Zwinne szacowanie pracochłonności w projektach programistycznych – studium przypadków

Maciej Łabędzki, Patryk Promiński, Adam Rybicki, Marcin Wolski

Poznańskie Centrum Superkomputerowo-Sieciowe

Abstrakt. Artykuł przedstawia doświadczenia zespołu programistycznego w zakresie szacowania pracochłonności. Autorzy, opierając się na znanych z literatury technikach estymacji, wskazują dobre i złe praktyki ich stosowania w zwinnym wytwarzaniu oprogramowania. Na przykładzie zrealizowanych projektów przedstawiono uwarunkowania do przeprowadzenia właściwej kalkulacji ilości pracy. Publikacja ma formę studium przypadku wskazując konkretne praktyki i łącząc je z problemami napotykanymi podczas szacowania kosztu wytworzenia oprogramowania. Doświadczenia zespołu deweloperskiego opisane w artykule dotyczą pracy w zwinnym podejściu Scrum.

Słowa kluczowe. wytwarzanie oprogramowania, szacowanie, pracochłonność, miary, inżynieria wymagań, story points, Scrum

JEL: L86

Wprowadzenie

Szacowanie pracochłonności jest ważnym zagadnieniem z zakresu zapewniania wysokiej jakości projektu informatycznego. Główną motywacją do tworzenia oszacowań jest potrzeba uzyskania odpowiedzi na pytanie: "W jakim czasie i przy jakim nakładzie środków jest możliwe zrealizowanie zadanej funkcjonalności?". Niekiedy zespół deweloperski może sam dostrzegać potrzebę tworzenia oszacowań, a innym razem będzie to praktyka przyjęta na poziomie organizacyjnym. W zależności od okoliczności i samego zespołu różnić się będzie nie tylko to co podlega szacowaniu, ale także sposób szacowania. Jednak użyteczność uzyskanych rezultatów zależna będzie od poprawności przyjętych założeń i wyboru właściwego podejścia.

Jak wskazuje dostępna literatura, problem szacowania kosztu wytworzenia oprogramowania sięga co najmniej lat sześćdziesiątych ubiegłego wieku (Nelson, Edward 1967; Farr, Zagorski 1964). Mnogość dostępnych publikacji naukowych świadczy zarówno o powszechności problemu jak i jego szerokim wymiarze (Jørgensen i inni 2007: 33-53). Przez okres ponad pięćdziesięciu lat zaproponowano wiele sposobów estymacji. Wśród nich są zarówno złożone jak i całkiem proste metody kalkulacji. Powstały eksperymentalne implementacje narzędzi automatyzujących takie obliczenia, np. z wykorzystaniem uczenia maszynowego (Srinivasan i inni 1995: 126-137), czy z użyciem sieci bayesowskich (Pendharkar i inni 2005: 615-624). W artykule nie podjęto próby analizy ich wszystkich, a tym bardziej – wskazania najlepszej z nich (Mahnic i Viljan 2011: 123-128; Bjarnason i inni 2016: 61-79; Torrecilla-Salinas i inni 2015: 124-144). Przedstawione zostały natomiast przykłady udanych i nieudanych zastosowań konkretnych praktyk, ich skuteczność oraz łatwość wdrożenia.

Publikacja przedstawia doświadczenie grupy programistów z dziesięcioletnim doświadczeniem w pracy z językiem Java. Zrealizowane przedsięwzięcia to w przeważającej większości systemy "szyte na miarę". Budowane są często od podstaw, rozwiązują nietypowe problemy i spełniają indywidualne wymagania zamawiającego. Zespół realizuje projekty małej i średniej skali w zwinnym podejściu Scrum (Schwaber i Beedle 2002; Schwaber i Sutherland 2013), a opisane w artykule doświadczenia odnoszą się do takich właśnie przypadków.

1. Projekty poddane analizie

Studium przypadku zostanie przedstawione na podstawie trzech referencyjnych projektów (nazwy dwóch z nich zostały zmienione). W każdym z nich były stosowane praktyki zwinne, natomiast różnice dotyczyły zasadniczo sposobu definiowania i dostarczania wymagań.

System OSW – projekt dotyczył przebudowy i rozwoju istniejącego systemu oświatowego. Realizowany przez okres 1 roku przez grupę pracującą w dwóch odrębnych lokalizacjach. Średnia liczba programistów zaangażowanych w projekt wynosiła 10 osób. Wiodącą technologią był język Java. Szczegółowe wymagania były znane z góry. Zespół oszacował wszystkie US (*ang. User Story*) dość dokładnie, ponieważ były one do siebie zbliżone funkcjonalnie. Jednocześnie ze względów technicznych nie mogły zostać sprowadzone do jednego ogólnego US w drodze generalizacji. W wyniku zmiany wymagań – co podczas trwania projektu miało miejsce wielokrotnie – US otrzymywało nowe oszacowanie. Zespół był w stanie precyzyjnie zaplanować pracę na najbliższy Sprint. Potrafił również określić termin ukończenia prac, a przy zmianie wymagań – przedstawić nowe oszacowanie.

System TOPO – budowa repozytorium informacji o topologii sieci komputerowej jako usługi bazowej dla systemów zewnętrznych. Prace w 6-osobowym zespole trwały 6 lat. Był to projekt w którym US znane są z góry i są zdefiniowane na dużym poziomie ogólności. Zasadniczo zespół deweloperski dzielił ogólne US na mniejsze, wewnątrz których identyfikował podzadania. Następnie dokonywał oszacowań dla tak zdefiniowanych US. US było tym dokładniej oszacowane, im mniej czasu pozostało do rozpoczęcia prac implementacyjnych.

System FOODIE – platforma wymiany wiedzy i zasobów elektronicznych dla przemysłu rolniczego[1]. Rozwijany od 16 miesięcy, w 3-osobowym zespole. Prototyp produktu posiadał dość ogólnie zdefiniowany cel. Wymagania funkcjonalne były identyfikowane na bieżąco, często w wyniku weryfikacji zrealizowanych przyrostów. W pierwszej fazie projektu brak wyraźnie określonych US - US posiadające oszacowania pozwalały na zaplanowanie dwóch najbliższych iteracji.

[1] https://www.foodie-cloud.org/

2. Podstawowe zagadnienia

Estymowanie ilości pracy to praktyka bardzo częsta, jednak nie absolutnie powszechna. Realizowanie prac bez oszacowań może mieć miejsce wszędzie tam, gdzie zespół nie musi raportować postępów swoich prac i nie ma wyznaczonego sztywnego terminu zakończenia przedsięwzięcia. Z perspektywy zespołu decyzja o prowadzeniu oszacowań należy do kierownika projektu, jednak w ujęciu ogólnym jest ona wynikiem konieczności kontrolowania kosztów i terminów. W projektach komercyjnych konkretne daty i kwoty są bardzo często elementem umowy, ponieważ przedsiębiorstwa alokują niezbędne do realizacji środki z wyprzedzeniem. W przypadku projektów badawczych, instytucja naukowa lub firma deleguje zespół, często grupa odbiorców systemu jest ograniczona, a efektów oczekuje w dłuższej perspektywie czasowej.

Jednak, bez względu na realny cel prowadzenia oszacowań, wartości te zawsze obarczone są błędem. Są one wynikiem działania mniej lub bardziej precyzyjnej intuicji zespołu, a nie kalkulacji twardych danych wejściowych. Z tego względu, z definicji, plany i prognozy oparte o takie dane są niedokładne.

2.1. Organizacja pracy w Scrumie

Bardzo popularną metodyką pracy zespołów programistycznych jest Scrum. Jako metodyka zwinna Scrum wymusza pracę w iteracjach i regularne przyrosty funkcjonalne. Chociaż sam Scrum wywodzi się z branży IT i tam najczęściej znajduje zastosowanie, to jednak stanowi sposób organizacji pracy nad dowolnym złożonym produktem, co podkreślają sami twórcy (Schwaber i Sutherland 2013).

Scrum wprowadza własną terminologię i jasno definiuje nowe pojęcia. Jednym z nich jest Rejestr Produktu (*ang. Product Backlog*). PB jest listą zadań dla zespołu pracującego nad produktem. Na liście znajdują się wszystkie zadania, bez względu na ich charakter, których wykonanie doprowadzi do osiągnięcia założonych celów. PB jest tworzony i pielęgnowany przez osobę odpowiedzialną za wyznaczanie kierunku prac w projekcie. Taka osoba zwykle najlepiej rozumie potrzeby przedsięwzięcia, a w terminologii scrumowej nazywana jest Właścicielem Produktu (*ang. Product Owner*). Do obowiązków tej osoby należy m.in. definiowanie zadań na poziomie merytorycznym (nie technicznym), nadawanie priorytetów i dbanie o to, by zadania były zrozumiałe dla wszystkich osób zaangażowanych w przedsięwzięcie. Strukturą PB przypomina stos, na wierzchu którego znajdują się zadania

wymagające realizacji w pierwszej kolejności. Pierwszeństwo wynika z kolei z tego, że zadania te mają dużą wartość biznesową (*ang. Business Value*), są zrozumiałe, zdefiniowane na dostatecznym poziomie szczegółowości, dobrze rozpoznane przez zespół i nie ma wątpliwości co do sensu ich realizacji. PO ma prawo redefiniowania priorytetów zadań (przenoszenia ich w ramach stosu na niższe lub wyższe pozycje) w zależności od zmieniających się uwarunkowań biznesowych. Ma również prawo do usuwania zadań i tworzenia nowych na dowolnym etapie prac projektowych. Narzędzie to pozwala Właścicielowi Produktu na bieżącą optymalizację wartości, jaką przedsięwzięcie ma dla organizacji.

Scrum nie narzuca formy definiowania zadań, a jedynie wymaga, by elementy PB były mierzalne pod kątem pracochłonności. Często jednak do opisu elementów Rejestru Produktu zespoły scrumowe stosują format *User Stories* (historyjki użytkownika). US jest znanym podejściem do definicji wymagań funkcjonalnych i niefunkcjonalnych. Dzięki mało formalnemu charakterowi US są zrozumiałe przez osoby nietechniczne i dzięki temu mogą być przez nie formułowane.

US jest jednozdaniowym opisem cechy, jakiej oczekuje się od budowanego produktu. Przykładowe US wygląda następująco: Użytkownik musi uwierzytelnić się zanim zacznie korzystać z aplikacji. Z US można dodatkowo skojarzyć specyfikację uszczegóławiającą (np. kryteria akceptacji, wyjątki). Istnieje też bardziej formalna postać US, dostarczająca standardowego zestawu informacji, zgodna ze wzorcem *„As a (role) I want (something) so that (benefit)*. Np. „Jako administrator chcę usuwać konta użytkowników po to, aby blokować dostęp osobom nieprzestrzegającym regulaminu”.

US dostarcza Właściciel Produktu. Jednak niekoniecznie jest ich autorem. US mogą być zebrane na drodze dyskusji z interesariuszami – osobami zainteresowanymi sukcesem projektu (np. potencjalnymi użytkownikami, sponsorami, analitykami). W Scrumie jednak to PO jest odpowiedzialny za ich zidentyfikowanie i umieszczenie w Rejestrze Produktu.

Pierwsze wymagania są zwykle formułowane na bardzo ogólnym poziomie abstrakcji. Wynika to m.in. z tego, że początkowo wizja produktu jest mglista. Bardziej szczegółowe wyobrażenie pojawia się w miarę upływu czasu. Dlatego właśnie zdarza się, że elementy Rejestru Produktu są jednocześnie bardzo ogólne i bardzo obszerne funkcjonalnie. Z tego powodu pracochłonność z nimi związana jest duża i trudna do oszacowania, a zespół nie jest w stanie zrealizować pojedynczego zadania w trakcie trwania iteracji. Takie zadania nazywa się

epikami (*ang. epic*). Epiki nie otrzymują oszacowań. Wymagają one analizy i rozbicia na mniejsze elementy – właśnie *User Stories*.

Niezależnie US mogą być grupowane w tzw. tematy (*ang. theme*). Pojedynczy temat grupuje US logicznie ze sobą związane. Np. temat „narzędzia administracyjne" agreguje funkcjonalności przeznaczone do codziennej pracy administratora.

Scrum definiuje wydarzenia obowiązkowe w procesie scrumowym. Są to: planowanie sprintu (iteracji), codzienny scrum, przegląd sprintu i retrospektywa. Podczas planowania sprintu zespół wybiera US, które zamierza zrealizować, na drodze porozumienia (negocjacji) z Właścicielem Produktu. Następnie zespół planuje zadania techniczne przynajmniej dla tych US, które zamierza zrealizować w pierwszej kolejności. Zatem pojedyncze US zdefiniowane ogólnie otrzymuje konkretne podzadania, które jasno określają sposób, w jaki zostaną one zaimplementowane.

Ze Scrumem związane jest pojęcie „definicja ukończenia" (*ang. Definition of Done*). DoD jest wzajemnym porozumieniem pomiędzy członkami zespołu scrumowego, określającym dokładnie, kiedy prace nad elementem PB można uznać za skończone. Np. US jest zrealizowane, jeśli istnieją automatyczne testy jednostkowe i integracyjne, serwer ciągłej integracji nie zgłasza zastrzeżeń, istnieją komentarze w kodzie i został wykonany przegląd kodu. Zespól zwykle zaczyna projekt z minimalną definicją DoD, którą systematycznie rozszerza w kolejnych iteracjach, tym samym podnosząc normy jakości stawiane przed produktem. Warto przy tym zauważyć, że oszacowanie pracochłonności pojedynczego US powinno uwzględniać również zadania wynikające z definicji ukończenia.

2.2. Rozróżnienie pomiędzy pracochłonnością, czasochłonnością i kosztem

Poniżej przedstawiono definicje podstawowych pojęć używanych w dalszej części dokumentu, tj. pracochłonności, czasochłonności i kosztu. Pojęcia te, choć mają różne znaczenie, są nierzadko stosowane zamiennie w praktyce zespołów programistycznych.

Czasochłonność jest to ilość czasu wymagana do realizacji US. Jest ona naturalnie wyrażana w jednostce czasu. Czasochłonność nie określa jednak ilości samej pracy a czas potrzebny do jej wykonania.

Pracochłonność jest to ilość pracy wymagana do realizacji US. W praktyce jest ona wyrażana w różnych jednostkach, zależnie od charakteru czynności związanych z wykonaniem

konkretnej pracy. Np. mogą to być linie kodu, ekrany wyświetlane użytkownikowi lub wypełniane przez niego formularze. Może być to również jednostka bardziej abstrakcyjna jak np. dobrze znane *Story Points* (Coelho i Basu 2012). Jednak bez względu na zastosowaną jednostkę, pracochłonność zawsze dostarcza odpowiedzi na pytanie o to, jak dużo pracy musi wykonać osoba realizująca US. Informacja o pracochłonności i wydajności osób zaangażowanych pozwala wyznaczyć czasochłonność.

Koszt należy rozumieć jako nakład środków, jakie musi przeznaczyć organizacja na realizację US. Zależnie od charakteru samej organizacji i przyjętych praktyk, koszt może być wyrażany np. w osobomiesiącach, kwocie pieniężnej lub innej dogodnej formie.

2.3. Wybór właściwej miary

Wybór miary pracochłonności, a tym samym – jednostki, może mieć znaczenie dla jakości gromadzonych danych i konstruowanych prognoz. Idealna miara to taka, dla której istnieje tylko jedna możliwa interpretacja wartości wyrażonych za jej pomocą. By lepiej to zobrazować, wystarczy porównać jednostkę miary "metr" ze stosowanym dawniej podejściem do wyrażania odległości w dniach i tygodniach. Stosowanie jednostki czasu miało sens, kiedy podróż możliwa była jedynie konno lub pieszo, a prędkość podróży była zawsze przewidywalna. Dziś taka miara może być myląca z uwagi na różnorodność dostępnych środków transportu.

Decyzja o wyrażaniu pracochłonności US i zadań technicznych w dniach i tygodniach (Jørgensen 2016) może się wydawać naturalnym wyborem, gdy celem nadrzędnym jest zbudowanie harmonogramu prac i zaplanowanie wydań oficjalnych, przetestowanych i wolnych od błędów wersji produktu (jak w przypadku projektu TOPO). Jednak, gdy zespół stanowią osoby o mocno zróżnicowanym doświadczeniu, pojawiają się różnice w wydajności pracy, a w efekcie – czas realizacji takich samych US i zadań technicznych bywa różny i zależy od przypisania do konkretnego członka zespołu.

Ponadto, tak oszacowane wartości nieustannie się dezaktualizują z uwagi na rosnącą wydajność pracy całego zespołu, co z kolei wynika bezpośrednio z samoistnego poszerzania wiedzy zespołu o dziedzinie problemu i coraz lepszej znajomości zastosowanych technologii. W efekcie wartości takie stają się bezużyteczne i wymagają re-estymacji.

Powyższych problemów można uniknąć stosując, zamiast miary czasu, miarę abstrakcyjną – Story Points.

2.4. Szacowanie w Story Points

Nazwa tej miary nawiązuje do pojęcia “User Story”, czyli tzw. historyjek użytkownika (Patton i Economy 2014). Określa ona, nie wprost, ilość pracy potrzebnej na zrealizowanie pojedynczej funkcjonalności budowanego produktu. Miara ta nie jest natomiast związana bezpośrednio z czasem realizacji US. Wartość wyrażona w Story Points (SP) określa złożoność (trudność) pracy niezbędnej do całkowitego zrealizowania danej funkcjonalności. Miara SP może być również użyta do estymacji całkowitego kosztu utrzymywania oprogramowania (Choudhari i Suman 2012: 761-765).

SP to miara abstrakcyjna i z tego względu może przysparzać trudności, gdy niedoświadczony zespół decyduje się na jej zastosowanie. Potwierdzają to doświadczenia z pracy w projekcie TOPO, kiedy w zespole zapadła decyzja o wdrożeniu zwinnego podejścia. Nie jest bowiem jasne to, jaką wartość oszacowania powinno otrzymać pierwsze US. W jego przypadku nie istnieje jeszcze żaden punkt odniesienia.

Problem można rozwiązać poprzez dogłębną dyskusję na temat sposobu realizacji kilku wybranych US. Dyskusja powinna rozwiewać wszelkie wątpliwości na temat technicznej realizacji US. Taka wiedza daje podstawy, by US nadać oszacowania w Story Points. Na tym etapie stosuje się wartości takie, jakie zespół uzna za stosowne. W ten sposób powstaje zbiór US referencyjnych.

Ważne jest, aby złożoności US ze zbioru referencyjnego były możliwie najdokładniej określone. Kolejne US szacuje się poprzez porównywanie do tego zbioru i w dużej mierze od tej precyzji zależy jakość kolejnych oszacowań. Dlatego US referencyjne należy wybrać świadomie.

Dodatkowo pomocna może być praktyka polegająca na szacowaniu zbioru referencyjnego w dniach lub godzinach. Tak wyznaczone liczby przypisuje się następnie US. Od tej chwili wartości te są już traktowane jako Story Points. Jest to ułatwienie dla osób silnie przywiązanych do jednostek czasu. Należy mieć jednak świadomość, że wartości te odnoszą się już do pracochłonności a nie czasochłonności. Wynika to ze wspomnianego wcześniej zjawiska rosnącej wydajności pracy i rosnącego doświadczenia zespołu.

Nie bez znaczenia jest również zakres samych liczb. W praktyce sprawdzają się wartości z zakresu 0-13. Ma to związek ze specyfiką ludzkiej percepcji, a konkretnie z łatwością, z jaką dokonuje się porównań, gdy liczby nie różnią się od siebie znacznie. Niektórzy zalecają również, by były to liczby z ciągu Fibonacciego. Doświadczenie zdobyte w projektach OSW,

TOPO i FOODIE pokazuje, że większe wartości są kłopotliwe. Analiza dużych US jest trudna i najczęściej takie US dzieli się na mniejsze lub identyfikuje się podzadania.

2.5. Planning Poker, czyli szacowanie zespołowe

Poker planistyczny (Mahnič i Hovelja 2012: 2086-2095; Moløkken-Østvold 2008: 2106-2117) to praktyka umożliwiająca szacowanie w grupie osób. Estymacji dokonuje się z wykorzystaniem kart (stąd nazwa Planning Poker). Efektem tej praktyki są dyskusje, w wyniku których zespoły poprawiają swoją świadomość na temat ilości pracy stojącej za realizacją US .

Jak pokazuje doświadczenie w pracy z tą techniką, indywidualne oszacowania członków zespołu dla pojedynczego US mogą mocno różnić się od siebie. W projekcie FOODIE zjawisko to widać bardzo wyraźnie. W skrajnych przypadkach pojedyncze US otrzymywało jednocześnie oszacowanie na 5 i 13 punktów. Planning poker rozwiązuje problem rozbieżności i pokazuje, jak ryzykowne jest powierzenie oszacowań pojedynczej osobie.

Z tą praktyką związana jest jeszcze jedna pułapka. Realne są przypadki, kiedy do dyskusji wcale nie dochodzi. Wybierana jest wtedy wartość proponowana przez większość lub przez osobę o dużym autorytecie. Taka osoba może wywierać mimowolną presję na pozostałych – sytuacja miała miejsce w projekcie TOPO. Innym powodem wpadania w tę pułapkę jest brak dostatecznego zaangażowania zespołu w proces szacowania.

3. Szacowanie pracochłonności

3.1. Szacowanie przez porównanie

Zgodnie z tą techniką wartość pracochłonności (np. liczbę Story Points) dla US_X określa się poprzez porównanie do podobnego US_Y, które zostało zrealizowane w przeszłości. W optymistycznych przypadkach podobieństwo między US jest widoczne na poziomie opisu zawartego w specyfikacji wymagań dostarczonej przez zamawiającego. Dla przykładu, w projekcie OSW złożoność US zależała od liczby operacji w implementowanym interfejsie Web Service, od struktury danych wejściowych oraz od liczby warunków walidacji tych danych. Wszystkie te informacje zawarto w dokumentacji wymagań funkcjonalnych. W praktyce tak szczegółowa specyfikacja jest rzadkością, jednak zamawiający produkt posiadał dużą świadomość, także techniczną, na temat finalnego rozwiązania. Dokładna analiza dokumentacji wymagań pozwoliła na precyzyjne określenie pracochłonności.

Jednak nie zawsze mamy do czynienia z tak komfortową sytuacją. Bardzo często same wymagania funkcjonalne są jasne i zespół posiada kompetencje do ich realizacji. Nadal jednak trudno porównać US, gdy podobieństw w definicji wymagań nie ma. W takiej sytuacji należy dokonać głębszej analizy w kierunku identyfikacji konkretnych podzadań wewnątrz US, a następnie dokonać porównania na tym poziomie. Najczęściej bowiem okazuje się, że analogię można dostrzec np. w liczbie nowych modułów, klas, metod lub elementów konfiguracji.

3.2. Szacowanie zadań trudnych do oszacowania

W praktyce zdarzają US, którym trudno jest nadać konkretną wartość pracochłonności metodą porównania. Problem może pojawić się, gdy cel nie jest jasno określony. Należy wtedy zidentyfikować przyczyny takiego stanu rzeczy, a następnie dobrać odpowiednie rozwiązanie.

Jednym z powodów dla którego US może być trudne do oszacowania jest jego zbyt szeroka definicja. Rozwiązaniem okazuje się wtedy ponowna analiza wymagania i/lub rozbicie US na mniejsze US lub identyfikacja podzadań, które łatwiej jest oszacować.

Zdarzają się także US w których zespół deweloperski nie ma pełnej wiedzy odnośnie technicznych aspektów ich realizacji. W sytuacji, w której choć jeden członek zespołu jest w stanie powiedzieć, jak złożone jest US, może on wprowadzić resztę zespołu w aspekty techniczne. Zespół deweloperski musi poświęcić wtedy więcej czasu, aby dokładniej przedyskutować złożoność takiego US i nadać oszacowanie.

Jeśli jednak trudność wynika z braku pomysłu na sposób rozwiązania US, warto wtedy wprowadzić zadanie pomocnicze, ukierunkowane na zbadanie/rozpoznanie problemu. Nie otrzymuje ono oszacowania. Z tej perspektywy potraktowane jest podobnie jak błąd (patrz "Szacowanie błędów"). I podobnie, wpływa na obniżenie wydajności, ponieważ praca nad nim jest niezbędna, ale niemożliwa do przewidzenia.

Metodyka Scrum definiuje pojęcie "*backlog refinement*". Jest to praktyka, w myśl której zespół regularnie spotyka się w celu identyfikacji i rozwiązania wszystkich wątpliwości i problemów związanych z definicją wymagań. M.in. właśnie wtedy szacuje się pracochłonność elementów Rejestru Produktu. Wspomniane zadanie pomocnicze jako element doskonalenia backlogu pozwala na lepsze poznanie wymagania i sposobu jego realizacji, a w konsekwencji – pracochłonności.

Strategia ta dobrze sprawdziła się w projekcie FOODIE, który został oparty o gotowe rozwiązanie bazowe (*ang. framework*), które od początku dostarczało gotowej implementacji

dla dużej części ogólnych wymagań projektu. Skuteczność oszacowań kolejnych US była więc w dużej mierze uzależniona, od znajomości rozwiązania bazowego oraz od jego elastyczności. Często US, które wydawały się łatwe do realizacji, w ostatecznym rozrachunku wymagały dużo więcej pracy, niż by wynikało z oszacowań. Posłużenie się zadaniami pomocniczymi przed nadaniem oszacowań przyczyniło się do uzyskania dokładniejszych estymacji.

3.3. Szacowanie błędów

Podejście do szacowania pracochłonności błędów (*ang. bug*) może być różne i zależy ściśle od charakteru współpracy między wykonawcą i zamawiającym.

W projektach OSW, TOPO i FOODIE z założenia nie były szacowane błędy oraz zadania, które nie wnoszą przyrostu funkcjonalnego. Przyjęto założenie, że projekty rozliczane są tylko i wyłącznie z funkcjonalności zamówionych, a wszystkie pozostałe zadania są konieczne do zagwarantowania wysokiej jakości produktu. Do przykładów takich zadań należą: naprawa błędów, wykonanie dokumentacji technicznej, zestawienie lub rekonfiguracja infrastruktury (np. serwer ciągłej integracji). Ponadto zadania związane z naprawą błędów są zazwyczaj trudne do oszacowania, gdyż większość pracy stanowi szukanie przyczyny usterki, a sama naprawa jest relatywnie szybka.

W rezultacie pracochłonność nieoszacowanych zadań widoczna jest w prędkości pracy. Każda nowa funkcjonalność niesie ze sobą ryzyko pojawienia się błędów oraz wymaga przygotowania np. fragmentu dokumentacji technicznej. Co jakiś czas pojawia się również konieczność rekonfiguracji środowiska. Realna prędkość jest zatem wynikiem prac nad funkcjonalnościami i nieoszacowanymi zadaniami.

Konsekwencją takiego podejścia mogą być sprinty, których efektywność mocno odbiega od średniej. Są to iteracje, w których pracuje się np. nad błędami.

Jeżeli z jakiegoś powodu potrzebna jest wiedza o realnej pracochłonności związanej z naprawą błędów, warto zapisywać informację o czasie spędzonym nad wszystkimi zadaniami. Narzędzia takie jak JIRA oferują stosowną funkcjonalność. Dzięki takiej wiedzy istnieje np. możliwość wyznaczenia relacji pomiędzy pracochłonnością wymaganą do naprawy błędów a pozostałą pracą.

4. Planowanie i prognozowanie

4.1. Szacowanie pojemności sprintu

Po wstępnym szacowaniu US (np. z wykorzystaniem miary Story Points), do pierwszego sprintu trafiają US wybrane przez członków zespołu w drodze dyskusji, na podstawie ich subiektywnego przeświadczenia odnośnie swoich możliwości. Ukończony sprint dostarcza informacji o możliwościach przerobowych zespołu. Suma oszacowań zrealizowanych historyjek jest pierwszym rozsądnym oszacowaniem pojemności kolejnego sprintu. US nieukończone zostaną przeniesione do kolejnej iteracji, a ich pracochłonność zostanie zaliczona na poczet tego sprintu, w którym zostaną one ukończone (np. osiągną zgodność z przyjętą definicją wykonalności – Definition of Done).

Kolejne sprinty można planować w oparciu o obserwację uśrednionej prędkości pracy (*ang. velocity*) wyrażonej w punktach na sprint. Prędkość mówi o zdolności zespołu do dostarczania pełnych funkcjonalności. Doświadczenie z projektów OSW, TOPO i FOODIE pokazuje, że wartość ta może zmieniać się w czasie. W fazie początkowej projektu wydajność pracy rośnie z iteracji na iterację. Dzięki temu można pozwolić sobie na zwiększenie pojemności kolejnych sprintów. W tym celu można posiłkować się wykresem prędkości. Pomocna może być również praktyka liczenia pojemności na podstawie średniej ważonej z kilku ostatnich iteracji – wcześniejsze sprinty otrzymują mniejsze wagi.

Po kilku, kilkunastu iteracjach realna prędkość może się ustabilizować. Ma to miejsce wtedy, gdy zespół poznał już technologię, a US są dobrze zdefiniowane i podobne do siebie (jak w projekcie OSW i TOPO).

4.2. Zmiany prędkości pracy

Z uwagi na czynniki losowe możliwe są wahania prędkości lub jej pojedyncze gwałtowne zmiany. Wahania mogą wynikać z nieprecyzyjnie zaplanowanych iteracji, kiedy US nieukończone przechodzą z jednego sprintu na drugi i są w całości zaliczane do wyników tego drugiego.

Gwałtowna zmiana prędkości świadczy o zmianach w sposobie realizacji pracy. Skoki w górę można zaobserwować, kiedy wdraża się rozwiązanie typu framework i implementacja pewnych części systemu znacznie upraszcza się. Przypadek ten można zinterpretować na dwa sposoby. Zespół może uznać, że wzrosła jego wydajność, ponieważ szybciej wykonuje tą samą

pracę (podobieństwo zadań na poziomie wymagania funkcjonalnego). Jednak na poziomie technicznym nie jest to już ta sama praca. US uprościły się i wymagają mniejszego nakładu pracy, by dostarczyć tą samą funkcjonalność, a w efekcie – taką samą wartość biznesową dla produktu. Można zatem rozważyć ponowną estymację niezrealizowanych jeszcze zadań, najprościej – mnożąc wszystkie zalegające US podobnego typu przez odpowiedni współczynnik. To podejście dostarcza lepszej jakości oszacowania i prognozy. Jednak samo w sobie jest bardziej pracochłonne, ponieważ wymaga od zespołu, by US za każdym razem porównywał na poziomie technicznym.

Skoki w dół również mogą sygnalizować zmianę sposobu realizacji pracy. Dało się to zaobserwować w projekcie OSW, który realizowany był modułowo. W pierwszej kolejności realizowano US z modułu X. W drugiej kolejności realizowano analogiczne prace w części Y. Jednak do budowy fundamentów modułu Y programiści zastosowali inne rozwiązania techniczne, które negatywnie wpłynęły na prędkość realizacji bardzo podobnych funkcjonalności.

4.3. Zmieniający się skład zespołu

Zmienność składu zespołu jest ryzykiem, które należy uwzględnić przy rozpoczęciu projektów programistycznych. Sytuacje, w których zmienia się skład zespołu lub zupełnie inny zespół zostaje przydzielony do pracy nad projektem są częstą praktyką. W takich przypadkach ujawniają się zalety stosowania abstrakcyjnej miary (jak np. Story Points). Po takiej zmianie oszacowania pozostają nadal aktualne. Korekcie podlega jedynie prędkość pracy, a w rezultacie wszystkie prognozy.

4.4. Zmienne zaangażowanie członków zespołu

Możliwe są sytuacje, w których organizacja dynamicznie alokuje zasoby ludzkie do projektu. Niewykluczone są przypadki, kiedy sumaryczne zaangażowanie osób, liczone np. w osobodniach, zmienia się ze sprintu na sprint. Tej trudności można sprostać poprzez kontrolę sumarycznego udziału osób. Na spotkaniu planistycznym zespół deklaruje swój udział, w oparciu o który następnie określona zostaje pojemność iteracji. Wszelkie wskaźniki i prognozy należy wtedy wyznaczać w odniesieniu do założonego średniego zaangażowania zespołu.

Istnieją jednak trudności w ustalaniu faktycznego udziału. Można przyjąć, że zespół angażuje się na zadeklarowanym poziomie. Jednak w skrajnych przypadkach nie udaje się

dotrzymać deklaracji. Wtedy warto skorygować zapisane wcześniej wartości w drodze dyskusji. Innym sposobem pomiaru zaangażowania jest zachowywanie informacji o czasie poświęconym na pracę nad wszystkimi US (np. funkcja “log work” w systemie JIRA).

Podsumowanie i wnioski

W artykule autorzy przedstawili zbiór praktyk z zakresu szacowania pracochłonności i praktyczne aspekty ich użycia w projektach wytwarzanych w metodykach zwinnych. Podstawą rozważań były projekty o różnej charakterystyce – biorąc pod uwagę m.in. dziedzinę, grupę odbiorców, wielkość zespołu deweloperskiego – i zastosowane podejścia do szacowania. Podsumowując, podejście konkretnego zespołu do problemu estymacji jest uzależnione od natury samego projektu. Poker planistyczny jest praktyką, która daje dobre efekty w każdym przedsięwzięciu, zwiększając precyzję oszacowań i podnosząc poziom zrozumienia US oczekujących na realizację. Gdy wymagania funkcjonalne są jasne, a sposób ich realizacji dobrze znany, estymację można oprzeć o krótkie opisy US. Inaczej jest w przypadku złożonych technologii lub niedoświadczonego zespołu. Wtedy dużo uwagi należy poświęcić planowaniu rozwiązań technicznych. Jednak bez względu na wybraną płaszczyznę rozważań dobre efekty przynosi estymowanie przez porównanie do pracy zrealizowanej. Należy przy tym dołożyć starań do zbudowania dobrej jakości referencyjnego zbioru US. Ponadto użycie abstrakcyjnej miary, jak np. Story Points, w przeciwieństwie do jednostek czasu, pozwala uwolnić się od pułapki zmieniającej się wydajności pracy

Rozważania i rekomendacje przedstawione w publikacji mogą być punktem wyjścia dla nowo formowanych zespołów oraz podstawą do dalszych modyfikacji, dopasowanych do indywidualnych potrzeb i warunków pracy

Podziękowania

FOODIE jest projektem badawczym współfinansowanym z programu CIP (Competitiveness and Innovation Framework Programme) 7 programu ramowego UE (FP7).

Literatura

Bjarnason, Elizabeth, et al. "A multi-case study of agile requirements engineering and the use of test cases as requirements." Information and Software Technology **77** (2016): 61-79.

Choudhari, Jitender, and Ugrasen Suman. "Story points based effort estimation model for software maintenance." Procedia Technology 4 (2012): 761-765.

Coelho, Evita, and Anirban Basu, Effort estimation in agile software development using story points, *International Journal of Applied Information Systems* (IJAIS) **3.7** (2012).

Farr, Leonard, and Henry J. Zagorski. Factors that Affect the Cost of Computer Programming. Volume Ii. a Quantitative Analysis. SYSTEM DEVELOPMENT CORP SANTA MONICA CA, 1964.

Jorgensen, Magne, and Martin Shepperd, A systematic review of software development cost estimation studies, *Software Engineering, IEEE Transactions* on **33.1** (2007): 33-53.

Jørgensen, Magne, and Stein Grimstad, The impact of irrelevant and misleading information on software development effort estimates: A randomized controlled field experiment, *Software Engineering, IEEE Transactions* on **37.5** (2011): 695-707.

Jørgensen, Magne, Unit Effects in Software Project Effort Estimation: Work-hours Gives Lower Effort Estimates than Workdays, *Journal of Systems and Software* **117** (2016): 274-81.

Mahnic, Viljan. "A case study on agile estimating and planning using scrum." Elektronika ir Elektrotechnika **111.5** (2011): 123-128.

Mahnič, Viljan, and Tomaž Hovelja, On using planning poker for estimating user stories, *Journal of Systems and Software* **85.9** (2012): 2086-2095.

Moløkken-Østvold, Kjetil, Nils Christian Haugen, and Hans Christian Benestad, Using planning poker for combining expert estimates in software projects, *Journal of Systems and Software* **81.12** (2008): 2106-2117.

Nelson, Edward Axel. Management handbook for the estimation of computer programming costs. No. TM-3225/000/01. SYSTEM DEVELOPMENT CORP SANTA MONICA CA, 1967.

Patton, Jeff, and Peter Economy, *User story mapping: discover the whole story, build the right product.* O'Reilly Media, Inc., 2014.

Pendharkar, Parag C., Girish H. Subramanian, and James A. Rodger, A probabilistic model for predicting software development effort, *Software Engineering, IEEE Transactions* on **31.7** (2005): 615-624.

Schwaber, Ken, and Mike Beedle, *Agile Software Development with Scrum*, Upper Saddle River, NJ: Prentice Hall, 2002.

Srinivasan, Krishnamoorthy, and Douglas Fisher, Machine learning approaches to estimating software development effort, *Software Engineering, IEEE Transactions* on **21.2** (1995): 126-137.

Schwaber, Ken and Jeff Sutherland. “The Scrum Guide - The Definitive Guide to Scrum: The Rules of the Game” (2013)

Torrecilla-Salinas, C. J., et al. "Estimating, planning and managing Agile Web development projects under a value-based perspective." Information and Software Technology **61** (2015): 124-144.

Title:

Agile effort estimation in software development projects – case study

Abstract:

The article presents software development team experience in effort estimation. The authors base on the estimation techniques widely described in literature indicating good and bad practices they use in agile software development. With regard to given examples of completed projects conditions to carry out a proper calculation of the effort were shown. This publication is a case study. The development team’s experience described in this article is tightly related to an agile methodology Scrum.

Keywords:

software development, estimation, effort, measurement, requirements engineering, story points, scrum