Inżynieria systemów informatycznych
Przegląd literatury

Andrzej Pióro, Arkadiusz Kniaź, Hubert Kozubek,
Jakub Grunas, Nikola Miszalska, Paweł Gelar

Grudzień 2021

Spis treści

1 TDD. Sztuka tworzenia dobrego kodu

1.1 Streszczenie

Kent Beck w książce pt.:"TDD. Sztuka tworzenia dobrego kodu."pokazuje korzyści z programowania sterowanego testami. Pokrótce technika ta polega na trzymaniu się dosyć prostego schematu. Zaczynamy od napisania testu który ma sprawdzać jeszcze nie istniejący kod. Następnie naszym celem jest napisanie kodu który sprawi zaliczenie wcześniej napisanego testu, przy czym kod ten nie musi być wyrafinowany może on wykorzystywać pewne rzeczy których normalnie byśmy nie napisali, bądź nie zostawili do wglądu osób postronnych. Możemy sobie na to pozwolić, ponieważ najważniejszym zadaniem jest zaliczenie wszystkich testów. Następnym krokiem natomiast jest refaktoryzacja, eliminuje się w niej wszelkie duplikacje i czyści poprzednio powstały kod. Na koniec zaczynamy wszystko od początku czyli nowego testu. W pierwszej części książki pokazane jest szczegółowo pisanie kodu sterowanego testami na przykładzie portfela wielowalutowego . Autor zaczyna od stworzenia planu aplikacji i wypisania w nim naszych oczekiwań. Jesteśmy świadomi tego, że nie jest to plan ostateczny. Pomimo tego, że celem naszej aplikacji będzie możliwość dodawania różnych walut, nie możemy od razu do tego przejść. Autor tłumaczy nam, że musimy stosować metodę małych kroków. Możemy teraz przystąpić do programowania, jednak nie jakieś klasy lub funkcji a naszego pierwszego testu. Test ten powinien sprawdzać jedną z podstawowych możliwości naszego programu. Kent Beck zaczyna od testu sprawdzającego możliwość mnożenia ilości dolarów przez skalar. Test od razu się załamuje więc przystępujemy do napisania klas i pól które zapewnią zaliczenie testu. Następnie przechodzimy do refaktoryzacji naszego kodu. Następnie autor przy pomocy metod times i equals dla klasy Dollar pokazuje nam dzielenie testów na mniejsze. Dowiadujemy się, że przydatna jest możliwość dowiedzenia się przez które metody załamują nam się testy. W tym celu powinniśmy unikać korzystania z wielu metod w jednym teście. Następnie podczas rozwoju programu powstaje klasa Franc. Przy jej okazji widzimy jak czasem skomplikowany może być proces usuwania duplikatów. Autor tworzy nową klasę Money. Następnie napotykamy problem refaktoryzacji testów. Kent Beck sugeruje aby nie być zbyt pochopnym z usuwaniem testów i dobrze się zastanowić przed takim czynem. Dobrym pomysłem może wtedy być zapisanie dlaczego dany test został usunięty. W końcu docieramy do momentu w którym wszystko co wcześniej zapewniały nam klasy Dolar i Franc może zapewnić nam klasa Money. I tutaj konieczne było usunięcie kilku testów, ponieważ sprawdzały one poprawne działanie klas których już nie ma. Podczas implementacji powstaje również klasa Bank, która odpowiedzialna jest przewalutowywanie. Wszystkie zmiany wymagają jednak refaktoryzacji. Autor pozostawia jednak czytelnika z projektem można by powiedzieć nieskończonym i przechodzi do podsumowania pierwszej części. Autor przypomina nam podstawowe elementy TDD takie jak stosowanie preparowania, triangulacji i oczywistej implementacji do doprowadzania do zaliczenia testów i możliwość dostosowywania częstości testowania do potrzeb. Pokazuje nam również statystyki związane z napisanym kodem. W drugiej części przechodzimy innego projektu, tym razem będzie to narzędzie do testowania. Zmieniamy również język na Python, który najlepiej według autora spełni to zadanie. Narzędzie to będzie testowane metodą TDD przy użyciu tego narzędzia. Przechodzimy przez metody, które mają sprawdzać czy testy zostały wywołane i takie które mają na celu zbieranie wyników o wynikach testów. Podczas pracy nad kolejnymi funkcjonalnościami wykorzystujemy dotychczas napisane oprogramowanie. Ważnym elementem naszego kodu jest fragment pozwalający zapanować nad awariami. W podsumowaniu tej części autor pokazuje, że może być ona przydatna gdy sami będziemy chcieli zaimplementować własne środowisko testowe. W trzeciej części poznajemy wzorce dla programowania sterowanego testami, które możemy sobie wybrać aby jak najlepiej się dopasowały do naszego stylu pracy i wspomagały nas w refaktoryzacji kodu. Część ta może być różnie odbierana przez różnych czytelników. W zależności od stopnia zaawansowania może ona służyć do zapoznania się z technikami programowania z zastosowaniem TDD lub do zgłębienia konkretnych kroków wykonywanych w danym wzorcu. Autor zaczyna od podstawy wzorców TDD. Omawia tutaj między innymi testowanie, izolowanie testów, tworzenie listy testów, zaczynanie od testów, a dalej zaczynanie od asercji i przygotowywanie danych testowych. Potem przechodzi do wzorca czerwonego paska, gdzie skupia się na tym w jakich sytuacjach pisać testy oraz rozróżnia ich rodzaje. Autor dodaje jednak też fragment o przerwach i dostosowanym miejscu pracy. Następnie przechodzi do wzorców testowania, które są jak autor podaje szczegółowymi technikami testowania. Pierwsza z nich test wtórny pozwala radzić sobie ze zbyt dużymi testami. Następne to atrapa wykorzystamy ją, gdy na przykład nie chcemy działać na całej bazie danych, samopodstawienie do na przykład sprawdzania komunikacji pomiędzy obiektami, łańcuch-dziennik do sprawdzania kolejności, symulowana katastrofa do przypadków rzadkich. Autor proponuje zawieszony test gdy programujemy sami czyli zostawić niedokończony problem, żeby następnego dnia szybciej wcią-

gnąć się w rytm pracy i czysta sprawa gdy programujemy w zespole czyli kończyć pracę danego dnia z wszystkimi testami zaliczonymi, aby ułatwiać pracę w zespole. Następnie przedstawione są wzorce zielonego paska takie jak preparacja czyli na przykład ustawianie stałych tylko po to, żeby jak najszybciej zaliczyć test, triangula czyli wyprowadzanie abstrakcji z testów, implementowanie oczywiste na przykład gdy implementujemy proste operacje i jedno na wiele... czyli o operacjach wykonywanych na kolekcjach. Potem są wzorce xUnit, czyli kolejno asercja do sprawdzenia czy testy poprawnie funkcjonuję, fikstury czyli usprawnienie tworzenia obiektów wspólnych dla wielu testów, fikstura zewnętrzna czyli zwalnianie zewnętrznych zasobów przydzielonych na potrzeby testu, metoda testowa czyli reprezentowanie przypadku testowego, test wyjątku czyli testowanie występowania spodziewanych wyjątków, wszystkie testy czyli zorganizowanie sprawnego uruchamiania wszystkich testów dla aplikacji. Następnie są przedstawione wzorce projektowe. Potem autor rozwija temat refaktoryzacji. Na koniec Kent Beck podaje odpowiedzi na kilka pytań na przykład Jak wskoczyć w nurt TDD? i Dlaczego TDD w ogóle działa?

1.2 Recenzja

„TDD. Sztuka tworzenia dobrego kodu" to pozycja autorstwa Kenta Becka- autorytetu w dziedzinie programowania, w której przedstawione jest tak zwane programowanie sterowane testami. TDD, czyli skrót od Test-Driven Development to technika tworzenia oprogramowania, której główną ideą jest pisanie testów przed utworzenie właściwego kodu. Książka rozpoczyna się wstępem, w którym autor wprowadza nas do tytułowej metody programowania, przedstawia jej najważniejsze zasady oraz zapoznaje z 5 kolejnymi krokami, które jak sam nazywa, stanowią swoistą rytmikę TDD. Książka została w przemyślany sposób podzielona na trzy części. Dodatkowo każda z nich podzielona jest na stosunkowo krótkie podrozdziały. Pozwoliło to na merytoryczne podejście do wypełnienia treścią, dzięki czemu czytelnik jest pozbawiony wrażenia, że czyta jeden długi tekst przepełniony wiedzą. Zawarte treści zostały przekazane w bardzo przystępny i przyjemny dla odbiorcy sposób. Autor nie tylko dokładnie tłumaczy nowe techniki, ale również podaje wiele przykładów, na których możemy się wzorować podczas lektury. Dodatkowo każdy rozdział kończy się krótkim podsumowaniem, w którym zwięźle przestawione są zmiany, które do tej pory udało nam się dokonać, zadania, które jeszcze stoją przed nami, oraz sformułowane wnioski. To na pewno duże ułatwienie dla czytelnika, który dopiero co poznaje metodę TDD. W pierwszej części przeprowadza nas przez projekt systemu wykonującego obliczenia w różnych walutach. Pokazując kolejne etapy pisania testów oraz kodu dokładnie tłumaczy na czym polega metoda TDD. Niejednokrotnie również przypomina o celu stosowania promowanej przez siebie techniki, którym jest stworzenie „przejrzystego kodu, który działa". Po pierwszym rozdziale, mimo swoich dużych starań, Kent Beck nie do końca zachęcił mnie do stosowania metody TDD. Nie byłam przekonana, czy tyle trudu i kilkadziesiąt stworzonych testów było warte tego, aby na koniec zobaczyć kilkulinijkowy, raczej intuicyjny kod. Choć na pewno nie można odmówić, że w metodzie tej można znaleźć również wiele pozytywów. W drugiej części autor dokładnie i po kolei omawia sposób tworzenia testów. Dobrym posunięciem z jego strony było użycie w tej części języka Python, który obecnie jest bardzo popularny. Dodatkowo plusem jest również to, że książka została napisana w sposób niewymagający od czytelnika wysokiego zaawansowania w językach, które użyto. Dzięki dokładnemu tłumaczeniu, podstawowa ich znajomość wystarczy, aby zrozumieć pisane kody. Bardzo pozytywnie odebrałam ostatni rozdział książki nazwany „Doskonaląc TDD", w którym można znaleźć odpowiedzi na wszystkie nurtujące pytania, które pojawiały się w głowie w trakcie czytania tej pozycji. Autor niezwykle trafnie przewidział jakie wątpliwości pojawią się u czytelnika, który dopiero poznaje omawianą metodę, dlatego uważam ten rozdział za bardzo udane zakończenie książki. Bez niego kończąc lekturę na pewno czułabym duży niedosyt. Treść jest urozmaicona humorystycznymi wtrąceniami i komentarzami autora skierowanymi wprost do czytelnika, napisanymi z lekkim przymrużeniem oka. I mimo, że zapewne nie każdemu spodoba się to poczucie humoru, uważam ten zabieg za bardzo udany. To jest jeden z powodów, które sprawiają, że książkę czyta się bardzo lekko. Co jest raczej miłym zaskoczeniem, biorąc pod uwagę, że książka jest o technice programowania.

Zasadnicza teść książki została obudowana wcześniej wspomnianym wstępem oraz skorowidzem, z którego szybko możemy odczytać, na której stronie znajduje się interesujący nas temat. To na pewno znaczące ułatwienie, w przypadku gdy po jakimś czasie chcemy wrócić do konkretnego zagadnienia. Po przeczytaniu można odnieść wrażenie, że książka jest raczej dla osób początkujących w metodzie programowania sterowanego testami. Przypuszczam, że te osoby, które początki już mają za sobą lub są bardziej zaawansowane, mogą nie odnaleźć zbyt wiele nowej wiedzy, a co najwyżej odświeżyć sobie podstawy. Recenzowana książka jest bardzo dobrym wprowadzeniem do tematu, którym jest programowanie sterowane testami. Ma dużą wartość dydaktyczną i na pewno

będzie bardzo przydatna w samokształceniu. Główne jej zalety to niezwykle przystępna forma przekazania wiedzy oraz budowa książki. Jej treść może zainteresować zarówno osoby działające w branży, jak i studentów kierunków informatycznych oraz pasjonatów. Ogólny mój odbiór tej pozycji jest pozytywny.

1.3 Dyskusja/krytyczne spojrzenie

W swojej książce Kent Beck proponując czytelnikom nowatorskie, nieoczywiste podejście do procesu pisania programu – metodę Test Driven Development – naraża się, jak każdy pomysłodawca nowej idei, na głosy nie tylko wsparcia, lecz także protestu od osób, którym metoda do nie przypadnie gustu. Największym i najczęściej pojawiającym się zarzutem, dotyczącym implementacji metody TDD w prawdziwych projektach, jest zbędna praco- i czasochłonność. Aby krok po kroku podążać zgodnie ze wskazówkami zawartymi w omawianej książce, musielibyśmy każdej, nawet najprostszej, funkcjonalności dedykować dodatkową pracę i czas na jej początkowe przetestowanie. W związku z tym zmuszeni jesteśmy pisać około dwukrotnie więcej. Ponadto taki proces wymaga ciągłej zmiany nastawienia i zamysłu pisanego kodu, tzn. co chwila przerzucamy się z pisania testów do tworzenia docelowego kodu programu. Z drugiej strony, takie podejście pozwala nam nie tylko na uniknięcie błędów w kodzie, których naprawa mogłaby zająć więcej niż czas przeznaczony na pisanie zgodnie z techniką TDD, lecz także na skupienie się na pisaniu programu małymi krokami, co ma swoje oczywiste zalety. Osiągamy dużo lepsze panowanie oraz dogłębne zrozumienie działania aplikacji. Kolejnym ważnym tematem podjętym przez autora jest refaktoryzacja kodu, dzięki której pozbywamy się zbędnych fragmentów oraz duplikacji po przetestowaniu danej funkcjonalności. Zauważmy jednak, że stosując metodę TDD i skupiając się początkowo jedynie na pozytywnym przejściu testów, możemy pozwolić sobie na napisanie różnych wspomnianych zbędnych fragmentów kodu, które później mogą okazać się trudne do wyeliminowania, nie ingerując w poprawne działanie aplikacji. Dodatkowo, posiadając dużą liczbę kodu, taka refaktoryzacja również zajmuje bardzo wiele czasu. Przeglądając Internet w poszukiwaniu opinii na temat analizowanej lektury, natrafiamy na zróżnicowane głosy: te krytyczne podobne do przedstawionych przez nas powyżej oraz te wciąż jednogłośnie chwalące metodę TDD, mimo upływu już ponad 20 lat od jej zaproponowania, i nazywające książkę Kenta Becka ponadczasowym klasykiem. Chcielibyśmy przytoczyć teraz niektóre z tych opinii. W 2014 roku duński programista David Heinemeier Hansson (DHH) umieścił na swoim blogu post (https://dhh.dk/2014/tdd-is-dead-long-live-testing.html) o wymownym tytule „TDD is dead". W poście tym Hansson wymienia różne wady, w jego opinii, procesu TDD, nazywanego inaczej w poście test-first. Głównym postawionym zarzutem jest fakt, że każdy, kto implementuje metodę TDD traci jakąkolwiek elastyczność pisania kodu i musi skupiać się głównie na pisaniu, często zbędnych, testów a nie programu. Ponadto programista traci czas i uwagę na wymyślanie przeróżnych sytuacji, w których jego kod może nie zadziałać, zamiast (znowu) skoncentrować się na pisaniu kodu. Co ciekawe, w odpowiedzi na wspomniany powyżej artykuł, sam Kent Beck udostępnił na swoim facebook'owym profilu post zaczynający się słowami „RIP TDD", w którym przyznał rację DHH i pisze, że musi odkryć nowe techniki, pozwalające na rozwiązanie licznych problemów przy programowaniu. Mimo tego, w Internecie możemy znaleźć opinie wielu programistów, którzy wciąż uważają metodę TDD za potrzebną, niezbędną i niezastąpioną. Przytoczyć tu można między innymi głos Kelly Emo, jednej z deweloperek w firmie Hewlett-Packard, która w artykule dla SD Times mówi, że zarzuty TDD stawiają głównie ci, którzy nie wiedzą do czego metoda ta służy. W swojej wypowiedzi wspomina o takich zaletach TDD jak znajdowanie potencjalnych słabości programu jeszcze przed jego napisaniem, czy możliwość lepszego dostosowywania się do potrzeb użytkownika pisząc małe fragmenty kodu. Podsumowując, możemy zauważyć, że technika Test Driven Development, będąca głównym tematem omawianej lektury, ma zarówno swoje wady, jak i zalety. Jako początkujący programiści na pewno możemy odnieść wiele korzyści z jej nauki i implementacji we własnych projektach, niezależnie od tego, czy będzie nam ona potrzeba w późniejszych etapach kariery. W przyszłości, przy tworzeniu nowego projektu, zawsze warto zastanowić się, czy implementacja TDD nie byłaby słuszna i przydatna w pracy.

1.4 Osobiste przemyślenia

1.4.1 Andrzej Pióro

Metoda Test Driven Development przedstawiona w książce bardzo mnie zaciekawiła. Została bardzo zrozumiale, od podstaw przedstawiona i nie wydaje się być szczególnie trudna w zastosowaniu. Niezwykle przydatne były również przedstawione przykłady. Tym bardziej , gdy zapamięta

się kolejne kroki, o których sam autor często przypomina. Jestem pewien, że przy najbliższych możliwych okazjach będę próbował wdrożyć tą metodę przy pisaniu swoich kodów. Myślę, że mimo tego, że wydaję się dość czasochłonna, to można ustrzec się przed nieoczekiwanymi błędami i w rezultacie zaoszczędzić czas. Podobało mi się również to, że autor dużą cześć poświęcił refaktoryzacji kodu, ponieważ jest to ważna część w pisaniu programu. W książce duża uwaga została zwrócona na to, jak ważny jest również ten etap. Negatywnie za to odebrałem przewijające się często humorystyczne przerywniki, który wytrącały mnie z uwagi podczas czytania książki. Uważam, że tego typu pozycje, które mają cel głownie dydaktyczny, nie powinny ich zawierać, ponieważ są mocno rozpraszające.

1.4.2 Arkadiusz Kniaź

Lektura książki pt.:"TDD. Sztuka tworzenia dobrego kodu.ąutorstwa Kent Beck okazała się być jedną z ciekawszych pozycji jakie miałem okazję przeczytać w ostatnim czasie. Książka dzięki podzieleniu na trzy części była przyjazna w odbiorze. Dla mnie osobiście książka okazała się pomocna w lepszym zrozumieniu programowania obiektowego w języku JAVA. Przykład przedstawiony w pierwszej części książki był wystarczająco prosty i przejrzysty, żeby bez większego problemu wdrożyć się w lekturę, a jednocześnie dobrze pokazywał proces programistyczny z wykorzystaniem techniki Test Driven Development. Przed spotkaniem z tą książką technika ta była mi obca, nie byłem nawet świadom, że istnieją jakiekolwiek techniki programowania, a mój własny kod powstawał raczej w nieregularnych przypływach weny bez żadnego uporządkowania. Tym większe było moje zaciekawienie gdy dowiedziałem się, że istnieją owe techniki, a ta konkretna wydaje się wprowadzać przyjazny schemat na którym można by się oprzeć. Ów schemat został moim zdaniem dobrze źareklamowany". Zamierzam spróbować techniki TDD przy najbliższej okazji i przetestować ją na samym sobie, żeby przekonać się czy jest tak dobra jak podaje nam autor.

1.4.3 Hubert Kozubek

Książka otworzyła przede mną drzwi do zupełnie nowego podejścia (o którego istnieniu wcześniej nie miałem pojęcia) wobec pisanego kodu. Abstrahując od faktu, czy metoda Test Driven Development jest w 2021 roku wciąż odpowiednia, świadomość o niej na pewno poszerzy moje horyzonty i umożliwi alternatywne spojrzenie na pisany kod. Osobiście, metoda pisania najpierw testów, a następnie kodu wydaje mi się na tyle inspirująca, że planuję zastosować ją do kilku projektów w przyszłości, aby nabrać ciekawego doświadczenia, a także samemu, na bazie własnej praktyki móc zabrać głos w dyskusji o jej przydatności. Odnosząc się natomiast do samej książki, bardzo podobała mi się jej ustrukturyzowana forma obrana przez autora – podział lektury na części, konkretne przykłady, konkretne zasady oraz kod pisany krok po kroku, umożliwiający „interaktywne" czytanie wraz z samodzielnym pisaniem kodu z książki. Podsumowując, oceniam książkę bardzo wysoko, a samą metodę uważam za wartą przetestowania.

1.4.4 Jakub Grunas

Kent Beck w swojej książce, mimo rzeczowej treści, ciekawych przykładów i przystępnej struktury, nie przekonał mnie do implementowania metody Test Driven Development we własnych projektach. Uważam, że takie podejście nie pasuje do mojego stylu pisania kodu i przyniosłoby mi więcej komplikacji niż pożytku. Ma to związek z czasochłonnością i rygorystycznymi zasadami omawianej techniki, które zmuszają do ciągłej weryfikacji własnego kodu. Nawet jeszcze przed napisaniem kodu należy zastanowić się, co może stanowić potencjalne problemy, aby wykryć je wszystkie zawczasu, co osobiście nie podoba mi się. Cieszę się jednak, że dane było mi przeczytać omawianą książkę, ponieważ dzięki temu wiem o istnieniu techniki Test Driven Development. Rozumiem także przedstawione zalety metody oraz ludzi, dla których jest ona użyteczna w projektach, a sam być może w przyszłości zmienię swoje nastawienie i również zacznę ją stosować pisząc własny kod.

1.4.5 Nikola Miszalska

Książka TDD autorstwa Kenta Becka zrobiła na mnie pozytywne wrażenie. Jest napisana w sposób bardzo przyjazny dla odbiorcy oraz jest łatwa w czytaniu. Było to dla mnie dość dużym zaskoczeniem, ponieważ po książce o takiej tematyce spodziewałam się raczej ciężkiej do przeczytania treści, przepełnionej suchymi informacjami. Cała treść jest przekazana bardzo zrozumiale, a książka ma duża wartość dydaktyczną. Mimo to, sama metoda programowania sterowanego testami mnie nie przekonała. Co najwyżej zaciekawiła, ponieważ na pewno nigdy wcześniej o niej nie

słyszałam. Wydaje mi się jednak, że nakład czasu , który trzeba poświęcić w celu napisania kodu jest zupełnie niewspółmierny do efektu, który otrzymujemy na końcu. Możliwe, że z ciekawości wypróbuję ją w przyszłości, przy możliwej okazji, ale wątpię abym wdrożyła ja na stałe. Choć oczywiście nie można być niczego pewnym, ponieważ może w przyszłości będzie potrzeba korzystania z niej. Ta opinia nie wpływa jednak na mój odbiór tej pozycji, ponieważ dotyczy już samej techniki. Na pewno książka jest warta zapoznania się z nią.

1.4.6 Paweł Gelar

Podczas lektury książki szczególnie zainteresowało mnie w metodzie TDD to, że wydaje się stosunkowo nieskomplikowana. Kent Beck pokazuje czytelnikowi pięć zasad do których trzeba się stosować podczas tworzenia kodu. Interesujący jest fakt, że pod tymi pięcioma zasadami można schować tak potężne narzędzie. Jednak przejście od teorii do praktyki niestety nie należy do najprostszych. Doceniam to jak dużą część książki autor poświęcił na kod i przykłady, pozwoliło mi to na lepsze zrozumienie całego procesu. Z całej książki najciekawsze jednak wydają mi się być przykłady z drugiej części książki. Na początku gdy je przeczytałem, nie wiedziałem o co chodziło w tym fragmencie autorowi, ale po ponownym i w większym skupieniu przeczytaniu tego fragmentu, mam wrażenie, że zrozumiałem te przykłady. Wydały mi się one po tym dosyć oczywiste, nie zawsze piszemy piszemy coś przy pisaniu czego, korzystamy z tego co właśnie piszemy. Ostatecznie jednak uważam, że metoda TDD jest przydatna i pozwala dużo ułatwić późniejszy proces sprawdzania poprawności kodu.

1.5 Powiązana literatura

1. Gerald Weinberg “Systems Thinking. Quality Software Management” - autor sam w swojej książce odwołuje się do tej pozycji.

2. Modern C++ Programming with Test-Driven Development: Code Better, Sleep Better, JeffLangr - książka o korzystaniu z TDD w języku programowania C++.

3. Kent Beck ”Smalltalk Best Practice Patterns” - autor odwołuje się do tej książki i daje również link do Amazona.

4. Test-Driven Development with Python: Obey the Testing Goat: Using Django, Selenium, andJavaScript, Harry J. W. Percival - książka o korzystaniu z TDD w Pythonie.

5. Extreme Programming Explained: Embrace Change, Kent Beck - autor odwołuje się do tej książki w jednym z rozdziałów.

2 Agile. Przewodnik po zwinnych metodykach programowania

2.1 Streszczenie

Książka „Agile, Przewodnik po zwinnych metodykach programowania" przedstawia techniki i metodyki związane ze zwinnym wytwarzaniem oprogramowanie. Na początku książki przedstawiona jest cała idea oraz definicja Agile. Agile jest przedstawiony jako zestaw metod i metodyk, a także jako sposób myślenia, które mają pomóc zespołom w efektywnym myśleniu i pracy oraz w podejmowaniu lepszych decyzji. Wyjaśnione są cele samej książki, podejścia zwinnego, oraz zakreślone są już pierwsze problemy jakie można napotkać wprowadzając od zera podejście zwinne. Struktura samej książki jest zrobiona w taki sposób, aby ułatwić czytelnikowi zrozumienie zagadnień przedstawionych w książce. W następnych rozdziałach są przedstawione wartości Agile. Występuje również pierwsze opowiadanie, w formie przykładu, jak można zastosować metodyki Agile i z jakimi problemami może się spotkać zespół w trakcie ich wprowadzania. Przedstawiony jest też model kaskadowy jako przeciwieństwo do podejścia zwinnego. Ideą modelu kaskadowego jest poświęcenie dużej ilości czasu na starcie projektu, w celu ustalenia dokładnych wymagań. Następnie rozpoczyna się etap tworzenia projektu, implementacji kodu wraz z sporządzaniem bardzo dokładnej dokumentacji. Na końcu następuje sprawdzenie końcowego produktu i jego konserwacja. Model kaskadowy nie pozostawia miejsca na zmiany początkowych założeń. Metodyki zwinne stawiają na lepszą komunikację pomiędzy członkami zespołu oraz planowanie grupowe w przeciwieństwie do stworzenia szczegółowego planu na starcie i bezrefleksyjne trzymanie się go. W dalszej części książki przedstawiane i dokładnie wytłumaczone zostają zasady Agile. Jest ich dwanaście, występują w książce występują w podanej kolejności:

1. Naszym priorytetem jest zapewnianie satysfakcji klientów dzięki szybkiemu i ciągłemu dostarczaniu wartościowego oprogramowania

2. Jesteśmy otwarci na zmiany wymagań nawet w późnych etapach prac. Procesy zwinne pozwalają poradzić sobie ze zamianami w celu zapewnienia klientom przewagi konkurencyjnej.

3. Często dostarczamy działające oprogramowanie; zajmuje to od kilku tygodni do kilku miesięcy, przy czym preferowane są krótsze okresy.

4. Najwydajniejszym i najskuteczniejszym sposobem przekazywania informacji zespołowi programistów i komunikowania się w jego ramach jest bezpośrednia rozmowa.

5. Pracownicy biznesowi i programiści muszą codziennie wspólnie pracować nad projektem

6. Opieraj projekt na zmotywowanych osobach. Zapewnij im potrzebne środowisko i wparcie oraz uwierz, że wykonają zadanie.

7. Główną miarą postępów jest działające oprogramowanie.

8. W procesach winnych ważna jest możliwość utrzymania tempa programowania. Sponsorzy, programiści i użytkownicy powinni móc w nieskończoność utrzymywać stałe tempo pracy.

9. Ciągła troska o techniczną doskonałość i dobry projekt są zgodne z podejściem zwinnym

10. Konieczna jest prostota rozumiana jako sztuka maksymalizowania niewykonanej pracy

11. Najlepsze architektury, wymagania i projekty są owocem pracy samoorganizujących się zespołów.

12. Zespół regularnie zastanawia się nad tym, jak zwiększyć swoją efektywność, a następnie odpowiednio usprawnia i dostosowuje swoje postępowanie.

Zasady te są dokładnie opisane i wytłumaczone w książce. W kolejnej części lektury przytoczona jest podejście Scrum. Opiera się ono na sprintach, czyli iteracjach projektu, a podstawowy schemat projektów opisuje role i techniki tej metodyki. Do roli zaliczają się Właściciel Projektu, Mistrz Młyna i zespół, natomiast do metodyk możemy zaliczyć sprint, codzienne spotkania czy rejestr zadań. Idea codziennych spotkań pozwala członkom na udzielenie odpowiedzi na pytania co zrobiłem od czasu ostatniego codziennego spotkania?, co zrobię do czasu następnego spotkania?, jakie przeszkody stoją mi na drodze? W dalszych częściach opisany jest sposób w jaki należy planować zadania w podejścia Scrum. Następnie przytaczane są konkretne metodyki należące do podejście Scrum, takie jak historie użytkownika czy wykres postępu prac. W szóstym rozdziale autorzy przechodzą do nowej metodyki zwinnej o nazwie XP czyli Extreme Programming. Podobnie jak Scrum metodyka ta obejmuje wartości i zasady. Na początku poznajemy nowy zespół, który w pracy nad swoim obecnym projektem ma problem z nadgodzinami i pracą w weekendy. Następnie przedstawione są techniki zawarte w XP. Autorzy podzielili je na te związane kolejno z programowaniem, integrowaniem kodu, planowaniem i zespołem. Następnie przedstawione są wartości i zasady XP. Przez cały czas co chwila wracamy do historii naszego nowego zespołu, żeby zobaczyć jak idzie mu praca w metodyce XP. W siódmym rozdziale kontynuujemy poznawanie XP i wracamy do naszego zespołu z poprzedniego rozdziału. Poznajemy tutaj antywzorce i żapachy kodu", czyli oznaki, że programista za bardzo kombinował przy swoim kodzie. Dowiadujemy się o żapachach kodu"takich jak: chirurgia odłamkowa, niesamodzielny kod, bardzo duże klasy, powtarzający się kod, kod spaghetti i kod lasagne. Autorzy mówią również o haczykach w kodzie i popadaniu w pułapkę budowania platformy. Dowiadujemy się jak ważne jest podejmowanie decyzji w ostatnim sensownym momencie, likwidowanie długu technicznego i refaktoryzacja. Poznajemy idee projektowania przyrostowego. Dalej czytamy o technikach energicznej pracy, całego zespołu i o tym, że ważne jest utrzymywanie stałego tempa pracy. Poznajemy przykład UNIX jako tworzonego przyrostowo i składającego się z bardzo samodzielnych części. W ósmym rozdziale metodykę lean, która w odróżnieniu od Scrum i XP nie obejmuje zestawu technik i jest raczej sposobem myślenia. Poznajemy wartości myślenia odchudzonego i możemy zauważyć, że wiele z nich poznaliśmy już przy okazji Scrum i XP. Dowiadujemy się o myśleniu w kategoriach opcji i programowaniu opartym na opcjach. Poznajemy nowy zespół pracujący nad aplikacją fotograficzną i dowiadujemy się jakie problemy może przynieść kreowanie herosów i myślenie magiczne. Następnie czytamy o dostrzeganiu i eliminowaniu marnotrawstwa. Poznajemy narzędzia w postaci mapy strumienia wartości i techniki pięciu pytań dlaczego. Następnie przechodzimy do kolejnej wartości lean, która jest dostarczaniem tak wcześnie, jak to tylko możliwe. Uczymy się wykorzystywać wykres warstwowy do wizualizacji prac w toku oraz radzić sobie z zatorami. Na koniec poznajemy system pull jako sposób prowadzenia projektów. W dziewiątym rozdziale poznajemy kanban, czyli metodzie doskonalenia procesów. Wracamy do zespołu poznanego w poprzednim rozdziale. Poznajemy zasady podejścia kanban. Czytamy o pojęciach używanych w tym świecie takich jak jednostki robocze i mapy przepływu pracy. Dowiadujemy się czym są tablice kanbanowe i jak z nich korzystać. Poznajemy bardzo ważną technikę w podejściu kanbanowym, czyli technikę ograniczania liczby zadań w toku poprzez tworzenie limitów. Dowiadujemy się w jaki sposób działa ta technika i że pomaga nam radzić sobie z problemem przeładownia. Dowiadujemy się jak mierzyć przepływ jednostek roboczych i jak nim zarządzać, poznajemy przy tym prawo Little'a. Dziesiąty ostatni rozdział w pełni został poświęcony dla coachów metodyk zwinnych. Autorzy przedstawiają tutaj w trzech krokach proces uczenia się i po kolei tłumaczą jak powinna wyglądać współpraca z zespołem na każdym z etapów. Wymienionych i opisanych jest również pięć zasad coachingu, czyli: przedsiębiorczość, entuzjazm, kondycja, podstawy i rozwijanie ducha pracy zespołowej.

2.2 Recenzja

Książka autorstwa Andrew Stellman i Jennifer Greene pt.:Agile. Przewodnik po zwinnych metodykach programowania"doczekała się swojej polskiej wersji w 2015 roku w wydawnictwie Helion. Światową premierę książka miała w listopadzie 2014, więc polscy czytelnicy wyjątkowo krótko musieli czekać na jej polską wersję. Autorzy książki wspólnie tworzą oprogramowanie i piszą od 1998 roku. Agile nie jest ich pierwszą książką, Andrew i Jennifer mają ich już kilka na swoim koncie. Pierwszą książką było Applied Software Project Management wydana już w 2005 roku przez O'Reilly Media. A ich kolejne książki to między innymi Head First PMP, Head First C i Beautiful Teams: Inspiring and Cautionary Tales from Veteran Team Leaders. Książka na stronach dziesię-

ciu rozdziałów opowiada o wartościach i zasadach Agile, metodykach Scrum i XP, podejściu Lean i technice Kanban. Na 352 stronach autorzy książki wprowadzają czytelnika w świat zwinnych metodyk programowania. Autorzy za cele książki obierają sobie: zrozumienie przez czytelnika idei napędzających efektywne zespoły Agile oraz wartości i zasady które je łączą, zrozumienie najpopularniejszych szkół Agile czyli Scrum, XP, Lean i Kanban i tego jak wszystkie pomimo różnic mogą być zwinne, nauczenie czytelnika konkretnych praktyk, ale i również przedstawić ramy wartości i zasady, których czytelnik miał zadanie wdrożyć, pomoc w lepszym zrozumieniu własnego zespołu i firmy, aby móc wybrać takie podejście, które najlepiej będzie pasować do naszego nastawienia, ale i pomóc nam i naszemu zespołowi w rozpoczęciu nauki nowego sposobu myślenia, które umożliwi stanie się zwinnym zespołem. Książka w każdym z rozdziałów zawiera kilka elementów, które w założeniu mają pomóc czytelnikowi w lepszym przyswajaniu wiedzy i szybszym zapamiętaniu najważniejszych elementów. Pierwszym elementem są historyjki, nie są one prawdziwym sprawozdaniem z realnych wydarzeń, ale przedstawiają sytuacje i problemy, które bez problemu mogły pojawić się w niejednym zespole programistycznym. Następnym elementem są ilustracje, ponieważ autorzy wiedzą, że ludzie uczą się na wiele sposobów i warto je zawrzeć, aby pomóc wielu czytelnikom w szybszym załapaniu tematu. Kolejnym są powtórzenia i choć mogą wydać się niepotrzebne, a nawet nudzić czytelnika, to pojawiły się w tej książce nie bez przyczyny. Autorzy mieli na celu nie nie stosowanie formy wielu książek technicznych, w których tylko przedstawia i opisuje się konkretne pomysły, a następnie przechodzi dalej, chcieli oni dzięki powtórzeniom pomóc szybciej czytelnikowi dotrzeć do momentu zrozumienia omawianych zagadnień. Kolejnymi elementami są: schemat w jakim przedstawiane są nam nowe zagadnienia, a więc jako uproszczone na początku i stopniowo z czasem pogłębiane i uszczegóławiane, miejscami nieformalny bądź konwersacyjny ton, który ma sprawić, aby książka była bardziej wciągająca, kluczowe punkty oznaczone specjalną ikoną, które mają pomóc czytelnikowi w upewnieniu się czy niczego nie pominął lub nie zdążyło uciec mu z głowy, są one swego rodzaju testem czy przeczytaliśmy poprzedzający je fragment, często zadawane pytania czyli kilka pytań i odpowiedzi na koniec każdego rozdziału, mają one na celu pokazanie czytelnikowi jakie pytania przychodziły innym do głowy i czasami pokazanie innej perspektywy na poruszane tematy, co możesz zrobić dzisiaj ze specjalną ikonką, czyli kilka porad dla czytelnika, które pozwolą mu wykorzystać zdobytą wiedzą najszybciej jak to możliwe, zwykle są to proste zadania do wykonania w naszym zespole programistycznym, gdzie możesz dowiedzieć się więcej czyli fragment w którym poza przypisami możemy znaleźć odniesienia do wielu książek, ponieważ jak przyznają sami autorzy nie są autorami wszystkich zawartych w niej pomysłów i w związku z tym dają nam możliwość poszerzenia naszej wiedzy u źródła, wskazówki dla coachów, czyli zestaw kilku porad specjalnie dla tej jednej grupy czytelników, jednak autorzy przekonują, że może być on przydatny w nauce nie tylko tej grupie czytelników. Książka w swoich pierwszych dwóch rozdziałach omawia Agile, jego główne wartości i zasady którymi powinny kierować się zwinne zespoły. Dalej kolejne dwa przedstawiają czytelnikowi metodologię Scrum, kolejne dwa szkołę XP i kolejne dwa odpowiednio Lean i Kanban. Na końcu książki znajduje się rozdział przeznaczony specjalnie dla coachów. Cała książka jest przyjemna w odbiorze i dobrze przekazuje najważniejsze informacje.

2.3 Dyskusja/krytyczne spojrzenie

W początkowej części książki "Agile. Przewodnik po zwinnych metodykach programowania" znajduje się stwierdzenie, że nie ma metody, która rozwiązywałaby wszystkie problemy. Autor jednak opisuje konkretne metodyki, jakby były one doskonałym remedium. Nie wspomina ani słowem o możliwych negatywnych stronach wprowadzenia ich. Uważa nawet, że wprowadzenie samych technik zaczerpniętych na przykład ze Scruma bez poznawania wartości kluczowych w tym podejściu będzie miało pozytywny (chociaż niewielki) wpływ na pracę zespołu. Także w przykładach z życia poza inżynierią oprogramowania występuje podobna tendencja. Przykładowo w omawianiu podejścia Kanban "wprowadziliśmy je" w gabinecie lekarskim, by zredukować powstające tam kolejki. Gdy w poczekalni znajdowało się już sześciu pacjentów wizyty kolejnych były przekładane. Nie rozważyliśmy problemów z kontaktem, dojazdem lub innymi planami pacjentów na ten dzień. Natomiast opisując alternatywne podejście "dowódź I kontroluj" (niezwinne) uważa, że problemem jest poproszenie lekarzy o szacunki na temat długości wizyty, gdyż lekarze nie są przygotowywani do szacowania podczas ich edukacji. Często używane są sformułowania typu "w poprzednim rozdziale przekonałeś się, jak zastosowanie technik zwinnych pomaga unikać nadgodzin". Problemem jest to, że historia w poprzednim rozdziale wydawała się wymyślonym przez autora dziecinnym opowiadaniem, które można streścić jako: "w pewnym zespole programistycznym programista Janusz często zmuszany był do zostawania w pracy po godzinach. Zespół zdecydował się wprowadzić podejście programowania ekstremalnego. Janusz dzięki lepszej jakości

kodu nie musiał już zostawać po godzinach w pracy". Jednocześnie równie prawdopodobna dla mnie jako czytelnika byłaby historia "Janusz pracował w firmie, gdzie nigdy nie musiał brać nadgodzin, jednak jego zespół postanowił wprowadzić podejście programowania ekstremalnego. Na skutek tego musiał często zostawać w pracy, bo przez programowanie w parach i pisanie testów jednostkowych wynikały opóźnienia w projekcie". Dlatego uważam, że argumentacja za metodykami zwinnymi w tej książce przez swoją jednostronność jest nieprzekonująca. By udowodnić, że podejście zwinne nie jest pozbawione wad poszukałem takowych w internecie. Wynika z tego, że nie zawsze warto stosować techniki Agile; czasem lepiej pozostać przy podejściu kaskadowym. Na przykład, gdy projekt musi być dostarczony od razu w dobrym stanie, gdy czas i koordynacja są kluczowe lub gdy klienci wolą nie ryzykować. https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/Why-Waterfall
-sometimes-wins-the-Agile-versus-Waterfall-debate. Również poszczególne metodyki mają własne problemy. Na przykład Scrum przez dokładne zarządzanie czasem pracy programisty przeciwdziała innowacji zachodzącej głównie w czasie poza planem lub przewidywalnym systemem. Przywiązuje dużą wagę do narzędzi i procesów, co stoi w sprzeczności z Manifestem programowania zwinnego. Cechuje się hipokryzją wobec zarządu - nie musi on się tłumaczyć z wykonanej pracy, jak programiści. Przeciwdziała zmniejszaniu się długu technicznego i usuwaniu problemów, gdyż są to osobne rzeczy, które właściciel produktu musi uznać za wartościowe. [https://medium.com/@ard_adam/why-scrum-is-the-wrong-way-to-build-software
-99d8994409e5]. Co do samej treści przedstawionej w książce nie mam zastrzeżeń, uważam, że jest to dobry podręcznik do ogólnego zapoznania się z podejściem zwinnym do tworzenia oprogramowania. Nie do końca przekonało mnie opisane w przedostatnim rozdziale podejście Kanban – Rozumiem, że przez ustalenie limitu prac w toku na ustalonym etapie pomoże zmniejszyć ilość jednocześnie tworzonych rozwiązań i skrócić czas przebywania jednego rozwiązania w systemie. Jednak wydaje mi się, że wprowadzenie go spowoduje jedynie utworzenie nowego "niewidzialnego" etapu nie uwzględnionego na tablicy kanbanowej, przez co faktyczna sytuacja się nie zmieni. Na przykład załóżmy, że istnieje limit 6 rozwiązań zaproponowanych przez kierownictwo (przed przystąpieniem do produkcji). Jeśli limit ten zostanie osiągnięty według mnie kierownictwo, gdy wpadnie na nowy pomysł po prostu doda go do właściwego etapu, gdy się zwolni miejsce, wirtualnie wydłużając kolejkę i eliminując cały sens podejścia. Wydaje mi się także, że dużym problemem w kanbanowych limitach prac w toku jest ograniczenie priorytetyzacji: czasem niemożliwe będzie zrobienie czegoś z priorytetem super-hiper-max, gdyby zaistniała taka sytuacja (na przykład ujawnienie podatności w aplikacji), bo limit prac w toku został osiągnięty. - brzmi to jak przeciwieństwo bycia zwinnym. Reasumując uważam, że książka przedstawia świat w czerni i bieli – podejścia zwinne zawsze dają pozytywne rezultaty, pozwalają programistom wracać na czas do domu, a projekty używające modelu kaskadowego zawsze będą sprawdzały się gorzej i są skazane na porażkę oraz dziesiątki dni spędzonych na nadgodzinach, a wcale tak nie jest.

2.4 Osobiste przemyślenia

2.4.1 Andrzej Pióro

Po przeczytaniu książki „Agile, Przewodnik po zwinnych metodykach programowania" miałem mieszane uczucia. Wynikało to głównie z tego, że pomimo książka świetnie opisywała konkretne zagadnienia, to wydawały się że momentami autorzy zbytnio je upraszczali. Pomimo tego książka była świetnym materiałem do nauki Agile. Najbardziej przydatny w zrozumieniu był rozdział opisujący 12 zasad podejścia zwinnego. Dzięki rozwinięciu i opisaniu każdej z nich można było uzyskać dokładniejszy wgląd w podejście zwinne. Zasada która najbardziej zapadła mi w pamięć i postaram się ją zastosować przy najbliższej okazji to codzienna współpraca pracowników biznesowych i programistów. Wydaje się ona być bardzo życiowa, gdyż tak samo jak było wspominane w książce, często osoby z tego samego zespołu mogą mieć bardzo różne cele i sądzić, że reszta drużyny postrzega problem tak samo i dążą do tego samego. Wprowadzenie codziennych spotkań byłoby niezwykle pomocne do usunięcia tego problemu. Dzięki temu, że zasada jest jasno zdefiniowana jako codzienna współpraca a nie np. „częsta", „systematyczna" czy „ciągła" to łatwo można odznaczyć jej wykonanie lub nie, podczas gdy częste czy systematyczne spotkania pozostawiają więcej możliwości interpretacji, przez co czasami można być pewnym, że wykonuje się je dobrze, a jest wręcz odwrotnie. Pomimo, że pozostałe zasady również wydawały się ważne, to wprowadzanie ich wszystkich jednocześnie, jako osobie niedoświadczonej, wydaje mi się nierealne. Z tego powodu postanowiłem rozpędzać się pomału i przy najbliższej okazji wprowadzić zasadę codziennych współpracy pomiędzy pracownikami biznesowymi i programistami.

2.4.2 Arkadiusz Kniaź

Książka pt.:Agile. Przewodnik po zwinnych metodykach programowaniaautorstwa Andrew Stellman i Jennifer Greene należy jednej z tych książek, których lektura wywarła na mnie największy wpływ. Książka wprowadziła mnie w świat Agile, pokazała mi jakie wartości i zasady panują w tym świecie. Zainteresowała mnie podejściem Scrum i jego wpływem na zespół, który z niego korzysta. Pomimo faktu, że nie miałem okazji(lub nie jestem tego świadom) pracować w ściśle kaskadowym podejściu do projektu, to teraz mam wrażenie, że wolałbym w takim nie pracować. Część poświęcona XP pokazała mi techniki takie jak programowanie w parach i TDD. Zainteresowała mnie tym jak w nieoczywisty sposób techniki te mogą przyspieszyć pracę programistyczną. Następnie rozdziały poświęcone Lean i Kanban przestraszyły mnie trochę tym, że mogę trafić w przyszłości na przełożonego, który cechował się myśleniem magicznym. Jak dotąd miałem szczęście nie współpracować z nikim kto w ten sposób podchodziłby do zadań, mam nadzieję, że tak pozostanie. Ostatecznie lekturę tej książki uznaję dobrze spędzony czas, poznałem wiele nowych konceptów i zostałem pozostawiony z ciekawością ich przetestowania.

2.4.3 Hubert Kozubek

Przed przeczytaniem książki zajrzałem do opinii pozostawionych na portalach, które umożliwiały jej zakup. Zdecydowaną większość stanowiły opinie zdecydowanie pozytywne, w tym w większości wystawiały je osoby które dopiero co zaczynały (a przynajmniej tak twierdziły) swoją podróż z Agilem. Naprawdę mnie to zachęciło do przeczytania tej książki, niemniej jednak poczułem się trochę zawiedzionym. Najbardziej zaskakujące w książce były przytoczone przypadki i historie. Odniosłem wrażenie, że autorzy chcieli na siłę przekonać czytelnika do swojego punktu widzenia. Osobiście uważam, że dobre argumenty powinny bronić się same i jedyne co należy zrobić to pomóc w ich zrozumieniu i wyjaśnieniu ewentualnych niedomówień. Dowód anegdotyczny na podstawie historii, która była zaznaczona jako historia wymyślona i fałszywa pozostawiał pewien niesmak. W szczególności jeżeli sama anegdota zdawała się nic nie wnosić i bardziej zagmatwać technikę niż ją rozjaśnić. Najgorsze dla mnie był fakt, że z takim przedstawieniem spotykamy się już na samym początku, gdzie czytamy opowieść o grupce nowych znajomych którzy poznali się w barze i rozmawiali przy piwie. Pomimo tego książkę mógłbym polecić komuś niezapoznanemu z technikami Agile, gdyż zdaję sobie sprawę, że do niektórych takie techniki opowiadania mogą przemawiać bardziej niż rozłożenie na czynniki pierwsze danej metodyki.

2.4.4 Jakub Grunas

Czytając książkę „Agile, Przewodnik po zwinnych metodykach programowania" autorstwa Andrew Stellman i Jennifer Greene można było się spotkać z wieloma metodykami i technikami gotowymi do zastosowania do pracy w zespole lub do projektów osobistych. Najbardziej zaskakujące, okazała się dla mnie metodyka najbardziej oczywista. Metoda ta jest jednym z filarów podejścia zwinnego. Chodzi o tworzenie iteracji danego produktu oprogramowania, najlepiej z jakimś stałym okresem czasowym. Metoda ta jest często wychwalana i polecana przez osoby które komercyjnie pracują w branży związanej z oprogramowaniem. Przez to, że jest tak często wspominana może się wydać czymś oczywistym i domyślnym, coś co każdy robi i stosuje. Z tego powodu może być niedostrzegana. Tak też było w moim przypadku. Ponieważ nie zdążyło mi się pracować nad większym grupowym projektem, a jedynie nad niewielkimi projektami osobistymi, to myślałem że to podejście niepotrzebnym dodatkiem. Jednak po przeczytaniu książki, a szczególnie rozdziału w którym było ona dokładniej wyjaśnione i omówione zacząłem dostrzegać pewne podobieństwa z zespołami przedstawionymi w książce, które stosowały podejście kaskadowe. Jednym z podstawowych błędów był brak możliwości szybkiego reagowania na zmiany, w szczególności gdy poświęciłem już dużą ilość czasu w implementację jakiejś funkcjonalności, przez to wolałem ją kończyć już w takim stanie w jakim to miało wyglądać na początku, zamiast nawet wrócić i pozmieniać niektóre elementy. Myślę że to właśnie na tą metodę, czyli dużo mniejszych iteracji produktu czy projektu, będę zwracał szczególną uwagę podczas następnej okazji.

2.4.5 Nikola Miszalska

Czytając książkę najbardziej zaskakujący wydał mi się jej format. W książce było zastosowanych wiele technik i metodyk na lepsze przyswajanie wiedzy. Po książce tego rodzaju spodziewałam się zdefiniowania metodyk, sposobu ich zastosowania oraz szczegółowego opisania. Taki sposób przedstawienia zagadnień na pewno byłby mniej atrakcyjny dla czytelnika. Dzięki technikom zastosowanym w książce łatwiej można było przyswoić wiedzę. Zdecydowanie do moich faworytów

należały ilustracje, bo wyrażały one więcej niż 1000 słów, czasami wystarczyło spojrzeć na ilustracje żeby zrozumieć całą ideę przedstawionej metody. Drugą ulubioną techniką było wypunktowanie kluczowych idei z rozdziału. Pomimo zasadności tych technik przedstawiania informacji pojawiły się również takie, które lekko zaciemniały odbiór. Dobrym przykładem były wskazówki dla coachów. Pomimo że na początku zagłębiałam się w ich tekst, to po pewnym czasie wydawały się one redundantne z punktami kluczowymi, czy bezpośrednio wynikały z przeczytanego tekstu w danym rozdziale. Odniosłam wrażenie, że książka ta jest przeznaczona dla osób rozpoczynających swoją przygodę z Agile, więc dodanie takiej techniki wydawało mi się trochę niepotrzebne i pomimo, że autor polecał czytanie ich również osobom nie trenującym innych, ja polecałbym ominąć je podczas czytania książki pierwszy raz.

2.4.6 Paweł Gelar

Czytając "Agile. Przewodnik po zwinnych metodykach programowania" czułem się w pewnym stopniu, jak gdy czytałem dialogi Platona (nie polubiłem tych utworów) – pomimo, że nazywa się to dialogiem jedna strona tylko przytakuje i wyraża podziw wobec drugiej, przez co utwór jest bardzo jednostronny. Tak samo ta książka ma nam przybliżyć wybrane metodyki programowania zwinnego pokazuje tylko ich pozytywne strony bez wspomnienia o negatywach - według autora jedynym negatywnym efektem próby wprowadzenia metodyk zwinnych do naszego zespołu jest brak zrozumienia ich przez programistów i innych pracowników, lub zarząd, przez co nie będą one w pełni, albo wcale skuteczne, jednak nic silne negatywnego się nie stanie. Jednak wydaje mi się, że same informacje, które zostały przekazane były zaprezentowane ciekawie i skutecznie – wiedza o zwinnych metodykach programowania raczej nie wyleci mi szybko z głowy. Chociaż większość diagramów wydawała mi się robiona raczej na siłę możliwe, że usprawniły one proces zapamiętywania. Mimo swoich problemów książka dostaje ode mnie +2 punkty za komiksy xkcd (chociaż nie było ich zbyt wiele). Niestety wydaje mi się trudno będzie mi wykorzystać w bliskiej przyszłości zdobytą dzięki tej książce wiedzę, gdyż projekty studenckie są tworzone nieregularnie, co uniemożliwia wykorzystanie konkretnych metodyk zwinnych (tygodniowe sprinty i codziennie spotkania, gdy pracuje się nad projektem 0-6 razy w tygodniu?)

2.5 Powiązana literatura

1. agilemanifesto.org - Manifest będący podstawą technik Agile
2. Agile Software Development: The Cooperative Game, 2nd Edition - Alistair Cockburn - Więcej na temat zasad i wartościach podejścia Agile
3. Agile Project Management with Scrum - Ken Schwaber - Źródło głębszej wiedzy na temat Scruma
4. Extreme Programming Explained: Embrace Change, 2nd Edition - Kent Beck, Cynthia Anders - Obszerne źródło wiedzy na temat programowania ekstremalnego
5. Succeeding with Agile - Mike Cohna - Więcej o wyzwaniach czekających na zespoły wprowadzające podejście zwinne