Ida Knapik[1], **Jan Werewka**[2]

Synergia zwinnego i szczupłego rozwoju oprogramowania na bazie Scrumban

Abstract: IT companies are constantly looking for effective methods of software development. Typically, these approaches are tailored to the type of software development work. The most popular are the agile, lean and proactive (classic) approaches. The study will assess the methodology that synergistically combines the agile and lean approaches. Representatives of these approaches are Scrum and Kanban. As a result of this assessment, optimal solutions for software development teams will be proposed. A case study of using a solution combining agile and lean approaches will be presented.

Streszczenie: W firmach informatycznych cały czas poszukiwane są efektywne metody rozwoju oprogramowania. Zwykle te podejścia dostosowywane są do rodzaju prac związanych z rozwojem oprogramowania. Do najbardziej popularnych należą podejście zwinne (agile), szczupłe (lean) i proaktywne (klasyczne). W opracowaniu poddana zostanie ocenie metodyka łącząca w sposób synergetyczny podejście zwinne (agile) i podejście szczupłe (lean). Reprezentantami tych podejść są Scrum i Kanban. W wyniku tej oceny zostaną zaproponowane optymalne rozwiązania dla zespołów rozwijających oprogramowanie. Przedstawiony zostanie przykład studialny (case study) użycia rozwiązania łączącego podejście zwinne i szczupłe.

Słowa kluczowe: Scrumban, Scrum, Kanban, zarządzanie projektami, wytwarzanie oprogramowania

Keywords: Scrumban, Scrum, Kanban, project management, software development

JEL classification codes: A2, C8, I2, Q5

[1] Akademia Górniczo-Hutnicza, AGH, Wydział Humanistyczny, Informatyka Społeczna.
[2] Dr hab. inż. Prof. Wyższej Szkoły Ekonomii i Informatyki w Krakowie, Zakład Informatyki.

Wstęp

W firmach informatycznych występuje silne dążenie do dostarczania klientom wartościowego oprogramowania w krótkich okresach czasowych. W tym celu rozwinięte zostały metodyki rozwoju oprogramowania oraz zarządzania produktem i projektem. Ogólnie te rozwiązania można podzielić na podejście zwinne (agile) zoptymalizowane ze względu na zmieniające się potrzeby klienta, szczupłe (lean) skoncentrowane na optymalizacji procesów produkcyjnych (procesów wytwarzania oprogramowania), proaktywne (nazwane także klasyczne lub tradycyjne) ukierunkowane na uwzględnienie i rozwiązywanie problemów mogących się pojawić w trakcie realizacji projektów. Do każdego podejścia opracowane zostały konkretne metodyki. Dla podejścia zwinnego najbardziej znanymi rozwiązaniami w obszarze rozwoju oprogramowania są Scrum oraz XP (Extremme programming). Natomiast podejście szczupłe reprezentowane jest przez metodyki Kanban, LSD (Lean Software Development) oraz szereg rozwiązań optymalizujących procesy wytwarzania oprogramowania takie jak ciągła integracja (CI – Continous Integration), ciągła dostawa (CD – Continous Delivery), rozwój oprogramowania bazujący na testach (TDD – Test Driven Development), kontenery, mikrousługi itd. Spośród metodyk klasycznych najbardziej znane są PMBOK i Prince 2.

Dodatkowo pojawiają się metodyki zwinne dostosowywane do wielkości firmy [1]. Przykładami takich rozwiązań są:

1. Nexus, opracowany w 2015 przez Kena Schwaber, w którym punktem wyjścia jest metodyka Scrum. Celem tego rozwiązania jest skalowanie w górę do 9 zespołów. Główną różnicę między Nexusem a Scrumem stanowi dodanie zespołu integracyjnego, który koncentruje się na rozwiązywaniu zależności i kwestii integracji między zespołami.
2. Large Scale Scrum (LeSS), opracowany przez Craiga Larman i Basa Vodde. Large Scale Scrum (LeSS) kładzie nacisk na proces skalowania przy jednoczesnym zachowaniu oryginalnej filozofii Scruma. Celem jest przyjęcie zasad Scruma jednego zespołu i dostosowanie ich do pracy w większych projektach.
3. Disciplined Agile 2.0 (wcześniej DAD) rozwijany przez Scotta Ambler i Marka Lines, to hybrydowe podejście Lean-Agile, łączące różne zasady Agile, w tym strategie Extreme Programming (XP), Unified Process (UP), Kanban, Lean, Outside In i Agile Modeling. Scott Ambler opracował Disciplined Agile Delivery podczas pracy w IBM Rational w latach

2006-2012, starając się zapewnić ramy na poziomie przedsiębiorstwa do dostarczania rozwiązań IT.

4. Scaled Agile Framework (SAFe) opracowany przez Deana Leffingwell. Podejście SAFe ma na celu pomóc dużym przedsiębiorstwom w dokonywaniu znaczących zmian organizacyjnych na dużą skalę poprzez dostarczanie dostosowywalnych, skalowalnych i elastycznych rozwiązań.

Oprócz przedstawionych metodyk pojawiają się rozwiązania hybrydowe. Najbardziej znaną hybrydą jest Scrum/XP będącą połączeniem Scruma i programowania ekstremalnego (XP). W pracy rozpatrywane jest rozwiązanie Scrumban, będące próbą wykorzystania zalet metodyki Scrum i Kanban.

Scrumban, czyli synergetyczne wykorzystanie podejść Scrum i Kanban

Nie ma jednego sposobu na wdrożenie Scrumban-a [2], zwykle podaje się kilka ścieżek dochodzenia do wdrożenia tej metodyki:

1. Zespół stosuje Scrum, ale uznajemy, że warto dodatkowo optymalizować proces rozwoju oprogramowania i zastosować elementy Kanbana. W takim przypadku możemy traktować, że Kanban jest swoistym rozszerzeniem Scruma.
2. Zespół stosuje Scrum, ale pewne elementy Scruma wydają się zbyt usztywniające proces wytwarzania oprogramowania. Przykładowo użycie sprintów (iteracji) wydaje się mało potrzebne, gdy głównie utrzymujemy oprogramowanie, a klient jest zainteresowany oprogramowaniem, w którym pojedyncze funkcjonalności są szybko wdrażane. W tym przypadku zespół przechodzi na stosowanie Kanbana rezygnując z pewnych właściwości Scruma.
3. Zespół stosuje podejście Kanban, lecz widzi potrzebę pracy zespołowej potrzebną do realizacji wydań oprogramowania.
4. Pozostałe podejścia w tym przejście bezpośrednio do pracy w Scrumban.

Istnieje wiele prób przedstawienia różnic i podobieństw Scruma, Kanbana i Scrumbana, przykładowo są to witryny internetowe [3] i [4].

Trochę inne spojrzenie na Scrumban daje nam praktyczne wdrożenie tej metodyki w procesach wytwarzania oprogramowania w firmach informatycznych. Praktyczne wdrożenie stawia nas przed konkretnymi wyzwaniami i szukaniem optymalnych rozwiązań [5]. To wdrożenie pozwoliło na zadanie sobie pytania czym jest właściwie Scrumban i jak opisać tę metodykę we właściwy sposób.

Budowa modelu Scrumban w języku ArchiMate stosowanym do modelowania architektury korporacyjnej

Metodyki zarządzania projektami można opisać na wiele sposobów. Jednym z bardziej naturalnych podejść jest opracowanie modelu metodyki w języku opisu architektury korporacyjnej. Języki ArchiMate lub BPMN mogą posłużyć do modelowania metodyk zarządzania projektami. W [6] przedstawiono model metodyki PMBOK zapisany w Archimate, podobnie model PMBOK-a starszej wersji opisano w BPMN [7]. Podobnie można zamodelować metodyki zwinne w ArchiMate jak to przedstawiono w [8] i [9].

Metodyki PMBOK i Scrum zostały dosyć dobrze opisane, dlatego stosunkowo łatwe było zbudowanie odpowiednich modeli. Dla metodyki Scrumban sytuacja jest dosyć nietypowa, gdyż przy jej opisie pojawiają się zdania, że pewne działania: „dokonywane są w zależności od potrzeb", "zależą od zdefiniowanych ról i potrzeb", „zazwyczaj są", „możliwe są" i itp. Świadczy to o rozmytości metodyki i przyjmowania rozwiązań w zależności od sytuacji. W związku z tym należy określić, które rozwiązania są zafiksowane w metodyce, a które mogą, lecz nie muszą być w niej zawarte, równocześnie określając zalety i wady takiego zawierania.

Taki sposób podejścia budzi skojarzenia z teorią zbiorów rozmytych (ang. fuzzy set) wprowadzoną przez Lotfiego A. Zadeh. Podejście zbiorów rozmytych zastosowano przy ocenie narzędzi architektury korporacyjnej inteligentnych miast [10]. Podobne zagadnienia były rozpatrywane przy empirycznym modelowaniu architektury korporacyjnej przy użyciu ArchiMate [11] oraz przy rozpatrywaniu komponentowej złożoności architektury korporacyjnej przy zastosowaniu języka Archimate [12].

W pracy proponowane jest użycie konceptów warstwy motywacji języka Archimate [13], które pozwolą na doprecyzowanie metodyki Scrumban. Podstawą do budowy modelu jest określenie założeń podejścia Scrumban (Tab. 1) przy uwzględnieniu relacji tego podejścia z metodykami Scrum i Kanban.

Tabela 1. Wybrane relacje pomiędzy metodykami Scrum, Kanban i Scrumban.

Scrum	Kanban	Scrumban
Zapis wymagań		
Wymagania zapisywane w rejestrze produktowym (Product Backlog) głównie w postaci historyjek, epików i tematów. Wymagania najbardziej pilne znajdują się na początku rejestru. Wymagania te przepisywane są w postaci pewnej grupy i przekształcane w postaci zadań do rejestru sprintu.	Wymagania zapisywane w rejestrze prac (Backlog) w postaci elementów pracy. Prace są szeregowane względem pilności wykonania. Prace pobierane są według kolejności w rejestrze.	Wymagania zapisywane w rejestrze prac (Backlog) w postaci elementów pracy. Prace są szeregowane względem pilności wykonania. Prace pobierane są według kolejności w rejestrze lub ewentualnie w postaci grupy.
Planowanie		
Na początku każdego sprintu (iteracji rozwoju oprogramowania trwającej od 1 tygodnia do miesiąca).	Brak precyzyjnego planowania. Planowanie możliwe po zakończeniu każdego elementu pracy. Ciągły przepływ pracy.	Planowanie możliwe po zakończeniu każdego elementu pracy. Możliwe jest użycie iteracji.
Estymacja		
Przed rozpoczęciem każdego sprintu. Elementy pracy powinny być małe by mieściły się w sprincie.	Opcjonalnie po zakończeniu pracy. Zespoły stosują metodę pobierania (pull) wymagań. Ograniczenie liczby prac w toku.	Estymacja dokonywana w razie potrzeby. Prace są pobierane do wykonania z rejestru według kolejności.
Zmiana zakresu pracy		
Tylko przed następnym sprintem.	Dokonywana w zależności od potrzeb.	Dokonywana w zależności od potrzeb.
Iteracje		
Iteracje nazywane sprintami o stałym okresie trwania. Zwykle trwające 1-4 tygodni.	Ciągła praca z krótkimi wydaniami.	Ciągła praca z krótkimi cyklami planowania i dłuższymi cyklami wydań (buckets).
Podstawowe role		
Zespół deweloperów, Scrum Master, właściciel produktu.	Definiowane w zależności od potrzeb.	Definiowane w zależności od potrzeb mogą być użyte role: zespół deweloperów, Scrum Master, właściciel produktu.
Spotkania		
Planowanie, wykonanie, przegląd i retrospektywa sprintu. Dzienne spotkania.	Nie są wymagane.	Dzienne spotkanie (Daily Standup). Mogą być użyte spotkania Scrum.

Własność (ownership)		
Właściciel produktu.	Zależy od zdefiniowanych ról i potrzeb.	Zależy od zdefiniowanych ról i potrzeb. Zazwyczaj jest to właściciel produktu.
Tablice (promienniki informacji, kontrola, monitorowanie)		
Rejestr produktowy, rejestr sprintu, tablica scrumowa, wykres wypalania.	Tablica kanbanowa, czas cyklu i przetwarzania, diagramy kumulacyjne.	Tablica procesowa, czas cyklu i przetwarzania, diagramy kumulacyjne.
Kiedy stosować		
Małe wymagania, zbyt mała wartość. Przyrosty oprogramowania możliwe. Możliwe kształtowanie wymagań. Plan produktu jest znany. Zespoły multidyscyplinarne.	Zbyt szybkie zmiany. Potrzeba uwzględnienia prac utrzymania i pielęgnacji oprogramowania.	Rozwijające się wymagania, niejasny plan rozwoju produktu, zbyt szybkie zmiany. Potrzeba uwzględnienia prac utrzymania i pielęgnacji oprogramowania.

Źródło: opracowanie własne.

Dodatkowo warto przeanalizować planowanie długoterminowe. W Scrumbanie opiera się ono w oparciu o metodę trzech wiader (tzw. bucket size planning). Opiera się na systemie „wiader", każde związane z różnym horyzontem planowania. Mamy wiadra z okresami planowania 1 roku, 6 miesięcy i 3 miesięcy. Najpierw planuje się co zostanie wykonane w horyzoncie rocznym, z horyzontu rocznego wybiera się zadania do wykonania w horyzoncie trzech miesięcy, itd. Z każdym takim przejściem uszczegółowiany jest zakres prac począwszy od planu, poprzez epiki, historie użytkownika i na zadaniach skończywszy.

W rozdziale przedstawiono szkic podejścia w definiowaniu modelu metodyki Scrumban. To podejście musi być dalej rozwijane, by w sposób bardziej formalny zdefiniować tę metodykę.

Case study

W rozdziale przedstawiona zostanie rama pracy (ang. framework) zespołu deweloperskiego bazująca na Scrumbanie. Przygotowany model bazuje na przykładzie dużej międzynarodowej firmy informatycznej. Opisane praktyki i zasady są wynikiem ustaleń między członkami zespołu oraz efektem narzuconych przez szczebel zarządzający zasad, które muszą być przestrzegane w związku z polityką firmy. Zaprezentowane podejście nie jest podejściem

uniwersalnym, które sprawdzi się w każdym zespole chcącym podjąć pracę z użyciem metodyki Scrumban. Może być ono jednak źródłem cennych obserwacji, które później można dostosować do potrzeb zespołu, bądź projektu, a także swego rodzaju przestrogą na jakie aspekty należy uważać i kłaść większy nacisk z powodu potencjalnych problemów. Wszystkie opisane praktyki są zgodne z założeniami metodyki Scrumban, która, jak powszechnie wiadomo, bazuje na elastyczności względem wymagań biznesowych i dopuszcza wprowadzanie modyfikacji w pracy zespołu.

Zespół deweloperski. Wdrożenie metodyki Scrumban rozpoczęto od procesu formowania nowego zespołu. Od samego początku współpracy zespół bardzo mocno akcentuje pracę zespołową (ang. teamwork). Przejawia się ona praktycznie w każdym aspekcie, poczynając od specjalistów z różnych dziedzin, którzy wymieniają się wiedzą z pozostałymi, przez wspólne omawianie nowych rozwiązań, poprzez tzw. kross-funkcjonalność, która nie ogranicza np. jednego dewelopera do pracy nad tylko jednym komponentem aplikacji. W sytuacji problemu, przykładowo pojawiającego się na środowisku produkcyjnym, jest on każdorazowo omawiany na forum. Dzięki międzyfunkcjonalności zespołu zawsze znajdzie się osoba, która ma pomysł jak rozwiązać dane zagadnienie. Czasami jest to deweloper, czasami tester, a czasami analityk. Deweloperzy tworzą między sobą mniejsze podzespoły, np. dwuosobowe i często pracują razem nad danym zadaniem. Z reguły wynika to ze złożoności zagadnienia bądź małego doświadczenia jednego z deweloperów. Zespół ma pod opieką dwie aplikacje. Jeśli w danym momencie druga aplikacja nie ma żadnych nowych zmian, wtedy cały zespół pracuje nad zadaniami związanymi z pierwszą aplikacją. Jeśli zadanie jest bardziej złożone, wtedy deweloperzy pracują nad nim w grupie 2 osobowej. Finalnie, takie podejście bardzo dobrze się sprawdza i nie powoduje niepotrzebnych przestojów w przepływie pracy. W skład zespołu, który został uzgodniony na szczeblu zarządzającym, wchodzą:

- Analityk, którego głównym zadaniem jest pisanie historyjek użytkownika w postaci przypadków testowych.
- Czterech deweloperów Java odpowiedzialnych za utrzymanie i rozwój aplikacji.
- Dwóch testerów manualnych, którzy za pomocą testów funkcjonalności sprawdzają, czy aplikacja działa poprawnie.
- Dwóch testerów automatycznych, którzy są odpowiedzialni za pisanie skryptów sprawdzających daną funkcjonalność, a także za testy regresji.

Dodatkowo, powołano kilku ekspertów oraz konsultantów, którzy czynnie uczestniczą w życiu zespołu:

- Konsultant ds. procesu wdrażania i ciągłego doskonalenia (ang. Development process and continuous improvement), który jest odpowiedzialny za prowadzenie niektórych spotkań, czuwanie nad prawidłowym procesem wdrażania nowych zmian, a także przygotowywaniem codziennych raportów z aktualnym stanem prac (tzw. Daily Update).
- Konsultant ds. historyjek użytkownika, który w razie pytań ze strony analityka udziela odpowiedzi, a także sprawdza poprawność przygotowywanych przypadków testowych.
- Konsultant ds. architektury systemu, który podejmuje kluczowe decyzje dotyczące strony technicznej aplikacji i jej komponentów.
- Konsultant ds. testów automatycznych, który czuwa nad poprawnym przebiegiem testów automatycznych i regresyjnych.
- Konsultant ds. testów użytkownika (ang. UAT) oraz procesu zarządzania zmianą (ang. Change Management), który jest odpowiedzialny za przebieg testów UAT oraz dostarczania informacji od użytkowników biznesowych na temat wymaganych zmian.
- Konsultant ds. zarządzania administracją, tzw. security coordinator, który jest odpowiedzialny za nadawanie dostępów dla członków zespołu czy zakładania tzw. requestów dla poszczególnych osób.
- Jeden deweloper baz danych, który wykonuje zadania związane z bazą danych, jednak jest ich na tyle mało, że występuje on w charakterze konsultanta, a nie członka zespołu.
- Konsultanci z poprzedniego zespołu, którzy przekazują wiedzę dotyczącą aplikacji w postaci dokumentacji oraz spotkań przekazywania wiedzy.

Aktualnie nie ma odpowiednika Scrum Mastera. Zespół rozważał wprowadzenie takiej roli, przypisując ją analitykowi. W zespole nie ma właściciela produktu. Jego rolę w pewnym stopniu spełniają powołani konsultanci (konsultant ds. historyjek użytkownika oraz konsultant ds. architektury systemu). Zespół jest stosunkowo młody, więc być może w toku pracy oraz rozwoju aplikacji zostanie powołana osoba pełniąca funkcję właściciela produktu (Product Owner).

Rejestr produktowy zespołu (Backlog). Rejestr zawiera wymagania biznesowe, zadania techniczne, incydenty, defekty etc. Zespół zajmuje się utrzymywaniem dwóch produktów, czyli tutaj można już mówić o Backlogu zespołu.

Każda aplikacja ma swoje oznaczenie – APP1 (aplikacja 1) i APP2 (aplikacja 2), dzięki czemu epiki oraz historyjki zawierają odpowiednie etykiety (APP1/APP2). W ten sposób łatwo rozróżnić jakiej aplikacji dotyczy dany epik czy historyjka. Pojawił się jednak problem priorytetyzacji. W pierwszej kolejności wybierane są zadania, które dostarczają największą wartość dla użytkowników końcowych. Kierowanie się wyłącznie wartościami biznesowymi jednej aplikacji nie sprawdziło się. Wprawdzie druga aplikacja nie ma graficznego interfejsu użytkownika GUI, to wciąż jest niezbędnym elementem, bez którego reszta ekosystemu nie będzie działała poprawnie. Zespół stanął przed dylematem jak najlepiej ustalać kolejność zadań dla obu produktów, tak, aby z jednej strony użytkownicy końcowi otrzymywali na czas zgłaszane wymagania, a z drugiej druga aplikacja była ciągle rozwijana, bez żadnych przestojów. Dodatkowo, charakter produktów dopuszcza częste zmiany priorytetów, np. gdy pojawi się incydent na produkcji, musi być on rozwiązany w pierwszej kolejności, a zadania biznesowe schodzą wtedy na dalszy plan. Tutaj pojawił się kolejny problem, czyli kto powinien podejmować decyzje o kolejności elementów w Backlogu. Od początku istnienia zespołu nie było wyznaczonej osoby, która byłaby za to odpowiedzialna. Poniekąd rozwiązaniem tego problemu są spotkania pielęgnacji rejestru produktowego (Backlog Refinement), na których analityk omawia z przedstawicielami biznesu decyzje o kolejności zadań. W praktyce są to tylko kwestie dotyczące jednej aplikacji, inne decyzje pozostają w gestii ustaleń między managerem a całym zespołem. Obecnie rejestr zespołu utrzymywany jest w oprogramowaniu Jira, w formie tablicy wirtualnej. Tablica ma możliwość wyszukiwania po zastosowanych filtrach – według aplikacji, wersji wydania, bądź statusu zaplanowane, niezaplanowane. Każde zadanie ma przypisaną wersję wydania, w którym będzie wdrażane na produkcję oraz osobę, która obecnie zajmuje się danym elementem.

Tablica Scrumban. Po dyskusjach wśród członków zespołu zdecydowano się na bardzo rozbudowaną tablicę Scrumban zawierającą następujące kolumny:

1. TODO (Do zrobienia) – zawiera wyłącznie epiki, które nie są jeszcze przypisane do nikogo, jedynie sygnalizują jakie prace będą do wykonania w najbliższym czasie.
2. PRE-ANALYSIS (Preanaliza) – zawiera wyłącznie epiki, ale już przypisane do konkretnej osoby. W tym statusie epiki są analizowane przez analityka i zaczyna się proces tworzenia historyjek użytkownika.
3. STORY CREATION (Tworzenie historyjek użytkownika) – na tym etapie zespół zaczyna działać na historyjkach użytkownika, analityk

tworzy dla historyjek scenariusze testowe, które później służą deweloperom i testerom. Użytkownicy biznesowi akceptują to, że napisana historyjka jest zgodna z początkowymi założeniami i została poprawnie zrozumiana.

4. TECHNICAL ANALYSIS (Analiza techniczna) – gotowe historyjki użytkownika analityk przedstawia deweloperom do zaimplementowania.
5. READY FOR DEVELOPMENT (Gotowe do rozwoju, rozbudowy) – po dyskusji analityka z deweloperami i potwierdzeniu, że wymagania są zrozumiałe, historyjki w tym statusie czekają na swoją kolej do realizacji.
6. DEVELOPMENT (Rozwój, rozbudowa) – deweloperzy implementują wymagane funkcjonalności zgodnie z przypadkami testowymi stworzonymi przez analityka.
7. READY FOR TESTING (Gotowe do testowania) – zaimplementowane historyjki oczekują na rozpoczęcie testów przez testerów.
8. TESTING (Testowanie) – historyjki są testowane, w zależności od rodzaju zmiany przeprowadzane są testy regresyjne, dedykowane testy automatyczne lub testy manualne.
9. READY FOR UAT (ang. User Acceptance Tests) (Gotowe do testów akceptacyjnych) – przetestowane oczekują na testy akceptacyjne, czyli powiązane z użytkownikiem końcowym.
10. UAT (Testy akceptacyjne) – faza ta powiązana z użytkownikiem końcowym lub osobą odpowiedzialną za procesy biznesowe pozwala zweryfikować, czy zaimplementowana funkcjonalność jest zgodna z pierwotnymi oczekiwaniami.
11. READY FOR RELEASE (Gotowe do wydania) – po pomyślnych testach z użytkownikami końcowymi, nowe funkcjonalności mogą być wystawione na środowisko produkcyjne.
12. DONE (Zrobione) – historyjka została zakończona pomyślnie, wymagane zmiany zostały wdrożone na środowisko produkcyjne i działają bez błędów.

Co bardzo istotne, tablica ma odzwierciedlać całą pracę zespołu. Zawiera więc zadania, które dotyczą zarówno jednej, jak i drugiej aplikacji. Każda kolumna ma określone WIP, czyli limity pracy w toku. W praktyce, przy utrzymywaniu dwóch aplikacji, często te limity nie są przestrzegane i w danej kolumnie jest więcej zadań. Nie jest to jednak problem, ponieważ zespół szybko radzi sobie z nadmiarową pracą.

Wydawanie oprogramowania. Zespół nie zakłada pracy w Sprintach, opiera się na wypuszczaniu nowych zmian na środowisko produkcyjne na podstawie otrzymywanych wymagań od biznesu, dążeniu do spłacenia długu technicznego oraz szybkiej reakcji na występujące błędy. Wyznacznikiem są wydania (tzw. release'y), realizowane średnio, choć nie zawsze, co 3 tygodnie w weekendy. Każde wydanie następuje, gdy zespół zakończy wymagany zakres prac, oczywiście w porozumieniu z interesariuszami. Epiki/historyjki wchodzące w zakres danego wydania mają przypisaną informację nt. wersji wydania. Jest to widoczne na tablicy scrumbanowej.

Celem każdego wydania jest wdrożenie nowych zmian na produkcję, najlepiej z dodatnią wartością biznesową. Każde wydanie ma określony zakres, który często w trakcie realizacji ulega zmianie. Najczęściej jest to wynikiem defektów występujących na produkcji, które zespół rozwiązuje z najwyższym priorytetem. Obecnie nowe zmiany zawsze są wdrażane w weekendy, kiedy użytkownicy nie korzystają z aplikacji. Zespół dąży jednak do wprowadzenia tzw. Mid Week Deployment'ów, które pozwolą na szybkie wprowadzanie zmian w trakcie tygodnia roboczego.

Co bardzo istotne, data wydania musi być dokładnie ustalona z użytkownikami biznesowymi, zespołem, ale także przedstawicielami innych aplikacji, które wchodzą w skład całego ekosystemu. W związku z tym powstał dedykowany kalendarz, który pokazuje możliwe terminy oraz niedostępne okna. Przykładowo, czerwiec 2022 wyglądał następująco:

1. 04/05 czerwca – okno możliwe do wdrażania nowych zmian.
2. 11/12 czerwca – okno możliwe do wdrażania nowych zmian.
3. 18/19 czerwca – okno, w którym pierwszeństwo mają zmiany techniczne.
4. 25/26 czerwca – okno niedostępne (tzw. Limited Window).

Zespół zdecydował się na termin 11 czerwca, kiedy zostaną wdrożone zarówno wymagania biznesowe, jak i zmiany spłacające dług techniczny aplikacji.

Przestrzeń do gromadzenia dokumentacji. Następną istotną kwestią jest miejsce gromadzenia dokumentacji oraz innych danych związanych z aplikacją. W związku z odziedziczeniem programu od innego zespołu, otrzymano także dokumentację. Tutaj pojawił się problem chaosu, ponieważ zawarte w niej informacje nie były zgodne ze stanem faktycznym np. kodu, a także była ona zgromadzona w postaci wielu plików znajdujących się w różnych lokalizacjach na wspólnym dysku. Z uwagi na to zespół oddelegował jedną osobę, w tym przypadku konsultanta ds. historyjek użytkownika, która zajęła się uporządkowaniem dostarczonej dokumentacji i przeniesieniem jej na de-

dykowaną dla zespołu przestrzeń na Confluence. Po wspólnych ustaleniach wewnątrz zespołu, stworzono następujące podstrony na Confluence:

1. Uporządkowana dokumentacja otrzymana od poprzedniego zespołu.
2. Opis procesu rozwoju oprogramowania – opis przepływu, błędów, statusów i innych szczegółów technicznych.
3. Opis środowisk – testowe, produkcyjne, a także sposoby połączenia ich z innymi komponentami ekosystemu.
4. Epiki oraz przygotowane do nich historyjki użytkownika z przypadkami testowymi.
5. Prod Queries – zapytania do bazy danych, które wyciągają konkretne dane, realizowane jako tzw. zadania ad hoc.
6. QA (ang. Quality Assurance) – opis strategii testowej oraz raporty z przeprowadzonych testów.
7. Historia wydań wdrażanych zmian, czyli tzw. release'ów, która zawiera opis w postaci tzw. Deployment Plan.
8. Opis innych komponentów systemu – szczegóły techniczne, logika biznesowa.

Spotkania zespołu. Zespół zakłada odbywanie czterech podstawowych spotkań:

- Dzienne spotkanie (Daily Stand up), odbywa się codziennie, czas trwania około 15 minut, obowiązkowe dla całego zespołu deweloperskiego. Zakres spotkania obejmuje: Omówienie przez każdego członka zespołu zakresu prac widocznych na tablicy; Omówienie blokad, przeszkód lub innych czynników, które uniemożliwiają dalszą realizację pracy; Omówienie bieżących spraw zespołu.
- Pielęgnacja rejestru produktowego (Backlog Refinement) odbywa się wtedy, kiedy zajdzie potrzeba i jest obowiązkowa dla analityka oraz konsultanta ds. historyjek użytkownika. Kluczowymi interesariuszami są przedstawiciele użytkowników biznesowych oraz osoby podejmujące kluczowe decyzje biznesowe odnośnie aplikacji. Zakres spotkania obejmuje: Omówienie i sprawdzenie zadań czekających w rejestrze produktowym zespołu; Ustalenie kolejności wykonywanych zadań, zgodnie z priorytetami biznesowymi oraz wymaganiami technicznymi. Na spotkaniach nie ma dokładnego szacowania konkretnych wymagań, jedynie ustalanie priorytetów i kolejności wykonywania wymagań zgłoszonych przez obecnych na spotkaniu interesariuszy. Spotkania pielęgnacji rejestru produktowego mają

charakter „biznesowy" – ustala się na nich kolejność dostarczanych wymagań zgłoszonych przez użytkowników biznesowych. Szczegółowe szacowanie wymagań odbywa się na spotkaniach planistycznych wewnątrz zespołu.

- Spotkanie planistyczne (Planning) odbywa się wtedy, kiedy zajdzie potrzeba. Wynika to z ciągłego aktualizowania rejestru prac przez analityka po spotkaniach pielęgnacji rejestru produktowego. Z kolei estymaty nie są wymagane do każdego zadania. Zakres spotkania obejmuje: Planowanie pracy na podstawie ustalonej podczas spotkania pielęgnacji rejestru produktowego kolejności zadań; estymowania (szacowanie) złożoności prac do wykonania, najczęściej podawane w roboczodniach (ang. mandays). Spotkanie planistyczne ma charakter bardziej techniczny – są na nim obecni wszyscy członkowie zespołu, od analityka, przez deweloperów, po testerów i każda z osób szacuje, ile czasu jest potrzebne z jej pespektywy do wykonania omawianego zadania. Użytkownicy biznesowi nie są obecni na tym spotkaniu, ponieważ dotyczy ono technicznych szczegółów, a nie aspektów stricte biznesowych. Po spotkaniu planistycznym analityk oraz konsultant ds. historyjek użytkownika wiedzą, ile mniej więcej potrzeba będzie czasu na zrealizowanie danego wymagania. Wtedy na spotkaniu pielęgnacji rejestru produktowego informują interesariuszy, ile zajmie zrealizowanie danego wymagania.
- Spotkanie retrospektywy (Retro) początkowo miało odbywać się co 2 tygodnie (z reguły w poniedziałek po weekendowym wypuszczeniu nowych zmian na środowisko produkcyjne), miało czas trwania 1 godzinę i było obowiązkowe dla całego zespołu deweloperskiego. Zakres spotkania obejmuje: omówienie elementów, które w ostatnim wydaniu zmian poszły dobrze, co należy poprawić i jakie działania należy w tym celu podjąć. W toku rozwoju prac, zespół podjął decyzję o rezygnacji z regularnego odbywania retrospektywy.

Monitorowanie i kontrola. Aktualnie zespół nie prowadzi monitorowania ani nie sporządza wykresów wypalania. Kontrolę czy zadania są realizowane w odpowiedniej kolejności obecnie sprawuje analityk. Codziennie odbywają się Daily Stand up. Można powiedzieć, że przegląd zakresu prac na najbliższe wydanie jest w pewnym stopniu wykonywany podczas spotkania planistycznego, choć nie zawsze omawiane są na nim wszystkie wymagania wchodzące w zakres wydania. Zespół jest bardzo elastyczny i dopasowuje potrzeby do obecnego stanu prac.

Definition of Done. Kolejną kwestią było ustalenie definicji ukończenia (ang. Definition of Done) zadań w poszczególnych statusach. Zespół zdecydował się na następujące ustalenia:

- TODO (Do zrobienia) – każde zadanie ma nadany priorytet i jest zrozumiałe.
- BUSINESS ANALYSIS (Analiza biznesowa) – realizowane zadanie jest w pełni zrozumiałe, historyjki użytkownika są stworzone zgodnie z zasadą INVEST i umieszczone na odpowiedniej podstronie na Confluence. Zawierają także odpowiednie kryteria akceptacji.
- TECHNICAL ANALYSIS (Analiza techniczna) – realizowane zadanie jest zrozumiałe zarówno na poziomie biznesowym, jak i technicznym, wszystkie szczegóły są zawarte w historyjkach użytkownika. Zadanie jest przeglądnięte i zrozumiałe przez cały zespół.
- DEVELOPMENT (Rozwój, rozbudowa) – kod z nowymi zmianami jest gotowy, zalogowany w repozytorium. Po sprawdzeniu kodu (tzw. code review) wykonywane są testy jednostkowe, a następnie zmiany są wdrażane na środowisko testowe.
- TESTING (Testowanie) – testy nowych zmian są wykonywane na środowisku testowym, wykryte błędy są naprawiane, testowane ponownie i zaakceptowane. Wyprowadzone zmiany muszą pokrywać wszystkie wymagania opisane w historyjkach. Jeśli jest możliwość, testowany jest cały przepływ oraz uruchamiane są testy regresyjne aplikacji.
- UAT (Testy akceptacyjne) – wprowadzane zmiany są wdrożone na środowisku przeznaczonym do testów akceptacyjnych, przeprowadzane są testy z użytkownikami końcowymi. Ewentualne problemy są naprawiane, testowane ponownie i akceptowane.

Podejście DevOps pozwala na automatyzację procesu rozwoju (ang. Development) oraz utrzymania (ang. Operation) oprogramowania [14]. Zespół wspólnie podjął decyzję o wdrożeniu tego rozwiązania, decydując się na otwarty łańcuch dostępnych narzędzi :

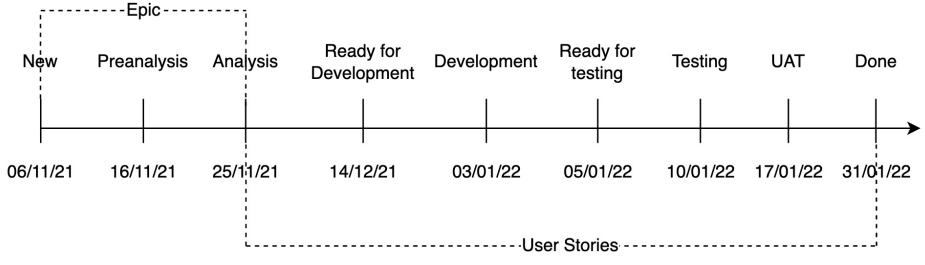
- Planowanie i śledzenie postępów prac, przechowywanie – Jira, Confluence.
- Komunikacja członków zespołu, z interesariuszami – Skype for Business, Microsoft Teams.
- Zintegrowane środowisko programistyczne (IDE) – IntelliJ IDEA.
- Rozproszony system kontroli wersji, dla kodu i skryptów – Git, Nexus Repository.

- Serwer ciągłej integracji – Jenkins, Bitbucket.
- Optymalizacja procesu testowania aplikacji – qTest.
- Konteneryzacja – Docker.
- Monitorowanie wydajności aplikacji – Splunk.

Wszystkie wymienione powyżej narzędzia bardzo dobrze ze sobą współgrają, dzięki czemu proces dostarczania nowych zmian jest sprawny i, na ile jest to możliwe, zautomatyzowany. Jakość wdrażanego oprogramowania jest zdecydowanie lepsza, co wpływa na zadowolenie użytkowników końcowych.

Historyjki użytkownika. Zespół bazuje na epikach, z których tworzone są historyjki użytkownika. Odnosi się to zarówno do wymagań zgłaszanych przez użytkowników biznesowych, jak i defektów, które należy naprawić. Za pisanie historyjek użytkownika (user stories) odpowiedzialny jest analityk, który może liczyć na wsparcie konsultanta ds. historyjek użytkownika. Na etapie układania historyjek, analityk korzysta z metody 5 WHY, która wywodzi się z koncepcji Lean Management. Z reguły tę metodę stosuje się w odniesieniu do defektów, czy innych problemów, jednak sprawdza się także przy pisaniu historyjek użytkownika. Dzięki temu każda zmiana jest bardzo dokładnie przemyślana, co minimalizuje ryzyko pominięcia jakiegoś aspektu. Tworząc historyjki, analityk wykorzystuje również podejście INVEST. Każda historyjka użytkownika pisana jest w postaci przypadków testowych. Z jednej strony zawierają w sobie wszystkie wymagane zmiany biznesowe, a z drugiej są to gotowe przypadki testowe dla testerów. Epik zawiera w sobie opis biznesowy, wysokopoziomowe wymagania użytkowników. Z kolei historyjka zawiera w sobie szczegóły techniczne, przedstawione w postaci scenariuszy testowych.

Rysunek 1. Oś czasu realizacji epiku



Źródło: opracowanie własne.

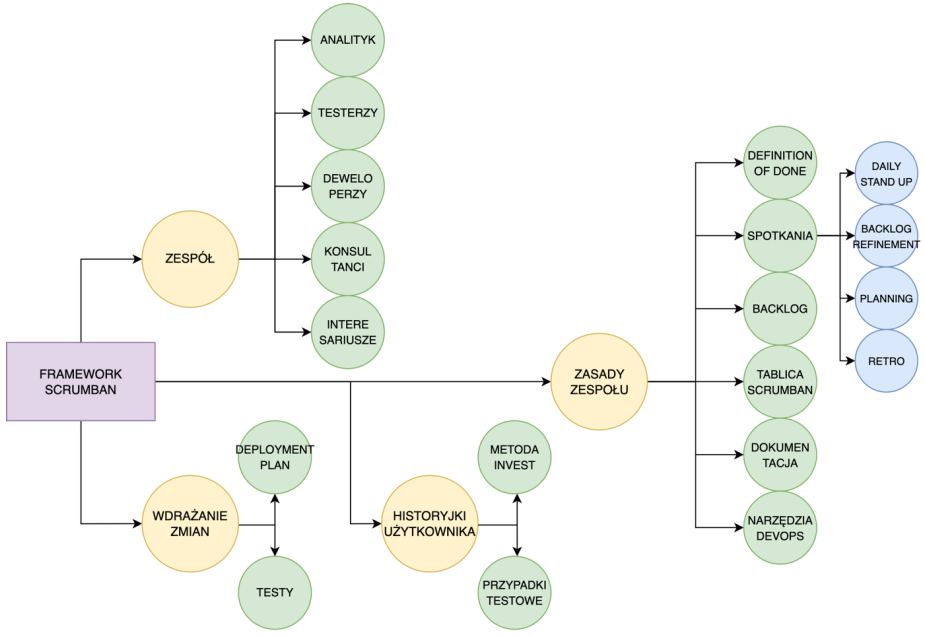
Oś czasu (Rys. 1) przedstawia, jak kształtowały się zmiany poszczególnych statusów, najpierw epiku, a później samej historyjki. Nowe wymagania zostały zgłoszone 6 listopada 2021, a wdrożone na środowisko produkcyjne 31 stycznia 2022. Czas realizacji wymagania nie był najkrótszy, jednak było to podyktowane dużą ilością innych zadań w toku.

Wdrażanie zmian na środowisko produkcyjne

Aby nowa zmiana mogła zostać pomyślnie wdrożona na produkcję, muszą być spełnione następujące warunki ze strony testów:

- Każda wdrażana historyjka została przetestowana przez testerów na środowisku testowym zgodnie z wymaganiami akceptacyjnymi (Acceptance criteria).
- Testy wydajnościowe (ang. Performance) oraz bezpieczeństwa (ang. Security) zostały pomyślnie wykonane.
- Testy regresyjne całej aplikacji przeszły pomyślnie, czyli ponad 90% przypadków testowych zostało poprawnie przetworzone.
- Testy UAT zostały pomyślnie wykonane przez użytkowników końcowych.
- Brak błędów krytycznych lub o wysokim priorytecie.
- Raport „Acceptance Phase Summary" został zatwierdzony przez osoby decyzyjne, m.in. właściciela biznesu (Business Owner), konsultanta ds. testów automatycznych, czy konsultanta ds. architektury systemu. Taki dokument zawiera szczegółowe wyniki przeprowadzonych testów i jest formalnym i ostatecznym wyrażeniem zgody na wyjście z nowymi zmianami na środowisko produkcyjne.

Rysunek 2. Elementy wdrożenia Scrumban.



Źródło: opracowanie własne.

Deployment Plan

Obowiązkowym elementem każdego wydania jest plan wdrożenia, tzw. Deployment Plan. Jest on przygotowywany przez deweloperów oraz analityka w formie podstrony na Confluence i zawiera następujące elementy:

- Zbiór wszystkich epików, historyjek, tasków, które będą wdrażane.
- Informacje o tzw. Change request'ach, które są wymagane przy każdych zmianach w systemie i zawierają szczegółowe informacje na temat wdrażanych zmian.
- Adresy mailowe do osób, które będą odpowiedzialne za wdrażanie zmian.
- Działania, jakie należy podjąć przed wdrożeniem zmian (np. testy bezpieczeństwa aplikacji (ang. Security Scan).
- Opisane krok po kroku działania samego wdrożenia:

a) Business pre check – sprawdzenie przez użytkowników biznesowych aplikacji (najczęściej polega na eksporcie danych zastanych).

b) Wyłączenie aplikacji.
c) Wprowadzenie zmian w bazie danych.
d) Wgranie nowych zmian przy pomocy job'a Jeniks.
e) Walidacja wprowadzonych zmian przez dewelopera (sprawdzenie czy aplikacja działa, czy użytkownicy mają do niej dostęp).
f) Weryfikacja wprowadzonych zmian przez użytkownika biznesowego (porównanie wyeksportowanych wcześniej danych ze stanem obecnym, sprawdzenie elementów GUI).
g) Mail informujący o wdrożeniu nowych zmian.

Wdrożenie wykonywane jest przed jednego dewelopera Java z zespołu, jednego konsultanta ds. bazy danych, jednego użytkownika biznesowego oraz jeśli zmiany dotyczą innych systemów, przez wyznaczone do tego osoby. Z reguły taki proces trwa maksymalnie do 2 godzin. Podsumowanie użytej struktury Scrumban przedstawia Rys. 2.

Zidentyfikowane problemy. Zespół, na którym bazuje opisane wdrożenie metodyki Scrumban, jest dość młodym zespołem, który powstał w listopadzie 2021. W toku rozwoju pracy i gromadzonego doświadczenia, można dostrzec kilka elementów, które wciąż wymagają pewnego dopracowania.

Pierwszy aspekt to odpowiedzialność za ustalanie priorytetów. Zespół od początku swojego istnienia nie miał wyznaczonej jednej osoby, która będzie zajmowała się tą kwestią. Jeszcze większy problem pojawił się w momencie przejęcia drugiej aplikacji, ponieważ jedna i druga wymagają równoczesnego rozwijania. W przypadku opisywanego zespołu, kwestia priorytetów jest niezwykle dynamicznym procesem. Często pojawiające się problemy na środowisku produkcyjnym wymagają reorganizacji pracy, przez co wymagania biznesowe, które nie zostały zgłoszone w wyniku defektu, są odciągane w czasie. Jednym z rozwiązań są częste spotkania pielęgnacji rejestru (Backlog Refinement), na których przedstawiciele użytkowników końcowych podają swoje priorytety, jednak wciąż wymaga to konfrontacji z wymaganiami drugiej aplikacji. Finalnie, decyzja o kolejności zadań jest wynikiem wspólnego porozumienia między analitykiem, konsultantem ds. historyjek użytkownika, konsultantem ds. architektury systemu, managerem wyższego szczebla oraz przedstawicielami użytkowników biznesowych.

Drugi aspekt dotyczy weekendowego wdrażania nowych zmian na środowisko produkcyjne. Obecne ograniczenia systemu pozwalają na wprowadzanie modyfikacji w aplikacji tylko w weekendy, kiedy aplikacja nie jest używana przez użytkowników. Jest to problematyczne zarówno od strony dewelope-

rów, jak i użytkowników biznesowych i konsultantów, który muszą pracować dodatkowy dzień w tygodniu. Ta kwestia była wielokrotnie poruszana na wielu spotkaniach i obecnie zespół pracuje nad rozwiązaniem technicznym, które umożliwi tzw. Mid Week Deployment'y, czyli wdrażanie zmian w dni robocze, bez negatywnego wpływu na użytkowników.

Kolejną kwestią jest chaos w dokumentacji. Ten problem był szczególnie widoczny na początku, kiedy większość dokumentów została odziedziczona po innym zespole. Było to bardzo uciążliwe, biorąc pod uwagę konieczność sprawdzania wielu szczegółów, szczególnie na początkowym etapie pracy z nową aplikacją. Podczas formowania się zasad zespołu i wypracowywania pewnych ustaleń, dokumentacja została poddana stopniowemu uporządkowaniu, choć jest to bardzo długotrwały proces i wciąż wymaga dopracowania.

Ostatni zaobserwowany aspekt dotyczy spotkań zespołu, a mianowicie brak regularnej retrospektywy. Scrumban dopuszcza modyfikację spotkań i dostosowania się do potrzeb każdego zespołu, jednak w tym przypadku częściowa rezygnacja z retrospektywy nie była najlepszą praktyką. Bardzo często kwestie, które powinny być omawiane na retrospektywie, są dyskutowane na Daily Stand up. Przez to wiele aspektów nie zostaje do końca omówionych i wielokrotnie są ponownie poruszane na następnym spotkaniu.

Dobre praktyki. Bazując na zgromadzonej wiedzy i doświadczeniu pracy w zespole scrumbanowym, można wyróżnić kilka zasad, które warto wdrożyć w swoim zespole, aby poprawić jego wydajność, a tym samym podnieść jakość dostarczanych usług.

1. Rozbudowana tablica scrumbanowa. Im więcej kolumn zawiera, tym lepiej widoczny jest przepływ pracy i zadania szybciej przesuwają się w prawo. Dokładnie widać co już zostało wykonane, a co jeszcze należy zrobić. Taka rozbudowana tablica, wbrew wielu opiniom, bardzo dobrze sprawdza się w praktyce i jest chętnie stosowana przez różne zespoły.
2. Historyjki jako przypadki testowe. Przestawianie historyjek użytkownika w formie przypadków testowych przy użyciu konstrukcji given, when, then pomaga w pracy nie tylko testerom, ale także deweloperom. Takie rozwiązanie jest bardzo czytelne, ponieważ zawiera wszystkie szczegóły, jednak bez zbędnych opisów.
3. Wyznaczenie jednej osoby odpowiedzialnej za ustalanie priorytetów. Dzięki temu zespół ma pewność, że jest ktoś, kto dba o porządek i kolejność zadań w rejestrze produktowym zgodnie z wymaganiami zarówno

biznesowymi, jak i technicznymi. Może to być manager, właściciel produktu bądź sam analityk.
4. Odbywanie spotkań, gdy zespół wyraża takie chęci. Ta zasada wynika poniekąd ze specyfiki metodyki Scrumban. Zespół może wspólnie podjąć decyzję jakie spotkania chce odbywać, a z których zrezygnuje bądź na pewien czas modyfikuje. Oczywiście należy pamiętać, że nie można całkowicie zaniechać wszystkich spotkań, jednak można je dostosować do aktualnego zakresu prac czy potrzeb zespołu.
5. Korzystanie z oprogramowania Jira. Ostatnia praktyka dotyczy używanych narzędzi. Oprogramowania Jira oferuje szeroki wachlarz przydatnych usług, od planowania, śledzenia postępu prac, przez gromadzenie dokumentacji, poprzez integrację z innymi aplikacjami pochodzącymi z ekosystemu Atlassian, np. qTest, czy Bitbucket. Takie rozwiązanie bardzo dobrze się sprawdza i pozwala na przechowywanie wielu informacji związanych z aplikacją w jednym miejscu.

Podsumowanie

Odpowiednio dobrana metodyka do zarządzania projektem to jeden z ważniejszych aspektów powodzenia pracy zespołu. Wpływa nie tylko na właściwą organizację zadań, sposobu i prędkości dostarczania przyrostów, ale także na wysoką jakość usług, a tym samym na zadowolenie klienta.

W wyniku badań literaturowych i na podstawie własnego doświadczenia można określić proste reguły wyboru metody spośród trzech podejść Scrum, Kanban oraz Scrumban (Tab. 2).

Tabela 2. Proste zasady wyboru pomiędzy podejściami Scrum, Kanban oraz Scrumban

Reguła	Scrum	Kanban	Scrumban
Czy zespół definiuje wymagania?	Definiowane z góry, nie przewiduje się zmian w czasie trwania Sprintu	Definiowane na bieżąco lub na żądanie	Definiowane z góry lub na żądanie
Czy zespół pracuje w iteracjach?	Tak	Nie	Zależy od potrzeb

Z czego wynikają limity zadań?	Z zaplanowanego Sprintu	Z pracy w toku (ang. Work In Progress)	Z pracy w toku oraz kolumny TODO
Czy zespół szacuje złożoność wymagań?	Tak, obowiązkowo przed każdym Sprintem	Nie	Opcjonalnie
Czy zespół odbywa regularne spotkania projektowe?	Tak, bez względu na potrzeby zespołu (codzienne, planistyczne, retrospektywy)	Tak, ale tylko codzienne, około 15-minutowe spotkania	Tak, ale w zależności od potrzeb zespołu
W jaki sposób zespół korzysta z tablicy?	Tablica jest resetowana przy każdym Sprincie i wszystkie zadania są przenoszone do kolumny TODO	Tablica jest w ciągłym użyciu i nie wymaga resetowania	Tablica jest w ciągłym użyciu, choć dopuszcza resetowanie jeśli jest taka potrzeba
Kiedy zespół priorytetyzuje zadania do wykonania?	Na etapie planowania Sprintu	Na bieżąco	Zarówno na etapie planowania iteracji, jak i na bieżąco
Jakiego rodzaju projekty realizuje zespół?	Wymagające stałej pracy zespołowej	Niewymagające stałej pracy zespołowej	Dopuszczające zarówno pracę zespołową, jak i indywidualną

Źródło: opracowanie własne.

Zaprojektowany w tej pracy framework metodyki Scrumban powstał na przykładzie istniejącego zespołu deweloperskiego. Cały opisany proces wdrożenia był implementowany dokładnie tak, jak zostało to nakreślone. Wyróżniono także wiele przydatnych praktyk, które mogą pomóc w ulepszeniu pracy zespołu scrumbanowego. Opisane w artykule zasady bardzo dobrze sprawdzają się w praktyce i mimo wielu potrzebnych usprawnień, praca zespołu przebiega płynnie, a użytkownicy końcowi są zadowoleni z otrzymywanych przyrostów. Zaproponowane rozwiązanie to tylko propozycja jak metodyka Scrumban może wyglądać praktycznie.

Literatura

1. *Scaling Agile for Enterprises: A Comparison of SAFe Agile, Nexus, Disciplined Agile 2.0 (DAD), and Large Scale Scrum (LeSS).* https://www.linkedin.com/pulse/scaling-agile-enterprises-comparison-safe-nexus-20-raj. Accessed 6 Feb 2023
2. Ponomareff D (2017) Kanban Zone. In: *Kanban Zone – Vis. Collab. Lean Agile Portf. Proj. Manag.* https://kanbanzone.com/2017/scrumban-blending-agile-scrum-kanban/. Accessed 6 Feb 2023
3. *Scrumban.* https://www.productplan.com/glossary/scrumban/. Accessed 6 Feb 2023
4. *Digital lean manufacturing. In: Deloitte Insights.* https://www2.deloitte.com/us/en/insights/focus/industry-4-0/digital-lean-manufacturing.html. Accessed 6 Feb 2023
5. Knapik I (2022) *Wdrożenie metodyki Scrumban w zespole deweloperskim na przykładzie dużej firmy informatycznej*
6. Werewka J (2018) *Developing Conformance Between Project Management and Enterprise Architecture Governance on the Basis of a PMBOK Case. In: Wilimowska Z, Borzemski L, Świątek J (eds) Information Systems Architecture and Technology: Proceedings of 38th International Conference on Information Systems Architecture and Technology – ISAT 2017.* Springer International Publishing, Cham, pp 233–242
7. Tomasz Kruzel, Jan Werewka (2011) *Application of BPMN for the PMBOK standard modelling to scale project management efforts in IT enterprises.* https://doi.org/10.13140/RG.2.1.1129.1364
8. Werewka J, Jamróz K, Pitulej D (2014) *Developing Lean Architecture Governance at a Software Developing Company Applying ArchiMate Motivation and Business Layers. In: Kozielski S, Mrozek D, Kasprowski P, et al (eds) Beyond Databases, Architectures, and Structures.* Springer International Publishing, pp 492–503
9. Werewka J, Spiechowicz A (2017) *Enterprise Architecture Approach to SCRUM Processes, Sprint Retrospective Example.* In: Ganzha M, Maciaszek LA, Paprzycki M (eds) Proceedings of the 2017 Federated Conference on Computer Science and Information Systems, FedCSIS 2017, Prague, Czech Republic, September 3-6, 2017. pp 1221–1228
10. Kopp A, Orlovskyi D (2022) *Enterprise Architecture Tools Assessment for Smart Cities Governance using Fuzzy Logic Techniques.* Tex J Eng Technol 6:9–16

11. Zhi Q, Zhou Z (2021) *Empirically Modeling Enterprise Architecture Using ArchiMate.* Comput Syst Sci Eng 40:357–374. https://doi.org/10.32604/csse.2022.018759
12. Lakhrouit J, Baïna K (2016) *Enterprise Architecture Complexity Component Based on Archimate Language.* In: Sabir E, Medromi H, Sadik M (eds) Advances in Ubiquitous Networking. Springer, Singapore, pp 535–546
13. *ArchiMate® 3.1 Specification.* https://publications.opengroup.org/c197. Accessed 29 Jul 2022
14. Molenda K, Partyka M, Werewka J *Dobór narzędzi rozwoju oprogramowania w cyklu DevO…* — Biblioteka Nauki. https://bibliotekanauki.pl/articles/2147418. Accessed 11 Jan 2023