi odpalane są wszystkie testy, podnoszona jest wersja w `pom.xml` oraz w razie konieczności wprowadzane są poprawki. Po pozytywnym przejściu przez proces testów gałąź `release/X.Y` jest scalana z gałęzią `master` a zmiana (ang. `commit`) jest otagowywany numerem wersji wdrożenia.

Obrazek poniżej przedstawia graficzną reprezentację procesu wdrożenia, tj. scalenia kodu z brancha integracyjnego `develop` do brancha stabilnego master.

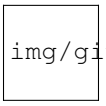

Fig.2.3.: Schemat scalania gałęzi `develop` z `master` za pośrednictwem `release`.

Rodziny branchy

Aby ułatwić wyszukiwanie wprowadzanych zmian w repozytorium oraz powiązania ich ze zleceniami i zadaniami w systemie do zarządzania projektami, Firma przyjęła konwencję nazywania gałęzi według następującego schematu:

```
feature/ID-1337-dodanie-nowej-funkcjonalnosci-do-modulu
bugfix/ID-1337-poprawka-wyswietlania-dokumentu-formularza
hotfix/ID-1337-poprawka-krytycznego-bledu-na-produkcji
```

Zgodnie z powyższym przykładem, nowa funkcjonalność powinna być poprzedzona stosownym przedrostkiem `feature/` a poprawka błędów `bugfix/`. Następnie po prefiksie następuje unikalny identyfikator zadania. Po identyfikatorze następuje zwięzły kilkuwyrazowy opis wprowadzonych modyfikacji. W opisie nie należy stosować polskich znaków diakrytycznych, aby uniknąć możliwości wystąpienia niekompatybilności pomiędzy systemami. Spacje w opisie funkcjonalności lub błędu powinny być zamienione na myślniki. Długość całego opisu wraz z identyfikatorem nie powinna przekraczać 80 znaków.

Dzięki zastosowaniu powyższej konwencji w repozytorium wszystkie zmiany będą należały do odpowiednich gałęzi funkcjonalności lub błędów i będą jednoznacznie opisane. Umożliwia to dokładne śledzenie wszystkich zmian i łączenie ich z odpowiednimi zleceniami w systemie do zarządzania projektem.

Branche `bugfix/*` i `hotfix/*`

Proces obsługi branchy `bugfix/*` i `hotfix/*` nieco się różni, chociaż schemat na rysunku wygląda bardzo podobnie.

Branche `bugfix/*` służą do poprawy błędów znalezionych podczas produkcji oprogramowania a system scalania ich z kodem źródłowym jest podobny do obsługi `feature/*`.

Branche `hotfix/*` natomiast odpowiadają za poprawkę błędów znalezionych na środowisku produkcyjnym. Dzięki takiej konwencji nazewniczej i separacji gałęzi ich obsługa, np. wdrożenie na środowisko, może być przyspieszona. Wszystkie zmiany które znajdą się w gałęziach `hotfix/*` mogą omijać standardową procedurę wdrożenia, tj. stworzenie brancha `release/X.Y` i odpalenie testów. Zmiany priorytetowe mają na celu natychmiastowe przywrócenie działania oprogramowania, np. po krytycznym błędzie na produkcji, gdzie każda sekunda zwłoki powoduje straty. Zmiany te, dopiero w późniejszym etapie poddawane są normalnemu procesowi testowania i weryfikacji. Mechanizm ten pozwala na szybkie "ugaszenie pożaru" i przywrócenie stabilności systemu. Ta funkcjonalność powinna być używana jedynie w uzasadnionych przypadkach.

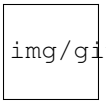

Fig.2.4.: Schemat scalania zmian z gałęzi z rodziny `bugfix/*` i `hotfix/*` do kodu źródłowego aplikacji.

Branche `feature/*`

Branche z rodziny `feature/*` służą do wprowadzania funkcjonalności do systemu. Ich nazewnictwo jest ściśle powiązane z systemem kontroli zadań (ang. *issue tracker*). Dzięki takiej separacji mamy pełną transparentność i możliwość śledzenia historii wprowadzanych zmian w projekcie.

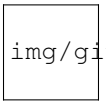

Fig.2.5.: Schemat scalania funkcjonalności `feature/*` z gałęzią `master`.

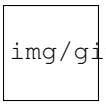

Fig.2.6.: Schemat scalania funkcjonalności `feature/*` z gałęzią `develop`.

Nazwa gałęzi dla kodu przeznaczonego do wdrożenia

Podczas procesu wdrożenia następuje moment wydzielenia gałęzi tzw. kandydata do wdrożenia (ang. *release candidate*) o nazwie:

```
release/X.Y
```

gdzie numery odpowiadają kolejnej wersji np. `release/1.4`. Konwencja nazewnicza wersji przedstawiona jest w osobnym podpunkcie.

Na wyżej wymienionej gałęzi przeprowadzane są testy i wprowadzane ewentualne poprawki zgodnie z procesem wprowadzania zmian i poprawek błędów przedwdrożeniowych. Po pomyślnej weryfikacji automatycznej następuje faza testów manualnych, zgodnie z procedurą i ścieżką ich przeprowadzania.

W miarę możliwości wszelkie akcje użytkownika końcowego lub testera powinno się automatyzować tak, aby proces weryfikacji odbywał się bezdotykowo a do jego wyników nie było zastrzeżeń.

2.3.4 Tagowanie

Po scaleniu gałęzi `release/X.Y` następuje proces oznaczania odpowiedniego momentu w historii przez tzw. tagowanie z etykietką o nazwie wersji zgodnej z odpowiednią konwencją. Dzięki temu w każdej chwili istnieje możliwość szybkiego powrotu do krytycznego momentu w repozytorium oraz zobaczenie logów zmian.

2.3.5 Proces Pull Request

Przed wprowadzeniem jakichkolwiek zmian do gałęzi integracyjnych wymagany jest proces tzw. *Pull Request*. Polega on na stworzeniu strony na której znajduje się wylistowany zmieniony kod, tj. dodane i usunięte linijki wraz ze zmodyfikowaną treścią. Na karcie *Pull Requesta* system do Ciągłej Integracji zamieszcza informacje o wyniku analizy i testów. Gdy wszystkie testy przejdą a zmiana uzyska zgodę (ang. *aproove*) przynajmniej dwóch osób pojawia się możliwość scalenia funkcjonalności do docelowego miejsca. Proces ten uodparnia kod na przypadkowe błędy. Większa ilość osób zaangażowanych w przegląd kodu procentuje w przyszłości w postaci zmniejszenia długu technicznego. Ponadto to rozwiązanie spełnia funkcję edukacyjną gdzie osoby z większym doświadczeniem mogą przekazać informacje swoim młodszym kolegom na temat konsekwencji zmian.

2.4 Procesy Continuous Integration i Delivery

Proces ciągłej integracji polega na automatycznym uruchamianiu testów aplikacji tj. testów jednostkowych, testów integracyjnych, testów regresyjnych i funkcjonalnych dla każdej wprowadzonej zmiany. Dzięki zastosowa-

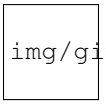

Fig.2.7.: Schemat momentu tworzenia *Pull Requesta* przy scalaniu zmian.

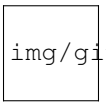

Fig.2.8.: Karta podsumowania *Pull Request* z informacjami wynikowymi z systemu budowania.

niu Continuous Integration (CI) programiści otrzymują natychmiastową informację czy ich zmiany w kodzie nie destabilizują pracy systemu oraz negatywnie wpływają na całą aplikację.

W zależności od kategorii definiuje się inny zestaw testów uruchamianych dla budowania i sprawdzania aplikacji.

Proces ciągłego dostarczania polega na doprowadzeniu procedury wdrożenia oraz budowania aplikacji do poziomu pozwalającego na otrzymywanie gotowych do wdrożenia paczek po każdej integracji nowej funkcjonalności. Continuous Delivery (CD) jest rozwinięciem procesu CI o przygotowanie przetestowanej paczki wdrożeniowej. Każda paczka musi być gotowa do wdrożenia na środowisko produkcyjne w dowolnym momencie.

Continous Delivery jest sposobem dostarczania oprogramowania służącym zautomatyzowaniu i polepszeniu tego procesu.

Automatyczne testowanie, ciągłe scalanie i wdrażanie oprogramowania pozwalają przygotować je w przejrzystej formie, aby mogły zostać poddane dalszym testom. To pozwala szybko, niezawodnie i wielokrotnie ulepszać produkt i naprawiać pojawiające się na różnych etapach produkcji błędy, uwzględniając opinie użytkowników, zmiany na rynku oraz zmiany w strategii biznesowej. Należy podkreślić, że zachowane jest przy tym minimalne ryzyko poważnej usterki oraz konieczność minimalnej poprawy instrukcji użytkowania.

Poszczególne mniejsze fragmenty wytwarzanego oprogramowania muszą przejść określone etapy walidacji na swojej drodze do publikacji. Kod jest kompilowany i pakowany za każdym razem, gdy dokonywana jest zmiana w systemie kontroli wersji. Następnie jest wielokrotnie testowany. Dopiero po tym może zostać wdrożony.

Wykorzystywanie metody Continuous Delivery w pracy Firmie jest skutkiem nastawienia na konkretne i dobre efekty. Dzięki niej oszczędza się także czas i pieniądze, które wykorzystuje się na dalszy, coraz bardziej zaawansowany rozwój.

Do procesu zarówno Continous Integration jak i Continous Delivery w Firmie wykorzystywany jest system Jenkins. Oprogramowanie to pozwala na automatyzację kroków procesu znacząco obniżając czas i koszt tworzenia kolejnych przyrostów. W kolejnych krokach działania tego narzędzia jest pobranie kodu źródłowego, jego kompilacja, poddanie go testom jednostkowym i funkcjonalnym a na samym końcu analiza statyczna kodu źródłowego i zbudowanie artefaktu gotowego do wdrożenia na środowisko testowe lub/i produkcyjne.

2.4.1 Continuous Integration

Proces ciągłej integracji polega na automatycznym uruchamianiu testów aplikacji tj. testów jednostkowcyh, testów integracyjnych, testów regresyjnych i funkcjonalnych dla każdej wprowadzonej zmiany. Dzięki zastosowaniu Continuous Integration (CI) programiści otrzymują natychmiastową informację czy ich zmiany w kodzie nie destabilizują pracy systemu oraz negatywnie wpływają na całą aplikację.

Proces CI w ramach Firmy jest realizowany za pomocą systemu do automatycznego budowania - Jenkins. System ten periodycznie odpytuje repozytorium o zmiany w repozytorium i w przypadku wykrycia takowych uruchamia z góry zdefiniowane akcje, tj. wymienione powyżej.

Plany w Jenkinsie dzielą się na różne kategorie:

- budowanie gałęzi stabilnej (master),
- budowanie gałęzi rozwojowej (develop),

- budowanie Pull Requestów,
- budowanie zmian w gałęziach z nowymi funkcjonalnościami (feature),
- budowanie zmian w gałęziach z poprawkami błędów (bugfix),
- plan wdrożeniowy.

W zależności od kategorii definiuje się inny zestaw testów uruchamianych dla budowania i sprawdzania aplikacji. Powyższy podział ma na celu umożliwienie uzyskania natychmiastowych i szybkich informacji po wprowadzeniu zmian (feature, bugfix). Dla Pull Requestów uruchamiane są testy integracyjne informujące czy wprowadzone zmiany nie destabilizują działania aplikacji. Natomiast dla zmian w gałęzi rozwojowej (zmiany po wprowadzeniu funkcjonalności) oraz gałęzi stabilnej (wdrożenia) uruchamiana jest dogłębna i długotrwała statyczna analiza kodu wraz z odpowiednim procesem.

2.4.2 Continuous Delivery

Proces ciągłego dostarczania polega na doprowadzeniu procedury wdrożenia oraz budowania aplikacji do poziomu pozwalającego na otrzymywaniu gotowych do wdrożenia paczek po każdej integracji nowej funkcjonalności. Continuous Delivery (CD) jest rozwinięciem procesu CI o przygotowanie przetestowanej paczki wdrożeniowej. Każda paczka musi być gotowa do wdrożenia na środowisko produkcyjne w dowolnym momencie.

Proces Continuous Delivery w Firmie jest realizowany za pomocą systemu automatyzacji budowania - Jenkins. System ten po przetestowaniu zmian w repozytorium generuje binarny artefakt, tj. skompilowaną paczkę kodu gotową do wdrożenia na środowisko produkcyjne i umieszcza ją w systemie Artifactory uprzednio nazywając ją odpowiednio stosownie do zbudowanych zmian.

Continous Delivery jest sposobem dostarczania oprogramowania służącym zautomatyzowaniu i polepszeniu tego procesu.

Automatyczne testowanie, ciągłe scalanie i ciągłe wdrażanie oprogramowania pozwalają przygotować je w przejrzystej formie, aby mogły zostać poddane dalszym testom. To pozwala szybko, niezawodnie i wielokrotnie ulepszać produkt i naprawiać pojawiające się na różnych etapach produkcji błędy, uwzględniając opinie użytkowników, zmiany na rynku oraz zmiany w strategii biznesowej. Należy podkreślić, że zachowane jest przy tym minimalne ryzyko poważnej usterki oraz konieczność minimalnej poprawy instrukcji użytkowania.

Poszczególne mniejsze fragmenty wytwarzanego oprogramowania muszą przejść określone etapy walidacji na swojej drodze do publikacji. Kod jest kompilowany i pakowany za każdym razem, gdy dokonywana jest zmiana w systemie kontroli wersji. Następnie jest wielokrotnie testowany. Dopiero po tym może zostać wdrożony.

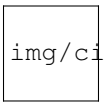

Fig.2.9.: Koncepcja metody Continuous Delivery. Poszczególne kroki procesu zapętlają się tworząc niekończącą się pętlę.

Wykorzystywanie metody Continuous Delivery w pracy Firmy jest skutkiem nastawienia na konkretne i dobre efekty. Dzięki niej Firma również oszczędza czas i pieniądze, które wykorzystuje na dalszy, coraz bardziej zaawansowany rozwój.

2.4.3 Continuous Deployment

Proces produkcji oprogramowania charakteryzujący się wdrażaniem zmian na środowisko produkcyjne po każdej wprowadzonej poprawce lub nowej funkcjonalności. Proces ten charakteryzuje dojrzałe oprogramowanie oraz 100% zaufanie testom automatycznym. Continuous Deployment jest rozwinięciem ideii Continuous Delivery.

2.5 Standardy zarządzania projektem i pracą zespołu

Agile można rozumieć jako sześć spójnych i łączących się dyscyplin:

- miękkie:
  - zarządzanie projektami,
  - biznes i produkty,
  - komunikacja i organizacja,
- techniczne:
  - devops,
  - architektura IT,
  - jakość i praktyki.

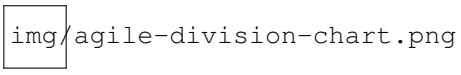

Fig.2.10.: Agile można rozumieć jako sześć spójnych i łączących się dyscyplin.

Firma dostrzegając zalety wytwarzania oprogramowania w sposób zwinny wprowadził usprawnienia w swoim procesie mające na celu umiejscowienie go w czołówce firm IT. Metodyki prowadzenia projektów i współpracy tj. Scrum i Kanban pozwalają firmie dostarczać wartość zamawiającemu zgodną z jego oczekiwaniem.

U podstaw adaptacji powyższych szkieletów organizacyjnych leży iteracyjność oraz stała współpraca z klientem. Do codziennych zastosowań stosowane jest połączenie metodyk Scrum i Kanban w zależności od projektu oraz zapotrzebowania.

W Agile w wytwarzaniu oprogramowania nacisk kładziony jest na dostarczanie software'u dostosowanego do potrzeb klienta w stałej współpracy z nim. W porównaniu z tradycyjnymi metodykami zarządzania projektami podejście to cechuje się odpowiedzią na zmiany wymagań, priorytetów oraz zakresów w trakcie trwania projektu.

Scrum jest systemem zarządzania pracą oraz frameworkiem komunikacyjnym. System ten przyjął się na świecie jako najpopularniejszy sposób prowadzenia projektów informatycznych. Obecnie wykorzystują go największe firmy z branży IT i nie tylko.

W myśl SCRUMu, oprogramowanie wytwarzane jest iteracyjnie i przyrostowo, a projekt jest podzielony na mniejsze, tygodniowe lub dwu tygodniowe fazy, zwane iteracjami lub Sprintami, następujące bezpośrednio po sobie.

Podstawowym celem metodyki SCRUM jest zatem kreowanie wartości na każdym ze stadiów przygotowania i realizowania projektu, mając na uwadze jego indywidualną specyfikę interakcje i współpracę z Klientami, dynamicznie zmieniające się otoczenie i dostosowywanie się do zmian oraz potencjał ludzki.

W Firmie zespoły odpowiedzialne za rozwój oprogramowania tworzą samoorganizujący i wielofunkcyjny zespół, tzw. SCRUM Team, w którego w skład wchodzą: Product Owner, Development Team, SCRUM Master.

Podczas planowania uczestnicy spotkania podejmują decyzje, które zadania będą wchodziły następnej iteracji, a co za tym idzie, jakie funkcjonalności zostaną oddane po kolejnym przyroście. Spotkanie w zależności od trwania Sprintu trwa od 2 do 4 godzin. Jego celem jest ustalenie w jaki sposób zostanie wyprodukowana część oprogramowania – precyzuje się dużo drobnych zadań, członkowie zespołu sami wybierają zadania dla siebie i na każde z nich przeznaczają maksymalnie 2 dni.

Aby uzyskać jak najlepsze wsparcie dla zwinnego wytwarzania oprogramowania Firma wdrożyła system JIRA. Oprogramowanie to pozwala na tworzenie tzw. Backlogów czyli spriorytetyzowanych list zadań oraz dzielenie ich na poszczególne Sprinty - iteracje. JIRA pozwala na podgląd zarówno, krótkoterminowych zadań operacyjnych jak i wizji długoterminowej całego projektu. Dzięki jej zastosowaniu zespoły dostały do dyspozycji narzędzie, które pozwala im na pracę wg. wymagań biznesowych i częste rozliczenie się z klientem.

2.5.1 Agile

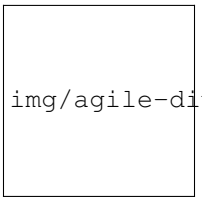

Fig.2.11.: Podział Agile ze względu na metody.

Filozofię Agile w prowadzeniu projektu najlepiej definiują 4 cechy stanowiące główne przesłanie tego podejścia a będące jego manifestem:

- ludzie i interakcje ponad procesy i narzędzia,
- działające oprogramowanie ponad obszerną dokumentację,
- współpraca z klientem ponad formalne ustalenia,
- reagowanie na zmiany ponad podążanie za planem.

Pełna treść manifestu Agile znajduje się w osobnym załączniku do tego dokumentu.

2.5.2 Scrum

Scrum jest systemem zarządzania pracą oraz frameworkiem komunikacyjnym. System ten przyjął się na świecie jako najpopularniejszy sposób prowadzenia projektów informatycznych. Obecnie wykorzystują go największe firmy z branży IT i nie tylko.

W myśl SCRUMu, oprogramowanie wytwarzane jest iteracyjnie i przyrostowo, a projekt jest podzielony na mniejsze, tygodniowe lub dwu tygodniowe fazy, zwane iteracjami lub Sprintami, następujące bezpośrednio po sobie. Cały proces produkcji oparty jest na 3 filarach:

- Przezroczystości - istotne informacje widoczne są dla odpowiedzialnych osób oraz stosowany jest wspólny język,
- Weryfikacji - dokonuje się regularnych przeglądów postępów prac, wyszukiwania artefaktów oraz identyfikacji wszelkich odchyleń,
- Adaptacji - dotyczy procesu i produktów, wprowadzana jest natychmiast.

Podstawowym celem metodyki SCRUM jest zatem kreowanie wartości na każdym ze stadiów przygotowania i realizowania projektu, mając na uwadze jego indywidualną specyfikę, interakcje i współpracę z Klientami, dynamicznie zmieniające się otoczenie i dostosowywanie się do zmian oraz potencjał ludzki.

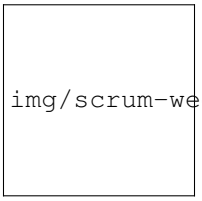

Fig.2.12.: Rozłożenie spotkań w tygodniu pracy zespołu SCRUM.

Scrum precyzyjnie określa sposób współpracy biznesu z IT w ramach projektu lub/i konkretnego modułu aplikacji. Scrum definuje pięć rodzajów spotkań:

- Sprint Planning,
- Backlog Refinement,
- Sprint Review,

- Retrospective,
- Daily Scrum.

Celem organizowania tych wydarzeń jest zapewnienie regularności w komunikacji wewnątrz SCRUM Teamu, minimalizacja ilości spotkań, zachowanie przezroczystości, weryfikacja oraz adaptacja. Ważną cechą spotkań jest ograniczenie czasowe.

Scrum sprawdza się wszędzie tam, gdzie jesteśmy w stanie stworzyć rejestr zmian oraz listę funkcjonalności do wprowadzenia w projekcie tzw. backlog. Dzięki niemu oraz priorytetyzacji poszczególnych elementów listy zespół ma świadomość wagi oraz kolejności wprowadzania poprawek do oprogramowania. Pracę można zaplanować a oddawanie kolejnych kawałków aplikacji odbywa się w krótkich przyrostach tj. tygodniowe lub dwutygodniowe sprinty. Należy podkreślić, że implementacja części zasad SCRUMa nie jest SCRUMem.

Wszystkie pojęcia związane z tą metodyką, tj. Scrumem, znajdują się w osobnym załączniku w tym dokumencie.

2.6 Jakość oprogramowania

Testowanie oprogramowania w Firmie odbywa się nie tylko w sposób manualny. Ważnym procesem jest testowanie zautomatyzowane. Polega to na tym, że programista lub/i tester przygotowuje odpowiednie narzędzia testowe, które w sposób jak najbardziej autonomiczny wykonują zadany scenariusz i oceniają jego rezultaty.

Zaletami testowania automatycznego są m.in.:

- Możliwość szybkiej i wydajnej weryfikacji poprawek błędów,
- Możliwość odtworzenia testu, co jest bardzo przydatne zwłaszcza przy sprawdzaniu, czy wskazywane błędy zostały usunięte,
- Możliwość kompleksowej analizy wyników testów,
- Szybsze i tańsze tworzenie sprawozdań,
- Nieomylność przy wprowadzaniu danych,
- Możliwość podania dużej liczby danych testowych,
- Szybsze wprowadzanie testów,
- Umożliwienie systematyczności,
- Oszczędzenie czasu testerów,
- Unikniecie błędów związanych z czynnikiem ludzkim.

Firma stosuje testy jednostkowe dla poszczególnych jednostek oprogramowania, lekkie testy integracyjne, służące wykryciu błędów w interakcjach pomiędzy modułami, oraz testy interfejsu, służące wykryciu błędów w interfejsach użytkownika.

W procesie testowania pomocne są Firmie również narzędzia pozwalające budować, gromadzić i umieszczać na serwerze archiwa, a także zapewniające przykładowe dane dla testów do łatwego wykorzystania i ponownego użycia.

Testy automatyczne są zatem świetnym uzupełnieniem testów manualnych, a ogólny zestaw testów znacząco wpływa na poprawę wydajności wytwarzanych produktów. W trakcie przeprowadzania testów następuje wdrożenie demonstracyjne prototypu projektu powstałego w danym Sprincie.

Firma w ramach przygotowania Projektu, określiła, że jednym z kluczowych elementów poprawnego działania aplikacji jest jego wydajność.

Decyzja taka wynikła z faktu docelowego wykorzystywania systemu Projektu do codziennej równoległej pracy znacznej liczby użytkowników.

W ramach sprawdzenia testów wydajnościowych systemu Projektu przygotowano scenariusze testów wydajnościowych i obciążeniowych.

Na ich podstawie wykonywane są testy mierzące wydajność aplikacji.

Aby utrzymać wysoki standard jakości oprogramowania został wdrożony system SonarQube. Oprogramowanie to pozwala na wyświetlanie wyników statycznej analizy kodu źródłowego w prostej i przejrzystej formie. Dzięki jego wykorzystaniu deweloperzy mają możliwość zobaczenia najczęściej popełnianych błędów oraz ich rozwiązania.

Kolejnym elementem, który znacząco przyczynia się do wysokiej jakości wytwarzanych rozwiązań jest tzw. system Pull Requestów, tj. oprogramowanie do przeglądu kodu (ang. *Code Review*). Zarówno do przetrzymywania repozytoriów kodu źródłowego jak i obsługi procesu Pull Request wykorzystywane jest narzędzie Stash. Aplikacja ta pozwala w prosty i intuicyjny wprowadzać komentarze do kodu źródłowego co wspiera komunikację programistów. Wg. obecnych ustawień każda zmiana w kodzie źródłowym wymaga akceptacji przynajmniej dwóch innych programistów oraz pozytywnego wyniku budowania i testów w systemie Continuous Integration.

2.6.1 Testowanie automatyczne

Testowanie oprogramowania w Firmie odbywa się nie tylko w sposób manualny. Ważnym procesem jest testowanie zautomatyzowane. Polega to na tym, że programista lub/i tester przygotowuje odpowiednie narzędzia testowe, które w sposób jak najbardziej autonomiczny wykonują zadany scenariusz i oceniają jego rezultaty.

Zaletami testowania automatycznego są m.in.:

- Możliwość szybkiej i wydajnej weryfikacji poprawek błędów,
- Możliwość odtworzenia testu, co jest bardzo przydatne zwłaszcza przy sprawdzaniu, czy wskazywane błędy zostały usunięte,
- Możliwość kompleksowej analizy wyników testów,
- Szybsze i tańsze tworzenie sprawozdań,
- Nieomylność przy wprowadzaniu danych,
- Możliwość podania dużej liczby danych testowych,
- Szybsze wprowadzanie testów,
- Umożliwienie systematyczności,
- Oszczędzenie czasu testerów,
- Uniknięcie błędów związanych z czynnikiem ludzkim.

Firma stosuje testy jednostkowe dla poszczególnych jednostek oprogramowania, lekkie testy integracyjne, służące wykryciu błędów w interakcjach pomiędzy modułami, oraz testy interfejsu, służące wykryciu błędów w interfejsach użytkownika.

Przykładem zastosowanych w Firmie testów integracyjnych są testy osadzone na kontenerze *Java EE*, wykonywane na innowacyjnych, rozszerzalnych platformach testowania, co umożliwia programistom łatwe tworzenie automatycznej integracji oraz testów funkcjonalnych dla middleware'u *Javy*.

W procesie testowania pomocne są Firmie również narzędzia pozwalające budować, gromadzić i umieszczać na serwerze archiwa, a także zapewniające przykładowe dane dla testów do łatwego wykorzystania i ponownego użycia. Ponadto, testy są w pełni zintegrowane z *IDE* (zintegrowanym środowiskiem programistycznym). Dzięki tym wszystkim rozwiązaniom testowanie jest proste, szybkie i wydajne.

Testy automatyczne są zatem świetnym uzupełnieniem testów manualnych, a ogólny zestaw testów znacząco wpływa na poprawę wydajności produktów wytwarzanych przez Firmę.

W trakcie przeprowadzania testów następuje wdrożenie demonstracyjne prototypu projektu powstałego w danym Sprincie.

2.6.2 System zapewnienia jakości kodu

Firma w ramach systemu zapewnienia jakości kodu, używa narzędzi umożliwiających regularne stosowanie cross-checków oraz *code review*. Wykorzystywany jest także system kontroli wersji służący do śledzenia zmian w kodzie źródłowym.

2.7 Projektowanie zorientowane na użytkownika - *User-centred design*

W Firmie znajduje się specjalna komórka zajmująca się opracowywaniem oprogramowania pod kątem jakości doświadczenia użytkowników.

Zespół User Experience prowadzi badania potrzeb użytkowników, prace analityczne i optymalizujące procesy biznesowe, opracowuje e-usługi, prowadzi prace nad ergonomią rozwiązań (usability, architektura informacji), tworzy projekty graficzne interfejsu użytkownika (*GUI*), implementacje stworzonych rozwiązań (*front-end development*), a także zapewnia testowanie powstałych rozwiązań z udziałem użytkowników (usability testing). Zespół dba również o dostosowywanie produktów do potrzeb osób niepełnosprawnych, co jest szczególnie istotne w kontekście publicznego dostępu do wybranych systemów informatycznych.

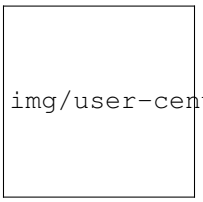

Fig.2.13.: Znaczenie User Experience

Rozwiązania tworzone są zgodnie z metodologią User-centered design, w której potrzeby, wymagania i ograniczenia końcowego użytkownika są szczegółowo badane na każdym etapie procesu projektowego. Ze względu na swoją specyfikę, zespół stale współpracuje z zespołami programistycznymi i włącza się w ich prace. Zajmuje się także wewnętrznym odbiorem stworzonych przez programistów systemów i sprawdza je pod kątem użyteczności. Całość działań jest możliwa dzięki iteracyjnemu systemowi tworzenia oprogramowania stosowanemu w Firmie.

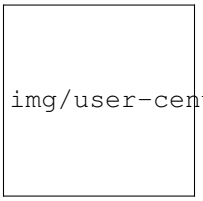

Fig.2.14.: Projektowanie zorientowane na użytkownika w iteracyjnym systemie wytwarzania oprogramowania

2.7.1 Proces planowania wydajności systemu

Firma w ramach przygotowania Projektu, określiła, że jednym z kluczowych elementów poprawnego działania aplikacji jest jego wydajność. Decyzja taka wynikła z faktu docelowego wykorzystywania systemu Projektu do codziennej równoległej pracy znacznej liczby użytkowników.

Proces planowania wydajności systemu zawierał:

- zaplanowanie, przygotowanie i wdrożenie odpowiedniej architektury sprzętowej i systemowej środowiska, na którym pracuje system Projektu;
- zaplanowanie, przygotowanie i wdrożenie odpowiedniej architektury projektu samej aplikacji;
- zaplanowanie, przygotowanie, wdrożenie i przeprowadzanie wielopoziomowych testów wydajnościowych systemu.

Pierwsze dwa punkty mają swoje odzwierciedlenie w Projekcie Technicznym. Celem niniejszego dokumentu jest doprecyzowanie trzeciego punktu.

Firma wdrożyła praktykę projektową Ciągłej Integracji (ang. *Continuous Integration*, *CI*).

2.7.2 Testy wydajnościowe

W ramach sprawdzenia testów wydajnościowych systemu Projektu przygotowano scenariusze testów wydajnościowych i obciążeniowych. Na ich podstawie wykonywane są testy mierzące wydajność aplikacji.

Scenariusze oraz wyniki testów bazujących na tych scenariuszach umieszczone są w osobnym, załączonym dokumencie “Scenariusze testów wydajnościowych i obciążeniowych”.

2.8 Dobre praktyki programistyczne

Wprowadzenie do procesu produkcyjnego Firmy dobrych praktyk inżynierskich (ang. good engineering practice) ma na celu zapewnienie, że rezultaty uzyskiwane w wyniku zastosowanych metod produkcji są źródłem korzyści dla społeczeństwa oraz że rozwijane metody produkcji są finansowo opłacalne. Założenia te realizowane są m.in. poprzez:

- wykorzystywanie współczesnej wiedzy poprzez zastosowanie teoretycznych i stosowanych zasad produkcji oprogramowania,
- racjonalne wykorzystywanie zebranych informacji,
- branie odpowiedzialności za podejmowanie decyzji zgodnych z bezpieczeństwem i dobrem publicznym oraz natychmiastowe ujawnianie czynników mogących stwarzać zagrożenie,
- unikanie, jeśli to tylko możliwe, mogących wystąpić konfliktów interesów i ujawnianie ich poszkodowanym stronom, jeśli istnieją,
- uczciwe i realistyczne formułowanie stwierdzeń i oszacowań, opartych na dostępnych danych,
- utrzymywanie i polepszanie kompetencji technicznych pracowników,
- ciągły rozwój technologii oraz wiedzy dotyczącej poprawnego jej stosowania i konsekwencji z tego płynących,
- podejmowanie zadań technologicznych przez pracowników tylko wtedy, gdy są wykwalifikowani poprzez szkolenia lub doświadczenie, lub po całkowitym ujawnieniu istotnych ograniczeń w ich umiejętnościach,
- rzetelne określenie wymagań i oczekiwanych trudności (dokonanie analizy i rozwiązania problemów oraz wyrażenie ich na piśmie),
- szukanie, oferowanie i akceptowanie szczerej i konstruktywnej krytyki pracy technicznej, w celu uznania i poprawienia błędów oraz słusznego przypisania wkładu w pracę innych osób,
- niedopuszczanie się krzywdzenia innych osób, niszczenia ich mienia, reputacji lub szkodzenia ich posadzie, poprzez fałszywe lub złośliwe działanie,
- wzajemne wspieranie się współpracowników w rozwoju zawodowym oraz pomoc w postępowaniu zgodnie z powyższymi normami etycznymi.

2.9 Proces zarządzania zmianą

Fig.2.15.: Procesy zarządzania zmianą.

2.9.1 Procesy wejścia dla zarządzania zmianą

Wejściem do procesu zmiany jest zgłoszenie otrzymane poprzez określony kanał kontaktowy. Pierwszym zadaniem związanym ze zgłoszeniem jest określenie jego typu (incydent, wniosek o usługę lub inne).

Przyjmuje się jednolity tryb nadsyłania zgłoszeń incydentów i wniosków o usługę w ramach prac nad aplikacją. W praktyce sposób postępowania z incydentami oraz wnioskami o usługę jest podobny, dlatego inicjalnie oba są zawarte w zakresie procesu Zarządzania Incydentem.

Wnioski o usługę realizowane są w trybie indywidualnym ale powtarzalnym (związanym z analiza ryzyka, czasochłonności, możliwości wprowadzenia zmiany w aktualnym stadium zaawansowania prac), natomiast w przypadku stwierdzenia incydentu, następują działania związane z Zarządzaniem Incydentem.

Przyczyna wystąpienia incydentu może okazać się oczywista, co skutkuje w szybkim rozwiązaniu i zamknięciu incydentu. W identyfikacji przyczyny przydatna jest Baza Wiedzy dotycząca istniejących rozwiązań. W przypadku, gdy nie jest możliwa szybka identyfikacja przyczyny i rozwiązanie zgłoszenia, następuje uruchomienie procesu Zarządzania Problemem.

2.9.2 Incydent (błąd w funkcjonalności aplikacji)

Incydent to każde zdarzenie, które nie jest częścią normalnego działania usługi, zakłóca tę usługę, które powoduje lub może powodować przerwę w dostarczaniu usługi, względnie obniżenie jej jakości.

2.9.3 Wniosek o usługę (zmiana funkcjonalności)

Wnioski o usługę mają charakter powtarzalny, obsługiwane są zawsze w ten sam sposób, dla których możliwe jest zagwarantowanie przez Firmę czasu realizacji.

2.9.4 Zarządzanie incydentem

Celem procesu Zarządzania Incydentem jest przywrócenie normalnego działania aplikacji i usług tak szybko, jak to możliwe oraz minimalizowanie niekorzystnego wpływu Incydentu na działanie aplikacji w przyszłości, tak by zapewnić najwyższy możliwy poziom jakości i dostępności świadczonych przez Projekt usług.

Przez “normalne działanie usług” należy rozumieć działanie usług określone w ustalonej z zamawiającym dokumentacji. Wszystkie zgłoszenia użytkowników, czyli incydenty i wnioski o usługę, powinny zostać zarejestrowane w Bazie Zgłoszeń. Jest to pierwsza czynność procesu Zarządzania Incydentem wykonywana przez I linię wsparcia tzw. Service Desk.

Użytkownik przy rejestracji swojego zgłoszenia otrzymuje unikalny numer referencyjny (nr zgłoszenia), na który może powołać się przy kolejnym kontakcie w sprawie danego zgłoszenia.

Następnym krokiem jest klasyfikacja incydentu.

Klasyfikacja obejmuje następujące czynności:

- identyfikację usługi, której dotyczy incydent i wybór odpowiedniej kategoryzacji,
- identyfikację komponentu usługi związanej z incydentem,
- identyfikację fragmentu kodu, którego dotyczy incydent,
- przypisanie priorytetu na podstawie wpływu danego incydentu na funkcjonowanie usług i aplikacji, użytkownik zgłaszając incydent może zażądać nadania konkretnego priorytetu zgłoszeniu przy czym w przypadku rozbieżności pomiędzy oceną zgłaszającego i Service Desk, priorytet jest uzgadniany przez Firmę z Klientem w późniejszym terminie, a w trakcie obsługi przypisywany jest priorytet określony przez zgłaszającego,
- konfrontacja danego incydentu z Bazą Wiedzy w celu znalezienia gotowego rozwiązania lub rozwiązania zastępczego,
- próba rozwiązania incydentu na podstawie gotowego rozwiązania lub własnego doświadczenia,

- jeśli rozwiązania nie ma lub jest nieskuteczne – wybór i przypisanie kompetentnej grupy wsparcia, mogącej rozwiązać incydent.

Kolejnym krokiem jest zamknięcie incydentu, polegające na zapewnieniu:

- (jeżeli zastosowano rozwiązanie zastępcze nieznajdujące się dotychczas w Bazie Wiedzy) poprawności wpisu w Bazie Wiedzy dotyczącego zastosowanego rozwiązania (czy wpis jest zwięzły i zrozumiały?),
- poprawnej klasyfikacji incydentu ze względu na przyczynę jego wystąpienia,
- sprawdzenia czy zastosowane rozwiązanie jest uzgodnione i zatwierdzone przez Klienta,
- wszystkie istotne informacje dotyczące incydentu są zarejestrowane, a w szczególności rejestrowane są:
- czas poświęcony na rozwiązanie incydentu,
- osoba zamykająca incydent,
- data i czas zamknięcia incydentu.

2.9.5 Zarządzanie problemem

Celem procesu Zarządzania Problemem realizowanego przez Firmę jest minimalizowanie niekorzystnego wpływu incydentów oraz zabezpieczenie przed ponownym pojawieniem się incydentów związanych z tą samą przyczyną.

W pierwszym kroku procesu pracownik obsługujący problem jest zobowiązany do zarejestrowania nowego Problemu w Bazie Wiedzy. Następnie dokonuje klasyfikacji problemu zgodnie z przyjętymi zasadami. Klasyfikacja dokonywana jest w celu odpowiedniej alokacji zasobów, tak aby problemy o najwyższym priorytecie (poziomie negatywnego wpływu na realizację celów funkcjonowania aplikacji i usług) były rozwiązywane w pierwszej kolejności.

W kolejnym kroku ten sam pracownik lub zespół diagnozuje problem oraz proponuje jego rozwiązanie. Po tym etapie, czyli w momencie gdy znana jest już przyczyna wystąpienia, problem staje się znanym błędem. Kolejne czynności zmierzające do trwałego wyeliminowania przyczyny zwane są Kontrolą błędu.

Rezultatem procesu Zarządzania Problemem musi być wpis do Bazy Wiedzy dokonany przez pracownika obsługującego problem, identyfikujący rozwiązanie lub akceptujący rozwiązanie zastępcze. Dany wpis powinien być na tyle zrozumiały i szczegółowy, aby kolejne zgłoszenia tego samego typu, nie wymagały ponownego uruchomiania procesu Zarządzania Problemem i mogły być rozwiązane przez Specjalistów.

Może wystąpić sytuacja, w której rozwiązanie problemu będzie wymagało od pracowników obsługujących problem wprowadzenia zmian. Zmiany są realizowane w procedurze Zarządzania Zmianą i w takim przypadku dopiero zastosowanie (przygotowanie, przetestowanie i wprowadzenie) zmiany pozwala zamknąć rozwiązany problem oraz wszelkie incydenty z nim powiązane.

2.9.6 Proces zarządzania zmianą

Zmiana, to dodanie, modyfikacja lub usunięcie czegokolwiek, co mogłyby mieć wpływ na działanie aplikacji i świadczone przez nią usługi. W ten sposób ogólna definicja zmiany obejmuje swym zakresem każdą zmianę w architekturze, procesach, narzędziach i innych elementach konfiguracji.

Celem procesu jest zapewnienie, aby na każdym etapie cyklu życia aplikacji i jej usług, wszelkie zmiany kontrolowane były poprzez standardowe metody i procedury, które pozwalają minimalizować zakłócenia w jakości świadczonych usług. Za proces zarządzania zmianą jest odpowiedzialna Firma.

Ogólny sposób obsługi zmian przedstawiony jest na schemacie poniżej.

Wejściem do procesu jest zgłoszenie Incydentu, lub złożony przez ITSM wniosek o zmianę (RFC, z ang. Request of Change).

Na etapie tworzenia aplikacji, większość zmian wynika ze zgłoszonych Incydentów, natomiast wniosek o zmianę dotyczy tylko procesu wdrożeniowego i może być zgłoszony tylko przez wskazanych pracowników Firmy (w szczególności dotyczy to zmian standardowych, dla których decyzja jest preautoryzowana).

Obsługa zadania zmiany rozpoczyna się od klasyfikacji i przypisania odpowiedniego priorytetu. Jeśli zmiana zostanie sklasyfikowana jako zmiana standardowa realizowana jest w uproszczony sposób. Zmiana standardowa jest określona wcześniej i decyzja o jej wdrożeniu jest automatycznie autoryzowana. Szczegóły dotyczące zmian standardowych i procedurze ich obsługi zostały określone w dokumentach roboczych dotyczących budowy i eksploatacji aplikacji i jej środowiska.

Jeśli zmiana nie jest zmianą eksploatacyjną następuje ocena zmiany. Każda taka zmiana przed wprowadzeniem musi zostać zatwierdzona. Z punktu widzenia procesu zarządzania zmianą bardzo istotne jest określenie trybu, w jakim zmiana ma być zatwierdzona. Tryb ten wynika bezpośrednio z charakteru zmiany. Ze względu na to, że sposób autoryzacji może trwać długo (potrzebne jest zwołanie zespołu wewnątrz Firmy, wymagana jest konsultacja z Klientem lub użytkownikami końcowymi) w pewnych sytuacjach może być to nieakceptowalne.

Dotyczy to szczególnie zmian, które wiążą się np. z krytycznymi poprawkami bezpieczeństwa, które powinny być wdrażane możliwie szybko, a jednocześnie proces musi zapewnić decyzję o wdrożeniu takiej zmiany. Dlatego też zmiany te klasyfikowane są jako pilne i decyzja o ich wdrożeniu leży w kompetencji.

Po autoryzacji planowanej zmiany kolejnym krokiem jest przygotowanie i realizacja zatwierdzonej zmiany. Zakres przeprowadzanej zmiany zawiera dokumentacja związana z obsługa Incydentu, lub dokument RFC. Dokumenty te w szczególności muszą uwzględniać przygotowanie planu implementacji zmiany oraz aktualizacji dokumentacji oraz systemu zarządzania konfiguracją.

Fig.2.16.: Schemat obsługi wniosku o zmianę.

Ostatnim działaniem w cyklu życia zmiany jest przegląd i zamknięcie zapisu zmiany. Przegląd powinien zapewnić, że wszelkie niezbędne informacje dotyczące zmiany zostały zapisane w rekordzie zmiany, w tym informacje dotyczące sukcesu/porażki wdrożenia zmiany. Umożliwia to prowadzenie analiz wykonanych zmian, w celu poszukiwania (i eliminacji) powtarzających się zmian.

Istotne jest to, że proces zarządzania zmianą nie realizuje samego wdrożenia zmiany – to domena procesu Zarządzania Wydaniami. Proces nie odpowiada także za identyfikację komponentów, na które dana zmiana może mieć wpływ oraz za aktualizacje rekordu zmiany – za te zagadnienia odpowiada proces Zarządzania Konfiguracją.

Zarządzanie Zmianami jest silnie powiązane z Zarządzaniem Konfiguracją i Zarządzaniem Wydaniami. Z tego względu planuje się implementację tych procesów równocześnie.

2.9.7 Zarządzanie wydaniami, wydajnością i wdrożeniami

W ramach procesu zarządzania wydaniami obowiązuje oddzielny proces opisany w dokumencie: Proces wersjonowania aplikacji.

W ramach zarządzania wydajnością, obowiązuje proces opisany w dokumencie: Proces planowania wydajności systemu.

W ramach zarządzania wdrażaniem wersji, obowiązuje proces oparty o schemat przedstawiony w dokumencie: Proces wdrażania wersji (obejmuje on swym zakresem wszelkie środowiska istotne dla prac programistycznych, testów i akceptacji).

2.9.8 Walidacja i testowanie usług

Warunkiem wdrożenia nowej wersji, a więc elementem procedury zarządzania zmianą – oprócz procesu zarządzania wydaniami i zarządzania wdrażaniem wersji jest również pozytywny wynik walidacji i testowania usług.

W Firmie stosujemy dwa podstawowe typy walidacji i testowania:

- Walidacja i testowanie usług które podlegają formalnym odbiorom w ramach realizowanego projektu. Mamy tutaj do czynienia z przejściem przez testy akceptacyjne określone na podstawie wymagań funkcjonalnych. W tym wypadku walidacja i testowanie polega na potwierdzeniu, że wszystkie przewidziane wcześniej testy zakończone zostały pozytywnie.
- Walidacja i testowanie kolejnych wydań. Tutaj mamy do czynienia z weryfikacją w następujących obszarach:
  - Testy regresji, czyli potwierdzenie, że system, po wgraniu poprawki, będzie działał poprawnie w zakresie nieobjętym poprawką.
  - Testy nowej/poprawionej funkcjonalności, czyli potwierdzenie, że planowane wraz z wdrożeniem funkcjonalności zostały odpowiednio zaimplementowane.

Zakres testów, którym podlegają usługi jest określony w dokumencie / jest ustalany na spotkaniach roboczych przed rozpoczęciem testów ... lub jak wyżej w treści – ze wynika z opisanych w poszczególnych dokumentach funkcjonalności.

2.9.9 Zarządzanie wiedzą

Celem Zarządzania Wiedzą jest zebranie wiedzy posiadanej przez pracowników Firmy tworzących aplikację i jej środowisko, ale także zapewnienie, że informacja jest dostępna przy założeniu: odpowiednia informacja trafia w odpowiednie miejsce lub jest dostarczana odpowiednim osobom.

W Firmie prowadzona jest Baza Wiedzy i każdy pracownik realizujący zadania związane z wytworzeniem produktu jest zobowiązany do korzystania i bieżącego uzupełnienia Bazy Wiedzy.

Zakres informacji utrzymywanych w Bazie Wiedzy w zakresie Projektu obejmuje następujące elementy:

- Dokumentacja systemu - całość dokumentacji wytwarzanej przez Firmę wraz z budową systemu.
- Informacje dotyczące eksploatacji Projektu, w tym obejścia, znane błędy i ich rozwiązania.
- Inna dokumentacja powstająca w trakcie wytwarzania i eksploatacji.

2.9.10 Podział odpowiedzialności

Zarządzający problemami

W zakresie obsługi problemów:

- zorganizowanie, utrzymanie i przeglądy procesu Zarządzania Problemem,
- przeglądy efektywności i sprawności działań mających na celu zapobieganie powstawaniu Problemów,
- dostarczanie informacji zarządczej do Zarządzającego Service Desk,
- zarządzanie zespołem ds. rozwiązywania Problemów, zapewnieniem zasobów niezbędnych do prawidłowego działania procesu,
- rozwój i utrzymanie systemu wspierającego proces Zarządzania Problemem,
- kontaktowanie się z dostawcami zewnętrznymi i zapewnienie, że wywiązują się oni z zapisanych ustaleń dotyczących rozwiązania problemu i/lub dostarczenia niezbędnych informacji dotyczących Problemu.

2.9.11 Zarządzający konfiguracją

Zakres odpowiedzialności:

- Uzgadnianie zakresu procesu, funkcji i przedmiotów, które mają być kontrolowane, informacji, które muszą być zarejestrowane w ramach Zarządzania Konfiguracją
- Opracowywanie standardów zarządzania konfiguracją,

- Zarządzanie pryncypiami, procesami oraz ich implementacją w ramach Zarządzania Konfiguracją
- Weryfikacja i akceptacja zmian w zakresie struktury Bazy Konfiguracji
- Zapewnianie dostępu do wiedzy w ramach ról biorących udział w procesie.
- Dbanie o odpowiedni poziom wiedzy i zaangażowanie osób w ramach ról biorących udział w procesie
- Zarządzenia prawami dostępu do Bazy Konfiguracji
- Zapewnienie efektywności procesu Zarządzania Konfiguracją

Za zarządzanie konfiguracją odpowiada wyznaczony pracownik po stronie Firmy.

Zarządzający zmianami

Zakres odpowiedzialności:

W zakresie zarządzania zmianą:

- dostosowanie procesu zarządzania zmianą (kanały komunikacji, przebieg działań, sposób opisu RFC w zależności od rodzaju zmiany) do organizacji,
- odbiór, przegląd, logowanie i przydzielanie priorytetu do wszystkich RFC we współpracy z inicjatorem,
- koordynacja oceny i szacowania wpływu zmiany na funkcjonowanie aplikacji,
- autoryzacja zmian eksploatacyjnych w warstwie technicznej aplikacji,
- harmonogramowanie zmian eksploatacyjnych,
- komunikowanie zaplanowanych zmian,
- dokonywanie przeglądów wykonanych zmian,
- opracowywanie raportów dotyczących zmian,
- raportowanie przypadków obchodzenia procesu zarządzania zmianą (zmiany wprowadzone z pominięciem procedur zarządzania zmianą).

W zakresie zarządzania wydaniem:

- ustalenie procesu zarządzania wydaniem, w zależności od rodzaju zmiany,
- grupowanie zmian w wydania,
- koordynacji wdrażania wydań.

Za rolę Zarządzającego Zmianami odpowiada wyznaczony pracownik po stronie Firmy.

Zrządzający ServiceDesk

Zarządzający Service Desk to rola realizowana przez wyznaczonego pracownika Firmy, która w całości odpowiada za funkcję Service Desk.

Zakres odpowiedzialności:

- zapewnienie prawidłowej obsługi wszystkich zgłoszeń obsługiwanych przez Service Desk,
- zapewnienie odpowiednich kompetencji, uprawnień i procedur do wykonywania wniosków o zmianę,
- usługę przewidzianych do realizacji przez Konsultantów Service Desk,
- zapewnienie odpowiedniej ilości pracowników w godzinach funkcjonowania Service Desk,
- tworzenie raportów i statystyk dotyczących funkcjonowania Service Desk,
- zapewnienie prawidłowego funkcjonowania kanałów komunikacji z Service Desk oraz narzędzia zarządzania zgłoszeniami

Ponadto w ramach zarządzania incydentami:

- Zorganizowanie i utrzymanie efektywnego i sprawnego procesu Zarządzania Incydentem:
  - zdefiniowanie aktywności w procesie,
  - zdefiniowanie niezbędnych procedur,
  - zdefiniowanie kategorii incydentów,
  - zdefiniowanie kluczowych wskaźników wydajności procesu,
  - przydzielenie pracownikom ról w procesie,
- zarządzanie linią wsparcia obejmujące między innymi:
  - kontrolę obciążenia poszczególnych pracowników,
  - kontrolę rozwiązań zastosowanych w zamkniętych incydentach,
  - sygnalizowanie/organizowanie niezbędnych szkoleń.
- monitorowanie skuteczności procesu Zarządzania Incydentem oraz proponowanie ulepszeń procesu:
  - na żądanie przygotowanie raportów i statystyk związanych z jakością procesu w oparciu o zdefiniowane kluczowe wskaźniki wydajności,
- rozwój i utrzymanie systemu wspierającego proces Zarządzania Incydentem.

Użytkownik

Zakres uprawnień:

W zakresie zgłoszeń (incydentów, wniosków o usługę i innych):

- inicjowanie zgłoszeń poprzez wybrany kanał kontaktu z Service Desk,
- przestrzeganie zaleceń Konsultantów Service Desk oraz innych pracowników Firmy przekazanych w ramach realizacji zgłoszenia,
- w razie braku akceptacji rozwiązania, ponowne otworzenie zgłoszenia,

W zakresie standardowych wniosków o usługę:

- zgłaszanie potrzeby realizacji wniosku o usługę,

W zakresie zarządzania zmianą:

- składanie propozycji modyfikacji aplikacji oraz związanych z nią usług.

Zarządzający wydaniami i wdrożeniami

Zakres odpowiedzialności:

- Całościowa odpowiedzialność za proces zarządzania wydaniami.
- Koordynacja prac zespołów przygotowujących wydanie.
- Przygotowywania raportów z postępów wdrożenia wydań.
- Weryfikacja stanu sprzętu i oprogramowania przed i po wdrożeniu wydania.
- Delegowanie czynności związanych z wdrażaniem wydań odpowiednim zespołom po stronie Wykonawcy

Za rolę Zarządzającego Wydaniami i Wdrożeniami jest odpowiedzialny wyznaczony pracownik po stronie Firmy.

Narzędzia wykorzystywane przy realizacji procedury zarządzania zmianą – i podległymi jej procesami.

2.10 Bezpieczeństwo aplikacji, kodu źródłowego i danych

Ataki na systemy informatyczne mają z reguły u swoich podstaw wprowadzenie do systemu danych, których typ nie został przewidziany w czasie jego tworzenia.

Istotą ataków bardzo często jest wprowadzenie takich danych, które system informatyczny, w którejś ze swoich warstw, zinterpretuje jako komendy lub polecenia, zmieniając w ten sposób jego działanie. Dane te wówczas mogą: nie wpływać bezpośrednio na działanie systemu, ale powodować udostepnienie danych, które nie miały być dostępne, przekształcać działanie systemu na skutek modyfikacji parametrów kontrolujących jego działanie.

Bez względu na charakter jaki posiadają niedozwolone dane wprowadzane do systemu - w zabezpieczaniu systemów przed atakami kluczowe jest weryfikowanie danych wejściowych.

2.10.1 Ogólne środki zaradcze

Akceptacja tylko tych danych, które zostały przewidziane w fazie projektowania systemu.

Metoda ta polega na weryfikacji danych wejściowych i zbadaniu czy spełniają one założone z reguły biznesowe kryteria jakości. Jeżeli nie - są odrzucane i nie mogą dostać się głębiej do struktur systemu. Jednym z przykładów może być zaimplementowanie w systemie weryfikacji poprawności danych w polu reprezentującym PESEL, która sprawdzi i wykluczy możliwości wprowadzenia w tym polu innych znaków niż cyfry (np. litery, znak plus).

Weryfikacja i odrzucanie danych wejściowych, przez wzgląd na zawartość złośliwą

Weryfikacja ta opiera się sprawdzeniu czy dane wejściowe nie noszą znamion próby ataku. Np. próba wykrywania w danych komend systemu operacyjnego (np. su), elementów języka SQL, czy też kont użytkowników systemowych (np. root).

Dla niektórych pól zastosowanie tej techniki jest w ogóle niemożliwe. Ingeruje ona w treść wprowadzanych danych również, kiedy nie stanowią one zagrożenia, a wyeliminowanie wszystkich potencjalnie groźnych słów kluczowych jest zadaniem trudnym w implementacji i utrzymaniu. Z tego powodu stosuje się sprawdzenia wystąpień jedynie symboli sterujących i oznaczanie ich jako nieaktywne - wówczas słowa kluczowe nie będą interpretowane jako polecenia systemowe, czy też kod wykonywalny.

Weryfikacja danych po stronie serwera

Weryfikacja danych po stronie serwera powinna odbywać się zawsze, nawet wtedy, kiedy została zaimplementowana po stronie aplikacji klienckiej, gdyż weryfikacja ta po stronie klienta jest bardzo słabym zabezpieczeniem. Weryfikację po stronie klienta należy traktować jako informację dla użytkownika, że system nie pozwoli na wprowadzenie danej treści, a nie realne zabezpieczenie.

Przekazywanie danych w systemie przez warstwy abstrakcji

Posługiwanie się warstwami abstrakcji w systemie nie tylko ułatwia wytwarzanie kodu źródłowego, ale również powoduje modyfikację danych w procesie komunikacji między warstwami. Zjawisko to zwiększa szansę na dostrzeżenie nieprawidłowości w danych np. poprzez błędy w procesie analizy składniowej (ang. parsing), a także zabezpiecza przed jednym z najbardziej powszechnych ataków jakim jest SQL injection.

Sposoby weryfikacji danych wejściowych

Możliwa weryfikacja:

- długości danych wejściowych oraz czy mieszczą się one w dopuszczalnych zakresach wartości i słownikach,
- czy dane wejściowe zawierają znaki niedozwolone,

- danych po stronie serwerowej systemu, nawet wtedy, kiedy po stronie klienckiej wykonuje się identyczne weryfikacje,
- plików cookie,
- danych wprowadzonych w pola ukryte,
- wartości ze względu na występowanie niedozwolonych słów i znaków, sugerujących próbę wprowadzenia danych złośliwych (np. elementów języka SQL, Java, itp.),
- parametrów wejściowych we wszystkich metodach publicznych klas, a także ich relacji do innych parametrów wejściowych oraz aktualnego stanu obiektu danej klasy,
- źródła pochodzenia danych (zabezpieczenie przed atakami typu CSRF - Cross-Site Request Forgery).

2.10.2 Kontrola dostępu

Podstawową kwestią ochrony systemów centralnych i końcowych oraz danych jest wprowadzenie odpowiedniej kontroli dostępu, którą należy stosować w celu:

- Ograniczenia możliwości działania użytkowników,
- Ograniczenia dostępu użytkowników do zasobów,
- Definicji funkcji, które użytkownicy mogą stosować do danych.

Mechanizmy kontroli dostępu powinny uniemożliwiać nieuprawnionym: przeglądania, modyfikowania i kopiowania danych. Dodatkowo, mogą powstrzymać przed zastosowaniem złośliwego kodu lub nieuprawnionych działań przez napastnika, wykorzystującego zależności infrastruktury.

Dostęp do informacji w systemach oraz dokumentacji a informacja publiczna

Udostępnieniu danych na podstawie "Ustawy o dostępie do informacji publicznym" nie podlegają informacje oraz kod źródłowy aplikacji zgromadzony w Bazie Wiedzy, Systemie Zgłoszeń, Repozytorium Kodu Źródłowego i w dokumentacji!

Wyżej wymienione repozytoria są objęte tzw. klauzulą informacji zastrzeżonej przedsiębiorstwa i nie powinny być udostępniane dla osób powołujących się na tą ustawę. W przypadku wstąpienia roszczącego na drogę sądową informacje te nie powinny zostać udostępnione bez prawomocnego wyroku sądu.

Użytkowanie sprzętu prywatnego

Aby zapewnić bezpieczeństwo środowiska pracy w organizacji regulamin polityki bezpieczeństwa Firmy zabrania używania urządzeń prywatnych do wykorzystania w celu służbowym w Firmie.

Odstępstwo od tej reguły może mieć miejsce tylko i wyłącznie za zgodą kierownika, architekta lub stosownego zastępcy Dyrektora i musi być dobrze umotywowane. W powyższych i uzasadnionych przypadkach muszą zostać spełnione obowiązujące w Firmie standardy bezpieczeństwa, w stopniu nie niższym niż te, dotyczące pracy na służbowych komputerach.

Na szczególną uwagę należy zwrócić aby:

- dysk musi być szyfrowany bezpiecznie,
- firewall musi być skonfigurowany,
- praca na użytkowniku musi się odbywać na użytkowniku pozbawionym praw administratora,
- system musi być bezpieczny, aktualny, wspierany przez producenta,
- w systemie nie ma malware'u (oprogramowanie antywirusowe w systemie Windows, chrootkit/debsums w Linux/*nix),
- nie przechowywanie danych/kopii zapasowych na zdalnych chmurach.

2.10.3 Weryfikacja i sprawdzanie danych wejściowych

Ataki na systemy informatyczne mają z reguły u swoich podstaw wprowadzenie do systemu danych, których typ nie został przewidziany w czasie jego tworzenia.

Istotą ataków bardzo często jest wprowadzenie takich danych, które system informatyczny, w którejś ze swoich warstw, zinterpretuje jako komendy lub polecenia, zmieniając w ten sposób jego działanie. Dane te wówczas mogą nie wpływać bezpośrednio na działanie systemu, ale powodować udostępnienie danych, które nie miały być dostępne, przekształcać działanie systemu na skutek modyfikacji parametrów kontrolujących jego działanie.

Bez względu na charakter jaki posiadają niedozwolone dane wprowadzane do systemu - w zabezpieczaniu systemów przed atakami kluczowe jest weryfikowanie danych wejściowych.

2.10.4 Ogólne środki zaradcze

Akceptacja tylko tych danych, które zostały przewidziane w fazie projektowania systemu

Metoda ta polega na weryfikacji danych wejściowych i zbadaniu czy spełniają one założone z reguły biznesowe kryteria jakości. Jeżeli nie - są odrzucane i nie mogą dostać się głębiej do struktur systemu. Jednym z przykładów może być zaimplementowanie w systemie weryfikacji poprawności danych w polu reprezentującym PESEL, która sprawdzi i wykluczy możliwości wprowadzenia w tym polu innych znaków niż cyfry (np. litery, znak plus).

Weryfikacja i odrzucanie danych wejściowych, przez wzgląd na zawartość złośliwą

Weryfikacja ta opiera się sprawdzeniu czy dane wejściowe nie noszą znamion próby ataku. Np. próba wykrywania w danych komend systemu operacyjnego (np. su), elementów języka SQL, czy też kont użytkowników systemowych (np. root).

Dla niektórych pól zastosowanie tej techniki jest w ogóle niemożliwe. Ingeruje ona w treść wprowadzanych danych również, kiedy nie stanowią one zagrożenia, a wyeliminowanie wszystkich potencjalnie groźnych słów kluczowych jest zadaniem trudnym w implementacji i utrzymaniu. Z tego powodu stosuje się sprawdzenia wystąpień jedynie symboli sterujących i oznaczanie ich jako nieaktywne - wówczas słowa kluczowe nie będą interpretowane jako polecenia systemowe, czy też kod wykonywalny.

Weryfikacja danych po stronie serwera

Weryfikacja danych po stronie serwera powinna odbywać się zawsze, nawet wtedy, kiedy została zaimplementowana po stronie aplikacji klienckiej, gdyż weryfikacja ta po stronie klienta jest bardzo słabym zabezpieczeniem. Weryfikację po stronie klienta należy traktować jako informację dla użytkownika, że system nie pozwoli na wprowadzenie danej treści, a nie realne zabezpieczenie.

Przekazywanie danych w systemie przez warstwy abstrakcji

Posługiwanie się warstwami abstrakcji w systemie nie tylko ułatwia wytwarzanie kodu źródłowego, ale również powoduje modyfikację danych w procesie komunikacji między warstwami. Zjawisko to zwiększa szansę na dostrzeżenie nieprawidłowości w danych np. poprzez błędy w procesie analizy składniowej (ang. parsing), a także zabezpiecza przed jednym z najbardziej powszechnych ataków jakim jest SQL injection.

Sposoby weryfikacji danych wejściowych

Możliwa weryfikacja:

- długości danych wejściowych oraz czy mieszczą się one w dopuszczalnych zakresach wartości i słownikach,
- czy dane wejściowe zawierają znaki niedozwolone,

- danych po stronie serwerowej systemu, nawet wtedy, kiedy po stronie klienckiej wykonuje się identyczne weryfikacje,
- plików cookie,
- danych wprowadzonych w pola ukryte,
- wartości ze względu na występowanie niedozwolonych słów i znaków, sugerujących próbę wprowadzenia danych złośliwych (np. elementów języka SQL, Java, itp.),
- parametrów wejściowych we wszystkich metodach publicznych klas, a także ich relacji do innych parametrów wejściowych oraz aktualnego stanu obiektu danej klasy,
- źródła pochodzenia danych (zabezpieczenie przed atakami typu CSRF - Cross-Site Request Forgery).

2.10.5 Szczegółowy opis zagrożeń i obrony

Cross-site Scripting

Atakiem typu cross-site scripting zagrożone są serwery sieci, serwery aplikacji i środowiska aplikacji. Ataki te są możliwe, kiedy napastnik używa aplikacji internetowej do wprowadzenia złośliwego kodu, często języka skryptowego JavaScript lub aktywnych zawartości, takich jak: ActiveX, VBscript, Shockwave, Flash, itp.

Złośliwy kod ukrywany jest często przez używanie technik kodujących, takich jak: Unicode.

Do dwóch głównych kategorii cross-site scripting zalicza się:

- przechowywanie: kod wejściowy przechowywany jest w bazie danych na stałe (np. login użytkownika, wiadomość, itp.),
- odbijanie: kod wejściowy wybiera trasę alternatywą do ofiary, np. e-mail.

Do głównych zagrożeń zalicza się:

- proste zakłócenia np. wyświetlanie nieoczekiwanej zawartości,
- przeadresowywanie użytkownika do innej strony,
- "porwania" (hijack) sesji,
- ujawnienia nieautoryzowanej zawartości i zmian zawartości witryny.

Środki kontrolno-zaradcze Należy sprawdzać czy nagłówki, pliki cookie, pola formularza, ciągi zapytań zawierają dozwolone parametry/treści. Aplikacje mogą zyskać znaczną ochronę przez konwersję następujących znaków w generowanych danych wyjściowych (języki mogą posiadać funkcje umożliwiające wykonanie tego w sposób automatyczny):

z	do
<	&lt;
>	&gt;
(	&#40;
)	&#41;
#	&#35;
&	&#38;

SQL Injection

Napastnicy mogą bezpośrednio przesyłać zapytania lub polecenia do silnika bazy danych, kiedy dane wejściowe użytkownika nie są rygorystycznie sprawdzane.

Niedostatecznie zweryfikowane parametry mogą zawierać polecenie SQL, które w momencie skierowania do aplikacji zostaną umieszczone w dynamicznym zapytaniu bazy danych, wykonywanym zgodnie z uprawnieniami konta aplikacji. Poziom zagrożenia wzrasta wraz z poziomem uprzywilejowania konta.

Skutkami mogą być:

- narażenie prywatności danych klienta,
- dostęp do osobistych danych klienta (dane finansowe, medyczne, itp.),
- nieuprawniona zmiana hasła administratora albo innych haseł klienta,
- nieautoryzowana zmiana danych i oddziaływanie na integralność bazy danych,
- utrata podstawowych tabel.

Środki kontrolno zaradcze

- Należy sprawdzać czy dane wejściowe są akceptowalne; jeśli nie - odrzucać je.
- Nigdy nie należy nadawać uprawnień administratora bazy danych użytkownikom aplikacyjnym. Aplikacja sieciowa powinna funkcjonować z minimalnymi przywilejami wymaganymi do wykonywania jej funkcji.
- Należy sprawdzać poprawność kodów wyjściowych i zwrotnych, aby zapewnić oczekiwane przetwarzanie.
- Należy weryfikować uprawnienia użytkownika do wykonywania zapytań na wybranych tabelach.
- Należy konwertować dane wejściowe do systemu do bezpiecznej postaci.

Wprowadzanie poleceń systemowych

Większość języków programowania zapewnia używanie poleceń systemowych i wiele aplikacji korzysta z tej funkcjonalności. Interfejsy systemowe w języku programowania i skryptowania przekazują polecenia wejścia do podległego systemu operacyjnego. Z kolei OS przetwarza dane wejściowe i zwraca wyniki do aplikacji (lub pliku bądź innego uchwytu systemowego) w formie binarnej lub tekstowej.

Zależnie od języka programowania lub skryptu i systemu operacyjnego, możliwa jest:

- wykonanie dowolnego polecenia przez system,
- zmiana parametrów przekazanych do komend systemu,
- wywołania dodatkowych poleceń w ramach poprawnie wykonywanych poleceń.

Środki kontrolno-zaradcze

- Należy sprawdzać czy dane wejściowe są akceptowalne; jeśli nie - odrzucać je.
- Nigdy nie należy pozwalać serwerowi sieci pracować jako ADMINISTRATOR lub ROOT.
- Aplikacja sieci powinna funkcjonować z minimalnymi uprawnieniami wymaganymi do wykonywania jej funkcji.
- Jeżeli polecenia OS muszą zostać użyte, wszystkie parametry wprowadzane do nich powinny być bardzo dokładnie sprawdzone. Dane wprowadzane przez użytkownika nie mogą być przekazywane wprost do polecenia systemowego bez analizy składniowej.
- Należy zaimplementować odpowiednie mechanizmy obsługi ewentualnych błędów, upływu przewidzianego czasu lub blokad podczas prośby.
- Należy sprawdzać poprawność kodów wyjściowych i zwrotnych, aby zapewnić właściwe przetwarzanie.
- Należy ograniczyć dostęp do programów wykonujących polecenia systemowe, np. cmd.exe.

Obchodzenie ścieżek

System plików serwera sieciowego może być użytkowany do czasowego lub trwałego zbierania informacji.

Jeżeli aplikacje i serwery sieciowe nie sprawdzają albo nie obsługują prawidłowo meta-znaków do opisu ścieżek (np. '../'), aplikacja może być narażona na atak obejścia ścieżki. Napastnik może stworzyć żądanie podania danych z fizycznej lokalizacji pliku, takie jak /etc/passwd (nazywane też groźbą ujawnienia pliku). Ataki takie są często wykonywane w połączeniu z wykonywaniem poleceń systemowych i SQL Injection.

Środki kontrolno-zaradcze

- Należy wykorzystywać funkcje normalizacji ścieżki zawartej w języku programowania.
- Należy usuwać niebezpieczne elementy ścieżek, takie jak '../' oraz ich warianty Unicode z danych wejściowych systemu.
- Należy używać bezwzględnych ścieżek, wykorzystując zmienne środowiskowe lub konfigurację do określenia lokalizacji plików i katalogów.
- Należy sprawdzać czy dane wejściowe są akceptowalne; jeśli nie - odrzucać je.

2.10.6 Meta-znaki

Znaki niedrukowalne i drukowalne, oddziałujące na zachowanie poleceń: systemu operacyjnego, języka programowania, procedur programu i pytań baz danych, są zwykle wprowadzane do parametrów kodowanych przez URL w ciągach zapytań.

Przykłady meta-znaków

Znak	Znaczenie
;	Dla dodatkowego wykonywania poleceń
|	Dla przekierowań strumienia wynikowego z programu do innych poleceń
!	Dla ponownego wykonywania poprzednio używanych poleceń
&	Dla dodatkowego wykonywania poleceń
x20	Spacje dla fałszowania URL i innych nazw
x00	Puste bajty dla odcinania ciągów znaków i nazw pliku
x04	EOF dla fałszowania zakończeń pliku
x0a	Nowe linie dla dodatkowego wykonania poleceń,
x0d	Nowe linie dla dodatkowego wykonania poleceń,
x1b	Klawisz Escape - zależny od OS
x08	Klawisz Backspace - zależny od OS (usuwanie plików logujących, zmienianie zawartości pliku)
x7f	Klawisz Delete - zależny od OS
~	Tylda - zależna od OS (automatyczne rozszerzenia nazw)

Środki kontrolno-zaradcze

- Wszędzie, gdzie to możliwe należy usuwać meta-znaki z danych wejściowych.
- Należy sprawdzać czy dane wejściowe posiadają oczekiwany typ danych.
- Analiza składniowa parametrów URL oraz danych formularzy w celu zablokowania, substytucji przez bezpieczne encje lub wyłączenia (ang. escape) takich znaków.

Bajty zerowe

Wiele aplikacji programowych dla dalszego postępowania i funkcjonowania, często przekazuje dane bezpośrednio do niższego poziomu funkcji C.

Jeżeli ciąg "XXX0YYY" zostanie poprawnie przyjęty przez aplikację, zostanie skrócony do postaci "XXX". Dzieje się tak dlatego, że zerowe bajty (0) są interpretowane jako zakończenie ciągu.

Aplikacje, które nie sprawdzają adekwatnie danych wejściowych mogą zostać oszukane poprzez wprowadzenie bajtów zerowych w "kluczowych" parametrach. Jest to zwykle wykonywane przez kodowanie URL bajtów zerowych (%00). W wyjątkowych sytuacjach możliwe jest użycie znaków Unicode.

Skutkami ataku mogą być:

- Udostępnienie ścieżki fizycznej, plików oraz informacji operacyjnych systemu
- Obcięcie ścieżki
- Wykonanie poleceń OS
- Wydanie polecenia parametrom
- Ominięcie kontroli podczas szukania podciągów w parametrach
- Odcięcie ciągów przekazanych do zapytań SQL

Środki kontrolno-zaradcze

- Przed czynnościami aplikacyjnymi należy sprawdzić wszystkie dane wejściowe i zapewnić poprawną interpretację danych.

Przepełnione bufory

Zjawisko to wiąże się z przekazaniem dużej ilości danych, przekraczających ilość oczekiwaną przez aplikację dla danego wejścia lub parametrów ciągu zapytań. Jedynym ze skutków przepełnienia bufora może być nieoczekiwane zachowanie aplikacji, która pozwoli napastnikowi wykonywać polecenia w jej kontekście. Ryzyko jest większe wtedy, kiedy aplikacja działa na poziomie systemu lub konta administratora systemu operacyjnego.

Środki kontrolno-zaradcze

- Należy sprawdzać ciągi danych wejściowych oraz odrzucać żądania wykraczające poza rozmiar wcześniej zdefiniowanego ciągu,
- Należy sprawdzać ciągi zapytań URL, zawartość oraz nagłówki i odrzucać jakiekolwiek żądania wykraczające poza ustalone wcześniej rozmiary zbioru,
- Uruchamiać aplikacje w kontekście konta o ograniczonych uprawnieniach, jeśli to możliwe.

2.10.7 Normalizacja

Normalizacja (ang. normalization lub canonicalization, c14n - dotyczące normalizacji do postaci kanonicznej) jest to proces konwersji na prostszą formę. Aplikacje sieciowe muszą obsługiwać normalizacje różnych danych wejściowych oraz wyjściowych, od kodowania URL do tłumaczenia adresu IP.

Unicode

Kodowanie Unicode jest sposobem przechowywania znaków z wieloma bajtami. Jeżeli dane wejściowe są dopuszczone, Unicode może zostać wykorzystany w celu ukrycia złośliwego kodu. Wiele sposobów kodowania tekstu wskazuje RFC2279.

Środki kontrolno-zaradcze

- Należy wybierać odpowiednią formę normalizacji i upewniać się czy wszystkie wprowadzane dane użytkownika są ustandaryzowane do tej formy, zanim jakakolwiek zatwierdzona decyzja zostanie wykonana.
- Kontrola bezpieczeństwa powinna być przeprowadzona po zakończeniu procesu kodowania.

Kodowanie URL

Tradycyjne aplikacje sieciowe przenoszą dane pomiędzy serwerem a klientem używającym protokołów HTTP lub HTTPS. Do głównych metod odbioru zalicza się:

Metoda	Opis
GET	kiedy dane są przekazywane w URL
POST	kiedy dane są przekazywane w nagłówkach HTTP

Jeżeli dane zawarte są w URL, konieczne jest kodowanie zachowujące odpowiednią składnię URL. RFC1738 definiuje URL a RFC2396 definiuje URI. Obydwa ograniczają dozwolone znaki w URL lub URI do podzbiorów zbiorów znaków US-ASCII. RFC1738 oznacza:

- Tylko alfanumeryczne, specjalne znaki "$-_.+!*'()," oraz znaki zastrzeżone używane do zastrzeżonych celów mogą zostać użyte jako niekodowane w obrębie URL.

Jednakże dane używane przez aplikacje sieciowe nie są ograniczane w ten sposób. Wcześniejsza wersja HTML pozwalała na pełen zakres zbioru znaków ISO-8859-1 (ISO Latin-1). Specyfikacja HTML 4.0 została rozszerzona, aby zezwolić na dowolne znaki w zbiorze Unicode.

Dla kodowania znaku w URL, 8-bitowy kod szesnastkowy poprzedzany jest prefixem %. Do przykładów zalicza się: zbiór znaków US-ASCII, który reprezentuje spację z dziesiętnym kodem 32 (20 w kodzie szesnastkowym). Korzystający z aplikacji sieciowych mają zatem możliwość widzieć spacje, które zostały zamienione na następujący ciąg znaków "%20" w URL.

Choć niektóre znaki nie potrzebują kodowania URL, kod 8 bitowy może być zakodowany.

W związku z tym, że kodowanie URL zezwala w rzeczywistości na przekazywanie dowolnych danych serwerowi, koniczne okazuje się podjęcie stosownych środków ostrożności przez aplikacje sieciowe. Brak ich może spowodować stan, w którym aplikacja będzie podatna na złośliwe działania.

Środki kontrolno-zaradcze

- Nie należy używać metody GET do zatwierdzania zmiany w formularzu; aby uniknąć dodawania danych do URL używaj HTTP POST.
- Jeśli URL ma być użyty do przekazywania danych do serwera sieci, należy ograniczyć rodzaje przekazywanych danych i nie zezwalać na dane tekstowe. Należy stosować zasady sprawdzenia w celu wyczyszczenia danych i zapewnienia ich poprawnego typu i rozmiaru.
- Nie należy opierać się na sprawdzeniu po stronie klienta.
- Dane wrażliwe, związane z bezpieczeństwem, lub obszerne objętościowo należy wysyłać wyłącznie za pomocą metody POST, ze względu na przechowywanie URL w logach dostępowych serwera.

2.10.8 Manipulacja parametrami

Napastnik może przeprowadzić atak na niewystarczająco zabezpieczone aplikacje, modyfikując dane zawarte w plikach cookie, nagłówkach HTTP lub URL w sposób niezgodny z zamierzeniami twórców aplikacji. Jeżeli aplikacja pozwoli na przyjęcie tak zmodyfikowanych danych (np. tokenu sesji), może dojść do przełamania zabezpieczeń.

Nie można zatem przyjąć, że dane przesłane do przeglądarki pozostaną niezmienione, chyba, że są kryptograficznie chronione na poziomie aplikacji. SSL nie chroni przed tego typu atakami, ponieważ dane są zmienione po stronie klienta, przed ich wysłaniem do serwera.

Manipulacja plikami cookie

Każda forma plików cookie przed odesłaniem ich do serwera może zostać zmanipulowana. Rozmiar manipulacji zależy od celów, do których zostały one użyte. Wiele plików cookie jest kodowanych jako Base64, co nie zapewnia kryptograficznej ochrony.

Środki kontrolno-zaradcze

- Nie należy ufać danym wejściowym użytkownika dla wartości, które są już znane.
- Należy używać jednego tokenu dla zidentyfikowania zbioru danych charakterystycznych dla danej sesji użytkownika zmagazynowanych w pamięci po stronie serwera.

Manipulacja polami formularza

Wybrane lub wprowadzone informacje są zwykle magazynowane jako wartości pola formularza i wysyłane do aplikacji przez polecenia HTTP (GET lub POST). HTML również może przechowywać wartości pola jako ukryte, które nie są wyświetlane na ekranie przez przeglądarkę, ale są gromadzone i przedstawione jako parametry podczas przesyłania formularzy.

Niezależnie od typu pola formularza (pole rozwijane, zaznaczenie lub bloki tekstowe), wszystkie mogą być zmodyfikowane przez użytkownika. W większości przypadków jest to możliwe przez edycję źródła strony.

Do przykładów manipulacji polem formularza od strony klienta zalicza się m.in.:

Zwiększenie przywilejów: zmiana wartości z 0 na 1 po to, aby móc przejść na tryb debugowania, co może powodować uruchomienie dodatkowych funkcji aplikacji, ujawnić hasła systemu i bazy danych, układu logicznego aplikacji, itp.

Kod początkowy:

```
<input name="debug" type="hidden" value="0">
```

Kod zmieniony:

```
<input name="debug" type="hidden" value="1">
```

Przepełnienie bufora: napastnik usuwa maksymalną długość wprowadzanych danych, aby usunąć po stronie klienta limit 10 znaków w polu ID użytkownika i próbować zastosować przeładowanie bufora.

Kod początkowy:

```
<input name="userid" type="hidden" maxlength="10">
```

Kod zmieniony:

```
<input name="userid" type="hidden">
```

Zwiększenie przywilejów: zmiana wartości ‘n’ na ‘y’ powodująca, stan, w którym aplikacja zwiększa przywileje dostępu do poziomu administratora.

Kod początkowy:

```
<input name="adminaccess" type="hidden" value="n">
```

Kod zmieniony:

```
<input name="adminaccess" type="hidden" value="y">
```

Środki kontrolno-zaradcze

- Zawsze należy sprawdzać dane wejściowe po stronie serwera. Nie należy polegać na sprawdzeniu ze strony klienta.

- Należy unikać pól ukrytych, używać pojedynczych tokenów sesji do wskazywania danych zmagazynowanych w cache po stronie serwera. Jeśli aplikacja wymaga sprawdzenia cech użytkownika, weryfikuje sesję plików cookie z tabelą sesji oraz wskazuje dane użytkownika w cache / bazie danych.
- Jeżeli nie ma możliwości wprowadzenia powyższych rozwiązań i konieczne jest użycie pól ukrytych, należy połączyć pary nazw i wartości w pojedynczy ciąg i dopisać tajny klucz (który nigdy nie pojawi się w danym formularzu) na końcu ciągu. Ciągiem nazywa się wychodzącą treść formularza. Jest dla niej generowany MD5, SHA lub podobny jednostronny hash nazywany "outgoing form digest" dodawany do formularza jako dodatkowe ukryte pole.
- Kiedy formularz zostaje odebrany przez serwer, pary nazw i wartości są ponownie łączone z tajnym kluczem tworząc przychodzącą treść formularza. Form digest przychodzącej treści formularza jest generowany i porównywany z zawartym w treści formularza. Jeżeli sumy kontrolne nie są identyczne, oznacza to, że ukryte pole zostało zmienione. Technika ta może być też stosowana w przypadku URL w celu uniemożliwienia manipulacji parametrami.

Manipulacja nagłówkiem http

Nagłówki HTTP wykorzystywane są do przekazywania danych z sieci klienta do serwera dla żądań HTTP i odwrotnie dla odpowiedzi HTTP.

Istnieje możliwość wprowadzenia kontroli nadchodzących nagłówków, ale w takich przypadkach należy pamiętać, że jeśli pochodzą one od klienta mogą być zmienione przez napastnika.

Jako przykład można zastosować nagłówek referencyjny, który zwykle zawiera URL strony, z której pochodzi żądanie. Istnieje możliwość sprawdzenia takiego nagłówka w celu weryfikacji, czy żądanie pochodzi z wiarygodnego URL (np. własnego), tak, aby przeszkodzić napastnikom zapisanie stron sieci, zmodyfikowanie formularzy i przesłanie ich z innego komputera.

Nie jest to jednak bezpieczny mechanizm, gdyż napastnik może zmodyfikować nagłówek referencyjny HTTP tak, aby wyglądał na pochodzący z wiarygodnej strony.

Środki kontrolno-zaradcze

- Nie należy polegać na nagłówkach bez dodatkowych mechanizmów ochronnych.

Manipulacje w URL

Formularze HTML mogą przedkładać swoje wyniki z zastosowaniem albo HTTP POST albo HTTP GET. W przypadku stosowania metody HTTP GET, wszystkie nazwy elementów i wartości formularza pojawiają się w ciągu zapytań URL, co daje szanse napastnikowi na łatwą manipulację wartościami lub próbę przekazania nieoczekiwanych danych.

Środki kontrolno-zaradcze

- Należy unikać używania parametrów w ciągu zapytań.
- Jeżeli parametry muszą być przedłożone do serwera, należy upewnić się czy towarzyszą im ważne tokeny sesji.
- Jeżeli parametru nie można usunąć z ciągu zapytań, należy go chronić kryptograficznie z zastosowaniem silnych algorytmów kryptograficznych.

Jest to możliwe za pomocą następujących metod:

- utajnianie całego ciągu zapytań,
- dodanie dodatkowego parametru w ciągu pytań, będącego sumą SHA-1. Nie zapobiega to przeglądaniu ciągu przez użytkownika, ale jeżeli aplikacja sprawdzi zwrócony hash i nie spełni żądań, w których hash nie pasuje, uniemożliwi ich zmianę i przedłożenie, odrzucając dane wprowadzone przez użytkownika.

2.10.9 Ujawnianie informacji i prywatność użytkownika

Napastnicy używają szeregu metod, aby uzyskać informacje, które mogłyby stanowić podstawę do przeprowadzenia ataku na witryny lub infrastruktury wspomagające.

Komendy po stronie klienta

Dodawanie i utrzymywanie komentarzy w kodzie źródłowym było standardową praktyką, usprawniającą późniejszy serwis. Praktyka ta ma zastosowanie do stron HTML, co w zależności od charakteru komentarzy może powodować ujawnianie wrażliwych informacji o strukturze witryny, jej podległej infrastrukturze albo członkach personelu. Komentarze często pozostawiane na stronach HTML zawierają nazwy serwera, błędy, struktury katalogów, adresy IP, zdebugowane informacje, nazwiska programistów, numery telefonów czy adresy emailowe.

Środki kontrolno-zaradcze

- Należy usuwać komentarze z kodu zanim zostaną przeniesione do usług produkcyjnych (oprócz dotyczących praw autorskich, licencji czy własności intelektualnej!).
- Należy upewniać się czy w procedurach zapewnienia jakości istnieje możliwość usunięcia wszystkich komentarzy przed przeniesieniem do produkcji.

Komendy debugowania

Często umieszcza się włączniki debugowania w HTML, aby umożliwić ich włączanie na dodatkowych poziomach logowania lub zgłaszania. Umieszczanie tego kodu (i logiki od strony serwera w celu interpretacji) w usługach produkcyjnych powoduje poważne zagrożenie, które zapewnia napastnikowi zwiększone przywileje dotyczące usług i podległej infrastruktury.

Środki kontrolno-zaradcze

- Należy usunąć wszelkie mechanizmy debugowania przed przeniesieniem aplikacji poza środowisko deweloperskie.
- Przed przeniesieniem do produkcji należy wykonać test tak, aby zapewnić usunięcie układu debugowania po stronie serwera.

Kody błędów

Niewłaściwa obsługa błędnego kodu umożliwia napastnikowi uzyskanie informacji niezbędnych do podjęcia ataku na aplikację sieci lub infrastrukturę wspomagającą. Mogą one zawierać:

- przepływ aplikacji,
- dodatkową informację serwera sieciowego,
- typ i wersję bazy danych,
- typ i wersję systemu operacyjnego,
- typ i wersję skryptu / języka programowania,
- fizyczne ścieżki,
- pliki otwarte do odczytu i do zapisu,
- nazwy, wartości, typy i cele zmiennych,
- segmenty kodu źródłowego skryptu i zapytań SQL,
- struktury baz danych i tabeli.

Środki kontrolno-zaradcze

- Należy unikać raportowania użytkownikowi komunikatów o błędach w systemach produkcji. Jeżeli są one jednak nieuniknione, muszą być odpowiednio zakodowane i nie mogą ujawniać informacji napastnikowi.
- W celu wychwytywania błędów dla wewnętrznej obsługi należy zapewnić właściwą rejestrację i logowanie.

Wyliczenie pliku / aplikacji

Jest to powszechna technika stosowana do identyfikacji aplikacji i plików, które mogą być podatne na wykorzystanie lub mogą stanowić podstawę ataku. Napastnicy poszukują:

- plików lub aplikacji wrażliwych,
- plików lub aplikacji ukrytych lub bez odnośników
- kopii lub plików czasowych.

Środki kontrolno-zaradcze

- Należy usuwać wszystkie pliki testowe z serwera sieci.
- Należy usuwać niechciane lub nieużywane pliki z serwerów.
- Należy wyszukiwać i usuwać kopie zapasowe i pliki tymczasowe.
- Należy blokować dostęp z zewnątrz do plików, które powinny pozostać na serwerze, ale użytkownik nie powinien mieć do nich dostępu.

Cache przeglądarki

Informacje wrażliwe często przechowywane są w pamięci cache przeglądarki i dostępne dla każdej osoby mającej dostęp do dysku twardego urządzenia (np. w komputerach biurowych, kawiarenkach internetowych czy w bibliotekach).

Środki kontrolno-zaradcze

- Aplikacje muszą przekazywać informacje wrażliwe wyłącznie zamierzonemu odbiorcy, tylko w przypadku kiedy jest to absolutnie konieczne.
- Jeśli to możliwe należy wcześniej wygaszać strony, które mogą zawierać wrażliwy materiał.
- Komenda “Pragma No-cache” na wszystkich stronach mogących zawierać materiał wrażliwy, informuje przeglądarki, że nie powinny przechowywać kopii stron.

Historia przeglądarki

Przeglądarki często zachowują historię ostatnio odwiedzonych witryn, które są podpowiadane, kiedy użytkownik zaczyna wprowadzać podobne URL. Adresy URL mogą często zawierać parametry, wykorzystane później do ujawnienia informacji, wystarczających do rozpoczęcia ataku.

Środki kontrolno-zaradcze

- Dane formularzy powinny być przekazywane z użyciem HTTP POST, ponieważ nie zostają dodane do URL. Nigdy z użyciem HTTP GET.

Autouzupełnianie

Przeglądarki internetowe obsługują funkcję Autouzupełniania. Dzięki niej dane wejściowe użytkowników mogą być zachowane dla przyszłego użycia i prezentowane użytkownikowi komputera po kliknięciu na pole formularza sieciowego z tą samą nazwą.

Jeżeli funkcja ta jest uruchomiona na komputerach wspólnych (w bibliotekach, biurach, kawiarenkach internetowych), informacja wprowadzana przez klientów do pól wejściowych (mogąca też zawierać dane osobowe czy finansowe), może być widzialna dla innych użytkowników korzystających z komputera.

Środki kontrolno-zaradcze

- Należy ostrzegać klientów o istnieniu funkcji i zalecać jej wyłączenie w przypadku korzystania z urządzeń wspólnych.
- Należy informować klientów, że funkcja zostaje włączona na wspólnie użytkowanych urządzeniach na ich własne ryzyko.
- Należy wyłączać funkcję w polach hasła/PIN.
- Należy wyłączać funkcję w polach kart i danych kont bankowych.
- Istnieje również możliwość całkowitego wyłączenia funkcji.
- Przechowywanie hasła i hasła zakodowane sprzętowo
- Poważne zagrożenie bezpieczeństwa powodować może włamanie do bazy danych, która przechowuje hasła.

Środki kontrolno-zaradcze

- Należy unikać przechowywania haseł, kodów PIN , itp. w postaci czystego tekstu, natomiast przechowywać hash hasła z użyciem jednostronnych algorytmów szyfrujących z użyciem pseudolosowej soli.
- Aby uniemożliwić przeglądarkom zapisywanie haseł, kodów PIN itp należy stosować formularze uwierzytelnienia (GAS)
- Jeżeli hasła bądź PINy muszą być przechowywane, w postaci umożliwiającej odtworzenie, należy zapewnić ich szyfrowanie przy użyciu silnych algorytmów szyfrujących, oraz zagwarantować bezpieczeństwo klucza szyfrującego.
- Edukacja użytkownika
- Nie każdy użytkownik komputera i Internetu jest ekspertem od bezpieczeństwa komputerów, w związku z tym wielu z nich nie rozumie, dlaczego bezpieczeństwo jest tak istotne.

Środki kontrolno-zaradcze

- Należy udzielać przemyślanych porad zatwierdzonych przez wydzielone komórki firmy oraz wykorzystywać aktualne informacje dostępne na stronach internetowych firmy.

Ukryte pola

Ukryte pola mogą być przydatne, jednak mogą też stanowić znaczące ryzyko dla aplikacji, jeżeli zostaną niewłaściwie wykorzystane do przechowywania wrażliwych informacji. Mogą być łatwo przejrzane, zmodyfikowane i odesłane przez napastnika.

Środki kontrolno-zaradcze

- Wartości, które mogą zostać użyte przez napastnika do uzyskania nieoczekiwanej odpowiedzi (względnie do otrzymania danych innej osoby albo wygenerowania warunku błędu mogącego stanowić podstawę do ataku) powinny być zawsze kodowane albo haszowane.
- Należy unikać przechowywania identyfikatorów sesji w tych polach.
- Nigdy nie należy przechowywać haseł ani PINów w ukrytych polach.
- Wszelkie dane osobowe (zdefiniowane w ustawie o ochronie danych osobowych) i informacje finansowe powinny być kodowane i przesyłane w szyfrowanej sesji SSL.
- Pola te powinny być zawsze rygorystycznie sprawdzane, po stronie serwera.
- Nigdy nie należy używać ukrytych pól do komend kontrolnych serwera sieci.

Historia konta

Użytkownicy aplikacji nie mogą sprawdzać, czy nieupoważnione osoby uzyskały dostęp do ich konta lub czy posługiwały się nim w sposób niewłaściwy.

Środki kontrolno-zaradcze

- Należy stosować wyświetlanie czasu ostatniego logowania, daty i adresu IP źródła po prawidłowym uwierzytelnieniu.
- Należy stworzyć szczegółową sekcję historii konta dla uwierzytelnionych użytkowników, obejmującą:
- odnotowany czas i datę,
- modyfikacje konta np. zmiana hasła,
- transakcje finansowe, itp.

Zgłaszanie incydentu

W przypadku pojawienia się podejrzanych zmian na koncie lub stronie użytkownika, musi on wiedzieć w jaki sposób zgłosić incydent firmie. Brak przejrzystej i prostej instrukcji niesie ryzyko nie zgłoszenia problemów.

Środki kontrolno-zaradcze

- Dostawcy powinni zachęcać użytkowników do zgłaszania incydentów oraz informować o sposobach kontaktu.
- Incydent powinien zostać zgłoszony do przełożonego liniowego, a ten powinien zgłosić go zgodnie ze ścieżką formalną do kierownika projektu lub/i stosownego dyrektora.

Informacje wrażliwe i kod źródłowy

Kod źródłowy od strony klienta jest łatwo zauważalny dla użytkowników. Wprowadzanie wrażliwych informacji zakodowanych sprzętowo do kodu źródłowego, może udostępnić napastnikowi informacje, które może on wykorzystać do przeprowadzenia ataku lub popełnienia oszustwa.

Środki kontrolno-zaradcze

- Nie należy kodować sprzętowo po stronie klienta informacji wrażliwych (identyfikatorów, haseł itp.).

Informacje wrażliwe i pliki cookie

Pliki cookie mogą być przeglądane i modyfikowane. Jeżeli zawierają informacje wrażliwe, mogą być wykorzystane do przeprowadzenia ataku lub popełnienia oszustwa.

Środki kontrolno-zaradcze

- Nie należy przechowywać danych osobowych ani informacji finansowych w plikach cookie.
- Nie należy przechowywać szczegółów uwierzytelnienia w plikach cookie.
- Jeżeli identyfikator sesji jest przechowywany w plikach cookie - należy zapewnić jego haszowanie.
- Zawartość plików cookie należy zabezpieczać przy pomocy bezpiecznych algorytmów szyfrujących.
- Aby zapobiec wysyłaniu przez przeglądarkę plików cookie przez nieszyfrowane połączenie - należy przeanalizować użycie etykiety bezpieczeństwa.

Kryptografia

Kryptografia służy do zapewnienia:

- poufności (dane są rozumiane wyłącznie przez upoważnione osoby)
- integralności (dane nie są zmienione w trakcie przesyłania)
- uwierzytelniania (dane pochodzą od określonej osoby)

Należy jednak pamiętać, że nie jest ona ostatecznym rozwiązaniem dla ochrony danych, a skomplikowaną funkcją kontrolną. Do listy problemów należy m.in:

- pozorne poczucie bezpieczeństwa,
- własne, niesprawdzone procedury kodowania,
- wykorzystanie niewiarygodnych i niepotwierdzonych procedur kodowania,
- odzyskanie systemu / danych,
- zarządzanie kluczami i ich odzyskiwanie,
- typ / moc algorytmu,
- długości kluczy,
- generowanie liczb kluczowych / losowych.

  Środki kontrolno-zaradcze Wdrażając kodowanie należy:

  - zapoznać się z wymaganiami firmy i bezpieczeństwa,
  - ściśle współpracować z technicznymi zespołami informatyki i bezpieczeństwa,
  - nie próbować samodzielnie opracowywać procedur kodowania,
  - nie wykorzystywać niezatwierdzonych lub niewiarygodnych procedur kodowania, tylko tych zaakceptowanych i zatwierdzonych,
  - dokumentować rozwiązania,
  - dokładnie testować rozwiązania (kodowanie, dekodowanie, odzyskiwanie),
  - zapewnić gruntowne sprawdzenie systemu zarządzania kluczami (manualnego lub informatycznego) oraz odpowiednie przeszkolenie personelu obsługi. Funkcjonować musi możliwość odzyskania zaszyfrowanych danych w celach dochodzeniowych,
  - stosować odpowiednie długości kluczy.

2.10.10 Kontrola dostępu

Podstawową kwestią ochrony systemów centralnych i końcowych oraz danych jest wprowadzenie odpowiedniej kontroli dostępu, którą należy stosować w celu:

- Ograniczenia możliwości działania użytkowników,
- Ograniczenia dostępu użytkowników do zasobów,
- Definicji funkcji, które użytkownicy mogą stosować do danych.

Mechanizmy kontroli dostępu powinny uniemożliwiać nieuprawnionym: przeglądania, modyfikowania i kopiowania danych. Dodatkowo, mogą powstrzymać przed zastosowaniem złośliwego kodu lub nieuprawnionych działań przez napastnika, wykorzystującego zależności infrastruktury.

Dostęp do informacji w systemach oraz dokumentacji a informacja publiczna

Udostępnieniu danych na podstawie "Ustawy o dostępie do informacji publicznym" nie podlegają informacje oraz kod źródłowy aplikacji zgromadzony w Bazie Wiedzy, Systemie Zgłoszeń, Repozytorium Kodu Źródłowego i w dokumentacji!

Wyżej wymienione repozytoria są objęte tzw. klauzulą informacji zastrzeżonej przedsiębiorstwa i nie powinny być udostępniane dla osób powołujących się na tą ustawę. W przypadku wstąpienia roszczącego na drogę sądową informacje te nie powinny zostać udostępnione bez prawomocnego wyroku sądu.

Użytkowanie sprzętu prywatnego

Aby zapewnić bezpieczeństwo środowiska pracy w organizacji regulamin polityki bezpieczeństwa Firmy zabrania używania urządzeń prywatnych do wykorzystania w celu służbowym w Firmie.

Odstępstwo od tej reguły może mieć miejsce tylko i wyłącznie za zgodą kierownika, architekta lub stosownego zastępcy Dyrektora i musi być dobrze umotywowane. W powyższych i uzasadnionych przypadkach muszą zostać spełnione obowiązujące w Firmie standardy bezpieczeństwa, w stopniu nie niższym niż te, dotyczące pracy na służbowych komputerach.

Na szczególną uwagę należy zwrócić aby:

- dysk musi być szyfrowany bezpiecznie,
- firewall musi być skonfigurowany,
- praca na użytkowniku musi się odbywać na użytkowniku pozbawionym praw administratora,
- system musi być bezpieczny, aktualny, wspierany przez producenta,
- w systemie nie ma malware'u (oprogramowanie antywirusowe w systemie Windows, chrootkit/debsums w Linux/*nix),
- nie przechowywanie danych/kopii zapasowych na zdalnych chmurach.

Klasyfikacja danych i autoryzacja dostępu

Dane mogą zostać niewłaściwie skontrolowane, a w efekcie bezprawnie ujawnione, w sytuacji, kiedy nie użyjemy klasyfikacji albo w przypadku, kiedy będzie ona niewłaściwa.

Bez efektywnej procedury uwierzytelniania i autoryzacji dostęp do danych lub systemu może zostać nieodpowiednio przyznany bez wiedzy właściciela systemu lub danych.

Środki kontrolno-zaradcze

- Wszystkie dane używane przez aplikacje muszą być sklasyfikowane zgodnie z zasadami stosowanymi przez grupę.
- Procedura autoryzacji musi być wprowadzona, regularnie przeglądana i udokumentowana.

Nieoczekiwany dostęp do zasobów

Napastnicy nie zawsze używają aplikacji w sposób zgodny ze sposobem ich funkcjonowania. Aby uzyskać dostęp do procedur, zasobów czy danych (zazwyczaj zamaskowanych przez układ logiczny aplikacji), próbują obejść wprowadzone zabezpieczenia aplikacji.

Środki kontrolno-zaradcze

- Należy zidentyfikować i udokumentować role i uprawnienia dostępu.
- Należy stosować zasadę najniższych możliwych uprawnień.

- Każdy chroniony zasób, przed udzieleniem dostępu, musi uwierzytelniać sesję użytkownika. Kiedy użytkownik składa zapytanie przez aplikację, oprócz odpowiedniej kontroli danych wejściowych, procedura powinna sprawdzać czy konto użytkownika ma uprawnienia do wykonania operacji zarówno w aplikacji, jak i bazie danych.

Ukryte zagrożenia lub dane wykorzystane w niewłaściwym celu na skutek nieodpowiedniej kontroli dostępu

Działania ochronne kluczowych zasobów, procedur lub danych bazujących na prostych technikach, np. przyjęciu konwencji nazywania plików czy ukrywanie plików i folderów, nie stanowią przeszkody dla napastników przed uzyskaniem do nich dostępu, o ile nie istnieje dodatkowa autoryzacja i kontrola. Większość profesjonalnych napastników korzysta z technik, które ujawniają takie zasoby.

Środki kontrolno-zaradcze

- Zawsze należy stosować odpowiednią kontrolę procedur, zasobów i danych oraz zadbać o stosowny poziom zabezpieczeń organizacyjnych.

Dostęp do kodu źródłowego

Ograniczenie dostępu do kodu źródłowego aplikacji rozwijanych w ramach Firmy ma na celu:

- poprawę bezpieczeństwa,
- zapewnienie braku możliwości wprowadzenia nieautoryzowanych zmian w kodzie źródłowym,
- kontrolę autoryzowanych zmian,
- możliwość śledzenia zmian w danych modułach i plikach.

Środki kontrolno-zaradcze

- Centralne repozytorium kodu źródłowego znajduje się na serwerze do którego dostęp jest kontrolowany. Zarówno część systemowa jak i aplikacyjna serwera repozytorium jest chroniona hasłem lub/i kluczem a uprawnienia są nadawane na podstawie przynależności do odpowiedniej grupy w katalogu użytkowników.

Serwer powinien pozwalać na nadanie uprawnień na minimum trzech poziomach:

- read-only - tylko do odczytu,
- read-write - odczyt i zapis,
- administrator - osoba nadająca uprawnienia, oraz kontrolująca proces.

Serwer powinien zapewniać separację pomiędzy projektami oraz repozytoriami i gałęziami (ang. branch) rozwojowymi w repozytoriach na podobnych zasadach jak powyżej.

Poszczególne projekty powinny odzwierciedlać strukturę projektową i być niedostępne dla osób nieprzydzielonych do danego projektu.

Dostęp fizyczny do kodu źródłowego

Aby zabezpieczyć się przed nieautoryzowanym dostępem fizycznym do kodu źródłowego Firma podjęła decyzję o wprowadzeniu procedur bezpieczeństwa oraz wprowadzenia sposobów ich egzekucji specjalnym rozporządzeniem dyrektora.

Do najczęstszych naruszeń bezpieczeństwa w zakresie fizycznego dostępu należą:

- publikacja w serwisach umożliwiających hostowanie kodu źródłowego tj. Github czy Bitbucket (nie dotyczy kodu objętego możliwością publikowania na Open Source - patrz odpowiedni załącznik),
- publikacja fragmentów kodu źródłowego w serwisach do wymiany snippetów np. Pastebin, Github,
- serwisy wymiany porad dotyczące kodu i problemów informatycznych tj. fora internetowe, Stackoverflow,

- publiczne komunikatory, których serwery należą do firm trzecich, tj. Google Hangouts, Facebook Messenger, HipChat (nie dotyczy usługi hostowanej na serwerach Firmy),
- wysyłanie fragmentów kodu źródłowego za pomocą poczty elektronicznej,
- kopiowanie plików, całego repozytorium lub dokumentów na dyskach przenośnych,
- fizyczne wynoszenie komputerów poza budynek firmy,
- przetrzymywanie danych na nieszyfrowanym nośniku, bez względu na fakt czy jest zamontowany na stałe czy wymienny,
- pozostawianie komputera na Open Space, lub w pomieszczeniach do których dostęp nie wymaga konieczności użycia karty dostępowej,
- pozostawienie komputera bez zablokowania go hasłem,
- automatyczna kopia zapasowa komputerów i składowanie danych na nieszyfrowanych dyskach.

Środki kontrolno-zaradcze Unikanie powyższych zagrożeń. Natychmiastowe zgłaszanie incydentów w wypadku zauważenia naruszenia, poprzez:

- osobiste, telefoniczne lub elektroniczne poinformowanie przełożonego liniowego o zaistniałym incydencie,
- zgłoszenie Incydentu w systemie ITSM Firmy,
- osobiście, telefonicznie lub elektronicznie poinformować o wystąpieniu incydentu odpowiedni zespół ds. bezpieczeństwa fizycznego / sieciowego.
- Egzekucja kary adekwatnej do naruszenia.

W przypadku konieczności zobrazowania problemu i poparcia go stosownym fragmentem dopuszcza się możliwość wklejenia zanonimizowanego fragmentu kodu:

- należy dołożyć wszelkich starań aby nie można było odczytać kontekstu kodu,
- należy dołożyć wszelkich starań aby kod był w miarę najkrótszy, tj. obrazował tylko i wyłącznie problematyczną linijkę / linijki, a nie większy zakres.
- kod przeznaczony do udostępnienia i który spełnia powyższe kryteria powinien być skonsultowany z kierownikiem projektu lub/i architektem.
- Przez Internet przekazuję ZASZYFROWANE informacje - nie odszyfrowane.
- Pracownicy Firmy powinni być świadomi, że użytkowanie tego zasobu jest monitorowane w celu ustalenia nieprawidłowych działań przy wykorzystaniu zasobów sieci.

Rejestracja zdarzenia

Rejestracja służy do zapisu zdarzeń podejmowanych przez użytkownika lub system, które później mogą zostać przejrzane oraz przeanalizowane. Rejestracją zdarzenia można posłużyć się do analizy problemu systemowego lub zagrożenia bezpieczeństwa. Rejestracja może:

- sygnalizować podejrzaną działalność,
- wykazać odpowiedzialność użytkownika poprzez śledzenie jego działań,
- dać możliwość rekonstrukcji zdarzeń po nieprawidłowym wykorzystaniu danych lub po wystąpieniu problemu,
- stanowić pomoc w postępowaniu sądowym.

Brak możliwości wykrycia i oceny skutków zagrożenia systemu

Brak należytych mechanizmów rejestracji zdarzenia w aplikacjach może powodować zmniejszenie możliwości weryfikacji obecności nieuprawnionej działalności i określania jej skutków dla systemów lub interesów firmy.

Środki kontrolno-zaradcze

- Podczas tworzenia procedur rejestracji zdarzeń należy uwzględnić takie kwestie jak:
- pliki rejestru muszą być sklasyfikowane według Polityki Bezpieczeństwa Systemów Teleinformatycznych,
- próby uwierzytelnienia np. (wy)logowanie, nieudane logowanie,
- próby autoryzacji, w tym czas, sukces/porażka, autoryzowany zasób lub funkcja, do których użytkownik żądający autoryzacji chciał uzyskać dostęp,
- funkcje administracyjne, takie jak: przeglądanie danych użytkownika, zarządzanie kontami, aktywacja lub deaktywacja rejestracji zdarzenia, itp.,
- rejestracja informacji debugowych nie może prowadzić do zapisania wrażliwych danych prywatnego konta użytkownika w rejestrze zdarzeń (np. haseł, czy kodów PIN),
- rejestry nie mogą usuwać istniejących zapisów bez ich skopiowania lub zarchiwizowania. Archiwa i kopie zapisów muszą być chronione i przechowane zgodnie z ich klasyfikacjami i celami,
- zawartości rejestrów mogą być ujawniane w uzasadnionych przypadkach tylko osobom mającym odpowiednią autoryzację właściciela systemu lub danych, którego system lub baza jest monitorowana,
- indywidualni użytkownicy nie mogą aktualizować ani usuwać pozycji w rejestrach zdarzeń. Pliki rejestru mogą być uaktualnione tylko przez serwis rejestracji zdarzenia,
- komunikaty sieciowe.

W sytuacji kiedy ocena ryzyka wskazuje na potrzebę jednoznacznego potwierdzenia działań wykonanych przez podmiot lub osobę, przeznaczone dla nich mechanizmy i rejestry muszą spełniać normy, które mają niepodważalną moc dowodową przed sądem.

2.11 Manifest Agile

Wytwarzając oprogramowanie i pomagając innym w tym zakresie, odkrywamy lepsze sposoby wykonywania tej pracy.

W wyniku tych doświadczeń przedkładamy:

- **Ludzi i interakcje** ponad procesy i narzędzia.
- **Działające oprogramowanie** ponad obszerną dokumentację.
- **Współpracę z klientem** ponad formalne ustalenia.
- **Reagowanie na zmiany** ponad podążanie za planem.

Doceniamy to, co wymieniono po prawej stronie, jednak bardziej cenimy to, co po lewej.

2.12 Scrum

2.12.1 Artefakty Scrum

Product Backlog

Product Backlog jest jedynym źródłem wymagań do projektu. Stanowi uporządkowaną, dostępną dla członków zespołu listę wszystkich elementów, które mogą być potrzebne w realizacji produkcji oprogramowania.

W szczególności, Product Backlog wymienia wszystkie:

- Cechy produktu,
- Wymagania,
- Usprawnienia,
- Poprawki.

Uporządkowanie elementów listy zależy od wartości, ryzyka, priorytetu oraz stopnia potrzeby realizacji danego elementu - im element jest wyżej na liście, tym szybciej rozpoczynają się prace nad nim. Większy nacisk kładziony jest na jego zrozumienie, zapewniany jest większy poziom szczegółowości oraz bardziej precyzyjnie szacowana jest przez zespół jego pracochłonność.

Product Backlog ewoluuje w trakcie trwania procesu wytwórczego w zakresie porządkowania, dodawania szczegółów, ponownego oszacowywania pracochłonności oraz aktualizowania elementów.

Nie rzadziej, niż co Sprint Review Meeting sprawdzane są w Product Backlogu postępy prac. Monitoruje się całkowitą pozostałą pracochłonność poszczególnych elementów Product Backlogu, porównując ją z wynikiem poprzednim oraz wyznacza się trend w pracy, prognozując datę końca produkcji.

SCRUM Team

Osoby zaangażowane w projekt tworzą samoorganizujący i wielofunkcyjny zespół, tzw. SCRUM Team, który składa się z:

- Product Owner - dba o maksymalizację wartości produktu i pracy Development Teamu, jest odpowiedzialny za zarządzanie Product Backlogiem, przygotowuje spotkania, ustala priorytety, prezentuje zadania Development Teamu, wyjaśnia wszelkie niejasności oraz ma prawo do anulowania Sprintu gdy np. jego cel stał się nieaktualny,
- Development Team - jest odpowiedzialny za pracę, dostarcza przyrosty funkcjonalności; jest samoorganizujący, wielofunkcyjny, bez struktury i podzespołów, składa się z 3-9 osób,
- SCRUM Master - wspiera organizację w dostosowaniu SCRUMa dla większej efektywności SCRUM Teamu oraz w pracy z innymi SCRUM Masterami, dba o SCRUM Team, czyli o rozumienie przez niego przebiegu procesu produkcji, długoterminowych planów i stosowania przyjętych zasad zwinnej produkcji. Wspiera Product Ownera w efektywnym zarządzaniu Product Backlogiem i przygotowywaniu spotkań. Wspiera Development Team w samoorganizacji, wielofunkcyjności i wytwarzaniu produktów o wysokiej wartości. Uczy go tworzyć elementy Product Backlogu, komunikuje wizje i cele tych elementów oraz usuwa wszelkie napotkane przeszkody.

Sprint

Sprint jest centralną częścią SCRUMa, trwającą poniżej miesiąca (zwykle od 5 do 10 dni). Stanowi zamknięty cykl wytwórczy, dający w wyniku działający prototyp produktu (wewnętrzny lub zewnętrzny), będący podzbiorem finalnego produktu, który rozrasta się z iteracji na iterację aż do produktu końcowego.

Ważne jest określenie celu danego Sprintu (tzw. Sprint Goal), który nadaje kierunek pracy zespołowi, pozostawiając jednocześnie elastyczność jeśli chodzi o sposób realizacji tego celu.

Członkowie zespołu powinni przestrzegać podstawowych zasad, tj.:

- nie dokonywać zmian wpływających na Sprint Goal,
- nie dokonywać zmian w Development Teamie,
- nie dokonywać zmian w standardach jakości,
- zakres prac może być uściślany i negocjowany tylko pomiędzy Product Ownerem a Development Teamem, i tylko na skutek przyrostu wiedzy.

Każdy Sprint dostarcza nowy, produkcyjnie gotowy przyrost funkcjonalności, a każdy przyrost tworzy całość ze wszystkim, co zostało dotychczas wytworzone.

Sprint Backlog

Sprint Backlog zawiera elementy Product Backlogu oraz plan dostarczenia przyrostu w Sprincie. Zarządza nim wyłącznie Development Team i aktualizuje go w czasie trwania Sprintu.

Tak jak w Product Backlogu, w Sprint Backlogu również sprawdzane są postępy prac, jednak nie rzadziej, niż co Daily SCRUM. Monitorowana jest całkowita pozostała pracochłonność poszczególnych elementów Sprint Backlogu i porównywana z wynikiem poprzednim oraz wyznaczany jest trend w pracy, szacujący prawdopodobieństwo osiągnięcia Sprint Goalu na koniec Sprintu.

Burndown Chart

Wykres takiego trendu określa się mianem Burndown (wykres wypalania). Zestawia on tempo pracy zespołu z pożądanym tempem pracy pozwalającym ukończyć Sprint lub cały projekt w terminie. Dzięki temu Burndown pozwala na pierwszy rzut oka ocenić sytuację dotyczącą realizacji zadań, w szczególności zdiagnozować ograniczenia.

Zdrowy Burndown Ponieważ wykres ma przedstawiać trend w pracy, na osi pionowej umieszcza się pozostałą do wykonania pracę, zaś na poziomej - czas. Pozostałą do wykonania pracę można mierzyć za pomocą sumy godzinowych oszacowań pozostałych zadań, liczby zadań lub w dowolny inny, odpowiedni do sytuacji sposób.

Burndown prognozujący nie zdążenie w terminie Jeśli na wykresie widać, że postępy pracy są wyraźnie wolniejsze, niż było to planowane, zespół powinien natychmiast poinformować o tym Właściciela Produktu i uzgodnić z nim, które zadania mają najwyższy priorytet, a które mogą zostać odłożone na później.

Burndown świadczący o zbyt długim utknięciu na zadaniach Jeśli wykres ma kształt "urwiska", czyli większość zadań jest domykana dopiero pod koniec Sprintu, może to świadczyć o tym, że praca dzielona jest na zbyt długie zadania. Redukuje to możliwość śledzenia aktualnej sytuacji w pracy nad projektem i stwarza zagrożenie, że zadanie się jeszcze bardziej wydłuży.

Może też swiadczyć to o tym, że zespół dostał zbyt wiele pracy do wykonania w danym czasie, co prowadzi do heroicznych wysiłków zespołu pod koniec Sprintu, a w rezultacie do natychmiastowej zapaści jakości i szybkiego wypalenia zepsołu.

W wyniku zastania powyższej sytuacji konieczne jest ustalenie priorytetów zadań.

Burndown świadczący o wzroście zakresu prac Jeśli wykres przez część Sprintu rośnie, zamiast maleć, może to świadczyć o:

- braku dobrze określonego zakresu sprintu podczas planowania,
- przypomnieniu sobie przez zespół o dodatkowych zadaniach,
- otrzymuje dodatkowych zleceń w trakcie trwania Sprintu,
- o napotkaniu nieprzewidzianych problemów technicznych.

Ponadto dobrym rozwiązaniem byłoby stosowanie eksperymentów technicznych przed przystąpieniem do tworzenia produktu. Konieczne skrócenie zadań lub weryfikacja ilości pracy oraz konieczność zadbania o lepszą ochronę zespołu, by dać mu swobodę sprawnej realizacji ustalonych zadań.

Definition of Done

Definition of Done (DoD), czyli definicja ukończenia jest wykazem działań wymaganych do realizacji zadań w procesie produkcyjnym. Działaniami takimi mogą być np. napisanie kodu, skomentowanie kodu, testowanie jednostkowe, testowanie zintegrowane, sporządzenie notatek, zaprojektowanie dokumentów, itp.

Dzięki określeniu DoD, wszyscy członkowie zespołu jednoznacznie rozumieją, co oznacza stwierdzenie „zadanie wykonane" (Done). Ponadto, zespół produkcyjny może skupić się na konkretnych elementach, które muszą zostać wykonane, aby zadanie zostało uznane za zrealizowane. W rezultacie, DoD pozwala dodać produktowi weryfikowalnych wartości, nie zmieniając jego funkcjonalności.

DoD określa się biorąc pod uwagę, jakie czynności realistycznie mogą zostać przez zespół wykonane. Z czasem lista tych czynności ulega zmianom i staje się bardziej rygorystyczna.

Skalowalność

SCRUM jest skalowalny, tzn. może być zastosowany w projektach, w których bierze udział duża ilość pracowników. Skalowanie zespołu powinno się odbywać ze względu na wymogi dotyczące funkcjonalności produktu, nie zaś ze względu na umiejętności członków zespołu.

Podział systemu na moduły i całego zespołu na mniejsze zespoły

Produkcja dużego systemu wymaga podzielenia jego architektury na mniejsze moduły (podsystemy). Podział ten odbywa się zgodnie z tym, jakie Klient dostrzega wartości w poszczególnych częściach systemu oraz zgodnie z możliwościami technologicznymi. Cały zespół również jest dzielony na mniejsze zespoły (Scrum Teamy), które zajmują się pracą nad przypisanymi im modułami systemu. Oprócz tych zespołów wydziela się również zespół poziomu systemu, w skład którego wchodzą architekci, liderzy zespołów, menedżerowie produktów oraz zespół zapewnienia jakości, który zajmuje się myśleniem, działaniem i wdrażaniem SCRUMa na poziomie systemu oraz uzupełnianiem Product Backlogu o testy integracyjne i demonstracje na poziomie systemu, punkty kontroli jakości oraz dystrybucje testowe.

W przypadku SCRUMa wprowadzanego na dużą skalę, pojawia się wymagany element w DoD: pomyślny wynik testów integracyjnych.

2.12.2 Wydarzenia Scrumowe

Backlog Refinement

Definicją tego spotkania jest wszelkiego rodzaju praca na backlogu, tj. np jego priorytetyzacja, dekompozycja zadań oraz czyszczenie rejestru zmian produktu. Podczas Refinementu zespół wraz z Product Ownerem, układa sobie pracę pod następne iteracje oraz dokonuje wstępnego oszacowania wielkości zadań oraz określenia kryteriów akceptacyjnych poszczególnych zadań.

Aby skutecznie przeprowadzić refinement należy ustalić co będzie produkowane - potrzeba uporządkowanego Product Backlogu, zrozumienia jego elementów, zrozumienia aktualnego stanu produktów, wyliczonej pojemności zespołu oraz wiedzy na temat historycznej wydajności zespołu.

Sprint Planning

Podczas planowania zespół wraz z właścicielem produktu podejmuje decyzję, które zadania będą wchodziły w skład następnej iteracji a co za tym idzie jakie funkcjonalności zostaną oddane po kolejnym przyroście.

W Sprint Planning Meetingu bierze udział cały SCRUM Team. Trwa, w zależności od czasu trwania Sprintu, od 2 do 4 godzin.

Celem tego spotkania jest zaplanowanie pracy na cały Sprint (w szczególności ustalenie Sprint Goalu), a więc ustalenie w jaki sposób zostanie to wyprodukowane - precyzuje się dużo drobnych zadań, członkowie zespołu sami wybierają zadania dla siebie i na każde z nich przeznacza się maksymalnie 2 dni.

Każdy etap projektu w każdym kolejnym Sprincie poddawany jest analizie szczegółowej, opartej o szeroki zakres informacji. Najważniejszym zadaniem tej analizy jest rozwiązywanie trudności i problemów. Jest to proces pracochłonny, lecz w wielu przypadkach konieczny do skutecznego usprawniania produkcji.

Sprint Review

Podczas spotkania Review zespół oddaje wykonane zadania właścicielowi produktu udowadniając spełnienie kryteriów akceptacyjnych każdego z zadań. Spotkanie to powinno zakończyć się akceptacją przyrostu oraz decyzją biznesową o wdrożeniu wytworzonych zmian.

Uczestnicy tego spotkania to SCRUM Team oraz interesariusze. Ma ono charakter nieformalny i trwa, w zależności od czasu trwania Sprinta, od 1 do 2 godzin.

Celem jest uzyskanie informacji zwrotnej od interesariuszy na temat zaprezentowanego przyrostu funkcjonalności (produktów).

Podczas Sprint Review Meetingu Product Owner prezentuje co zostało zrobione i co nie zostało zrobione, omawia Product Backlog oraz prezentuje prognozowane daty ukończenia produkcji, zaś Development Team przedstawia przebieg pracy - co nie sprawiło problemów, gdzie zostały napotkane problemy, jak te problemy zostały rozwiązane i jaki jest przyrost funkcjonalności oraz odpowiada na zadawane pytania. Wszyscy wspólnie uzgadniają temat dalszego planu działania, co jest wkładem do następnego Sprint Planning Meetingu.

Retrospective

Zamknięte spotkanie zespołu, który omawia problemy napotkane podczas właśnie zakończonej iteracji. Po retrospektywie zespół wyciąga wnioski z sukcesów oraz porażek. Planuje także eksperymenty, tj. nowe podejście do pracy lub zmiany w organizacji zespołu w przyszłej iteracji mające na celu usprawnienie procesu. Wnioski z takiego spotkania powinny być spisane i poddane do wiadomości członkom zespołu, ale nie ujawniane innym.

Udział w tym spotkaniu bierze cały SCRUM Team. Trwa ono, w zależności o czasu trwania Sprintu, od 1 do 2 godzin.

Cel spotkania to spojrzenie wstecz na wykonywaną pracę w celu jej polepszenia - praca powinna być coraz bardziej efektywna, a także dająca pracownikom satysfakcję. Cały zespół analizuje, jak przebiegał proces wytwórczy, jak ten proces usprawniały wykorzystywane narzędzia oraz jak kształtowały się relacje między pracownikami. Wszyscy dążą do konkluzji, jakie elementy pracy warto powtórzyć, a jakie można usprawnić. Sprawdzane jest też, czy wytworzone w Sprincie elementy oprogramowania należy zmodyfikować.

Daily Scrum

Do jednego z najważniejszych spotkań należy codzienny Scrum, tj. krótkie maksymalnie 15 minutowe zebranie zespołu podczas, którego członkowie opowiadają o problemach napotkanych przy realizacji zadań z poprzedniego dnia oraz o zamiarach na kolejną dobę. Na tym spotkaniu powinno się skupić na zadaniach przybliżających zespół do osiągnięcia tzw. celu sprintu, tj. najważniejszego motywu przewodniego iteracji.

W spotkaniach tych uczestniczy Development Team. Odbywają się one codziennie o tej samej porze, w tym samym miejscu i trwają 15 minut.

Daily SCRUMs mają na celu synchronizowanie pracy zespołu oraz ustalanie planu działania na następne 24h. Odbywa się to poprzez udzielenie odpowiedzi przez każdego członka zespołu na 3 pytania:

- co zrobił wczoraj,
- co będzie robił dzisiaj,
- jakie ma problemy.

Zaletą tych spotkań jest usprawnienie komunikacji, wykluczenie straty czasu na nieproduktywne, czasochłonne rozmowy oraz usunięcie potencjalnych przeszkód w pracy. Ponadto, Daily SCRUM promuje samodzielność i szybkie podejmowanie decyzji oraz wpływa na poprawienie świadomości postępu prac projektowych w zespole.

2.12.3 SCRUM of SCRUMs

Duża liczba SCRUM Teamów jest wyzwaniem koordynacyjnym i komunikacyjnym. Potrzeba też zapewnienia zintegrowania poszczególnych modułów, tak aby tworzyły jednolity docelowy system. W celu zapewnienia dobrej organizacji pracy wielu zespołów, organizowane są Eventy o nazwie SCRUM of SCRUMs. Uczestniczą w nich liderzy poszczególnych SCRUM Teamów.

Spotkania te odbywają się codziennie, najlepiej po zespołowych Daily SCRUMach. Każdy uczestnik odpowiada wtedy na 3 pytania: co zespół zrobił wczoraj, co zespół będzie robił dzisiaj, jakie zespół ma problemy.

2.13 Najczęściej używane technologie

2.13.1 Język i technologie

- Java 8
- AngularJS
- Spring
- REST + JSON
- Selenium

2.13.2 Platforma i środowisko

- Linux
- Puppet

2.13.3 Narzędzia developerskie

- Jira
- GIT
- Bitbucket
- Jenkins
- SonarQube
  - Squid
  - Checkstyle
  - Findbugs
  - PMD
  - Jacoco
  - PIT Test
- Artifactory
- Netbeans / Eclipse / InteliJ IDEA
- SonarLint

CHAPTER 3

Architecture

3.1 Reactive

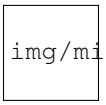

Fig.3.1.: Reactive System Traits

3.1.1 Reactive Systems

Responsive

The system responds in a timely manner if at all possible. Responsiveness is the cornerstone of usability and utility, but more than that, responsiveness means that problems may be detected quickly and dealt with effectively. Responsive systems focus on providing rapid and consistent response times, establishing reliable upper bounds so they deliver a consistent quality of service. This consistent behaviour in turn simplifies error handling, builds end user confidence, and encourages further interaction.

Resilient

The system stays responsive in the face of failure. This applies not only to highly-available, mission critical systems — any system that is not resilient will be unresponsive after a failure. Resilience is achieved by replication, containment, isolation and delegation. Failures are contained within each component, isolating components from each other and thereby ensuring that parts of the system can fail and recover without compromising the system as a whole. Recovery of each component is delegated to another (external) component and high-availability is ensured by replication where necessary. The client of a component is not burdened with handling its failures.

Elastic

The system stays responsive under varying workload. Reactive Systems can react to changes in the input rate by increasing or decreasing the resources allocated to service these inputs. This implies designs that have no contention points or central bottlenecks, resulting in the ability to shard or replicate components and distribute

inputs among them. Reactive Systems support predictive, as well as Reactive, scaling algorithms by providing relevant live performance measures. They achieve elasticity in a cost-effective way on commodity hardware and software platforms.

Message Driven

Reactive Systems rely on asynchronous message-passing to establish a boundary between components that ensures loose coupling, isolation and location transparency. This boundary also provides the means to delegate failures as messages. Employing explicit message-passing enables load management, elasticity, and flow control by shaping and monitoring the message queues in the system and applying back-pressure when necessary. Location transparent messaging as a means of communication makes it possible for the management of failure to work with the same constructs and semantics across a cluster or within a single host. Non-blocking communication allows recipients to only consume resources while active, leading to less system overhead.

3.2 Microservices

Todo: Przepisanie microservices jako osobny katalog z podziałem na tematy

Todo: Przepisanie microservices jako osobne szkolenie

Todo: Reactive manifesto, reactive programming

Todo: spock framework

Todo: Blockchain uruchamianie kodu

Todo: Function as a Service

Todo: Database sharding

Todo: Przykład edok w cloud i bazy po stronie klienta

Contents

- *Cechy systemów*
- *Choreografia > Orkiestracja*
  - *Hermes*
- *Przepisywanie architektury*
  - *Tworzenie nowej usługi*
- *Platforma uruchamiania*
  - *Mesos*
- *Monitoring*
- *Poziom organizacyjny*
  - *Domain Driven Design*
  - *Polyglog Programming i Polyglog Persistance*
  - *Ludzie*
  - *Handoff*
- *Wiązanie usług (coupling)*
- *Audyt i Compliance*
- *Microdata*
- *SLA usług*
- *Microservice testing*
- *Architecture*
  - *Monolithic architecture*
  - *Microservices architecture*
- *API*
  - *Cechy API*
  - *API gateway*
- *Service discovery*
  - *Client-side discovery*
  - *Server-side discovery*
- *Service registry*
- *Self registration*
- *3rd party registration*
- *Instancje*
  - *Multiple service instances per host*
  - *Single service instance per host*
  - *Service instance per VM*
  - *Service instance per Container*
- *Serverless deployment*
- *Baza danych*
  - *Database per Service*

3.2.1 Spojrzenie na Mikroserwisy z perspektywy biznesu

- szybkie wypuszczanie MVP nowych produktów
- ROI wdrożenia
- spójność systemów

3.2.2 Cechy rozproszonych systemów

- Niezależne domeny awarii
- Możliwość pisania w wielu językach
- Równoległość komponentów (concurrency)
- Brak globalnego zegara i możliwości jednoznacznego określenia czasu i kolejności

3.2.3 8 błędnych założeń

The Eight Fallacies of Distributed Computing – Peter Deutsch, 1991

- sieć jest niezawodna:
    - sieć w serwerowni jest niezawodna
    - MTBF routera jest 50k h
    - netsplit w publicznych cloudach są normalne
    - zwiększa się latency
- opóźnienia w sieci są zerowe
- przepustowość sieci jest nieskończona
- sieć jest bezpieczna:
    - większość aplikacji jest chroniona z zewnątrz
    - brak szyfrowania wewnątrz sieci
- Topologia sieci się nie zmienia:
    - przeliczenie BGP i zmiana spanning tree
    - ścieżki sieciowe się zmieniają
    - pojawiają się nowe instancje

- Istnieje tylko jeden administrator:
  - różni ludzie z różną wiedzą
  - inaczej konfigurują maszyny
  - jeden serwer może być lepiej skonfigurowany
- Koszt transportu jest zerowy
  - narzut czasowy na serializację, deserializację, stos TCP
  - czas transportu po medium jest niezerowy
- Sieć jest jednorodna
  - sieć składa się z różnych urządzeń
  - mogą być różnie stabilne
  - mogą mieć różne charakterystyki

3.2.4 Poświęcenie

- zapewnienie spójności kosztem dostępności
- zapewnienie wysokiej dostępności kosztem spójności
- wzajemnie się wykluczające
- nie ma ACID!:
  - Atomicity
  - Consistency
  - Isolation
  - Durability

Todo: dwufazowe kommity

3.2.5 BASE

- Basically:
  - Available (w większości możemy wykonać pewne operacje)
  - Soft State (tylko operacje, których stan możemy odbudować, np. przez przegenerowanie cache)
  - Eventually consistent (system jest pomiędzy stanem spójnym i niespójnym)

3.2.6 Cechy systemów

- Brak transakcyjności
- Zastosować mechanizm rekompensacji (np. raz w nocy usuwać zduplikowane dane)
- Brak gwarancji, że komunikat wysłany do hosta zostanie wysłany tylko raz (np. jeżeli dwa razy zostanie wysłany komunikat przez bankomat o naliczeniu opłaty, to operacja zostanie wykonana przez bank tylko raz)

3.2.7 Choreografia > Orkiestracja

- Choreografia:
  - informujemy system o zdarzeniu
  - system subskrybuje się do eventów
  - system reaguje na zmiany stanów
- Orkiestracja:
  - usługa jest odpowiedzialna za informację o zmianie stanu

Hermes

- https://github.com/allegro/hermes
- usługa subskrybuje się do danego topicu
- gdy zajdzie zdarzenie
- system wypycha je do subskrybentów
- nakładka na *Apache Kafka*
- zarządza dostarczaniem wiadomości *only once policy*
- throttling
- load ballancing
- security policy dla wiadomości

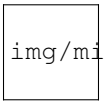

Fig.3.2.: Architektura systemu Hermes

- można zapchać sieć, gdy ma się zcentralizowaną infrastrukturę
- wykorzystanie HTTP/2.0 (multipleksowanie połączeń http, kompresja nagłówków, TLS)

Note: Jeżeli jedna usługa pada i to pociąga za sobą cały system, to nie jest to architekura *Microservices*.

3.2.8 Przepisywanie architektury

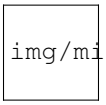

Fig.3.3.: Architektura systemu zgodna z Sidecar

- Anti Corruption Layer (ACL)
- Tworzenie nowych funckonalności na nowej platformie
- zapewnienie spójności systemów
- kontrola czy dane w nowym systemie są spójne z nowym
- przepisywanie całości

- wdrożenie ludzi:
  - zatrudnianie w nowej technologii
  - konwersja obecnych pracowników

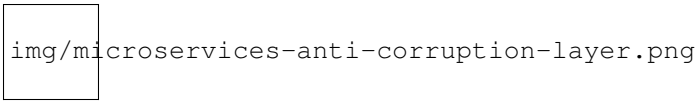

Fig.3.4.: Anti Corruption Layer

Tworzenie nowej usługi

- end to end:
  - założenie repo w Bitbucket
  - projekt w JIRA
  - CI/CD
  - Deployment
  - Repozytorium artefaktów
  - Publikowanie metryk
  - Testy security
  - Monitoring i logowanie
- *one-click-project*
- automatyzacja powtarzających się czynności za pomocą pluginów (*gradle* i *axion*)

3.2.9 Platforma uruchamiania

- Usługi uruchamiane w różnych datacenter jednocześnie
- Wykorzystanie public i private cloud jednocześnie

Mesos

- Tworznie logicznego klastra, który przykrywa infrastrukturę
- Możliwość dzielenia klastra na biznesowe komponenty i przydzielenia im zasobów
- Możliwość definiowania wykorzystywanych zasobów przez usługę
- Dynamiczne alokowanie zasobów

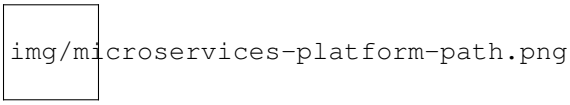

Fig.3.5.: Ścieżka rozwoju platform uruchomieniowych w architekturze mikrousługowej

3.2.10 Monitoring

- automatyczne zapinanie metryk do usług
- raportowanie poziomu SLA
- alerting
- definiowanie progów alertownia
- wykrywanie anomalii (na podstawie dotychczasowej historii, machine learning)

3.2.11 Poziom organizacyjny

- powiązania pomiędzy usługami
- przepływy danych

Domain Driven Design

- Poziom Strategiczny i Taktyczny
- Wzorce Strategiczne: Domain Distillation, Bounded Context
- nauka Product Ownerów
- DDD na poziomie strategicznym
- definicja corowych usług
- ułożenie biznesu i IT
- podział na domeny:
  - Lead PO dla domeny
  - Solutions Archtect pomiędzy domenami
- DDD na poziomie taktycznym do decyzji zespółu

Polyglog Programming i Polyglog Persistance

- overhead związany z wielością usług
- nowe technologie
- różne działające równoległe wersje np. baz danych
- Deprecation policy
  - Przykład Webapi
  - Przykład Visual Fox Pro -> Java
  - Przykład Twitter API

Ludzie

- poziom wiedzy jest nierówny
- różna wiedza na temat spójności systemów
- różne doświadczenie
- zmiana zespołów
- próg wejścia

- wdrożenie ludzi:
  - zatrudnianie w nowej technologii
  - konwersja obecnych pracowników
  - zmiana przyzwyczajeń
  - zmiana języka programowania i technologii
- Ludzie muszą testować
- Wymiana wiedzy pomiędzy ludźmi (eurowizja)
- Hackatony wdrożeniowe

Handoff

- ze względu na bardzo rozproszone środowisko ludzie uruchamiają swoje usługi sami
- duża i rozproszona wiedza na temat działania systemu
- utrzymywanie przez zespół
- przekazywania usług
- zmiany HRowe
- dyżury w każdym zespole

3.2.12 Wiązanie usług (coupling)

- zaprzecza systemowi wysyłania eventów
- ze względu na rozwój domen w różnym tempie pojawia się pokusa, aby obejść usługę i samemu zaimplementować funkcjonalność

3.2.13 Audyt i Compliance

- problemy z monitotowaniem
- problemy z rozproszoną wiedzą
- sprawdzanie czy wszystko się liczy poprawnie
- wyciąganie raportów i danych audytowych:
  - monolit - jedno zapytanie do bazy dancyh i joiny
  - microservices - dane są rozproszone (różne systemy, bazy danych, technologie)
- tworzenie audit logów
- przygotowanie systemu od początku pod audyty

3.2.14 Microdata

- eksport danych do Hadoopa
- normalizacja danych z różnych technologii i baz danych
- brak informacji na świecie jak to robić
- inny sposób dostępu do danych dla analityki (dostęp do miliardów rekordów po HTTP i API nie jest optymalny)
  - Replikacja baz danych

  - BSON
  - Protocol Buffers (Protobuf)
  - Trhift

3.2.15 SLA usług

- Definiowanie SLA
- Koszt inwestycji w zwiększenie dostępności np. z 4 na 5 dziewiątek)
- ROI z wprowadzenia poszczególnych usług
  - zmienjszone latency
  - większa stabilność
  - większa redundantność
- Każdy system może mieć inną charakterystykę i inne cechy

3.2.16 Microservice testing

- https://martinfowler.com/articles/microservice-testing/
- Historia ze stubami w dużym polskim telecomie

3.2.17 Architecture

- Duży próg wejścia:
  - Wymaga bardzo dobrego ekosystemu narzędziowego
  - Wymaga automatyzacji
  - Wymaga stworzenia i wdrożenia wielu różnych technologii
  - Tworzenie technologii, które skalują się horyzontalnie
  - Zmiana myślenia
  - Wdrożenie ludzi
- Dla większości firm nie przynosi to korzyści (sic!)
- SOA zrobiona porządnie (wywalone tematy związane z Enterprise)

Monolithic architecture

Build an application with a monolithic architecture. For example:

- a single Java WAR file.
- a single directory hierarchy of Rails or NodeJS code

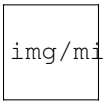

Fig.3.6.: Monolithic architecture

Microservices architecture

- Architect the application by applying the Scale Cube (specifically y-axis scaling) and functionally decompose the application into a set of collaborating services. Each service implements a set of narrowly, related functions. For example, an application might consist of services such as the order management service, the customer management service etc.
- Services communicate using either synchronous protocols such as HTTP/REST or asynchronous protocols such as AMQP.
- Services are developed and deployed independently of one another.
- Each service has its own database in order to be decoupled from other services. When necessary, consistency is between databases is maintained using either database replication mechanisms or application-level events.

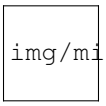

Fig.3.7.: Microservices Architecture

3.2.18 API

Cechy API

- Werisonowane
- Stabilne
- Deprecation policy
- HTTP
- REST
- JSON

Todo: przykład stabilności webapi i mobilnych stron

Todo: wersjonowanie w nagłówkach HTTP i q=...

Todo: http://allegro.tech/2015/01/Content-headers-or-how-to-version-api.html

---

Todo: POST, PUT, PATCH, GET, DELETE

API gateway

- Implement an API gateway that is the single entry point for all clients. The API gateway handles requests in one of two ways. Some requests are simply proxied/routed to the appropriate service. It handles other requests by fanning out to multiple services.
- Rather than provide a one-size-fits-all style API, the API gateway can expose a different API for each client. For example, the Netflix API gateway runs client-specific adapter code that provides each client with an API that's best suited to it's requirements.

- The API gateway might also implement security, e.g. verify that the client is authorized to perform the request
- Netflix API gateway, Zuur

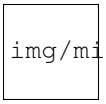

Fig.3.8.: Microservices API gateway

3.2.19 Service discovery

Client-side discovery

- When making a request to a service, the client obtains the location of a service instance by querying a Service Registry, which knows the locations of all service instances.
- Eureka is a Service Registry
- Ribbon Client is an HTTP client that queries Eureka to route HTTP requests to an available service instance

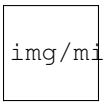

Fig.3.9.: Microservices client side discovery

Server-side discovery

- When making a request to a service, the client makes a request via a router (a.k.a load balancer) that runs at a well known location. The router queries a service registry, which might be built into the router, and forwards the request to an available service instance.
- AWS Elastic Load Balancer (ELB), Kubernetes, Marathon

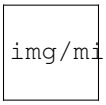

Fig.3.10.: Server side-discovery

3.2.20 Service registry

- Implement a service registry, which is a database of services, their instances and their locations. Service instances are registered with the service registry on startup and deregistered on shutdown. Client of the service and/or routers query the service registry to find the available instances of a service.
- Eureka, Apache Zookeeper, Consul, Etcd

3.2.21 Self registration

- A service instance is responsible for registering itself with the service registry. On startup the service instance registers itself (host and IP address) with the service registry and makes itself available for discovery. The client must typically periodically renew it's registration so that the registry knows it is still alive. On shutdown, the service instance unregisters itself from the service registry.

- Apache Zookeeper, Netflix Eureka

3.2.22 3rd party registration

- A 3rd party registrar is responsible for registering and unregistering a service instance with the service registry. When the service instance starts up, the registrar registers the service instance with the service registry. When the service instance shuts downs, the registrar unregisters the service instance from the service registry.
- Netflix Prana - a "side car" application that runs along side a non-JVM application and registers the application with Eureka.
- AWS Autoscaling Groups automatically (un)registers EC2 instances with Elastic Load Balancer
- Joyent's Container buddy runs in a Docker container as the parent process for the service and registers it with the registry
- Registrator - registers and unregisters Docker containers with various service registries
- Clustering frameworks such as Kubernetes and Marathon (un)register service instances with the built-in/implicit registry

3.2.23 Instancje

Multiple service instances per host

- Run multiple instances of different services on a host (Physical or Virtual machine).
- There are various ways of deploying a service instance on a shared host including:
- Deploy each service instance as a JVM process. For example, a Tomcat or Jetty instances per service instance.
- Deploy multiple service instances in the same JVM. For example, as web applications or OSGI bundles.

Single service instance per host

- Deploy each single service instance on it's own host

Service instance per VM

- Package the service as a virtual machine image and deploy each service instance as a separate VM

Service instance per Container

- Package the service as a (Docker) container image and deploy each service instance as a container
- Kubernetes, Marathon/Mesos, Amazon EC2 Container Service

3.2.24 Serverless deployment

- Use a deployment infrastructure that hides any concept of servers (i.e. reserved or preallocated resources)-physical or virtual hosts, or containers. The infrastructure takes your service's code and runs it. You are charged for each request based on the resources consumed.
- To deploy your service using this approach, you package the code (e.g. as a ZIP file), upload it to the deployment infrastructure and describe the desired performance characteristics.

- The deployment infrastructure is a utility operated by a public cloud provider. It typically uses either containers or virtual machines to isolate the services. However, these details are hidden from you. Neither you nor anyone else in your organization is responsible for managing any low-level infrastructure such as operating systems, virtual machines, etc.
- AWS Lambda, Google Cloud Functions, Azure Functions

3.2.25 Baza danych

Database per Service

- Keep each microservice's persistent data private to that service and accessible only via its API.

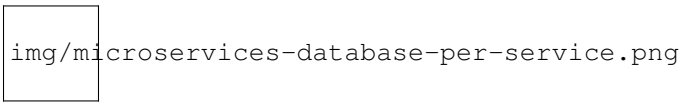

Fig.3.11.: Database per Service

Todo: Wiele baz danych w jednej usłudze

Todo: Mieszane, usługi mają jedną bazę danych

Shared database

- Use a (single) database that is shared by multiple services. Each service freely accesses data owned by other services using local ACID transactions.

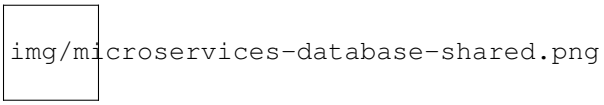

Fig.3.12.: Shared database

Database triggers

- Reliably publish events whenever state changes by using database triggers. Each trigger inserts an event into an EVENTS table, which is polled by a separate process that publishes the events.
- Czy są ok?
- Czym się różni struct od Class

3.2.26 Microservice chassis

- Build your microservices using a microservice chassis framework, which handles cross-cutting concerns
- Spring Boot, Spring Cloud, Dropwizard

3.2.27 Zdarzenia

Event-driven architecture

- Use an event-driven, eventually consistent approach. Each service publishes an event whenever it update it's data. Other service subscribe to events. When an event is received, a service updates it's data.

Event sourcing

- Reliably publish events whenever state changes by using Event Sourcing. Event Sourcing persists each business entity as a sequence of events, which are replayed to reconstruct the current state.

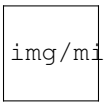

Fig.3.13.: Event sourcing

Application events

- Reliably publish events whenever state changes by having the application insert events into an EVENTS table as part of the local transaction. A separate process polls the EVENTS table and publishes the events to a message broker.

3.2.28 CQRS - Command Query Responsibility Segregation

- Split the system into two parts. The command side handles create, update and delete requests. The query side handles queries using one or more materialized views of the application's data.

3.2.29 Transaction log tailing

- Reliably publish events whenever state changes by tailing the transaction log.

CHAPTER 4

Cloud

4.1 Introduction

Contents

4.1.1 Wprowadzenie do technologii i platform chmurowych

- zapoznanie uczestników z podstawowymi pojęciami oraz standardami związanymi z chmurami obliczeniowymi,
- zapoznanie z dobrymi praktykami,
- wprowadzenie w problematykę projektowania rozproszonych aplikacji opartych na infrastrukturze chmury obliczeniowej,
- przybliżenie zagadnień związanych z usługami chmury Amazon Web Services. Szkolenie kładzie główny nacisk na architekturę i projektowanie systemów. Składa się z teoretycznych prezentacji dobrych praktyk i przykładowych systemów oraz praktycznych warsztatów z projektowania systemów. W programie są także ćwiczenia z podstaw konfiguracji i administracji kluczowymi usługami w chmurze – ćwiczenia te mają na celu zapoznanie od strony praktycznej z podstawowymi usługami i wprowadzenie w kontekst techniczny (jednak nie jest to szkolenie z administracji usługami AWS).
- Wprowadzenie do tworzenia aplikacji na Heroku i Google App Engine
- Porównanie platform Amazon AWS, Heroku, Google App Engine
- IaaS, PaaS, SaaS

4.1.2 Określenie potrzeb i wybór platformy

- PaaS, IaaS, SaaS
- jakie są dostępne platformy?
- jakie mam potrzeby?
- jak dobrać odpowiednią platformę do moich potrzeb?
- szczegółowa charakterystyka PaaS, Iaas, SaaS

Fig.4.1.: Cloud DevOps

4.1.3 Przykłady platform

IaaS

- Google Compute Engine
- Amazon AWS
- Rackspace
- ecloud24
- Open Stack

PaaS

- Github Pages
- Google App Engine
- Heroku

- Cloudera
- Open Shift

SaaS

- Force
- Google Apps

4.1.4 Inne *aaS

- Data
- Security
- Logging
- Payment

4.1.5 Ekosystem narzędziowy a cloud

- CI/CD: Travis, Bitbucket, CircleCI
- IM: Rocket, HipChat, Slack
- SCM: Github, Bitbucket
- Service Discovery
  - server side, client side
  - DNS, Amazon ELB, Route 53, Netflix Eureka (client side), własne rozwiązania
- Load Ballancing: Elastic Load Ballancers (AWS)
- Service Catallogue
- Authentication: OAuth
- Messaging: Kafka, Hermes

Distributed Logging

- Elastic Search
- Logstash
- Kibana

Hearthbeat detecting

- statsd + graphite (Grafana)
- pingdom

Monitoring

- new relic
- nagios
- zabbix
- tessera - dashboard statystyk z Graphite
- selena

Alerting

- cabot

Bazy danych

- Document: MongoDB
- RDBMS: PostgreSQL, MySQL, Oracle, MSSQL
- KV: Redis
- Graph: neo4j

Kontenery i wirtualizacja

- Vagrant
- Docker
- Rockit
- Mesos, Swarm, Kubernetes

Netflix

- chaos gorilla
- chaos monkey
- hystrix

Service Discovery

- DNS
- AWS Elastic Load Balancer
- Własne usługi

Configuration

- Zookeeper

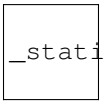

Fig.4.2.: Amazon AWS services overview

4.2 Amazon AWS

4.2.1 Produkty

- https://aws.amazon.com/products/

Compute

Nazwa	Opis
Amazon EC2	Virtual Servers in the Cloud
Amazon EC2 Container Registry	Store and Retrieve Docker Images
Amazon EC2 Container Service	Run and Manage Docker Containers
Amazon Lightsail	Launch and Manage Virtual Private Servers
Amazon VPC	Isolated Cloud Resources
AWS Batch	Run Batch Jobs at Any Scale
AWS Elastic Beanstalk	Run and Manage Web Apps
AWS Lambda	Run Your Code in Response to Events
Auto Scaling	Automatic Elasticity

Storage

Nazwa	Opis
Amazon S3	Scalable Storage in the Cloud
Amazon EBS	Block Storage for EC2
Amazon Elastic File System	Managed File Storage for EC2
Amazon Glacier	Low-Cost Archive Storage in the Cloud
AWS Storage Gateway	Hybrid Storage Integration
AWS Snowball	Petabyte-scale Data Transport
AWS Snowball Edge	Petabyte-scale Data Transport with On-board Compute
AWS Snowmobile	Exabyte-scale Data Transport

Database

Nazwa	Opis
Amazon Aurora	High Performance Managed Relational Database
Amazon RDS	Managed Relational Database Service for MySQL, PostgreSQL, Oracle, SQL Server, and MariaDB
Amazon DynamoDB	Managed NoSQL Database
Amazon ElastiCache	In-Memory Caching System
Amazon Redshift	Fast, Simple, Cost-Effective Data Warehousing
AWS Database Migration Service	Migrate Databases with Minimal Downtime

Networking

Nazwa	Opis
Amazon VPC	Isolated Cloud Resources
Amazon CloudFront	Global Content Delivery Network
Amazon Route 53	Scalable Domain Name System
AWS Direct Connect	Dedicated Network Connection to AWS
Elastic Load Balancing	High Scale Load Balancing

4.2.2 Regions

Code	Name
us-east-1	US East (N. Virginia)
us-east-2	US East (Ohio)
us-west-1	US West (N. California)
us-west-2	US West (Oregon)
ca-central-1	Canada (Central)
eu-west-1	EU (Ireland)
eu-central-1	EU (Frankfurt)
eu-west-2	EU (London)
ap-northeast-1	Asia Pacific (Tokyo)
ap-northeast-2	Asia Pacific (Seoul)
ap-southeast-1	Asia Pacific (Singapore)
ap-southeast-2	Asia Pacific (Sydney)
ap-south-1	Asia Pacific (Mumbai)
sa-east-1	South America (São Paulo)

4.2.3 AWS Command Line Interface (CLI)

- Access Key

```
brew install awscli

aws configure
aws ecr get-login
docker login -u AWS -p [...] -e none https://624006931819.dkr.ecr.eu-central-1.
↪amazonaws.com
docker build -t docker .
docker tag docker:latest 624006931819.dkr.ecr.eu-central-1.amazonaws.com/
↪docker:latest
docker push 624006931819.dkr.ecr.eu-central-1.amazonaws.com/docker:latest
```

```
FROM ubuntu
RUN echo 'ehlo world'
```

4.2.4 Tworzenie aplikacji w oparciu o platformę Amazon AWS

- Provisioning środowiska
- Tworzenie aplikacji
- Storage
- Cache

- Bazy danych
- Zarządzanie hostami
- Tworzenie reguł

4.2.5 Usługi w Amazon AWS

- 10 minute tutorials

EC2

ELB - Elastic Load Ballancer

- Czym jest ELB
- Jak działa
- Rodzaje Load Ballancerów
  - An Application Load Balancer makes routing decisions at the application layer (HTTP/HTTPS), supports path-based routing, and can route requests to one or more ports on each container instance in your cluster *[LB]*.

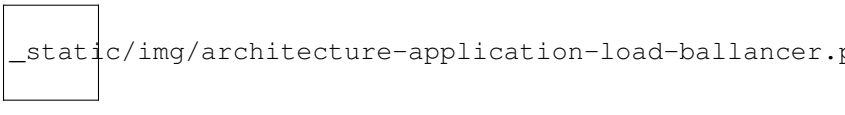

Fig.4.3.: Application Load Balancer *[LB]*

  - A Classic Load Balancer makes routing decisions at either the transport layer (TCP/SSL) or the application layer (HTTP/HTTPS) *[LB]*.

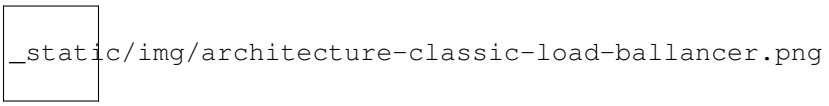

Fig.4.4.: Classic Load Balancer *[LB]*

- Application Load Ballancer
  - Application Load Balancers allow containers to use dynamic host port mapping (so that multiple tasks from the same service are allowed per container instance) *[LB]*.
  - Application Load Balancers support path-based routing and priority rules (so that multiple services can use the same listener port on a single Application Load Balancer) *[LB]*.

Amazon EC2 Container Service (ECS)

- Pozwalają na uruchomienie kontenerów *Docker* na platformie *Amazon EC2*

Lambda

- Uruchamianie funkcji w Cloud, bez konieczności stawiania środowiska
- Obsługiwane języki programowania

4.2.6 Zadania

Lambda functions

Stwórz w *Amazon AWS* lambda function który wyświetli *Ehlo World*.

Korzystanie z *Amazon AWS*

- Załóż konto na *Amazon AWS*
- Stwórz wolumen danych
- W panelu sterowania uruchom maszynę z poziomu *Free Tier* z *Ubuntu LTS AMI* z zamontowanym wolumenem
- W konfiguracji sieciowej maszyny ustaw możliwość połączenia z maszyną na portach:
  - 80
  - 443
  - 8080
  - 9000
  - 8081
- Jaki jest adres IP maszyny?
  - zewnętrzny
  - wewnętrzny
  - czym to się różni?
  - z którego korzystać?

4.3 Cloud Foundry

Todo: Sprawdzić: https://brooklyn.apache.org/v/latest/ops/locations/

Cloud Foundry is an open platform as a service (PaaS) that provides a choice of clouds, developer frameworks, and application services. Cloud Foundry makes it faster and easier to build, test, deploy, and scale applications.

- https://www.infoq.com/presentations/pcf-cd
- https://www.infoq.com/presentations/devops-cloud-foundry
- https://www.infoq.com/presentations/trends-cloud-foundry

Fig.4.5.: Cloud Foundry Architecture

Fig.4.6.: Cloud Foundry Architecture

4.3.1 Technology

Name	Description
cf	Cloud Foundry is an open platform as a service (PaaS)
cflinuxfs2-rootfs	System plików
diego	Diego is the new container runtime system for Cloud Foundry, replacing the DEAs and Health Manager.
garden-runc	Guardian is a simple single-host OCI container manager. It implements the Garden API which is used in Cloud Foundry.
bosh-aws-cpi	
bosh	

4.3.2 Bosh

BOSH is an open source tool chain for release engineering, deployment and lifecycle management of large scale distributed services. It's the default deployment and management platform for Cloud Foundry operators.

BOSH installs and updates software packages on large numbers of VMs over many IaaS providers with the absolute minimum of configuration changes.

- https://bosh.io/docs/bosh-components.html

The Director is the core orchestrating component in BOSH. The Director controls VM creation and deployment, as well as other software and service lifecycle events.

Stemcell

A stemcell is a versioned Operating System image wrapped with IaaS specific packaging.

- https://bosh.io/docs/stemcell.html

A typical stemcell contains a bare minimum OS skeleton with a few common utilities pre-installed, a BOSH Agent, and a few configuration files to securely configure the OS by default. For example: with vSphere, the official stemcell for Ubuntu Trusty is an approximately 500MB VMDK file. With AWS, official stemcells are published as AMIs that can be used in your AWS account.

By introducing the concept of a stemcell, the following problems have been solved:

- Capturing a base Operating System image
- Versioning changes to the Operating System image
- Reusing base Operating System images across VMs of different types
- Reusing base Operating System images across different IaaS

Release

A release is a versioned collection of configuration properties, configuration templates, start up scripts, source code, binary artifacts, and anything else required to build and deploy software in a reproducible way.

- https://bosh.io/docs/release.html

A release is the layer placed on top of a stemcell.

By introducing the concept of a release, the following concerns are addressed:

- Capturing all needed configuration options and scripts for deployment of the software
- Recording and keeping track of all dependencies for the software

- Versioning and keeping track of software releases
- Creating releases that can be IaaS agnostic
- Creating releases that are self-contained and do not require internet access for deployment

Deployment

A deployment is a collection of VMs, built from a stemcell, that has been populated with specific releases and disks that keep persistent data. These resources are created based on a manifest file in the IaaS and managed by the BOSH Director, a centralized management server.

```
---
name: redis-us

releases:
- name: redis
  version: 15.9 # <--- bump version

resource_pools:
- name: all-machines
  stemcell:
    name: bosh-aws-xen-ubuntu-trusty-go_agent
    version: 2972
  network: default
  cloud_properties:
    instance_type: m1.small
    availability_zone: us-east-1a
...
```

4.3.3 Bosh Lite

- https://github.com/cloudfoundry/bosh-lite

Create

```
git clone https://github.com/cloudfoundry/bosh-lite
cd bosh-lite
vagrant up
```

Upgrade

```
git pull
vagrant box update
vagrant destroy
vagrant up
```

4.3.4 CF Release

CF-Release is the BOSH release repository for the Cloud Foundry platform. To deploy Cloud Foundry, start with cf-release, build a custom manifest for your deployment, and let BOSH take over from there.

4.3.5 Diego

Diego is the new container runtime system for Cloud Foundry, replacing the DEA (Droplet Execution Agent) and Health Manager.

- https://github.com/cloudfoundry/diego-release

Cloud Foundry has used two architectures for managing application containers:

- Droplet Execution Agents (DEA)
- Diego.

With the DEA architecture, the Cloud Controller schedules and manages applications on the DEA nodes. In the newer Diego architecture, Diego components replace the DEAs and the Health Manager (HM9000), and assume application scheduling and management responsibility from the Cloud Controller.

- https://github.com/cloudfoundry/diego-release/tree/develop/examples/bosh-lite

From the standpoint of your application, here's what you need to know: In Diego, you now have the choice to push a one-use function (a Task) or a more traditional application that stays resident (a Long-Running Process, or LRP) – a good example of an LRP might be a web server that you need always listening for traffic, while a Task may be something like a database migration as part of a release, or a task that examines recent data for something specific. Before, in DEA, you really only pushed processes that were expected to stay resident. Diego's brain and health monitor makes sure these tasks are balanced as well as possible - spreading out CPU-intensive tasks across virtual machines, or balancing memory, et cetera. While before some of this was done as part of the cloud controller, now the Diego environment handles it itself.

Getting a bit further into the trees, pushing an application to Cloud Foundry using Diego would:

- Contacts the Diego Brain which immediately sets up Auctioneer to announce to the diego cells that there is a new task or LRP that needs to be added, and how many cells it should use.
- Lets the Converger know what the application expects to have running at any time, so that if there is a change, it can immediately set up a replacement.
- The Diego Cells run the task at hand, constantly updating the bulletin board system with necessary information (such as CPU usage) that allow the auctioneer and the converger to ensure the app is running according to plan. Diego uses etcd to handle the BBS.
- What isn't handled by the BBS is handled by Consul - this is mostly locks to make sure only the right process is handling the right task (as an example in Diego, there can be only one Auctioneer at any time, but if that Auctioneer goes away, something else must pick up the lock) or load balancing.
- Various other Diego-specific processes (Nsync, TPS, stager, and so forth) all exist as brokers to provide information from the cells to the right ingestors to ensure things are pushed in a safe manner, and information gets back to the right channels when things are not so safe.
- http://www.starkandwayne.com/blog/demystifying-cloud-foundrys-diego/

Droplet Execution Agents (DEA) -> Diego

- Warden -> Garden
- Health Manager (HM9000) -> nsync, BBS, and Cell Rep
- DEA Placement Algorithm -> Diego Auction
- Message Bus (NATS)

DEA architecture	Diego architecture	Function	Δ notes
Ruby	Go	Source code language	
DEA	Diego Brain	High-level coordinator that allocates processes to containers in application VMs and keeps them running	DEA is part of the Cloud Controller. Diego is outside the Cloud Controller.
DEA Node	Diego Cell	Mid-level manager on each VM that runs apps as directed and communicates "heartbeat", application status and container location, and other messages	Runs on each VM that hosts apps, as opposed to special-purpose component VMs.
Warden	Garden	Low-level manager and API protocol on each VM for creating, configuring, destroying, monitoring, and addressing application containers	Warden is Linux-only. Garden uses platform-specific Garden-backends to run on multiple OS.
DEA Placement Algorithm	Diego Auction	Algorithm used to allocate processes to VMs	Diego Auction distinguishes between Task and Long-Running Process (LRP) job types
Health Manager (HM9000)	nSync, BBS, and Cell Reps	System that monitors application instances and keeps instance counts in sync with the number that should be running	nSync syncs between Cloud Controller and Diego, BBS syncs within Diego, and Cell Reps sync between cells and the Diego BBS.
NATS Message Bus	Bulletin Board System (BBS) and Consulthrough HTTP, HTTPS, and NATS	Internal communication between components	BBS stores most runtime data; Consul stores control data.

4.3.6 Consul

Consul is a tool for service discovery and configuration. Consul is distributed, highly available, and extremely scalable.

- https://github.com/hashicorp/consul

Consul provides several key features:

- Service Discovery - Consul makes it simple for services to register themselves and to discover other services via a DNS or HTTP interface. External services such as SaaS providers can be registered as well.
- Health Checking - Health Checking enables Consul to quickly alert operators about any issues in a cluster. The integration with service discovery prevents routing traffic to unhealthy hosts and enables service level circuit breakers.
- Key/Value Storage - A flexible key/value store enables storing dynamic configuration, feature flagging, coordination, leader election and more. The simple HTTP API makes it easy to use anywhere.
- Multi-Datacenter - Consul is built to be datacenter aware, and can support any number of regions without complex configuration.

4.3.7 Guardian

A simple single-host OCI (Open Container Initiative) container manager.

- https://github.com/cloudfoundry/garden-runc-release

Components

- Gardeners Question Time (GQT): A venerable British radio programme. And also a test suite.
- Gardener: Orchestrates the other components. Implements the Cloud Foundry Garden API.
- Garden Shed: RootFS and volume management. Where stuff is kept in the garden.
- RunDMC: A tiny wrappper around RunC to manage a collection of RunC containers.
- Kawasaki: It's an amazing networker.

4.3.8 CLI - Command Line Interface

```
Before getting started:
  config    login,l      target,t
  help,h    logout,lo

Application lifecycle:
  apps,a        logs      set-env,se
  push,p        ssh       create-app-manifest
  start,st      app
  stop,sp       env,e
  restart,rs    scale
  restage,rg    events

Services integration:
  marketplace,m        create-user-provided-service,cups
  services,s           update-user-provided-service,uups
  create-service,cs    create-service-key,csk
  update-service       delete-service-key,dsk
  delete-service,ds    service-keys,sk
  service              service-key
  bind-service,bs      bind-route-service,brs
  unbind-service,us    unbind-route-service,urs

Route and domain management:
  routes,r        delete-route    create-domain
  domains         map-route
  create-route    unmap-route

Space management:
  spaces         create-space    set-space-role
  space-users    delete-space    unset-space-role

Org management:
  orgs,o       set-org-role
  org-users    unset-org-role

CLI plugin management:
  plugins           add-plugin-repo      repo-plugins
  install-plugin    list-plugin-repos

Commands offered by installed plugins:

Global options:
  --help, -h                        Show help
  -v                                Print API request diagnostics to stdout
```

Help

```
cf help
```

Deployment

```
cf push myapp -p <filename>.jar
cf app myapp
```

Scaling

```
cf scale myapp -i 2
```

Marketplace

```
cf marketplace |grep mysql

cf create-service p-mysql 100mb mydb
cf bind-service myapp mydb
cf restart myapp
```

4.3.9 Web Platform

- https://console.run.pivotal.io

Login

```
cf login -a api.run.pivotal.io
cf push myapp
```

4.3.10 PCF Dev

A lightweight Pivotal Cloud Foundry® (PCF) installation that runs on a single virtual machine (VM) on your workstation. PCF Dev is intended for application developers who want to develop and debug their applications locally on a PCF deployment.

- https://network.pivotal.io/products/pcfdev

4.3.11 Zadania

pcfdev

- Zainstaluj na czystym *Ubunutu* na *Vagrant* `pcfdev`

Bosh Lite

- Uruchom *Bosh Lite* na *Vagrant*

Deploy to local workstation

- Uruchom aplikację https://github.com/cloudfoundry-samples/spring-music CF lokalnie
- Podłącz aplikację do bazy danych *MySQL*

Tip: cf dev

Deploy to *Pivotal Web Services* (*PWS*)

- Uruchom aplikację https://github.com/cloudfoundry-samples/cf-sample-app-spring.git w *PWS*
- Podłącz aplikację do bazy danych *ElephantSQL*
- Przeskalują aplikację:
  - ilość instancji = 2
  - ilość ramu = 1 GB
  - ilość miejsca na dysku = 512 MB

Diego

- Uruchom *Diego* na *Bosh Lite* z poprzedniego zadania

4.4 Google App Engine

4.4.1 Tworzenie aplikacji w oparciu o platformę Google App Engine

- Tworzenie aplikacji
- Storage
- Cache
- Bazy danych

Tworzenie aplikacji

```
app.yaml
git clone https://github.com/GoogleCloudPlatform/appengine-guestbook-python.git
dev_appserver.py .
```

- http://localhost:8000 - admin console
- http://localhost:8080 - app

App.yaml for static pages

```
runtime: php55
api_version: 1
threadsafe: true

handlers:
 - url: /
   static_files: www/index.html
```

```
  upload: www/index.html

- url: /(.*)
  static_files: www/\1
  upload: www/(.*)
```

4.5 Google Cloud

4.6 Heroku

4.6.1 Tworzenie aplikacji w oparciu o platformę Heroku

- Tworzenie aplikacji
- Storage
- Cache
- Bazy danych

Set up

```
heroku login
```

Prepare the app

```
git clone https://github.com/heroku/python-getting-started.git
cd python-getting-started
```

Deploy the app

```
heroku create
git push heroku master
heroku ps:scale web=1
```

View logs

```
heroku logs --tail
```

Define a Procfile

```
web: gunicorn gettingstarted.wsgi --log-file -
web: python manage.py runserver 0.0.0.0:5000
```

Scale the app

```
heroku ps
heroku ps:scale web=0
heroku ps:scale web=1
```

Declare app dependencies

- `requirements.txt`

Run the app locally

```
heroku local web -f Procfile.windows
heroku local web
```

Push local changes

```
git commit -am "Changes"
git push heroku master
```

Provision add-ons

```
heroku addons:create papertrail
heroku addons
heroku addons:open papertrail
```

Start a console

```
heroku run python manage.py shell
heroku run bash
```

Define config vars

```
heroku config:set TIMES=2
heroku config
```

Provision a database

```
heroku addons
heroku config
heroku pg
heroku pg:psql
```

4.6.2 Backup

Create

```
heroku pg:backups:capture
```

Schedule

```
heroku pg:backups:schedule DATABASE_URL --at '02:00 UTC'
heroku pg:backups:unschedule DATABASE_URL
heroku pg:backups:schedules
```

Download

```
heroku pg:backups:url b001
heroku pg:backups:url
heroku pg:backups:download
```

Status

```
heroku pg:backups
heroku pg:backups:info b001
```

Delete

```
heroku pg:backups:delete b101
```

Restore

```
heroku pg:backups:restore b101 DATABASE_URL
heroku pg:backups:restore 'https://s3.amazonaws.com/me/items/mydb.dump' DB_URL
```

Visibility

```
heroku pg:psql -c "select * from pg_stat_activity where application_name = 'heroku-
↪postgres-backups'"
```

4.6.3 Buildpack

- co to?
- do czego to służy?
- jak to robić?

4.6.4 Zadania

Uruchamianie aplikacji

- Ściągnij repozytorium:
  - https://github.com/AstroMatt/esa-time-perception
- Załóż konto na *Heroku*
- Stwórz nową aplikację
- Dodaj remote *Heroku* do lokalnego repozytorium *GIT*
- Uruchom aplikację na *Heroku*
- Uruchom polecenie na platformie w cloud:

```
python manage.py migrate
```

- Zrób dump bazy danych

4.7 JClouds

Apache jclouds® is an open source multi-cloud toolkit for the Java platform that gives you the freedom to create applications that are portable across clouds while giving you full control to use cloud-specific features.

4.7.1 Prividers

- https://jclouds.apache.org/reference/providers/

4.7.2 Installation

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/
↪2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0␣
↪http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <properties>
    <jclouds.version>2.0.1</jclouds.version>
  </properties>
  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <version>1.0-SNAPSHOT</version>
  <dependencies>
    <dependency>
        <groupId>org.apache.jclouds</groupId>
        <artifactId>jclouds-all</artifactId>
        <version>${jclouds.version}</version>
      </dependency>
  </dependencies>
</project>
```

4.7.3 Examples

- https://github.com/jclouds/jclouds-examples

4.8 OpenShift

4.9 OpenStack

CHAPTER 5

Appendices

5.1 Glossary

5.2 Copyright

5.2.1 MIT License

Bibliography

[LB] http://docs.aws.amazon.com/AmazonECS/latest/developerguide/service-load-balancing.html