Techniczne aspekty w pracy Scrum Mastera

Paweł Lasek

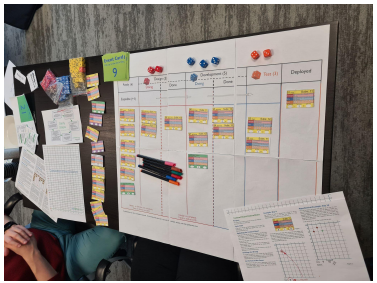
O mnie

Paweł Lasek

Projekty i doradztwo dla firm
w branżach/domenach:

- Ubezpieczenia
- Telekomunikacja
- Media
- Finanse
- Bankowość
- Cloud, Big Data
- R&D / Startupy / Nowe technologie

Agile Transformation Consultant
Agile Coach, Scrum Master
Scrum and Kanban Trainer
Background techniczny:
Programista, Developer



Agile Manifesto (Snowbird, Utah 2001)

Odkrywamy nowe metody **programowania** dzięki praktyce w **programowaniu** i wspieraniu w nim innych.

W wyniku naszej pracy, zaczęliśmy bardziej cenić:

- **Ludzi i int** ów i narzędzi
- **Działające oprogramowanie** od szczegółowej dokumentacji
- **Współpracę z klientem** od negocjacji umów
- **Reagowanie na zmiany** od realizacji założonego planu.

VA: ~~**oprogramowanie**~~ **produkty / usługi**

Oznacza to, że elementy wypisane po prawej są wartościowe, ale większą wartość mają dla nas te, które wypisano po lewej.

https://agilemanifesto.org/iso/pl/manifesto.html

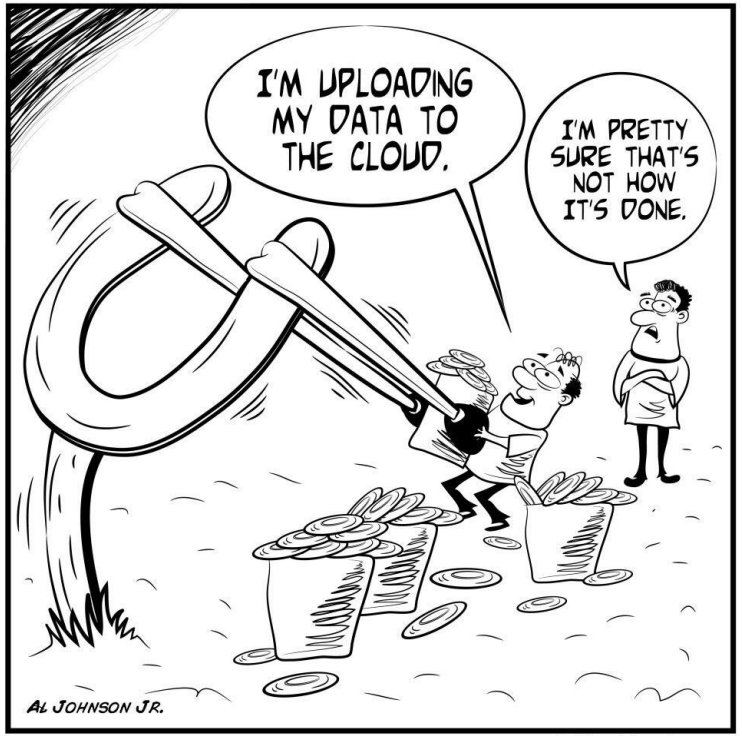
Agile - Pryncypia

- Najwyższy priorytet ma dla nas zadowolenie klienta dzięki wczesnemu i ciągłemu wdrażaniu wartościowego **oprogramowania**.
- Dostarczając funkcjonujące **oprogramowanie** często, w kilkutygodniowych lub kilkumiesięcznych odstępach. Im częściej, tym lepiej.
- Działające **oprogramowanie** jest podstawową miarą postępu.
- Ciągłe **skupienie na technicznej doskonałości** i dobrym projektowaniu zwiększa zwinność.
- Najlepsze **rozwiązania architektoniczne**, wymagania i projekty pochodzą od samoorganizujących się zespołów.

Co daje znajomość aspektów technicznych SMowi?

- jako **servant leaderowi** łatwiej zdobyć zaufanie i szacunek developerów
- może **efektywniej** pomagać Scrum Teamowi skupić się na wytwarzaniu wartościowych Incrementów zgodnych z **Definicją Ukończenia(DOD)**
- może **efektywniej** pomagać **usuwać impedimenty techniczne** ograniczające postępy Scrum Teamu
- pomaga w **usuwaniu barier** pomiędzy interesariuszami a Scrum Teamami

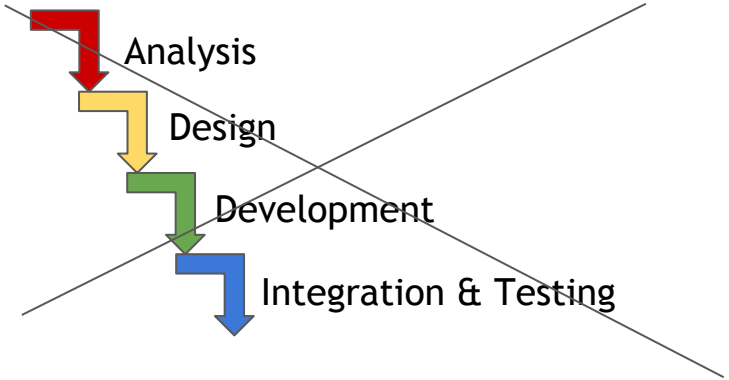
Świadome używanie wiedzy technicznej

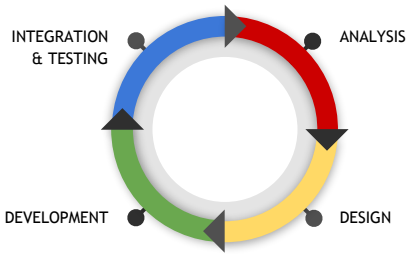


Na co uważać używając wiedzy technicznej jako SM

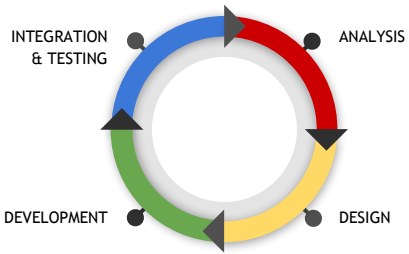
- Facylitacje dyskusji technicznych
- Sugerowanie rozwiązań technicznych zespołowi - nie zakładaj że wiesz lepiej niż zespół
- Rozumienie powagi problemów technicznych
- Rozmowy ze sceptykami zwinnych metod prac
- Scrum Mama - czyli branie odpowiedzialności za wszystkie problemy techniczne na siebie

Agile software development

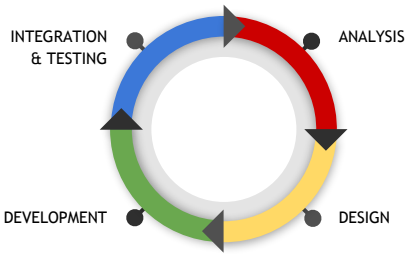




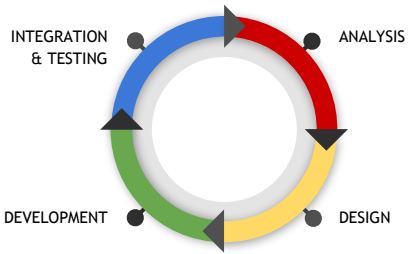
Iteration 1



Iteration 2



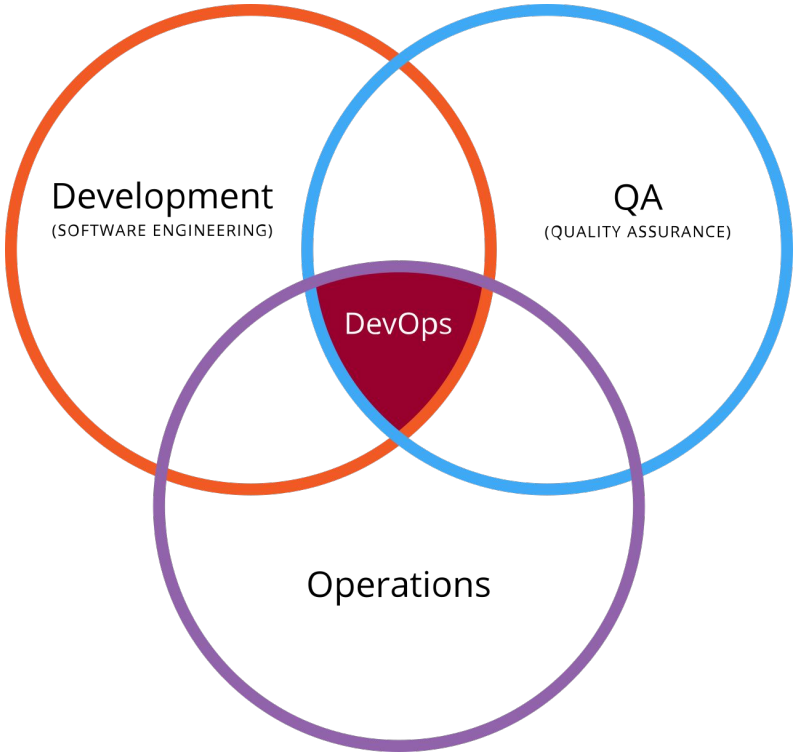
Iteration 3



Iteration **n**

DevOps

DevOps (ang. Development and Operations) to **kultura** która łączy obszary rozwoju (ang. development), eksploatacji (ang. operations) oraz zapewnienia jakości (ang. quality assurance).

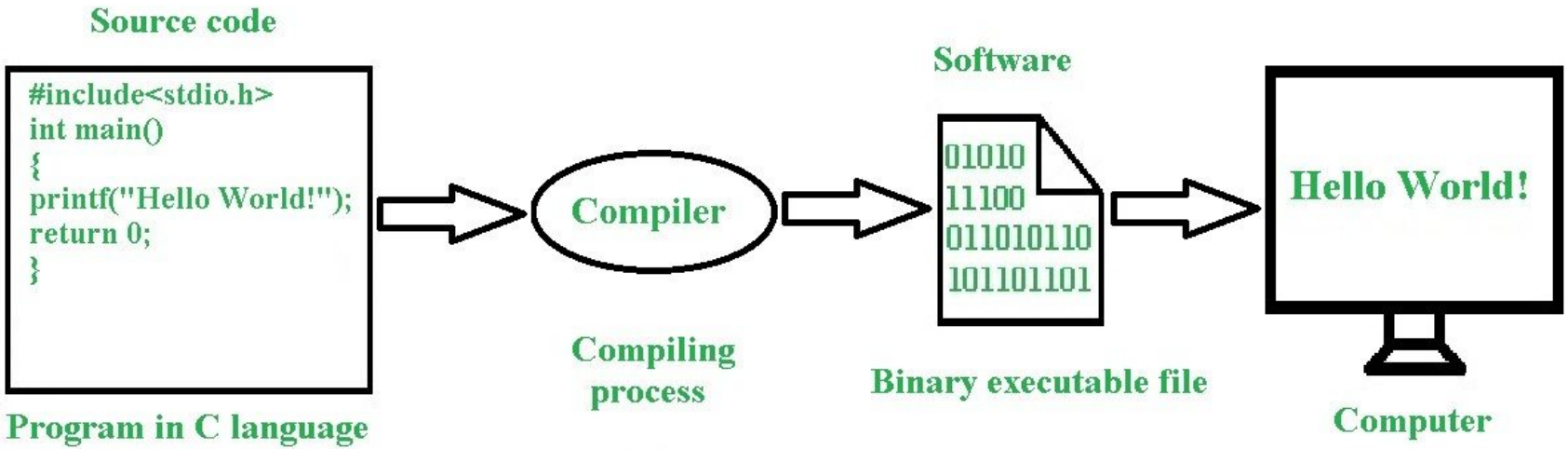


Kultura DevOps kładzie duży nacisk na **automatyzację wszystkiego co możliwe**, co pozwala na **wyeliminowanie ludzkich błędów** w przypadku wielokrotnego wykonywania powtarzalnych czynności.

https://en.wikipedia.org/wiki/DevOps

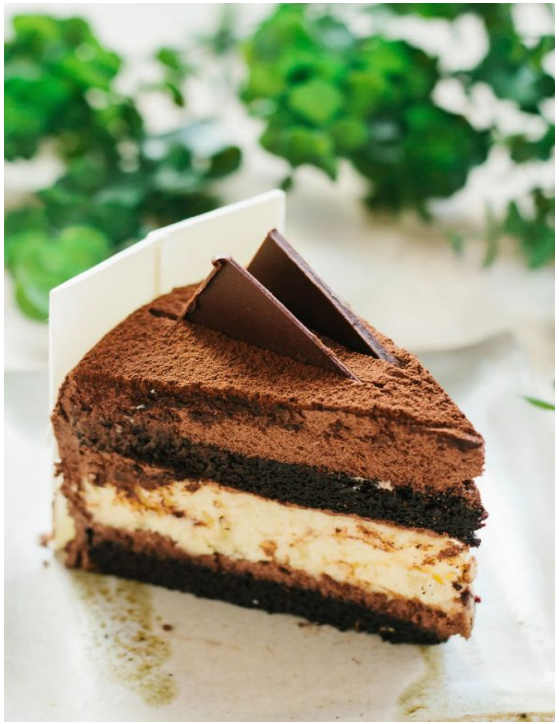
Jak powstaje oprogramowanie?

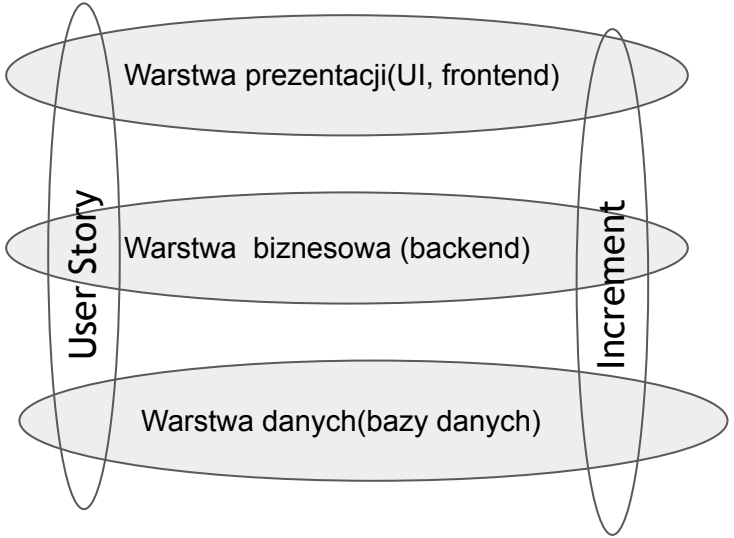
Programowanie to mówienie komputerowi, co ma robić - *Linus Torvalds*



https://www.geeksforgeeks.org/how-software-is-made/

Warstwy oprogramowania





Oprogramowanie wysokiej jakości

- Dług techniczny (ang. technical debt) pod kontrolą
- Refaktoring kodu (ang. code refactoring)
- Przegląd kodu (ang. code review)
- Testy (Piramida Testów), automatyzacja testów
- Środowiska oprogramowania
- System kontroli wersji oprogramowania
- Ciągła integracja (ang. continuous Integration - CI) oraz ciągłe wdrażanie (ang. continuous deployment - CD)
- Kultura DevOps

Dług techniczny

Dług techniczny to domniemany koszt dodatkowych przeróbek spowodowany wyborem łatwego (ograniczonego) rozwiązania teraz zamiast zastosowania lepszego podejścia, które zajęłoby więcej czasu.

Kiedy rozpoznać - przykłady:

- nieczytelny lub zduplikowany kod,
- brak testów, brak automatyzacji wdrożeń,
- brak efektywnych narzędzi,
- duża ilość błędów,
- brak środowiska testowego,
- wydłużony czas dodawania nowych funkcjonalności

https://vincentdnl.com/drawings/technical-debt

Dług techniczny - na co zwracać uwagę

- Czy mamy świadomość jego istnienia? Czy zwiększamy go i spłacamy świadomie?
- Czy developerzy i organizacja mają świadomość jaki wpływ ma dług techniczny na efektywność i dostarczanie wartości dla klienta?
- Czy go mierzymy? Czy używamy metryk np : Ilość defektów, czas dodawania nowej funkcjonalności, pokrycie testami
- Czy mamy przestrzeń w sprincie na spłatę długu technicznego?
- Czy mamy wystarczające kompetencje w zespole aby pracować nad długiem technicznym?

Refactoring

- Refactoring powinien być traktowany jako **codzienna rutyna** i dowód na to, że proces i kod są w dobrej kondycji
- **Brak refactoringu** umieszcza nas na ścieżce do **długu technicznego** i dużego przebudowywania systemu w przyszłości (zwykle oznaczającego zamrożenie dostarczania wartości biznesowej)

Na co zwracać uwagę:

- Czy zespół w ogóle refakturuje kod, czy tylko ‘łączy na sznurki’ pod deadline?
- Czy refaktoring nie staję się np: celem sprintu…?
- Czy refaktorowany jest kod który faktycznie jest w użyciu?

Przegląd kodu (ang.code review)

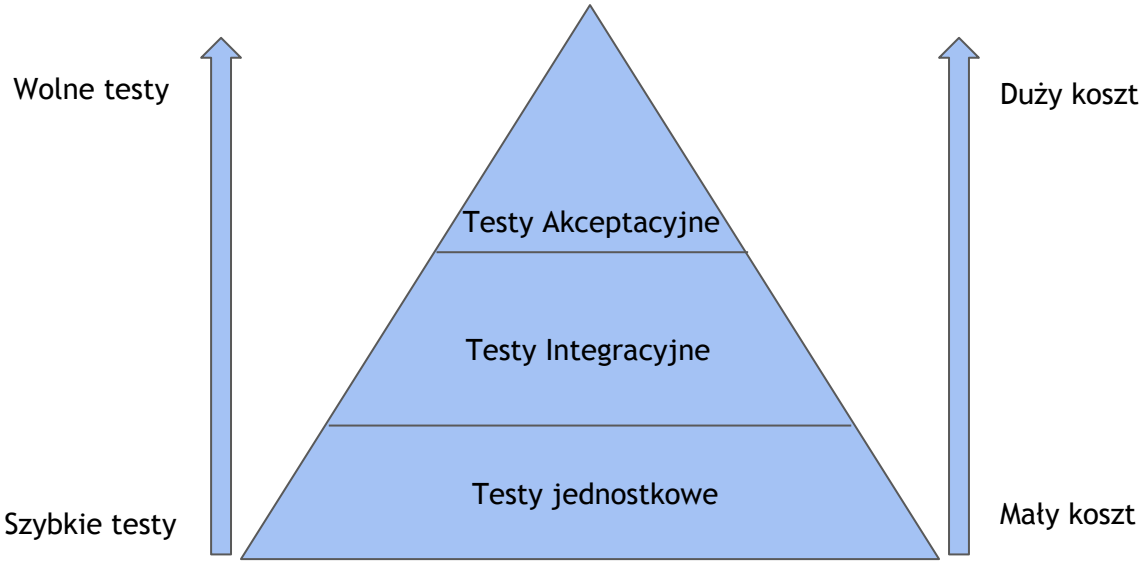
Co daje robienie code review?

- Zwiększa jakość kodu: łatwość konserwacji, jednolitość, czytelność.
- Umożliwia znajdowanie błędów w kodzie.
- Zwiększa poczucie wzajemnej odpowiedzialności za kod.
- Sprawdza kod pod względem przyjętych dobrych praktyk dev.

Na co warto zwrócić uwagę?

- Czy code review jest w ogóle robione?
- Jak duże ilości kodu są wysyłane jednorazowo do review?
- Czy code review jest częścią Definicji ukończenia DOD?
- Czy code review jest robione na bieżąco? Czy odkładane np na koniec sprintu?

Piramida testów



Testowanie oprogramowania - na co zwracać uwagę

- Czy zespół w ogóle testuje oprogramowanie?
- Czy zespół ma kompetencje związane z testowaniem oprogramowania?
- Czy oprogramowanie jest testowane tylko i wyłącznie przez osobę która go stworzyła? (“U mnie działa!”) ?
- Czy testy są częścią Definicji Ukończenia (DOD)?
- Czy testy są automatyzowane? Czy tylko manualne?
- Czy zespół nie rezygnuje z testów pod naciskiem zbliżającego się deadline?

Środowiska oprogramowania

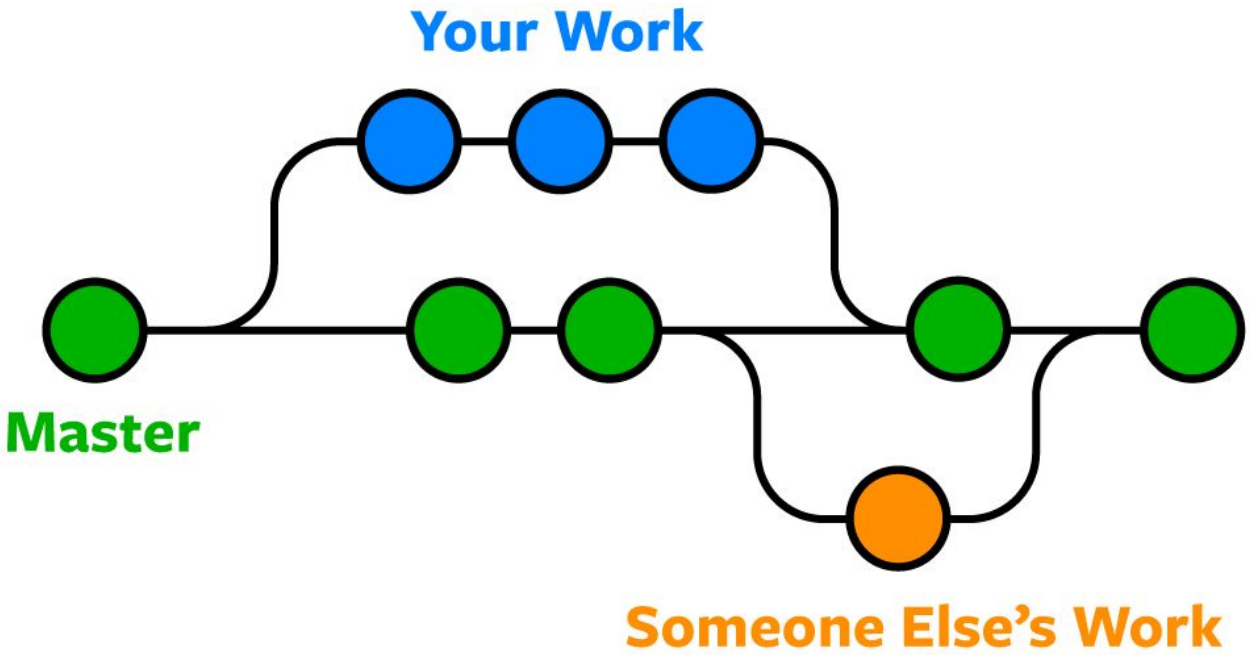
Development	Używane przez developerów do tworzenia nowych funkcjonalności. Brak danych klienckich.
Testing	Używane do testów przez developerów. Brak danych klienckich.
Staging	Używane do testów przez developerów i/lub klientów do testów akceptacyjnych(UAT). Ograniczona ilość danych klienckich
Production	Używane przez klientów (LIVE), na docelowym serwerze. Pełne dane klienckie.

https://en.wikipedia.org/wiki/Deployment_environment

Środowiska oprogramowania - na co zwracać uwagę

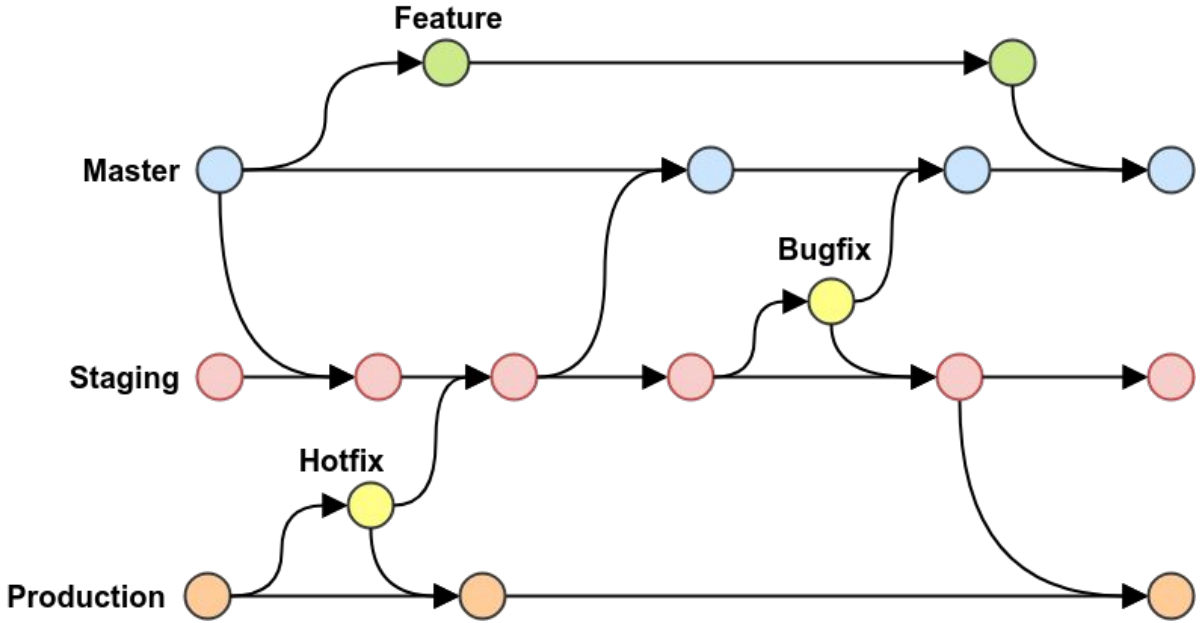
- Czy w ogóle istnieją inne środowiska niż tylko ‘Production’?
- Czy zespół testuje oprogramowania dopiero na środowisku produkcyjnym?
- Czy środowiska są utrzymywane w dobrym stanie i “używalne”?
- Czy istnieją jasne i zrozumiałe dla wszystkich zasady używania środowisk? Np. nie testujemy na produkcji

System kontroli wersji oprogramowania



https://www.nobledesktop.com/image/gitresources/git-branches-merge.png

System kontroli wersji - pojęcia



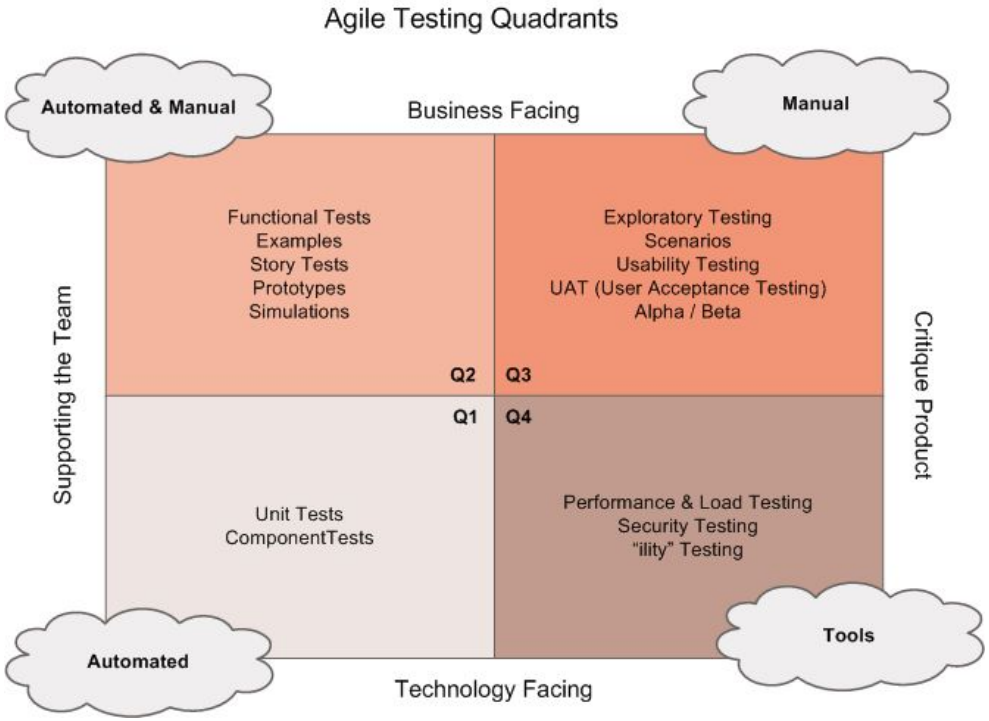
https://blog.programster.org/git-workflows

CI/CD

Continuous Integration (CI) to praktyka programistyczna, która wymaga od programistów jak najszybszej integracji kodu ze współdzielonym repozytorium (w najlepszym przypadku kilka razy dziennie)

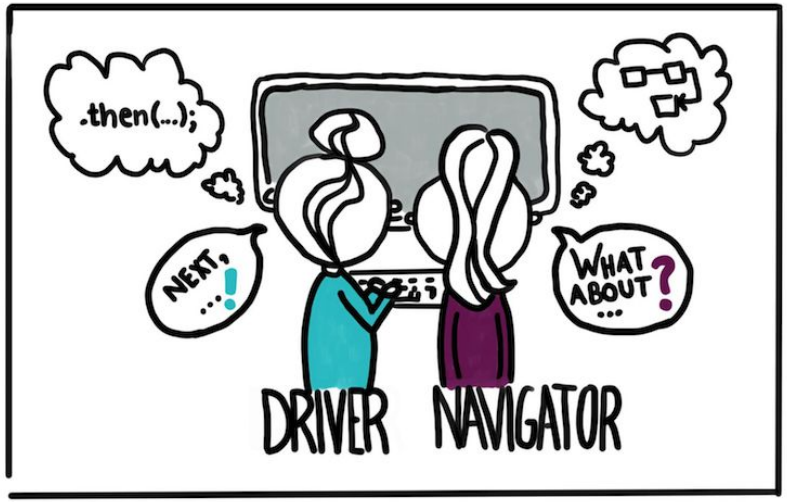
Continuous Delivery (CD) to podejście, w którym zespoły produkują oprogramowanie w krótkich cyklach, zapewniając, że oprogramowanie może być niezawodnie wydane w dowolnym momencie. Wydanie oprogramowania powinno być zautomatyzowane.

Agile testing



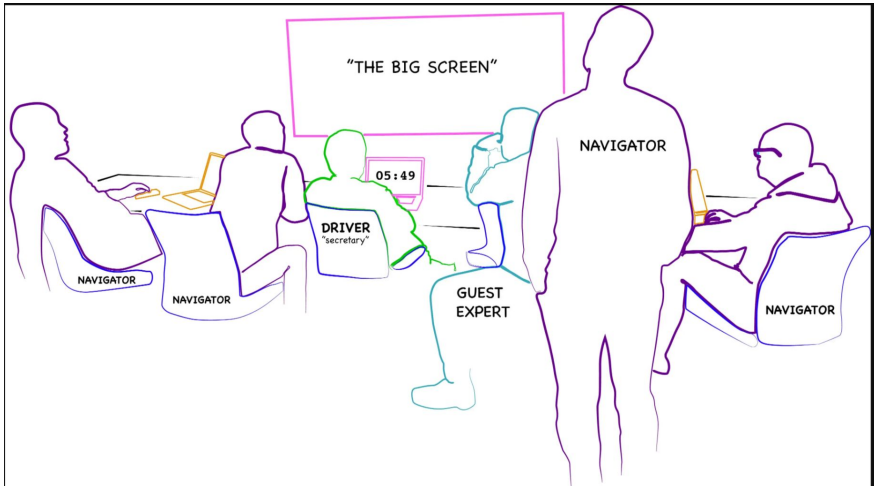
https://lisacrispin.com/wp-content/uploads/2011/11/Agile-Testing-Quadrants.png

Pair Programming



https://martinfowler.com/articles/on-pair-programming.html

Mob Programming

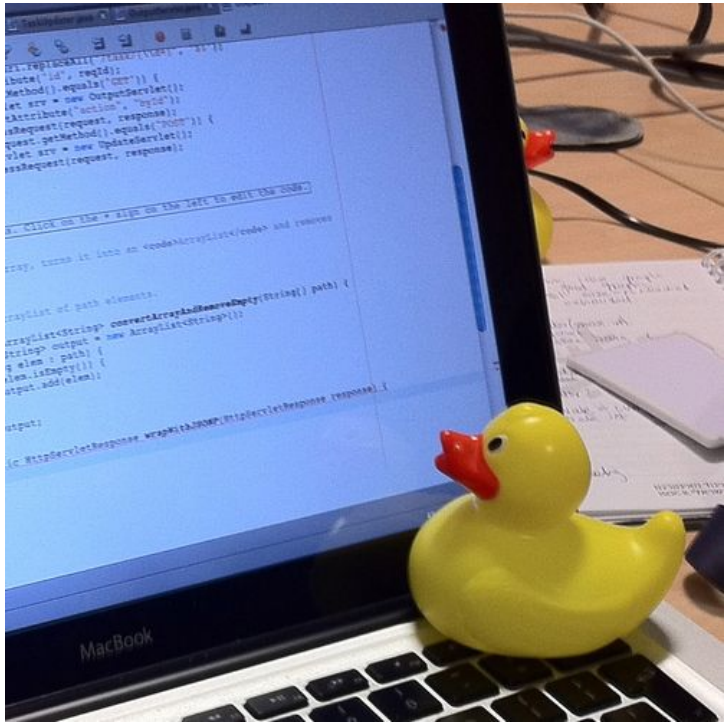


https://betterprogramming.pub/mob-programming-is-quicker-than-you-think-568402c98b2e

Gumowa Kaczka

Metoda polega na tym, że **programista**, próbując znaleźć błędy w kodzie (inspekcja kodu), **trzyma w pobliżu gumową kaczuszkę** lub inny przedmiot nieożywiony.

Linia po linii, programista tłumaczy kaczuszce lub innemu obiektowi przewidywane funkcje każdego segmentu kodu – podczas sprawdzania powinny wyjść na jaw błędy stworzonej aplikacji.



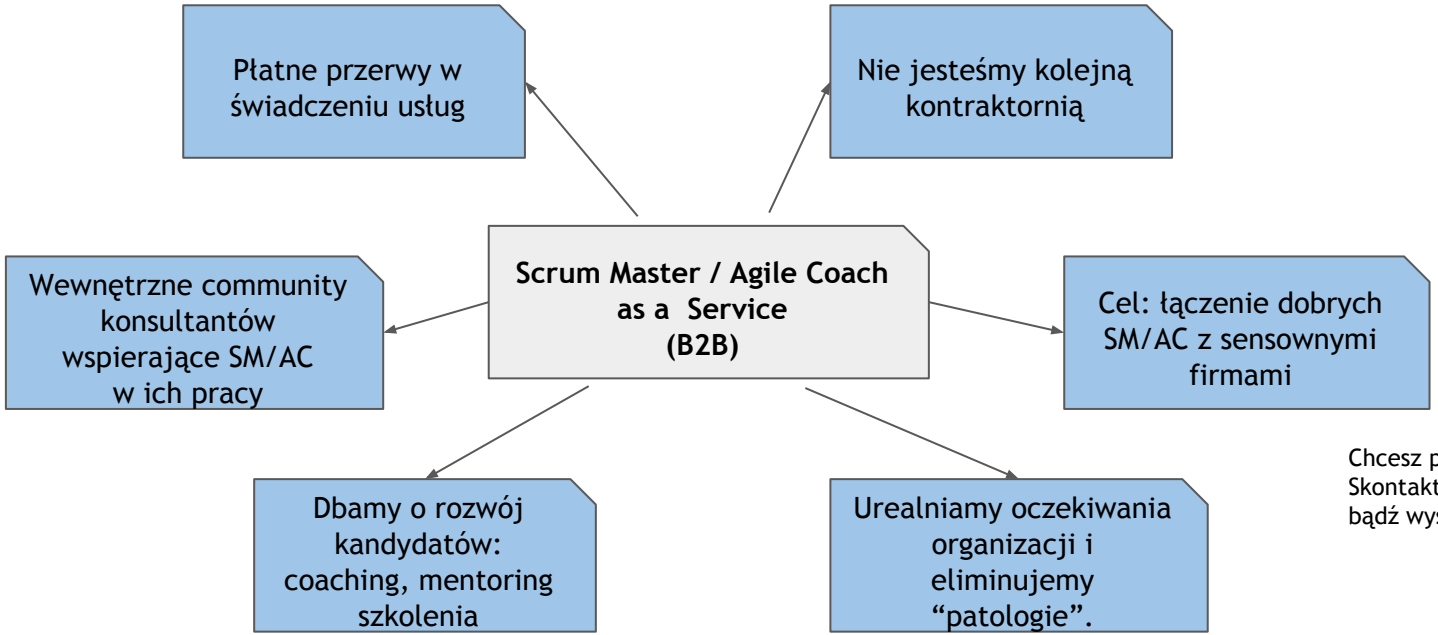
https://upload.wikimedia.org/wikipedia/commons/d/d5/Rubber_duck_assisting_with_debugging.jpg

Co może zrobić SM żeby być lepszym w aspektach technicznych?

- Poprosić developerów żeby pokazali jak wygląda ich praca
- Czytać artykuły , książki o rzeczach technicznych dla osób nietechnicznych
- Brać udział w wydarzeniach takich jak to :)
- Pójść na warsztat z programowania dla osób nietechnicznych (są już takie na rynku).
- Wybrać się na szkolenie np: Applying-Professional-Scrum-For-Software-Development
- Czytać książki: np: “Agile Development. Filozofia programowania zwinnego” - James Shore, Shane Warden.

Podsumowanie

We match purpose driven companies with Agile experts



Chcesz poznać szczegóły?
Skontaktuj się telefonicznie,
bądź wyślij nam wiadomość.

Anna Pałys
mobile: +48 518 488 233
email: anna.palys@valkir.net

Dziękuję

Paweł Lasek

paweł.lasek@valkir.net
https://pl.linkedin.com/in/lasek

https://valkir.pl

Skontaktuj się z autorem tej prezentacji (Paweł) celem zadania dodatkowych pytań (poprzez e-mail LinkedIn).

Rozważ solidną edukację i wsparcie doradcze na przykład z Valkir Academy.

Pytania?