Uniwersytet im. A. Mickiewicza w Poznaniu
Wydział Matematyki i Informatyki
kierunek: Informatyka

Praca magisterska

# Metodyka scrum w małych i średnich projektach informatycznych.

**Scrum methodology in small and medium software projects.**

Adam Franciszek Kędziora

Promotor:
dr Krzysztof Dyczkowski

Poznań, 2011

Poznań, 07.07.2011

# Oświadczenie

Ja, niżej podpisany Adam Franciszek Kędziora
student Wydziału Matematyki i Informatyki Uniwersytetu im. Adama Mickiewicza w Poznaniu oświadczam, że przedkładaną pracę dyplomową pt: Metodyka scrum w małych i średnich projektach informatycznych.
napisałem samodzielnie. Oznacza to, że przy pisaniu pracy, poza niezbędnymi konsultacjami, nie korzystałem z pomocy innych osób, a w szczególności nie zlecałem opracowania rozprawy lub jej części innym osobom, ani nie odpisywałem tej rozprawy lub jej części od innych osób. Oświadczam również , że egzemplarz pracy dyplomowej w formie wydruku komputerowego jest zgodny z egzemplarzem pracy dyplomowej w formie elektronicznej. Jednocześnie przyjmuję do wiadomości, że gdyby powyższe oświadczenie okazało się nieprawdziwe, decyzja o wydaniu mi dyplomu zostanie cofnięta.

.....

# Spis treści

# Rozdział 1

# Wstęp

Od 1968 roku, kiedy na konferencji w Garmisch został ukuty termin „inżynieria oprogramowania“, przemysł informatyczny nieustannie poszukuje jak najsprawniejszego sposobu na wytwarzanie oprogramowania. Najsprawniejszego, to znaczy pozwalającego tworzyć oprogramowanie szybko, możliwie niewielkim nakładem pracy ludzkiej, z jak najmniejszą ilością błędów i jak najlepiej odpowiadające potrzebom zamawiającego użytkownika. Przez te lata zmieniały się wielokrotnie uwarunkowania rynku, a przez to i oczekiwania klientów.

W ciągu ostatniego dziesięciolecia, bazując na obserwacjach czynionych od 1986 roku (patrz [7]), silnie rozwinęła się koncepcja wytwarzania oprogramowania w sposób zwinny. Najbardziej znanym zbiorem praktyk zwinnego wytwarzania oprogramowania jest metodyka scrum. Niestety, jeżeli chodzi o opracowania tej metodyki w języku polskim, sprowadzają się one jedynie do przetłumaczenia podstawowego opisu metodyki (tj, publikacji [4]). Aby uzyskać kompleksową wiedzę z zakresu metodyki nie wystarczy również przeczytać pojedynczej pozycji w języku angielskim - wiedza na temat scruma jest rozproszona po wielu książkach i artykułach, czasami dostępna jedynie poprzez rozmowy z praktykami scruma. Wiele opracowań podaje różne informacje w sposób, który wskazywałby na to, iż są one ze sobą sprzeczne - wynika to z konieczności dostosowywania metodyki do konkretnego zastosowania.

Celem tej pracy jest kompleksowy opis metodyki scrum, stworzony w oparciu o najnowszą obecnie dostępną literaturę, informacje zebrane na informatycz-

nych konferencjach branżowych oraz moje doświadczenia związane z implementacją scruma w miejscu pracy. Nie staram się napisać opracowania zawierającego wszystkie aspekty wiedzy związanej z tą metodyką, gdyż byłoby to zadanie karkołomne.

Praca adresowana jest do czytelnika, który posiada wiedzę na temat tradycyjnego sposobu tworzenia oprogramowania w modelu kaskadowym, lub spiralnym. Praca nie jest kompleksowym podręcznikiem dla przyszłego menadżera projektu - wiedza tu zawarta musi współgrać z podstawową wiedzą z zakresu zarządzania projektami, którą można znaleźć na przykład w PMBOK Guide wydawanym przez Project Management Institute.

Rozdział 2 opisuje czym jest ruch „agile“ i jakie są jego podstawowe zasady - są one integralną częścią metodyki scrum i powinny być naczelnym wyznacznikiem prawidłowości działań dowolnego zespołu scrumowego. Rozdział 3 opisuje w bardzo pobieżny sposób metodykę scrum, wprowadzając niezbędne pojęcia, jednak nie wyjaśniając ich w pełni - ma on na celu jedynie pokazanie obrazu całości, tak, by wiedzę zawartą w dalszych rozdziałach można było osadzać w pewnej całości. Rozdziały 4-7 opisują dokładnie poszczególne elementy metodyki (odpowiednio sposób zarządzania wymaganiami, iteracje i planowanie, role ludzkie i elementy osadzające zespół w organizacji) w najprostszej konfiguracji, czyli dla jednego niewielkiego zespołu. Ostatni, rozdział 8 zawiera opis sposobów skalowania scruma na większą liczbę zespołów, a więc i na większe projekty.

W pracy wielokrotnie używam angielskich terminów oraz spolszczeń (a nie tłumaczeń) tych terminów, jest to powszechnie przyjęta w literaturze praktyka. Podjęta próba przełożenia ich na język polski skutkowała tekstem niespójnym i trudnym do zrozumienia.

# Rozdział 2

# Metodyki agile

Metodyki agile powstały na fali ruchu niezadowolenia wobec sztywnych i silnie sformalizowanych metodyk wytwarzania oprogramowania oraz zarządzania projektami. Według twórców tego ruchu (tj. ruchu agile) metodyki te, nie były już adekwatne do aktualnie istniejących możliwości, jakie dawały miedzy innymi nowe techniki programistyczne, przez co korzystanie z nich (tj. metodyk ) powodowało duże marnotrawstwo zasobów.

## 2.1 Agile Manifesto

"We are uncovering better ways of developing software by doing it and helping others do it.

Through this work we have come to value:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more. "

Powyższy oryginalny tekst „agile manifesto“, znajduje się na stronie internetowej [1] i jest bazą - a w zasadzie jest jego kwintesencją[1] - całego ruchu agile. Twórcy manifestu na podstawie swoich doświadczeń uznali, że rzeczy wymienione po prawej stronie (procesy i narzędzia, dokładna dokumentacja, negocjowanie kontraktów, trzymanie się planu) rzeczywiście są ważne, lecz rzeczy po lewej (ludzie i interakcje między nimi, działające oprogramowanie, współpraca z klientem, reagowanie na zmiany) - istotniejsze. Można się nie zgadzać z tymi poglądami, szczególnie w specyficznych przypadkach, np. w przypadku napisania biblioteki programistycznej, daleko bardziej może być pożądana dokładna dokumentacja niż duża ilość działających funkcji, a w przypadku wykonywania oprogramowania dla instytucji rządowej - bardzo dokładny kontrakt może być ważniejszy niż wola współpracy. Ogólnie jednak manifest odniósł niewątpliwy sukces i jego wpływ można zauważyć również w nowych “odchudzonych” wersjach starszych metodyk, jak OpenUP będący potomkiem metodyki RUP .

Cztery punkty manifestu powstały w oparciu o dwanaście zasad (z [1]), które autorzy uznali za najważniejsze dla zapewnienia, by wytwarzanie oprogramowania było jak najlepsze - tak dla twórców oprogramowania, jak i klientów je zamawiających. Większość z nich jest sformułowana na podstawie doświadczeń (głównie złych) z dotychczasowym procesem wytwarzania oprogramowania.

1. “Naszym naczelnym zadaniem jest satysfakcjonowanie klienta poprzez szybkie i ciągłe dostarczanie wartościowego oprogramowania.”

2. “Witaj i akceptuj zmianę nawet późno w produkcji. Zwinny proces wykorzystuje zmiany dla zapewnienia przewagi konkurencyjnej klientowi.”

3. “Dostarczaj działające oprogramowanie często, od kilku tygodni, do kilku miesięcy, z naciskiem na terminy krótsze.”

4. “Ludzie biznesu i twórcy oprogramowania muszą pracować razem, codziennie w ciągu projektu.”

[1] Agile manifesto powstał 11-13 lutego 2001, stworzony był między innymi przez Kena Schwabera - który w 1995 stworzył metodykę Scrum - oraz Kenta Becka - który rok wcześniej wydał książkę o „extreeme programming“ - co jest dowodem na to, że metodyki zwinne, wbrew powszechnej opinii (błędnej) nie wywodzą się z manifestu zwinności a na odwrót.

5. “Buduj projekty w oparciu o zmotywowanych ludzi. Zapewnij im środowisko i wsparcie, którego potrzebują i wierz, że wykonają powierzone im zadanie.”

6. “Najlepszą i najbardziej efektywną metodą przekazywania informacji do i wewnątrz zespołu tworzącego oprogramowanie, jest rozmowa twarzą w twarz.”

7. “Działające oprogramowanie jest najważniejszą miarą postępu.”

8. “Procesy zwinne promują nieprzerwane wytwarzanie oprogramowania. Sponsorzy, twórcy oprogramowania i użytkownicy, powinni być w stanie w nieskończoność utrzymywać stałe tempo rozwoju.”

9. “Ciągłe poświęcanie uwagi technicznej doskonałości i dobremu projektowaniu polepsza zwinność.”

10. “Prostota - sztuka maksymalizacji pracy nie robionej - jest niezmiernie ważna.”

11. “Najlepsza architektura, wymagania i projekty powstają w samo-organizujących się zespołach.”

12. “W regularnych odstępach czasu, zespół zastanawia się, jak stać się bardziej efektywnym, następnie dostosowuje do tego swoje zachowania.”

Tak samo, jak w przypadku czterech punktów manifestu, tak i tu, można się nie zgadzać z autorami w pewnych kwestiach (szczególnie w punktach 2, 7, 8 czy 11), kolejne mogą się okazać w praktyce trudne (aż do niemożliwości) do zrealizowania (4), lub budzić kontrowersje i generować nieprzewidziane ryzyka (5, 6, 10). Ogólne przesłanie tych wytycznych jednak, przynajmniej na pierwszy rzut oka zdaje się być całkiem sensowne: “buduj oprogramowanie dla klienta i z klientem” stawiane w opozycji do: “buduj oprogramowanie dla ścisłego wypełnienia kontraktu”.

## 2.2 Agileowe metodyki zarządzania projektem

Agile pozwala przede wszystkim na szybkie reagowanie na zmiany - toteż zwinne metodyki zarządzania projektem z założenia będą się sprawdzały tam, gdzie w toku projektu przewiduje się w dużo zmian. Jest to bardo silny argument w rękach orędowników agile - dla wielu kontrahentów obietnica tego, że wprowadzanie zmian będzie przyjmowane przez zespół wykonujący jako coś na porządku dziennym, jest miłym zaskoczeniem po pracy z zespołami, gdzie na dowolną zmianę potrzeba było wykonać dużą ilość procedur zarządczych, dodatkowo wprowadzać aneks do umowy i zwykle coś dopłacić „extra“. Niestety, każdy medal ma dwie strony, tutaj drugą stroną jest trudność dokładnego przewidzenia terminu i budżetu projektu - większa niż w przypadku przeprowadzenia dokładnego planowania i późniejszego wykonywania planów [2].

Aby jakakolwiek szanująca swoje pieniądze firma zastosowała zwinne podejście do wytwarzania swoich produktów, należy ten problem możliwie minimalizować. We wszystkich znanych mi metodykach zwinnych problem minimalizuje się poprzez “Just In Time planning” (za [7]) czyli planowanie w momencie dokładnej potrzeby oraz modyfikację zakresu przy współudziale zainteresowanych zmianą.

[2] Praktyka dużych innowacyjnych projektów (szczególnie projektów dla Armii Amerykańskiej) pokazuje jednak, że w klasie pewnych problemów tradycyjne metodyki radzą sobie z oszacowywaniem ilości pracy równie źle jak można spodziewać się tego po metodykach zwinnych - budżety przekraczane są wielokrotnie, czas wykonania przedłuża się proporcjonalnie.

# Rozdział 3

# SCRUM - wprowadzenie

SCRUM, znaczy młyn - nazwa metodyki wywodzi się od określenia stosowanego w grze w rugby. Jak w każdej metodologii zwinnej, projekt dzieli się w czasie na iteracje - tu nazywane sprintami, sprinty trwają od 2 do 6 tygodni[1] i powinny być możliwie tej samej długości w danym projekcie[2]. Sprinty składają się na wydania (ang. "release"), które mogą komponować od jednego do nawet kilkudziesięciu sprintów, zwykle jednak wartość ta oscyluje między 3 a 12. Wydania nie muszą mieć od początku ustalonych: dokładnej daty zakończenia ani dokładnego zakresu. Jeżeli wydanie ma zakres doprecyzowany w drobnych szczegółach od samego początku, to najpewniej oznacza, że wykonujemy coś źle, niezgodnie z duchem agile, najpewniej zakładamy wtedy, że w projekcie nie będą następować zmiany, lub że rola klienta kończy się na podpisaniu kontraktu, a nade wszystko nie dokonujemy wszelkich starań dla osiągnięcia celu nadrzędnego, czyli zadowolenia klienta z końcowego produktu (nawet jeżeli to klient nalega na dokładny kontrakt uważając, że dokładnie wie, czego potrzebuje).

Metodyka SCRUM opiera się silnie na relacjach samoistnie tworzących się między członkami zespołu. Zakłada się, że zespół będzie dokonywał pewnej sa-

[1] Przychylam się do tezy, że jeżeli tylko jest taka możliwość - sprinty powinny jak najbardziej zbliżać się do poziomu 2 a nie 6 tygodni.

[2] Niekiedy stosuje się dłuższe sprinty na początku - często w zredukowanym zespole - po to, by wykonać czynności niezbędne do właściwego startu projektu (czasami w tym celu otwiera się wręcz osobny projekt). Czasami również wydłuża się ostatni sprint w wydaniu, by zagwarantować zapas czasu na przygotowanie finalnej wersji oprogramowania - szczególnie gdy sprinty w czasie projektu były krótkie.

moorganizacji i jest to rzecz, która silnie wpływa na wydajność pracy w metodyce (patrz [6]). Nie ma wyróżnionych wielu ról czy hierarchii. Jedyne role, jakie SCRUM określa to:

1. Scrum master (mistrz młyna) - osoba dbająca o proces i rozwiązująca problemy zewnętrzne zespołu. Mistrz młyna odpowiedzialny jest też za to, by proces był ciągle udoskonalany.

2. Product owner (właściciel produktu) - osoba która decyduje o tym co ma być zrobione przez zespół w danym sprincie oraz zapewniająca, że w danym sprincie zestaw ten nie ulegnie zmianom. Do obowiązków właściciela produktu należy również na przykład ustalanie daty wydania, czy dbanie o autonomię pracy zespołu.

3. The team (członkowie zespołu) - osoby o zróżnicowanych kompetencjach które wykonują pracę nad projektem. Zespół decyduje, ile jest w stanie zrobić w danym sprincie. Decyduje również decyduje o tym, jak dana rzecz ma być zrobiona (jaką techniką, z użyciem jakich narzędzi), o ile to nie zależy ściśle od interesu biznesowego właściciela produktu.

4. Others (wszyscy inni) - wszelkie inne osoby, wchodzą w to przyszli użytkownicy, zarząd firmy, członkowie innych projektów itp. Ważnym jest, że to nie oni wykonują dany projekt - tak więc nie mogą ingerować w proces jego wykonywania bezpośrednio. Rolą mistrza młyna i właściciela produktu jest zagwarantowanie tego warunku. Z punktu widzenia metodyki -nie ma różnych innych - prezes jest tak samo innym dla procesu jak sprzątaczka.

Role 1-3 nazywa się “świnkami” a 4 - “kurczakami”[3]. Brak wyróżniania ról posuwa się do tego stopnia, że zdecydowanie zaleca się unikania odgórnego nadawania ról wykonawczych w zespole - jak architekt, programista baz danych czy specjalista od interfejsów użytkownika. Oczywistym jest fakt posiadania przez poszczególnych członków zespołu odmiennych kompetencji, jednak nie powinny

[3]Określenia te pochodzą od historyjki o kurczaku i świni chcących założyć restaurację. Gdy kurczak przedstawiał świni propozycję założenia knajpy, ta zapytała się “a jak ją nazwiemy?”, w odpowiedzi usłyszała “jajka na boczku”, świnia skwitowała rozmowę tymi słowami: “O nie! Ja się poświęcę, a ty będziesz jedynie zaangażowany!”.

one przesądzać o tym, co członkowie mogą zrobić - a w zasadzie - czego mogą chcieć się podjąć zrobić, bo w scrumie nie występuje (przynajmniej nie powinna) nigdy sytuacja nakazania wykonania czegoś komuś. W ten sposób doszliśmy jakby "z powrotem"[4] do ostatniego z podstawowych konceptów metodyki, czyli codziennego spotkania zwanego właśnie daily scrumem, czyli młynem [5]. To ono jest centralną częścią zwykłego dnia pracy. Na takim spotkaniu, które odbywa się codziennie o tej samej porze (zwykle rano), w tym samym miejscu i do tego na stojąco, w ciągu maksymalnie 15 minut członkowie zespołu raportują wykonywane prace oraz zgłaszają się do kolejnych zadań.

[4]Patrz pierwsze zdanie tego akapitu.

[5]Swoją drogą, w firmie, w której pracuję gdy wprowadzałem podstawową wiedzę na temat tej metodologii, prezes stwierdził: "Scrum, znaczy młyn, tak? Czyli tobie na wizytówce w sumie powinniśmy wpisać 'młynarz' ". Mnie nie przeszkadzałoby takie określenie, szczególnie, że mogłoby ułatwić kontakty na konferencjach, gdzie panuje raczej koleżeńska atmosfera. Jednak osoba odpowiedzialna za HR w firmie była zdecydowanie niekontenta i pomysł niestety przepadł, przypisano mi daleko bardziej tradycyjną łatkę koordynatora projektu.

# Rozdział 4

# Opisywanie wymagań w scrumie

Opisywanie wymagań to prawdopodobnie najważniejsza część dowolnego projektu informatycznego - wynika to z faktu, że błędy popełnione na tym etapie są najkosztowniejsze w usuwaniu. Odmiennie niż na przykład w RUP - w scrumie nie ma zdefiniowanych rozległych reguł odnośnie tego zagadnienia. Twórcy metodyki wyszli z założenia, że skomplikowane metody tworzenia wymagań są nieskuteczne i przede wszystkim nieprzejrzyste, a przez to ich efekty - trudne do analizowania. Całość zagadnienia jest w scrumie objęta przez user stories (sekcja 4.1) dzielące się na konkretne techniczne zadania, epiki (sekcja 4.2), które są większymi słabo zdefiniowanymi zadaniami oraz backlogi, które są po prostu kontenerami na dwa wcześniejsze artefakty.

## 4.1 User story

Podstawowy cykl życia user story:

1. User story jest pisane przez product ownera.
2. User story jest wstępnie estymowane przez team.
3. User story jest umieszczane w zbiorze “wymagania potencjalnie przeznaczone do implementacji”.
4. User story jest rozpatrywane jako wymaganie do implementacji.
5. User story jest definiowane aż do momentu, gdy wszyscy uznają, że mają pełen obraz zadania, który jest spójny.
6. User story jest przeznaczane do realizacji - zespół zobowiązuje się je wykonać w konkretnym sprincie.
7. User story jest wykonywane przez zespół.
8. User story jest akceptowane przez product ownera jako wykonane.

Spis ten jakkolwiek pobieżny i być może nie do końca zrozumiały, jednak daje dobry ogólny obraz tego złożonego artefaktu.

User stories są podstawowym typem zadań wykonywanych przez zespół. Każde user story powinno być kompletną funkcjonalnością, która będzie z punktu widzenia product ownera przydatna sama w sobie - to znaczy, że nie kwalifikuje się do tego miana na przykład “refaktoring struktury bazy danych”, ale już “zwiększenie responsywności systemu do 10ms przy 1000 użytkowników” jak najbardziej. Podstawowa konstrukcja user story jest znacząco prostsza niż bardzo popularny use-case z RUP, ma ona postać:

<aktor>
chce/potrzebuje <opis wykonywanej czynności>
po to by/ponieważ <powód>

Tak prosta konstrukcja pozwala na tworzenie ich przez właściciela produktu szybko, nie zajmując mu czasu potrzebnego na inne zadania, jednocześnie taka

maksymalnie kilku zdaniowa konstrukcja pozwala szybko zapoznać się co ma być wykonane. Zauważmy, że częścią tej krótkiej definicji jest „powód“, czyli wyjaśnienie, po co użytkownikowi dana funkcjonalność. Jest to element niezbędny do tego, by czytający jak i (przede wszystkim) piszący mogli się wczuć w rolę użytkownika i lepiej zrozumieć wymaganie. W inny sposób zaprojektujemy ekran wpisania danych do korespondencji, jeżeli chcemy by użytkownik wypełniał go w celu bycia poinformowanym o zmianach w rozkładzie lotów (użytkownik musi mieć wtedy możliwość wyboru różnych form kontaktu, czy też podania dwóch telefonów), a inaczej, jeżeli wypełnia go po to, byśmy mogli mu wysyłać informacje o promocjach (forma musi by wtedy oszczędna, nienarzucająca się i minimalnie kłopotliwa w wypełnieniu).

Kolejnym niezbędnym elementem user story jest estymata wysiłku potrzebnego do wykonania zadania/trudności zadania. Ważnym jest, że taką estymatę musi wykonać zespół - a nie zlecający zadanie właściciel produktu. Wstępne oszacowanie można wykonać nawet w wolnej chwili[1], natomiast bardziej wnikliwe, wykonuje się przy na początku sprintu podejmowaniu się zadania. Jednostką miary trudności zadania są “story points”, abstrakcyjna jednostka, która nie odzwierciedla ani osobogodzin potrzebnych do wykonania, ani też nie jest porównywalna pomiędzy zespołami. Dzięki takiemu systemowi nie mamy problemu chociażby z poczuciem rozliczania z godzin poświęconych na zadanie przez zespół - zespół wie, że średnio na sprint (sekcja 5.1) jest w stanie wykonać na przykład 20 punktów, jednak zadanie za 3 punkty może okazać się czasochłonne choć proste w wykonaniu, natomiast zadanie za 7 - wymagające krótkiego, ale bardzo dużego wysiłku i umiejętności, które posiada tylko jeden lub dwóch, silniej obciążonych członków zespołu. Koniec końców zespół wykonuje określoną liczbę punktów a nie zadania zaplanowane na dokładnie określony czas.

[1] Aby usprawnić proces tworzenia backlogu produktu i spotkania planistyczne sprintu, co tydzień w czwartek biorę jednego lub dwóch członków zespołu i wraz z właścicielem produktu estymujemy szybko 2 do 5. zadań (tych, które są najwyżej w hierarchii). Proces taki zajmuje nam średnio 10 minut a skrócił czas potrzebny na spotkanie planistyczne o godzinę, co daje 20 minut zysku na tydzień - niby niewiele, ale dodatkowym zyskiem jest to, że spotkanie planistyczne, na którym wszyscy muszą być skupieni jednocześnie jest znacząco krótsze. Po roku usprawniania procesu wstępnych estymat planowania sprintu rzadko przekraczają 50% ramy czasowej na nie przeznaczonej - natomiast estymaty wstępne wraz ze wspólna rozmową nad wymaganiami powiększyły się do godziny-dwóch na sprint - przez co znacząco wzrosła jakość estymat.

Podstawową wadą story pointów jest fakt ich niewielkiej dokładności w pierwszych sprintach zespołu oraz wysokie prawdopodobieństwo tego, że członkowie zespołu będą mieli trudność w przestawieniu się z estymacji czasowych (zwykle wysoce nieskutecznych) na punktowe. Jeżeli okazuje się, że zespół uznaje zadanie za niewykonalne w czasie jednego sprintu - zadanie trzeba rozdzielić na oddzielne części (pamiętając o tym, że każda z nich musi być potencjalnie wdrażalną funkcjonalnością!). Może się zdarzyć, że zadanie będzie tak duże, że zespół nie jest w stanie w ogóle wyestymować jego złożoności - wtedy takie zadanie nie jest user story, a epikiem(sekcja 4.2), który zostanie podzielony na wiele mniejszych user stories.

Jako, że każda user story musi być przydane z punktu widzenia product ownera, musi on tę przydatność jakoś wyrazić. Narzędzie do tego nazywa się business value i jest to kolejna jednostka punktowa bez przełożenia na nic innego niż subiektywna ocena product ownera (i nie jest to wadą!), oczywiście w duchu uzyskania jak najwartościowszego produktu. Jest to niejako waluta[2] jaką organizacja zarobi jeżeli sprzeda daną funkcjonalność. Nie ma górnej granicy wartości biznesowej, nie ma też ustalonych sposobów jej przyznawania - jedyna zasada jaką trzeba zachować, to ta, że funkcjonalności lepiej sprzedawalne, ważniejsze dla produktu mają wyższa wartość niż te mniej ważne.

Kolejna ważna rzecz o której należy pamiętać, to konieczność testowalności każdego user story - nie oznacza to, że do każdego wymagania należy od razu pisać testy akceptacyjne (choć oczywiście jest to niezmiernie dobrą praktyką - lecz wykraczająca poza zakres omawianej właśnie metodyki), jednak każde zapisane wymaganie (zapisane a nie przyjęte do realizacji !) musi spełniać warunek: “Da się do niego napisać jednoznaczne testy.”. Umiejętność decydowania czy do czegoś da się napisać testy przed właściwym napisaniem testów jest kluczowa dla sprawnego definiowania user stories przez product ownera - jeżeli będzie on pisał testy od razu do każdego user story, to zajmie mu to bardzo dużo czasu (którego zwykle nie ma - więc pisać ich nie będzie), z drugiej zaś strony, jeżeli nie będzie na ten warunek w ogóle zwracał uwagi - wymagania będą często okazy-

[2]Product owner w projekcie którym kierowałem już przez długi czas (ponad miesiąc) nie był w stanie zrozumieć czym jest business value. Dopiero porównanie jej do pieniędzy jakie warte są poszczególne wymagania, pozwoliło mu rozumieć i prawidłowo tę miarę stosować.

wały się niemożliwe do uczciwego wyestymowania. W drugim przypadku może się zdarzyć jeszcze rzecz gorsza: zadania będą przyjmowane przez zespół bez dokładnego zrozumienia a później wykonywane niezgodnie z wizją właściciela produktu (czemu będzie winien on właśnie, ponieważ zapewnienie jasności wymagań leży w jego odpowiedzialnościach). Oczywiście można również podczas odbioru wykonanego zadania przymykać oko na niedociągnięcia, ale chyba nie o to chodzi. Jeżeli product owner należy do osób które nie potrafią ocenić kryterium testowalności - dobrze jest gdy pomagają mu w tym członkowie zespołu.

## 4.2 Epic

Podstawowy cykl życia epika:

1. Product owner definiuje user story.
2. Ktoś z zespołu stwierdza, że user story jest zbyt duże, by wyestymować je realnie na jeden sprint.
3. Właściciel produktu zamienia user story na epika - często pomaga mu w tym scrum master lub team.
4. Tworzone są mniejsze user story, których łączny zakres odpowiada całemu epikowi.
5. User stories przechodzą swój cykl życia.
6. Epic uznaje się za wykonany; alternatywą jest usunięcie go już po punkcie czwartym.

Zadania zbyt duże, by dało się je wyestymować lub estymowane na wiele sprintów, nie mogą być bezpośrednio podjęte przez zespół, zadania takie jednak w procesie niechybnie będą się pojawiać, gdyż w oparciu o nie łatwiej jest planować strategię długofalową, czyli release(sekcja 5.3). Tak naprawdę jedynym czynnikiem, który odróżnia epika od zwykłego user story(sekcja 4.1) jest niemożność wyestymowania tego pierwszego przez zespół. Tak więc jego konstrukcja i cele są takie same - wykonanie zadania powinno nadal generować przydatną funkcjo-

nalność i wartość biznesową. Przez daleko większy zakres jaki mogą obejmować epiki - często bardzo dyskusyjnym jest kryterium testowalności - epik sam w sobie z pewnością musi być możliwy do przedstawienia w formie jasnej wizji, ale już same testy akceptacyjne pisze się raczej do konkretnych user stories, które powstają z jego podziału.

Dobrym przykładem epika może być: “Użytkownik chce mieć możliwość zarządzania reklamami pojawiającymi się w serwisie, aby uzyskiwać przychody od reklamodawców” - załóżmy, że w kontekście tworzenia systemu cms. Takie wymaganie jest możliwe do wyobrażenia sobie przez team, czy product ownera, jednak samo w sobie może okazać się zbyt duże na jeden sprint i powinno być podzielone na zadania mniejsze. Przy podziale zadań wyłaniają się dokładniejsze specyfikacje poszczególnych ich części wynikowych i może się okazać, że wizja opisana na początku, zostaje wdrożona tylko w pewnym stopniu a inne elementy początkowo nieprzewidziane są dokładane w ich miejsce - nie jest to nic niebezpiecznego ani nadzwyczajnego.

Innym sposobem wykorzystania epika jest zastosowanie go jak węzełka na palcu - wstawienie go do wydania jedynie po to by wiedzieć, że czymś czego jeszcze nie możemy dokładnie zdefiniować musimy się jednak zająć. Warto jednak możliwie szybko zamienić go na mniejsze, lepiej zdefiniowane epiki i user stories, a nasz węzełek usunąć - na dłuższą metę wprowadza on po prostu „białą plamę na naszej mapie do celu“.

## 4.3 Product backlog

Cykl życia product backloga:

1. Właściciel produktu tworzy wymagania (epiki i user stories) z, których chciałby by składał się produkt.
2. Następuje priorytetyzacja backlogu.
3. Backlog jest przedstawiany na spotkaniu planistycznym sprintu.
4. Elementy backlogu przyjmowane są do wykonania.
5. Elementy wykonane usuwa się z backlogu produktu.
6. powrót do kroku 1.

Backlog produktu jest to spriorytetyzowany spis wszystkich user stories i epików, jakie aktualnie są wymyślone i spisane przez product ownera, które zamierza przeznaczyć do wdrożenia. Z backlogu wybiera się zadania do wykonania w konkretnych sprintach - zaczynając od tych o najwyższym priorytecie.

Są dwie szkoły priorytetyzowania backlogu - jedna mówi o priorytetyzowaniu każdej pozycji innym priorytetem - to znaczy w formę wektora, druga mówi o priorytetyzowaniu poprzez dzielenie całości na zbiory o różnej ważności. Sposób ze zbiorami ma tę zaletę nad wektorowym, że priorytetyzowanie przeprowadza się znacząco szybciej (spogląda się na element i po prostu kieruje się do jednego ze zbiorów), poza tym product ownerom, którzy przyzwyczajeni są do priorytetyzowania zadań w ten sposób, może być niezmiernie trudno przestawić się na priorytetyzowanie wektorowe.

Zaletą sposobu wektorowego jest unikanie najgorszej bolączki priorytetyzowania przez zbiory - 40-stu zadań o najwyższym priorytecie - w sposobie wektorowym zadanie o najwyższym priorytecie jest jedno. Problem może jednak wynikać z nieufności właściciela produktu - jeżeli uzna on, że wybieranie zadań o niskim priorytecie (nie najwyższym) jest sposobem zespołu na uznanie pewnych zadań za nieważne. Jednym z rozwiązań takiego problemu jest przedstawienie właścicielowi produktu priorytetów nie jako “które zadania są najważniejsze” a

"które zadania chcesz by były wykonywane wcześniej". Posiadanie w backlogu produktu elementów na 2-4 kolejne sprinty, w tym około połowę wyestymowaną w miarę dokładnie przez cały zespół a resztę pobieżnie w małych podzespołach wydaje się być stanem optymalnym - sprawia to, że członkowie zespołu widzą najbliższą, najbardziej prawdopodobną perspektywę, a jednocześnie nie ma dużej presji przeciw wprowadzaniu zmian w strategii rozwoju oprogramowania w kolejnych sprintach.

### 4.3.1 Release i Sprint backlog.

Identyczną formę przybierają zbiory zadań odpowiednio dla: release i sprintu - nazywamy je odpowiednio backlogiem sprintu i backlogiem releasa. Pewnym elementem który odróżnia backlog sprintu od pozostałych jest jednak to, że ustala się go raz i już nie zmienia - wynika to z faktu, niezmienności wymagań dotyczących sprintu w czasie sprintu. Z drugiej strony backlog wydania jest silnie zmienny - choć nie rośnie wszerz tak bardzo jak product backlog - raczej z czasem konkretyzuje się coraz bardziej a elementy niewykonywalne w danym wydaniu są z niego usuwane, nadal pozostając w product backlogu [3].

### 4.3.2 Backlog elementów wydanych.

Zbiór zadań już wykonanych - terminu używa się rzadko, a sam artefakt mimo pozornie wysokiej funkcji informacyjnej (pokazuje ile już mamy zrobione), tak na prawdę jest mało przydatny, ponieważ dużo ważniejsza jest informacja "ile jeszcze należy zrobić". Backlog rzeczy wydanych jest natomiast dobrym narzędziem do prezentowania kierownikom wyższego szczebla jakie efekty przyniósł projekt.

[3] Warto zauważyć, że przy zastosowaniu oprogramowania wspomagającego prace w scrumie (jak Atlassian Greenhopper lub IBM Rational Jazz) - sprint i release backlogi w zasadzie tworzą się same i nie trzeba angażować w nie czasu właściciela produktu w dodatkowy sposób - priorytety ustawione w głównym backlogu automatycznie odwzorowywane są w "małych" backlogach.

# Rozdział 5

# Iteracje w scrumie

## 5.1 Sprint

Cykl życia sprintu:

1. Przeprowadzana jest pierwsza część sprint planning meeting na którym określa się które user stories będą wykonywane w danym sprincie[a].
2. Przeprowadzana jest część druga sprint planningu, na którym dzieli się user stories na konkretne zadania do wykonania - w razie czego modyfikuje się zakres sprintu.
3. Zakres sprintu jest mrożony - od tej chwili nie można już go zmieniać.
4. Wykonywana jest codzienna praca zespołu, odbywają się daily scrumy.
5. Przeprowadza się sprint review, product owner ocenia, czy zadania zostały wykonane, niezaakceptowane zadania wracają do backlogu produktu.
6. Przeprowadza się sprint retrospection - wykrywa się problemy w procesie, które utrudniają wydajniejszą pracę i opracowuje się metody zaradcze.

[a] Jeszcze przed rozpoczęciem sprintu właściciel produktu powinien mieć przygotowaną odpowiednią ilość wymagań!

Sprint jest iteracją znaną z innych metodologii. Trwa on od 2. do 6. tygodni. W każdym sprincie wykonuje się określoną liczbę user stories, zależną od wydol-

ności wykonawczej teamu obliczanej na podstawie danych historycznych. Każdy sprint musi być tej samej długości - jeżeli na początku zdecydujemy się na sprinty na przykład trzytygodniowe a po pewnym czasie stwierdzimy, że zespół nie jest w stanie wykonywać zadań na czas - powinniśmy zająć się dokładniejszą granulacją zadań, gdyż wydłużenie czasu sprintu przyniesie brak efektu, lub efekt odwrotny do zamierzonego[1].

## 5.2 Spotkania w czasie sprintu

Jako że scrum jest metodologią lekką - nie ma w niej opisanych wielu spotkań z zarządem, walnego zebrania interesariuszy itp. W ciągu każdego sprintu jednak kilka spotkań odbyć się musi. Ich wspólną cechą jest to, że mają z góry określony maksymalny przedział czasu, w jakim muszą się zmieścić - to uniemożliwia przeciąganie ich niepotrzebnie, a zmusza do szybkiego i ścisłego wypełnienia ich celu.

### 5.2.1 Sprint planning meeting

Sprint planning meeting przeprowadza się na początku każdego sprintu, dzieli się ono na dwie części - obie powinny trwać po godzinie na każdy tydzień trwania sprintu - tak więc dla sprintu czterotygodniowego każde z nich powinno trwać 4 godziny (razem 8). Oczywiście limity godzin nie muszą być wykorzystane - to, że planowanie może zająć maksymalnie 3 godziny nie oznacza, ze nie można go skończyć w 2. Na obydwu spotkaniach obecny musi być scrum master i cały team, w czasie pierwszej części product owner musi być obecny, w czasie drugiej osiągalny.

[1] W firmie, w której pracuję, za zgodą zarządu (spowodowaną między innymi niepełnym zrozumieniem powodów [również przeze mnie], dla których sprinty muszą być zawsze tej samej długości) postanowiliśmy przeprowadzić eksperyment polegający na wydłużeniu sprintu, który normalnie trwał dwa tygodnie do trzech, gdyż zespół regularnie na podsumowaniu sprintu nie dostarczał rozwiązania przetestowanego. Efektem tego eksperymentu było całkowite rozbicie umiejętności estymowania pracochłonności zadań przez zespół (co jest związane bezpośrednio z faktem, że story points nie wyrażają bezpośrednio czasu potrzebnego na dane zadanie!).Wynikiem tego na koniec sprintu była niemożność zaprezentowania nawet nieprzetestowanej wersji któregokolwiek z przyjętych zadań! Jedynym pozytywnym efektem było to, że w kolejnych sprintach zespół więcej uwagi przykładał do estymowania zadań, jednak z pewnością można było to osiągnąć bez straty trzech tygodni pracy zespołu.

Pierwsza część spotkania służy określeniu tego co będzie w danym sprincie wykonywane - zaczyna się od przedstawienia przez product ownera aktualnego stanu product backlogu. Kolejnym krokiem jest wybranie przez product ownera pierwszego user story, jaki chciałby, aby wykonano w danym sprincie. Team precyzyjnie ustala z product ownerem co należy wykonać aby zadanie było uznane za wykonane - fachowo mówiąc - ustalają intersubiektywną wizję wykonania zadania. Rolą scrum mastera na tym etapie jest zapewnienie tego, że team i product owner wzajemnie się rozumieją - jeżeli ma on odczucie, że zrozumienie jest niepełne - powinien niezwłocznie interweniować. Gdy wszyscy już rozumieją co trzeba wykonać, dokonuje się estymacji zadania w story pointach - team musi dojść do zgody co do złożoności zadania. Jeżeli okaże się, że zadanie jest zbyt duże, by można było wykonać je w pojedynczym sprincie - staje się ono automatycznie epikiem i albo od razu wydziela się z niego mniejsze user story, albo zostawia się je (gdy dzielenie go nie jest proste i product owner potrzebuje na to więcej czasu) i przechodzi niezwłocznie do kolejnego zadania. Wyestymowane zadanie przenosi się do backlogu sprintu (patrz 4.3.1).

Proces ten powtarza się dokąd suma story pointów jest mniejsza od wydajności zespołu (sekcja 7.1.2) na sprint. Jeżeli product owner jest usatysfakcjonowany otrzymaną listą funkcjonalności, które będą wykonywane w danym sprincie - kończy się część pierwsza - od tej chwili nie można juz nic dołożyć do zadań na dany sprint. Jeżeli natomiast product owner nie jest zadowolony z listy zadań - musi jakieś z niej usunąć by móc wstawić inne. Dzięki temu procesowi - product owner decyduje o tym co będzie wykonane - a zespół, ile będzie wykonane.

Druga część spotkania służy wnikliwemu przeanalizowaniu wybranych zadań i podzieleniu ich na konkretne zadania techniczne i odkrywcze. Mamy już gotową listę user stories na dany sprint (z pierwszej części spotkania), analizujemy zadania zaczynając od tych, które wydają się najbardziej ryzykowne (te, z którymi był największy problem przy estymowaniu). Zadania techniczne powinny być dobierane tak, by jedna para programistów (grafików, specjalistów od interfejsów użytkownika itp.) była w stanie je wykonać w ciągu jednego dnia - raczej należy unikać zadań mniejszych. Natomiast zadania większe powinny pojawiać się jedynie jeśli zebrani widzą, że inaczej nie zdążą omówić wszystkich user stories które sprawiały problemy przy estymacji. Nie wszystkie user stories muszą być rozpla-

nowane dokładnie - może nawet zdarzyć się, że jakieś user stories zostaną uznane za takie, które nie budzą żadnych wątpliwości. Samo dzielenie na zadania powinno być żywą dyskusją wszystkich obecnych, połączoną z częstymi rysunkami na białych tablicach czy flipchartach - jeżeli spotkanie wydaje się nudne i jedna osoba ciągle proponuje rozwiązania które inni bezapelacyjnie przyjmują, oznacza to, że team albo potrzebuje przerwy, albo jakiegoś rodzaju stymulacji przez scrum mastera.

Poza oczywistymi korzyściami płynącymi z wydzielania konkretnych czynności do wykonania, spotkanie przynosi dwie wartości dodane: po pierwsze istnieje możliwość skorygowania estymacji i ewentualnego zmniejszenia zakresu sprintu, po drugie - dzięki wkładowi product ownera - dokładniej precyzuje się wymagania. Product owner może (i powinien) w każdym momencie wyrażać swoje uwagi co do tego co chce otrzymać (szczególnie, gdy widzi, że zespół źle go zrozumiał), nigdy natomiast nie wolno mu wkroczyć na teren odpowiedzialności zespołu czyli "w jaki sposób spełnić wymagania" - nadzorcą nad tą niejednokrotnie bardzo cienką granicą (tym cieńszą im większą wiedzę z dziedziny wytwarzania produktu posiada product owner!) jest oczywiście scrum master.

Między obydwoma częściami spotkania należy zawsze robić przerwę[2], w jej czasie warto coś zjeść.

Ze względu na to, że podczas spotkań może zajść potrzeba znaczącego moderowania dyskusji (szczególnie przestrzegania kompetencji) - scrum master powinien o takiej ewentualności powiadomić product ownera, jak i team oraz wyjaśnić

[2]Moim zdaniem najlepiej po pierwszym z nich zakończyć dzień pracy lub zająć się rzeczami do zrobienia "w tak zwanym międzyczasie" - rzeczy takich zawsze się zbiera sporo, więc raczej nie powinno być z tym problemu. Może się wydawać, że dobrym pomysłem jest wykonanie obydwu spotkań w jeden dzień - jedno po drugim, tak by "tracić" na nie jeden dzień. Jest to jednak bardzo zły pomysł, dla dowolnego przypadku w którym spotkania mają mieć długość większą niż 4 godziny łącznie z przerwą między nimi (czyli nawet dla sprintów 2 tygodniowych sprintów, jeżeli mistrz młyna wie, że nie zawsze udaje się spotkania skrócić). Obydwa spotkania potrafią być stresujące oraz są wyjątkowo wyczerpujące poznawczo, szczególnie dla mistrza młyna i właściciela produktu - nawet jeżeli inni członkowie zespołu stwierdzą, że "mogą dalej" - oni dwaj będą tak wyczerpani intelektualnie, że drugie spotkanie będzie słabej jakości. Kolejną zaleta robienia spotkań dzień po dniu, jest danie czasu wszystkim na ogarnięcie w myślach zadań które trzeba wykonać - dzięki temu druga część idzie dużo sprawniej, bo nie traci się czasu na oswojenie się z nowymi user stories. Po roku pracy w danym zespole, miedzy innymi dzięki większej ilości wkładanej w definiowanie wymagań pracy przez cały zespół - spotkania idą już tak szybko, że czasami nawet rezygnujemy z długiej przerwy między nimi (zastępując ją przerwa na herbatę lub żabkę), aby w czasie obiadu obgadać co będziemy robili danego dnia.

im, że wszelkie próby ograniczenia wypowiedzi są tylko i wyłącznie podporządkowane sprawnemu przebiegowi procesu i nie powinny być odbierane osobiście.

### 5.2.2 Sprint review

Spotkanie to przeprowadza się na koniec sprintu i służy głównie ocenie które zadania udało się wykonać a które nie. Rozpoczęcie spotkania oznacza definitywny koniec prac nad czymkolwiek w danym sprincie - a dzięki znanemu od początku niezbyt odległemu terminowi - uzyskuje się ciągły efekt nieuchronnego terminu oddania prac, co dla większości osób jest bardzo pomocne w utrzymaniu wysokiego tempa pracy. Spotkanie to powinno mieć nieformalny charakter - w szczególności pomocna jest zasada nieprezentowania dokonań na slajdach.

Prezentowane powinny być gotowe produkty - nie można pokazywać produktów nieskończonych (za [3, 2]). Oprogramowanie powinno być pokazywane w środowisku możliwie zbliżonym do produkcyjnego - najlepsze do tego są serwery działu zapewniania jakości. Motywatorem dla teamu może być to, że na spotkanie podsumowujące może przyjść nie tylko product owner (który musi być obecny) ale także interesariusze czy członkowie innych zespołów [3].

Spotkanie to nie ma ustalonego kształtu - zwykle powinno zacząć się od przedstawienia głównego celu sprintu przez któregoś z członków zespołu, następnie poszczególni członkowie prezentować powinni niuanse działania oprogramowania i wykorzystane rozwiązania. Na podsumowaniu nie powinno przedstawiać się artefaktów innych niż gotowe produkty - dopuszczalne jest to jedynie w celu lepszego zilustrowania działania oprogramowania i powodów podjętych decyzji projektowych. W żadnym wypadku nie może się zdarzyć sytuacja, w której prezentując efekty pracy, próbuje się zmusić słuchaczy do zrozumienia procesu tworzenia oprogramowania.

Po prezentacji każdego z obecnych prosi się o opinie - co należy w ich ocenie zmienić, jakie funkcjonalności dodać, co się podobało. Jest to jedyne spotkanie na którym „kurczaki“ mają prawo głosu - jednak nie prawo do podejmowania decyzji o drodze rozwoju projektu. Ich wypowiedzi muszą być informacją wejściową dla

[3] Nawet jeżeli organizacja, w której przeprowadzany jest scrum jest niewielka - raczej nie zdarzy się sytuacja, że wszystkie osoby będą wiedziały czym na bieżąco zajmuje się zespół.

product ownera, który na ich podstawie przedefiniowuje product backlog - dodaje nowe elementy i przestawia priorytety starych.

Całość podsumowania jest ograniczona do godziny na każdy tydzień trwania sprintu - w tym czasie team musi zaprezentować produkt, muszą zostać zebrane informacje zwrotne od słuchaczy i wprowadzone niezbędne zmiany w backlogu.

### 5.2.3 Sprint retrospection

Retrospekcję wykonuje się po sprint review i jej celem jest doskonalenie naszego procesu scrumowego. Retrospekcja powinna zwykle trwać nieco krócej niż podsumowanie sprintu (3 godziny dla czterotygodniowego sprintu według [3]). Warto korzystać z gotowej listy punktów do przejrzenia, by móc śledzić postępy.

Jest to najmniej formalne ze wszystkich spotkań w całej metodyce - w zasadzie mogłoby się odbywać nawet poza miejscem pracy, jeżeli scrum master uznałby, że w ten sposób zespół lepiej (obiektywniej) będzie mógł spojrzeć na swoją pracę. Nie ma też ustalonego żadnego przebiegu (biorąc pod uwagę tylko założenia metodyki) - zazwyczaj chodzi po prostu o to, by wywiązała się dyskusja na temat procesu, w który zespół jest zaangażowany. Rolą scrum mastera na tym spotkaniu nie jest dostarczanie zespołowi gotowych rozwiązań, a moderowanie dyskusji tak, by zespół sam odnalazł najlepsze dla siebie sposoby usprawniania procesu. Najbardziej popularnym sposobem rozpoczęcia spotkania jest zadanie każdemu z członków zespołu pytań: „Co udało się zrobić dobrze w tym sprincie - o czym chcielibyśmy pamiętać ?” i „Co można ulepszyć w następnym sprincie - gdzie potrzebujemy jeszcze włożyć trochę pracy ?”. Wszystkie odpowiedzi spisuje się na tablicy i priorytetyzuje się od tych, którymi team chce się zająć najpierw, do tych, którymi chce się zająć na końcu. Przedstawiony scenariusz jest jedynie pewną wskazówką - w każdym zespole będzie inny przebieg spotkania i inne problemy będą na nim poruszane.

W wyniku tego spotkania, zawsze powinny powstać pewne ustalenia, które w kolejnym sprincie będą wprowadzane w życie. Team musi też zdecydować się na taką liczbę usprawnień, jaką jest w stanie wprowadzić - nie więcej. Widać tu analogię do samego procesu tworzenia produktu. Proces ulepszany jest w takich samych iteracjach, jasno definiuje się co należy zrobić i podejmuje się tylko zadań

możliwych do wykonania. Różnica polega na tym, że w tym przypadku właścicielem produktu nie jest jedna osoba, a cały zespół - to przecież oni pracują w danym procesie (a każdy proces scrumowy jest inny!).

Ciekawy jest pewien fakt: wśród opracowań metodyki nie ma zgodności co do składu osobowego tego spotkania - według niektórych źródeł na spotkaniu powinien być obecny cały zespół włącznie z product ownerem i scrum masterem(patrz [3]) - a według innych - właściciel produktu nie powinien się pojawiać. Ogólnie przyjętą zasadą jest jednak to, że spotkanie to jest zamknięte dla „kurczaków“. Jeżeli zespół ma szczęście posiadać product ownera, który chce przede wszystkim osiągać cele długoterminowe i rozumie, że aby to robić efektywnie potrzebne jest wiele czynników takich jak: wyszkolenie zespołu, jego dobre samopoczucie czy też właściwa specyfikacja wymagań, a jednocześnie zespół sprawia wrażenie zdyscyplinowanego - product owner będzie zdecydowanie pomocny na takim spotkaniu. Jeżeli jednak product owner nie sprawia tak dobrego wrażenia - scrum master powinien w jak najwcześniejszym momencie zapobiegać pojawianiu się go na retrospekcji - jako, że ewentualne korzyści z jego przybycia są znacząco mniejsze niż szkody, jakie może wyrządzić w czasie gdy wszyscy powinni zastanawiać się nad konstruktywnymi metodami poprawy procesu zamiast wytykać sobie wzajemnie błędy.

To, jak duże zaangażowanie wykazują członkowie zespołu podczas tego spotkania jest najlepszą miarą ich zaangażowania w projekt[4]. Jeżeli zespołowi nie zależy na uczestniczeniu w retrospekcjach i trzeba za każdym razem zmuszać ich do wyrażania swoich opinii - wtedy jest to znakiem, że należy popracować nad morale zespołu.

### 5.2.4 Daily scrum

Daily scrum jest najkrótszym spotkaniem w całej metodyce, ale też najczęściej występującym, przez wielu jest uznawany za kwintesencję scruma (nie bez po-

[4]Niekoniecznie w produkt - może się zdarzyć, że mimo iż na co dzień wydaje się, że programiści nie są zainteresowani wykonywaną przez nich pracą - tak naprawdę mają bardzo dużą motywację do wykonywania swojej pracy sumiennie i tak, by przynosiła efekty! Może się zdarzyć również sytuacja odwrotna - zespół jest bardzo zadowolony z tego ,że bawi się nową technologią, architekturą, czy ładnymi interfejsami użytkownika, ale tak na prawdę członkom zespołu nie zależy na wytworzeniu wartościowego dla właściciela produktu oprogramowania.

wodu nazwa). Trwa do 15 minut i ma bardzo prostą budowę: każdy członek zespołu po kolei mówi co zrobił dnia poprzedniego, co zamierza zrobić dzisiaj i jakie problemy przeszkadzają w posuwaniu się naprzód. Takie codzienne raportowanie sprawia, że scrum master ma ciągły obraz postępu prac w danym sprincie, poza tym zespół może sugerować rozwiązania problemów dotykających innych członków zespołu.

Pożądanym było by, żeby członkowie zespołu jeszcze przed spotkaniem przejrzeli sobie sprint backlog i zastanowili się nad tym jakie zadania warto by wzięli na warsztat. Zdarza się jednak, że któreś zadanie nie zostanie wybrane przez nikogo - zasady scruma nie mówią co robić w takim przypadku, ponieważ, zasady scruma odnoszą się do idealnej sytuacji, w której cały zespół jest codziennie zmotywowany do pracy i każdemu z jego członków przyświeca jasny cel - wykonania zadań na dany sprint na czas[5]. Jednak same projekty scrumowe odbywają się w rzeczywistym świecie i mimo, że co do zasady każdy powinien samodzielnie wybierać sobie zadanie - jeżeli jacyś członkowie teamu nie wiedzą co mają zrobić i nikt inny z teamu nie potrafi im odpowiedniego zadania wskazać - scrum master powinien im zaproponować jedno z takich zadań (wtedy albo znajdą coś bardziej pożytecznego do zrobienia albo wykonają "niechciane zadanie" - obydwa przypadki są pożądane).

Cechami charakteryzującymi dobry daily scrum są niezmienność czasu i miejsca spotkania oraz nieuczestniczenie w nim nikogo spoza zespołu scrumowego, wskazanym jest natomiast, by co najmniej na kilku spotkaniach w tygodniu pojawiał się product owner. Te zasady sprawiają, że w teamie wyrabia się nawyk - co sprzyja organizacji pracy i buduje poczucie obowiązku. Na każdym spotkaniu przecież poszczególne osoby deklarują, że coś konkretnego w jakimś określonym terminie wykonają.

Kolejne cechy to już wspomniane na początku ograniczenie czasowe do 15. minut, oraz niewspomniana jeszcze a wiążąca się z nim zasada, iż daily scrum

[5]Oczywiście do takiego stanu należy dążyć - jednak w realnym świecie nie zawsze jest to możliwe - chociażby ze względu na to, że osoba która powinna motywować (mistrz młyna) może mieć lepsze lub gorsze dni, a czasem zdarza się , że zły nastrój jednego z członków zespołu udziela się wszystkim - jest to szczególnie niebezpieczne zważywszy na niewielkie przedziały czasu, a więc i niewielkie możliwości kompensacji tych "gorszych dni".

odbywać się powinien na stojąco[6]. Dzięki temu, że spotkanie jest krótkie i z zasady trudno w nim o wyszukane formalizmy - zespoły techniczne nie odbierają go jako niepotrzebnego odciągania od pracy. 15 minut to czas, w którym bez problemu każdy może się skupić i dzięki temu minimalizuje niedoinformowanie zespołu o całokształcie sprintu, co może się zdążyć na dłuższych spotkaniach gdzie nie przez cały czas wszyscy mogą być zaangażowani.

Raporty przedstawiające postępy prac powinny być możliwie krótkie i możliwie nie mijające się z prawdą - jako ,że są one podstawową informacją uzyskiwaną przez scrum mastera odnośnie stanu projektu. Aby zwiększyć prawdopodobieństwo uzyskania rzetelnych danych - scrum master musi znaleźć kompromis między słuszną złą oceną niewykonania zadeklarowanych zadań a niedopuszczaniem do sytuacji, gdy niechęć konfrontacji sprawi, że członkowie zespołu zatajać będą niedopracowane elementy. Najgorszym przypadkiem jest sytuacja, gdy członkowie zespołu nie umieją samodzielnie prawidłowo oceniać momentu wykonania zadania. Oznacza to, że definicja wykonania nie jest dla nich dostatecznie jasna. Środkiem zaradczym jest dokładniejsze precyzowanie dowolnych zadań jakie podejmowanych przez takie jednostki. Jeżeli problem dotyczy większej części zespołu oznacza to jednoznacznie, że na sprint planningach należy poświęcić więcej uwagi na analizowanie definicji wykonania.

Zadania podejmowane przez team powinny być na tyle małe, by dało się je precyzyjnie estymować. Dobrą zasadą jest to, by zadania były możliwe do wykonania w ciągu maksymalnie 2. dni[7]. Należy zawsze dokładnie przestrzegać terminów wykonania zadań odkrywczych - gdyż te z natury rzeczy potrafią się ciągnąć bardzo długo i przez wiele dni dawać złudzenie “rozwiązania za rogiem”. Egzekwowaniem terminów zadań o określonym czasie musi zajmować się scrum master a nie przydzielona do nich osoba, co więcej, jeżeli chce uzyskać się z nich określone wyniki - zadaniami takimi należy zająć się częściej niż raz dziennie na codziennym młynie - aby osoba wykonująca zadanie miała poczucie upływającego czasu i nie musiała zakończyć pracy w połowie.

Zespół nie powinien czuć oporów przed zgłaszaniem problemów - lepiej zająć

[6]Do przekonania się o słuszności tej zasady wystarczy przeprowadzić dwa - trzy daily scrumy na siedząco i porównać ich średni czas do wykonywanych na stojąco.

[7]Jeżeli wykorzystuje się technikę programowania parami, i stosuje się to w innym celu niż tylko transfer wiedzy - zadania nie powinny przekraczać jednego dnia!

się dwa razy problemami błahymi, niż trzeci raz nie usłyszeć o problemie ważnym. W tej części (części dotyczącej rozwiązywania problemów) pracy scrum mastera przeszkodami mogą być nadmierna wola samodzielnego rozwiązywania problemów przez członków zespołu[8] przy braku umiejętności ich rozwiązywania oraz trudności w identyfikowaniu przez członków zespołu czynników przeszkadzających w pracy.

## 5.3 Release

Cykl życia wydania:

1. Product owner wyznacza pierwsze wytyczne do release - przewidywany czas zakończenia i ramowy zakres (release backlog) [a].
2. Przeprowadzane są sprinty które składają się na release.
3. W miarę upływu czasu aktualizuje się backlog i czas wypuszczenia release.
4. Kończy się release, planuje się kolejny lub mrozi się prace nad produktem.

[a] Jeżeli zespół jest nowo utworzony - siłą rzeczy nie wiadomo z jaką prędkością wykonywać będzie zadania, więc początkowe założenia najpewniej będą ulegały dużym zmianom.

Release jest największą miarą czasu, jak i wykonanej pracy przewidzianą w opisach metodyki; w wydaniu powstać musi produkt, który jest spójny i który rzeczywiście można 'wydać', tj. sprzedać, użyć w firmie zamawiającego, czy opublikować jak „naszą klasę“ i przyjmować już użytkowników, by kupowali „wirtualne kanapki“.

W wielu (w większości) opisach scruma koncept release nie jest w ogóle opisywany jako oddzielny byt - nadal występuje on wtedy w formie dostarczonego produktu, można by więc uznać, że wydzielanie go jako osobnego artefaktu jest zbędne. Wydanie to jednak bardzo dobra jednostka do wykorzystania przy wpasowywaniu projektu w tradycyjne podejście do zażądzania projektami a więc i konceptem łatwo zrozumiałym dla menadżerów wyższego szczebla, wyszkolo-

[8] Symptomy takiego przypadku, to niezgłaszanie żadnych problemów przy jednoczesnym ewidentnym “kręceniu się w kółko”.

nych i pracujących przez lata bez korzystania z technik zwinnych (wiecej w [5]). Zdefiniowanie wydania sprawia dodatkowo, że team ma jasną wizję, po co wykonuje zadania i na co powinni zwrócić dodatkowo uwagę, a product owner - po co je tworzy. Wydania częściej pomija się przy projektach wykonywanych dla zewnętrznego klienta niż przy projektach wykonywanych wewnętrznie - wynika to z tego, że projekty wewnętrzne częściej są projektami ciągłymi i same w sobie niekoniecznie muszą przedstawiać cel możliwy do ogarnięcia przez team [9]. Release pomocne są również, jeżeli zewnętrzne uwarunkowania wymuszają pewien harmonogram prac[10]. ujmując cały projekt w serię wydań, można dużo łatwiej przekonać podmioty zainteresowane tym harmonogramem, że rzeczywiście jest lub będzie realizowany.

### 5.3.1 Release planning meeting

Release planning meeting przeprowadza się na początku release i ustala się na nim ogólny plan funkcjonalności wykonywanych w danym release. Czas jaki poświęca się na to spotkanie, powinien odpowiadać mniej więcej dwóm godzinom na każdy tydzień sprintu - tj. trwać tyle ile trwają łącznie obydwie części sprint planning meeting. Zauważmy, że długość spotkania jest zależna od sprintu a nie od release - czyli od rytmu pracy zespołu a nie od złożoności zadania do wykonania - wynika to z tego, że release planning meeting powinno służyć jedynie opracowaniu spójnej wizji na najbliższy czas, która będzie intersubiektywna dla wszystkich członków zespołu. Efektem tego spotkania jest lista funkcjonalności do zaimplementowania w formie backlogu wydania oraz przewidywana data wykonania tych funkcjonalności. Obydwa artefakty nie powinny być postrzegane jako kontrakt jak w przypadku sprintu, a raczej jako wskazówka dla całego zespołu - punkt, do którego się dąży.

[9] Na przykład produkt Adobe Photoshop - rozwijany jest od ponad 20. lat - cały czas w formie rozwoju oprogramowania - ani razu w jego historii nie zdarzyło się, by kolejna wersja była pisana “od zera”. Gdyby więc nie podział na wersje i podwersje - bardzo trudno byłoby wyznaczać termin kiedy jakiś konkretny produkt jest wytworzony. Sytuacja diametralnie komplikuje się w przypadku oprogramowania webowego - tutaj nie można mówić o numerowanych wersjach - zmiany w serwisach są wprowadzane na bieżąco - więc koncepty takie jak wydania, trzeba wyznaczać w sposób niejako sztuczny - inaczej zespół nie ma żadnego konkretnego celu na horyzoncie.

[10] Przykładem niezmiernie aktualnym są projekty unijne.

Przebieg spotkania powinien być niemal izomorficzny do przebiegu pierwszej części sprint planningu - jedną ważną różnicą jest konieczność jasnego wytłumaczenia całemu zespołowi przez product ownera kontekstu wydania, celu wydania i powodów, dla których ten cel jest dla klienta wartościowy. Jest to potrzebne aby zespół był w stanie racjonalnie pomóc (pamiętajmy, że w zespole z założenia znajdują się osoby posiadające wszystkie umiejętności potrzebne do wykonania oprogramowania, a więc również analitycy !) product ownerowi wybrać (ewentualnie utworzyć nowe) zadania jakich wykonanie sprawi, że cel wdania zostanie osiągnięty.

# Rozdział 6

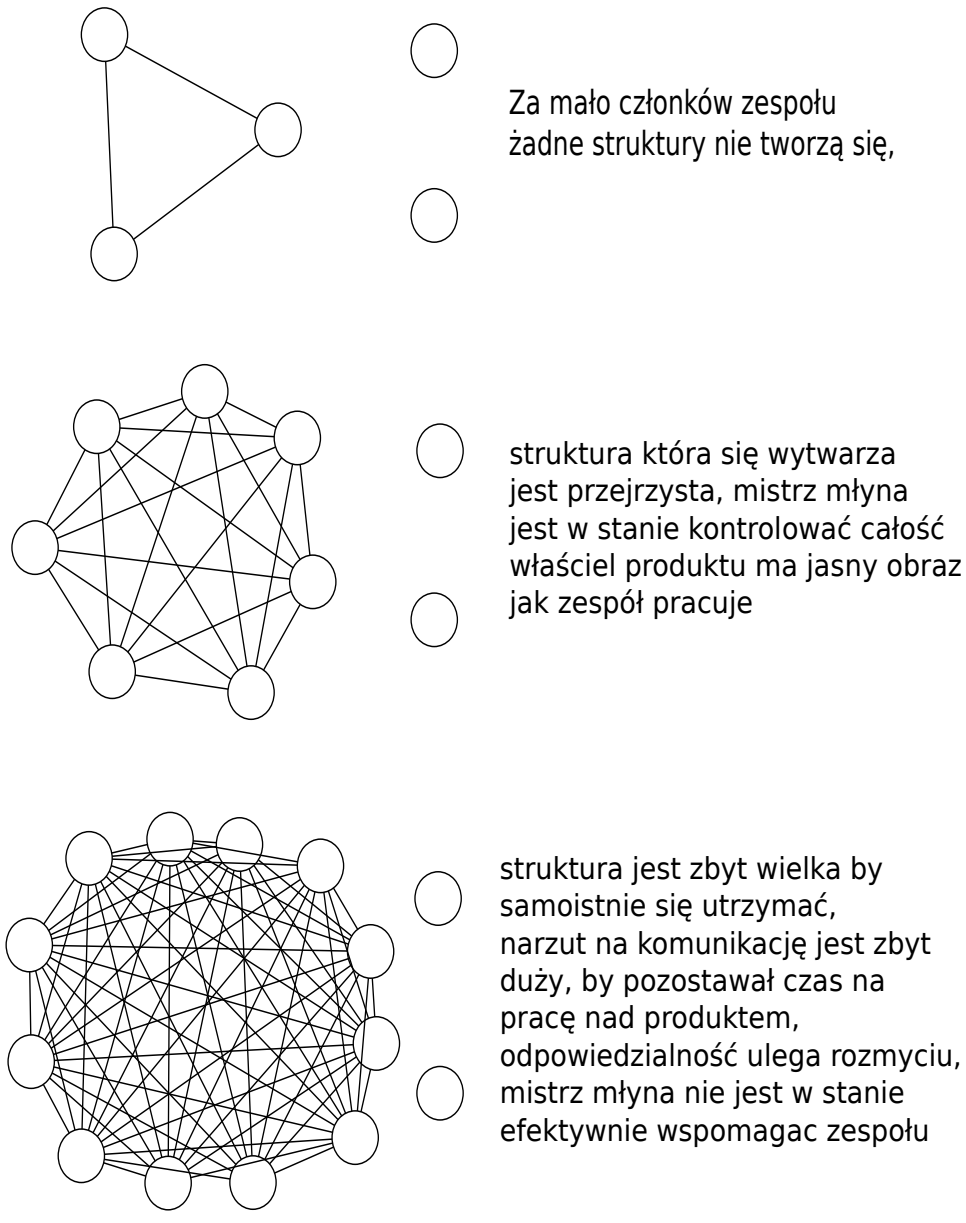
# Ludzie w scrumie

Najważniejszym czynnikiem w dowolnym projekcie są pracujący w nim ludzie.

W Scrumie nie ma zaangażowanych zbyt wielu ludzi - z zasady tylko tylu ilu jest potrzeba, co więcej - zespoły scrumowe powinny być wielkości od 5 do 9 osób (za [4]), plus scrum master i product owner. Zbyt mała liczba osób sprawia, że nie wytwarza się samoorganizacja i większość zalet metodyki nie może zaistnieć, zbyt duża natomiast sprawia, że narzut na bezpośrednią nie hierarchiczną komunikację jest zbyt wielki i pojawia się klasyczny problem mitycznego osobomiesiąca - problem zilustrowany na rysunku 6.1. [1] .

W tabeli 6.2 porównane są pokrótce trzy role metodyki - nie daje ona pełnego obrazu żadnej z nich, ale pozwala na pokazanie różnic między nimi. W szczególności ważne są różnice między scrum masterem a product ownerem oraz między odpowiedzialnościami teamu i product ownera.

[1] Jeżeli problem który trzeba w projekcie rozwiązać jest zbyt duży dla tej wielkości zespołu, stosuje się metody skalowania scruma opisane dalej.




Rysunek 6.1: Wielkość zespołu scrumowego.

| | scrum master | product owner | team |
|---|---|---|---|
| odpowiedzialności | | | |
| | • czy scrum się toczy<br>• co przeszkadza w pracy | • co ma być zrobione<br>• kto przeszkadza w pracy | • jak ma być zrobione<br>• ile może być zrobione |
| zadania | | | |
| | • moderowanie spotkań<br>• usuwanie problemów zespołu | • tworzenie product backlog<br>• osłanianie zespołu przed osobami trzecimi | • wykonywanie zadań<br>• estymowanie zadań i możliwości zespołu |

Tablica 6.2: Porównanie ról w scrumie.

## 6.1 Scrum master

Podstawowe cele pracy scrum mastera (za [3]):

- Zwiększanie produktywności zespołu w każdy możliwy sposób.
- Dbanie o proces ulepszania praktyk wytwarzania produktu, tak by każdy zbiór nowych funkcjonalności był potencjalnie wdrażalny.
- Usuwanie barier między zespołem a właścicielem produktu, tak, by właściciel produktu mógł bezpośrednio kierować wytwórstwem produktu.
- Uczenie właściciela produktu tego jak może on realizować swoje cele przez scrum.
- Zwiększanie morale zespołu poprzez kanalizację kreatywności i wprowadzania dobrych praktyk do procesu.
- Utrzymywanie informacji na temat postępów prac zespołu aktualnej i dostępnej dla wszystkich zainteresowanych.
- Przeciwdziałanie próbom nacisku przez osoby zewnętrzne, jak i przez właściciela produktu w trakcie trwania sprintu.

Wielu osobom zaczynającym naukę o scrumie scrum master kojarzy się jednoznacznie z menadżerem projektu - nie jest to do końca prawdą, gdyż rolą scrum mastera nie jest zarządzanie projektem a umożliwianie wykonywania projektu. Przykładem wspomagania a nie zarządzania jest: wspomaganie teamu w wyborze odpowiednich zadań na dany dzień a nie dyktowanie teamowi, co musi wykonać. Jeżeli członkowie zespołu wiedzą, co trzeba wykonać, to nie należy im narzucać innego toku wykonania zadań - w końcu to oni są ekspertami w swojej dziedzinie[2]. Ważnym elementem wspomagania jest oczywiście motywowanie zespołu -

[2]Sytuacja wygląda oczywiście bardziej skomplikowanie, jeżeli scrum master pełni jednocześnie rolę architekta systemu - co zdarza się często, chociażby z powodu posiadania przez architekta wysokiej pozycji społecznej w technokratycznej społeczności informatyków, i związanego z tym szacunku i swego rodzaju naturalnego uznawania za lidera - a więc cech wysoce pożądanych u scrum mastera, który ma prowadzić zespół. W takim przypadku prawdopodobne jest, że do zespołu nie będzie przydzielona inna osoba o dostatecznie wysokich kompetencjach i decyzje o tym co i jak zrobić danego dnia będzie podejmował ten sam człowiek, który jest scrum masterem - ale

tutaj nieodzowne są naturalne predyspozycje mistrza młyna, które tylko do pewnego stopnia można wyszkolić.

Kolejną czynnością jaką wykonuje scrum master jest usuwanie problemów zespołu - od najbardziej banalnych, jak np. brak kawy, do bardziej skomplikowanych, jak np. konieczność zorganizowania szkolenia z wybranej technologii. Oczywistym jest, że nie musi być tak, że scrum master ma kompetencje dokonywania zakupów w firmie - zwykle jego rola ograniczy się w tym przypadku do przekonania osoby odpowiedzialnej za zaopatrzenie, że kawa jest potrzebna i że należy ją kupić możliwie najszybciej. Problemy te raportowane powinny być na daily scrumach (sekcja 5.2.4). Część problemów może być zauważona bezpośrednio przez scrum mastera - przykładem takiego problemu może być brak umiejętności samoorganizacji teamu. Jeżeli identyfikacja takiego problemu nastąpi na początku sprintu, nie ma sensu czekać aż do retrospekcji (patrz 5.2.3) jeżeli środki zaradcze można zaproponować i wdrożyć natychmiast. Innym typem problemu może być zauważenie przez team, że błędnie oszacowano zadania do wykonania - rolą scrum mastera w takim wypadku jest omówienie sytuacji z teamem i product ownerem.

Scrum master powinien być dla product ownera (jak i dla zespołu) punktem uzyskiwania informacji na temat tego, co należy zgodnie z metodyką w danym momencie zrobić i jakie są możliwości w danej sytuacji. Wiedza ta powinna być dostarczana zarówno w przypadku bezpośredniego pytania, jak i w przypadku, gdy jakieś ważne reguły scruma będą złamane w wyniku działań podejmowanych (lub niepodejmowanych) przez product ownera. Najbardziej standardowym przykładem takiego zachowania jest sytuacja, gdy zdenerwowany product owner wchodzi do pokoju zespołu i oznajmia, że w tym sprincie muszą zrobić jeszcze system pomocy kontekstowej. Prawidłową reakcją scrum mastera powinno być poinformowanie product ownera o możliwości przerwania sprintu w sposób nieplanowany, dokonania przeglądu sprintu, retrospekcji i rozpoczęciu kolejnego.

Do obowiązków scrum mastera należy także moderowanie wszystkich spotkań przewidzianych w metodyce. Szczególnie należy tu zaznaczyć sprint retrospection (patrz 5.2.3), która jest głównym narzędziem do katalizowania ulepszania procesu. Z powodu moderowania spotkań w zespołach, które z samego zało-

jako architekt-członek zespołu a nie scrum master!

żenia składają się ze specjalistów, którzy niejednokrotnie mogą prezentować całkowicie odmienne charaktery (patrz [7]), scrum master musi cechować się silną osobowością. Jednostki słabe absolutnie nie nadają się do tej roli - nie będą w stanie zaprowadzić na spotkaniach porządku, co zaowocuje z jednej strony regularnie przekraczanymi ograniczeniami czasowymi na tych spotkaniach, z drugiej zaś strony niewypełnianiem zadań jakie te spotkania mają spełniać.

## 6.2 Product owner

Podstawowe cele pracy product ownera:

- Uzyskiwanie maksymalnego zwrotu z inwestycji w projekt - tj. uzyskiwanie efektów o najwyższej możliwej wartości biznesowej.
- Definiowanie kształtu produktu projektu i strategii jego rozwoju poprzez definiowanie funkcji do wykonania w projekcie.
- Priorytetyzowanie zadań w celu osiągnięcia jak najlepszej długofalowej wydajności zespołu.
- Ochrona członków zespołu przed bezpośrednim wpływem osób trzecich na projekt [a].
- Spełnianie celów interesariuszy związanych z projektem.

[a] również bezpośrednich interesariuszy !

Rola product ownera w przeciwieństwie do scrum mastera może osobie postronnej wydawać się rolą mało ciekawą i nie do końca ważna - jednak to on ma najwięcej cech tradycyjnego menadżera projektu. Osoba w tej roli podejmuje wszystkie strategiczne decyzje dotyczące projektu i jest za nie odpowiedzialna przed interesariuszami.

Z punktu widzenia teamu najważniejszymi funkcjami product ownera są: definiowanie zadań, ich definicji wykonania oraz ewaluacja wykonania zadań - inaczej mówiąc to product owner mówi co będzie zrobione, ustala kryteria jakie wykonana praca ma spełniać i decyduje czy rzeczywiście te kryteria zostały speł-

nione. Oczywiście zadanie ochrony członków zespołu przed osobami zewnętrznymi jest nie mniej ważne - jednak prawidłowo wykonywane nie powinno być nawet zauważone.

Jako że nadrzędnym celem product ownera jest zapewnienie wytwarzania jak najwyższej wartości w projekcie (w całym projekcie oraz na każdym z jego etapów!) - powinien on się cechować albo dobrą znajomością problematyki produktu jaki wytwarza, albo mieć dobrych doradców i słuchać ich zdania. Jeżeli np. product owner uzna kwestię bezpieczeństwa danych na serwerach pocztowych za niewymagającą szczególnej uwagi (w końcu żaden użytkownik nie będzie z niej korzystał!) to po włamaniu się na konta użytkowników i ujawnieniu ich danych, firmę czeka wypłata odszkodowań i zmniejszenie rzeczywistej rentowności projektu.

Product owner nie musi być ekspertem od metodyki scrum, a na pewno nie jest potrzebne (a wręcz lepiej jest gdy taka sytuacja nie ma miejsca) by był ekspertem w technologii lub metodzie wytwarzania produktu projektu. Pierwsze należy do scrum mastera, który ma obowiązek product ownera o scrumie uczyć a drugie jest odpowiedzialnością i własnością teamu. Product owner posiadający wiedzę na temat wytwarzania będzie w sposób naturalny odbierał zespołowi prawo do decydowania o tym, jak należy wykonywać pracę - najprawdopodobniej ze złym skutkiem (product owner nawet jeżeli jest ekspertem - to raczej osobiście nie wykonuje produktu!).

Nie sposób powiedzieć, która z dwóch przedstawionych ról (scrum master, product owner) jest ważniejsza dla projektu, lub kto stoi w projekcie nad kim, zresztą nawet zastanawianie się nad tym przez osoby wykonujące te role, oznacza, że być może nie rozumieją tego, że projekty wykonują całe zespoły. W cyklu sprintu występują momenty, że to scrum master podporządkowuje się decyzjom product ownera(patrz 5.2.1), jak i odwrotnie (patrz 5.2.3).

## 6.3 Team

Podstawowe cele pracy teamu:

- Decydowanie o ilości zadań, jakie są w stanie jako team wykonać.
- Ocena pracochłonności zadań stawianych przez product ownera.
- Decydowanie o najefektywniejszej metodzie wykonania produktu.
- Tworzenie produktu projektu zgodnie z ustaloną z właścicielem produktu definicją wykonania.

Członkowie „The Team“ są osobami odpowiedzialnymi całościowo za wytwarzanie produktu projektu zgodnie z zadanymi kryteriami - co ważne, nie są oni porządkowani w żadne hierarchie czy podzespoły, więc do pracy w zespołach scrumowych preferowani będą zawsze pracownicy multidyscyplinarni nad specjalistami w jednej konkretnej dziedzinie. Oczywistym jest, że niektórzy z członków zespołu będą posiadać umiejętności, których nie będzie miał nikt inny - jednak w trakcie pracy powinien następować stopniowy transfer wiedzy i umiejętności, między innymi poprzez pracę w parach. Nie wszystkie umiejętności potrzebne do tworzenia oprogramowania mogą być łatwo przekazywane między różnymi osobami - zaliczyć można do mich na przykład tworzenie grafiki, czy opracowywanie dokładnych planów interfejsów użytkownika. W pierwszym przypadku jest to wynik konieczności posiadania umiejętności artystycznych, w drugim - bardzo kompleksowej wiedzy. Testowanie jest przykładem umiejętności, która może ulegać transferowi między członkami zespołu. Jeżeli na przykład w danym zespole znajduje się tylko jeden tester, to po kilku sprintach, co najmniej kilku programistów nie powinno mieć problemu z wykonywaniem testów. Jest to tym bardziej ważne, że z natury rzeczy liczba testów po koniec sprintu będzie większa niż na początku, a liczebność zespołu jest stała. Zawodowy tester będzie zapewne lepiej wykonywał swoją pracę niż przyuczeni programiści, więc powinien zgłaszać się do zadań, które ocenia jako najbardziej ryzykowne.

Członkowie zespołu powinni zawsze być świadomi, że to oni wytwarzają produkt sprintu a nie mistrz młyna, i że to oni deklarują ile są w stanie wykonać - co oznacza, że to w ich najlepszym interesie a nie w interesie mistrza młyna czy

właściciela produktu jest dbanie o to, by praca posuwała się naprzód i by zawsze podejmować się zadań które są niezbędne do wykonania, nawet jeżeli nie są w pełni zgodne z ich profilem zawodowym.

Rzeczą niepożądaną jest jeżeli członkowie zespołu nie czują się zobowiązani do wykonywania zadań jakich się podjęli, sprawia to, że nie zachodzą podstawowe relacje na których bazuje scrum - samoorganizacja (brak potrzeby samoorganizacji) oraz uczciwa estymacja złożoności zadań, dzięki której możliwe jest określenie, w jakim stadium znajduje się projekt (nie da się prawidłowo estymować zadań, jeżeli regularnie nie są wykonywane na czas i zespół nie odczuwa z tego powodu żadnych konsekwencji).

# Rozdział 7

# Osadzenie scruma w organizacji

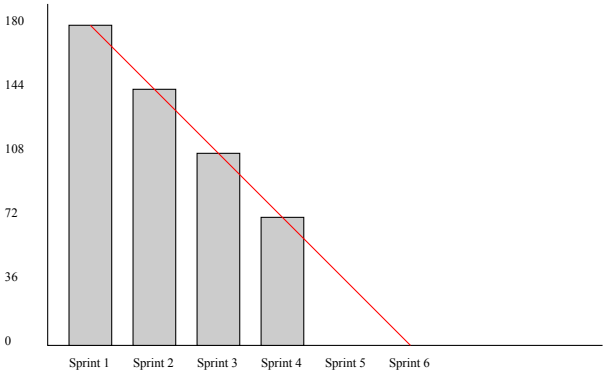
## 7.1 Wykresy i analizy

Jedną z podstawowych własności jakich oczekuje się po dowolnej metodyce zarządzania projektami jest możliwość analizy wyników jakie osiąga zespół projektowy oraz planowanie działań w oparciu o te analizy. Scrum nie jest wyjątkiem w tym zakresie i również posiada narzędzia potrzebne do tego celu. W ramach metodyki opisane jest kilka rodzajów wykresów stworzonych w tym celu.

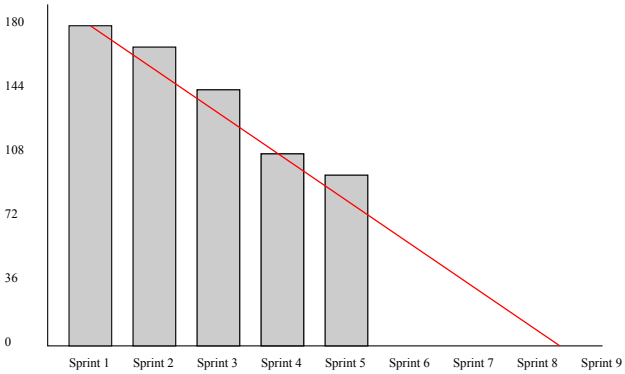
### 7.1.1 Burndown chart

Najważniejszym narzędziem obrazującym postęp prac w scrumie jest burndown chart - można go sporządzać dla release oraz dla sprintu a ukazuje ilość pozostałej pracy oraz prędkość z jaką wykonywane są kolejne zadania.

Na rysunku 7.1 przedstawiłem prosty przykład wykresu dla bardzo nieskomplikowanego przykładu, gdzie wydanie zaczęto od elementów backloga produktu ocenionych łącznie na 180 story pointów, a w każdym sprincie wykonywano po 36 punktów. Linia widoczna na wykresie to wskaźnik przewidywanego zakończenia release - tj. zaimplementowania wszystkich wymaganych funkcjonalności. Taki przypadek jednak prawie na pewno się nie przydarzy - zwykle pierwsze sprinty pozwalają na wytworzenie znacząco mniejszej ilości funkcjonalności niż kolejne, zdarzać się będą również sytuacje, gdy sprinty przyniosą mniej funkcjonalności niż zakładano. Realny wykres znajduje się na rysunku 7.2, jednak



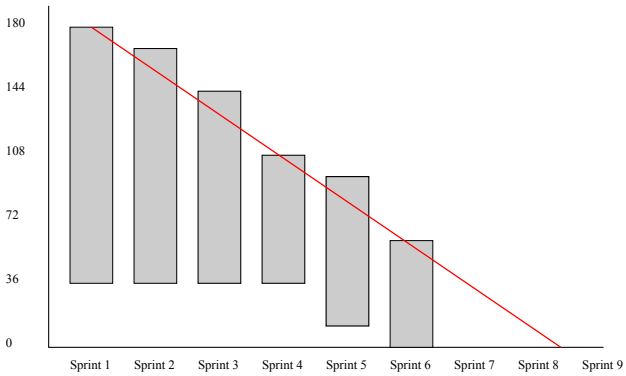

Rysunek 7.1: Wykres wypalenia w idealnym świecie.



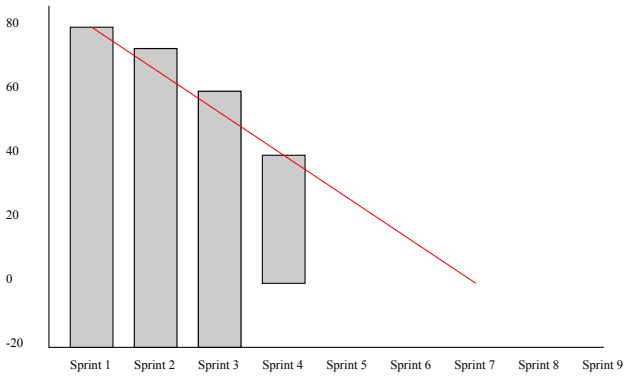

Rysunek 7.2: Wykres wypalenia w realnym świecie.

również on nie oddaje zbyt dobrze rzeczywistości, ponieważ pokazuje nam tylko ile pracy zostało, nie uwzględniając możliwości dodania lub odjęcia wymagań między sprintami. Nie wiemy czy w sprincie piątym zespół nie poradził sobie z jakimś dużym zadaniem, czy też może właściciel produktu dodał dodatkowe wymagania. Aby rozwiązać ten problem wymyślono ulepszony wykres - w którym praca wykonana przez zespół zawsze obniża wysokość słupka z danego sprintu o wartość wykonanych funkcjonalności, ale słupki nie muszą sięgać zera. Jeżeli w którymś sprincie dodane zostały wymagania (rysunek 7.3) - wcześniejsze słupki podnosi się o wartość dodanych wymagań, a w sytuacji odwrotnej (rysunek 7.4) - podnosi się skalę.

Najważniejszą wadą tego rodzaju wykresu jest to, że standardowo narzędzia do wspomagania zarządzania projektami nie generują tego diagramu, lecz tradycyjny.




Rysunek 7.3: Dodanie wymagań.




Rysunek 7.4: Usunięcie wymagań.

Jak już wspomniałem, diagram ten można zastosować również do pojedynczego sprintu - wtedy na osi odciętej umieszcza się dni pracy (na przykład bez sobót i niedziel, jeżeli w naszym zespole są to dni wolne) zamiast sprintów. Generowanie wykresu wypalenia dla sprintu wydaje się być całkiem sensownym pomysłem, ponieważ pozwala na szybkie określenie czy w sprincie uda się wykonać zaplanowane zadania, czy nie. Aby jednak wykres był miarodajny musi zajść sytuacja, w której user stories podjęte przez zespół są małe w stosunku do długości sprintu - inaczej mamy sytuację, gdy przez 80% czasu sprintu słupki utrzymują się początkowej wartości a na samym końcu spadają bardzo gwałtownie.

### 7.1.2 Team velocity

Z analizy wyników zespołu można określić wskaźnik prędkości wykonawczej zespołu mierzonej w ilości wykonanych story pointów na sprint, co pozwala miarodajnie (po 3-4 sprintach) planować zadania na sprinty. Trzeba pamiętać, że ponieważ story points są nieporównywalne między zespołami - ich prędkość mierzona właśnie w sp/sprint również jest nieporównywalna! Są dwa sposoby określania prędkości zespołu. Jeden sposób to zwykła średnia z dotychczasowych sprintów, posiadający tę niewątpliwa zaletę, że jest bardzo prosty w obliczeniu i większość narzędzi wspomagających prowadzenie projektu scrumowego oblicza go za scrum mastera. Inny sposób, a właściwie rodzina sposobów, to różnego rodzaju średnie ważone - uwzględniające fakt, że ostatni sprint lepiej odwzorowuje aktualną prędkość zespołu niż poprzednie(za [6]). Dość wiarygodny wynik uzyska się gdy wagi obliczać się będzie jako $W_n = \frac{1}{2^{m-n}}$, gdzie n równe będzie numerowi sprintu a m liczbie sprintów. łatwiejszą jednak i możliwą do wykonania w locie operacją jest wyciągnięcie średniej ze średniej arytmetycznej (dla wszystkich sprintów - wyliczonej automatycznie narzędziem) i ostatniego sprintu. Do oszacowania wydajności zespołu należy jednak zawsze brać poprawkę ze względu na czynniki takie jak: dodatkowe dni wolne, urlopy czy drobne zmiany składu zespołu, które mimo iż niezgodne z metodyką - w realnym świecie są nieuniknione. Jeżeli zdarza się sytuacja, w której zmiany w zespole są znaczne, niestety dane historyczne stają się mało przydatne - z tego samego powodu, dla którego prędkość zespołu jest nieporównywalna między dwoma różnymi zespołami.

## 7.2 Infrastruktura w firmie a scrum

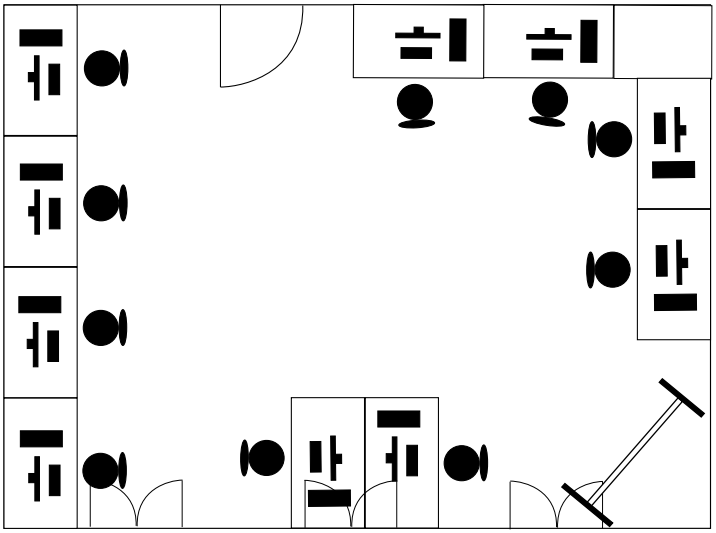
Scrum nie ma zbyt dużych wymagań co do infrastruktury projektowej, poza jednym ważnym wymogiem - od samego początku metodyka wymagała by zespół pracujący nad projektem wykonywał swoją codzienną pracę w jednym pomieszczeniu, bez żadnych ścianek działowych[1]. Scrum nie wymaga również żadnego specjalnego oprogramowania - całość projektu można bardzo efektywnie prowadzić przy użyciu tablic i żółtych karteczek.

Nie zmienia to jednak faktu, że zarówno odpowiednio przygotowane pomieszczenia, jak i oprogramowanie wspomagające zarządzanie projektem czy zespołowe tworzenie oprogramowania znacząco ułatwiają pracę i podnoszą jej efektywność.

### 7.2.1 Pokój zespołu

Każdy scrumowy zespół powinien pracować w jednym pomieszczeniu. Z uwagi na ten fakt, pomieszczenie to powinno być raczej większe niż mniejsze,około $50m^2$ wydaje się być wartością optymalną dla zespołu w którym pracuje 9 osób i mistrz młyna (czyli maksymalnie duży zespół). Taka wielkość wynika nie tyle z przepisów bhp wedle których, przy założeniu, ze pomieszczenie ma $3m$ wysokości, minimalna powierzchnia użytkowa dla pracy 10 osób wynosi $43\frac{1}{3}m^2$( daje to $13m^3$przestrzeni na pracownika oraz ponad $4,3m^2$ powierzchni, gdzie bez problemu po odjęciu ustawowych $2m^2$ wolnej powierzchni można ustawić biurko, krzesło i jeszcze zostanie nieco miejsca na wspólną szafkę i jakąś roślinę). Wynika z faktu, że do pracy zespołu potrzebna jest chociażby przestrzeń w której wszyscy członkowie zespołu mogą się zebrać i na stojąco odbyć codzienny scrum. Dobrze jest jeżeli w pomieszczeniu można ulokować zespół w 2-3 grupy zadaniowe, wygospodarowany jest kąt z tablicą do projektowania a przy każdym biurku miejsce do pracy przez dwie osoby naraz. Przykładowy rozkład takiego pomieszczenia przedstawiłem na rysunku 7.5. Złą sytuacją jest taka, gdy scrum master posiada osobny pokój do pracy - rodzi to z jednej strony problemy związane z budowa-

[1]Od czasów powstania metodyki przeprowadzono wiele przypadków wdrożenia scruma w zespołach pracujących zdalnie - jednak zawsze wiązało się to z jakąś formą zapewnienia ciągłej komunikacji miedzy członkami i zawsze bardziej pożądaną sytuacją pozostawała kolokacja.



Rysunek 7.5: Pomieszczenie zespołu scrumowego

niem się hierarchii - co skutkuje zmniejszaniem chęci teamu do samoorganizacji, z drugiej zaś ogranicza kontakt zespołu z scrum masterem.

### 7.2.2 Narzędzia do zarządzania projektem

W opisach metodyki zwykle bazuje się na założeniu intensywnego wykorzystania tablicy z zadaniami reprezentowanymi przez żółte karteczki. O ile ta metoda sprawia, że w czasie codziennych scrumów rzeczywiście wszyscy widzą jakie zadania jeszcze trzeba wykonać w danym sprincie, o tyle sprawia to, ze wszelka dokumentacja procesu którą mógłby być ktoś zainteresowany w przyszłości musi być wykonywana przez scrum mastera i aktualizowana na bieżąco wraz ze zmianami w tablicy. Poza tym problemem - każdy może podejść i swobodnie przemieszać kartki na tablicy.

Na rynku istnieje co najmniej kilka dobrych, godnych polecenia systemów pozwalających na efektywne zarządzanie procesem scrumowym. Wiele z narzędzi jest darmowych, rozwijanych na licencjach opensource, darmowych dla zespołów do 10 osób, lub dostępnych dla takich zespołów za drobną opłatą. Niestety, jak wynika z analizy przeprowadzonej w firmie w której pracuję - narzędzia darmowe nie posiadają wszystkich potrzebnych funkcji, jak: widok tablicy zadań na sprint, czy integracja z zewnętrznym systemem wiki, do utrzymywania wymagań.

Z godnych polecenia produktów płatnych można wymienić Rational jazz firmy IBM oraz Jira/Greenhopper firmy Atlassian.

System firmy IBM jest silnie zintegrowany z repozytorium kodu, systemem budowania i kilkoma innymi modułami, dodatkowo jest darmowy dla małych zespołów (do 10 osób). Niestety, jego silna integracja z zestawem narzędzi dostarczanym razem z nim, sprawia, że decydując się na niego nie możemy komfortowo pracować na innych narzędziach, co ze względu na wykorzystanie innego systemu automatycznego budowania oraz zintegrowanego środowiska developerskiego przekreśliło jego zastosowanie w naszym przypadku. Dodatkowo Jazz jest propozycją dosyć nową, więc na rynku nie ma dostępnej dużej ilości dodatków.

System firmy Atlassian dla zespołów do 10 osób kosztuje 20 dolarów, za dodatkowe 10 możemy kupić bardzo dobre wiki (Confluence), system automatycznego budowania (choć tu akurat polecałbym darmowy hudson) czy oprogramowanie single sign-on, które pozwala na integrację całości z resztą infrastruktury firmowej. Niestety cena rozwiązania dla większej liczby osób bardzo szybko wzrasta, tak więc, jeżeli istnieje zapotrzebowanie na więcej licencji niż 10, należy liczyć się z wydaniem co najmniej kilku tysięcy dolarów. Jira jest systemem obecnym na rynku od wielu lat, przez co istnieje do niej bardzo dużo dodatków, a jeżeli rekrutujemy programistów, to z wysokim prawdopodobieństwem mieli oni już z nią styczność (chociażby dlatego, że dla projektów opensource jest darmowa).

Niezależnie od rozwiązania jakie się wybierze, aby w pełni wykorzystać zalety płynące z jego zastosowania, należy wprowadzić w zespole pewne zasady. Zasady te powinny wymagać wiązania zapisów do repozytorium kodu, z zadaniami aktualnie wykonywanymi, czy rejestrowania czasu spędzonego nad zadaniami. Tylko jeżeli narzędzia będą wypełnione rzeczywistymi danymi, możemy korzystać z analizy wydajności zespołu i efektywnie planować kolejne sprinty. W innym przypadku lepiej robić to ręcznie - oprzemy się wtedy pokusie wykorzystania niedokładnych danych.

Przy wdrażaniu narzędzi należy pamiętać o tym, że robi się to, aby usprawnić proces a nie go utrudnić lub wypaczyć. Narzędzie nie zastąpi na przykład codziennych scrumów, mimo iż takie zapewnienia możemy przeczytać w niektórych materiałach promocyjnych.

### 7.2.3 Narzędzia do zespołowego tworzenia oprogramowania

Jednym z wymogów scruma jest to, aby samo-organizujący się zespół wykonawczy wytwarzał produkt zgodnie z najlepszymi dostępnymi dla niego dobrymi praktykami. W przypadku tworzenia oprogramowania do takich praktyk należą między innymi ciągła integracja kodu i testowanie testami automatycznymi, jak i dokumentowanie kodu metod publicznego api wystawianego przez pisane przez nas moduły.

Aby zapewnić ciągłe testowanie i sprawdzanie pisanego kodu warto użyć narzędzia do ciągłej integracji, jak hudson, czy bamboo. Narzędzie takie po każdym wpisie do repozytorium, zbuduje całą aplikację i uruchomi wszystkie testy automatyczne, jakie zdefiniujemy. Warto, by nieudana próba zbudowania lub uruchomienia testów była w jakiś sposób raportowana na bieżąco - można w tym celu podłączyć głośnik do serwera budującego i napisać skrypt, który w przypadku błędu oznajmia to zespołowi.

Niestety, nawet najlepszy system automatycznego budowania i odpalania testów na nic się nie zda, jeżeli testy nie będą napisane. W zapewnieniu dobrego pokrycia kodu testami, jak i spełnianiu innych metryk stanowiących o dobrej jakości kodu (procent udokumentowanego publicznego api, liczba powtórzeń kodu, poziom skomplikowania metod czy powiązania między klasami), może nam pomóc narzędzie o nazwie sonar. Pozwala ono na statyczną analizę kodu aplikacji napisanych w języku Java, podaje procent pokrycia testami jednostkowymi (choć niestety niezbyt dokładnie - istnieje lepsze narzędzie do tego - nazywa się clover, ale kosztuje 1200 dolarów) i inne metryki. W ramach procesu scrumowego można wyznaczyć jeden dzień w tygodniu, który będzie służył poprawianiu takich statystyk - dzięki temu poprawi się jakość kodu a co za tym idzie zmniejszy się liczba błędów wykrytych na etapie wdrożenia systemu.

# Rozdział 8

# Skalowanie scruma

Jak napisałem wcześniej, jeden zespół w scrumie może składać się maksymalnie z 9. osób oraz scrum mastera i product ownera - dla wielu projektów będzie to wielkość niewystarczająca. Aby rozwiązać ten problem opracowano sposób skalowania scruma na większe ilości zespołów(opisane między innymi w [3]). Bardzo ciekawą własnością odkrytą przy tej okazji była liniowa skalowalność metodyki. Dodawanie kolejnych członków zespołu liniowo podnosiło prędkość wytwarzania oprogramowania - w każdych sprawdzonych warunkach (nawet dla setek zespołów!) (patrz [6]). Sprawia to, że scrum jest metodyką świetnie nadającą się dla dowolnej wielkości projektów, mimo iż wiele osób do dziś błędnie uważa, że metodyki „agile“ nadają się jedynie dla niewielkich firm lub ich oddziałów. Oczywiście nie obala to problemu mitycznego osobo-miesiąca, ponieważ zyski w wydajności pojawiają się dopiero po pewnym czasie od dodania kolejnych osób - więc ratowanie dodatkowymi etatami projektów na finiszu niestety nic nie zmieni.

## 8.1 Pierwszy poziom - ilość zespołów między 1 a 3

W przypadku konieczności posiadania nie więcej jak 3. zespołów skalowanie można przeprowadzić bardzo prostą metodą. Scrum master, jeżeli nie ma żadnych innych obowiązków może być przydzielony do nawet 3 zespołów naraz, to samo tyczy się product ownera. Każdemu zespołowi należy wtedy wyzna-

czyć inne dni rozpoczęcia i zakończenia sprintu oraz przeprowadzać codzienne scrumy o nieco innych godzinach - minimalnie co 20 minut (scrum master musi się chwilę przygotować przed każdym z nich). Poprzez współdzielenie product ownera oraz scrum mastera zapewniona zostaje podstawowa synchronizacja zespołów na płaszczyźnie wymagań i rozwoju projektu z punktu widzenia rozwijanych funkcjonalności.

Wszystkie zespoły korzystają z jednego backlogu produktu, na spotkaniach planistycznych zadania brane są z niego do wykonania w sprintach poszczególnych zespołów. Oznacza to, między innymi, że albo zespoły ustalą jeden wspólny sposób estymowania story pointów dla zadań albo dla każdego zadania będzie trzeba trzymać estymaty w trzech skalach. Obydwa rozwiązania mają wady i zalety. Pierwsze sprawia, że potrzebna jest większa synchronizacja między zespołami, a wstępne estymaty muszą być przygotowane we względnie dużych grupach (więc odpowiednio dużo osób jest wyłączona z normalnej pracy na czas potrzebny do estymowania) z równym udziałem każdego z zespołów (dla trzech zespołów, by uzyskać realną reprezentację każdego z nich potrzeba w zależności od wielkości zespołów 6-9. osób). Zaletą ustalania estymat w jednolitej skali, w reprezentatywnej grupie jest dodatkowy transfer wiedzy między zespołami, jaki odbywa się przy rozpatrzaniu trudności zadań. Drugi sposób wymaga zwykle modyfikacji istniejących systemów wspomagających zarządzanie, poza tym, jeżeli estymaty są widoczne przez inne zespoły - te które estymują zadania w późniejszym terminie mogą wpaść w pułapkę zakotwiczenia. Zaletą tego sposobu jest daleko mniejsze zaangażowanie we wstępną estymatę, a co za tym idzie większą ilość czasu można poświęcić na pracę nad produktem.

Synchronizację na poziomie architektury tworzonego systemu uzyskuje się dwutorowo. Po pierwsze, dzięki ułożeniu wszystkich spotkań w niekolidujących terminach, delegaci zespołów mogą jako „kurczaki“ uczestniczyć w spotkaniach innych zespołów - nie powinna być to więcej niż jedna lub dwie osoby, aby nie wprowadzać niepotrzebnego tłoku. Drugim torem powinny być spotkania osób specjalizujących się w konkretnych obszarach (na przykład baz danych lub interfejsów użytkownika). Jako że bieżące pytania i problemy rozwiązywać można po codziennych scrumach każdego zespołu - na spotkaniach synchronizacyjnych powinno się raczej omawiać strategię implementacji rozwiązań, tak, aby cały system

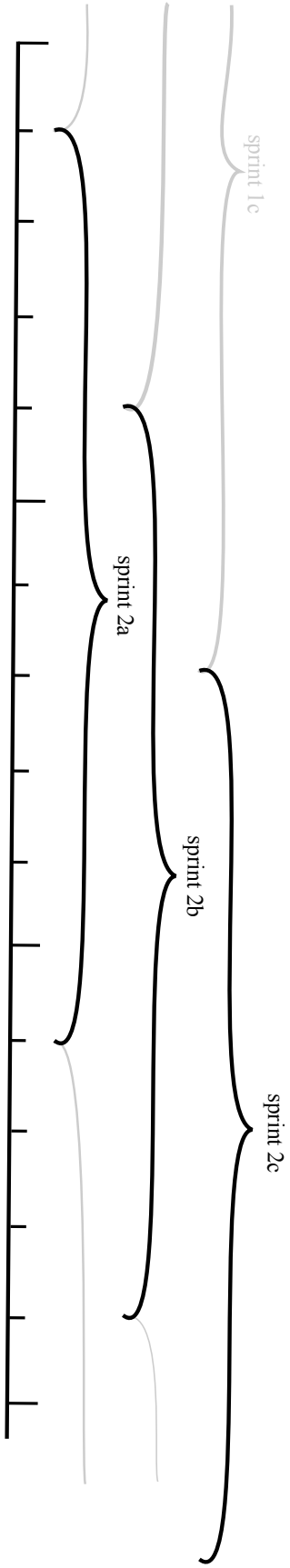
był spójny. Przykładowy rozkład spotkań dla dwutygodniowych sprintów na rysunku 8.1 pokazuje, jak bardzo obciążeni spotkaniami są w takim układzie scrum master i product owner. Z powodu tak dużego obłożenia zadaniami, ani scrum master ani product owner nie mogą zajmować się większą liczbą zespołów. Trzeba pamiętać, że scrum master poza tym musi pomiędzy spotkaniami krążyć po zespołach i sprawdzać czy coś nie spowalnia ich pracy, a product owner, zazwyczaj ma też inne obowiązki do spełnienia.

Nie należy natomiast starać się zapewnić synchronizacji poprzez okresowe zmiany składów zespołów - zaburza to silnie funkcjonowanie zespołów i ich samoorganizację. Zmiany personalne w zespołach należy traktować jako ostateczność - wszelkie problemy natury personalnej powinno się rozwiązywać w inny sposób, a braki w umiejętnościach raczej poprzez szkolenie.

Ważną kwestią jaką należy wspomnieć, jest fakt, że scrum master musi być permanentnie dostępny dla wszystkich zespołów - co oznacza, że tego rodzaju skalowanie musi być przeprowadzone w obrębie maksymalnie jednego budynku - a lepiej jednego biura. Ponieważ scrum master musi dzielić swój czas między kilka zespołów - rzeczą naturalną jest to, że nie będzie miał dla każdego z nich tyle czasu ile miałby przy zajmowaniu się jednym zespołem, z tego powodu w każdym zespole powinna znaleźć się jedna osoba, która w naturalny sposób jest uznawana przez grupę za lidera zespołu (w przypadku informatyków zazwyczaj będzie to oznaczać osobę uznaną za najlepszą w grupie, co sprawia, ze należy rozsądnie dobierać zespoły. Jeżeli w jakimś zespole najlepszą osobą będzie osoba nienadająca się na rolę lidera - może to spowolnić pracę całego zespołu).

## 8.2 Poziom drugi - ilość zespołów większa niż 3

Jeżeli projekt jest duży, 3 zespoły mogą się niewystarczające. Zapewne są osoby, które są w stanie prowadzić 4 czy nawet 5 zespołów jednocześnie - jednak prawdopodobieństwo znalezienia kogoś takiego i skłonienia go do tej pracy jest dość nikłe. W takim przypadku stosuje się technikę scruma scrumów - każdy zespół ma swojego scrum mastera i przypisanego product ownera (choć może być ich




**tydzień pierwszy** :
poniedziałek:
12:00 - podsumowanie sprintu 1a - zespół A
14:00 - retrospekcja sprintu 1a - zespół A
wtorek:
10:00 - spotkanie planistyczne sprintu 2a część pierwsza - zespół A
12:30 - spotkanie planistyczne sprintu 2a część druga - zespół A
środa:
11:00 - 14:00 - spotkanie architektów systemu
czwartek:
12:00 - podsumowanie sprintu 1b - zespół B
14:00 - retrospekcja sprintu 1b - zespół B
piątek:
10:00 - spotkanie planistyczne sprintu 2b część pierwsza - zespół B
12:30 - spotkanie planistyczne sprintu 2b część druga - zespół B
**tydzień drugi**:
poniedziałek:
11:00 - 14:00 - spotkanie architektów baz danych
wtorek:
12:00 - podsumowanie sprintu 1c - zespół C
14:00 - retrospekcja sprintu 1c - zespół C
środa:
10:00 - spotkanie planistyczne sprintu 2c część pierwsza - zespół C
12:30 - spotkanie planistyczne sprintu 2c część druga - zespół C
czwartek:
11:00 - 14:00 - spotkanie programistów GUI
piątek:
10:00 - 15:00 - spotkanie analityków z włascicielem produktu
**tydzień trzeci:**
poniedziałek:
12:00 - podsumowanie sprintu 2a - zespół A
14:00 - retrospekcja sprintu 2a - zespół A
wtorek:
10:00 - spotkanie planistyczne sprintu 3a część pierwsza - zespół A
12:30 - spotkanie planistyczne sprintu 3a część druga - zespół A
środa:
11:00 - 14:00 - spotkanie architektów systemu
czwartek:
12:00 - podsumowanie sprintu 2b - zespół B
14:00 - retrospekcja sprintu 2b - zespół B
piątek:
10:00 - spotkanie planistyczne sprintu 3b część pierwsza - zespół B
12:30 - spotkanie planistyczne sprintu 3b część druga - zespół B
...

**codziennie :**
9:00 - codzienny scrum zespołu A
9:20 - codzienny scrum zespołu B
9:40 - codzienny scrum zespołu C

10:00 - 16:00 -
spotkania i codzienna praca

Rysunek 8.1: Przykładowy rozkład spotkań dla dwutygodniowego sprintu.

nieco mniej niż zespołów). Sprinty wszystkich zespołów są zsynchronizowane - nie powinny być między sobą przesunięte o więcej niż dwa dni. Poza normalnymi sprintami - product ownerzy i scrum masterzy tworzą zespół nadrzędny, do którego przypisany jest kolejny scrum master i product owner. Jako, że scrum scrumów podlega takim samym ograniczeniom jak zwykły scrum - łatwo wywnioskować, że pod każdy może podlegać maksymalnie 5 zespołów (a i to tylko w wypadku, gdy właścicieli produktu jest mniej niż zespołów!). Długość sprintów nadrzędnego scruma musi być wielokrotnością (w szczególności trwać tyle samo!) scrumów podrzędnych - i z przyczyn oczywistych musi zaczynać się i kończyć w innym momencie niż scrumy podrzędne.

W przypadku hierarchicznego scruma scrumów modyfikacji ulegają planning meetings. Nadrzędne spotkania przypominają bardziej planowanie wydania niż zwykle planowanie sprintów. Zmiana jest spowodowana istnieniem wielu backlogów produktu - na spotkaniach scruma nadrzędnego rozpatruje się większy zbiór zadań niż na spotkaniach zespołów wykonawczych. Dodatkowo, jeżeli sprinty w obydwu poziomach mają tę samą długość - należy planować wyprzedzająco i tworzyć backlogi poszczególnych zespołów z myślą o 2-3 najbliższych sprintach a nie tylko o sprincie, który będzie planowany w najbliższym czasie. Spotkania planistyczne zespołów wykonawczych wyglądają prawie tak samo jak w przypadku jednego zespołu, z tą różnicą, że członkowie zespołu muszą baczyć na to, czy zadania których wykonania się podejmują nie są w jakiś sposób zależne od pracy innych zespołów - czego mogliby nie zauważyć będący na spotkaniu scruma nadrzędnego: właściciel produktu i mistrz młyna.

Poza synchronizację na poziomie zarządzania projektem, należy zadbać o synchronizację na poziomie architektury oprogramowania - sposób na to nie odbiega od poprzedniego omówionego.

Struktura taka została przetestowana nawet dla 4. poziomów zagnieżdżenia (patrz [3]) i dała liniowy (biorąc pod uwagę jedynie zwiększenie ilości team memberów) przyrost wydajności mierzonej w punktach funkcyjnych (patrz [6]).

## 8.3 Poziom trzeci - wiele zespołów wraz z profilowaniem zespołów do różnych zadań

Ostatni poziom profilowania rozdziela pewne klasy zespołów. Klasy te dzieli się nie w zależności od wykorzystywanych technologii czy umiejętności (choć różne umiejętności są potrzebne w różnych klasach) a ze względu na „odległość od klienta“. Jeżeli tworzymy duży system, który następnie sprzedajemy różnym przedsiębiorcom - będziemy mieli klasę zespołów zajmujących się sprzedażą i wdrażaniem systemu, klasę zespołów zajmujących się obsługą błędów i pomocą użytkownikom, ale jednocześnie potrzebować będziemy zespołów zajmujących się bieżącym pisaniem oprogramowania i rozwojem nowych funkcjonalności.

Warstwy te w działaniu przypominają elementy łańcucha kanban - zespoły bliżej klientów firmy stają się klientami dla właściciela produktu z zespołów tworzących system. Taki sposób podziału sprawia, że można sensownie przeprowadzić dystrybucję osób umiejących rozmawiać z klientami i osób które z klientami spotykać się nie powinny.

Poza poziomym podziałem na warstwy, jeżeli system składa się z względnie niezależnych modułów - warto przydzielać zespoły do pracy nad konkretnym modułem. Następuje wtedy lepszy efekt synergii, gdyż członkowie zespołu mają efektywnie mniejszy produkt do opanowania. Dzielenie zespołów w piony zajmujące się modułami, trudno przeprowadzić, jeżeli system technicznie nie jest do tego przystosowany, co niestety jest codziennością, szczególnie w systemach komputerowych sprzed 1995. Próba wprowadzenia podziału na zespoły zajmujące się poszczególnymi modułami w przypadku silnego powiązania wszystkich części kodu nie przyniesie efektów pozytywnych, a jedynie sprawi, że programiści będą mieli wewnętrzne mentalne przyzwolenie na nie wykonywanie pracy do końca, gdyż będzie ona poza jurysdykcją ich zespołu .

# Spis rysunków

# Nomenklatura

Dobre praktyki - są to pewne wzorce zachowań wypracowane przez branżę i powszechnie uznane za usprawniające wykonanie zadań, polepszające jakość produktu czy ułatwiające późniejsze wsparcie produktu.

Iteracja - przedział czasu w procesie tworzenia oprogramowania; po każdej iteracji powinna następować ewaluacja wykonanej pracy; w większości metodologii iteracje są stałej długości i oscylują w okolicach miesiąca.

Metodyka - jest to zbiór zasad, wzorów artefaktów, sposobów postepowania służący jakiemuś celowi. Przykładami metodyk są np. metodyki zarządzania projektami, metodyki wytwarzania oprogramowania czy metodyki prowadzenia badań. Metodyki tworzy się w celu uzyskania powtarzalnego dobrego rezultatu wykonywanych czynności.

OpenUP - metodyka zwinna bedąca rozwinięciem RUP - promowana silnie przez Eclipse Foundation jako odpowiednia metodyka do prowadzenia niewielkich projektów opensource - nie zdobyła jednak szerszego wdrożenia.

Produkt - jest to mierzalny efekt jaki powinien być przynoszony przez projekt, bezpośrednio spełniający cel uruchomienia projektu. Dla projektu, którego celem jest stworzenie oprogramowania - produktem jest oprogramowanie; dla projektu którego celem jest zwiększenie liczby obsługiwanych klientów, produktem będzie większa liczba obsługiwanych klientów.

RUP - Rational Unified Process - metodyka wytwarzania oprogramowania, stworzona przez firmę Rational, obecnie będąca częścią IBM. Prawdopodobnie najbardziej znana i rozpoznawalna metodyka tradycyjna; jest podzielona

na iteracje, jednak cały rozwój oprogramowania jest ramowo zaplanowany przed tym podziałem. Metodyka ta jest bardzo często stosowana w przedsiębiorstwach o silnie shierachizowanej strukturze.

Test akceptacyjny - zapisany zestaw kroków wraz z oczekiwanymi rezultatami które mają się dokonać po wykonaniu tych kroków; pozwalający stwierdzić, że funkcjonalność została wykonana.

Testowalność - możliwość jasnego określenia czy dany warunek jest spełniony czy nie. W przypadku wymagań oznacza to możliwość sprecyzowania jasnego testu, który określi czy wymaganie jest, czy nie jest spełnione.

# Bibliografia


[1] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Shwaber, Jeff Sutherland, and Dave Thomas. Agile manifesto. web, February 2001.

[2] Abrachan Pudussery. The sprint review meeting. web, June 2008.

[3] Ken Shwaber. *Agile Project Management with Scrum*. Microsoft Press, 2004.

[4] Ken Shwaber and Jeff Stuherland. Scrum guide. scrum.org, February 2010.

[5] Michele Sliger and Stacia Broderick. *The Software Project Manager's Bridge to Agility*. Pearson Education, 2008.

[6] Jeff Stuherland and Ken Shwaber. *The Scrum Papers: Nut, Bolts, and Origins of an agile Framework*. Jeff Sutherland, draft edition, May 2010.

[7] Hirotaka Takeuchi and Ikujiro Nonaka. The new new product development game. *Harward Business Review*, pages 137–146, January 1986.