Wymagania w Agile

Krystian Kaczor

# Czas na Agile

No i stało się. Nadszedł ten dzień, kiedy szef poinformował Cię, że nadszedł czas na zmianę sposobu pracy na Agile. Jeśli miałeś wprowadzenie Agile robione metodą skoku na głęboką wodę, to możesz poczuć się jak szeregowiec Cage, główny bohater filmu z Edge of Tomorrow. Nie ma czasu na wyjaśnienia, weź nowe narzędzia i biegnij. Różnica jest taka, że Ty nie będziesz miał/ miała kolejnego podejścia w przypadku porażki projektu. Dla całego zespołu zmieni się część rzeczywistości dotycząca pracy, a może nawet zmieni się sam zespół. Zmiana będzie dotyczyła wielu aspektów pracy, na potrzeby tego artykułu musimy zawęzić zakres omawianych tematów. Proponuję, żebyśmy tym razem skupili się na temacie wymagań. Pracując z metodami Agile przez ostatnie 8 lat powiem Ci, że możemy również bezpiecznie założyć, że podejściem, które będzie Ciebie dotyczyło to framework Scrum. Zgodnie z koncepcją nauki sztuk walki, Shu-Ha-Ri najpierw potrzebujesz poznać podstawy.

Żeby dobrze omówić pracę z wymaganiami w Agile potrzeba przynajmniej jednego dnia szkoleniowego, wypełnionego ćwiczeniami praktycznymi. Ci, którzy byli na moich szkoleniach, wiedzą, o czym mówię. Poruszę tutaj najważniejsze zagadnienia i rozwieję kontrowersje, jakie narosły wokół tematu. Są narzędzia i koncepcje, które niewątpliwie w cyklu życia oprogramowania pomogą zespołowi utrzymać kierunek działań i sprawdzać słuszność podjętych decyzji. Jakie dokładnie? Dowiesz się w dalszej części artykułu.

# Co zmienia Agile?

Z punktu widzenia wymagań Agile wprowadza kilka znaczących zmian. Przyjrzyjmy się im po kolei.

Po pierwsze, wchodzi w życie to co jeży włosy na karku analityków biznesowych i testerów, czyli zasada pochodząca z Agile Manifesto, która mówi, że **zmiana wymagań jest mile widziana**. Oczami wyobraźni można zobaczyć jak klient ciągle zmienia zdanie i zespół nigdy nie jest w stanie dostarczyć dobrego produktu. Od razu mogę zapewnić, że tak nie będzie. Zmiany wymagań nie mogą być zupełnie nieograniczone i zachodzić w dowolnym momencie. Zespół pracuje w iteracjach określanych nazwą Sprint i wymagania przyjęte do realizacji na ten czas nie mogą zostać zastąpione innymi, bo ktoś zmienił zdanie. Dlaczego? Ponieważ taka zmiana powoduje, że szacowanie i planowanie wykonane na spotkaniu: "Planowanie Sprintu" można od razu wyrzucić do kosza. Tym samym wyrzucamy do kosza umowę między Właścicielem Produktu, a Zespołem na to, co miało być dostarczone na koniec Sprintu. Wymagania w trakcie pracy można doprecyzować, ale to nie oznacza , że można „odwracać kota ogonem”. Możliwa jest natomiast negocjacja zakresu, kiedy zespół lub klient dowiedzą się czegoś nowego. I tutaj zwykle pojawiają się pierwsze konflikty IT z biznesem. „No, ale jak to, przecież mieliśmy być Agile, a teraz nie można dowolnie zmieniać wymagań, planów i zasobów?” W tym momencie Scrum Master rusza na pomoc i edukuje biznes oraz chroni Zespół przed wpływami. Agile stawia na reagowanie na zmiany ponad negocjację kontraktów, więc trzeba umożliwić implementację informacji zwrotnej, którą otrzymamy przynajmniej na "Przeglądzie Sprintu". Metody zwinne są często wybierane przez organizacje, żeby lepiej reagować na potrzeby rynku i stać się bardziej konkurencyjnymi. Co to oznacza w praktyce? Możemy dobrze zaplanować okres od jednego do dwóch miesięcy i należy spodziewać się zmian w dalszych Sprintach. W niektórych organizacjach może być tak, że maksymalny stały horyzont planowania to będą maksymalnie 2 Sprinty tygodniowe. W odniesieniu do wymagań oznacza to, że szczegółowa elaboracja zakresu bardziej odległych Sprintów nie ma sensu i byłaby stratą czasu. Praca nad wymaganiami odbywa się tutaj zgodnie z zasadą "Just In Time" zaczerpniętą z Lean i następuje w etapach. W związku z tym Rejestr Produktu będzie przypominał swoim kształtem górę lodową. Na czubku góry będziemy mieli małe elementy (np. User Story) opracowane wystarczająco i gotowe do implementacji w kolejnych 1-2 Sprintach. Elementy zaplanowane na kolejne iteracje będą tym większe i bardziej ogólne, im ich implementacja jest oddalona w czasie. Rejestr Produktu ciągle się zmienia i niektóre elementy mogą nigdy nie zostać zaimplementowane, a inne zupełnie się zmienić.

Skąd wiadomo, czy User Story jest opracowana wystarczająco, żeby ją zaimplementować w Sprincie? Właściciel Produktu i Zespół Developerski wspólnie uzgadniają ten standard i określają go jako Definicje Gotowości. Powinny się znaleźć tutaj takie elementy jak: zgodność z szablonem, Warunki Satysfakcji, kroki demonstracji, makieta ekranu, szkic przepływu danych itp. Jednak w żadnym wypadku "User Story" nie powinna narzucać rozwiązania i zawierać szczegółów technicznych.

Zdecydowanie mniej negatywnych myśli przywodzi na myśl fakt **zbliżenia klienta lub przedstawiciela biznesu do zespołu**. Stały i bezpośredni kontakt z biznesem to elementy, które na pewno wpływają korzystnie na jakość wymagań. Klient, czy użytkownik końcowy to źródło informacji zwrotnej, dzięki której zespół buduje to co jest potrzebne i ma wartość.

Jak już wcześniej wspomniałem, **nie ma czasu ani potrzeby na dogłębną analizę wymagań i projektowanie z góry**. Opracowujemy tyle ile potrzeba i kiedy potrzeba. Pojawiają się zatem pytania o to, skąd wiemy jaki poziom szczegółowości jest potrzebny i jak planować dalszą przyszłość. Odpowiem na te pytania w dalszej części artykułu.

# Nowa forma wymagań

Scrum Guide nie mówi nam za dużo na temat wymagań. Właściwie mówi bardzo niewiele. Rejestr Produktu (ang. *Product Backlog*) jest jedynym źródłem wymagań i zawiera uporządkowaną listę wszystkiego, co jest potrzebne w produkcie oraz wszystkich zmian wymaganych w tym produkcie. Lista nigdy nie jest kompletna, i będzie się zmieniać. Rejestr Produktu jest złożony z wszystkich funkcjonalności, zadań i poprawek jakie trzeba wykonać dla Produktu, który buduje zespół. Każda z tych rzeczy nosi nazwę: Elementu Rejestru Produktu (ang. *Product Backlog Item*) i posiada przynajmniej minimum takie właściwości jak: wartość, oszacowanie, opis. Koniec i kropka.

Przyjęło się, że w Scrum korzystamy z narzędzia wprowadzonego w Programowaniu Ekstremalnym, którym jest User Story. User Story ma na celu skupienie uwagi na realnym użytkowniku systemu i rozpoczęcie dyskusji na temat, co jest potrzebne i co można zrobić, że zaspokoić potrzebę tego użytkownika lub rozwiązać jego problem. Tylko tyle i aż tyle.

Ciekawostką samą w sobie jest to, że tak naprawdę to nie sama User Story tworzy wymaganie. User Story jest pretekstem do rozpoczęcia rozmowy na temat wymagania i doprecyzowania wymagania poprzez testy akceptacyjne, które potwierdzą właściwą implementację. Ten koncept został sformułowany przez Rona Jeffrie jako 3C.

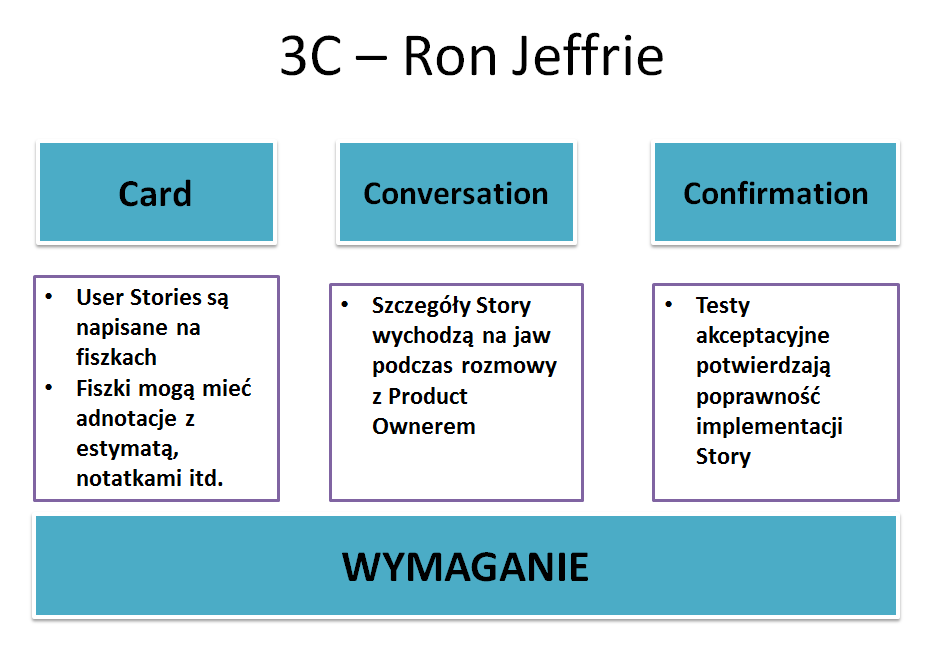


Figure 1 Koncept 3C autorstwa Ron'a Jeffrie

Słyszałem o przypadkach, gdzie przejście z tradycyjnych wymagań na User Story polegało na komendzie wyszukaj „System powinien ” i zamień na „Jako użytkownik, chcę ”. Język polski ze swoją gramatyką nie ułatwia takiej migracji i dobrze, bo nie o to chodzi. Scrum Guide wyraźnie mówi, że Rejestr Produktu jest jedynym źródłem wymagań, więc niedopuszczalne jest umieszczanie linków do wymagań w Sharepoint, czy innym systemie. Nikt nie będzie skakał w tą i z powrotem, żeby upewnić się, że rozumie wymaganie i niczego nie pominął. To zwykła strata, która powinna być wyeliminowana zgodnie z filozofią Lean.

Są przynajmniej dwie popularne metody pisania User Story. Jedna z nich jest bardziej znana - jest to forma często określana, jako Podstawowa struktura User Story, przedstawiona w ramce.

**Podstawowa struktura User Story**

***Jako*** <konkretny użytkownik, Persona, rola w systemie>, ***chcę*** <potrzeba>, ***żeby*** <problem do rozwiązania, cel do osiągnięcia>.

***Warunki Satysfakcji***

Równie ciekawą i mniej znaną alternatywą jest stosowanie struktury pochodzącej z **Five Ws**[[1]](#endnote-1), czyli Who, When, What, Where, Why, która jest stosowna na przykład w dziennikarstwie. Po polsku będzie to Kto, Co, Kiedy, Gdzie, Dlaczego. Zobacz w ramce jak wygląda struktura takiej User Story.

**Struktura User Story z Five Ws**

***Jako*** <kto> <kiedy> <gdzie>, ***chcę*** <co>, ***ponieważ*** <dlaczego>.

***Warunki Satysfakcji***

W dalszej części artykułu będziemy skupiali się na pierwszej, bardziej popularnej formie. Zauważ, że nieodzownym elementami User Story są odpowiedzi na pytania: Kto potrzebuje funkcjonalności?, Co to za funkcjonalność? i Dlaczego jest potrzebna?

Przyjrzyjmy się, co dokładnie wpisujemy w poszczególne elementy

Jako

* Persona;
* Konkretny użytkownik;
* Ewentualnie rola w systemie;

Chcę

* Potrzeba;
* Problem;
* Otwarte pytanie;
* Wyraź znaczenie dla klienta;
* Nie może to być rozwiązanie np.: chcę pole tekstowe;

Ponieważ/Żeby

* Powód;
* Wartość dla klienta;
* Często reprezentuje prawdziwy problem;

Porównajmy teraz wymagania w formie IEEE 830 i User Story, żeby zobaczyć efekt i korzyści płynące z używania tej struktury.

Zapraszam Cię do ćwiczenia, które polega na określeniu, jaki produkt Zespół ma zbudować w kolejnej iteracji na podstawie wymagań. Po przeczytaniu każdego zdania, zastanów się, co to jest.

* Produkt powinien mieć silnik benzynowy dużej mocy.
* Produkt powinien mieć 4 koła.
* Produkt powinien mieć kierownicę.
* Produkt powinien mieć stalową karoserię.
* Produkt powinien być czerwony.

Niech zgadnę, to może być Ferrari, furgonetka dostarczająca mleko, samochód opancerzony do przewozu kosztowności, quad i kilka innych rzeczy.

Sprawdźmy jak szybko określisz, co Zespół ma zbudować czytając poniższe User Story.

*Jako właściciel działki rekreacyjnej, chciałbym móc kosić trawę szybko i łatwo, żeby nie tracić czasu na pracę podczas wypoczynku.*

*Jako właściciel działki rekreacyjnej, chciałbym siedzieć w wygodnej pozycji kosząc trawę, żeby nie mieć bólu pleców wieczorem.*

Szybko domyśliłeś się, że to jest kosiarka do trawy, prawda? I to jest jedna z zalet User Story. Od razu wiadomo, kto potrzebuje tej funkcjonalności, jaki ma problem lub potrzebę i jak sobie wyobraża rozwiązanie. Może się okazać, że jest wiele sposobów rozwiązania tego problemu lub, że istnieje już sposób zaspokojenia tej potrzeby. Ten sposób może nie być idealny, ale wystarczający na tą fazę rozwoju produktu.

Co się stanie, jeśli pominiemy niektóre elementy? Sprawdźmy.

*Jako użytkownik chcę ponownie zarezerwować lot.*

Czego tutaj brakuje? Brakuje problemu do rozwiązania lub celu do osiągnięcia i konkretnego użytkownika. Może to być użytkownik, który lata często na tej samej trasie i chce, żeby system mu podpowiadał ostatnio wybrane opcje, może to być użytkownik, który ma kilka ulubionych miejsc docelowych i chciałby wybierać je z listy, może to być użytkownik korporacyjny, który chce dodać kolejne osoby do podróży służbowej. Zauważ jak będą różniły się implementacje techniczne ze względu na faktyczną potrzebę użytkownika.

*Jako Babcia Zosia\* chcę ponownie zarezerwować ostatnio wybrany lot, żeby uniknąć wypełniania całego formularza wyszukiwania i przechodzenia przez wszystkie kroki rezerwacji.*

\*Babcia Zosia to osoba latająca zawsze na tej samej trasie, odwiedzająca wnuki w Anglii.

*Jako kierowca ciężarówki chcę widzieć zaplanowaną trasę.*

Czego tutaj brakuje? Mamy wprawdzie kierowcę ciężarówki, ale nie wiemy, po co mu ta mapa trasy, czyli brakuje potrzeby lub problemu. I znowu możemy się domyślać, że kierowca chce zobaczyć czas przejazdu pomiędzy punktami trasy. A może chce zobaczyć natężenie ruchu na trasie? Może chce zobaczyć samą listę punktów, gdzie dostarcza towar? A może chce zobaczyć spodziewany czas przejazdu i aktualny, żeby wiedzieć czy wyrabia plan? A może chce …? Jednak każde takie domyślanie się to założenie, które może się zemścić na Zespole.

*Jako kierowca ciężarówki chcę widzieć planowany czas dojazdu do kolejnego punktu na trasie i numer telefonu kolejnego klienta, żeby móc skontaktować się z klientem zanim dojadę.*

Z kilku względów dobrze jest pisać i sortować User Story na tak zwanych fiszkach (ang. index cards). Rozmiar fiszki ogranicza ilość informacji, jaką da się zapisać, a zatem ogranicza rozmiar User Story. Fiszki mogą być przekazywane z ręki do ręki i każdy może dopisać kolejne Warunki Satysfakcji. To bardziej angażuje grupę niż obserwowanie jednej osoby wpisującej tekst, który jest wyświetlany przez projektor. Wszelkie sortowanie i grupowanie takich fiszek jest też dużo łatwiejsze i bardziej interaktywne. Na koniec można wprowadzić zmiany do systemu przechowywania naszego Product Backlog, na przykład Atlassian Jira. Spójrz na przykładową User Story.

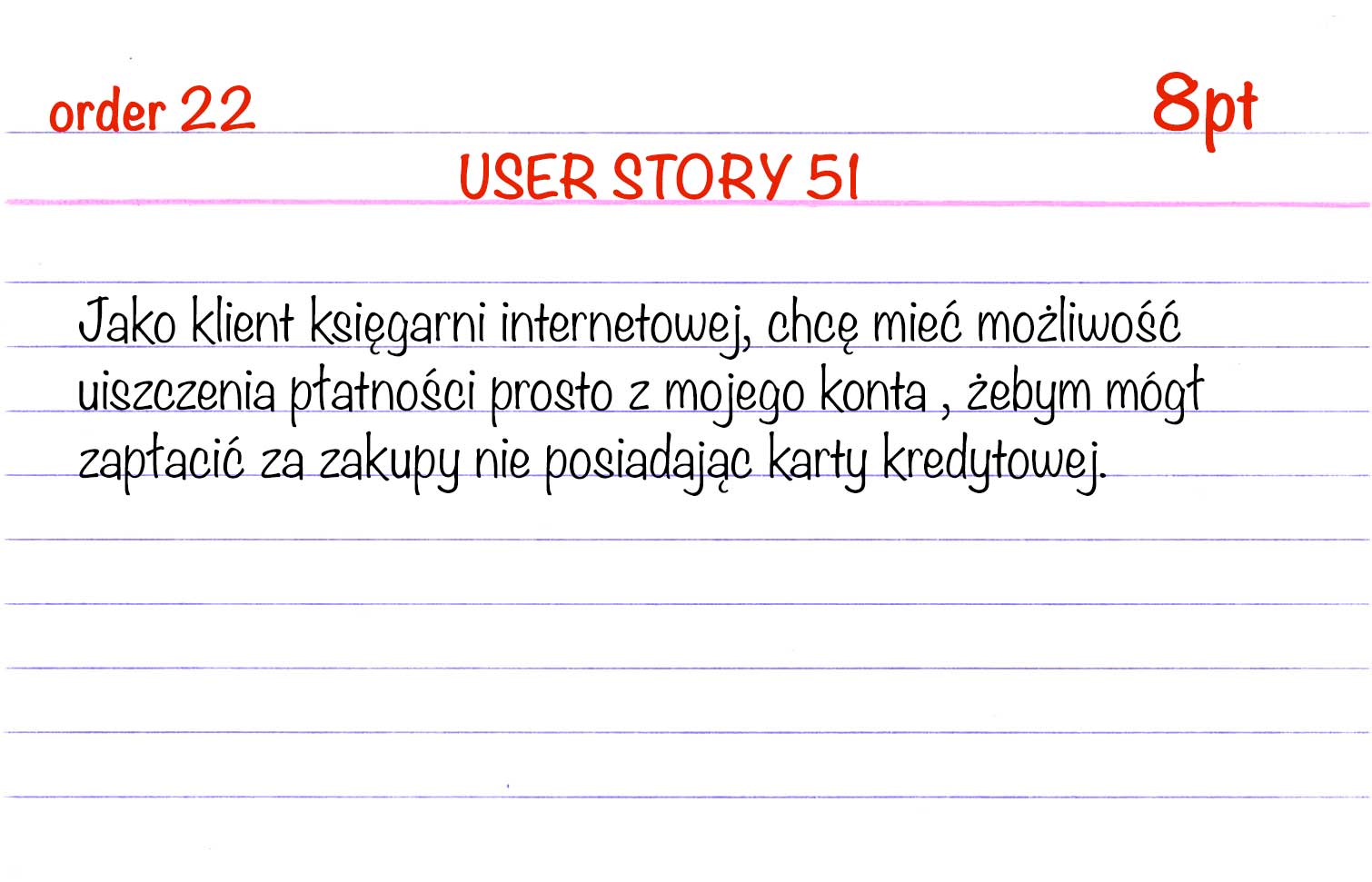


Figure 2 User Story na fiszce

Jakie element są wyróżnione? Mamy tutaj wszystkie elementy, które wymaga od nas Scrum, żeby taką Story uznać za pełnoprawne Product Backlog Item, czyli Opis, Porządek na liście (Order 22) i Oszacowany rozmiar (5 punktów) i dodatkowo podane jest Id User Story używane w systemie przechowywania Backlog’u (USER-STORY-51).

W formie User Story można także przedstawiać wymagania zupełnie techniczne takie jak konieczność zainstalowania oprogramowania, aktualizacji wersji oprogramowania, czy implementację interfejsu.

*Jako system weryfikacji płatności chcę otrzymywać wszystkie transakcje w XML, żeby móc przekazać środki z karty klienta na konto właściciela sklepu.*

*Jako tester, chcę API dla bazy danych, żebym mógł łatwo usuwać całe transakcje, kiedy testuję system.*

*Jako użytkownik systemu chcę aktualizacji biblioteki jQuery do najnowszej wersji, żeby móc bez przeszkód korzystać z systemu w nowej wersji przeglądarki Firefox.*

Teraz przejdźmy do drugiej strony naszej fiszki, czyli spójrzmy na Warunki Satysfakcji (ang. Acceptance Criteria), lub jak kto woli Warunki Akceptacji. Tutaj zaczyna się precyzowanie ustaleń i określanie działania systemu. Testy Akceptacyjne zapisane jako Warunki Satysfakcji mają za zadanie potwierdzenie, że Story została zaimplementowana tak, jak spodziewał się tego klient. Kiedy zacznie się implementacja, programiści zaczną pisać kod spełniający te warunki, a testerzy zaczną pisać Przypadki Testowe rozszerzające te warunki oraz wynikające z metod testowania, np. warunki brzegowe, tabele decyzyjne i tak dalej. Warunki Satysfakcji są wysokopoziomowymi testami akceptacyjnymi i nie zastępują pełnego testowania funkcjonalności. Jak dobierać metody i priorytety testowania? Pełna odpowiedź na to pytanie wykracza poza ramy omawianych tutaj zagadnień. Teraz powiem Ci krótko: „To zależy od ryzyka”. Acceptance Criteria mogą być zapisane w postaci pytań takich, jak „Czy mogę zabrać kartę z bankomatu po trzech nieudanych próbach?”, notkach o tym co przetestować „Przetestuj niepoprawny numer CCV karty”, lub w postaci odpowiedzi na zadane pytania „Karta jest blokowana po trzech nieudanych próbach i zostaje w bankomacie”. Zobacz poniżej jak to może wyglądać dla naszej przykładowej User Story.

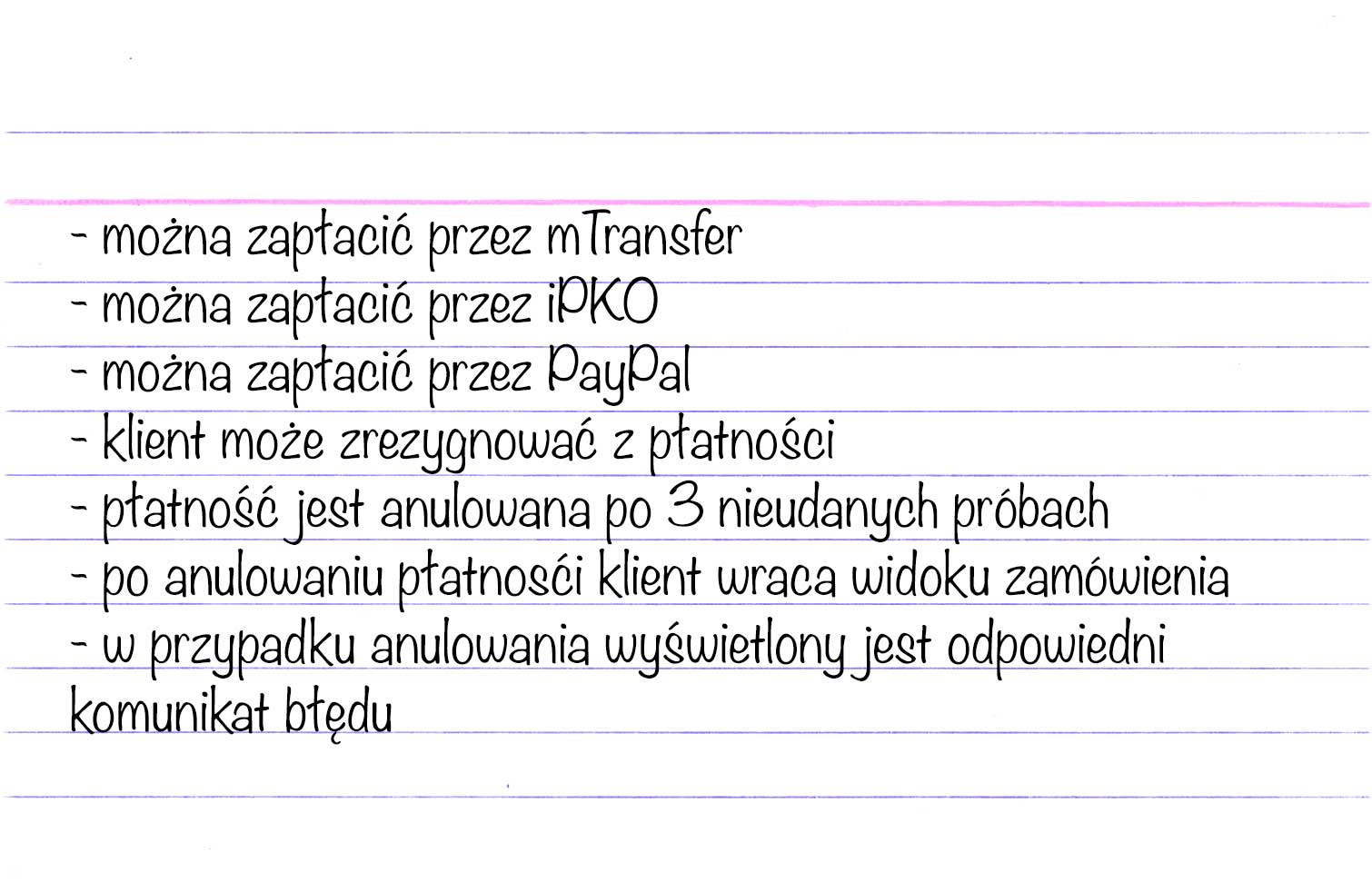


Figure 3 Warunki Satysfakcji dla User Story na fiszce

# Nowe role

Żeby promować samoorganizację i stworzyć wspólne poczucie odpowiedzialności za wyniki pracy, w Scrumie mamy trzy role: Scrum Master, Właściciel Produktu (ang. Product Owner) i Developer. Jak to wpływa na wymagania i kto jest za nie odpowiedzialny?

**Scrum Master** jest odpowiedzialny za procesy związane ze Scrum i przestrzeganie zasad frameworku. **Developer** jest odpowiedzialny za współpracę w Zespole, organizację swojej pracy i dostarczenie działającego oprogramowania na koniec Sprintu. **Właściciel Produktu** jest odpowiedzialny za dostarczanie wymagań i zapewnienia ich dobrego zrozumienia w zespole. Jako przedstawiciel biznesu powinien również powiedzieć Zespołowi Deweloperskiemu jakie są priorytety kolejnych User Story z punktu widzenia ich wartości dla biznesu i realizowania celów organizacji. Z kolei Zespół jest odpowiedziany za szacowanie wielkości User Story. Dlaczego? Ponieważ jako osoby, które będą wykonują pracę wiedzą ile ona zajmie i które elementy są skomplikowane do dostarczenia, a które proste.

Hej, ale nie widzę tutaj roli Analityka Biznesowego, co się z nim stało? Deweloper to każda osoba, która wykonuje pracę w celu dostarczenia potencjalnie gotowego do wydania kawałka wartościowego oprogramowania. Zatem możliwe jest, że Analityk to jeden z Developerów. Tak samo Tester będzie określany jako Developer, ale będzie wykonywał pracę w swojej dziedzinie specjalizacji. Analityk z uwagi na obszar specjalizacji może zostać także Właścicielem Produktu. Takie podejście ma sens, jeżeli to on lub ona faktycznie podejmuje decyzje co do określenia wymagań. Jeżeli nie jest to osoba z wystarczającą władzą w organizacji i musi pytać o zdanie i decyzje inną osobę, to taki Właściciel Produktu jest jedynie proxy pomiędzy prawdziwym PO, a Zespołem Deweloperskim. Czy jest to efektywne rozwiązanie? Najczęściej nie, bo tworzy tylko zbędny bufor. Analityk jako proxy sprawdzi się dobrze jako Właściciel Produktu, jeżeli Zespół Scrum pracuje z zewnętrznym klientem.

No dobrze, skoro już otworzyliśmy ten temat, to odpowiedzmy sobie na pytanie, jakie cechy posiada **dobry Właściciel Produktu**.

Szukamy osoby, która spełnia następujące kryteria:

* zna domenę biznesu,
* ma jasną Wizję Produktu,
* potrafi zarządzać zaangażowaniem interesariuszy,
* ma kontakt z końcowymi użytkownikami system,
* nie boi się podejmowania decyzji,
* ma określone zdanie i nie zmienia go w trakcie Sprintu,
* ma autorytet do samodzielnego podejmowania decyzji związanych z Produktem,

Pojawia się też pytanie, kto pisze User Story? Jest to odpowiedzialność Właściciela Produktu, ale Zespołu może go w tym wspierać, a Scrum Master uczyć jak wypełniać obowiązki. Omawianie, szacowanie i dzielenie User Story wspiera negocjację wymagań.

# Kiedy opracowywać wymagania i do jakiego poziomu?

Wymagania w świecie metod zwinnych mają pewną hierarchię. Na samej górze znajduje się Wizja Produktu, która powinna być zakomunikowana i jasna dla Zespołów tworzących Produkt. Na tej postawie będą określane tematy do zrealizowania i cele kolejnych wydań. Następnie tematy będą podzielone na Epiki lub od razu na User Story. Każdy Sprint powinien realizować Cel Sprintu zgodny z Celem Wydania i zawierać User Story, których dostarczenie jest niezbędne.

Tematy i Epiki to elementy grupujące Rejestru Produktu, które pomagają określać priorytety i wartość biznesową funkcjonalności.

Epik to bardzo duża Story, która wymaga dopracowania i trudno jest prawidłowo oszacować jej wielkość. Prawdopodobnie w trakcie Pielęgnacji Rejestru zostanie rozbity na kilka User Story. Epik zwykle reprezentuje funkcjonalność, którą trzeba zaimplementować.

Temat to grupa User Story podobna pod pewnym względem. Możemy grupować User Story ze względu na użytkownika, którego dotyczą, obszar aplikacji itp.

W Scrum mamy trzy oficjalne momenty, kiedy pracujemy nad Rejestrem Produktu, ale to nie oznacza, że nie ma pracy wykonywanej pomiędzy. Tak więc mamy Pielęgnację Rejestru Produktu, która zajmuje do 10% czasu Zespołu przeznaczonego na pracę w Sprincie. Jest to czynność, która musi zajść i może być dowolnie zaplanowana w trakcie Sprintu. Celem pielęgnacji jest opracowywanie kolejnych elementów Rejestru Produktu tak, żeby mieć gotowe User Story w ilości od 1 do 2 Sprintów. Taki proces jest potrzebny przede wszystkim, żeby Zespół nie musiał czekać na wsad do nowych Sprintów oraz, żeby nie próbować robić tego na Planowaniu Sprintu, bo takie postępowanie niepotrzebnie wydłuży spotkanie. W trakcie pielęgnacji Zespół Developerski i Właściciel Produktu szacują, doprecyzowują i dzielą User Story.

Planowanie Sprintu, to moment kiedy Zespół Developerski i Właściciel Produktu wybierają i negocjują zakres User Story, które zobowiązują się dostarczyć w Sprincie. Bierzemy pod uwagę prędkość Zespołu określaną jako Velocity oraz dostępność członków zespołu w tym konkretnym Sprincie. Dla jasności dodam, że Zespół Developerski jest w 100% zaangażowany w prace nad jednym Produktem lub ewentualnie projektem, a mówiąc o dostępności mam na myśli święta państwowe, zwolnienia lekarskie i urlopy.

W trakcie Sprintu może nadal dochodzić do renegocjacji elementów w wypadku, gdy Zespół Deweloperski trafił na przeszkody lub dowiedział się nowych faktów o wykonywanej pracy i wie, że nie zdąży dostarczyć wszystkich User Story czy wręcz odwrotnie, ma pojemność na dostarczenie większej ilości, niż zaplanował. W tym drugim przypadku pomaga nam to, że wcześniej przygotowaliśmy pulę User Story w ilości 1-2 Sprintów i teraz z łatwością możemy dać Zespołowi kolejną z góry Rejestru.

Ostatnim momentem, gdzie formalnie zajmujemy się Rejestrem Produktu jest Przegląd Sprintu. Przejrzenie tego, co Zespół dostarczył, zmiany w organizacji i demo rozwiązania mogą wpłynąć na zmianę priorytetów lub zmianę w elementach rejestru.

Równolegle, przez cały czas Właściciel Produktu powinien pracować z interesariuszami i nieustannie opracowywać Rejestr Produktu, żeby zawsze prezentował on aktualny stan oraz reprezentował realizację celów organizacji.

# Podsumowanie

Z punktu widzenia osoby pracującej w sztywnej strukturze korporacyjnej i przyzwyczajonej do papierkowej roboty, praca w metodach miękkich może wydawać się mało poważna i ryzykowna. Jednak przy bliższym poznaniu Agile zyskuje ze względu na łatwość reakcji na zmiany, utrzymywalne tempo pracy i bliski kontakt z klientem. Jak widać nadal potrzebne będą procesy i jasne zasady wykonywania pracy. W nowej rzeczywistości, w jakiej obecnie znajdują się organizacje, w których pracujemy, potrzebne są metody, które wykorzystują potencjał profesjonalnych pracowników i pozwalają na łatwą reakcję na zmiany. Pisanie szczegółowych wymagań przez pojedyncze osoby i próba zgadywania jak ma wyglądać efekt końcowy zawsze przegra w konkurencji z samoorganizującymi się zespołami i dostosowywaniem się do informacji zwrotnej po zaprezentowaniu kolejnej wersji działającego oprogramowania.

Osoby zainteresowane dokładnym poznaniem metod pracy z wymaganiami odsyłam do mojej książki „Scrum i nie tylko. Teoria i praktyka w metodach Agile”.

# Bibliografia

Krystian Kaczor, *Scrum i nie tylko. Teoria i praktyka w metodach Agile*, PWN, 2014

Dean Leffingwell, *Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise (Agile Software Development Series)*, Upper Saddle River, NJ, Addison-Wesley 2011

Scrum Guide wersja Lipiec 2013, Ken Schwaber, Jeff Sutherland, http://www.scrumguides.org

M. Cohn, *User Stories Applied: For Agile Software Development*, Upper Saddle River, NJ, Addison-Wesley 2004.

1. http://en.wikipedia.org/wiki/Five\_Ws [↑](#endnote-ref-1)