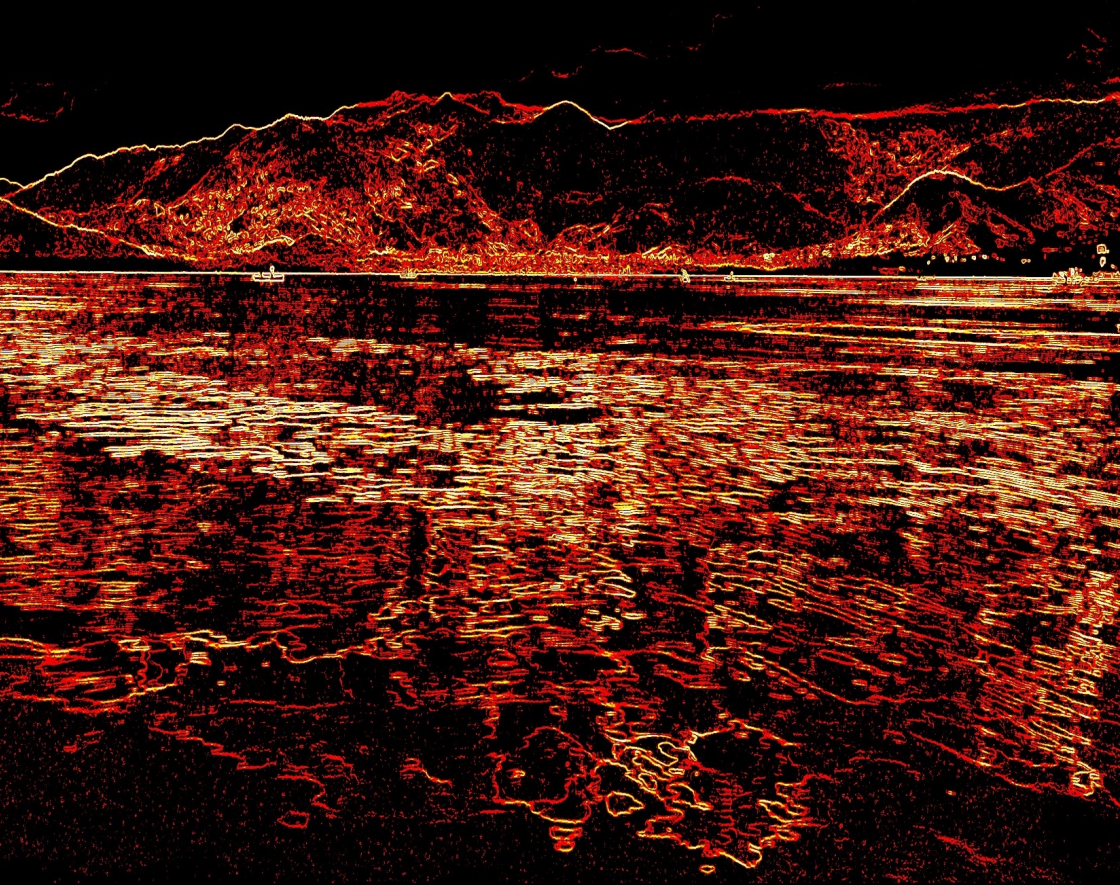
Mity i Problemy w Agile

Dlaczego tak często Agile w organizacjach nie działa?



Wiktor Żołnowski

Mity i Problemy w Agile

Dlaczego tak często Agile w organizacjach nie działa?

Wiktor Żołnowski

This book is for sale at http://leanpub.com/problemywagile

This version was published on 2014-06-13

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Tweet This Book!

Please help Wiktor Żołnowski by spreading the word about this book on Twitter!

The suggested tweet for this book is:

Właśnie zabieram się do czytania książki: ”Mity i problemy w Agile” by @streser https://leanpub.com/problemywagile/

The suggested hashtag for this book is #problemywagile.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search?q=#problemywagile

Spis treści

O Książce

Książka "Mity i Problemy w Agile" powstała na podstawie moich doświadczeń związanych z pełnioną przeze mnie rolą niezależnego Coacha Agile oraz doradcy pomagającego organizacjom podczas transformacji w kierunku Agile. Więcej na temat naszej działalności możecie znaleźć na naszych stronach[1].

Myślą przewodnią książki było omówienie najczęściej spotykanych problemów pojawiających się na drodze organizacji podczas transformacji agileowej. Wokół Agile krąży również wiele mitów, które stopniowo staram się w poniższej publikacji obalać. Takie podejście wydaje się być znacznie ciekawsze niż szczegółowe wykładanie teorii, którą inni wykładali już wielokrotnie.

Książka ta skierowana jest przede wszystkim do praktyków. Nie tylko tych, którzy są już wyjadaczami w Agile ale także tych, którzy dopiero swoją przygodę z transformacją rozpoczynają. Dla tych bardziej doświadczonych pokazanie z pewnej perspektywy niektórych problemów może okazać się kluczowe w doskonaleniu procesów wytwarzania oprogramowania. Dla początkujących wiedza o potencjalnych problemach może pomóc w ich uniknięciu już na samym początku transformacji.

Ci którzy mnie znają wiedzą, że moim ulubionym zajęciem jest kwestionowanie otaczającej nas rzeczywistości. To właśnie dlatego kilka lat temu rozpocząłem swoją karierę w IT jako tester. Ciągle zadając pytania o to jak zepsuć testowane oprogramowanie zdobywałem coraz to nowe umiejętności. Same umiejętności psucia oprogramowania okazały się być dla mnie

[1] http://www.agileszkolenia.pl

niewystarczające. Zadając kolejne pytania o to skąd biorą się błędy w oprogramowaniu i dlaczego nie można tego zrobić lepiej zainteresowałem się procesami wytwarzania oprogramowania. I tak oto w pewnym momencie przekopując się przez różne modele i metody wytwarzania oprogramowania odkryłem Scrum i Agile - wtedy już wiedziałem, że to jest właśnie odpowiedź na pytanie, o to jak wytwarzać oprogramowanie lepiej?!

Miałem przyjemność pracować w kilku zespołach i w kilku różnych organizacjach. W niektórych z nich Agile działało dobrze i dawało niesamowicie pozytywne efekty a w innych znaczenie gorzej. Ciągłe poszukiwania i zadawanie pytań o to dlaczego czasem Agile działa a czasem nie w efekcie doprowadziło do powstania tej właśnie książki.

W międzyczasie zapoczątkowałem ogólnokrajową idee spotkań Bring Your Own Problem[2]. Spotkania te powstały ponieważ zauważyłem, że na naszych szkoleniach oraz na różnych konferencjach najciekawsze rzeczy dzieją często podczas przerw kawowych, kiedy to uczestnicy mają okazję porozmawiać o swoich problemach. O problemach zaczerpniętych z codziennej pracy, charakterystycznych dla ich kontekstu. Bardzo często okazywało się, że wiele osób spotyka się z bardzo podobnymi problemami. Niektórzy radzili sobie z nimi lepiej a inni gorzej. Bez względu na to, każdy miał coś wartościowego do powiedzenia i potężny bagaż doświadczeń, którym chętnie się dzielił. To właśnie na tych spotkaniach utwierdziłem się, że tego typu publikacja może okazać się przydatna dla wielu praktyków Agile.

Niektóre z opisanych problemów pojawiły się wcześniej w dosyć skróconej formie na moim blogu[3], a także opowiadałem o nich na różnych konferencjach, gdzie po raz pierwszy były przez

[2] http://byop.pl
[3] http://blog.testowka.pl

odbiorców testowane.

Książka jest ciągle w fazie rozwoju. Jak do tej pory ukazały się trzy rozdziały poświęcone kolejno porównaniu Agile do klasycznych metod wytwarzania oprogramowania, roli Managera w Agile, oraz nieporozumieniom wokół Scrum.

W planach są kolejne rozdziały miedzy innymi o celach transformacji, nieporozumieniach wokół Kanbana, zespołach krosfunkcjonalnych i pewnie jeszcze kilka innych.

Książka się ciągle rozwija - rozwija nie tylko w mojej głowie ale także w głowach Waszych - czytelników. Dzięki LeanPub mogę na bieżąco otrzymywać od Was feedback i pomysły na dalsze rozdziały. Serdecznie Was do tego zachęcam!

Publikacja ta jest (jak na razie)dostępna już za minimalną kwotę 0.99$. Przy każdej okazji rozdaję też kody promocyjne pozwalające na darmowe jej pobranie. Ale, że nie ma nic za darmo to chciałbym abyście w zamian za darmowy lub bardzo tani dostęp do kolejnych publikacji dali też coś od siebie - przesyłajcie proszę wszelkie uwagi i pomysły na mojego maila [wiktor.zolnowski@gmail.com]lub publikujcie swoje komentarze na stronie książki w LeanPub.

Niezwykle miłym jest to, że niektórzy z Was postanowiło zapłacić za książkę, po tym jak otrzymali ją za darmo i przeczytali. Jeszcze raz serdecznie dziękuję!

“Największym wrogiem wiedzy nie jest ignorancja, lecz iluzja wiedzy”

Stephen Hawking

Agile to taki iteracyjny Waterfall

Rozpoczynając pracę nad tą książką zastanawiałem się nad tym jaki temat będzie dobry na początek? Jaki mit obalić jako pierwszy? Najlepiej zacząć od "podstaw" i tematów, które jeśli nie zostaną dobrze zrozumiane to mogą mieć największe negatywne konsekwencje. Jedynie dobre zrozumienie genezy i podstawowych wartoci Agile a co za tym idzie różnic pomiędzy Agile a klasycznym podejściem projektowym pozwala na uniknięcie wielu błędów i problemów.

Zanim zaczniemy rozważania na temat wyższości Agile nad Waterfall i na odwrót, warto zastanowić się nad tym, czym Agile tak właściwie jest?

Na przestrzeni ostatnich kilkunastu lat Agile zyskiwało na popularności stając się jedną z najpopularniejszych, o ile nie najpopularniejszą metodą wytwarzania oprogramowania. Ten progres przyniósł wiele nieoczekiwanych konsekwencji, gdyż ilość nie zawsze idzie w parze z jakością. Niestety coraz częściej spotykam się totalnym brakiem zrozumienia tematu przez osoby, które za transformacje Agile w swoich i cudzych organizacjach się zabierają. Zbyt często skupiamy się na samych metodach i narzędziach pomijając kontekst w jakim te narzędzia powstały i były z powodzeniem stosowane. Niestety tracimy wtedy możliwość oceny czy dane narzędzie lub metoda jest odpowiednia do zastosowania w naszym kontekście.

Jak wiele Agile jest w Agile?

Coraz więcej organizacji decyduje się na wykorzystywanie elementów zwinnych metodyk zarządzania projektami. Bycie Agile stało się trendem. Jeśli twoja firma nie jest Agile to klienci nie chcą z Tobą rozmawiać. Taki trend oprócz wielu pozytywnych z punktu widzenia propagatorów Agile korzyści spowodował również wszechobecną dewaluację słowa "Agile". Teraz każdy jest Agile - cokolwiek by to nie znaczyło. Byleby tylko móc się tym pochwalić. Powstają nowe firmy prześcigające się w wymyślaniu certyfikacji Agile dla organizacji tylko po to, by mogły się one chwalić kolejnym papierkiem przed swoimi klientami. Ale czy taki papier cokolwiek znaczy? Czy nie przerabialiśmy już tego wcześniej? Czyż nie tak było z normami ISO w IT czy np. CMMI? Certyfikacja ma zapewne również pozytywne strony - jak na przykład pewna standaryzacja, czy w ogóle motywacja do jakichkolwiek działań więc nie chciał bym byście źle zrozumieli moje intencje. Nie neguję certyfikacji jako takiej, lecz poddaję w wątpliwość motywację stojącą za decyzjami o takiej czy innej certyfikacji w organizacjach.

Przez ostatnich kilka lat zadawałem na różnych konferencjach i wydarzeniach dla praktyków IT trzy pytania:

1. Kto z Was pracuje w organizacji, która chwali się tym, że jest Agile, używa Scrum, Kanban lub czegoś innego w tym stylu?
2. Kto z Was jest pewien, że to, co tak naprawdę robi jego organizacja, można z czystym sumieniem nazwać Agile?
3. U kogo ten Agile działa spełniając oczekiwania postawione przed transformacją w Agile?

O ile na pierwsze pytanie pozytywnie odpowiadała znacząca

większość uczestników konferencji, to już przy drugim w górze pozostawało zazwyczaj tylko kilkanaście rąk. Przy trzecim pytaniu zazwyczaj pozostawało kilka osób, które przeważnie mogłem wymienić z imienia i nazwiska - świat entuzjastów i praktyków Agile w Polsce jest dosyć mały i praktycznie wszyscy się znamy.

Okazuje się, że wiele organizacji chwalących się tym, że są Agile z prawdziwą zwinnością nie ma zbyt wiele wspólnego. Mało tego, pracownicy tychże organizacji doskonale zdają sobie sprawę z tego, że w tym co robią na co dzień jest wiele błędów i niedomówień powodujących różne dysfunkcje. Wiele razy słyszałem, że stosujemy "tak jakby Scrum", albo "no mamy taki Agile według naszych korporacyjnych standardów". Kilka pytanych osób jest w stanie nawet wymienić kilka najważniejszych dysfunkcji, które obserwują w swoich oraganizacjach ale większość raczej bardziej "czuje", że coś jest nie tak niż jest w stanie wskazać konkretne błędy i problemy. Wniosek nasuwa się jeden - świadomość i wiedza na temat tego czym są Zwinne Metody wytwarzania oprogramowania jest nadal bardzo niska w organizacjach z branży IT.

W efekcie Agile staje się kolejnym buzz-wordem, przez wielu wręcz znienawidzonym. W ten sposób rodzi się pierwszy mit do obalenia - "Agile nie działa". Przyszło nam odnaleźć się w takiej a nie innej rzeczywistości, gdzie rzeczy pomimo tego, że nazwane po imieniu nie zawsze są tym, czym by się wydawały. Pozostaje jednak pytanie co my - w tym także Ty, Drogi Czytelniku możemy z tym zrobić? Pierwszym krokiem jest wiedza, która mam nadzieję po przeczytaniu tego i kolejnych rozdziałów pozwoli nam na ustalenie dalszych, konkretnych działań.

Czym jest Agile? O co w tym wszystkim chodzi? Co jest ważne a co nie? Nie chcę tutaj cytować definicji tego, czym jest Agile, ani też tworzyć swoich własnych. Najważniejsze jest w

tym wszystkim to, co za pomocą Agile, Lean czy czegokolwiek innego chcemy osiągnąć, oraz to, do czego dane metody zostały stworzone. Tutaj pojawia się kolejny problem, który próbowałem wielokrotnie wybadać zadając trzecie pytanie dotyczące spełnienia oczekiwań stawianych przed transformacją Agile. Wiele organizacji i osób rozpoczynających takie zmiany nie ma skonkretyzowanych celów anie oczekiwań. W następstwie błądzą oni trochę na ślepu próbując osiągnąć jak najwięcej zmieniając trochę tu trochę tam i tworząc w efekcie "swóje własne podejście do Agile", przy okazji generując wiele problemów.

Ruch wokół Agile zapoczątkowali ludzie, którzy dostrzegli nowe możliwości wynikające z szybkiego rozwoju technik i narzędzi programistycznych. Wraz z rozwojem technologii a także rosnącą liczbą specjalistów od programowania rosła ilość wytwarzanego oprogramowania, które zaczęło pojawiać się w niemal każdej dziedzinie naszego życia. Oprogramowanie zdominowało świat który otacza nas obecnie.

Początkowo programowanie było domeną pasjonatów, którzy poświęcali całe swoje życie, by rozwiązywać nietrywialne problemy przy użyciu komputera. Z czasem wytwarzanie oprogramowania stało się jednak bardzo lukratywnym biznesem - zwłaszcza po sukcesach startupów takich jak Microsoft czy Apple (później na przykład Google i Facebook). Dodatkowo nowe technologie i łatwy dostęp do wiedzy (popularyzacja Internetu) skusiły wiele osób do tego, by rozpocząć swoją przygodę z programowaniem. Prekursorzy Agile dostrzegli, że wraz z ilością produkowanego oprogramowania spada jego jakość. Produkcja na masową skalę bardzo często prowadziła do sytuacji, w których ludzie pracowali latami nad aplikacjami, które nigdy nie zostały użyte, bądź też nie cieszyły się popularnością wśród potencjalnych użytkowników. Aplikacje takie oczywiście nigdy

na siebie nie zarobiły a inwestorzy tracili na nich często niemałe pieniądze.

Agile obok Lean Software Development oraz Lean Startup było i jest obecnie jedną z odpowiedzi na pytanie jak wytwarzać oprogramowanie optymalizując koszty jego wytwarzania oraz maksymalizując zyski płynące z dostarczania użytecznej funkcjonalności użytkownikom?

Za wytwarzanym oprogramowaniem praktycznie zawsze kryje się jakiś model biznesowy, a co za tym idzie biznes. Biznes to organizacja. Agile i Lean starają się odpowiedzieć na pytanie jaka powinna być organizacja, by mogła wytwarzać wartościowe oprogramowanie w optymalny sposób. Oznacza to, że wspomniane wcześniej transformacje Agile przeważnie, docelowo znacznie wykraczają poza dział IT, obejmując całe organizacje. W zasadzie większość powszechnie stosowanych metod zwinnych takich jak na przykład Scrum czy Kanban docelowo mają wywierać umiarkowany nacisk na całą organizację i przy pomocy empirycznego podejścia inkrementalnie wprowadzać w niej zmiany.

Czy Agile daje gwarancję sukcesu?

Odpowiedź na to pytanie brzmi oczywiście: Nie - Agile tak samo jak każda inna, mniej lub bardziej popularna metoda pracy, nie daje gwarancji tego, że zawsze wszystko będzie działało i przynosiło korzyści. Agile nie jest też odpowiednim podejściem do zarządzania pracy w każdej organizacji - czasami ilość zmian koniecznych do wprowadzenia nie jest warta potencjalnych, lecz niepewnych korzyści.

Główną zaletą zwinnych metod wytwarzania oprogramowania jest to, że skupiają się one na ciągłym udoskonalaniu procesów oraz narzędzi. Duży nacisk kładziony jest także na rozwój umiejętości pracowników, gdyż to właśnie indywidualne umiejętności oraz praca zespołowa są podstawą do osiągnięcia sukcesu. Dzięki temu bez względu na to czy próby wdrażania pewnych praktyk zakończą się powodzeniem w efekcie po takich eksperymentach nasza organizacja przynajmniej w teorii powinna być odrobinę lepsza. Prawda jest jednak taka, że czasami zwiększenie świadomości co do tego jak bardzo stosowane obecnie procesy są mało efektywne w porównaniu do zwinnych metod może wprowadzać niemały dysonans poznawczy i prowadzić do obniżenia motywacji pracowników. To tylko jedno z wielu ryzyk i zagrożeń związanych z rozpoczęciem transformacji Agile w organizacji.

Agile skupia się na skracaniu pętli informacji zwrotnej dotyczącej tego, czy to, w jaki sposób pracujemy i to, co jest efektem naszej pracy ma sens i jest najbardziej optymalne oraz efektywne. Jeśli na podstawie takiej informacji wnioskujemy, że potrzebna jest zmiana, to po prostu coś zmieniamy. Dzięki przyspieszonemu feedbackowi nie musimy się długo zastanawiać i planować tego jaką decyzje podjąć? Często podejmujemy decy-

zję taką jaka "wydaje się" nam najlepsza w danym momencie i bardzo szybko ją testujemy w praktyce. Efekty widoczne są bardzo szybko - często już po jednej iteracji, więc równie szybko możemy zmienić decyzję i w najgorszym wypadku wrócić do status quo sprzed wprowadzonych zmian. Takie podejście jest z goła odmienne od tradycyjnego zarządzania projektami, gdzie wszystko musi być w maksymalnym stopniu szczegółowo zaplanowane i przewidziane a zmiana planów oznacza uruchomienie bardzo powolnych i skomplikowanych procedur.

Odnośnie pętli zwrotnej to nawet pisząc tą książkę generuję wersję testową kilkanaście razy dziennie by zobaczyć jak wprowadzone właśnie zmiany wyglądają w gotowym produkcie. Do rąk czytelników kolejne wersje również trafiają dosyć często - a przynajmniej staram się by tak było. To chyba można uznać za Agile.

Dlaczego większość wdrożeń Agile kończy się niepowodzeniem?

Jak już wspomniałem wcześniej w wielu przypadkach wdrożenie Agile w organizacji nie przynosi oczekiwanych efektów. Bywa tak zwłaszcza wtedy, gdy decyzja o zastosowaniu wspomnianych metodyk zapada zbyt późno - gdy organizacja jest już pełna dysfunkcji. Często zmiany wymagają czasu - a im większa organizacja, tym tego czasu potrzeba więcej.

Powodów niepowodzeń jest znacznie więcej - na przykład już samo stwierdzenie, że Agile można wdrożyć jest błędne. Aby organizacja była zwinna potrzebna jest *gruntowna transformacja* (a nie tylko wdrożenie), która zmieni sposoby zachowań poszczególnych ludzi oraz interakcji między nimi. Agile nie można wdrożyć (w sensie zakończyć wdrożenie), gdyż transformacja Agile jest procesem ciągłym i nigdy się nie kończy. Bycie Agile oznacza ciągłe wprowadzanie zmian, eksperymentowanie i dostosowywanie się do rynku i otaczającej nas rzeczywistości. Ta zmiana sposobu myślenia na temat transformacji Agile jest wymagana u wszystkich osób zaanagażowanych w proces nie tylko wśród osób zarządzajacych organizacją.

W tej książce będę dosyć często odnosił się do Scrum, gdyż jest to jedna z najpopularniejszych ostatnio zwinnych metod stosowana powszechnie w wielu firmach. Warto zaznaczyć, że Scrum to nie to samo co Agile, jak niektórzy uważają. Nawet poprawne wdrożenie Scrum (z moich obserwacji nadal dosyć rzadkie) nie gwarantuje osiągnięcia zwinności organizacji. Po szczegółowy opis Scrum odsyłam do Scrum Guide[4]. Zalecam lekturę w oryginalnej wersji anglojęzycznej, gdyż powszechnie

[4] http://www.scrumguides.org/

stosuje się terminologię w tym języku, a polskie tłumaczenia brzmią wręcz fatalnie.

Scrum Guide opisuje w sposób zupełnie wystarczający całą metodykę - wystarczy jednynie przeczytać te kilkanaście stron bardzo uważnie. Sam regularnie wracam do tej lektury. Traktuję to jako pewnego rodzaju ćwiczenie, podczas którego uważnie czytam analizując znaczenie każdego przeczytanego słowa (oczywiście w wersji oryginalnej - angielskiej, by nie sugerować się tym, co mogło zostać utracone podczas tłumaczenia). Praktycznie za każdym razem odnajduję tam coś innego.

W tym miejscu zapewne określicie mnie mianem jakiegoś fanatyka Scrum - nie, nie jestem fanatykiem. Scrum jest jednym z narzędzi, którego używam w swojej pracy Coacha i doradcy. Niemniej jednak należę do osób, które uważają, że dobra znajomość narzędzi to podstawa ich poprawnego użycia, dlatego tak wiele uwagi poświęcam na eksperckie opanowanie tej czy innej techniki. Ponadto zauważyłem, że wiele problemów organizacji, z którymi współpracuję, wynika z niewłaściwego lub niekompletnego zrozumienia często wydawało by się drobnych i wręcz nieistotnych szczegółów dotyczących metod pracy. Dobra znajomość wielu narzędzi pozwala również odpowiednio dobrać narzędzia do sytuacji.

Bardzo często słyszę od naszych klientów pytanie: ile to potrwa? Początek transformacji jest trudny zwłaszcza, gdy mamy do czynienia z organizacją, która ma już jakąś przeszłość. Możemy tutaj wyróżnić co najmniej dwa rodzaje organizacji - takie, które stosowały już z powodzeniem zdefiniowane metodyki wytwarzania oprogramowania i takie, które do tej pory wytwarzały oprogramowanie w sposób nazwijmy to odrobinę chaotyczny. Często granica między jednym a drugim dosyć szybko się zaciera, zwłaszcza, gdy organizacja szybko rośnie.

Wbrew pozorom w przypadku organizacji, która wcześniej stosowała cięższe podejście kaskadowe (ang. Waterfall) transformacja może być łatwiejsza niż w przypadku organizacji, w których żadnych zdefiniowanych metodyk wcześniej nie było. Mimo to wypracowane wcześniej nawyki mogą stanowić pewien problem, dlatego nie ma raczej reguły - wszystko zależy od kontekstu. Czas potrzebny do osiągnięcia widocznych efektów transformacji zależy od bardzo wielu czynników i nie da się go z góry przewidzieć. Jednym z głównych i najbardziej zmiennych czynników w tej układance są ludzie i ich motywacja do tego by coś zmienić, a w zasadzie by zmiana stała się procesem ciągłym - często nie tylko w pracy ale i w całym ich życiu. Z mojego doświadczenia wynika, że czasem by widoczne były efekty wprowadzanych zmian potrzebne jest nawet kilkanaście miesięcy (coaching dla klienta korporacyjnego) a czesm zmiany są widoczne po kilku dniach od pierwszego spotkania z zespołem (coaching w małym startupie - przykład opisany w mojej książce "Agile Transforacje[5]".

[5] https://leanpub.com/agile-transformacje

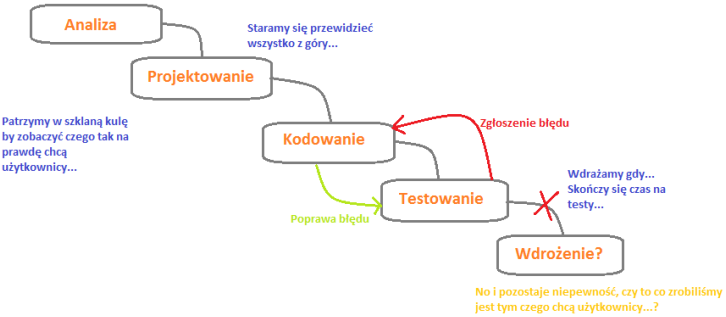
Scrum a Waterfall

Scrum często nazywany jest "iteracyjnym podejściem do wytwarzania oprogramowania". O ile stwierdzenie to nie jest błędne, to Scrum ma jeszcze jedną, dużo bardziej istotną cechę - *jest inkrementalny*. Oznacza to, że co iterację będziemy dostarczać kolejny inkrement - kolejną wersję działającej, potencjalnie gotowej do wydania na produkcję aplikacji. Ta inkrementalność jest właśnie tym, co najbardziej odróżnia (z perspektywy developera) Scrum od innych iteracyjnych podejść do wytwarzania oprogramowania, często bazujących nawet na modelu kaskadowym.

Inkrementem jest coś, co możemy uznać za działającą funkcjonalność dodającą jakąś wartość do naszego produktu. Zatem nie nazwiemy inkrementem przygotowanej dokumentacji czy też planu, nie nazwiemy inkrementem projektu architektury, nie nazwiemy inkrementem raportu z testów, itd. Inkrement musi być używalny przez potencjalnych użytkowników. Inkrement możemy zdefiniować jako widoczna zmiana w zachowaniu systemu wnosząca konkrentą, zdefiniowaną wartośc dla interesariuszy tego systemu.

W tradycyjnym modelu kaskadowym wyróżnialiśmy poszczególne etapy wytwarzania oprogramowania (realizacji projektu informatycznego). Przeważnie są to kolejno: Analiza Wymagań, Tworzenie Projektu Architektury, Implementacja, Testy, Wdrożenie. Podejście to działało całkiem nieźle, gdy tworzyliśmy aplikacje pudełkowe, które po zakończeniu lądowały na półce w sklepie i albo się sprzedawały albo nie - ale to już nie był nasz problem, gdyż w międzyczasie zaczynaliśmy już kolejne projekty. Na każdy etap takiego projektu mogliśmy z góry alokować pewną ilość czasu i innych zasobów, a następnie monitorować i optymalizować na bieżąco ich wykorzystanie.

Problemy przeważnie pojawiały się dopiero na etapie testów, gdzie nagle okazywało się, że zanim projekt będzie mógł przejść w fazę wdrożenia, potrzebne są poprawki błędów, które zostały wykryte. Jeśli błędy dotyczyły samej fazy developmentu to nie był to jeszcze duży problem - wystarczyło zgłosić błąd, poczekać, aż developer na szybko coś połata, sprawdzić i oznaczyć jako problem rozwiązany. Gorzej jeśli błędy wykryte podczas testowania zostały zaimplementowane w fazie projektowania rozwiązań albo co gorsza, na etapie ustalania wymagań powodując na przykład, że niektóre wymagania są ze sobą sprzeczne lub niekompletne. W takim przypadku należało cofnąć się aż do fazy zbierania wymagań, następnie zmienić architekturę projektu, zaimplementować zmiany i jeszcze raz przetestować cały system. Tak - jasne… Tak wynikałoby z teorii - niestety w swojej karierze nigdy nie spotkałem się z takim podejściem. W zasadzie oczywistym jest to, że takie cofanie się byłoby zupełnie nieopłacalne (dla projektów innych niż elektrownie jądrowe czy statki kosmiczne). Tymczasem znacznie częściej po prostu łatano na szybko znalezione błędy tak, by było jak najtaniej i najszybciej. Między innymi stąd wzięło się często zadawane przeze mnie pytanie: czy poprawianie błędów oznacza poprawianie jakości oprogramowania? Tego typu poprawki na szybko mogą jedynie obniżyć jakość kodu i wygenerować szereg potencjalnych problemów z jego utrzymywalnością, które dopiero się pojawią - w ten sposób powstaje dług techniczny.



Czy coś jest nie tak z podejściem kaskadowym?

W wytwarzaniu oprogramowania - bez względu na model - koszty błędów są tym większe, im później zostały one wykryte. Dla modelu kaskadowego powstało kilka obejść tego problemu - jak na przykład próba zaangażowania testerów od samego początku trwania projektu tak, by testowali już samą dokumentację - na przykład "Model W[6]". Pomimo starań niezwykle często okazywało się, że po wydaniu na produkcję, dana aplikacja nawet zgodna ze specyfikacją nie była używalna bądź też potrzebna użytkownikom. Ewidentnie coś było nie tak. Agile wydawał się być pierwszą od lat odpowiedzią na pytanie jak wytwarzać oprogramowanie wysokiej jakości, które będzie jednocześnie wartościowe dla użytkowników. Właśnie magiczna pętla informacji zwrotnej, oraz jej skracanie do minimum stały się kluczowym czynnikiem pozwalającym na maksymalizację zysków. Właśnie dzięki skracaniu tej pętli oraz praktykom i narzędziom, które w tym celu powstały, narodziła się między innymi idea Lean Startup.

[6]http://sqa.fyicenter.com/FAQ/Software-Development-Models/Software_Development_Models_W_Model.html

Water-Scrum-Fail i W-Agile.

Na jednej z największych w Polsce konferencji dla testerów miałem okazję wysłuchać wykładu, którego temat brzmiał mniej więcej tak: "Scrumowy zespół testerski". Sam tytuł brzmi przerażająco - nie ma czegoś takiego jak Scrumowy zespół testerski. To nie jest Scrum! To nawet nie jest blisko do Scrum.

Zamiast krytykować, zacząłem zastanawiać się z czego to wynika i przypomniała mi się wtedy pewna historia opowiadana przez Tomka Włodarka - Trenera Scrum i Agile Coacha(ST) pracującego ówcześnie dla pewnej korporacji. Pewna pani project manger (PM) postanowiła udowodnić mu, że w zasadzie to nie ma różnicy pomiędzy Scrum a Waterfall. Wyglądało to mniej więcej tak:

- PM: Skoro mamy czterotygodniową iterację to możemy ją sobie podzielić na poszczególne etapy i na przykład przez pierwszy tydzień planować i projektować, później dwa tygodnie kodować, a ostatni tydzień poświęcić na testy i wdrożenie.
- ST: Nie, nie możemy…
- PM: Ale poczekaj, popatrz, jak nasi architekci skończą projektowanie, możemy przenieść ich do sprintu innego zespołu, któremu zaprojektują prace na ich fazę developmentu…
- ST: Zaraz, zaraz, ale to jest Waterfall…
- PM: No nie jest Waterfall bo przecież będziemy pracować w sprintach. No dobrze, to może robimy sobie zespół architektów pracujących w Scrum, zespół developerów i zespół testerów. Będziemy tylko zarządzać ich pracą.

- ST: Ale czym to się będzie różnić od Waterfall?
- (Po czym ST narysował to na tablicy w postaci kaskady)
- ST: No i co to jest Waterfall czy Agile?
- PM: W... A... W... W-Agile... (czyt. Ładżajl).

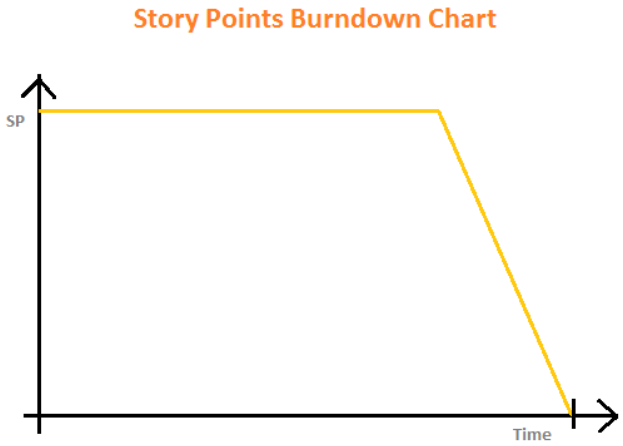
Powyższy przykład jest może aż nazbyt jaskrawy, ale tego typu wdrożenia W-Agile miałem wątpliwą przyjemność oglądać naprawdę w wielu organizacjach. Większość osób, które zgłaszają się do doradcy w dziedzinie Agile i Scrum, to ludzie, którzy mają duże problemy z wdrożeniem Scrum i transformacją Agile. Jak to jeden z czytelników mojego bloga[7] podsumował: "Ze zmian cieszy się tylko dziecko z mokrą pieluchą" - coś w tym jest. Organizacje, które nie mają problemów, lub raczej takie, które tych problemów jeszcze nie dostrzegają, nie zwracają się do nas z prośbą o pomoc w przeprowadzaniu zmian.

Do dwóch typów organizacji (zdefiniowane procesy lub organizacja chaotyczna), w których rozpoczynamy zazwyczaj transformację Agile możemy dodać trzeci typ - organizację, która próbowała już swoich sił w Scrum ale niestety nie udało się i powstała pewna hybryda, która niekoniecznie działała dobrze - typowy Water-Scrum-Fail.

Wracając do W-Agile (jak się okazuje niezwykle powszechnego w wielu firmach), jednym z narzędzi służących do wykrywania tego problemu może być wykres Story Point Burndown Chart. Wykres ten przedstawia ilość Story Points, czy też innych jednostek estymacji, spalonych (zrealizowanych) podczas iteracji. Wykres ten opada tylko wtedy, gdy dane wymaganie

[7] http://blog.testowka.pl

zostanie zrealizowane od początku do końca zgodnie z Definition of Done. Jeśli wykres wygląda tak, jak na załączonym obrazku - przez pierwszą część Sprintu jest płaski, a ostatniego dnia gwałtownie opada do zera to wiedz, że coś się dzieje... A mianowicie wiedz, że masz najprawdopodobniej do czynienia z Water-Scrum-Fail.



Story Point Burndown Chart - Water-Scrum-Fail

Wykres taki (jeśli oczywiście w Sprincie mieliśmy więcej niż jedno wymaganie - co jest zalecane) wraz ze Sprint Burndown Chart (pokazującym ilość godzin pozostałych do spalenia),który wygląda "dobrze", pokazują, że najprawdopodobniej wszystkie wymagania były rozpoczęte na raz, po czym na koniec zostały przetestowane na szybko, z dużym ryzykiem niepowodzenia i wątpliwą rzetelnością. Sytuacja taka jest niepożądana, gdyż nagle może okazać się, że zabraknie nam czasu na testy i wtedy Sprint zakończy się bez inkrementu.

Problemy tego typu mają swoje korzenie przeważnie w "silosowości organizacji". Tradycyjne metody wytwarzania oprogramowania narzucały mocno wyspecjalizowaną strukturę organizacyjną. Mieliśmy osobne działy analizy, programowania i testów. Osoby w tych działach najprawdopodobniej nigdy nie pracowały bezpośrednio ze sobą w jednym zespole. Jeśli taki sztywny podział nadal istnieje i pracownicy tych działów nie pracują RAZEM to nie ma mowy o działającym Scrum w naszej organizacji. Z powodu licznych biurokratycznych problemów wiele organizacji decyduje się na tworzenie "wirtualnych" - kros-funkcjonalnych zespołów Scrumowych złożonych z przedstawicieli poszczególnych działów. Rozwiązanie takie nie wprowadza rewolucji w strukturach organizacyjnych, a ma szanse działać. Niemniej jednak członkowie takiego zespołu muszą pracować RAZEM a co za tym idzie mieć wspólne cele. Jeśli widzimy problemy z Water-Fail-Scrum znaczy to, że nasz "zespół" jest tak naprawdę grupą ludzi a niekoniecznie zespołem, lub jego członkowie po prostu nie wiedzą jak wygląda praca zespołowa i czym różni się od pracy grupowej.

Scrum sam w sobie nie narzuca kolokacji zespołu, niemniej jednak sugeruje ją. Ja też to Wam sugeruję! Posadzenie testerów bliżej programistów i fizyczne zbliżenie ich do siebie daje naprawdę bardzo dobre efekty mogące poprawić sytuację już od pierwszych dni.

Tutaj przypomina mi się historia początków mojej pracy jako tester w Code Sprinters - raczej zwinnej firmy, do której przyszedłem z klasycznej kaskadowo, CMMI-owej organizacji. Od samego początku zostałem posadzony w jednym pokoju, przy jednym dużym stole z programistami, co było niemałą odmianą w stosunku do wcześniejszej pracy w zespole testowym odizolowoanym od programistów trzema piętrami biurowca. Już

pierwszego dnia znalazłem błąd w oprogramowaniu więc postanowiłem go zgłosić do bugtrackera. Chciałem się wykazać, wiec zrobiłem ładne screenshoty, opisałem krok po kroku wszystko co robiłem by możliwe było odtworzenie błędu, dodałem odpowiednie kategorie i ustaliłem priorytet. Po chwili wszyscy w pokoju dostali maila z informacją o zgłoszonym błędzie. Jakie było moje zdziwienie, gdy pierwszą reakcją jednego z kolegów - programistów było pytanie: "PO CO to zgłaszałeś?". Odpowiedzi miałem conajmniej kilka i każda z nich była bardzo dobra, w myśl dobrych praktyk testerskich etc. Niemniej jednak jego argumenty były mocniejsze - dużo szybsze i mniej pracochłonne było by po prostu powiedzenie na głos, że znalazłem taki błąd we właśnie implementowanej funkcjonalności. Docelowo poprawka i retesty zajęła znacznie mniej czasu niż samo wypełnianie zgłoszenia. Tego dnia nauczyłem się, że dużo lepiej jest pogadać z programistami jeśli mamy taką możliwość niż pisać bezsensowną dokumentację. A gdy błędy poprawiamy na bieżąco zamiast odkładać je na później to nie ma problemu z tym, że coś nam umknie i nie zostanie poprawione. Na bugtracker trafiały jedynie zgłoszenia rzeczy, które nie były na tyle pilne by zajmować się nimi od razu - nie były szkodliwe i uciążliwe dla klientów.

Kolejnym krokiem rozwiązania problemów związanych z brakiem pracy zespołowej i komunikacją w zespole, w razie gdyby sama kolokacja zespołu nie zadziałała od razu, jest wprowadzenie limitu Work in Progress (WIP). Oznacza to, że narzucamy zespołowi (czy raczej podsuwamy im taką genialną myśl) ilość zadań, które mogą być "rozgrzebane" w danym momencie.

Na początek proponuje limit WIP = 1! Teraz powiecie, że to niedorzeczne - jak grupa np. 7 osób ma pracować nad jednym wymaganiem?! Przecież będą sobie ciągle wchodzić w drogę! Szczerze - nie wiem jak, ale wiem, że developerzy to inteligentne

bestie i wymyślą jak to zrobić najbardziej efektywnie - tak właśnie buduje się samo-organizację zespołu. Przykładowe studium przypadku z zastosowania tego rozwiązania wygląda zazwyczaj mniej więcej tak:

- proponujemy zespołowi limit Work In Progress = 1,
- zespół wyraża oburzenie i stwierdza, że przecież to głupie i nie będzie działało, ale obiecują spróbować,
- testerzy na początku iteracji jak zwykle się nudzą (ale krócej niż zwykle),
- programiści oddają pierwsze wymaganie i zaczynają się nudzić,
- testerzy pod presją ze strony programistów ostro testują, by jak najszybciej oddać wymaganie,
- w końcu z nudów programiści pomagają testerom testować, lub piszą jakieś automaty,
- przy następnym wymaganiu testerzy dostrzegają, że w zasadzie jeśli usiądą razem z programistami i zaczną wcześniej testować, to szybciej zakończą dane wymaganie,
- programiści, dzięki pracy nad testowaniem poprzedniego wymagania lepiej rozumieją pracę testerów i są bardziej otwarci na współpracę z nimi,
- poprawia się komunikacja z testerami, testowanie zaczyna sie znacznie wcześniej niż w momencie "oddania" funkcjonalności do testów.
- do układanki możemy dopisać analityków, architektów etc…

Oczywiście przeważnie będzie to trwało trochę dłużej niż jedną iterację, zanim zespół się dotrze, ale nawet ze staty-

stycznego punktu widzenia, tego typu praca nie powinna mieć zasadniczego wpływu na ogólną prędkość zespołu. Mało tego, empirycznie dowiedziono[8], że tego typu podejście (ang. Single Piece Flow) jest bardziej efektywne niż praca nad kilkoma zadaniami równocześnie.

Aby tego typu praca była możliwa przy minimalnym ryzyku, na Sprint Backlogu zadania powinny być w takiej samej kolejności jak na Product Backlogu. Dzięki temu w najgorszym wypadku, gdy skończy się Sprint, będziemy mieli *zrobione do końca* najważniejsze zadania, a te, które nie zostały zakończone, będą zrobione w kolejnej iteracji (jeśli będzie to nadal miało sens - pamiętajcie, plany się zmieniają!).

[8] http://youtu.be/JoLHKSE8sfU

Czym więc jest Agile?

Jeśli twoja organizacja jest w stanie *wydawać inkrementalnie oprogramowanie* i na tej podstawie *uzyskiwać wartościowe informacje zwrotne* dotyczące tego *czy to, co produkujecie jest tym, czego potrzebuje rynek* to z pewnością jesteście bliżej Agile niż dalej.

Jeśli ta *informacja jest szybka* - maksymalnie co kilka tygodni to istnieje duże prawdopodobieństwo, że będziecie w stanie bardzo szybko *reagować na potrzeby rynku* i zaoszczędzicie dużo pieniędzy, które prawdopodobnie zainwestowali byście w tworzenie czegoś, co jest niepotrzebne użytkownikom.

Jeśli oprócz informacji zwrotnej na temat produktu jesteście w stanie pozyskać informacje na temat *efektywności waszej pracy* i *skuteczności stosowanych metod*, a na tej podstawie jesteście w stanie *ciągle udoskonalacie* to, w jaki sposób wytwarzacie oprogramowanie to w zasadzie nic więcej nie potrzeba do tego, by wasza transformacja Agile stała się ciągłym procesem.

Jeśli ponadto stworzyliście sobie środowisko do *bezpiecznego popełniania błędów* zarówno tych w kodzie jak i tych w stosowanych metodach to gwarantuję wam, że to wszystko razem sprawi, iż będziecie uczyć się i doskonalić swoje metody pracy bardzo, bardzo szybko.

Agile można podsumować w dwóch słowach: *"Inspect & Adapt"*. Eksperymentujemy i sprawdzamy rezultaty naszych eksperymentów, a na ich podstawie planujemy nowe. Wszystko w jak najkrótszych pętlach i odpowiednim, bezpiecznym i transparentym środowisku.