PLANOWANIE ZADAŃ W ZWINNYCH METODYKACH PROJEKTOWYCH

Paulina RUSSAK, Mateusz RUSSAK

Streszczenie: W dzisiejszych czasach coraz więcej projektów prowadzonych jest przy użyciu zwinnych metodyk zarządzania. Wraz ze wzrostem wielkości i złożoności projektów pojawia się również problem odpowiedniego zaplanowania prac oraz ustalenia ich zakresu na najbliższy okres. Niniejsza praca stanowi przegląd i podsumowanie aktualnego stanu badań w tym zakresie oraz zawiera propozycję formalnego opisu problemu.

Słowa kluczowe: podejście zwinne, szeregowanie zadań, zarządzanie projektem, przegląd stanu badań

1. Wprowadzenie

W ciągu ostatnich lat zaobserwowano bardzo dynamiczne zmiany w zarządzaniu projektami w obszarze informatyki (ang. *information technology*, IT). Planowanie pracy w modelu kaskadowym (ang. *wartefall*) okazało się nieskuteczne w praktyce [1]. Dlatego kierownicy produktu (ang. *Product Managers*) zaczęli poszukiwać nowego wsparcia w procesie planowania, kierując się w stronę metodologii lepiej dostosowanych do dynamicznego charakteru pracy z oprogramowaniem. Jedną z takich alternatyw okazały się zwinne metodyki zarządzania projektami (ang. *agile*). W skład tej grupy zalicza się m.in. następujące techniki: Scrum, Kanban, Extremme Programming (XP) [2].

Najnowsze obserwacje pokazują, że na przestrzeni ostatnich lat zauważono znaczny wzrost zainteresowania wyżej wymienioną alternatywą [3]. Badania przeprowadzone przez firmę VersionOne w 2016 roku, w których udział wzięło ponad 50 000 zespołów z 1000 firm, potwierdzają wzrostowy trend popularności zwinnych metodyk zarządzania projektami [4]. Ankietowani wskazali jako najczęściej wykorzystywaną technikę Scrum (58%), na drugim miejscu uplasowało się połączenie Scrum'a z Xp czyli Scrum/XP Hybrid (10%). Kanban'a używało 5% respondentów. Dzięki zastosowaniu metod zwinnych co najmniej 58% projektów skończyło się terminowo i co najmniej w 48% przypadkach poprawiła się jakość produktów. Co więcej zaobserwowano wzrost zadowolenia klientów aż o 46% [4].

W skład grupy metodologii zwinnej wchodzi wiele szczegółowych technologii, które łączą wspólne zasady przedstawione w Agile Manifesto [5]. Opierają się one na ciągłym ulepszaniu oraz nadawaniu wyższego priorytetu kontaktom międzyludzkim i komunikacji. Procesy i ustalenia określone na drodze formalnej rozpatrywane są w drugiej kolejności. W Declaration of Independence (DOI) zdefiniowano główne reguły zwinnego zarządzania projektami: 1) ciągłe ulepszanie, 2) iteracyjny rozwój oprogramowania, 3) tworzenie oprogramowania na podstawie scenariuszy (ang. *scenario-driven*), 4) zorientowanie na biznes [6, 7].

Zwinne zarządzanie projektami pozwala na dynamiczny rozwój oprogramowania i nie wymaga określania szczegółowej specyfikacji wszystkich zadań do realizacji już na początku projektu. Wiąże się to z istotnym problemem dynamicznego szacowania ilości i wielkości zadań wchodzących w skład danego produktu oraz wyznaczaniu ścieżek krytycznych projektu. Zadaniem PO (ang. *Product Owner*) na początku wydania oprogramowania oraz w

każdej iteracji jest określenie aktualnego zakresu prac. Dalej należy wstępnie uszeregować zadania do wykonania, głównie dla określenia ścieżki krytycznej. Niniejsza praca ma na celu podsumowanie aktualnego stanu badań w zakresie metodyk zwinnych w zarządzaniu projektem, przeanalizowanie złożoności problemu oraz zaproponowanie wstępnego, teoretycznego opisu zagadnienia, który można poddać dalszej analizie.

W rozdziale 2 opisano główne zasady i hasła związane ze zwinnym podejściem do wytwarzania oprogramowania. W rozdziale 3 przedstawiono przegląd literatury na temat podejścia do problemu szeregowania zadań w iteracyjnych metodach projektowych oraz zestawienie proponowanych rozwiązań. W rozdziale 4 zaprezentowano różne modele występujące dotychczas w literaturze. W rozdziale 5 został przedstawiony autorski opis problemu szeregowania zadań w zwinnych metodykach projektowych. W rozdziale 6 zawarto wnioski oraz dyskusję kierunków dalszych badań.

2. Podejście zwinne do wytwarzania oprogramowania

Wytwarzanie oprogramowania jest procesem dynamicznym i często nieprzewidywalnym [8]. Wykorzystanie zwinnych metodyk zarządzania projektami wprowadza dodatkowe poziomy planowania pracy z różnym stopniem uszczegółowienia. Z perspektywy zarządzania projektami proces dostarczania oprogramowania w metodologiach *agile* składa się z następujących faz [9]:

1. *Koncept*. Na tym etapie następuje wybór projektu wraz z określeniem celu i trendu oraz określona zostaje deweloperska wizja początkowa produktu.
2. *Rozpoczęcie*. Działania mające na celu przygotowanie klienta, zespołu i środowiska do działania poprzez m.in. stworzenie zespołów, ustalenie zakresu odpowiedzialności, wizualizację wymogów oraz wstępnej architektury systemu.
3. *Dostarczanie* (ang. *delivery process*). Na tym szczeblu dostarczane są działające elementy systemu będące odpowiedzią na zmieniające się wymagania klientów. Proces ten składa się z następujących faz: 1) planowanie wydania (RP, ang. *release planning*) podczas którego następuje wczesne oszacowanie (EE, ang *early estimate*) wielkości zadań do wykonania (US, ang *User Stories*) oraz ustalenie w którym wydaniu będą realizowane [8], 2) planowanie iteracji (IP, ang. *iteration planning*) zawierający rozbicie US na zadania programistyczne (ang. *task*, zazwyczaj obejmujące zakres prac dla jednego programisty) oraz określenie ilości zadań na najbliższą iterację, 3) iteracja (ang. *iteration*) będące etapem realizowania zadań programistycznych poprzez pisanie testów, kodu, dziennym monitorowaniem postępu prac oraz naprawą defektów, 4) przegląd iteracji (IR, ang. *iteration review*) podczas którego następuje przegląd wykonanych US, przedstawienie wersji demonstracyjnej dla klienta oraz przeprowadzenie retrospektywy [10] w zespołach oraz 5) *wydanie* (ang. *release*), czyli przygotowanie paczki gotowego oprogramowania dla klienta [11].
4. *Produkcja*. Obsługa i wsparcie działającego w środowisku docelowym oprogramowania oraz naprawa potencjalnych błędów.
5. *Zakończenie*. Ukończenie prac nad projektem wraz ze wsparciem technicznym.

Fazy dostarczania elementów oprogramowania realizowane są w trzech pętlach: pętla wydania (RL, ang. *release loop*), pętla iteracji (IL, ang. *iteration loop*) oraz pętla codziennej pracy (DL, ang. *daily loop*) [11, 12]. Określenie ilości oraz długości każdej z pętli odbywa się na etapie rozpoczęcia projektu. Na każdym etapie planowania pracy uwzględniany jest nie tylko zakres prac na obecnymi poziomie, ale również na poziomie nadrzędnym [11].

W celu ujednolicenia nazewnictwa warto wprowadzić pojęcie pętli projektu (PL, ang *projekt loop*) obejmujące prace nad wszystkim US, które weszły w skład projektu.

W przedstawionym szczególnym przypadku problem szeregowania zadań pokrywa się z zagadnieniem planowania na najbliższą pętlę pracy. Klienci na początku określają główny cel projektu oraz interesujące ich US. Następnie projekt dzielony jest na wydania, w skład których wchodzą jedynie wybrane US wg ustalonych priorytetów. Najbardziej szczegółowym etapem planowania jest IP, gdzie zapada decyzja jakie zadania programistyczne będą realizowane w najbliższym czasie.

3. Przegląd literatury

Koncentrując się na kwestii szeregowania zadań przeprowadzono analizę aktualnych sformułowań problemu planowania pracy w zwinnych metodykach zarządzania projektami oraz opracowanych rozwiązań. Warto na tym etapie zauważyć, że w porównaniu do znacznej ilości prac dotyczących ustalania priorytetów [13], definiowania zależności [14] czy opisujących sposoby szacowania [15] jest niewiele prac zagłębiających temat szeregowania zadań w ujęciu planowania.

Rozpoczynając od sposobów formułowania problemu warto zwrócić uwagę na pracę [11]. Wyróżniono w niej podział planowania na dwie części: planowanie wydania i planowanie iteracji oraz dokonano klasyfikacji projektu na feature-driven (FD) i date-driven (DD). W projektach FD najważniejsze jest przede wszystkim zrealizowanie określonych US bez sztywnych ram czasowych. W podejściu DD określony jest czas zakończenia danego projektu, a jego celem jest zrealizowanie największej ilości US zapewniających wysoki poziom zadowolenia klienta [16, 17].

Przechodząc kolejno do zagadnienia planowania w pracy Szőke [12] został sformułowany model szeregowania na etapie IL mający na celu minimalizację czasu wykonania zadań. Model ten powstał poprzez rzutowanie projektu prowadzonego metodologią agile do NP-trudnego problemu RCPS (ang. *resource-constrained project scheduling problem*) [18]. Następnie w artykule Dennea [19] zaproponowano zastosowanie IFM (ang. *incremental founding method*) [20] przy użyciu podejścia heurystycznego do planowania pracy w celu maksymalizacji Net Presented Value [21]. W kolejnej pracy stworzonej przez Alencara [22] przedstawiono problem planowania kolejnego wydania jako zadanie programowania liniowego całkowitoliczbowego (PLC). Głównym kryterium optymalizacji była maksymalizacja dochodu. W pracy Karlssona [13] pokazano, że planowanie wydania oraz wybór wymagań mogą być wspierane przez wizualizację za pomocą diagramów UML, co będzie sprawdzało się w mniejszych projektach. Interesującym podejściem jest również Planning Game [23]. Priorytety US ustala się wg dwóch kategorii: funkcjonalności oraz ryzyka, gdzie w każdej kategorii zadania układa się na trzech stosach. Bazując na szacunkach czasowych, kalkulowana jest data ukończenia wydania, a klienci wybierają zadania bazując na ułożonych stosach [24]. W opisywanej technice wykorzystuje się algorytmy sortujące do podziału zadań na stosy [13]. W pracy Greera [25] przedstawiono propozycję na optymalne alokowanie zasobów dla każdej iteracji poprzez wykorzystanie algorytmu genetycznego.

Podsumowując, do problemu szeregowania zadań w projektach prowadzonych metodologiami *agile* można podejść na wiele sposobów. Analiza literatury wskazuje głównie na zainteresowanie badaczy planowaniem wydań oraz poszczególnych iteracji [12, 14]. Dodatkowo projekty mogą być ograniczone czasowo lub nastawione głównie na realizację konkretnych US [23]. Warto również zwrócić uwagę na kwestię przypisania ludzi

do zespołów oraz stopnia migracji [22]. Znalezienie optymalnego rozwiązania szeregowania zadań oraz alokacji zasobów należy do problemów NP-trudnych [12].

4. Charakterystyka wybranych modeli szeregowania

Na podstawie analizy literatury przedstawionej w rozdziale 3 wyłoniono trzy najbardziej interesujące podejścia do problemu planowania w zwinnych metodykach zarządzania projektami. Poniżej zostały przedstawione rozważania teoretyczne na temat wybranych metod w celu przygotowania autorskiego opisu problemu do dalszych badań.

4.1. Metoda RCPS

RCPS (*resource-constrained project scheduling*) jest problemem optymalizacji kombinatorycznej, gdzie rozważane są zasoby dyskretne o ograniczonej dostępności oraz zadań o estymowanym czasie trwania zależnym od zasobów wraz z powiązaniami przyczynowo-skutkowymi pomiędzy zadaniami [18]. Rozwiązanie problemu sprowadza się zwykle do wyznaczenia uszeregowania o minimalnej długości przez przypisanie do każdego zadania terminu rozpoczęcia oraz zasobów, tak aby zostały zachowane zależności pomiędzy zadaniami. W proponowanym podejściu uwzględnia się również aktualną dostępność zasobów.

Mając na uwadze wyżej podane informacje, pierwszą omawianą metodą jest podejście polegające na przeniesieniu zwinnego podejścia do problemu RCPS rozszerzonego o dwa dodatkowe warunki [12]. Pierwszym z nich jest przypisywanie z góry US do określonych zasobów przed rozpoczęciem procesu szeregowania, a drugim - określone czasowo iteracje (ang. *timeboxed*). Ogólny problem szeregowania dotyczy alokacji limitowanych zasobów (w rozważanym przypadku są to zasoby ludzkie – ang, *manpower*) do zadań w określonym z góry (zaplanowanym) czasie. Bazując na dotychczasowej analizie, optymalizacji można dokonać na wielu płaszczyznach, jednak głównym celem powinna być maksymalizacja zadowolenia interesariuszy przy zachowaniu najkrótszego możliwego terminu zakończenia projektu. W związku z powyższym określono jako główny cel algorytmu minimalizację całkowitego czasu wykonania projektu.

Kolejnym ważnym do zdefiniowana elementem są zasoby. W zwinnym podejściu zaleca się, aby zespoły traktować jako jednolitą strukturę, bez wyróżnionych ról, stąd w zastosowanym rozwiązaniu wyróżniono tylko jeden typ zasobów: deweloper. Dodatkowo warto zwrócić uwagę na fakt, że złożoność problemu wzrasta wraz z ilością zależności pomiędzy zadaniami. Zatem mamy

$$\mathrm{C} = \min S_i \,, \tag{1}$$

gdzie: i – iteracja,

S_i – uszeregowany wektor zadań.

Funkcja celu została przedstawiona w równaniu (1), gdzie S_n oznacza uszeregowany wektor zadań do wykonania na koniec i-tej iteracji, przy następujących ograniczeniach: 1) różnica pomiędzy startem zadania następującego a jego poprzednika powinna być większa niż wielkość precedensu pomiędzy tymi zadaniami, 2) wymagania zasobu nie mogą być większe niż całkowity jego zakres, 3) start kolejnej iteracji musi być poprzedzony zakończeniem poprzedniej.

Jako rozwiązanie problemu zaproponowano zmodyfikowaną wersję algorytmu LTP (ang. *longest proccessing time first*), czyli AF (ang. *assined task first*). Za pomocą algorytmu

AF uszeregowano najpierw zadania, które miały już przypisane zasoby, co pełniło uzupełnienie do decyzji podjętych podczas IP. Złożoność obliczeniowa zaproponowanego rozwiązania wynosi O(n+m), co pozwala na wykorzystanie w rzeczywistych (praktycznych) aplikacjach.

4.2. Algorytmy genetyczne

Algorytmy genetyczne są analogią do naturalnych procesów ewolucyjnych występujących w przyrodzie [26]. Wykorzystywane są do rozwiązywania problemów NP-trudnych, których nie da się sprowadzić do problemów wielomianowych. Badania pokazują, że genetyczne podejście pozwala na znalezienie wysokiej jakości rozwiązania optymalnego lub bliskiego optymalnemu nawet dla zagadnień na dużą skalę.

Zastosowanie algorytmów genetycznych jako wsparcie w planowaniu w zwinnych metodykach projektowych zostało zaproponowane w [25] i zaimplementowane na poziomie IL, tak aby wynik uwzględniał czas ukończenia projektu. W tym celu zasoby i wymagania dla danej iteracji powinny być narzucane z góry i poddawane ponownemu planowaniu po każdym cyklu. Podczas nowego planowania zaleca się aktualizację priorytetów zadań oraz wprowadzanie nowych ograniczeń i wymagań. Powyższe podejście, poprzez umożliwienie wprowadzania zmian w danej inkrementacji, wymaga często aktualizacji w późniejszych, niezrealizowanych iteracjach.

Badany algorytm jest kombinacją kar i nagród mających na celu znalezienie rozwiązania optymalnego. Kary są zdefiniowane jako stopień odchyleń pomiędzy monotonicznymi właściwościami wymagań. W przypadku dwóch wymagań monotoniczne właściwości są bardziej satysfakcjonujące, jeżeli wymaganie jest oceniane jako bardziej obiecujące od drugiego. Jak wynika z równania (2) dążymy do minimalizacji całkowitej wartości kar p_i, dla i-tej iteracji:

.

$$A = \min \Sigma_{i=k}^{n} p_i, \tag{2}$$

gdzie: i – jak wyżej,
k – aktualna realizowana iteracja,
n – maksymalna ilość iteracji,
p – wartość kar.

Kolejny krok to określenie wartości nagród. Korzyści z przypisywania indywidualnych wymagań dla każdej iteracji są wyrażane przez różnicę pomiędzy najmniejszym możliwym priorytetem US a aktualnym zadanym przez interesariuszy w odniesieniu do ich iteracyjnej wartości przypisania. Im wyższa otrzymana wartość produktu, tym wcześniej dane wymagania pojawiają się w etapie produkcji, co ma pozytywny wpływ na finałową wartość biznesową. Funkcja celu w przypadku analizy nagród jest funkcją maksymalizacją (3), gdzie b_i określa benefity dla i-tej iteracji.

$$\mathrm{B} = \max \Sigma_{i=k}^{n} b_i, \tag{3}$$

gdzie: i, n, k – jak wyżej,
b – wartość nagród.

Znając wartości kar i nagród, ostateczną postać funkcji celu można zdefiniować jako maksymalizację liniowej kombinacji funkcji (2) i (3) dla z góry określonej wartości α.

Funkcja ta została przedstawiona w równaniu (4), gdzie α wyraża wagę danych wymagań dla interesariuszy. Wszystkie rozwiązania uzyskane z opisywanej metody są Pareto optymalne. Głównym celem maksymalizacji funkcji (4) jest przeprowadzenie operacji krzyżowania i mutacji, stąd funkcja ta jest wywoływana na każdym kroku optymalizacji wykonywanym przez algorytm genetyczny. Algorytm kończy swoje działania, gdy nie ma dalszej poprawy w szukanych rozwiązaniach.

$$C(\alpha) = \max((\alpha - 1)A + \alpha B), \tag{4}$$

gdzie: α – wagi wymagań interesariuszy,
A – równanie (2),
B – równanie (3).

Rozwiązania znalezione za pomocą opisywanego algorytmu są optymalne lub bliskie optymalnym. Fakt ten spowodowany jest tym, że z zasady algorytmy genetyczne nie gwarantują znalezienia rozwiązania optymalnego. Można zaobserwować dużą zmienność wyników w zależności od różnych uruchomień aplikacji. Zmiany te dotyczą zarówno wartości wskaźników mutacji jak i krzyżowania. Jednakże po przeprowadzeniu empirycznych doświadczeń istnieje wysokie prawdopodobieństwo znalezienia satysfakcjonującego rozwiązania, szczególnie dla większej liczby obliczeń z wysoce zróżnicowanymi parametrami.

4.3. Podejście hybrydowe oparte na programowaniu liniowym całkowitoliczbowym

Programowanie liniowe całkowitoliczbowe należy do grupy dokładnych metod optymalizacji dyskretnej, w której zadanie zdefiniowane jest jako funkcja liniowa, a zestaw ograniczeń liniowych przyjmuje wartości całkowitoliczbowe [27].

Rozwiązanie hybrydowe zaproponowane przez [28] polega na połączeniu dwóch podejść: 1) *art of RP* - bazującym na ludzkiej intuicji, 2) *science of RP* - matematycznym sformalizowaniu problemu i zastosowaniu komputerowych modeli do generowania lepszego rozwiązania. W zwinnych metodykach projektowania podczas planowania kolejnych iteracji polega się przede wszystkim na podejściu ludzkim. PO (właściciele produktu) po każdej iteracji rozmawiają ze wszystkimi interesariuszami, wpisują otrzymane dane do arkuszy kalkulacyjnych i bazując na swojej wiedzy oraz intuicji planują kolejną iterację [23]. Niestety wzrost ilości interesariuszy, zespołów oraz długości projektu powodują, że opisywany problem staje się bardzo złożony. W celu pomocy właścicielom produktu w planowaniu kolejnych iteracji podjęto próbę sformalizowania problemu.

Wynikiem planowania jest uszeregowanie będące mieszaniną różnych aspektów, takich jak priorytet, ryzyko, satysfakcja czy zwrot z inwestycji. W celu znalezienia balansu pomiędzy wyżej wymienionymi aspektami wprowadzono następujące założenia: 1) wartość zadowolenia została zdefiniowana jako suma priorytetów WAS(j, m) (ang. *weighted average satisfaction*) ustalonych przez wszystkich interesariuszy dla j-tego zadania w m-tym wydaniu, 2) każdą wartość WAS (j,m) określa się jako średnią z dwóch wymiarów ustalania priorytetów: priorytet wg pilności oraz wg ważności dla interesariuszy, 3) pilność jest określana przez każdego interesariusza dla każdego US, 4) suma wymagań dla danego typu zasobu nie może przekraczać całkowitej przepustowości.

Bazując na przedstawionych założeniach, celem badań jest maksymalizacja funkcji F(x) dla wszystkich RP (określonych symbolem x) z zastrzeżeniem istnienia ograniczonych zasobów oraz zależności pomiędzy zadaniami. Funkcja celu wygląda następująco:

$$C = \max F(x), \tag{5}$$

gdzie:

$$F(x) = \Sigma_{m=1\ldots M}\Sigma_{j:x(j)=m} WAS(j,m), \tag{6}$$

gdzie: j – zadanie,
m – wydanie,
x – pętla wydania,
WAS - wartość zadowolenia.

Przedstawione wyżej sformułowanie problemu stanowi formę zadania PLC. Cele i ograniczenia sformułowane zostały jako funkcje liniowe a zmienne decyzyjne jajko liczby całkowite. Wykorzystując podejście heurystyczne generowane są serie rozwiązań zbliżonych do siebie jakościowo, ale różniących się od siebie strukturą. Dzięki takiemu podejściu ostateczna decyzja, jak będzie wyglądało następne wydanie należy do PO. To właśnie właściciel produktu na podstawie swojej wiedzy i doświadczenia podejmuje ostateczną decyzję, które proponowane rozwiązanie z sugerowanej serii jest najlepsze

5. Opis problemu szeregowania zadań

Niniejszy rozdział zawiera autorski, formalny zapis problemu szeregowania zadań w zwinnych metodykach zarządzania projektami. Zapis ten jest wariancją rozwiązań zaproponowanych w przytaczanej literaturze rozszerzony o dodatkowe założenia. Główne zadanie, to zdefiniowanie funkcji celu oraz ograniczeń, tak aby stanowiło podstawę do dalszych badań. W celu sformułowania modelu szeregowania należy najpierw wprowadzić listę pojęć i przedstawić zależności pomiędzy nimi:

Wydanie. Projekt składa się z określonej liczby wydań R, które posiadają datę rozpoczęcia oraz czas trwania wyrażony w ilości iteracji. Warunkiem rozpoczęcia kolejnego wydania jest ukończenie poprzedniego. Na początku każdego wydania odbywa się planowanie, mające na celu określenie zakresu prac na podstawie wczesnych oszacowań.

Iteracja. Iteracja *i* to trwający od 1-4 tygodni okres wytwarzania oprogramowania przez zespoły. Głównym celem jest ukończenie zaplanowanych na ten okres US. Na początku każdej iteracji zespoły deweloperskie rozbijają US na zadania programistyczne i dokonują dokładnych szacunków wielkości US. Czas trwania pojedynczej iteracji wyrażany jest w dniach roboczych.

Zespół. Zespół deweloperski *team* odpowiadający za realizację US. Wielkość zespołu wyrażana jest w osobodniach/iteracja.

US/Feature. Zadanie do zrealizowania *f*, określone przez interesariuszy. Zadanie ma ustalony priorytet, listę zależności pomiędzy innymi zadaniami oraz przepustowość – czyli ile maksymalnie lub minimalnie osobodni można poświęcić na dany US w iteracji.

Priorytet. Priorytet *w(f)* określający, jak ważny jest dany US dla interesariuszy na następujących poziomach: wysoki, średni, niski.

Szeregowanie zadań odbywa się na początku wydania, kiedy na podstawie szacunków podanych przez ekspertów ustala się wstępny plan pracy na dane wydanie. Stworzony plan ma na celu pomoc w ustaleniu ile i jakie US zostaną zrealizowane. Podczas trwania wydania po każdej iteracji odbywa się aktualizacja danych (m.in. podawany jest bieżący czas do ukończenia zadania oraz alokacja czasowa dokonana przez zespoły), a następnie

kalkulowany jest nowy plan na pozostałą część wydania. Opisane postępowanie ma na celu pomoc interesariuszom w lepszym prognozowaniu ewentualnych opóźnień lub przestojów.

W opisywanym systemie wielkość US określana jest w osobodniach. US mogą być zadaniami niezależnymi, pojawiającymi się w systemie lub posiadać jeden z dwóch typów zależności: 1) *c* - zadania, które trzeba wykonać równolegle, 2) *p* – zadania, których wykonanie powinno się odbyć w określonej kolejności (lista poprzedników). Zadanie rozpoczęte w i-tej iteracji musi być kontynuowane w iteracji i+1 i kolejnych, aż do ukończenia (przerwy w realizacji zadań są niedozwolone). Każde zadania mają również zdefiniowany minimalny $time_{min}(f)$ i maksymalny $time_{max}(f)$ czas, który zespół może poświęcić na pracę nad danym US w iteracji. Celem takiego zabiegu jest zapobieganie sytuacji, w których przydzielone zostanie zbyt mało lub za dużo zasobów. Zaistnienie pierwszego przypadku spowodowałoby bardzo powolną i niedokładną realizację zadania, a w drugim przypadku generowałyby się wolne przebiegi. Suma czasów *time(f,t)* alokowanych w czasie wszystkich iteracji musi być równa całkowitemu czasowi US $time_{total}(f)$ (równianie 7).

$$\Sigma_i \Sigma_{team} time(f, i, team) = time_{total}(f), \tag{7}$$

gdzie: i – iteracja,
f – zadanie do zrealizowania,
team – zespół deweloperski,
time – czas wykonania zadania.

Zespół może w każdej iteracji pracować nad jednym lub wieloma US. Warunkiem koniecznym w tym przypadku jest nieprzekroczenie maksymalnego czasu przydzielonego dla danego zespołu *Cap (team)*:

$$\Sigma_f time(f, i, team) \leq Cap(team) \tag{8}$$

gdzie: i, f, time, team – jak wyżej,
Cap – pojemność.

Uwzględniając wyżej wymienione ograniczenia sformułowano funkcję celu dla planowania wydania w następującej postaci:

$$C = max \Sigma_f \; w(f), \tag{9}$$

gdzie: f – jak wyżej,
w(f) - priorytet zadania *f*.

Z równania (9) wynika, że głównym celem każdego wydania jest realizacja jak największej liczby US o wysokim priorytecie dla interesariuszy. W momencie ustalenia zakresu pracy na dane wydanie rozpoczyna się praca w iteracjach. Podejście to wymaga modyfikacji funkcji celu do następującej postaci:

$$C = \begin{cases} \min \quad t_{całkowity} \\ \\ \min \sum_f^{f \epsilon E} w(f) \end{cases} \tag{10}$$

gdzie: f – jak wyżej, E – zadania które nie zmieściły się w planowanym czasie wydania, $t_{całkowity}$ – całkowity czas realizacji wszystkich zadań w danym wydaniu.

Jak wynika z równania (10) planowanie iteracyjne powinno przede wszystkim uwzględniać realizację zaplanowanych US w jak najkrótszym czasie i minimalizację strat spowodowanych opóźnieniami lub błędami w szacunkach.

Przedstawiony wyżej opis problemu dotyczy planowania zadań w projektach IT prowadzonych z wykorzystaniem metodyk agile. Główną różnicą w przedstawionym problemie (w porównaniu do opisów z rozdziału 4) jest uwzględnienie faktu, że w danej iteracji nad konkretnymi US pracować może ograniczona liczba osób. Przypisanie większej ilości zasobów spowodowałoby przestoje w produkcji oprogramowania i powstawanie opóźnień. Opisywany model uwzględnia również priorytety interesariuszy, ale nie proponuje modelu ich kalkulacji – jest to zagadnienie do dalszych rozważań.

6. Konkluzja

Badania i rozważania podjęte w pracy nie wyczerpują całokształtu problematyki, a wręcz są jedynie wstępem do dalszej pracy. Kolejnym krokiem, jaki można podjąć jest wybranie algorytmu szeregującego rozwiązującego przedstawiony problem. Bazując na podstawie analizy dokonanej w rozdziale 4 można zastosować algorytmy genetyczne lub podjąć próbę sformułowania problemu w postaci zadania PLC.

Na zakończenie rozważań należy przypomnieć, że zdefiniowanie problemu planowania zadań w zwinnych metodykach projektowych oraz poszukiwanie jego rozwiązania ma na celu dostarczenie narzędzi do wspierania decyzji, z którymi kierownicy produktu spotykają się w codziennej pracy. Narzędzie to nie powinno jednak zastąpić doświadczenia i intuicji osób prowadzących projekt.

Literatura

1. Cockburn, A.; Highsmith, J.: Agile software development: the business of innovation. 2001, 34 (9), 120-127.
2. Davis, B.: Agile Practices for Waterfall Projects: Shifting Processes for Competitive Advantage; J. Ros Pub.: Plantation, FL, 2013.
3. Dingsøyr, T.; Dybå, T.; Pekka, A.: A Preliminary Roadmap for Empirical Research on Agile Software Development., 2008.
4. 11th Annual State of Agile Survey | The Largest, Longest-Running Agile Survey; VersionOne, 2016.
5. Manifesto for Agile Software Development. http://agilemanifesto.org/ (accessed Dec 30, 2016).
6. Declaration of Independence. https://pmdoi.org/ (accessed Dec 30, 2016).
7. Ambler, S.: Scaling agile software development through lean governance. SDG '09 Proceedings of the 2009 ICSE Workshop on Software Development Governance, May 17, 2009, 1-2.
8. Chrapko, M.: SCRUM. O zwinnym zarządzaniu projektami; Helion, 2013.
9. The Agile System Development Life Cycle (SDLC). www.ambysoft.com (accessed Dec 30, 2016).
10. Rubin, K.: Essential Scrum: a practical guide to the most popular agile process; Addison-Wesley, 2012.
11. Cohn, M.: Agile Estimating and Planning; Prentice Hall PTR: Upper Saddle River, 2009.
12. Szőke, Á.: Decision Support for Iteration Scheduling in Agile Environments. Lecture Notes in Business Information Processing Product-Focused Software Process Improvement 2009, 156-170.

13. Karlsson, L.; Thelin, T.; Regnell, ; Berander, P.: Pair-wise Comparisons versus Planning Game Partitioning—experiments on Requirements Prioritisation Techniques. Empirical Software Engineering 12.1 2006, 3-33.
14. Carlshamre, P.: An industrial survey of requirements interdependencies in software product release planning. Requirements Engineering, 2001.
15. Boehm, B.; Madachy, R.; Steece, B.: Software cost estimation with Cocomo II with Cdrom. Prentice Hall PTR 2000.
16. Ambler, S.: Feature Driven Development (FDD) and Agile Modeling. http://agilemodeling.com/ (accessed Jan 05, 2017).
17. Schraml, T.: Date-Driven Development. http://www.dbta.com/ (accessed Jan 5, 2017).
18. Demassey, S.: Resource-Constrained Project Scheduling. In Resource-Constrained Project Scheduling, Models, Algorithms, Extensions and Applications; Polytech Tours: Toulouse, 2008; pp 21-35.
19. Denne, M.: The incremental funding method: Data-driven software development. IEEE software 21.3 2004, 39-47.
20. Alencar, A.: Analysing the incremental funding method and its software project scheduling algorithms. International Journal of Information Processing 2013, 7 (2), 30-38.
21. Kum Khiong, Y.; Talbot, B.; Patterson, J.: Scheduling a project to maximize its net present value: an integer programming approach. European Journal of Operational Research 1993, 64 (2), 188-198.
22. Marijan von Akker: Software product release planning through optimization and what-if analysis. Information and Software Technology 2001, 50 (1), 101-111.
23. Beck, K.: Extreme programming explained: embrace change; Addison-Wesley Professional, 2000.
24. Martin, R. C.: Agile software development: principles, patterns, and practice; Prentice Hall PTR, 2003.
25. Greer, D.; Ruhe,.: Software release planning: an evolutionary and iterative approach. Information and software technology 2004, 46 (4), 242-253.
26. Jong, K. A. D.: An Analysis of the Behaviour of a Class of Genetic Adaptive Systems. PhD Thesis, University of Michigan 1975.
27. Smutnicki, C.: Algorytmy szeregowania zadań; Politechnika Wrocławska: Wrocław, 2012.
28. Gunther, R.; Saliu,.: The art and science of software release planning. IEEE software 2005, 22 (6), 47-53.

Mgr inż. Mateusz RUSSAK
Mgr inż. Paulina RUSSAK
Katedra Informatyki Technicznej - W4/K9
Politechnika Wrocławska
ul. Janiszewskiego 11/17
50-372 Wrocław
tel: (0-48) 513 318 105
e-mail: paulina.russak@pwr.edu.pl
mateusz.russak@pwr.edu.pl