Politechnika Śląska
Wydział Matematyki Stosowanej
Kierunek Informatyka
Studia stacjonarne I stopnia

Projekt inżynierski

Automatyzacja procesów w przemyśle IT

Kierujący projektem:
dr inż. Zdzisław Sroczyński

Autorzy:
Artur Kasperek
Michał Płonka
Patryk Musiol

Gliwice 2020

Projekt inżynierski:

Automatyzacja procesów w przemyśle IT

kierujący projektem: dr inż. Zdzisław Sroczyński

autorzy: Artur Kasperek, Michał Płonka, Patryk Musiol

Podpisy autorów pracy

.....

.....

.....

Podpis kierującego projektem

.....

Oświadczenie kierującego projektem inżynierskim

Potwierdzam, że niniejszy projekt został przygotowany pod moim kierunkiem i kwalifikuje się do przedstawienia go w postępowaniu o nadanie tytułu zawodowego: inżynier.

Data Podpis kierującego projektem

Oświadczenie autorów

Świadomy/a odpowiedzialności karnej oświadczam, że przedkładany projekt inżynierski na temat:
Automatyzacja procesów w przemyśle IT
został napisany przeze mnie samodzielnie.
Jednocześnie oświadczam, że ww. projekt:

- nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 roku o prawie autorskim i prawach pokrewnych (j.t. Dz.U. z 2018 r. poz. 1191, z późn. zm.) oraz dóbr osobistych chronionych prawem cywilnym, a także nie zawiera danych i informacji, które uzyskałem/am w sposób niedozwolony,
- nie była wcześniej podstawą żadnej innej urzędowej procedury związanej z nadawaniem dyplomów wyższej uczelni lub tytułów zawodowych.
- nie zawiera fragmentów dokumentów kopiowanych z innych źródeł bez wyraźnego zaznaczenia i podania źródła,
- złożona w postaci elektronicznej jest tożsama z pracą złożoną w postaci pisemnej.

Podpisy autorów pracy

Artur Kasperek, nr albumu:1234, ...(podpis:).....
Michał Płonka, nr albumu:1234, ...(podpis:).....
Patryk Musiol, nr albumu:1234, ...(podpis:).....

Gliwice, dnia .....

Spis treści

Wstęp

TODO dodać wstęp

1. Automatyzacja a branża IT

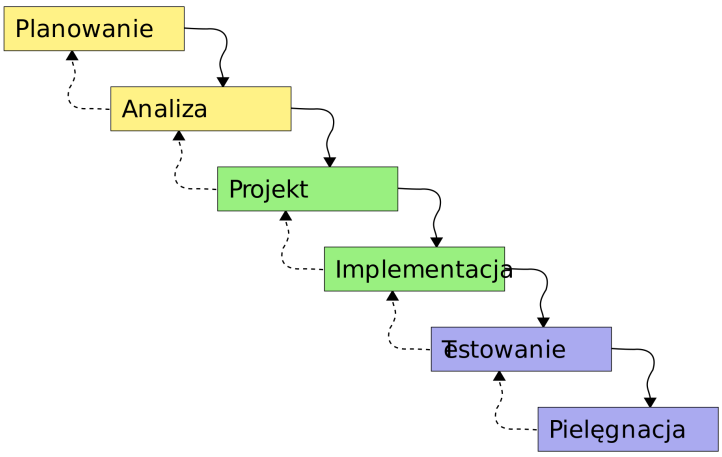
1.1. Charakteryzacja branży IT

W 2001 roku doszło do jednej z bardziej znaczących publikacji dla szerokopojętego biznesu IT. W roku tym został opublikowany *Manifesto for Agile Software Development* autorstwa między innymi Kenta Becka, Roberta C. Martina oraz Martina Fowlera. Manifest ten opisywał rewolucyjne jak na tamte czasy praktyki [1]:

- Satysfakcja klienta dzięki wczesnemu i ciągłemu dostarczaniu oprogramowania.
- Zmiany wymagań mile widziane nawet na późnym etapie programowania.
- Dostarczanie działającej wersji oprogramowanie często (bardziej tygodnie niż miesiące).
- Bliska kooperacja między programistami a ludziami zajmującymi się biznesem
- Projekty powstają wokół zmotywowanych osób, którym należy ufać.
- Komunikacja w cztery oczy jest najlepszą formą komunikacji
- Działający produkt jest najlepszym wskaźnikiem postępu prac.
- Zrównoważony rozwój, pozwalający na utrzymania stałego tempa.
- Ciągła dbałość o doskonałość techniczną i dobry design.
- Prostota - sztuka projektowania systemu bez dużej komplikacji systemu.
- Najlepsze architektury, wymagania i designy powstają dzięki samoorganizującym się zespołom
- Zespół regularnie zastanawia się, jak zwiększyć skuteczność, i odpowiednio się dostosowuje.

Propozycje przedstawione przez autorów manifestu były dużą zmianą w stosunku do podejścia które było używane powszechnie w tamtych czasach.

W latach 80 oraz 90 powszechnie stosowaną techniką była metodologia Waterfall zobrazowana na Rysunku 1. Poszczególne etapy projektowe były wykonywane tylko

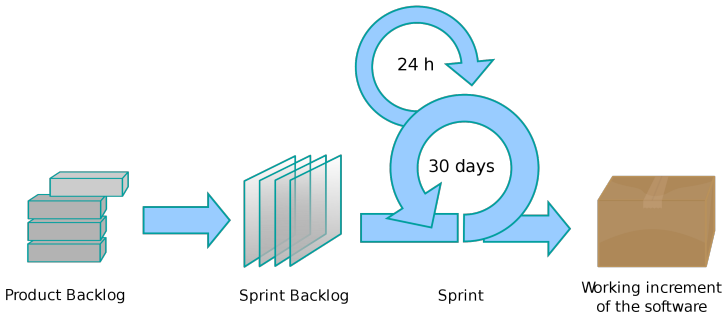


Rysunek 1: Metodologia Waterfall

raz podczas procesu tworzenia oprogramowania. Z tego faktu wszelakie zmiany na późniejszym etapie projektowym były trudne w realizacji. Konkurencja na rynku oprogramowania komputerowego była na tyle niewielka, że producenci oprogramowania nie musieli przejmować się zanadto uwagami od użytkowników - to sprawiało, że Waterfall spełniał swoje zadania.

Pierwsze dziesięciolecie XXI wieku spopularyzowało jedno z największych osiągnięć ludzkości - Internet. Nowa rzeczywistość w której ludzkość coraz więcej czasu spędza przed urządzeniami elektronicznymi postawiła przez twórcami oprogramowania nowe wymagania. Dodatkowo coraz większe grono przedsiębiorców zaczyna dostrzegać w produkcji oprogramowania zyski. Użytkownicy zaczynają coraz bardziej spoglądać na przyjemny dla oka wygląd oprogramowania oraz jego prostotę. Metodyka zaproponowana przez autorów manifestu Agile zdaje się świetnie wpisywać w wizję tworzenia oprogramowania na miarę XXI wieku.

Jedną z bardziej znanych implementacji Agile jest metodologia o nazwie Scrum. Zakłada się w niej, że oprogramowanie powstaje w procesie kolejnych inkrementacji. Każda iteracja jest nazywana sprintem. Sprint ma z góry zdefiniowane ramy czasowe w których będzie on trwał. Na podstawie różnych czynników biznesowych



Rysunek 2: Framework Scrum

opiekun projektu decyduje które zadania powinny trafić do danego sprintu, a które są mniej priorytetowe i mogą pozostać w tzw. backlogu. Efektem końcowym danego sprintu jest działający produkt, który jest wzbogacony o rzeczy dodane podczas trwania sprintu. Scrum sam w sobie nie narzuca ile powinien trwać dany sprint, czy też jaki system powinien być stosowany do śledzenia zadań. Wszystko zależy od preferencji danego zespołu programistycznego. Integralną częścią każdego sprintu jest retrospektywa. Na tym spotkaniu zespół dyskutuje jakie zmiany należy dokonać w procesie by uefektywnić pracę. Dzięki elastycznemu podejściu i możliwości ulepszania procesu Scrum wydaje się być dobrym rozwiązaniem dla zespołów które wypuszczają oprogramowanie regularnie oraz zmieniają je na podstawie opinii użytkowników.

1.2. Agile a automatyzacja

Spełnienie wymagań wymienionych w manifeście Agile wydaje się być trudne w kontekście częstego wypuszczania działającej wersji. Oczekuje się tego by regularnie zespół deweloperski publikował działającą wersję podglądową oprogramowania dla osób nietechnicznych. Problem ten można rozwiązać na conajmniej dwa sposoby:

- Manualny - Członek zespołu deweloperskiego regularnie według wymagań zajmuje się budowaniem wersji podglądowej oraz udostępnia ją osobom zainteresowanym
- Automatyczny - Zespół deweloperski ustawia automatyczne procesy które na serwerze budującym tworzą wersję podglądową aplikacji oraz publikują ją dla

osób zainteresowanych

Proces automatyczny jest preferowanym sposobem publikacji oprogramowania. Ma on kilka zalet nad sposobem manualnym. Nie tracimy czasu specjalisty który musiałby poświęcić go na zbudowanie i publikację aplikacji. Drugą zaletą jest fakt, że serwer za każdym razem robi te same kroki podczas procesu budowania. Tym sposobem wykluczamy możliwość popełnienia błędu przez człowieka.

Dobre praktyki związane z częstym budowaniem podglądowej wersji oprogramowania są określane jako DevOps. Len Bass, Ingo Weber oraz Liming Zhu w swojej książce [2] określają DevOps jako zbiór praktyk których celem jest zmniejszenie czasu publikacji zmian na serwerze produkcyjnym przy jednoczesnej trosce o wysoką jakość. W praktyce często członkiem zespołu deweloperskiego jest tzw. DevOps. Jego zadaniem jest automatyzacja wszelakich procesów oraz często także utrzymanie środowiska produkcyjnego. Osoba na tym stanowisku powinna się cechować dobrą znajomością systemu operacyjnego który jest używany na serwerach produkcyjnych oraz deweloperskich. Ponadto powinna być zorientowana w różnych rozwiązaniach chmurowych które współcześnie są coraz częściej używane.

1.3. Git - kamień milowy dla deweloperów

Ciężko byłoby sobie wyobrazić obraz dzisiejszego przemysłu IT gdyby nie system kontroli wersji Git. Jego autorem jest Linus Torvalds. Torvalds stworzył go jako dodatkowy projekt który miał pomóc w pisaniu jądra Linuxa. Dzięki Git'owi każdy członek zespołu deweloperskiego ma dostęp do wspólnego repozytorium gdzie każdy może publikować swoje zmiany. Jedną z ważniejszych funkcji Git'a jest możliwość tworzenia własnych rozgałęzień kodu gdzie dany programista wysyła swoje zmiany. Dalej w procesie merge'owania jest możliwe połączenie zmian danego dewelopera z kodem innych programistów. Dzięki tej cesze Git nadaje się świetnie do wszelakich projektów programistycznych w których pracuje kilku programistów równolegle. Z biegiem lat Git stał się standardem.

Opisać git workflow TODO!!

W obecnych czasach popularne są rozwiązania SaaS takie jak GitHub, GitLab czy też BitBucket. Dzięki takiemu rozwiązaniu nie musimy się przejmować utrzymaniem własnego serwera Git, dodatkowo platformy SaaS zapewniają nam wszelakie aktualizacje które usprawniają system. Platformy takie jak GitHub zapewniają

narzędzie które ułatwiają proces tworzenia oprogramowania. Narzędzie te są dopasowane by działać dobrze z naszym repozytorium. Przykładem takiego rozwiązania jest *GitHub Pages*, jest to technologia która pozwala publikować stronę www na podstawie plików które są częścią repozytorium. Użytkownik definiuje na której gałęzi oraz w którym folderze znajdują się pliki z stroną internetową. Od tego momentu GitHub automatycznie stwarza nam stronę internetową dostępną pod subdomeną *nazwauzytkownika.github.io*.

1.4. Ciągła integracja

1.5. Ciągłe dostarczanie

1.6. Ciągłe dowożenie

1.7. Serwery automatyzujące PaaS kontra self-hosted

1.8. GitOps - czym jest?

2. Wirtualizacja i orkiestracja

myślę, że opisanie tego jak wirtualizacja pomaga w CI/CD zasługuje na swój rozdział. Szczególnie mógłbym ten rozdział poświęcić dockerowi z faktu, że zdominował rynek. Warto tutaj byłoby napisać też o rozwiązaniach które były używane w przeszłości a zostały wyparte jak vagrant. Dodatkowo w rozdziale mozemy opisać na czym polega orkiestracja - Kubernates, docker swarm

3. Hudson oraz Jenkins - klasyczne narzędzia do CI/CD

tutaj mógłbym zamieścić trochę informacji historycznych jak język java zrewolucjonował inżynierię oprogramowania i dalej rozwinąć, że pokłosiem tego było powstanie Jenkinsa. Myślę, że mógłbym tutaj opisać najbardziej uniwersalne rzeczy które są w nim używane. Wydaje mi się że mógłbym poszukać alternatyw do Jenkinsa by rozszerzyć ten dział. Jako część tego rozdziału mógłbym zawrzeć problem gdzie konieczne jest użycie Jenkinsa i to byłoby zrobione jako część praktyczna. Wbrew pozorom do niektórych zastosować Jenkins nadal jest używany, np do środowisk gdzie QA musi odpalać środowisko z różnymi parametrami

4. Platformy SaaS z wbudowanym CI/CD

tutaj mógłbym rozwinąć rozdział 3 i powiedzieć, że coraz mniej używa się Jenkinsa i obecnie popularne są platformy jak Github Actions, CircleCI czy gitlabCI, które są trigerrowane przez commity, pushe czy też tworzenie tagów z pomocą systemu wersji git. Myślę, że mógłbym tutaj zrobić nawiązanie do części praktycznej pracy: 4a) continues delivery w React Native na przykładzie GitHub actions 4b) continues deploymeny na przykładzie aplikacji backendowej w javaScriptcie 4c) continues deployment strony internetowej hostowanej przez GitHub actions za pomocą circleCI W każdym z tych podrozdziałów opisałbym jaki jest ogólny problem i dalej opisałbym już konkretne rozwiązanie

5. Testy a continues integration

mógłbym tutaj powiedzieć więcej co to są testy jednostkowe, integracyjne oraz e2e. Następnie mógłbym powiedzieć trochę więcej, że dzięki poprawnemu CI nasza aplikacja będzie miała mniej błędów oraz będzie trudniej o regresję. Tutaj mógłbym nawiązać do części praktycznej i opisać problem w którym przydają się testy e2e. Myślę, że mógłbym te testy napisać w niejakim frameworku cypress.js i skonfigurować to na GitHub actions,

6. Podsumowanie

myślę, że cały wydźwięk podsumowania powinien być nastawiony na to że dzięki CI/CD programiści mogą efektywnie działać, ich kod szybko znajduje się na wersji produkcyjnej i dzięki odpowiednio skonfigurowanemu workflow mogą być bardziej pewni, że ich zmiany nie zepsują istniejących funkcjonalności

Literatura

[1] Kent Beck; James Grenning; Robert C. Martin; Mike Beedle; Jim Highsmith; Steve Mellor; Arie van Bennekum; Andrew Hunt; Ken Schwaber; Alistair Cockburn; Ron Jeffries; Jeff Sutherland; Ward Cunningham; Jon Kern; Dave Thomas; Martin Fowler; Brian Marick (2001). "Principles behind the Agile Manifesto"

[2] Bass, Len; Weber, Ingo; Zhu, Liming (2015). DevOps: A Software Architect's Perspective