POLITECHNIKA KOSZALIŃSKA
Wydział Elektroniki i Informatyki

mgr inż. Włodzimierz Wysocki

Agentowo–obiektowy model wsparcia procesu wytwarzania oprogramowania

Rozprawa doktorska

Promotor:
prof. Cezary Orłowski

Promotor pomocniczy:
dr hab. prof. nadzw. PK
Grzegorz Bocewicz

Koszalin 2017

Motto

Czego nie mogę zbudować,
tego nie mogę zrozumieć.

Richard Feynmann 1988

Stworzenie jakiejś rzeczy
oraz stworzenie
i pełne zrozumienie idei tej rzeczy
są często aspektami tego samego,
niepodzielnego procesu.

Paul K. Feyerabend [31]

Spis treści

Spis treści **5**

Spis symboli i oznaczeń **9**

Wprowadzenie **13**

1. Ocena procesu wytwarzania oprogramowania – stan badań **19**

1.1. Stan badań oceny procesów wytwarzana oprogramowania **19**

1.1.1. Tradycyjne metodyki wytwarzania 19

1.1.2. Zwinne metodyki wytwarzania 24

1.1.3. Zwinna transformacja 29

1.1.4. Porównanie metodyk tradycyjnych i zwinnych 29

1.1.5. Modele zwinnej transformacji organizacji IT 31

1.1.6. Modele dojrzałości organizacji IT 34

1.2. Podsumowanie stanu badań **37**

1.2.1. Tradycyjne podejście do oceny procesów wytwarzania 38

1.2.2. Koncepcja oceny procesów wytwarzania wykorzystująca model 39

1.2.3. Planowanie procesu wytwarzania oparte na modelu 40

2. Układ eksperymentu dla oceny procesu wytwarzania oprogramowania **43**

2.1. Wprowadzenie **43**

2.2. Analiza środowiska projektowego i procesu wytwarzania **44**

2.2.1. Zmienne wejściowe procesu wytwarzania oprogramowania 44

2.2.2. Zmienne wyjściowe procesu wytwarzania oprogramowania 45

2.2.3. Analiza procesu wytwarzania oprogramowania 47

2.3. Układ eksperymentu dla oceny procesu wytwarzania **52**

2.3.1. Zmienna wejściowa zakres projektu 52

2.3.2. Zmienna wejściowa metodyka wytwarzania 53

2.3.3. Zmienna wejściowa plan projektu 57

2.3.4. Zmienna wejściowa zespół projektowy 60

2.3.5. Zmienne wyjściowe opisujące wykonane prace 60

2.4. Podsumowanie układu eksperymentu **62**

3. Model podstawowy i uproszczony procesu wytwarzania oprogramowania **65**

3.1. Wprowadzenie **65**

3.2. Koncepcja budowy modelu procesu wytwarzania oprogramowania **65**

3.2.1. Budowa rdzenia modelu podstawowego 66

3.2.2. Interakcja elementów wewnętrznych rdzenia modelu ... 71
3.2.3. Rozszerzenia rdzenia modelu ... 76
3.3. Analiza dziedzinowa modelu podstawowego ... 87
3.4. Dobór metody modelowania ... 88
3.4.1. Przegląd metod modelowania ... 89
3.4.2. Koncepcja zastosowania wybranej metody modelowania ... 92
3.5. Analiza możliwości weryfikacji zasadności modelu uproszczonego ... 93
3.5.1. Weryfikacja replikatywna ... 93
3.5.2. Weryfikacja prognostyczna ... 94
3.6. Podsumowanie ... 95
4. Budowa uproszczonego modelu procesu wytwarzania oprogramowania ... 97
4.1. Model procesu wytwarzania w widoku ogólnym ... 97
4.1.1. Model uproszczony w postaci czarnej skrzynki ... 97
4.1.2. Model uproszczony w postaci białej skrzynki ... 100
4.2. Model procesu wytwarzania w widoku szczegółowym ... 102
4.2.1. Środowisko agentowe i jego elementy ... 102
4.2.2. Agenty ... 104
4.3. Implementacja modelu AGOMO w systemie wieloagentowym ... 111
4.4. Metoda oceny procesów RUP ... 112
4.4.1. Etap pierwszy – określenie wartości zmiennych wejściowych modelu ... 114
4.4.2. Etap drugi – przetworzenie informacji o wykonanych zadaniach ... 115
4.4.3. Etap trzeci – dostrojenie modelu do badanego projektu ... 115
4.4.4. Etap czwarty – symulacja procesu wytwarzania ... 115
4.4.5. Etap piąty – ocena procesu wytwarzania oprogramowania ... 116
4.5. Podsumowanie i ocena konstrukcji modelu uproszczonego oraz opracowanej metody ... 119
5. Weryfikacja replikatywna modelu AGOMO ... 121
5.1. Wprowadzenie ... 121
5.2. Eksperyment badawczy dla weryfikacji zasadności replikatywnej modelu ... 121
5.2.1. Badana organizacja IT i projekt ... 121
5.2.2. Plan eksperymentu weryfikacji zasadności replikatywnej modelu ... 121
5.2.3. Realizacja etapu pierwszego – ustalenie wartości zmiennych wejściowych ... 122
5.2.4. Realizacja etapu drugiego – przetworzenie zadań projektu UBEZP ... 126
5.2.5. Realizacja etapu trzeciego – dostrojenie modelu ... 126
5.2.6. Realizacja etapu czwartego – symulacja procesu wytwarzania i weryfikacja replikatywna modelu ... 127
5.2.7. Realizacja etapu piątego eksperymentu – porównanie badanego procesu z wzorcem 130

5.3. Podsumowanie – znaczenie wyników eksperymentu i procesów weryfikacji replikatywnej 131

6. Weryfikacja prognostyczna modelu 133

6.1. Wprowadzenie 133

6.2. Metoda planowania procesów RUP 133

6.2.1. Etap pierwszy metody – ustalenie wartości zmiennych wejściowych modelu 134

6.2.2. Etap pierwszy metody – wybranie projektu referencyjnego 134

6.2.3. Etap trzeci metody – dostrojenie modelu do projektu referencyjnego 135

6.2.4. Etap czwarty metody – Symulacja procesu RUP 135

6.3. Eksperyment badawczy w procesie weryfikacji prognostycznej 135

6.3.1. Etap pierwszy eksperymentu – określenie wartości zmiennych wejściowych modelu 136

6.3.2. Etap drugi eksperymentu – wybór projektu referencyjnego 137

6.3.3. Etap trzeci eksperymentu – dostrojenie modelu AGOMO do wybranego projektu referencyjnego 138

6.3.4. Etap czwarty eksperymentu – wykonanie symulacji procesu wytwarzania oprogramowania opartego na modelu AGOMO 139

6.3.5. Etap piąty eksperymentu – porównanie wyniku symulacji z przebiegiem badanego projektu 139

6.4. Podsumowanie rozdziału – znaczenie procesów weryfikacji prognostycznej modelu AGOMO 144

7. Podsumowanie i wnioski 147

7.1. Dalsze prace nad modelem 149

8. Spis literatury 153

Spis rysunków 159

Spis tabel 161

Dodatek A. Rozmiar oprogramowania 163

1. Obliczanie wartości AGOMO–UCP na podstawie modelu przypadków użycia 163

Czynnik AGOMO–UUCW 163

Czynnik AGOMO–TCF 164

Czynnik AGOMO–ECF 164

Dodatek B. Aplikacja AGOSIM–APP 167

1. Instalacja aplikacji 167

2. Uruchomienie 167

3. Funkcje aplikacji 167

Praca z projektami 168

Wprowadzanie parametrów wejściowych symulacji 168

Uruchomienie symulacji **171**
Analiza wyników **172**
Wyniki sumaryczne **173**
Zawartość repozytorium **174**
Harmonogram prac **175**
Obciążenie pracą **176**
Koszty prac w projekcie **177**
Dodatek C. Aplikacja AGOCOMP–APP **179**
1. Instalacja aplikacji **179**
2. Uruchomienie **179**
3. Funkcje aplikacji **179**
Dodatek D. Interaktywne wykresy **181**
1. Powiększenie obszaru wykresu **181**
2. Przesuwanie powiększonego obszaru po wykresie **181**
3. Zmiana powiększenia **181**
4. Przywrócenie początkowego widoku **181**
5. Opcje dodatkowe **181**

Spis symboli i oznaczeń

Symbol	Opis	Równanie Strona Algorytm
A	Zbiór typów artefaktów	3.1
AGOCOMP–APP	Aplikacja wspierająca metodę AGOMAS	str. 130
AGOMAS	Metoda oceny procesu wytwarzania (ang. *AGent–Object Model software process ASsessment method*)	str. 97
AGOMO	Agentowo–obiektowy model procesu wytwarzania oprogramowania	str. 97
AGOPLAN	Metoda planowania procesu wytwarzania	str. 133
AGOSIM–APP	Aplikacja wspierająca model AGOMO – symulator wraz z interfejsem graficznym i narzędziami do analizy	str. 112
ao_i	Instancja artefaktu	3.12
ap_i	Zbiór typów artefaktów wyjściowych, produkowanych przez *i*–ty typ zadania $ap_i \subseteq A$	3.4
aw_i	Zbiór typów artefaktów wejściowych *i*–tego typu zadania $aw_i \subseteq A$	3.4
b	Zachowanie agenta	4.15
$b_{i,k}$	K–te zachowanie i–tego agenta	4.14
ci	Numer bieżącej iteracji, $ci \in N$	Alg. 3.1
cv	Zbiór bieżących działań	Alg. 3.1
D	Zbiór dyscyplin metodyki	2.2
d_i	Dyscyplina	?
E	Zbiór etapów metody	2.2
EE_i	Zbiór krawędzi grafu e_i, $EE_i \subset VE_i \times VE_i$.	2.4
e_i	Etap *i*–tej iteracji planu PP^S, $e_i \in E$ metodyki wytwarzania M	2.11
e_i	Przepływ działań *i*–tego zdarzenia codziennego	2.13
e_i	*I*–ty etap metodyki M zawierający przepływ działań.	2.3
e_i	Etap *i*–tej iteracji planu PP, $e_i \in E$ metodyki wytwarzania M	2.7
er_i	Informacja o błędzie, $er_i \in \{0,1\}$	3.12
ez	Współczynnik efektywności strategii komunikacyjnych wewnątrz zespołu	3.9
ezs	Przeskalowany współczynnik efektywności strategii komunikacyjnych wewnątrz zespołu	3.9
F	Zbiór stanów końcowych	4.15
G	Zbiór agentów	4.13
g_i	I–ty agent zbioru G	4.14
Ii	*I*–ta iteracja planu PP	2.6
I^S	*I*–te zdarzenie planu projektu metodyki Scrum zapisane w postaci iteracji	2.10
I^S	Zdarzenia planu projektu zapisane w postaci iteracji	2.9

K	Zbiór kategorii działań metody	2.2
K	Zbiór kategorii działań	2.17
k_i	I–ta kategoria zbioru K	2.17
KM	Zbiór kompetencji	3.5
km_i	I–ta kompetencja zbioru KM	3.5
kmw_n	Wartość n–tej kompetencji i–tej osoby, $kmw_n \in \{1,2,3,4,5\}$	3.6
L	Zbiór etykiet	4.15
le	Liczba etapów metodyki M,	2.3
lgb_i	Liczba zachowań i–tego agenta	4.14
li	Liczba iteracji planu projektu PP	2.6
lk	Liczba kategorii zbioru K	2.17
lkm	Liczba kompetencji	3.5
lko_i	Liczba działań i–tej kategorii	2.18
lo	Liczba działań metodyki M	2.5
lot_i	Liczba typów zadań i–tego działania	2.5
lp	Liczba osób w zespole projektowym	2.14
lr_i	Liczba ról projektowej i–tego członka zespołu projektowego	2.15
$lrpw$	Liczba zmian wartości zmiennych wejściowych modelu	3.11
lw	Liczba wykonanych prac	2.16
lwr	Liczba wykonanych prac w badanym projekcie	4.17
lws	Liczba wykonanych prac w modelu	4.18
lzc	Liczba zdarzeń codziennych planu projektu PP^S.	2.12
M	Metodyka wytwarzania	2.2
O	Zbiór działań metodyki	2.2
$o_{i,j}$	J–te działanie i–tej kategorii zbioru K	2.18
$o_{i,k}$	Działania $o_{i,k} \in T$ metodyki M	2.5
pd_i	Data i czas i–tej zmiany w umownych jednostkach czasu	3.11
P_i	I–ty członek zespołu projektowego Z	2.14
P_i	I–ta osoba zespołu projektowego	3.6
p_i	Zbiór instancji artefaktów utworzonych na podstawie instancji artefaktu ao_i	3.2
PP	Plan projektu	2.6
PP^S	Plan projektu dla metodyki Scrum	2.9
pw_i	Nowa wartość zmiennej wejściowej pz_i i–tej zmiany	3.11
pz_i	Zmienna wejściowa i–tej zmiany	3.11
R	Zbiór ról projektowych	2.2
$r_{i,k}$	K–ta rola projektowa i–tego członka zespołu projektowego	2.15
RP	Repozytorium	3.3
RPW	Rejestr parametrów wejściowych	3.11
S	Skończony zbiór nazywany stanem	4.15
s_0	Stan początkowy	4.15
s_i	Zbiór instancji artefaktów źródłowych, na podstawie których utworzono instancję artefaktu ao_i	3.2
SZZ	System zarządzania zmianą	str. 60
T	Zbiór elementarnych typów zadań metody	2.2

T	Zbiór przejść	4.15
t_c	Całkowity czas wykonania zadania	3.8
t_i	Czas trwania *i*–tej iteracji planu PP^S liczony w dniach i godzinach.	2.11
t_k	Czas trwania komunikacji	3.8
t_k	Czas wymagany na komunikację	3.10
t_{sk}	Parametru modelu określającego średni czas wymagany na komunikację	3.10
tw_i	Wynikowy czas wykonania i–tego typu zadania	3.7
t_z	Czas wykonania zadania właściwego	3.8
u.j.t.	Umowne jednostki czasu przedstawiające czas symulacji	str. 68
VE_i	Zbiór wierzchołków grafu e_i, $VE_i \subset O$ metodyki M,	2.4
W	Wykonane prace	2.16
w_i	I–te zadanie wykonanych prac	2.16
X	Harmonogram prac	2.20
x_i	Czas rozpoczęcia *i*–tego zadania w_i	2.20
Z	Zespół projektowy	2.14
ZC	Zbiór zdarzeń codziennych planu projektu PP^S	2.12
zc_i	*I*–te zdarzenie codzienne	2.12
zc_i	I–te zdarzenie codzienne metodyki Scrum	2.13
zcr_i	Czas rozpoczęcia *i*–tego zdarzenia codziennego	2.13
zct_i	Czas trwania *i*–tego zdarzenia codziennego	2.13
$Zd_{i,n}$	Wartość procentowa 0–100% dla dyscypliny *n*, odnosząca się do zakresu projektu ZP do realizacji w *i*–tej iteracji planu PP	2.8
z_i	Procentowa wartość zakresu projektu ZP do realizacji w *i*–tej iteracji planu PP	2.7
zn	Zakres nowej części projektu w UCP	4.15
zo	Zakres projektu w UCP do odzyskania z poprzedniej wersji systemu	4.15
ZP	Zakres projektu informatycznego	2.1
$z^{RUP}{}_i$	Procentowy zakres prac z_i^{RUP} zaplanowanych do realizacji w *i*–tej iteracji	2.8
γ	Funkcja wyszukująca w repozytorium instancji artefaktu typu $a \in A$ i tworząca instancję, jeśli nie została znaleziona	3.3
ζ_{tz}	Funkcja sprawdzająca, czy typ zadania tz jest zakończony	3.3
ζ_v	Funkcja sprawdzająca, czy działanie v jest zakończone	Alg. 3.1
τ	Funkcja tworząca w repozytorium instancję artefaktu typu $a \in A$	3.3
$\varphi_{a,t}$	Funkcja określająca liczbę instancji artefaktów typu $a \in A$ w dyskretnej chwili czasu t	3.3
$\omega_{a,tz,t}$	Funkcja wyszukująca instancje artefaktów typu $a \in A$ nieprzetworzonych przez zadania typu $tz \in T$ w dyskretnej chwili czasu t	3.3

Wprowadzenie

Ocena procesu wytwarzania oprogramowania ma zasadnicze znaczenie zarówno dla ulepszania samego procesu, jak i poprawy jakości wytworzonego oprogramowania. Większość organizacji wytwarzających oprogramowanie używa i stosuje metodyki umożliwiające wsparcie i ulepszanie procesów wytwórczych, ponieważ badania empiryczne pokazały, że jakość procesu wytwarzania jest bezpośrednio związana z produktywnością organizacji i wysoką jakością wytworzonego oprogramowania. Dotychczasowa potrzeba oceny istniejących w organizacji procesów wytwarzania wzrosła w związku z dużym zainteresowaniem zwinną transformacją organizacji. Wzrost ten tłumaczy się związkiem między dojrzałością procesów wytwarzania organizacji a ryzykiem zwinnej transformacji. Próby transformacji organizacji o niskim poziomie dojrzałości procesów wytwarzania są obarczone wysokim ryzykiem, ponieważ organizacji brakuje wiedzy o strategicznych celach, która sterowałyby wprowadzeniem zwinnych metodyk.

Luka badawcza

Do oceny procesów wytwarzania zwykle stosowany jest model dojrzałości CMMI. Jednak jego zastosowanie jest związane z dużymi kosztami wynikającymi z potrzeby przygotowania odpowiedniej dokumentacji oraz zatrudnienia certyfikowanych ekspertów do przeprowadzenia oceny dojrzałości. Tradycyjne podejścia, polegające na ręcznej ocenie jakościowej, są nieefektywne, ponieważ są pracochłonne, ograniczone przez kompetencje ekspertów i nacechowane subiektywizmem. Ponadto ocena za pomocą modelu CMMI, skupiając się na realizacji celów w obszarach procesowych, nie bierze pod uwagę wpływu struktury zespołu projektowego na proces wytwarzania oprogramowania.

W celu przezwyciężenia ograniczeń tradycyjnych metod oceny zaproponowano podejście oparte na zastosowaniu modelu procesu wytwarzania oprogramowania. Ocena w tym podejściu polega na porównaniu badanego procesu wytwarzania z wzorcowym, wygenerowanym przez model. Dzięki temu otrzymane rozwiązanie jest ilościowe, obiektywne i efektywne kosztowo. Kolejną zaletą tego podejścia jest ograniczenie wymaganych informacji do surowych (nieprzetworzonych) danych o zadaniach wykonanych w badanym procesie wytwarzania oprogramowania. Model procesu wytwarzania oprogramowania uwzględniający cechy projektu i metodykę wytwarzania wymaga użycia tradycyjnego, obiektowego paradygmatu. Natomiast uwzględnienie wpływu struktury zespołu projektowego, w tym współpracy i komunikacji członków zespołu na proces wytwarzania można uzyskać dzięki rozszerzeniu modelu o paradygmat agentowy. Dzięki połączeniu obydwu paradygmatów modelowania, wynikowy agentowo–obiektowy model pozwoli na zautomatyzowanie oceny, która uwzględni wpływ wszystkich elementów środowiska projektowego na proces wytwarzania oprogramowania. Agentowo–obiektowy model procesu wytwarzania oprogramowania wspomoże zarządzanie projektami informatycznymi nie tylko przez umożliwienie ilościowej i obiektywnej oceny procesu wytwarzania i środowiska projektowego, ale również przez usprawnienie planowania środowiska projektowego oraz procesu wytwarzania oprogramowania.

Pytania badawcze

Dane jest:

- środowisko projektowe, składające się z: zakresu i planu wytwarzania projektu informatycznego, zespołu projektowego oraz metodyki wytwarzania oprogramowania,
- informacje o zadaniach wykonanych w badanym procesie wytwarzania.

P1. Czy zastosowanie agentowo–obiektowego modelu procesu wytwarzania oprogramowania pozwoli na ocenę środowiska projektowego? Przez ocenę rozumiane jest określenie zgodności badanego procesu wytwarzania z wzorcowym, będącym wynikiem symulacji modelu.

P2. Czy zastosowanie agentowo–obiektowego modelu procesu wytwarzania oprogramowania umożliwi planowanie prac środowiska projektowego? Przez planowanie rozumiane jest wyznaczenie harmonogramu prac dla przyjętego środowiska projektowego, spełniające zadane ograniczenia.

Cel badawczy

Celem badawczym pracy jest opracowanie metod oceny i planowania procesu wytwarzania i środowiska projektowego, bazujących na zbudowanym agentowo–obiektowym modelu procesu wytwarzania oprogramowania.

Ograniczenia

W celu ustalenia zakresu prac, uproszczenia modelu procesów wytwarzania oprogramowania oraz ułatwienia prac nad weryfikacją zasadności modelu, przyjęto następujące ograniczenia:

- opracowane metody mają zastosowanie jedynie w przypadku organizacji stosującej metodykę wytwarzania zgodną z Rational Unified Process (RUP),
- model procesów wytwarzania oprogramowania jest ograniczony do:
    - procesów wytwarzania zgodnych z RUP,
    - jednego projektu,
    - jednego zespołu projektowego,
    - poziom kompetencji członków zespołu projektowego został ograniczony do ról projektowych,
    - pełnego cyklu życia oprogramowania.

Szczegółowe ograniczenia dotyczące modelu i opartych na nim metod, można znaleźć w podrozdziale 3.3 Analiza dziedzinowa modelu.

Cel utylitarny

Celem utylitarnym pracy jest opracowanie systemu bazującego na agentowo–obiektowym modelu procesu wytwarzania umożliwiającego zastosowanie metody oceny i planowania środowiska wytwarzania oprogramowania i procesu wytwarzania oprogramowania. Opracowany system został opisany w dodatkach B, C i D niniejszej rozprawy.

Struktura rozprawy

W związku z przedstawionymi celami i zakresem, praca została podzielona na siedem części.

Pierwsza część pracy została poświęcona przeglądowi istniejących metodyk wytwarzania oprogramowania i ich porównaniu. Zaprezentowano również stan badań nad dopasowaniem i unowocześnianiem metodyk wytwarzania stosowanych w organizacji oraz przedstawiono, powiązane z oceną ryzyka transformacji organizacji, istniejące sposoby oceny dojrzałości organizacji i jej procesów wytwarzania oprogramowania. Ponieważ istniejące metody oceny procesów organizacji charakteryzują się złożonością, brakiem możliwości automatyzacji i niską efektywnością kosztową, pierwsza część rozprawy kończy się propozycją nowego podejścia w ocenie procesu wytwarzania oprogramowania opierającego się na zastosowaniu modelu procesu wytwarzania oprogramowania oraz koncepcją wykorzystania tego podejścia do planowania środowiska projektowego i procesu wytwarzania oprogramowania.

Druga część pracy skupia się na opracowaniu układu eksperymentu. Prace nad układem eksperymentu przebiegają od analizy przygotowania środowiska projektowego, do realizacji procesów wytwarzania według głównych metodyk, opisanych w części pierwszej rozprawy. Wynikiem analizy jest zestaw zmiennych wejściowych i wyjściowych opisujących system rzeczywisty w sposób umożliwiający ocenę oraz planowanie procesu wytwarzania.

Trzecia część rozprawy opisuje model podstawowy, czyli pasujący do wielu różnych układów eksperymentu. Model podstawowy przedstawiono jako rozbudowę zmiennych wejściowych i wyjściowych układu eksperymentu o elementy wewnętrzne, tworzące rdzeń modelu reprezentujący podstawowe składowe procesu wytwarzania oprogramowania. Na tym rdzeniu oparto rozszerzenia i podmodele związane z różnymi aspektami procesu wytwarzania skupionymi wokół zespołu projektowego, procesu wytwarzania i zapewniania jakości. Każdą propozycję rozbudowy opatrzono zestawem pytań, na które tak rozszerzony model mógłby udzielić odpowiedzi. Model podstawowy zawiera w sobie wiele tematów i aspektów, które nie tylko były opisywane jako koncepcje w artykułach, ale dla których sporządzano prototypowe rozwiązania w postaci arkuszy kalkulacyjnych lub kodu programu. Analiza dziedzinowa modelu podstawowego pozwoliła wyodrębnić te elementy, które są absolutnie niezbędne do realizacji celu. Jej wynikiem jest model uproszczony, dla którego wybrano odpowiednie podejście do modelowania. Trzecia część kończy się badaniem możliwości weryfikacji i propozycją weryfikacji zasadności replikatywnej i prognostycznej na podstawie eksperymentów.

W czwartej części rozprawy przedstawiono budowę uproszczonego modelu procesu wytwarzania oprogramowania, dostosowanego do układu eksperymentu wynikającego z celu jego budowy. Model uproszczony został najpierw przedstawiony w ujęciu ogólnym. Opisano jego zmienne wejściowe i wyjściowe, a następnie strukturę wewnętrzną na poziomie modułów funkcjonalnych. W dalszej części rozdziału przedstawiono szczegółową budowę modelu – opisano elementy wewnętrzne i ich interakcje, przy zastosowaniu do modelowania podejścia wieloagentowego. Agenty i środowisko agentowe przedstawiono w sposób formalny, za pomocą równań. Zachowania agentów realizujące algorytmy opisano jako zbiór automatów skończonych. Wykonano w pełni funkcjonalną implementację modelu w środowisku java, w postaci systemu do symulacji i analizy wyników. Na podstawie modelu AGOMO opracowano metodę AGOMAS do oceny procesów RUP organizacji.

W piątej części rozprawy w celu weryfikacji replikatywnej modelu przedstawiono plan eksperymentu oparty na metodzie AGOMAS. W ramach przeprowadzonego eksperymentu przedstawiono badaną organizację i scharakteryzowano badany projekt, a następnie przeprowadzono

pięć etapów eksperymentu. W jego wyniku model odtworzył przebieg badanego projektu informatycznego. Utworzony przez model harmonogram działań został użyty jako wzorzec procesu do porównania z harmonogramem badanego projektu. Efektem tego porównania była wysoka wartość wskaźnika zgodności, co zostało zinterpretowane jako wysoka zgodność badanego procesu wytwarzania z przyjętą metodyką, co sugerowało niskie ryzyko procesu zwinnej transformacji. Niskie ryzyko transformacji organizacji zostało potwierdzone przez udaną transformację badanej organizacji. Zatem cele eksperymentu zostały osiągnięte, co oznacza potwierdzenie zarówno zdolności modelu do odtworzenia przebiegu rzeczywistego procesu wytwarzania oprogramowania, jak i przydatność metody AGOMAS do oceny procesów wytwarzania oprogramowania organizacji na podstawie analizy danych zakończonych projektów informatycznych.

W szóstej części rozprawy opisano proces weryfikacji prognostycznej modelu AGOMO. Weryfikacja miała na celu sprawdzenie zdolności modelu AGOMO do prognozowania procesów wytwarzania oprogramowania nieznanego modelowi systemu rzeczywistego. Rozdział zaczyna się od opisu metody AGOPLAN – metody planowania procesów RUP bazującej na modelu AGOMO. Następnie, zgodnie z przedstawioną metodą, przeprowadzono eksperyment mający na celu weryfikację zasadności predykcyjnej modelu AGOMO i poprawności opracowanej metody. Wyniki porównania planu procesu wykonanego z wykorzystaniem metody AGOPLAN z rzeczywistym procesem wytwarzania prowadzą do wniosku o potwierdzeniu przez eksperyment zasadności predyktywnej modelu AGOMO. Powyższy wniosek potwierdza również przydatność metody AGOPLAN do planowania procesów RUP dla projektów informatycznych.

W siódmej części rozprawy dokonano podsumowania prac, przedstawiono wnioski i zarysowano zakres dalszych prac rozwojowych nad wynikami zaprezentowanymi w rozprawie. Przedstawiono zarówno przyszłe kierunki badań, jak i możliwości komercyjnego wykorzystania modelu AGOMO oraz opracowanych na jego bazie metod.

Podziękowania

Tej pracy nie byłoby bez wsparcia wielu osób i organizacji, które w tym miejscu pragnę wymienić. Są to:

- prof. Cezary Orłowski, mentor i promotor tej pracy, który motywował mnie do pracy, dając jednocześnie solidne oparcie dzięki swojej wiedzy, zaangażowaniu, szerokiemu spojrzeniu na kontekst prowadzonych badań i włączeniu w prace zespołu zajmującego się tematyką zwinnej transformacji;
- dr hab. inż. Grzegorz Bocewicz, profesor Politechniki Koszalińskiej, promotor pomocniczy tej pracy, który mimo wielu obowiązków zawsze znajdował czas, żeby mnie wspomóc swoimi cennymi uwagami oraz konstruktywnymi propozycjami zmian;
- pracownicy Zakład Podstaw Informatyki i Zarządzania Politechniki Koszalińskiej, którzy poświęcali wiele uwagi moim wystąpieniom i prezentacjom, po których udzielali mi merytorycznych informacji zwrotnych;
- pracownicy Zakładu Zarządzania Technologiami Informatycznymi Politechniki Gdańskiej, którzy w przyjazny sposób pomogli mi rozszerzyć horyzonty i zmienić perspektywę spojrzenia na istotę moich poszukiwań;

- pracownicy Wyższej Szkoły Bankowej w Gdańsku, w szczególności dr inż. Artur Ziółkowski i dr inż. Irena Bach–Dąbrowska za współpracę i inspirację;
- mój przyjaciel Remigiusz Szyndler, który bardzo mi pomógł w przygotowaniu angielskojęzycznych wersji moich artykułów i prezentacji.

Wszystkim im za inspirację, wsparcie i pomoc gorąco dziękuję!

1. Ocena procesu wytwarzania oprogramowania – stan badań

Pod koniec XX wieku rozpoczęła się epoka informacji, który charakteryzuje się tym, że głównym zasobem strategicznym, na którym opiera się wzrost i rozwój jednostek, organizacji i społeczeństw jest informacja. Za przetwarzanie i dostarczanie informacji odpowiedzialne jest oprogramowanie. Stąd wytwarzanie oprogramowania pełni kluczową rolę w rozwoju nowoczesnego społeczeństwa. Oprogramowanie jest wszędzie, produkcja większości dóbr materialnych jest sterowana przez oprogramowanie i większość komercyjnych usług polega na systemach informatycznych.

W początkowym etapie rozwoju informatyki, około 60 lat temu, oprogramowanie było wytwarzane w sposób nieuporządkowany, który często jest nazywany metodą „koduj i naprawiaj" [10]. Programy zwykle były pisane bez specyfikacji i bez całościowego planu. Projekt powstawał na podstawie wielu krótkoterminowych planów i doraźnych decyzji. Była to najprostsza i zarazem najdroższa metoda wytwarzania oprogramowania. Kiedy systemy stawały się coraz większe i bardziej złożone, utrzymanie i rozbudowa systemów wytworzonych w sposób chaotyczny stało się trudne. Narastające trudności stały się przyczyną wprowadzenia wytwarzana oprogramowania opartego na metodykach.

Definicja 1.1 Metodyka wytwarzania oprogramowania dzieli prace związane z jego wytwarzaniem na osobne fazy (etapy) zawierające działania, które posłużą lepszemu planowaniu i zarządzaniu. Metodyka może zawierać definicję wytwarzanych produktów lub artefaktów, które są wytwarzane i gromadzone przez zespół projektowy w celu wytworzenia lub utrzymania systemu informatycznego.

Celem niniejszego rozdziału jest wprowadzenie w tematykę zmiany metodyk wytwarzania oprogramowania w organizacjach wytwarzających oprogramowanie. Na początku zostaną przedstawione i porównane główne, tradycyjne i zwinne, metodyki wytwarzania oprogramowania. W kolejnej części zostanie pokazany aktualny stan badań nad modelami ułatwiającymi zmianę metodyk stosowanych w organizacji. Następnie zostanie przedstawiony najczęściej stosowany model oceny dojrzałości organizacji. Rozdział zostanie zakończony podsumowaniem i propozycją nowej metody oceny procesów wytwarzania organizacji.

1.1. Stan badań oceny procesów wytwarzana oprogramowania

W tym rozdziale zostaną przedstawione tradycyjne i zwinne metodyki wytwarzania z punktu widzenia aktualnego stanu badań. Następnie, po porównaniu metodyk, zostaną pokazane prace nad modelami transformacji organizacji IT. W dalszej części przedstawiono model dojrzałości organizacji i jego związki ze zwinną transformacją organizacji.

1.1.1. Tradycyjne metodyki wytwarzania

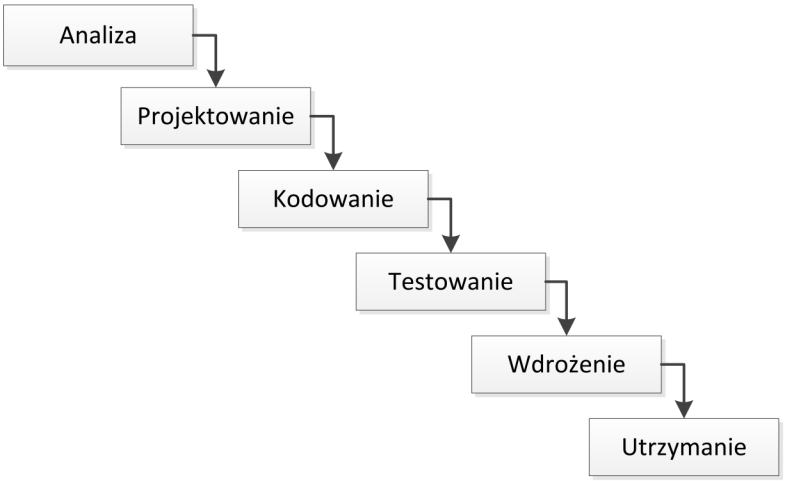
Tradycyjne metodyki wytwarzania oprogramowania są sterowane przez planowanie. Na początku ustalany jest zakres, termin i budżet projektu. Prace nad projektem podzielone są na

etapy, które składają się na cykl życia projektu. Praca zaczyna się od pozyskania i udokumentowania całego zestawu wymagań, po których następuje analiza i projektowanie, tworzenie kodu, testowanie i wdrożenie gotowego systemu. Układ prac skutkuje brakiem elastyczności i odporności na zmiany wymagań w trakcie prac nad projektem. Informacje zwrotne od klienta nie są brane pod uwagę w trakcie prac nad produktem, czego wynikiem jest, że 60% funkcjonalności opracowanego systemu jest rzadko lub w ogóle nie używana. Z powodu dużego poziomu obciążenia formalnymi wymaganiami dotyczącymi dokumentacji, metodyki tradycyjne są również nazywane ciężkimi. Skuteczność tradycyjnych metodyk jest niska, wg badania Standish Group z roku 2012 [17] roku 57% projektów kończy się niedotrzymaniem warunków umowy, przekroczeniem budżetu lub terminu, a 29% projektów jest zakończonych całkowitą porażką. Martin Fowler krytykuje te metodyki jako biurokratyczne [18]. Uważa, że wysiłek wydatkowany na podążanie za wymogami danej metodyki spowalnia cały proces wytwarzania.

Poniżej zostaną przedstawione najważniejsze ciężkie metodyki: kaskadowa, spiralna i RUP [55], jako przedstawicieli kolejnych etapów rozwoju tych metodyk.

Kaskadowa metodyka wytwarzania

Kaskadowa metodyka wytwarzania została zaproponowana w roku 1970 przez Winstona Royce'a [85]. Jest to najstarsza i najbardziej znana z metodyk. Podejście kaskadowe kładzie nacisk na przejścia między ściśle zdefiniowanymi fazami. Każda z faz składa się z określonych czynności i produktów, które muszą zostać wytworzone, zanim zostanie uruchomiona kolejna faza procesu. Nazwy faz różnią się w zależności od implementacji, jednak idea pozostaje taka sama. Fazy metodyki kaskadowej zostały pokazane na Rys. 1.1. W pierwszej fazie należy określić, co system ma robić. Jej produktem są wymagania systemowe. W następnej fazie należy odpowiedzieć na pytanie jak to osiągnąć? Produktem jest projekt systemu. W trzeciej fazie developerzy tworzą kod systemu. Gdy cały kod systemu zostaje utworzony, w czwartej fazie procesu system jest testowany. Po przejściu fazy testów, w fazie wdrożenia tworzona jest dokumentacja, prowadzone są szkolenia i system jest wdrażany u klienta. Ostatnia faza to utrzymanie pracującego systemu.



Rys. 1.1 Fazy metodyki kaskadowej

źródło: Opracowanie własne na podstawie [55]

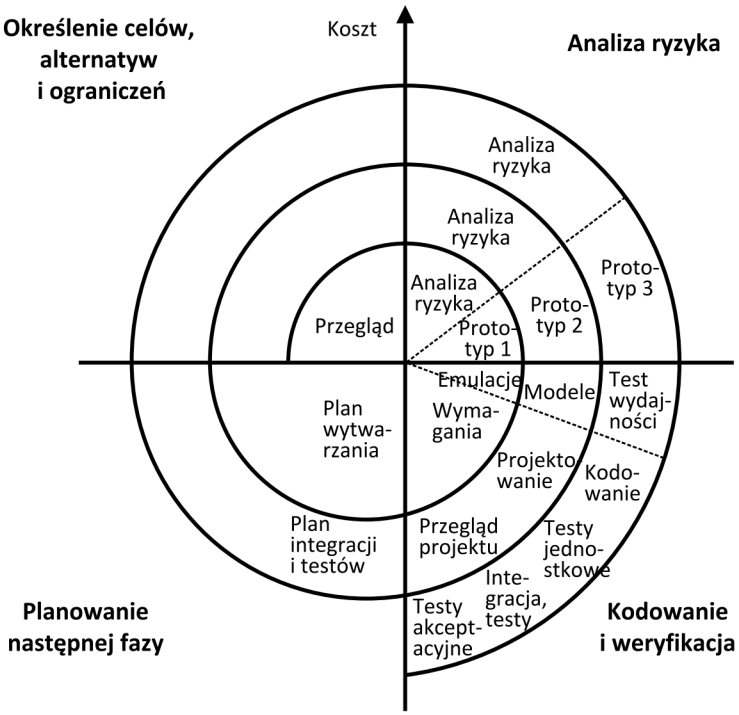
Model kaskadowy może być stosowany w następujących przypadkach:

- wymagania są jasno określone, zrozumiane i ich zbieranie jest zakończone,
- zakres projektu jest stabilny,
- technologia jest dobrze znana,
- brak niejasnych lub niespójnych wymagań,
- zasoby, które wymagają specjalistycznej wiedzy są ogólnie dostępne,
- projekt jest krótki [55].

Z modelu kaskadowego wyewoluowały kolejne metodyki tradycyjne, z których zostaną omówione te, które mają największy wpływ na rozwój inżynierii oprogramowania. Są to: model spiralny i metodyka RUP.

Model spiralny

Model spiralny jest znany również pod nazwą model Boehmego [10]. Proces jest reprezentowany przez spiralę, a nie przez sekwencję czynności, jak w modelu kaskadowym. Spirala ma wiele cykli, a każdy cykl reprezentuje fazę procesu. Proces nie ma stałych faz, takich jak analiza czy projektowanie. Elementy składowe cykli są dobierane w zależności od potrzeby. Ryzyko jest szacowane i ograniczane w trakcie całego procesu. Struktura modelu spiralnego jest pokazana na Rys. 1.2.



Rys. 1.2 Struktura procesu wytwarzania modelu spiralnego

źródło: Opracowanie własne na podstawie [55]

Jego składowymi są cztery główne fazy:

- określenie celów – identyfikacja celów do osiągniecia w bieżącej fazie projektu,
- analiza i zapobieganie ryzyku – kluczowe ryzyka są zidentyfikowane, przeanalizowane i uzyskana jest informacja o możliwości zapobieżeniu ryzyku,
- kodowanie i weryfikacja – kiedy ryzyko zostało ograniczone, tworzony jest kod projektu oraz są tworzone i przeprowadzane testy projektu,
- planowanie – projekt podlega przeglądom i tworzony jest plan następnego cyklu spirali.

Model spiralny jest stosowany w następujących warunkach:

- projekty o dużym zakresie i budżecie,
- kiedy oszacowanie ryzyka jest czynnikiem krytycznym,
- wymagania nie są jasno określone ani złożone,
- kiedy koszty i oszacowanie ryzyka są istotne,
- dla projektów obciążonych średnim i dużym ryzykiem,
- gdy projekt jest długotrwały i wymagania mogą się zmieniać z powodu zmiany priorytetów ekonomicznych [55].

Jedną z realizacji modelu spiralnego jest metodyka MSF stosowana w firmie Microsoft. Dużym wkładem modelu spiralnego do metodyk wytwarzania oprogramowania jest silnie zaakcentowana konieczność analizy ryzyka i wprowadzenie idei wytwarzania oprogramowania w kolejnych przyrostach funkcjonalności.

Rational Unified Process (RUP)

Metodyka Rational Unified Process (RUP) powstała w firmie Rational Software, przejętej później przez IBM. RUP jest ciężkim, iteracyjnym podejściem, które bierze pod uwagę potrzebę wprowadzania zmian i adaptacji podczas procesu wytwarzania oprogramowania. W metodyce

RUP oprogramowanie jest projektowane i wytwarzane przyrostowo podczas kolejnych iteracji procesu. Każda iteracja zawiera podzbiór czynności z wszystkich dyscyplin wytwarzania, takich jak wymagania, analiza, projektowanie, implementacja i testowanie. Metodyka zapewnia uporządkowane podejście do przydzielania zadań i odpowiedzialności w ramach organizacji w celu pomyślnego ukończenia projektów [59, 61].

Głównym celem RUP jest zapewnienie wyprodukowania wysokiej jakości oprogramowania zgodnego z wymaganiami klienta, w przewidywalnym terminie i określonym budżecie. Model zawiera wskazówki, szablony i porady dotyczące użycia narzędzi przez zespół projektowy, aby w pełni wykorzystać możliwości metodyki.

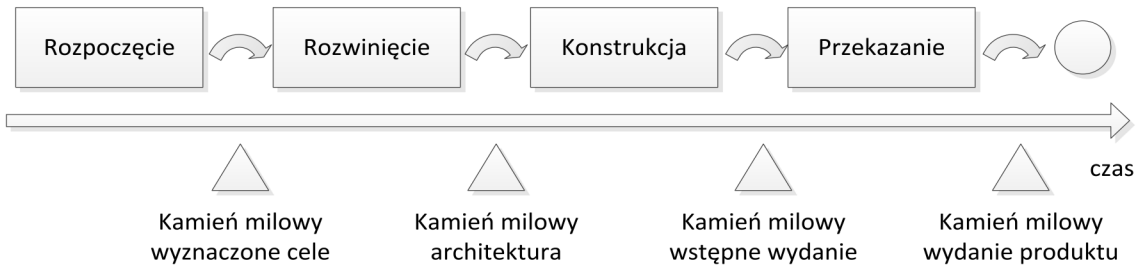
Poniżej zamieszczono wskazówki RUP oparte na najlepszych praktykach:

- wytwarzaj oprogramowanie iteracyjnie,
- zarządzaj wymaganiami,
- użyj architektury opartej na komponentach,
- użyj wizualnego modelowania oprogramowania,
- weryfikuj jakość oprogramowania,
- kontroluj zmiany w oprogramowaniu.

Proces wytwarzania w RUP składa się z następujących faz:

- rozpoczęcie: (*zrozumieć, co należy zbudować*) – w tej fazie określany jest zakres projektu, szacowany koszt, ryzyko, ustalany cel biznesowy, środowisko i zarys architektury;
- opracowanie: (*zrozumieć, jak to zbudować*) – w tej fazie są określone szczegółowe wymagania, architektura jest sprawdzona, ustalone środowisko projektu i skonfigurowany zespół projektowy;
- konstrukcja: (*budowa produktu*) – w tej fazie jest wytworzone i przetestowane oprogramowanie oraz wyprodukowana dokumentacja;
- przekazanie: (*przekazanie produktu użytkownikom*) – w tej fazie przeprowadzone są testy systemowe i testy akceptacyjne oraz produkt jest poprawiony i wdrożony.

Fazy RUP składają się z jednej lub wielu iteracji i powinny być zakończone wyprodukowaniem zestawu artefaktów nazywanych kamieniem milowym fazy. Koncepcja faz została przedstawiona na Rys. 1.3.



Rys. 1.3 Fazy i kamienie milowe procesu RUP
źródło: Opracowanie własne na podstawie [59]

Model RUP jest stosowany do wytwarzania następujących systemów:

- systemy rozproszone,

- bardzo duże lub złożone systemy,
- systemy łączące wiele obszarów biznesowych,
- systemy ponownie wykorzystujące inne systemy,
- rozproszone wytwarzanie systemu [55].

Dużą zaletą metodyki RUP jest precyzyjna definicja elementów składowych oraz możliwość dostosowania metodyki do organizacji i projektu. Twórcy metodyki zachęcają do analizy, które artefakty są rzeczywiście niezbędne i które działania składowe procesu można pominąć. Jednak mimo iteracyjnego podejścia do wytwarzania oprogramowania, gdzie każda iteracja ma przynieść określone biznesowe korzyści klientowi, rzeczywiste przekazanie oprogramowania następuje na końcu całego procesu – w fazie przekazania. Ta cecha metodyki RUP sprawia, że należy do metodyk tradycyjnych.

1.1.2. Zwinne metodyki wytwarzania

W wyniku problemów z uwzględnieniem zmian w wymaganiach i niskiej skuteczności tradycyjnych metodyk wytwarzania, wielu konsultantów niezależnie rozwinęło metodyki i praktyki, które stały się znane pod nazwą zwinnych. Jest wiele zwinnych metodyk wytwarzania [52] i każda z nich posiada unikalne cechy, ale wszystkie opierają się na tych samych wartościach i zasadach opisanych w Manifeście Agile [118]. Wszystkie używają ciągłej komunikacji, planowania, testowania i integracji. Zwinne metodyki wytwarzania pomagają w wytworzeniu dobrego oprogramowania, ale tym co je definiuje jest zachęta do współpracy oraz podejmowanie dobrych i szybkich wspólnych decyzji. Do zwinnych metodyk wytwarzania oprogramowania należą: Adaptive Software Development (ASD), Feature Driven Development (FDD), Crystal Clear, Dynamic Software Development Method (DSDM), Rapid Application Development (RAD), SCRUM, Extreme Programming (XP). Poniżej przedstawiamy manifest, który stanowi wspólną podstawę zwinnych metodyk oraz najważniejsze i najczęściej stosowane metodyki XP i Scrum.

Manifest programowania zwinnego

W lutym 2001 roku siedemnastu reprezentantów różnych zwinnych metodyk zdecydowało zawiązać sojusz zwinności (Agile Alliance), aby lepiej promować swoją wizję wytwarzania oprogramowania. Wynikiem ich działań było powstanie manifestu programowania zwinnego (Agile Manifest). Większość technik zwinnego programowania było znanych przed zawiązaniem sojuszu, jednak po nim zostały zgrupowane w gotowy do użycia model.

Oto wartości, na których skupili się uczestnicy sojuszu:

„Odkrywamy nowe metody programowania dzięki praktyce w programowaniu
i wspieraniu w nim innych.
W wyniku naszej pracy, zaczęliśmy bardziej cenić:

Ludzi i interakcje od procesów i narzędzi
Działające oprogramowanie od szczegółowej dokumentacji
Współpracę z klientem od negocjacji umów
Reagowanie na zmiany od realizacji założonego planu.

Oznacza to, że elementy wypisane po prawej są wartościowe,
ale większą wartość mają dla nas te, które wypisano po lewej.” [118]

Manifest programowania zwinnego zawiera 12 zasad:

1. Najwyższy priorytet ma dla nas zadowolenie klienta dzięki wczesnemu i ciągłemu wdrażaniu wartościowego oprogramowania.
2. Bądźcie gotowi na zmiany wymagań nawet na późnym etapie jego rozwoju. Procesy zwinne wykorzystują zmiany dla zapewnienia klientowi konkurencyjności.
3. Dostarczajcie funkcjonujące oprogramowanie często, w kilkutygodniowych lub kilkumiesięcznych odstępach. Im częściej, tym lepiej.
4. Zespoły biznesowe i deweloperskie muszą ściśle ze sobą współpracować w codziennej pracy przez cały czas trwania projektu.
5. Twórzcie projekty wokół zmotywowanych ludzi. Zapewnijcie im potrzebne środowisko oraz wsparcie i zaufajcie, że wykonają powierzone zadanie.
6. Najbardziej efektywnym i wydajnym sposobem przekazywania informacji zespołowi deweloperskiemu i wewnątrz niego jest rozmowa twarzą w twarz.
7. Działające oprogramowanie jest podstawową miarą postępu.
8. Procesy zwinne umożliwiają zrównoważony rozwój. Sponsorzy, deweloperzy oraz użytkownicy powinni być w stanie utrzymywać równe tempo pracy.
9. Ciągłe skupienie na technicznej doskonałości i dobrym projektowaniu zwiększa zwinność.
10. Prostota – sztuka minimalizowania ilości koniecznej pracy – jest kluczowa.
11. Najlepsze rozwiązania architektoniczne, wymagania i projekty pochodzą od samoorganizujących się zespołów.
12. W regularnych odstępach czasu zespół analizuje możliwości poprawy swojej wydajności, a następnie dostraja i dostosowuje swoje działania do wyciągniętych wniosków [118].

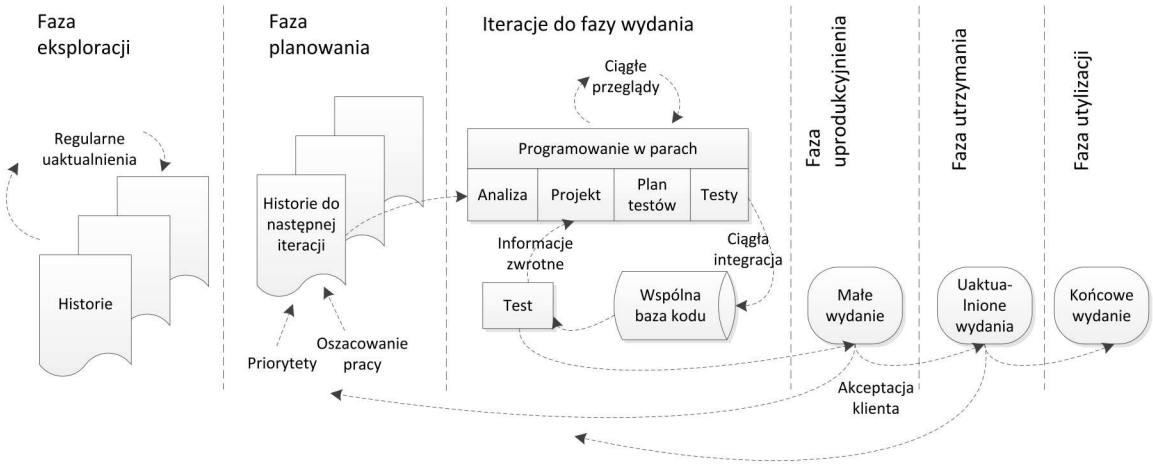
Publikacja Manifestu przyniosła bezprecedensowe zmiany w dziedzinie inżynierii oprogramowania. Trudno podać przykład wydarzenia w XX wieku, które przyniosłoby tak wielkie zmiany w metodykach programowania, technikach, narzędziach i najlepszych praktykach. Ten niespotykany rozwój został szybko zaakceptowany przez praktyków, co doprowadziło do szerokiego rozpropagowania idei podejścia zwinnego.

Programowanie ekstremalne XP

Przyczyną powstania programowania ekstremalnego (XP) była potrzeba skrócenia długich cykli wytwarzania tradycyjnych metodyk [7]. Programowanie ekstremalne charakteryzuje się krótkimi cyklami wytwarzania, przyrostowym planowaniem, ciągłymi informacjami zwrotnymi od klienta, poleganiem na komunikacji i ewolucyjnym projektowaniu [8]. Działając zgodnie z tymi wartościami, programiści reagują na zmieniające się otoczenie z dużo większą odwagą. Jak pisze Williams [106], członkowie zespołu XP spędzają kilka minut na programowaniu, kilka na zarządzaniu projektem, kilka minut na projektowaniu, kilka na odbiorze informacji zwrotnych, kilku minut na pracy z zespołem. Jest to powtarzane wiele razy w ciągu dnia pracy. Określenie ekstremalny wzięło się stąd, że te zdroworozsądkowe zasady i praktyki są podniesione do ekstremalnego poziomu. Poniżej zamieszczono podsumowanie określeń i zasad XP [7]:

- Planowanie – programista szacuje ilość pracy potrzebną do implementacji historii klienta. Klient na podstawie oszacowania decyduje o zakresie i czasie wydania implementacji.
- Małe/krótkie wydania – aplikacja jest wytwarzana jako seria małych, często uaktualnianych wersji. Nowe wersje są wydawane z dowolnym okresem – od dni do miesiąca.
- Metafora – system jest zdefiniowany przez klienta i programistę jako zbiór metafor opisujący działanie systemu.
- Prosty projekt – nacisk na zaprojektowanie rozwiązania najprostszego z możliwych, nadmiarowa złożoność i zbędny kod mają być natychmiast usuwane.
- Refaktoryzacja – polega na restrukturyzacji systemu usuwającej powielone elementy, ułatwiającej komunikację, upraszczającej i zwiększającej elastyczność bez zmiany funkcjonalności programu.
- Programowanie w parach – cały produkcyjny kod powinien być napisany przez dwóch programistów pracujących przy jednym komputerze.
- Wspólna własność – żadna pojedyncza osoba nie może być właścicielem lub być odpowiedzialna za część kodu. Raczej każdy powinien móc zmieniać dowolny fragment kodu w dowolnym czasie.
- Ciągła integracja – nowy kawałek kodu jest integrowany z systemem tak szybko, jak tylko jest gotowy. Po integracji cały system musi być zbudowany na nowo i wszystkie testy muszą przejść pozytywnie, żeby zmiany zostały zaakceptowane.
- 40–godzinny tydzień – nikt nie może brać nadgodzin przez dwa tygodnie pod rząd. Inaczej jest to traktowane jako problem.
- Klient w zespole – klient musi być dostępny przez cały czas dla członków zespołu projektowego.
- Standardy kodowania – zasady kodowania istnieją i są przestrzegane przez programistów, co zwiększa spójność i ułatwia komunikację w zespole projektowym.

Cykl życia projektu XP pokazany na Rys. 1.4 [8], został podzielony na sześć faz: eksplorację, planowanie, iteracje do wydania, produkcja, utrzymanie i utylizację. W fazie eksploracji klient wypisuje karty z historiami, które mają wchodzić w skład programu. To prowadzi do fazy planowania, w której historiom użytkownika są przypisywane priorytety i wytwarzane jest pierwsze wydanie. Następnie, w fazie iteracje do wydania, zespół projektowy w pierwszej iteracji tworzy architekturę całego systemu, z którą następnie jest integrowany i testowany tworzony kod programu. Rozszerzone testowanie i sprawdzanie wydajności zanim system zostanie wydany klientowi jest wykonywane w fazie produkcji. Pomysły i sugestie, które powstają w tej fazie zostają odłożone i będą realizowane w zaktualizowanych wydaniach wytworzonych w fazie utrzymania. W fazie utylizacji systemu wszystkie historie użytkownika zostały już zaimplementowane, powstała cała niezbędna dokumentacja i nie ma potrzeby zmiany architektury, projektu i kodu systemu.



Rys. 1.4 Fazy cyklu życia projektu XP

Opracowanie własne na podstawie [8]

Programowanie ekstremalne skupia się na dostarczaniu wartości biznesowej klientowi i eliminowaniu marnotrawienia zasobów. Zespół projektowy stosuje praktyki inżynierskie zapewniające wysoką jakość dostarczonego oprogramowania. Programowanie ekstremalne stało się w praktyce inżynierskiej zbiorem dobrych praktyk, bardzo chętnie wykorzystywanych przez inne zwinne metodyki, np. Scrum.

Metodyka SCRUM

Scrum jest iteracyjnym, przyrostowym procesem wytwarzania dowolnego produktu lub zarządzania dowolną pracą. Scrum jest zorientowany na to, jak członkowie zespołu powinni współpracować, aby uzyskać elastyczny system, dostosowujący się do ciągle zmieniającego się środowiska. Na końcu każdej iteracji produkowany jest zbiór funkcjonalności. Termin „scrum" pochodzi z gry rugby, gdzie oznacza powrót piłki z autu do gry, wymagający pracy zespołu [90].

Scrum nie wymaga ani nie dostarcza żadnej metody lub praktyki specyficznej dla dziedziny wytwarzania oprogramowania. Zamiast tego wprowadza określone praktyki zarządzania i narzędzia w różnych fazach. Pozwala to ograniczyć chaos wynikający z nieprzewidywalności i złożoności [82].

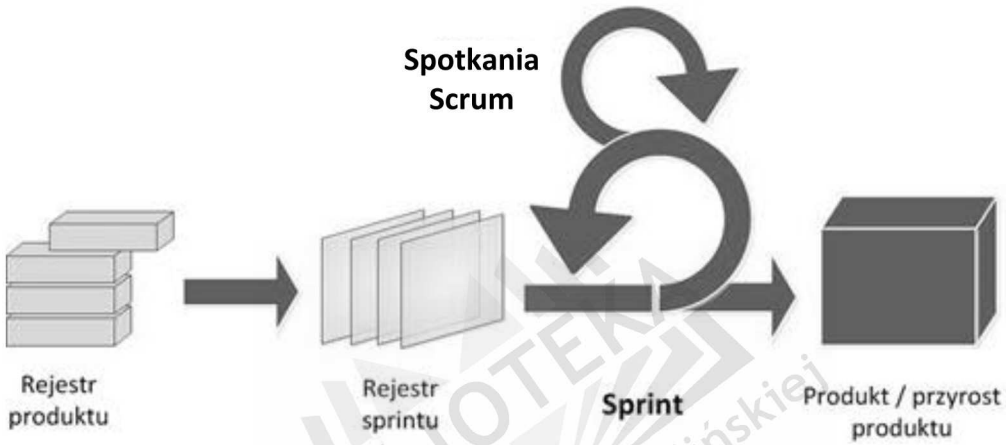
W procesie Scrum zespół projektowy używa trzech artefaktów. Wszystkie są otwarte i dostępne dla członków zespołu projektowego. Oto one:

- Rejestr produktu – ewoluująca lista wszystkich cech i zmian, które mają być wprowadzone w produkcie w wydaniu. Lista posiada priorytety i zawiera zmiany wymagane przez klientów, dział marketingu i zespół projektowy. Odpowiedzialnym za zarządzanie Rejestrem produktu jest właściciel produktu.
- Rejestr sprintu – lista biznesowych i technologicznych cech, rozszerzeń i poprawek błędów, która jest przypisana do konkretnego sprintu. Wymagania podzielone są na konkretne zadania. Zadania mają przydzielonego wykonawcę i oszacowaną liczbę godzin pracy, która jest wymagana do ukończenia. Szacowania pracochłonności mogą zmieniać się w trakcie pracy nad zadaniem. Kiedy wszystkie zadania są ukończone, dostarczana jest nowa iteracja systemu.

- Wykres wypalenia – Liczba godzin pozostała do zakończenia zadań sprintu jest umieszczana na wykresie, który jest przedstawiony zespołowi projektowemu.

Główne praktyki używane w metodyce scrum są przedstawione poniżej.

- Sprinty – najczęściej 30–sto dniowe procedury przystosowania do zmian w zmiennych środowiska (wymagania, czas, zasoby, wiedza, technologia). Muszą zakończyć się wersją, która potencjalnie może być dostarczona klientowi. Narzędziami do pracy zespołu są spotkania planowania sprintu, rejestr produktu i codzienne spotkania scrum.
- Codzienny scrum – codzienne spotkanie zespołu trwające około 15 minut. Spotkania służą przedstawieniu postępów prac oraz identyfikacji problemów i trudności, z którymi spotykają się członkowie zespołu.



Rys. 1.5 Podstawowe elementy metodyki Scrum

źródło: [82]

Ogólny zarys metodyki Scrum jest pokazany na Rys. 1.5. Elementy metodyki zostały omówione poniżej.

- Planowanie sprintu – spotkanie, w którym uczestniczy zespół projektowy, kierownictwo i właściciel produktu. Właściciel produktu jest reprezentantem klienta lub zespołu klientów. Właściciel produktu tworzy i ustala priorytety elementów rejestru produktu. W trakcie planowania sprintu właściciel produktu wybiera, które cechy produktu zostaną zrealizowane w następnym 30–sto dniowym przyroście (nazywanym sprintem). Zwykle kieruje się przy tym najwyższą wartością biznesową i oceną ryzyka. Dodatkowo obierany jest cel sprintu, który służy jako minimalne, wysokopoziomowe kryterium sukcesu sprintu. Cel pozwala zespołowi projektowemu bardziej skupić się na całościowym obrazie niż na cechach wybranych do implementacji.
- Ciągła integracja – podczas sprintu kod jest codziennie integrowany i poddawany testom regresyjnym.
- Spotkania Scrum – krótkie, 15–sto minutowe spotkania są przeprowadzane codziennie o stałej porze. Spotkania przeprowadzane są w sali wyposażonej w tablicę, dzięki czemu można wypisać zadania i problemy. Na spotkaniach scrum tylko członkowie zespołu i scrum master mogą zabierać głos. Każdy z członków zespołu odpowiada na trzy pytania:
    - co zrobiłem od ostatniego spotkania scrum?
    - co mam zamiar zrobić do następnego spotkania?

  - jakie mam problemy z realizacją zadań?
- Spotkania Scrum są najistotniejszą częścią metodyki. Podczas ich trwania są podejmowane zobowiązania wobec całego zespołu projektowego, co zwiększa odpowiedzialność, podtrzymuje podążanie za wyznaczonymi celami i utrzymuje projekt na właściwym kursie. Spotkania Scrum mogą stać się trudne do zarządzania, jeśli będzie w nich uczestniczyć zbyt dużo osób. Dlatego zaleca się optymalną liczbę siedmiu osób biorących udział w spotkaniu. W przypadku większych zespołów są one podzielone na mniejsze grupy, które mają osobne spotkania Scrum. Jeden przedstawiciel z każdej z grup bierze następnie udział w spotkaniu „scrum of scrums". Przedstawiciel grupy odpowiada na standardowe trzy pytania spotkania scrum. Dzięki tej praktyce najważniejsze informacje są przekazywane pomiędzy grupami.
- Przegląd sprintu – spotkanie odbywa się na koniec każdego sprintu. Na spotkaniu odbywa się przegląd postępu prac, nowe cechy oprogramowania są demonstrowane klientowi, kierownictwu, użytkownikom i właścicielowi produktu. Spotkanie jest prowadzone przez Scrum mastera. Najnowsza wersja produktu jest prezentowana właścicielowi produktu z wszystkimi funkcjami, mocnymi i słabymi stronami i problemami. Prezentacja jest skupiona na produkcie, prezentacje formalne w postaci slajdów są zabronione.
- Cykl jest kontynuowany i odbywa się kolejne spotkanie planowania sprintu, aby wybrać cechy dla następnego sprintu.

Scrum dostarcza doskonałego, zespołowego podejścia do ustalania priorytetów i dostarczania wyników prac opartym na koncepcji stale ewoluującego rejestru produktu. Siłą Scruma jest jego prostota, ponieważ łatwo jest go opisać i rozpocząć z nim pracę. Jednakże w Scrumie nie ma koncepcji projektu, jest tylko rejestr zaległości prac do wykonania.

1.1.3. Zwinna transformacja

Wiele organizacji jest zainteresowanych wdrożeniem zwinnych metodyk wytwarzania z powodu szybkich wydań, elastyczności projektu i ulepszenia komunikacji w zespole [9]. Przyczyniło się to do wzrostu ich popularności [109] i wiele organizacji zaczęło stosować te rozwijające się metodyki.

Definicja 1.2 Proces przystosowania organizacji IT do użycia zwinnych metodyk jest nazwany zwinną transformacją.

Ze zwinną transformacją wiąże się duże ryzyko niepowodzenia [68]. Niepowodzenie zwinnej transformacji skutkuje stratami wynikającymi z niedostarczenia projektu informatycznego i wycofaniem się organizacji z drogi radykalnego ulepszania metodyk wytwarzania i zarządzania projektami IT [17]. Oznacza to ryzyko niepowodzenia przy wykonywaniu kolejnych projektów. Poszukiwane są sposoby zapobiegania temu ryzyku [17], co znajduje wyraz w badaniach procesów zwinnej transformacji organizacji IT.

1.1.4. Porównanie metodyk tradycyjnych i zwinnych

Metodyki tradycyjne opierały się na założeniu, że wymagania co do systemu informatycznego można określić na pierwszym etapie projektu i na tej podstawie zaplanować i zrealizować dalsze prace. Założenia metodyk zwinnych wynikają z obserwacji, że wdrożenie części zaplanowanego systemu prowadzi do lepszego zrozumienia całości rozwiązywanego problemu.

W związku z tym zmieniają się priorytety dotychczasowych wymagań, powstają nowe wymagania lub pojawia się potrzeba wprowadzenia zmian. Dają one narzędzia do zarządzania zmianami w trakcie realizacji projektu i poprawiają skuteczność realizacji nowoczesnych projektów informatycznych.

Wieloaspektowe porównanie cech metodyk zwinnych i tradycyjnych zostało umieszczone w Tab. 1.1. Część porównywanych aspektów można określić jako czynnik ludzki, inne jako techniczny. Większość z nich to czynniki ludzkie, jednak niektóre kluczowe aspekty można określić jako techniczne. Jest między nimi taka różnica, że o ludzkich aspektach decydują ludzie, natomiast aspekty techniczne wynikają z przyjętego planu implementacji projektu. Aspekty techniczne to model wytwarzania, wymagania użytkowników, architektura i przemodelowanie. Są one bezpośrednio związane z wymaganiami i poziomem umiejętności technicznych wynikających z systemu informatycznego [25].

Tab. 1.1 Porównanie cech metodyk zwinnych i tradycyjnych
Źródło: [97]

Aspekt	Metodyki tradycyjne	Metodyki zwinne
Podstawowa hipoteza	Systemy mogą być w pełni opisane i przewidywalne, mogą być opracowane dzięki szerokiemu i dokładnemu planowaniu	Oprogramowanie dostosowane do potrzeb klienta, o wysokiej jakości jest wytwarzane przez małe zespołu stosujące zasadę ciągłego ulepszania projektu i testowania polegającego na szybkiej informacji zwrotnej i zmianie
Styl zarządzania	Autokratyczny, nakazy i kontrola	Zdecentralizowany, przywództwo i współpraca
Zarządzanie wiedzą	Sprecyzowane	Taktyczne
Komunikacja	Formalna	Nieformalna
Model wytwarzania	Model cyklu życia (kaskadowy, spiralny lub model zmodyfikowany)	Model ewolucyjno–dostarczeniowy
Struktura organizacyjna	Mechaniczna (biurokratyczna, wysoka formalizacja), przeznaczona dla dużych organizacji	Organiczna (elastyczna i uczestnicząca, zachęcająca do współpracy), przeznaczona dla średnich i małych organizacji
Kontrola jakości	Dokładne planowanie i ścisła kontrola. Trudne i późne testowanie	Stała kontrola wymagań, projektu i rozwiązań. Stałe testowanie
Wymagania użytkowników	Szczegółowe i ustalone przed kodowaniem/implementacją	Interakcyjne pozyskiwanie
Koszt ponownego startu	Wysoki	Niski
Kierunek wytwarzania	Stały	Łatwy do zmiany
Testowanie	Po zakończeniu kodowania	Każda iteracja
Zaangażowanie klienta	Niskie	Wysokie
Dodatkowe umiejętności	Nic szczególnego	Umiejętności interpersonalne i podstawowa wiedza o biznesie

Aspekt	Metodyki tradycyjne	Metodyki zwinne
wymagane od deweloperów		
Odpowiednia skala projektu	Wielka skala	Mała i średnia skala
Deweloperzy	Zorientowani na plan, z odpowiednimi umiejętnościami, dostęp do zewnętrznego źródła wiedzy	Zwinni, z zaawansowaną wiedzą, wspólnie pracujący i współdziałający
Klienci	Z dostępem do wiedzy, współdziałający, reprezentatywni i umocowani	Oddelegowani, posiadający wiedzę, współdziałający, reprezentatywni i umocowani
Wymagania	Bardzo stabilne, znane z góry	Emergentne, szybkozmienne
Architektura	Projekt dla bieżących i przewidywanych wymagań	Projekt dla bieżących wymagań
Przemodelowanie	Kosztowne	Tanie
Rozmiar	Wielkie zespoły i projekty	Małe zespoły i projekty
Główne cele	Wysokie bezpieczeństwo	Szybka wartość

Podsumowując porównanie zawarte w Tab. 1.1, można zauważyć, że różnice w zwinnych i tradycyjnych metodykach są duże i występują w większości wymienionych aspektów. Jest to powodem wysokiego odsetka nieudanych prób wprowadzenia zwinnych metodyk w tradycyjnych organizacjach [68].

1.1.5. Modele zwinnej transformacji organizacji IT

Zwinne metodyki wytwarzania są często dobrze przyjmowane zarówno przez kierownictwo, jak programistów, ponieważ uwalniają od wielu formalnych obciążeń narzucanych przez tradycyjne metodyki wytwarzania. Wprowadzenie zwinnych metodyk jest zwykle spostrzegane w perspektywie „wszystko albo nic". W praktyce jednak niewiele organizacji jest w stanie psychologicznie lub technicznie zaadaptować zwinne metodyki w krótkim czasie. Zwykle transformacja organizacji IT trwa kilka lat. Może odbywać się przyrostowo, w procesie adoptującym do organizacji kolejne praktyki zwinnych metodyk. Nawet wtedy trudno wybrać praktykę odpowiednią do sytuacji tak, aby nie utracić entuzjazmu wśród członków zespołów projektowych [100] oraz żeby nie wzbudzić w zespołach projektowych zjawiska oporu wobec zmiany [39]. Aby pomóc kierownictwu organizacji i inżynierom wytwarzania w podejmowaniu decyzji dotyczących procesu transformacji powstały próby jego opisu w postaci modelu.

Do tej pory zaproponowano kilka modeli zwinnej transformacji organizacji IT. Sidky i inni [93] zaproponowali wskaźnik oceny zwinności SAMI, aby oszacować stopień, w jakim organizacja może stać się zwinna. Na podstawie tego indeksu zdefiniowano proces dostosowania do zwinności, którego pierwszym krokiem jest wyszukanie czynników przerywających – wskaźników braku gotowości organizacji do rozpoczęcia zwinnej transformacji. Wynikiem pierwszego etapu jest informacja, czy zwinna transformacja może zostać rozpoczęta. Drugim etapem procesu dostosowania jest oszacowanie możliwości dostosowania organizacji do zwinności, którego wynikiem jest stopień potencjalnej zwinności. Bazując na nim, w trzecim, ostatnim etapie, proponowany jest zestaw najbardziej odpowiednich zwinnych praktyk. Model został z powo-

dzeniem użyty do doboru zwinnych metodyk [94]. Pomimo elastyczności modelu proponującego zestaw zwinnych praktyk z uwzględnieniem wskaźnika potencjalnej zwinności organizacji, nie gwarantuje jednak on tego, że zaproponowany zbiór zwinnych praktyk będzie najlepszym wyborem dla organizacji.

Model Agile Adoption and Improvement Model (AAIM) wprowadza sześć poziomów dostosowania zwinności [77]. Na każdym z nich AAIM szuka możliwości włączenia odpowiednich zwinnych zasad do organizacji. Model wykorzystuje technikę pomiaru zwinności 4–DAT, który dostarcza konkretnych wskaźników do ilościowej oceny poziomu zwinności procesu wytwarzania oprogramowania. Jednak oparcie tego modelu na ilościowej skali pomiarowej sprawia, że jest zależny od poprawności sum, które powinny reprezentować stopień zwinności procesu.

Zaproponowano również, wynikający z doświadczenia, model dostosowania do zwinnych praktyk zbudowany na sytuacyjnej inżynierii metod w oparciu o fragmenty metod i procesów [56]. Znaczącym wkładem tego modelu jest nacisk na użycie uprzednich doświadczeń organizacji ze stosowaniem zwinnych praktyk przy proponowaniu zbioru praktyk do zastosowania.

Opracowano także model dostosowania zwinnych praktyk w rozproszonym środowisku [98]. Model posiada wsparcie w narzędziu, zbudowanym na podstawie [13], oceniającym stopień zwinności i sformalizowania niezbędny w projekcie. Użycie modelu jest podzielone na cztery etapy:

- ewaluacja – określenie stopnia rozproszenia projektu,
- rozpoczęcie – utworzenie rozproszonych zespołów projektowych i określenie sposobu, w jaki zwinne praktyki zostaną włączone w infrastrukturę wytwarzania,
- przejście – włączenie zwinnych praktyk w czynności procesu wytwarzania,
- stan stabilny – aby określić podstawy do płynnego dodawania kolejnych rozproszonych zespołów do projektu.

Model Strategic Agile pre–Adoption analysis Framework (SAAF), którego celem jest analiza zwinnych praktyk, w celu określenia ich potencjalnego wpływu na strategiczne cele organizacji oraz możliwości rozwiązania przez nie problemów występujących w bieżącym procesie wytwarzania [28]. W tym celu zaproponowano zbiór technik analizy zwinnych praktyk i zbiór komponentów wspierających tą analizę. Dzięki podjęciu tej analizy przed przyswojeniem nowych praktyk można przewidzieć niedopasowanie między strategicznymi celami organizacji a rozważanymi praktykami. Model SAAF nie dostarcza jednak narzędzi do analizy bieżącego procesu wytwarzania oprogramowania służących identyfikacji problemów.

Podsumowując, większość aktualnych modeli zwinnej transformacji posiada trzy wspólne cechy [84]:

- zwykle składają się z poziomów,
- każdy proponuje miarę oceny stopnia zwinności organizacji lub procesu,
- zapewniają sposób zarządzania zależnościami zwinnych praktyk, gdy są iteracyjnie włączane do organizacji.

Innym nurtem w zwinnej transformacji jest podejście adaptacyjne, które jest oparte na adaptacyjnej teorii strukturacji [50]. Opisywany jest m.in. model adaptacyjny zwinnych metodyk, bazujący na teorii strukturacji i wielu studiach przypadków [15]. Zaproponowano również dwa podejścia do adaptacji fragmentów zwinnych metodyk zgodnie ze współczynnikami organizacji i projektu [5]. Uwzględniono także fakt, że zwinne praktyki są wypierane przez praktyki alternatywne, zaproponowano uprzywilejowanie zwinnych praktyk podczas ich adaptacji w organizacji [66].

Obok modeli zwinnej transformacji powstałych w wyniku prac badawczych, istnieją także modele komercyjne, które są opracowane i wykorzystywane przez firmy konsultingowe do wspomagania transformacji organizacji ich klientów. Przykładem modelu komercyjnego jest The Enterprise Transition Framework (ETF) będący własnością firmy Agile42 przedstawiony na stronie internetowej [112]. Zastosowanie modelu ETF jest podzielone na cztery fazy:

- oszacowania – początkowa ocena aktualnego wielowymiarowego stanu organizacji, która pozwala opracować właściwą zwinną strategię;
- strategii – zespół liderów opracowuje pierwszą wersję mapy zwinnej strategii, która łączy informacje o stanie organizacji z celami biznesowymi,
- pilotażu – zostaje wybrany projekt pilotażowy, który pozwoli zrealizować bez ryzyka eksperymenty, dzięki którym ludzie, zespoły i organizacja uczy się funkcjonować w nowy sposób;
- wdrożenia – po zakończeniu z sukcesem projektu pilotażowego, zespół liderów inicjuje wdrożenie procesu w całej organizacji.

Wymienione cztery fazy są powtarzane w cyklu, który ma gwarantować ciągłe ulepszenie procesów biznesowych organizacji podlegającej transformacji. Jako że jest to model komercyjny, udostępniono jedynie informacje ogólnikowe, niezbędne z marketingowego punktu widzenia. Najbardziej interesująca, z punktu widzenia oceny gotowości metoda oszacowania stanu organizacji, nie została jednak opisana ani sformalizowana w dostępnych materiałach.

Modele zwinnej transformacji dostarczają narzędzi do oceny zwinności praktyk stosowanych w organizacji i na tej podstawie określają które zwinne praktyki i w jakiej kolejności należy wprowadzić. Brakuje im jednak poza nielicznymi wyjątkami metody oceny, czy i w jakim stopniu organizacja jest gotowa do rozpoczęcia procesu zwinnej transformacji.

Z praktyki wynika, że o powodzeniu zwinnej transformacji decyduje gotowość organizacji do realizacji złożonych projektów informatycznych.

Definicja 1.3 Gotowość organizacji do zwinnej transformacji jest to zdolność do realizacji różnorodnych projektów informatycznych zgodnie z przyjętą metodyką.

Określenie stopnia gotowości organizacji, przed podjęciem decyzji o transformacji tradycyjnych procesów wytwórczych, pozwoli ograniczyć ryzyko, którym jest obarczona transformacja. Transformacja powinna być rozpoczynana tylko w przypadku osiągnięcia przez organizację odpowiedniego stopnia gotowości.

Stopień gotowości organizacji może być zależny od oceny tradycyjnych procesów wytwarzania stosowanych przed transformacją [107, 108]. W celu określenia gotowości organizacji, należy więc dokonać oceny jej procesów wytwarzania oprogramowania.

1.1.6. Modele dojrzałości organizacji IT

Modele dojrzałości procesów wytwarzania oprogramowania umożliwiają klasyfikację wydajności organizacji IT zajmujących się wytwarzaniem oprogramowania. Modele dojrzałości otwierają przed organizacjami IT drogę do ciągłego usprawniania ich procesów i wyznaczają na niej kierunek zmian. Model dojrzałości może być użyty zarówno w połączeniu z tradycyjnymi, jak i zwinnymi metodykami wytwarzania oprogramowania. Poniżej przedstawiony zostanie najbardziej rozpowszechniony model dojrzałości CMMI. Następnie zostanie omówiona możliwość połączenia modelu CMMI ze zwinnymi praktykami. Przykładowe wdrożenie metodyki Scrum w organizacji o wysokim poziomie dojrzałości pokazuje, że modelu CMMI można użyć do kierowania procesem zwinnej transformacji organizacji IT.

Model dojrzałości CMMI

Model CMM (Capability Maturity Model) powstał w roku 1991 na bazie najlepszych praktyk w wytwarzaniu oprogramowania. Opisuje on ewolucyjną metodę ulepszania procesów organizacji z prowadzonych ad hoc i niedojrzałych do zdyscyplinowanych i dojrzałych [71]. Model CMM został opracowany przez Software Engineering Institute przy uniwersytecie Carnegie Mellon w Pittsburghu. Model ten miał wielki wpływ na ulepszenie procesów wytwarzania i poprawę jakości oprogramowania na całym świecie [72].

W roku 2002 opracowano nową i rozszerzoną wersję modelu o nazwie CMMI [20]. Ostatnie i w akronimie oznacza integrację. Ta wersja modelu integruje inżynierię oprogramowania, inżynierię systemów oraz praktyki pozyskiwania produktów i usług w jeden model dojrzałości.

CMMI definiuje 25 obszarów procesowych do zrealizowania. Dla każdego obszaru procesowego określone są wymagane cele, spodziewane praktyki i zalecane praktyki. Co więcej, zbiór ogólnych praktyk musi być zrealizowany dla wszystkich procesów. Jednak w przeciwieństwie do modelu CMM, nowy model większą wagę przykłada do realizacji celów i nie skupia się na realizowanych praktykach [18]. Oznacza to, że celem nowego modelu dojrzałości jest sprawdzenie czy organizacja potrafi skutecznie działać, a nie w jaki sposób to osiąga.

Poziom dojrzałości jest scharakteryzowany przez zbiór obszarów procesowych, powiązanie to zostało pokazane w Tab. 1.2. Wartość poziomu dojrzałości jest określana przez osiągnięcie ogólnych i specyficznych celów należących do związanych z nim obszarów procesowych [20].

Tab. 1.2 Poziomy dojrzałości wraz z odpowiadającymi im obszarami procesowymi modelu CMMI

Źródło: [20]

Poziom dojrzałości	Obszary procesowe
1. **Początkowy** – oprogramowanie tworzone chaotycznie, bez żadnych formalnych procedur	brak
2. **Zarządzany** – stosowane są podstawowe techniki śledzenia projektu – śledzi się koszt, harmonogram oraz funkcjonalność. Stosuje się techniki pozwalające na powtarzanie udanych projektów na podstawie informacji zapisanych przy okazji poprzednich	Zarządzanie wymaganiami, Planowanie projektu, Monitoring i kontrola projektu Zarządzanie umowami z poddostawcami Miary i analizy Zapewnienie jakości procesu i produktu Zarządzanie konfiguracją
3. **Zdefiniowany** – proces wytwórczy jest opisany, wszystkie wykonywane czynności są udokumentowane w postaci procedur lub instrukcji. Standardy i procedury modyfikowane są w miarę zmieniających się uwarunkowań technologicznych lub organizacyjnych.	Rozwój wymagań, Rozwiązania techniczne Integracja produktu, Weryfikacja, Walidacja Koncentracja na procesie organizacyjnym Definicja procesu organizacyjnego Szkolenia organizacyjne Zintegrowane zarządzanie projektem + zintegrowany rozwój produktu i procesu Zarządzanie ryzkiem, Analiza decyzji i rozwiązań
4. **Zarządzany ilościowo** – podczas projektów stosuje się szczegółowe metryki dotyczące samego procesu oraz jakości produktu	Przebieg procesu organizacyjnego Ilościowe zarządzanie projektem
5. **Optymalizujący** – stosuje się praktyki mające na celu ciągłe poprawianie procesu wytwarzania oprogramowania	Innowacje organizacyjne i wdrożenia Analiza przyczyn i rozwiązań

W reprezentacji stałej poziom **początkowy** (poziom 1) jest opisany jako procesy niestabilne, chaotyczne i działania doraźne. Sukces organizacji sklasyfikowanej na tym poziomie bardziej zależy od kompetencji poszczególnych osób i ich heroicznych wysiłków, niż od realizacji dobrze zdefiniowanego procesu biznesowego. Organizacja na takim poziomie jest w stanie wydać dobry produkt, chociaż większość projektów jest opóźniona lub ma przekroczony budżet. Co więcej organizacje na tym poziomie w momentach kryzysowych zwykle porzucają ustalony proces.

Poziom **zarządzany** (poziom 2) charakteryzuje organizacje, które osiągnęły wszystkie ogólne i specyficzne cele wyznaczone dla tego poziomu. Wymagania projektu są zarządzane, procesy są planowane, wykonywane, szacowane i kontrolowane. Porządek i dyscyplina są utrzymywane w czasach kryzysu, o ile określone praktyki przeważają. Wtedy projekty są realizowane i zarządzane zgodnie z planem i dokumentacją. Celem tego poziomu dojrzałości jest uzyskanie zarządzanych i identyfikowalnych wymagań, procesów, produktów i usług.

Poziom **zdefiniowany** (poziom 3) charakteryzuje organizacje, które osiągnęły wszystkie ogólne i specyficzne cele wyznaczone dla poziomów niższych oraz tego poziomu. Różnica między poziomem 2 a poziomem 3 występuje w zakresie obowiązywania norm, opisów procesów i procedur. Na poprzednim poziomie normy, procesy i procedury mogły różnić się dla każdego projektu. Na poziomie 3 są one ujednolicone dla całej organizacji, a wszystkie wyjątki

powinny być udokumentowane. Kolejna różnica to poziom szczegółowości opisu i oceny procesu. Na tym poziomie procesy są zarządzane aktywnie na podstawie wiedzy o wzajemnych relacjach aktywności, produktów i usług.

Poziom **zarządzany ilościowo** (poziom 4) charakteryzuje organizacje, które osiągnęły wszystkie ogólne i specyficzne cele wyznaczone dla poziomów niższych oraz tego poziomu. Na tym poziomie wybrane podprocesy podnoszą wydajność procesu przez użycie statystycznych i innych ilościowych technik jako kryterium zarządzania. Jakość i wydajność są rozumiane jako określenia statystyczne i zarządzane przez cały cykl życia procesu. Zmiany wydajności procesu są rozpoznawane i korygowane, aby zapobiec w przyszłości podobnym zdarzeniom. Wartości ilościowych miar wydajności procesu są przechowywane w repozytorium pomiarów, aby ułatwić podejmowanie w przyszłości decyzji na podstawie aktualnych danych. Główna różnica między poziomem 3 i 4 dotyczy przewidywalności wydajności procesu. Na poziomie 4 wydajność jest wyznaczana na podstawie danych ilościowych, na poziomie 3 na podstawie danych jakościowych.

Poziom **optymalizujący** (poziom 5) charakteryzuje organizacje, które osiągnęły wszystkie ogólne i specyficzne cele. Na tym poziomie procesy są udoskonalane w sposób ciągły, na podstawie ilościowej analizy głównych przyczyn zmienności procesu. Działania na tym etapie mają na celu ciągłe doskonalenie procesów wytwórczych, dzięki przyrostowemu ulepszaniu i zastosowaniu nowoczesnych technologii. Te ulepszenia są mierzone przez ilościowe wskaźniki i są w sposób ciągły monitorowane, aby nadążać za zmianami w sytuacji biznesowej. Krytyczna różnica między poziomem 4 a poziomem 5 polega na rodzaju obserwowanych zmian w procesach. Na poziomie 4, nacisk jest skierowany na wybrane przyczyny zmienności procesów, więc można osiągnąć statystyczną przewidywalność wyników. Jednak, mimo że proces daje przewidywalne wyniki, to mogą one nie wystarczyć do osiągnięcia założonych celów. Na poziomie 5 główne ośrodki koncentrują się na wspólnych przyczynach zmienności procesu w celu poprawy jego wydajności. Dzięki temu cele ilościowe w zakresie doskonalenia procesów są osiągane przy jednoczesnym zachowaniu wyjściowej statystycznej przewidywalności.

Ważne jest, aby pamiętać, że modele procesu CMMI nie zawierają opisów gotowych do użycia procesów wytwarzania. Zamiast tego CMMI zapewnia sposób oceny stanu zdolności organizacji do wytwarzania oprogramowania w sposób powtarzalny i przewidywalny.

Na podstawie danych zamieszczonych w Tab. 1.2 można zauważyć, że poziomy dojrzałości 1–3 odnoszą się do procesów wytwarzania oprogramowania, natomiast poziomy 4–5 opisują procesy organizacyjne na poziomie całego przedsiębiorstwa.

Zastosowanie modelu CMMI do zwiększenia wydajności procesu wytwarzania oprogramowania jest wyzwaniem dla całej organizacji. Herbsleb i inni pokazali, że średni czas wymagany dla podniesienia poziomu dojrzałości organizacji o jeden poziom wynosi między 21 a 37 miesięcy [40]. Ponad trzy czwarte organizacji doświadczyło, że realizacja głównych działań w celu poprawy procesu wytwarzania trwało dłużej, niż to było oczekiwane. Wysiłek z tym związany opłaca się, ponieważ „dojrzałość zarządzania procesem wytwarzania oprogramowania jest pozytywnie skojarzona z wydajnością projektu” [45].

Połączenie modelu dojrzałości CMMI i zwinnych metodyk wytwarzania

Model dojrzałości CMMI jest powiązany z tradycyjnymi metodykami wytwarzania oprogramowania. Obszary procesowe oraz ogólne i szczegółowe cele są na wysokim poziomie abstrakcji, natomiast praktyki wykorzystywane do osiągnięcia tych celów wynikają z metod tradycyjnych i dziedziczą z nich wszystkie negatywne cechy, jak nadmiar biurokracji, tworzenie zbędnej dokumentacji i koncentracja na procesach, zamiast na ludziach. Od czasu ogłoszenia manifestu zwinnych technik programowania [118] rozważana jest możliwość połączenia tych dwóch światów: modelu CMMI i zwinnych metodyk wytwarzania oprogramowania. Twórca modelu SW–CMM Paulk już w roku 2001 zadał pytanie, czy można zrealizować cele modelu dojrzałości przy pomocy technik programowania ekstremalnego [73]. Wynikiem prowadzonych przez niego badań była koncepcja połączenia dobrych stron obydwu podejść. W kolejnych latach problem połączenia modelu dojrzałości ze zwinnymi metodykami jest rozważany ponownie [41, 95, 101]. Dostarczane są przykłady zastosowań i opracowywane techniki zwiększania wydajności [63]. Najczęstszym wynikiem analizy realizacji celów modelu CMMI przez praktyki zwinnych metodyk jest możliwość osiągnięcia trzeciego poziomu dojrzałości [34], jednak opisywane są również przykłady osiągnięcia najwyższego poziomu dojrzałości modelu CMMI przez zastosowanie praktyk metodyki Scrum [43, 99]. W podanym przykładzie wprowadzenie organizacji z poziomu 1 dojrzałości modelu CMMI na poziom 5 spowodowało zmniejszenie pracochłonności projektu do 69% wartości początkowej. Następnie w tej samej organizacji przeprowadzono zwinną transformację tak, aby utrzymać uzyskany najwyższy poziom dojrzałości. Do realizacji celów ogólnych i szczegółowych modelu CMMI zastosowano dobre praktyki metodyki Scrum. Po udanym procesie zwinnej transformacji obniżono pracochłonność projektów do 35% wartości początkowej.

1.2. Podsumowanie stanu badań

Celem rozdziału było wprowadzenie w tematykę oceny procesów wytwarzania oprogramowania. Złożyło się na to omówienie metodyk wytwarzania oprogramowania, pokazanie aktualnego stanu badań nad modelami ułatwiającymi zmianę metodyk organizacji oraz ich związek z modelami oceny dojrzałości organizacji. Podsumowanie rozdziału zakończy się propozycją autorskiej metody oceny i planowania procesów wytwórczych organizacji.

Tradycyjne – ciężkie metodyki wytwarzania opierają się na planowaniu, przestrzeganiu procedur i zgodności z procesem, formalnej komunikacji oraz szczegółowej dokumentacji. Zwinne – lekkie metodyki kładą nacisk na reagowanie na zmiany, bezpośrednią komunikację i współpracę z klientem, aby uzyskać wartościowy produkt.

Ciężkie i lekkie metodyki różnią się niemal w każdym aspekcie, co pokazuje zestawienie w Tab. 1.1. Mimo tych różnic, działania składające się na wytwarzanie oprogramowania, czyli pozyskiwanie wymagań, projektowanie, kodowanie i testowanie są w obydwu metodykach takie same. Różni się jednak perspektywa, w której są one widziane. Największa różnica tkwi w aspekcie ludzkim, stąd najistotniejsza do wprowadzenia zwinnych metodyk jest zmiana kultury organizacji.

Zwinna transformacja organizacji IT jest trudnym procesem, który często kończy się niepowodzeniem. Skutkiem nieudanej transformacji jest nie tylko niedostarczenie projektu, często taka

organizacja wycofuje się z procesu transformacji i wraca do nieskutecznych i niekonkurencyjnych metodyk ciężkich. Proces transformacji jest badany od lat i tworzone są modele ułatwiające przeprowadzenie tej radykalnej zmiany.

Większość modeli zwinnej transformacji, opisanych w przedstawionej literaturze jako pierwszy etap przyjmuje ocenę aktualnego stanu organizacji lub stanu procesu wytwórczego organizacji. Jednak żaden z modeli nie precyzuje w jaki sposób istniejące procesy wytwórcze mają być mierzone czy analizowane. Modele w ocenie istniejących procesów ograniczają się jedynie do identyfikacji zastosowanych w nich zwinnych praktyk. Tylko model SAAF w swojej pierwszej fazie oceny określa, czy organizacja jest gotowa do przeprowadzenia zwinnej transformacji. Nie dostarcza jednak technik pomiaru procesów organizacji podlegającej zmianom. Metoda oceny efektywności organizacji i jej procesów jest zdefiniowana przez modele dojrzałości organizacji.

1.2.1. Tradycyjne podejście do oceny procesów wytwarzania

Większość organizacji wytwarzających oprogramowanie używa i stosuje metodyki umożliwiające wsparcie i ulepszanie procesów wytwórczych, ponieważ badania empiryczne pokazały, że jakość procesu wytwarzania jest bezpośrednio związana z produktywnością organizacji i wysoką jakością wytworzonego oprogramowania [16, 57]. Najszerzej stosowaną z tych metodyk opisuje model dojrzałości organizacji CMMI.

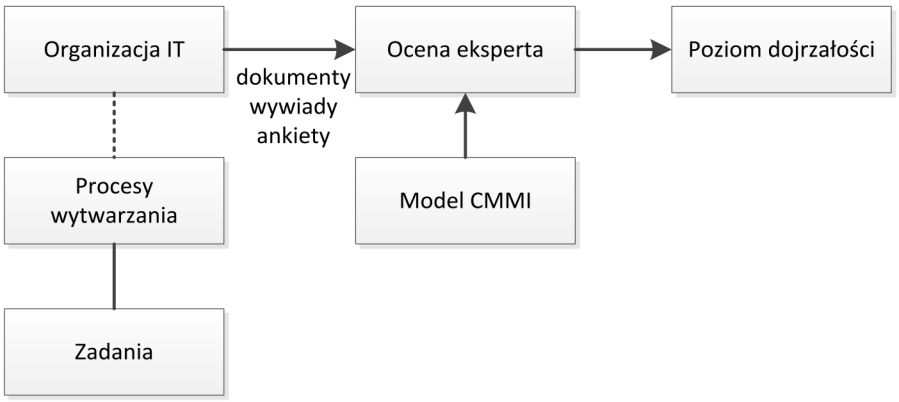
Użycie modelu CMMI do oceny dojrzałości organizacji z jednej strony umożliwia poprawę efektywności procesu wytwarzania i jakości wytworzonego produktu, a z drugiej strony może wskazywać na poziom ryzyka przeprowadzenia zwinnej transformacji. Na potrzeby niniejszej rozprawy założono, że między poziomami dojrzałości organizacji a poziomem ryzyka transformacji może istnieć zależność pokazana w Tab. 1.3. Ryzyko transformacji jest wysokie w przypadku organizacji, której procesy wytwarzania są przypadkowe, chaotyczne i nie realizują we właściwy sposób przyjętej metodyki wytwarzania oprogramowania. Brak zdefiniowanych procesów odpowiada w modelu CMMI pierwszemu poziomowi dojrzałości. Ryzyko transformacji organizacji jest na średnim poziomie, kiedy jej procesy wytwarzania są dobrze zdefiniowane i powtarzalne. Cele ogólne i szczegółowe związane z obszarami procesowymi poziomu drugiego można w łatwy sposób zrealizować za pomocą praktyk zwinnych metodyk. Niskie ryzyko transformacji jest osiągnięte wówczas, gdy procedury i normy dotyczące procesu wytwarzania są opisane i przestrzegane w całej organizacji. W tym wypadku warunki do sprawnego przebiegu transformacji organizacji są spełnione w wysokim stopniu.

Tab. 1.3 Zależność między poziomem dojrzałości organizacji a ryzykiem zwinnej transformacji
Źródło: Opracowanie własne

Poziom dojrzałości CMMI	Ryzyko transformacji
Poziom 1 – początkowy	Bardzo wysokie
Poziom 2 – zarządzany	Średnie
Poziom 3 – zdefiniowany i powyżej	Niskie

Tradycyjny przebieg oceny dojrzałości organizacji za pomocą modelu CMMI jest przedstawiony na Rys. 1.6. Ocena jest dokonywana przez eksperta lub zespół ekspertów na podstawie

wywiadów, ankiet i dokumentów dostarczonych przez badaną organizację. Eksperci na ich podstawie sprawdzają, które z celów obszarów procesowych są realizowane w organizacji i jakiemu poziomowi dojrzałości one odpowiadają.

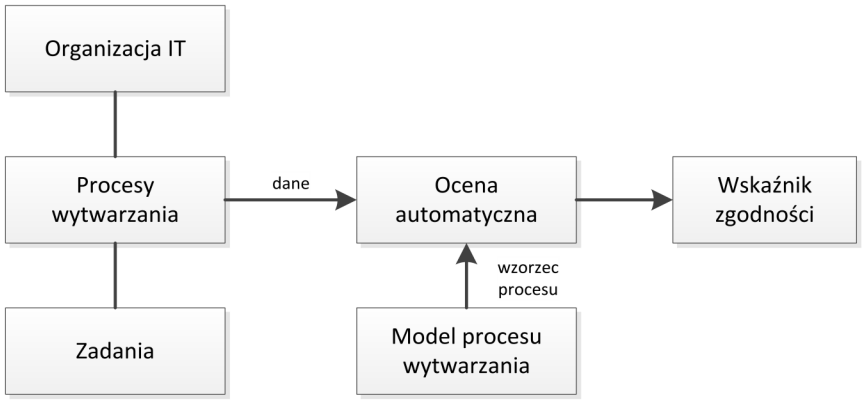


Rys. 1.6 Ocena dojrzałości organizacji w modelu CMMI
Źródło: Opracowanie własne

W trakcie oceny dojrzałości nie są bezpośrednio badane rzeczywiste przebiegi procesów organizacji, sprawdzana jest tylko dokumentacja. Zależność oceny od działań eksperta jest przyczyną jej ograniczeń: nieefektywności czasowej, ograniczeniu poziomem wiedzy eksperta i często subiektywnością oceny. Kolejnym ograniczeniem może być potrzeba badania artefaktów wytworzonych w procesie, ponieważ treść niektórych artefakty może być zastrzeżona lub niejawna.

1.2.2. Koncepcja oceny procesów wytwarzania wykorzystująca model

Koncepcja oceny procesów wytwarzania oprogramowania organizacji wykorzystująca model została opracowana w celu przezwyciężenia ograniczeń tradycyjnego podejścia do oceny. Przedstawiane podejście do oceny jest półautomatyczne i może być całkowicie zautomatyzowane, co zwiększa wydajność i pozwala oprzeć ocenę na analizie ilościowej dużych zasobów danych pochodzących bezpośrednio z procesu wytwarzania oprogramowania. Jednocześnie zmniejsza ono wymaganie co do zatrudnienia wysoko wykwalifikowanych pracowników, co obniża koszty przedsięwzięcia. Ocena jest podejmowana na podstawie analizy ilościowej, a nie jakościowej, jak w podejściu tradycyjnym co zwiększa jej obiektywizm. Schemat procesu wyznaczania wskaźnika oceny pokazano na Rys. 1.7.



Rys. 1.7 Koncepcja oceny procesu wytwarzania oprogramowania organizacji oparta na modelu

Źródło: Opracowanie własne

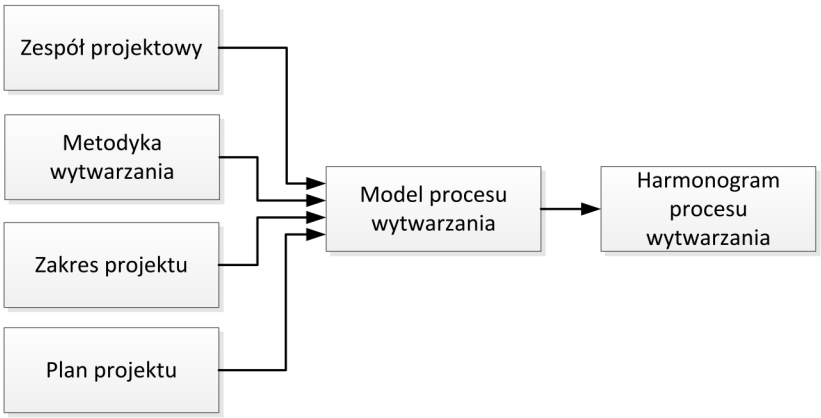
W procesie oceny działania ekspertów zostały zastąpione automatycznym porównaniem badanego procesu z wynikiem działania modelu. Dokumenty i wywiady zostały zastąpione przez dane dotyczące zadań wykonanych w procesie wytwarzania. Model dojrzałości CMMI został zastąpiony przez model procesu wytwarzania oprogramowania. Wynikowy wskaźnik oceny procesu jest to procent zadań wykonanych zgodnie z modelem. Przy porównywaniu wynikowej wartości procentowej z poziomami modelu CMMI, można przyjąć, że odpowiada poziomom dojrzałości CMMI od 1 do 3, ponieważ pierwsze trzy poziomy modelu CMMI badają realizację celów z obszarów dotyczących procesu wytwarzania.

Zastosowanie modelu procesu wytwarzania oprogramowania do oceny takiego procesu wypełnia lukę istniejącą w dotychczasowych badaniach. Model procesu wytwarzania dostarcza wzorca, który zostanie porównany badanym procesem wytwarzania. Model ten powinien umożliwiać opis procesu wytwarzania oprogramowania na poziomie abstrakcyjnym i ogólnym, aby uwzględniał różne metodyki wytwarzania oprogramowania i umożliwiał konfigurowanie metodyk na potrzeby badania procesów zaimplementowanych w organizacji.

Proponowana koncepcja oceny może służyć nie tylko do poprawiania efektywności procesów wytwarzania, podniesienia jakości wytworzonego oprogramowania, ale również może wskazywać na poziom ryzyka zwinnej transformacji organizacji. W takim wypadku proponowana ocena miałaby zastosowanie na wstępnym etapie istniejących modeli wspierających zwinną transformację. Ten wstępny etap oceny istniejących procesów wytwarzania mógłby pomóc w podjęciu decyzji o rozpoczęciu transformacji, której kolejne kroki wynikałyby z przyjętego modelu adaptacji zwinnych praktyk.

1.2.3. Planowanie procesu wytwarzania oparte na modelu

Model zastosowany w ocenie procesu wytwarzania, może być również wykorzystany do jego planowania. Koncepcja planowania wykorzystująca model została pokazana na Rys. 1.8.



Rys. 1.8 Koncepcja użycia modelu procesu wytwarzania do planowania.
Źródło: Opracowanie własne

Model na podstawie wartości zmiennych wejściowych, zawierających informacje o zespole projektowym, wybranej metodyce wytwarzania, zakresie i planie projektu tworzy szczegółowy harmonogram procesu wytwarzania danego projektu. Wynikowy harmonogram może być dostosowany do zadanego budżetu i terminu przez odpowiednie dobranie wartości parametrów wejściowych. Kierownik projektu, który jest odpowiedzialny za odpowiednie zaplanowanie harmonogramu prac w procesie wytwarzania oprogramowania dobiera skład zespołu projektowego, konfiguruje metodykę wytwarzania oraz określa plan projektu tak, aby projekt o określonym zakresie spełniał podane ograniczenia co do terminu zakończenia prac, kosztów, jakości i wymagań.

Przedstawione koncepcje opierają się na zastosowaniu modelu procesu wytwarzania oprogramowania. Projektowanie modelu rozpocznie się w następnym rozdziale od analizy środowiska projektowego i procesów wytwarzania oprogramowania w celu ustalenia odpowiedniego układu eksperymentu. Na podstawie tak wyznaczonego układu eksperymentu, w kolejnych rozdziałach zostanie utworzony projekt modelu podstawowego i uproszczonego.

2. Układ eksperymentu dla oceny procesu wytwarzania oprogramowania

2.1. Wprowadzenie

W poprzednim rozdziale przedstawiono problem oceny procesów wytwórczych organizacji IT oraz koncepcję jego rozwiązania dzięki zastosowaniu agentowo–obiektowego modelu procesu wytwarzania. Rozszerzeniem tej koncepcji jest zastosowanie modelu do planowania procesu wytwarzania. W celu konstrukcji modelu, mającego zastosowanie do oceny i planowania procesu wytwarzania, w bieżącym rozdziale zostanie określone czym jest system rzeczywisty, ustalony jego zakres oraz zmienne wejściowe i wyjściowe. Po przeprowadzeniu analizy systemu rzeczywistego zostanie ustalony układ eksperymentu, który jest niezbędny do wytyczenia zakresu i kierunku dalszych prac nad modelem.

System rzeczywisty jest to źródło danych obserwowalnych. Ważną właściwością systemu rzeczywistego jest określenie segmentu rzeczywistości, który powinien zostać wyodrębniony. Zgodnie z Rys. 2.1 system rzeczywisty posiada zestaw zmiennych wejściowych, które oddziałują na system z zewnątrz oraz zmienne wyjściowe, których wielkość jest rezultatem wartości przyjmowanych przez zmienne wejściowe.

Rys. 2.1 System rzeczywisty
Źródło: Opracowanie własne

Rozpatrywanym w tej pracy systemem rzeczywistym jest proces wytwarzania oprogramowania. Zmienne wejściowe i wyjściowe zostaną określone w kolejnym podrozdziale, po przeprowadzeniu analizy tego procesu przy zastosowaniu najważniejszych metodyk.

Układ eksperymentu zgodnie z definicją B. P. Zeiglera jest to „wyszczególnienie warunków w jakich system jest obserwowany lub w jakich przebiegają eksperymenty. Jako taki, układ eksperymentu jest operacyjnym sformułowaniem celów, które są motywacją projektowania systemu modelowania i symulacji.” [110].

W niniejszej rozprawie przyjęty układ eksperymentu wynika z potrzeby zastosowania projektowanego modelu do oceny i planowania procesów wytwarzania organizacji. Ocena badanego procesu będzie polegać na pomiarze zgodności harmonogramu badanego procesu z harmonogramem wzorcowym, wyznaczonym przez model. Układ eksperymentu zostanie określony w sposób formalny przez podanie definicji zmiennych wejściowych i wyjściowych systemu rzeczywistego.

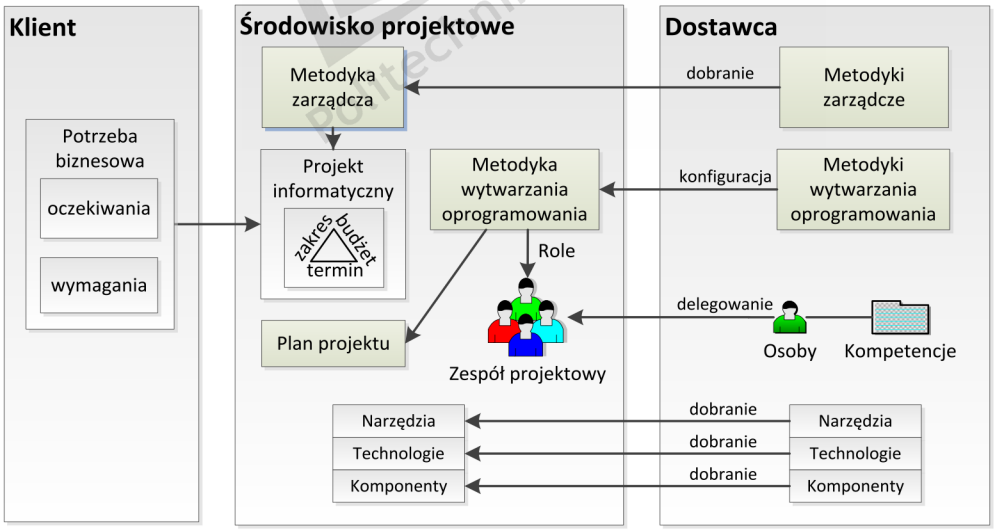
2.2. Analiza środowiska projektowego i procesu wytwarzania

W celu określenia zmiennych wejściowych procesu wytwarzania oprogramowania, należy rozpatrzeć jego fazę planowania, ponieważ konstruowany model ma wspierać planowanie procesu wytwarzania. W tej fazie zostaje opracowany plan wytwarzania projektu i wybrana odpowiednia metodyka zarządzania projektem oraz metodyka wytwarzania oprogramowania. Następnie przydzielane są odpowiednie zasoby do wykonania projektu, składające się z: członków zespołu projektowego, technologii informatycznych, wspierających je narzędzi, komponentów oraz serwerów i stacji roboczych. Tak skonstruowane z metodyk, zasobów i wiedzy bezpośrednie otoczenie projektu, zostało nazwane środowiskiem projektu. Pojęcie środowiska projektu w klasycznej literaturze dotyczącej zarządzania oznacza bliższe i dalsze otoczenie projektu, w postaci osób, organizacji, kontekstu politycznego, społecznego oraz prawnego, określanego w celu przeprowadzenia odpowiednich analiz [54]. W niniejszej pracy zastosowano pojęcie środowiska projektowego w węższym znaczeniu, często stosowane w literaturze dotyczącej zarządzania projektami informatycznymi [88].

Definicja 2.1 Środowisko projektowe składa się z projektu określonego przez trójkąt ograniczeń (zakres, budżet, termin), metodyk zarządzania i wytwarzania wraz z przydzielonymi do projektu zasobami w postaci wiedzy, technologii, komponentów oraz członków zespołu projektowego.

2.2.1. Zmienne wejściowe procesu wytwarzania oprogramowania

Analiza środowiska projektowego zostanie przeprowadzona w celu określenia zmiennych wejściowych procesu wytwarzania oprogramowania. Elementy środowiska projektowego i źródła ich pochodzenia pokazano na Rys. 2.2.



Rys. 2.2 Dobór elementów środowiska projektowego jako źródło wiedzy o zmiennych wejściowych systemu rzeczywistego

Źródło: Opracowanie własne

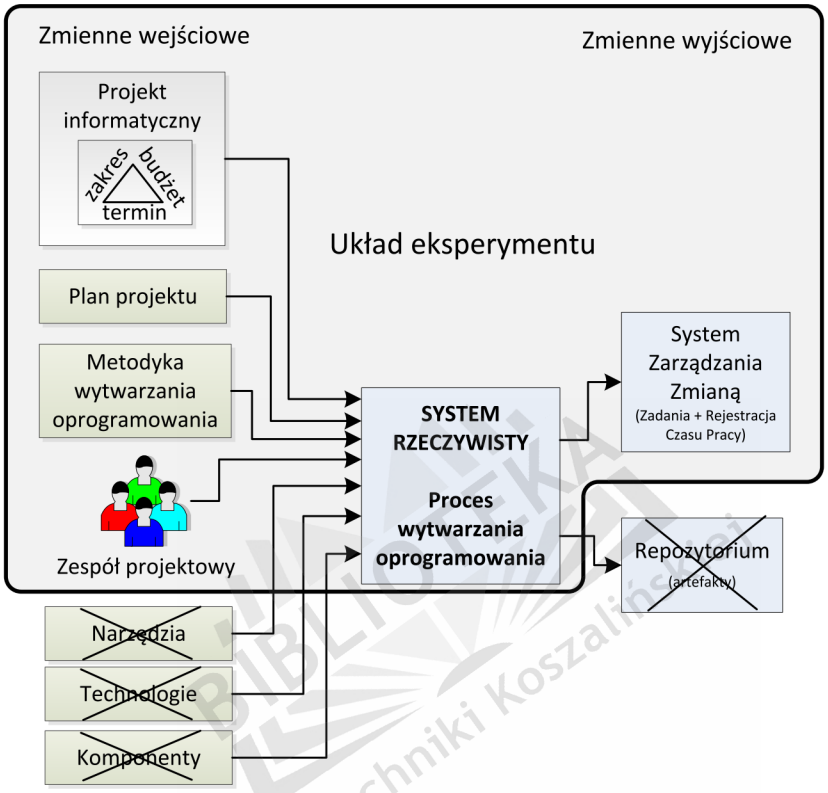
Środowisko projektowe powstaje jako wynik współpracy organizacji klienta z organizacją dostawcy, za jego utworzenie jest odpowiedzialny kierownik projektu [88]. Motywacją do rozpoczęcia projektu informatycznego jest potrzeba biznesowa klienta. Potrzeba biznesowa doprowadza do spotkania klienta z dostawcą oprogramowania. Klient przedstawia swoje oczekiwania i wymagania dotyczące projektu informatycznego. Na tej podstawie przedstawiciele organizacji klienta i dostawcy negocjują trójkąt ograniczeń projektu. W skład tego trójkąta wchodzi zakres, budżet i termin ukończenia projektu. Trójkąt projektowy jest pierwszym elementem środowiska projektowego utworzonym przez organizację dostawcy w celu wytworzenia oprogramowania zgodnie z potrzebami klienta. Organizacja dostawcy ze swoich zasobów dobiera i konfiguruje składowe elementy niezbędne do wytworzenia zamawianego produktu. Kolejnym składnikiem środowiska projektowego jest metodyka zarządzania projektem. Następnie z metodyk wytwarzania stosowanych dotychczas przez organizację dostawcy zostaje wybrana najbardziej odpowiednia dla rozpoczynającego się projektu. Wybrana metodyka zostaje skonfigurowana odpowiednio do projektu. Z praktyk zawartych w metodyce dostawca dobiera te, które zapewnią odpowiednią jakość wytworzonego oprogramowania. Kolejnym elementem środowiska projektowego jest zespół projektowy, do którego zostają delegowani pracownicy organizacji dostawcy. Osoby zostają przydzielone do zespołu projektowego ze względu na posiadane kompetencje i wymagane do obsadzenia role projektowe, wynikające z przyjętej metodyki wytwarzania i specyfiki projektu. Często osoby przydzielane do nowotworzonego projektu są już zaangażowane w innych projektach i w związku z tym część czasu pracy poświęcają innym zajęciom. Na podstawie przyjętej metodyki wytwarzania oprogramowania, wymagań klienta i składu zespołu projektowego zostaje opracowany plan projektu. Następnymi składnikami środowiska projektowego są narzędzia, które zwykle są dobierane przez dostawcę z ograniczonego zasobu narzędzi, na które dostawca ma wykupioną licencję lub narzędzi typu open source poznanych i wykorzystywanych wcześniej. Technologie są dobierane na podstawie wymagań niefunkcjonalnych przez architektów systemowych. Na podstawie wymagań funkcjonalnych prowadzona jest analiza możliwości użycia gotowych komponentów, modułów lub bibliotek będących w posiadaniu organizacji dostawcy. Gotowe komponenty zapewniające funkcjonalności określone w wymaganiach projektu dołączane są do powstającego środowiska projektowego.

2.2.2. Zmienne wyjściowe procesu wytwarzania oprogramowania

Członkowie zespołu projektowego wykonują pracę wynikającą z przydzielonych im zadań. Praca wykonywana podczas realizacji zadań przydzielonych członkom zespołu projektowego jest zapisywana w systemie rejestracji czasu pracy. Rejestracja czasu pracy jest zwykle zintegrowana z systemem zarządzania zmianą, w którym przechowywane są zadania przydzielane członkom zespołu projektowego. Z punku widzenia badania dojrzałości procesu wytwarzania oprogramowania, najistotniejszą informacją o przebiegu procesu wytwarzania jest kto, kiedy i jakiego typu wykonywał zadania. Te informacje, tworzone w trakcie procesu wytwarzania, są przechowywane w systemie zarządzania zmianą i na nich będzie oparty tworzony wzorzec procesu wytwarzania. Zatem zmienna wyjściowa zawierająca prace wykonane w procesie wytwarzania i ich harmonogram należą do układu eksperymentu.

W trakcie pracy nad zadaniami wytwarzane są dokumenty, diagramy, wymagania, elementy specyfikacji, kod źródłowy, przypadki testowe i inne elementy powstającego oprogramowania. Wszystkie te elementy nazywane są artefaktami i przechowywane są w repozytorium projektu,

do którego mają dostęp wszyscy członkowie zespołu projektowego. Przechowywanie artefaktów w repozytorium jest niezbędne do działania procesu, ponieważ do wykonania kolejnych zadań potrzebne są artefakty utworzone podczas wykonywania wcześniejszych zadań. Obserwacja zmian informacji zawartej w repozytorium, np. kiedy artefakty są utworzone, jest bezużyteczna z punktu widzenia oceny procesu wytwarzania i w związku z tym została pominięta w układzie eksperymentu.



Rys. 2.3 Zidentyfikowane zmienne wejściowe i wyjściowe procesu wytwarzania oprogramowania oraz przyjęty układ eksperymentu

Źródło: Opracowanie własne

Na Rys. 2.3 pokazano zidentyfikowane zmienne wejściowe i wyjściowe procesu wytwarzania oprogramowania. Zmienne, które nie należą do układu eksperymentu zostały na rysunku wykreślone. Biorąc pod uwagę ustalony cel i zastosowanie projektowanego modelu ograniczono liczbę zmiennych wejściowych systemu do czterech:

- Zakres projektu – informacja o rozmiarze projektu, wpływająca na pracochłonność, plan projektu i koszty,
- Metodyka wytwarzania – informacja o sposobie wykonywania prac nad projektem,
- Plan projektu – informacja, jak prace mają zostać podzielone na etapy i iteracje,
- Zespół projektowy – informacja, kto należy do zespołu projektowego i jakie są jego kompetencje.

Pozostałe zmienne wejściowe, to jest metodyka zarządzania, narzędzia, technologie i komponenty, nie zostały uwzględnione w układzie eksperymentu. Metodyka zarządzania została pominięta, ponieważ jej wpływ na proces wytwarzania jest dużo mniejszy, niż wpływ metodyki wytwarzania. Zmienna określająca narzędzia nie ma wpływu na proces wytwarzania, ponieważ

organizacje najczęściej stosują stały zestaw narzędzi wspierających wytwarzanie oprogramowania, w zależności od doświadczenia i zakupionych licencji. Zmienne określające technologie i komponenty stosowane w procesie wytwarzania nie zostały uwzględnione w układzie eksperymentu, ponieważ ich wpływ na ocenę procesu nie został potwierdzony przez dostępne dane.

2.2.3. Analiza procesu wytwarzania oprogramowania

W bieżącym podrozdziale, w celu uzyskania jak najbardziej ogólnego opisu zmiennych wejściowych modelu, proces wytwarzania oprogramowania zostanie przeanalizowany w perspektywie stosowania tradycyjnych i zwinnych metodyk wytwarzania. Umożliwi to opracowanie układu eksperymentu pozwalającego na planowanie i ocenę procesów wytwarzania realizowanych według różnych metodyk. W kolejnych podrozdziałach zostaną przeanalizowane procesy wytwarzania prowadzone według podstawowych metodyk opisanych w rozdziale 1.

Proces wytwarzania realizowany zgodnie z metodyką kaskadową

Metodyka kaskadowa, opisana w podrozdziale 1.1.1, stanowi podstawę, na której zbudowane są metodyki tradycyjne. Struktura zmiennych wejściowych powinna umożliwiać opis projektu wytwarzanego zgodnie z tą metodyką. Plan projektu składa się z sześciu etapów, w każdym z nich są wykonywane prace tylko określonego typu i są wykonywane w całości, bez podziału na iteracje. Np. W etapie analizy są zbierane i analizowane wszystkie istniejące wymagania, w etapie projektowania jest wykonany projekt całego systemu. Każdy z etapów kończy się weryfikacją i walidacją. W przypadku niezadawalających wyników proces wytwarzania może zostać wycofany do jednego z poprzednich etapów.

Powyższe cechy procesu powinny być reprezentowane przez następującą wartość zmiennych wejściowych:

- Zakres projektu – zakres projektu informatycznego zwykle jest oszacowany jako rozmiar wyrażony w punktach funkcyjnych (FP)[53] lub w roboczogodzinach np. W przypadku użycia metody COCOMO [11].
- Plan projektu – zawartość zmiennej powinna zawierać listę etapów do zrealizowania i przepływy działań w każdym z etapów. W każdym z etapów jest potrzeba określenia procentowego, jaką część zakresu należy w nim zrealizować. Standardowo jest to 100%, wartości większe pozwalają opisać sytuację ponownego wykonywania prac danego etapu. W Tab. 2.1 przedstawiono zawartość planu projektu w sytuacji, gdy w etapie testowania, w wierszu 4 tabeli, został wykryty błąd projektu. Proces powrócił do etapu projektowania, gdzie zmieniono 20% projektu. Ta zmiana spowodowała wzrost pracochłonności w kolejnych etapach projektu. Możliwość kilkukrotnego wykonywania etapów w przypadku błędów jest zapisana przy pomocy iteracji.
- Metodyka wytwarzania – powinna definiować typy zadań, pogrupowane w działania.
- Zespół projektowy – powinien zawierać nazwiska i role projektowe osób wchodzących w skład zespołu.

Zmienną wyjściową procesu jest harmonogram prac, który powinien zawierać informacje o etapie wykonania, typie zadania, wykonującej osobie, iteracji wykonania, czasie rozpoczęcia i długości trwania.

Tab. 2.1 Przykładowy plan projektu realizowanego zgodnie z metodyką kaskadową w przypadku wykrycia błedów projektu

Źródło: Opracowanie własne

L.p.	Etap	Zakres
1	Analiza	100%
2	Projektowanie	100%
3	Kodowanie	100%
4	Testowanie	100%
5	Projektowanie	120%
6	Kodowanie	120%
7	Testowanie	120%
8	Wdrożenie	120%

Proces wytwarzania realizowany zgodnie z metodyką spiralną

Metodyka spiralna, opisana w podrozdziale 1.1.1, należy do tradycyjnych i podobnie jak metodyka kaskadowa składa się z wielu kolejnych etapów. Jej analiza jest ważna, ponieważ jest rozszerzona względem metodyki kaskadowej o etapy analizy ryzyka i fakt, że oprogramowanie jest tworzone przyrostowo. Wartości zmiennych wejściowych są podobne, jak przy zastosowaniu metodyki kaskadowej:

- Zakres projektu – wartość procentowa zakresu do realizacji w etapie i iteracji pozwala zaplanować przyrostowe wytwarzanie oprogramowania oraz tworzenie kolejnych, ulepszonych prototypów metodyki spiralnej,
- Metodyka wytwarzania – powinna uwzględniać nowe zadania związane z analizą ryzyka,
- Zespół projektowy – powinna być rozszerzona o role osób zajmujących się analizą ryzyka.

Harmonogram prac metodyki kaskadowej jest wystarczający do opisu prac nad zadaniami metodyki spiralnej, co pokazują przykładowe początkowe etapy zamieszczone w Tab. 2.2.

Tab. 2.2 Przykładowy plan projektu procesu wytwarzania zrealizowanego zgodnie z metodyką spiralną.

Źródło: Opracowanie własne

L.p.	Etap	Zakres
1	Plan wytwarzania	33%
2	Przegląd	33%
3	Analiza ryzyka	33%
4	Prototyp	33%
5	Symulacje	33%
6	Wymagania	33%
7	Plan wytwarzania	66%
8	Przegląd	66%
9	Analiza ryzyka	66%
...	...	...

Proces wytwarzania realizowany zgodnie z metodyką Rational Unified Process (RUP)

Metodyka RUP, opisana w podrozdziale 1.1.1, podobnie jak poprzednie metodyki składa się z kolejnych etapów. Przyczyną zamieszczenia tutaj analizy metodyki RUP jest rozszerzenie przez nią dotychczas rozpatrywanych metodyk o możliwość wykonania każdego z etapów w kilku iteracjach. Iteracje umożliwiają przyrostowe wytwarzanie oprogramowania. Prace w RUP podzielne są na dyscypliny i w każdym z etapów wykonywane są prace należące do wielu dyscyplin. Metodyka RUP wprowadza również dużo, precyzyjnie określonych, ról projektowych.

W związku z tym, zmianie ulegną wartości następujących zmiennych wejściowych i wyjściowych:

- Zakres projektu powinien być oszacowany jako rozmiar wyrażony w punktach przypadków użycia UCP (Use Case Points) [19]. UCP jest standardową metodą szacowania projektów wytwarzanych przy użyciu metodyki RUP,
- Plan projektu – zawartość zmiennej powinna zawierać listę etapów do zrealizowania, przepływy działań w każdym z etapów oraz listę iteracji etapu. Dla każdej iteracji jest potrzeba określenia, jaki procent zakresu projektu ma zostać zrealizowany, osobno dla każdej z ośmiu dyscyplin, wyróżnionych przez metodykę RUP.
- Metodyka wytwarzania – zmienna powinna definiować typy zadań, pogrupowane w działania. Typy zadań powinny być uzupełnione o dyscyplinę, do której należą.
- Zespół projektowy – zmienna powinna zawierać nazwiska i role projektowe osób wchodzących w skład zespołu.

Zmienna wyjściowa procesu – harmonogram prac powinna być rozszerzona o informację o iteracji, w której zadanie było wykonane.

Tab. 2.3 Przykładowy plan projektu procesu wytwarzania zrealizowanego zgodnie z metodyką RUP

Źródło: Opracowanie własne

L.p.	Etap	Wyma–gania	Proje–ktowa–nie	Kon–stru–kcja	Testo–wanie	Wdra–żanie	Zarzą–dzanie	Środo–wisko
1	Rozpoczęcie	25%	5%	2%	2%	2%	15%	20%
2	Opracowanie	50%	40%	10%	20%	5%	30%	30%
3	Opracowanie	75%	70%	15%	50%	10%	45%	40%
4	Konstrukcja	85%	85%	45%	70%	15%	60%	50%
5	Konstrukcja	95%	95%	80%	80%	25%	75%	60%
6	Konstrukcja	100%	98%	98%	90%	35%	90%	70%
7	Wdrażanie	100%	100%	100%	100%	100%	100%	100%

Proces wytwarzania realizowany zgodnie z metodyką programowania ekstremalnego (XP)

Metodyka programowania ekstremalnego, opisana w podrozdziale 1.1.2, jest metodyką zwinną, przyrostową i iteracyjną. Jest analizowana, ponieważ istnieje potrzeba sprawdzenia, czy można opisać zwinny proces wytwarzania za pomocą dotychczas używanych zmiennych, opisujących procesy zrealizowane wg metodyk tradycyjnych. Metodyka XP, podobnie jak RUP, składa się z kolejnych etapów powtarzanych w iteracjach. O jej zwinności, oprócz realizacji wielu postulatów manifestu zwinnego wytwarzania oprogramowania, świadczy faza uprodukcyjnienia, którą kończy się każda iteracja wytwarzania. W tej fazie kolejne, małe wydanie jest dostarczane klientowi, gdzie przechodzi testy akceptacyjne i jest wdrażane do produkcji.

W związku z tym, zmianie ulegną wartości następujących zmiennych wejściowych:

- Zakres projektu powinien być oszacowany jako rozmiar wyrażony w punktach przypadków użycia UCP (Use Case Points) [19]. UCP jest standardową metodą szacowania projektów wytwarzanych przy użyciu RUP, jednakże może być zastosowana do metodyk zwinnych przy założeniu, że przypadek użycia odpowiada historii użytkownika [4].
- Plan projektu – wartość zmiennej powinna objąć listę etapów do zrealizowania, przepływy działań w każdym z etapów oraz listę iteracji etapu. Dla każdej iteracji jest potrzeba określenia, jaki procent zakresu projektu ma zostać zrealizowany na jej końcu. Przykładowy plan projektu dla dwóch przyrostowych wydań oprogramowania podano w Tab. 2.4.
- Metodyka wytwarzania – zmienna powinna definiować typy zadań, pogrupowane w działania.
- Zespół projektowy – zmienna powinna zawierać nazwiska i role projektowe osób wchodzących w skład zespołu.

Zmienna wyjściowa procesu – harmonogram prac powinien być zapisany tak, jak dla metodyki RUP.

Tab. 2.4 Przykładowy plan projektu procesu wytwarzania zrealizowanego zgodnie z metodyką XP

Źródło: Opracowanie własne

L.p.	Etap	Zakres
1	Faza eksploracji	100%
2	Faza planowania	50%
3	Iteracje do wydania	50%
4	Faza uprodukcyjniania	50%
5	Faza planowania	100%
6	Iteracje do wydania	100%
7	Faza uprodukcyjniania	100%
8	Faza utrzymania	100%

Proces wytwarzania realizowany zgodnie z metodyką Scrum

Metodyka Scrum, opisana w podrozdziale 1.1.2, jest istotna dla prowadzonej analizy, ponieważ różni się znacznie od poprzednio analizowanych metodyk. Jest metodyką zwinną, która nie składa się z etapów, mimo że oprogramowanie jest wytwarzane iteracyjnie i przyrostowo. Scrum składa się z następujących zdarzeń, powtarzanych cyklicznie:

- Spotkanie planowania sprintu – spotkanie rozpoczynające sprint,
- Sprint – jest to ograniczona czasowo iteracja, która zwykle trwa 1, 2 lub 4 tygodnie. W trakcie sprintu wykonywane są prace ze wszystkich dyscyplin
- Spotkanie przeglądu sprintu – jest to spotkanie zamykające etap prac nad sprintem
- Retrospektywa sprintu – jest to spotkanie podsumowujące etap prac nad sprintem
- Codzienny scrum – jest to codzienne spotkanie zespołu projektowego w celu synchronizacji prac nad zadaniami i wymiany informacji między członkami zespołu projektowego
- Scrum of scrums – jest to codzienne spotkanie delegatów zespołów projektowych w celu synchronizacji prac nad projektem i wymiany informacji między zespołami projektowymi

W związku z odmienną organizacją procesu wytwarzania, zmienne wejściowe i wyjściowe powinny zostać dostosowane w następujący sposób:

- Zakres projektu powinien być oszacowany jako rozmiar wyrażony w punktach przypadków użycia UCP (Use Case Points)[19]. UCP jest standardową metodą szacowania projektów wytwarzanych przy użyciu RUP, jednakże może być zastosowana do metodyk zwinnych przy założeniu, że przypadek użycia odpowiada historii użytkownika [4].
- Plan projektu – w Tab. 2.5 pokazano przykładową wartość zmiennej. Mimo, że metodyka nie zawiera etapów, w celu ujednolicenia struktury zmiennych, przyjęto konwencję, zgodnie z którą zdarzenia Scruma zostały zapisane jako etapy. Etapy zostały wykorzystane w tym celu, ponieważ to etap zawiera informację o przepływie działań. Ta informacja jest także niezbędna do opisania zdarzenia, ponieważ każde zdarzenie składa się z innych działań składowych. Iteracja również została zreinterpretowana i wykorzystana ponownie. Została użyta do opisu realizacji zdarzenia w każdym ze sprintów. Dodatkowo do iteracji został dołączony numer sprintu i czas trwania zdarzenia. Numer sprintu pozwala określić zdarzenia cykliczne, występujące codziennie spotkania, dla

których określona jest również godzina rozpoczęcia. Czas trwania zdarzenia jest odpowiednikiem zakresu projektu do wykonania w iteracji. W metodyce Scrum w sprincie wykonywany jest taki zakres projektu, jaki uda się wytworzyć w określonym czasie.

- Metodyka wytwarzania – zmienna powinna definiować typy zadań, pogrupowane w działania.
- Zespół projektowy – zmienna powinna zawierać nazwiska i role projektowe osób wchodzących w skład zespołu. Metodyka Scrum zawiera jedynie trzy role projektowe, które nie wystarczają do precyzyjnego określenia kompetencji członków zespołu projektowego. Proponujemy zatem użycie ról projektowych metodyki RUP do uszczegółowienia ról metodyki Scrum.

Zmienna wyjściowa procesu – harmonogram prac powinien być rozszerzony o informację o iteracji oraz o sprincie, w których zadanie było wykonane.

Tab. 2.5 Przykładowy plan projektu procesu wytwarzania zrealizowanego zgodnie z metodyką Scrum

Źródło: Opracowanie własne

L.p.	Etap – zdarzenie	Sprint	Czas trwania	Rozpo–częcie
1	Codzienny Scrum	Każdego dnia	15 min	09:00
2	Codzienny Scrum of Scrums	Każdego dnia	15 min	09:30
3	Planowanie sprintu 0	0	8h	
4	Sprint zerowy	0	14 dni	
5	Przegląd sprintu 0	0	4h	
6	Retrospektywa sprintu 0	0	3h	
7	Planowanie sprintu 1	1	8h	
8	Sprint 1	1	30 dni	
9	Przegląd sprintu 1	1	4h	
10	Retrospektywa sprintu 1	1	3h	

2.3. Układ eksperymentu dla oceny procesu wytwarzania

W poprzednim podrozdziale określono zmienne wejściowe i wyjściowe systemu rzeczywistego, a następnie wybrano zmienne istotne ze względu na cel modelowania, którym jest planowanie i ocena procesu wytwarzania oprogramowania. Na tej podstawie przeprowadzono analizę procesów wytwarzania prowadzonych według wielu odmiennych metodyk, co pozwoliło określić niezbędną zawartość i strukturę wewnętrzną wybranych zmiennych. Na podstawie tych prac można w tym podrozdziale przeprowadzić formalny opis układu eksperymentu, na który składają się należące do układu wejściowe i wyjściowe zmienne systemu rzeczywistego. Dalsza część podrozdziału zawiera opis zmiennych wejściowych i wyjściowych będących elementami układu eksperymentu.

2.3.1. Zmienna wejściowa zakres projektu

Zakres projektu jest to zmienna wejściowa systemu rzeczywistego, która została włączona do układu eksperymentu, ponieważ od jej wartości zależy liczba wytwarzanych artefaktów, liczba i czas wykonania elementarnych zadań, i w konsekwencji całkowita pracochłonność procesu

wytwarzania oprogramowania. Duże projekty informatyczne wymagają większych nakładów pracy niż małe. Zakres projektu jest oszacowany w różnych metodykach jako liczba punktów funkcyjnych (FP) [53], liczba roboczogodzin lub liczba punktów przypadków użycia (UCP)[19]. Zakres można ujednolicić jako liczbę punktów AGOMO–UCP. Szacowanie zakresu projektu za pomocą dostosowanej metody Use Case Points oraz przeliczanie z FP lub roboczogodzin na AGOMO–UCP jest przedstawione w Dodatku A (Rozmiar oprogramowania) do rozprawy. Zakres projektu powinien umożliwiać wyrażenie rozmiaru zarówno nowych projektów – tworzonych od podstaw, jak też projektów polegających na rozbudowie istniejącego systemu oraz projektu tworzonego na podstawie systemów poprzedniej generacji. W tym celu wartość zakresu projektu ZP została wyrażona w postaci pary liczb:

$$ZP = (zn, zo) \tag{2.1}$$

gdzie:
zn – zakres nowej części projektu w AGOMO–UCP, wymagania do pozyskania w trakcie warsztatów i wywiadów z udziałowcami, $zn \in N$,
zo – zakres projektu w AGOMO–UCP do odzyskania z poprzedniej wersji systemu, wymagania odzyskiwane z oprogramowania odziedziczonego, $zo \in N$.

Na przykład oszacowanie projektu strony internetową wielkiej północnoamerykańskiej spółki wytwarzającej oprogramowanie wyniosło 180 UCP = 36 AGOMO–UCP, co odpowiada pracochłonności 360 roboczodni [67]. Rozmiar projektu w tym przykładzie wynosi:

$$ZP = (36, 0)$$

2.3.2. Zmienna wejściowa metodyka wytwarzania

Metodyka wytwarzania jest to zmienna wejściowa systemu rzeczywistego, która została włączona do układu eksperymentu, ponieważ zawiera najistotniejsze informacje o składowych procesu wytwarzania. Metodyka wytwarzania, zmienna, której zawartość określa proces wytwarzania oprogramowania, jest przedstawiona jako zestaw informacji o:

- typach zadań – elementarnych, niepodzielnych operacjach w procesie wytwarzania oprogramowania,
- działaniach – zbiorach typów zadań połączonych wspólnym celem,
- etapach – przepływach działań wyznaczających kolejność wykonywania prac.
- rolach projektowych – rolach w projekcie, które mogą pełnić członkowie zespołu projektowego.

Metodyka wytwarzania M jest to sekwencja składająca się ze zbioru elementarnych typów zadań T, pogrupowanych w działania O, zbioru etapów metody E, zawierających przepływ działań w etapie, zbioru kategorii prac, składający się z podzbiorów zbioru działań O oraz zbioru ról projektowych R. Elementy składowe metodyki zostały pokazane na Rys. 2.6

$$M = (D, E, K, O, R, TZ) \tag{2.2}$$

gdzie:
D – zbiór dyscyplin metody,
E – zbiór etapów metody,
K – zbiór kategorii działań metody został opisany w pkt. 2.3.5,
O – zbiór działań metodyki,

R – zbiór ról projektowych,
TZ – zbiór elementarnych typów zadań metody.

Dyscypliny D grupują, działania, role i przepływy według dziedziny działalności, do której należą. Zmienna metodyka może zawierać różne zbiory dyscyplin, przyjętym w pracy standardem jest metodyka RUP, która wyróżnia osiem podstawowych dyscyplin, w tym pięć dyscyplin technicznych:

- Pozyskanie wymagań – celem jest opisanie tego, co system powinien robić. Wymagania zbierane są przez analityków, którzy odkrywają je, klasyfikują i dokumentują. Proces zbierania wymagań polega na dyskusji i uzgodnieniach pomiędzy tworzącymi system a klientem.
- Analiza i projektowanie – tworzy model projektowy i opcjonalnie model analityczny systemu. Model analityczny zapewnia abstrakcję od kodu źródłowego – to znaczy, służy on jako wytyczne do stworzenia tego kodu. Model projektowy składa się z klas zorganizowanych w pakiety i podsystemy z dobrze określonymi interfejsami. Służy to wyodrębnieniu komponentów w fazie implementacji. Zawiera także opis, które obiekty klas współpracują w celu realizacji przypadków użycia
- Implementacja – podział na podsystemy i warstwy, utworzenie komponentów z klas i obiektów, wytworzenie i wykonanie testów jednostkowych, integracja pełnego systemu
- Testowanie – sprawdzenie interakcji między obiektami, weryfikacja integracji komponentów, przetestowanie implementacji wymagań, identyfikacja defektów
- Wdrożenie – produkcja zewnętrznych dystrybucji systemu, pakowanie, dystrybucja, instalowanie, zapewnienie pomocy i wsparcia użytkownikom

i trzy dyscypliny pomocnicze:

- Zarządzanie przedsięwzięciem – zarządzanie ryzykiem, zgrubne planowanie faz projektu, szczegółowe planowanie iteracji, monitorowanie postępów prac
- Zarządzanie konfiguracją i zmianą – zarządzanie wersjonowaniem artefaktów, zarządzanie zależnościami, zarządzanie zleceniami zmian w artefaktach, zarządzanie stanami i miarami artefaktów
- Środowisko – identyfikacja i ocena narzędzi, instalowanie i konfigurowanie narzędzi dla zespołu projektowego, wspieranie narzędzi i procesów w całym projekcie

We wszystkich metodykach wytwarzania cykl życia projektu składa się z następujących po sobie etapów. Każdy etap ma swój cel nazywany kamieniem milowym i sposób osiągnięcia tego celu określony jako przepływ działań. W metodykach tradycyjnych proces składa się z etapów, których celem jest kolejno zebranie wymagań i skonstruowanie architektury, zaprojektowanie systemu, zakodowanie funkcjonalności systemu, przetestowanie i wdrożenie gotowego systemu u klienta. W metodykach zwinnych etapy nie różnią się między sobą. Celem każdego z etapów jest dostarczenie wartości biznesowej klientowi poprzez wydanie następnej wersji systemu informatycznego. W metodyce Scrum występuje zerowy sprint, którego celem jest opracowanie zawartości rejestru produktu i który może być uznany za osobny etap. Poza tym wszystkie zdarzenia metodyki Scrum, takie jak: planowanie sprintu, codzienny scrum, sprint,

przegląd sprintu, retrospektywa, mają swoje odrębne cele do osiągnięcia i sposoby ich realizacji wyrażone przez przepływ działań zapisany w postaci grafu.

Zbiór E określa z jakich etapów składa się metoda M.

$$E = \{e_i | i = 1 \dots le\} \tag{2.3}$$

gdzie:
le – liczba etapów metodyki M,
e_i – i–ty etap metodyki M zawierający przepływ działań.

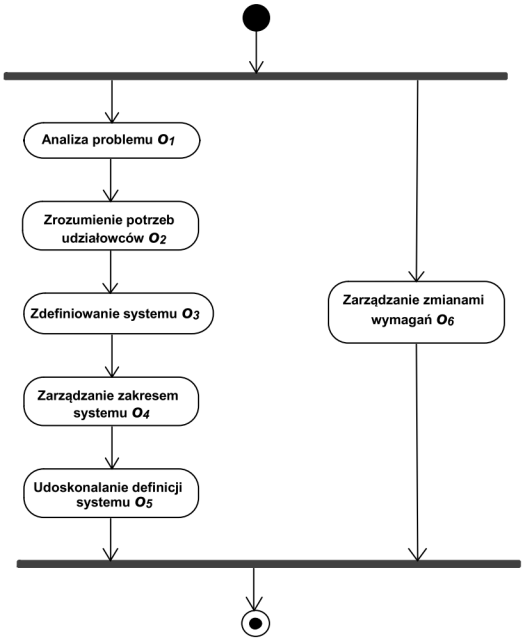
Przepływy określają, w jakiej kolejności działania powinny być wykonane, aby otrzymać rezultat w postaci systemu informatycznego. Przepływy składają się z działań grupowych, które pozwalają porządkować działania elementarne. Działania grupowe składają się z działań sekwencyjnych, równoległych i warunkowych. Przepływy opisane działaniami grupowymi w naturalny sposób mogą być wyrażone w postaci diagramów aktywności UML. W modelu przepływy działań związanych z określoną iteracją są reprezentowane przez graf skierowany, który jest uproszczoną, statyczną postacią diagramu aktywności. Uproszczenie grafu polega na braku działań warunkowych, więc wygenerowany graf powinien być odpowiednio dostosowany do warunków.

Przepływ etapu e_i jest scharakteryzowany przez acykliczny graf skierowany (DAG) określający kolejność wykonywanych działań.

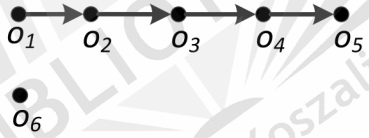
$$e_i = (VE_i, EE_i) \tag{2.4}$$

gdzie:
VE_i – zbiór wierzchołków grafu e_i, $VE_i \subset O$ metodyki M,
EE_i – zbiór krawędzi grafu e_i, $EE_i \subset VE_i \times VE_i$.

Na przykład w metodyce RUP działania dyscypliny wymagań są reprezentowane przez diagram aktywności UML pokazany na Rys. 2.4. Na diagramie oznaczono działania $o_1 - o_6$, które zostały przedstawione jako graf DAG zamieszczony na Rys. 2.5.



Rys. 2.4 Diagram aktywności dyscypliny wymagań metodyki RUP
Źródło: Opracowanie własne



Rys. 2.5 Przepływ działań dyscypliny wymagań w postaci grafu skierowanego
Źródło: Opracowanie własne

Zbiór O określa, z jakich działań składa się metoda M. Działania $o_{i,k}$ są podzbiorami zbioru elementarnych typów zadań T metody M.

$$O = \{o_{i,k} | i = 1..lo, k = 1..lot_i\} \tag{2.5}$$

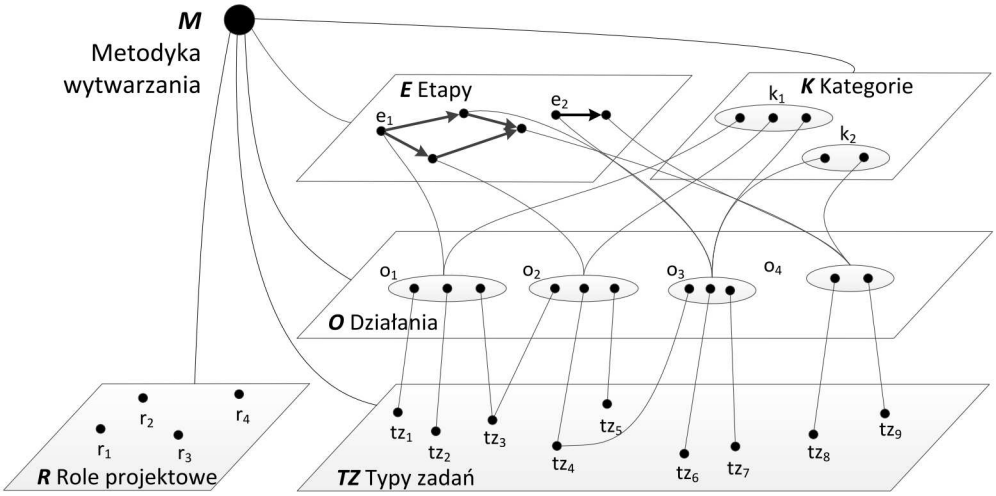
gdzie:
$o_{i,k}$ – działania $o_{i,k} \in TZ$ metodyki M,
lo – liczba działań metodyki M,
lot_i – liczba typów zadań i–tego działania

Na przykład działanie o_1– Analiza problemu z Rys. 2.4 składa się w metodyce RUP z następujących typów zadań:

- sporządź wspólny słownik (ang. *Capture a Common Vocabulary*),
- wyszukaj aktorów i przypadki użycia (ang. *Find Actors and Use Cases*),
- utwórz dokument wizji (ang. *Develop Vision*),

- utwórz plan zarządzania wymaganiami (ang. *Develop Requirements Management Plan*).

Są to elementarne typy zadań należące do zbioru TZ metodyki RUP.



Rys. 2.6 Elementy składowe zmiennej metodyka wytwarzania

Źródło: Opracowanie własne

Elementy składowe metodyki wytwarzania zostały przedstawione w postaci przykładu na Rys. 2.6. Jak widać na rysunku, metodyka składa się ze zbioru ról projektowych $R = \{r_1, r_2, r_3, r_4\}$, zbioru etapów $E = \{e_1, e_2\}$, gdzie etapy te są grafami skierowanymi, wierzchołkami których są elementy zbioru działań O: $e_1 = (\{o_1, o_2, o_3, o_4\}, \{(o_1, o_2), (o_1, o_4), (o_2, o_3), (o_4, o_3)\})$, $e_2 = (\{o_3, o_4\}, \{(o_3, o_4)\})$, zbioru kategorii $K = \{k_1, k_2\}$, której elementy są podzbiorami elementów zbioru działań O: $k_1 = \{o_1, o_2, o_3\}$, $k_2 = \{o_3, o_4\}$, zbioru działań $O = \{o_1, o_2, o_3, o_4\}$, którego elementy są podzbiorami zbioru typów elementarnych TZ, $o_1 = \{tz_1, tz_2, tz_3\}$, $o_2 = \{tz_3, tz_4, tz_5\}$, $o_3 = \{tz_4, tz_6, tz_7\}$, $o_4 = \{tz_8, tz_9\}$, zbioru typów elementarnych $TZ = \{tz_1, tz_2, tz_3, tz_4, tz_5, tz_6, tz_7, tz_8, tz_9\}$, którego elementy są atomowymi, niepodzielnymi zadaniami, należącymi do metodyki wytwarzania oprogramowania.

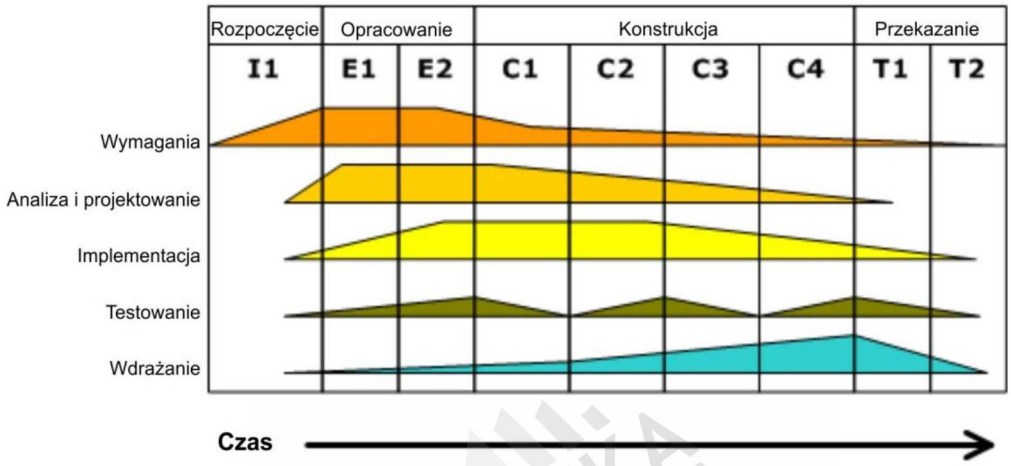
2.3.3. Zmienna wejściowa plan projektu

Plan projektu jest to zmienna wejściowa systemu rzeczywistego, która została włączona do układu eksperymentu, ponieważ określa w jakich etapach i iteracjach będzie realizowany proces wytwarzania oprogramowania. W wykonanej w podrozdziale 2.2 analizie zawartość zmiennej plan projektu różni się w poszczególnych metodykach. Częścią wspólną dla wszystkich metodyk jest określenie etapu i zakresu projektu zaplanowanego do wykonania w iteracji. Prace wykonywane w każdym z etapów metodyk RUP i spiralnej są wykonywane w jednej lub wielu iteracjach. Przykład iteracyjnych prac został pokazany na Rys. 2.7. W metodyce RUP przepływ prac każdej iteracji zawiera działania wszystkich dyscyplin, jednak między kolejnymi iteracjami i etapami zmienia się waga i nacisk na uzyskanie wyników różnych działań.

- Na etapie rozpoczęcia najważniejsze jest zrozumienie całości wymagań określenie zakresu tworzonego systemu.
- Na etapie opracowania uwaga jest zwrócona na wymagania, ale wykonywana jest pewna część prac związanych z projektowaniem i implementacją z uwagi na potrzebę

wytworzenia wykonywalnego prototypu architektury, który służy jako szkielet do zbudowania pełnego systemu informatycznego w kolejnych etapach.

- Na etapie budowy najistotniejsze dziedziny to projektowanie i implementacja oraz testowanie. W tym etapie dochodzi do utworzenia pierwszej działającej wersji systemu.
- Na etapie przekazania największy nacisk położony jest na czynności dziedziny wdrażania, testowanie, usuwanie usterek i implementację brakujących funkcjonalności systemu.



Rys. 2.7 Idea zróżnicowanej wagi czynności poszczególnych dyscyplin w kolejnych iteracjach procesu wytwarzania.

Źródło: Opracowanie własne na podstawie [58]

Plan projektu jest sekwencją iteracji składających się z etapu i zaplanowanego zakresu prac.

$$PP = (I_{1,} \dots, I_{li}) \tag{2.6}$$

gdzie:
I_i – *i*–ta iteracja planu *PP,*
li – liczba iteracji planu projektu *PP.*

Iteracja składa się z etapu i procentowego zakresu prac do wykonania.

$$I_i = (e_i, z_i) \tag{2.7}$$

gdzie:
e_i – etap *i*–tej iteracji planu *PP,* $e_i \in E$ metodyki wytwarzania M,
z_i – procentowa wartość zakresu projektu ZP do realizacji w *i*–tej iteracji planu *PP.*

W metodyce RUP procentowy zakres prac z_i^{RUP} zaplanowanych do realizacji w *i*–tej iteracji jest podzielony na 8 dyscyplin RUP.

$$z_i^{RUP} = \{zd_{i,n} | n = 1..8\} \tag{2.8}$$

gdzie:
$zd_{i,n}$ – wartość procentowa 0–100% dla dyscypliny *n*, odnosząca się do zakresu projektu ZP do realizacji w *i*–tej iteracji planu *PP.*

W metodyce Scrum procentowy zakres prac nie występuje. Zakres prac jest określony przez czas trwania sprintu lub innego zdarzenia. W przyjętym zapisie zdarzenia metodyki są przedstawione w postaci iteracji o określonym czasie trwania. Typy zdarzeń są reprezentowane przy pomocy etapu, czyli przepływu działań zdarzenia. W Scrumie oprócz zaplanowanych zdarzeń o określonym czasie trwania, występujących w jednej sekwencji, występują też zdarzenia codzienne. Dla nich oprócz czasu trwania jest określona godzina rozpoczęcia. Zatem plan projektu dla metodyki Scrum PP^S jest przedstawiony w postaci sekwencji:

$$PP^S = (I^S, ZC) \tag{2.9}$$

gdzie:

PP^S – plan projektu dla metodyki Scrum,
I^S – zdarzenia planu projektu zapisane w postaci iteracji,
ZC – zdarzenia codzienne planu projektu.

Przykładem zdarzeń reprezentowanych za pomocą iteracji jest spotkanie planowania sprintu, sprint i przegląd sprintu. Zdarzenia codzienne to 15–sto minutowe spotkania scrum oraz przy pracy w wielu zespołach, spotkania scrum of scrums.

Iteracje planu projektu metodyki Scrum I^S jest to lista iteracji:

$$I^S = (I_1^S, \dots, I_{li}^S) \tag{2.10}$$

gdzie:
I_i^S – *i*–te zdarzenie planu projektu metodyki Scrum zapisane w postaci iteracji

Pojedyncza iteracja metodyki Scrum jest sekwencją, która składa się z informacji o przepływie działań iteracji i czasie trwania iteracji, liczonym w dniach i godzinach:

$$I_i^S = (e_i, t_i) \tag{2.11}$$

gdzie:
e_i – etap *i*–tej iteracji planu PP^S, $e_i \in E$ metodyki wytwarzania M,
t_i – czas trwania *i*–tej iteracji planu PP^S liczony w dniach i godzinach.

Zbiór zdarzeń codziennych planu projektu PP^S może zawierać jedno lub dwa zdarzenia. Zdarzenia codzienne mają wyższy priorytet niż działania wynikające z iteracji. Działania zwykłych zdarzeń Scrum są zawieszane na czas wykonywania działań zdarzeń codziennych.

$$ZC = \{zc_i | i = 1 \dots lzc\} \tag{2.12}$$

gdzie:
ZC – zbiór zdarzeń codziennych planu projektu PP^S,
zc_i – *i*–te zdarzenie codzienne,
lzc – liczba zdarzeń codziennych planu projektu PP^S.

Zdarzenie codzienne metodyki Scrum jest sekwencją, która składa się z informacji o przepływie działań zdarzenia, czasie trwania zdarzenia, liczonym w godzinach i minutach oraz o godzinie rozpoczęcia zdarzenia.

$$zc_i = (e_i, t_i, r_i) \tag{2.13}$$

gdzie:
zc_i – zdarzenie codzienne metodyki Scrum,

e_i– przepływ działań *i*–tego zdarzenia codziennego $e_i \in E$,
zct_i– czas trwania *i*–tego zdarzenia codziennego,
zcr_i– czas rozpoczęcia *i*–tego zdarzenia codziennego.

2.3.4. Zmienna wejściowa zespół projektowy

Informacje o zespole projektowym jest to zmienna wejściowa systemu rzeczywistego, która została włączona do układu eksperymentu, ponieważ liczba i kompetencje członków zespołu projektowego mają niezwykle istotne wpływ na przebieg procesu wytwarzania, termin dostarczenia produktu i powodzenie projektu. Wiele modeli przyjmuje założenie, że dwie osoby zrealizują projekt w czasie dwukrotnie krótszym, niż jedna osoba. Jest to zbyt duże uproszczenie nie sprawdzające się w rzeczywistych warunkach. W celu uniknięcia go, określono kompetencje członków zespołu używając ról projektowych. Zespół projektowy składa się z osób, z których każda może pełnić wiele ról projektowych i jedna rola projektowa może być pełniona przez wiele osób. Zestaw ról projektowych jest określony przez zmienną wejściową M – metodyka. Zmienna Z – zespół projektowy zawiera osoby wchodzące w skład zespołu.

$$Z = \{P_i | i = 1 \dots lp\} \tag{2.14}$$

gdzie:
Z – zespół projektowy,
P_i – *i*–ty członek zespołu projektowego Z
lp – liczba osób w zespole projektowym.

Każda osoba należąca do zespołu projektowego może pełnić wiele ról projektowych.

$$P_i = \{r_{i,k} | k = 1 \dots lr_i\} \tag{2.15}$$

gdzie:
$r_{i,k}$ – k–ta rola projektowa i–tego członka zespołu projektowego, $r_{i,k} \in R$ zbioru ról projektowych metodyki wytwarzania M,
lr_i – liczba ról *i*–tej osoby zespołu projektowego Z.

2.3.5. Zmienne wyjściowe opisujące wykonane prace

Informacja o wykonanych pracach jest to zmienna wyjściowa systemu rzeczywistego, która została włączona do układu eksperymentu, ponieważ w celu oceny procesu wytwarzania oprogramowania porównujemy badany przebieg prac z wzorcowym, wynikającym z przyjętej metodyki i charakterystyki projektu i zespołu projektowego. Badany przebieg procesu jest obserwowany na wyjściu systemu rzeczywistego w postaci zmiennej W zawierającej informację o wykonanych pracach i zmiennej X zawierającej harmonogram prac. Zmienna W składa się z zadań, które zostały wykonane w trakcie wytwarzania oprogramowania.

$$W = \{w_i | i = 1 \dots lw\} \tag{2.16}$$

gdzie:
W – wykonane prace,
w_i – *i*–te zadanie wykonanych prac W,
lw – liczba wykonanych prac.

Wykonane prace są zapisane w systemie zarządzania zmianą (SZZ). SZZ przechowuje informacje o tym, kto zlecił prace, kto ją wykonał i kto zweryfikował. Informacja o tym, jakie za-

danie zostało wykonane jest zapisana jako opis w języku naturalnym. Brakuje precyzyjnej informacji o typie elementarnego zadania, które zostało wykonane. Informację tą można wydobyć z ograniczoną dokładnością, analizując zawartość opisu w języku naturalnym oraz role osób, które zleciły, wykonały i zweryfikowały zadanie. Wynikiem analizy jest kategoria wykonanego zadania. Zbiór kategorii działań K jest dostarczony przez zmienną M– metodykę wytwarzania (2.2). Poziomy organizacji typów zadań w kategorie zostały pokazane na Rys. 2.8.

$$K = \{k_i | i = 1 \dots lk\} \tag{2.17}$$

gdzie:
K – zbiór kategorii działań
k_i – i–ta kategoria zbioru kategorii K,
lk – liczba kategorii metodyki M.

Kategoria k_i zgrubnie określa, do jakiego typu działania może należeć typ wykonanego zadania elementarnego.

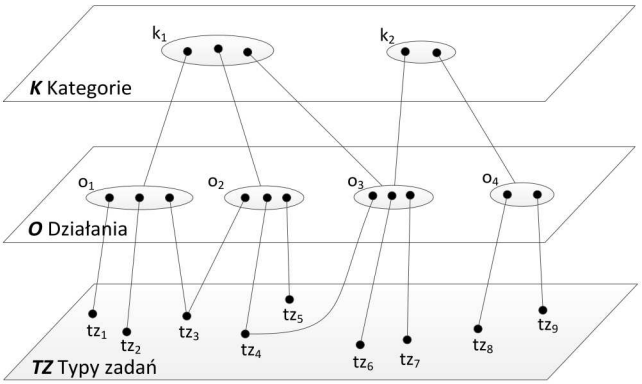
$$k_i = \{o_{i,j} | j = 1 \dots lko_i\} \tag{2.18}$$

gdzie:
$o_{i,j} \subset$ O,
lko_i – liczba działań i–tej kategorii.

Przykładem kategorii w metodyce RUP jest zbieranie wymagań. W skład kategorii zbierania wymagań wchodzą wszystkie działania z przykładu pokazanego na Rys. 2.4, czyli:

- analiza problemu,
- zrozumienie potrzeb udziałowca,
- zdefiniowanie systemu,
- zarządzanie zakresem systemu,
- udoskonalanie definicji systemu,
- zarządzanie zmianami wymagań.

Na Rys. 2.8 pokazano przykładowy zbiór kategorii $K = \{k_1, k_2\}$, w celu przedstawienia struktury omawianych danych. Kategorie k_1 i k_2 są podzbiorami zbioru działań O, który składa się z podzbiorów TZ, zbioru elementarnych typów zadań metodyki wytwarzania.



Rys. 2.8 Przykład zbioru kategorii zadań wykonanych prac
Źródło: Opracowanie własne

Po przypisaniu wykonanym zadaniom odpowiednich kategorii, zadanie w_i można zapisać jako sekwencję składającą się z osoby, która wykonała zadanie, iteracji, w której wykonano zadanie i kategorii wykonanego zadania.

$$w_i = (P_i, I_i, k_i) \tag{2.19}$$

gdzie:
P_i – osoba wykonująca *i*–te zadanie, $P_i \in T$,
I_i – iteracja, w której zostało wykonane *i*–te zadanie, $I_i \in PP$,
k_i – kategoria *i*–tego zadania wykonanych prac, $k_i \in K$.

Z wykonanymi pracami jest związany ich harmonogram:

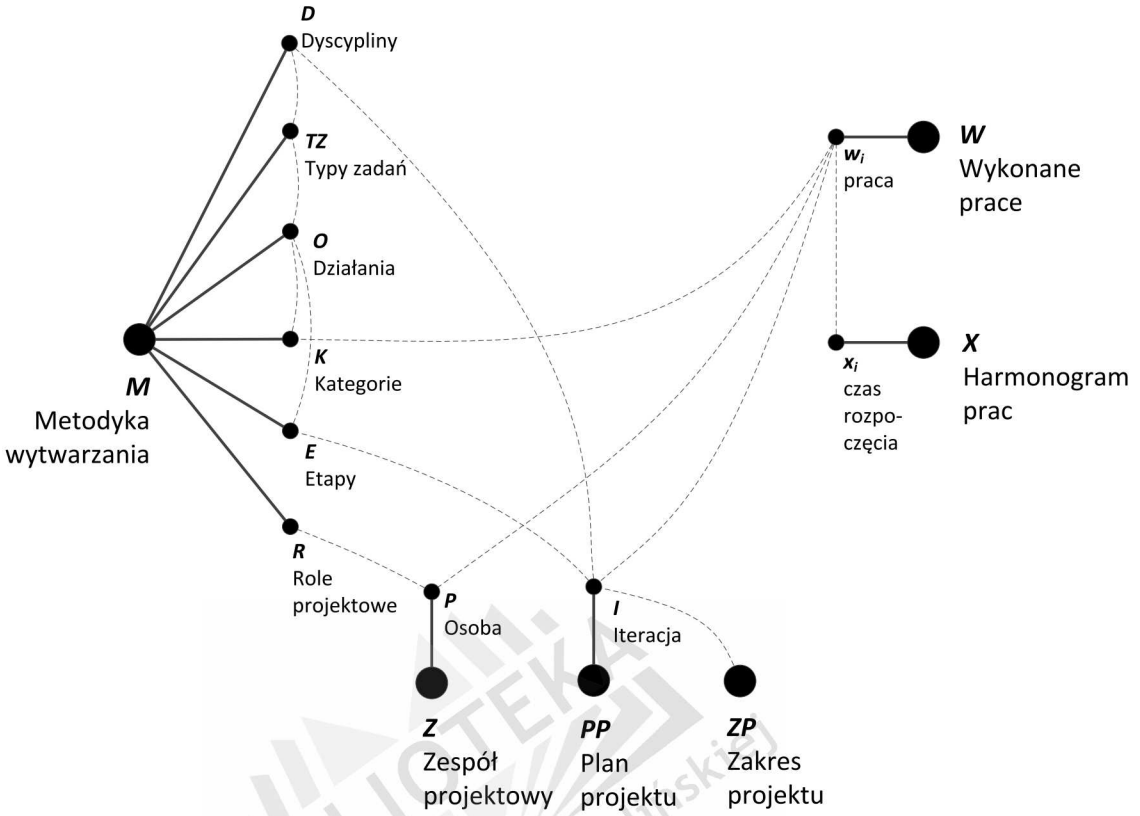
$$X = \{x_i \,|i = 1 \ldots lw\} \tag{2.20}$$

gdzie:
X – harmonogram prac,
x_i – czas rozpoczęcia *i*–tego zadania w_i,
lw – liczba wykonanych prac.

2.4. Podsumowanie układu eksperymentu

W rozdziale 2 opisano system rzeczywisty – proces wytwarzania oprogramowania oraz jego zmienne wejściowe i wyjściowe. Następnie z istniejących zmiennych systemu rzeczywistego, kierując się potrzebą oceny i planowania procesu wytwarzania oprogramowania, utworzono odpowiedni układ eksperymentu. Dla układu eksperymentu przeanalizowano etap rozpoczęcia i dalsze etapy procesu wytwarzania oprogramowania dla głównych, zarówno tradycyjnych, jak zwinnych metodyk wytwarzania. Wynikiem analizy jest określenie czterech zmiennych wejściowych i dwóch zmiennych wyjściowych, które umożliwiają badanie systemu rzeczywistego pod kątem oceny procesu wytwarzania. Zmienne i powiązania między nimi zostały pokazane na Rys. 2.9. Zmienne wejściowe to: metodyka wytwarzania, zespół projektowy, plan projektu i zakres projektu. Zmienne wyjściowe to wykonane prace i ich harmonogram. Na rysunku zmienne zostały oznaczone dużymi punktami. Małe punkty, połączone z nimi liniami ciągłymi to ich elementy składowe. Linie przerywane oznaczają powiązania między zmiennymi. Np. zmienna PP plan projektu składa się z iteracji I, które są powiązane z etapami E i mają

wyznaczony procent zakresu projektu ZP do wytworzenia. Iteracje są również powiązane z wykonanymi pracami w_i.



Rys. 2.9 Powiązania między zmiennymi opisującymi system rzeczywisty w przyjętym układzie eksperymentu

Źródło: Opracowanie własne

Opis systemu rzeczywistego przy pomocy zmiennych pozwala na zadanie pytań dotyczących procesu wytwarzania projektu. Głównym pytaniem jest:

- Czy projekt o danym zakresie, zrealizowany przez zespół projektowy zgodnie z metodyką i założonym planem, osiągnął wskaźnik zgodności na wymaganym poziomie?

Jednak opis układu eksperymentu za pomocą zmiennych pozwala zadać inne, równie interesujące pytania:

- Jaka metodyka pozwoli osiągnąć zadany wskaźnik zgodności procesu wytwarzania?
- Jaki skład zespołu projektowego pozwoli osiągnąć zadany wskaźnik zgodności procesu wytwarzania?
- Czy zmiana zakresu projektu pozwoli osiągnąć wyższy wskaźnik zgodności procesu wytwarzania?
- Jakie zmiany zakresu projektu pozwolą utrzymać zadany wskaźnik zgodności projektu?
- Jaki plan projektu pozwoli na zwiększenie wskaźnika zgodności projektu?

Ujęcie systemu rzeczywistego przy pomocy układu eksperymentu, będącym formą czarnej skrzynki o określonych zmiennych wejściowych oraz niezbędnych do obserwacji zmiennych wyjściowych, daje możliwość zbudowania modelu systemu. Taki model, przy założonych war-

tościach zmiennych wejściowych, powinien generować odpowiednie wartości zmiennych wyjściowych. Harmonogram prac wygenerowany przez model powinien ściśle odpowiadać wybranej metodyce i realizować przyjęty plan prac. Na podstawie porównania harmonogramu wygenerowanego przez model z rzeczywistym harmonogramem zostanie wyznaczona ocena badanego procesu wytwarzania. Model podstawowy spełniający te założenia zostanie zbudowany w następnym rozdziale.

3. Model podstawowy i uproszczony procesu wytwarzania oprogramowania

3.1. Wprowadzenie

W poprzednim rozdziale zdefiniowano układ eksperymentu, który tworzy ramy do zaprojektowania modelu podstawowego. Na podstawie analizy środowiska projektowego wybrano zmienne wejściowe modelu. W wyniku analizy różnych metodyk wytwarzania określono ich strukturę. Cel modelu, czyli planowanie i ocena procesu wytwarzania, wyznaczył strukturę zmiennych wyjściowych. Dzięki tym pracom został utworzony model w postaci czarnej skrzynki. W tym rozdziale opis dotyczący układu eksperymentu zostanie rozszerzony tak, aby objął elementy wewnętrzne modelu i ich interakcje, pozwalające odtworzyć przebieg procesu wytwarzania. W ten sposób powstanie rdzeń modelu podstawowego procesu wytwarzania oprogramowania. Model podstawowy jest to model, który pozwala uzyskać wszystkie reakcje wejścia–wyjścia systemu rzeczywistego, czyli jest zasadny dla wszystkich dopuszczalnych układów eksperymentu [110]. Tą elastyczność, charakterystyczną dla modelu podstawowego, uzyskano dzięki podziałowi na wspólny rdzeń oraz rozszerzenia. Rozszerzenia są zbiorem podmodeli, elementów i interakcji, pozwalają wykroczyć poza ramy wyznaczone przez przyjęty układ eksperymentu.

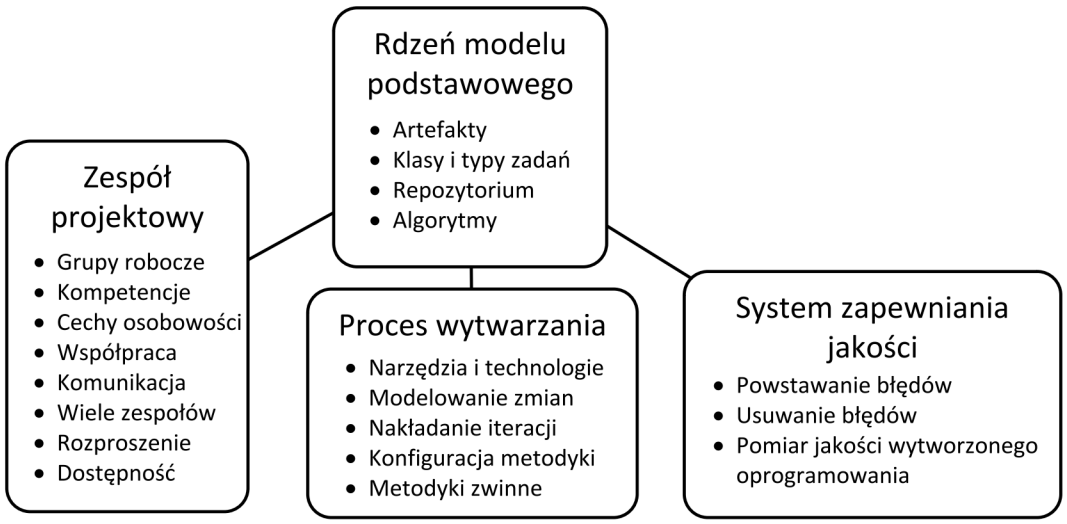
Wyniki badań i prac nad rdzeniem i rozszerzeniami modelu podstawowego zostaną przedstawione w podrozdziale 3.2. Następnie w podrozdziale 3.3 zostanie przeprowadzona analiza dziedzinowa, w której z modelu podstawowego zostaną wyodrębnione elementy niezbędne do realizacji celu wyznaczającego układ eksperymentu. W ten sposób powstanie koncepcja modelu uproszczonego. Na podstawie tej koncepcji, w podrozdziale 3.4 po przeprowadzeniu przeglądu metod modelowania zostanie wybrana metoda modelowania i opisana możliwość realizacji tej metody w stosunku do modelu uproszczonego. Następnie w podrozdziale 3.5 zostanie przeanalizowana możliwość weryfikacji zasadności tak zaprojektowanego modelu uproszczonego. Podsumowanie prac nad modelem podstawowym i uproszczonym umieszczono w podrozdziale 3.6.

3.2. Koncepcja budowy modelu procesu wytwarzania oprogramowania

W celu umożliwienia zastosowania w różnych układach eksperymentu, model podstawowy został podzielony na uniwersalny rdzeń i opcjonalne, konfigurowalne rozszerzenia, co przedstawiono na Rys. 3.1. Konstrukcja rdzenia modelu została oparta na pomyśle podziału procesu wytwarzania na elementarne operacje, aby precyzyjnie oddać zadania wykonywane w procesie wytwarzania. Pracom nad konstrukcją rdzenia modelu towarzyszyła intuicja, że na zasadzie emergencji, z wielu elementarnych zadań w mikroskali, wykonywanych równolegle przez zespół projektowy, wyłoni się obraz przebiegu procesu wytwarzania na poziomie makroskali [60, 70].

Dzięki możliwości dołączania rozszerzeń, uwzględniono w modelu kluczowe czynniki opisujące zespół projektowy takie, jak cechy osobowe, kompetencje członków zespołu oraz ich wpływ na komunikację i współpracę. Kolejne grupy rozszerzeń dotyczą procesu wytwarzania oprogramowania i systemu zapewniania jakości. Zarówno dążenie do precyzji opisu, jak próba

uwzględnienia czynnika ludzkiego oraz szerokie spojrzenie na proces wytwarzania wynika z ponad dwudziestopięcioletniej pracy autora, głównie w charakterze dewelopera, w szerokim zakresie projektów informatycznych.



Rys. 3.1 Koncepcja modelu podstawowego zbudowanego z uniwersalnego rdzenia i opcjonalnych rozszerzeń modelu

Źródło: Opracowanie własne

W kolejnych podrozdziałach, zgodnie ze strukturą pokazaną na Rys. 3.1, zostanie przedstawiona budowa wewnętrzna i algorytmy sterujące rdzenia modelu, a następnie zostaną omówione kolejne grupy rozszerzeń dotyczące zespołu projektowego, procesu wytwarzania i systemu zapewniania jakości.

3.2.1. Budowa rdzenia modelu podstawowego

Rdzeń modelu został oparty głównie na elementach składowych wyróżnionych w metodyce RUP. Metodyka RUP charakteryzuje się mocną strukturą i podejściem procesowym, w którym proces wytwarzania został podzielony na elementarne zadania, które przetwarzają jednostkowe porcje informacji nazywane artefaktami. Elementarne zadania są następnie składane w działania, przepływy, iteracje i etapy. Zadania i artefakty są bardzo podobne we wszystkich metodykach wytwarzania. Na przykład zadania związane z kodowaniem występują zarówno w tradycyjnych i zwinnych metodykach wytwarzania. W tradycyjnych metodykach wytwarzania zadanie kodowania jest wykonywane na podstawie dokumentów wymagań i projektowych, przechowywanych w repozytorium. W metodykach zwinnych projekt jest najczęściej wykonywany przez dewelopera na podstawie ogólnych wytycznych dotyczących architektury systemu i szczegółowych informacji pozyskanych od właściciela produktu.

W rozdziale 2 rozprawy opisano zmienne wejściowe i wyjściowe modelu, co daje możliwość przedstawienia modelu jako czarnej skrzynki. W celu opisania wnętrza tego modelu należy rozszerzyć wcześniejszy opis o informacje o elementach składowych. W następnych podrozdziałach zostaną opisane następujące elementy rdzenia modelu: typy i instancje artefaktów, repozytorium przechowujące instancje artefaktów oraz typy i klasy zadań przetwarzających artefakty.

Artefakty

Artefakty są elementami modelu, które reprezentują porcje informacji używane, zmieniane lub wytwarzane przez proces. Artefakty to namacalne elementy projektu: byty, które są wytwarzane w projekcie lub używane podczas pracy nad końcowym produktem. Artefakty są informacją wejściową niezbędną do wykonania zadań oraz są również informacją wyjściową powstałą podczas wykonywania innych zadań. Artefakty mogą przyjmować różne formy, takie jak:

- model, jak model przypadków użycia lub model projektowy
- element modelu – element składowy modelu, jak pojedynczy przypadek użycia, klasa lub podsystem
- dokument, jak dokument wizji, analiza ryzyka lub analiza biznesowa
- kod źródłowy,
- uruchamialny program, jak np. działający prototyp architektury

Wśród artefaktów wyróżniono dwie klasy: artefakty elementarne i artefakty ogólne. Elementarne opisują pojedyncze, szczegółowe wymagania, przypadki użycia, aktorów, klasy, przypadki testowe. Ogólne opisują system informatyczny jako całość, na wysokim poziomie abstrakcji. Są to modele składające się z artefaktów elementarnych albo dokumenty opisujące pewne aspekty systemu, jak dokument architektury systemu, czy dokument wizji systemu.

Po uwzględnieniu dodatkowych elementów metodyki i ich powiązań, definicja metodyki wytwarzania powinna zostać rozszerzona o zbiór typów artefaktów A:

$$M = (D, E, K, O, R, TZ, A) \tag{3.1}$$

gdzie:
D – zbiór dyscyplin metody,
E – zbiór etapów metody,
K – zbiór kategorii działań metody został opisany w pkt. 2.3.5,
O – zbiór działań metody,
R – zbiór ról projektowych,
TZ – zbiór elementarnych typów zadań metody,
A – zbiór typów artefaktów.

Instancje artefaktów wchodzące w skład systemu mogą być ze sobą połączone relacjami. Wyróżniono dwie relacje łączące artefakty:

- strukturalna – artefakty elementarne mogą być częścią artefaktu ogólnego. Np. przypadek użycia jest częścią modelu przypadków użycia.
- pochodzenia – artefakty są związane relacją pochodzenia, gdy na podstawie artefaktu – poprzednika relacji jest utworzony artefakt wynikowy – następnik relacji. Przykładem może być utworzenie instancji artefaktu klasa analityczna na podstawie instancji artefaktu przypadek użycia.

Obydwie relacje uproszczono do jednej – relacji pochodzenia. W tym uproszczeniu artefakt ogólny jest traktowany jako artefakt wynikowy. Zatem instancja artefaktu jest określona jako sekwencja:

$$oa_i = (a_i, s_i, p_i) \tag{3.2}$$

gdzie:
oa_i – instancja artefaktu
a_i – typ artefaktu,
s_i – zbiór instancji artefaktów źródłowych, na podstawie których utworzono instancję artefaktu ao_i,
p_i – zbiór instancji artefaktów utworzonych na podstawie instancji artefaktu ao_i.

Repozytorium artefaktów

Repozytorium artefaktów jest elementem modelu, który jest odpowiedzialny za przechowywanie instancji artefaktów pogrupowanych według typów, nadawanie artefaktom unikalnych identyfikatorów, tworzenie instancji artefaktów o podanym typie oraz wyszukiwanie instancji artefaktów wg podanego typu.

W modelu możliwość realizacji zadań elementarnych t_i jest uwarunkowana dostępem do wymaganych artefaktów typu ai_i. Typy i liczba dostępnych artefaktów w modelu jest określana w postaci repozytorium, definiowanym jako sekwencja:

$$RP = (A, \varphi, \omega, \gamma, \tau) \tag{3.3}$$

gdzie:
A – zbiór typów artefaktów środowiska projektowego,
$\varphi(a,t) = \varphi_{a,t}$ – funkcja określająca liczbę instancji artefaktów typu $a \in A$ w czasie t w umownych, dyskretnych jednostkach czasu symulacji u.j.t.,
$\omega(a,tz,t) = \omega_{a,tz,t}$ – funkcja wyszukująca instancji artefaktów typu $a \in A$ nieprzetworzonych przez zadania typu $tz \in T$ *w* dyskretnej chwili czasu t,

$\gamma(a) = \gamma_a$ – funkcja wyszukująca w repozytorium instancji artefaktu typu $a \in A$ i tworząca instancję, jeśli nie została znaleziona.

$\tau(a) = \tau_a$ – funkcja tworząca w repozytorium instancję artefaktu typu $a \in A$.

Zadania produkujące instancje artefaktów umieszczają utworzone instancje w repozytorium, gdzie są dostępne dla innych zadań, a po wyszukaniu stają się ich artefaktami wejściowymi.

Typy zadań

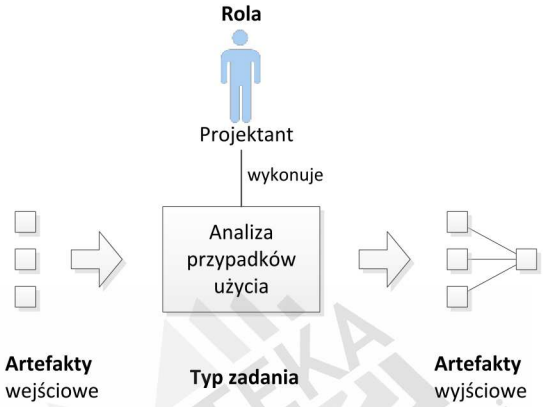
Typy zadań są składowymi modelu reprezentującymi elementarne typy zadań procesu wytwarzania oprogramowania, takie jak: utworzenie przypadku użycia, zaprojektowanie klasy, zaimplementowanie klasy. Zamieszczona w poprzednim rozdziale definicja metodyki wytwarzania zawiera zbiór typów zadań elementarnych TZ (2.2), lecz nie precyzuje czym jest typ zadania. Zadanie jest to jednostka pracy jaką wykonuje osoba pełniąca wymaganą rolę. Zadanie ma jasno określony cel, który jest wyrażony jako wytworzenie lub przetworzenie istniejących artefaktów na nowe artefakty. Istnieje wiele elementarnych typów zadań różniących się od siebie typami przetwarzanych artefaktów i rolą projektową wymaganą do wykonania zadania. Typ elementarnego zadania $tz_i \in T$ można zatem zdefiniować jako sekwencję:

$$tz_i = (r_i, aw_i, ap_i, tzd_i) \tag{3.4}$$

gdzie:
tz_i – i–ty typ zadania,
r_i – rola projektowa wymagana do wykonania *i*–tego typu zadania,

aw_i – zbiór typów artefaktów wejściowych *i*–tego typu zadania $aw_i \subseteq A$,
ap_i – zbiór typów artefaktów wyjściowych, produkowanych przez *i*–ty typ zadania $ap_i \subseteq A$,
tzd_i – czas wykonania *i*–tego typu zadania.

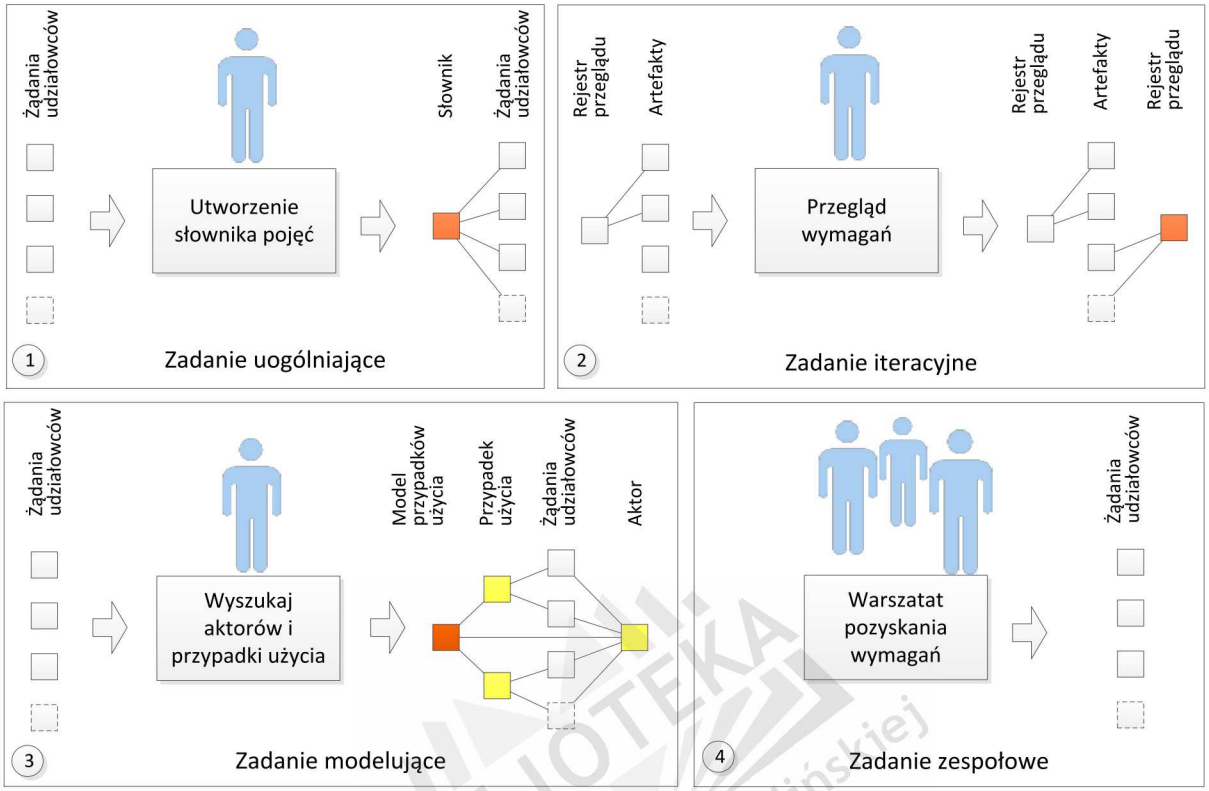
Zadania mogą być wykonywane przez pojedynczą osobę lub przez osobę współpracującą z innymi osobami. Przykładem zadania wykonywanego przy współpracy wielu osób jest zadanie przeprowadzenia warsztatów w celu pozyskania wymagań od klienta. Zadanie zorganizowania warsztatów może być wykonane przez osobę pełniącą rolę analityka systemowego. Analityk systemowy współpracuje przy wykonywaniu zadania z dostępnymi udziałowcami ze strony klienta, projektantami, architektami i specyfikatorami wymagań.



Rys. 3.2 Typ zadania, rola projektowa i artefakty
Źródło: Opracowanie własne

Czas wykonywania zadania jest wyrażony w umownych jednostkach czasu (u.j.t.), a jego wartość zależy od typów i liczby przetwarzanych artefaktów. Jeżeli zadanie przetwarza elementarne artefakty wejściowe na artefakty wyjściowe, to czas wykonania zadania jest liniowo zależny od liczby przetwarzanych artefaktów. Jeżeli zadanie przetwarza artefakty ogólne, opisujące z pewną dokładnością cały system informatyczny, to czas wykonania zadania jest liniowo zależny od wielkości projektu. Liniowość zależności czasu przetwarzania zadania od zakresu projektu została wprowadzona w celu uzyskania skalowalności modelu. Skalowalność modelu zapewnia, że pracochłonność projektu o dwukrotnie większym zakresie będzie dwukrotnie większa.

W modelu procesu wytwarzania występuje 77 typów zadań, które należą do czterech głównych klas zadań, pokazanych wraz z przykładami na Rys. 3.3.



Rys. 3.3 Cztery klasy zadań z przykładami
Źródło: Opracowanie własne

Głowne klasy typów zadań zostały wydzielone ze względu na sposób przetwarzania artefaktów wejściowych na artefakty wyjściowe, są to:

1. Zadania uogólniające jest to klasa zadań przetwarzających wiele artefaktów elementarnych w jeden artefakt ogólny. Przykładem jest zadanie polegające na utworzeniu słownika systemu. Artefakty żądania udziałowców są przeglądane w celu wyszukania pojęć i terminów do umieszczenia w wynikowym artefakcie – słowniku systemu.
2. Zadania iteracyjne jest to klasa zadań przetwarzających artefakty elementarne w jeden artefakt ogólny dla każdej iteracji. Przykładem zadania iteracynego jest zadanie przeglądu wymagań. Odpowiada ono sytuacji, kiedy w każdej iteracji organizowane są przeglądy nowoutworzonych wymagań. Wynikiem wykonania zadania jest utworzenie artefaktu – raportu przeglądu, osobno dla każdej iteracji.
3. Zadania modelujące jest to klasa zadań przetwarzających artefakty elementarne na elementy modelu i łączące je w jeden model. Przykładem zadania modelującego jest zadanie wyszukania aktorów i przypadków użycia. Wykonanie zadania polega na przetworzeniu żądań udziałowców w celu identyfikacji przypadków użycia i aktorów. W trakcie przetwarzania na podstawie artefaktów żądań udziałowców tworzone są, w odpowiednich proporcjach, artefakty przypadków użycia i aktorów, połączone we wspólny artefakt – model przypadków użycia.

4. Zadania zespołowe jest to klasa zadań wymagających współpracy miedzy osobami pełniącymi różne role. Przykładem zadania zespołowego jest warsztat przypadków użycia, do którego wykonania zaaangażowana jest grupa członków zespołu projektowego o określonych rolach, pod kierownictwem analityka systemowego. Grupa ta w trakcie warsztatów wytwarza pewną liczbę artefaktów typu żądanie udziałowcy (ang. *Stakeholder Request*). Wymagana liczba osób, pełnione przez nich role, czas trwania warsztatów i liczba wytworzonych artefaktów są ustalone za pomocą wartości parametrów wewnętrznych symulacji.

3.2.2. Interakcja elementów wewnętrznych rdzenia modelu

Użycie elementów wewnętrznych modelu do reprezentacji procesu wytwarzania przebiega dwutorowo. Pierwsza część działań jest związaną z kontrolą przebiegu i wyznaczaniem bieżących prac do wykonania. Druga część działań polega na wykonywaniu przez zespół projektowy działań wyznaczonych przez część kontrolującą. Interakcje elementów modelu są opisane w formie algorytmów umieszczonych w dwóch kolejnych podrozdziałach, zgodnie z wprowadzonym podziałem.

Realizacja planu projektu

Realizacja planu projektu polega na sprawdzaniu, czy bieżące działania wykonywane przez zespół projektowy zostały zakończone. W miejsce zakończonych działań wskazywane są kolejne, zgodnie z grafem przepływu działań określonym przez metodykę wytwarzania. Po zakończeniu bieżącej iteracji realizowane są następne, aż do zrealizowania całego planu. Algorytm realizacji planu projektu został zamieszczony w Alg. 3.1. W trakcie wykonywania planu projektu w procesie wytwarzania dwie zmienne wskazują na bieżący stan realizacji: ci wskazuje numer bieżącej iteracji, cv zawiera zbiór bieżących działań, które mogą być wykonywane przez zespół projektowy. Bieżące działania inicjowane są zbiorem początkowych wierzchołków grafu przepływu działań pierwszej iteracji. Po rozpoczęciu pętli repeat uruchamiana jest procedura wykonywania zadań przez zespół projektowy. Po jej zakończeniu zamknięte działania cv_i są usuwane ze zbioru bieżących działań cv i dodawane są do zbioru kolejne wierzchołki w grafie przepływu. Jeżeli zbiór bieżących działań jest pusty, to numer iteracji jest zwiększany, a do bieżących działań zapisywane wierzchołki grafu przepływu następnej iteracji (15–16). Jeżeli plan projektu został ukończony, to algorytm kończy swoje działanie.

Alg. 3.1 Algorytm realizacji planu projektu

```
1.    ci := 1 ;
2.    cv := {v_i|(w, v_i) ∈ EE_ci, w ∈ VE_ci} ;
3.      repeat
4.        wykonaj_zadania(cv);
5.        for each v_i ∈ cv
6.          begin
7.            if ζ_{v_i} then
8.              begin
9.                cv := cv/v_i;
10.               cv := cv ∪{w|(v_i, w) ∈ KE_ci};
11.             end;
12.         end;
13.       if cv = {∅}
14.       then begin
15.         ci := ci + 1;
16.         cv := {v_i|(w, v_i) ∈ EE_ci, w ∈ VE_ci};
17.       end
18.     until ci > li;
```

gdzie:
ci – numer bieżącej iteracji, $ci \in N$;
cv – zbiór bieżących działań,
w – dowolny wierzchołek przepływu etapu bieżącej iteracji I_{ci}, $w \in VE$,
EE_{ci} – zbiór wierzchołków grafu przepływu bieżącej iteracji,
VE_{ci} – zbiór krawędzi grafu przepływu bieżącej iteracji,
ζ_{v_i} – funkcja sprawdzająca, czy i–te działanie zostało zakończone.

Funkcja ζ_{v_i} wykorzystana w algorytmie sprawdza, czy wszystkie typy zadań, które się składają na działanie są zakończone. Jeśli tak, to działanie jest zakończone. Zakończone typy zadań, to takie, które osiągnęły cel wyznaczony w planie projektu dla bieżącej iteracji. Np. projekt ma rozmiar 100 UCP i w pierwszej iteracji wyznaczono cel 40% dla dyscypliny wymagań, to zadanie pozyskiwania wymagań na warsztatach powinno wyprodukować 80 instancji artefaktów typu żądanie klienta. Liczba 80 wynika z tego, że celem iteracji jest 40UCP, czyli 40 instancji artefaktów typu przypadek użycia, co odpowiada 80 żądaniom klienta.

Działania składają się z wielu typów zadań. Kolejność wykonywania zadań tych typów jest zależna od dostępności artefaktów w repozytorium. W systemie rzeczywistym wiele zadań może być wykonanych dopiero po wykonaniu innych zadań. Np. interfejs użytkownika może być zaprojektowany i wykonany dopiero wtedy, gdy zostanie zaprojektowana struktura bazy danych. Zależności między zadaniami nie zostały w modelu odtworzone na poziomie szczegółowym, tzn. na poziomie pojedynczych zadań. Zostały one odwzorowane na wyższym poziomie abstrakcji za pomocą grafu przepływu działań.

Wykonywanie zadań przez zespół projektowy

W trakcie wykonywania zadań przez zespół projektowy, na podstawie wartości zmiennej cv zawierającej bieżące niezakończone działania, dla każdej osoby wchodzącej w skład zespołu projektowego wyszukiwane są niezakończone zadania pasujące do przydzielonych danej oso-

bie ról projektowych, zgodnie z algorytmem Alg. 3.2. Sposób wykonania zadania, dla znalezionego niezakończonego typu zadania, jest zależny od klasy typu zadania, opisanej w podrozdziale 3.2.1. Algorytmy wykonania zadań różnych klas podano poniżej.

Alg. 3.2 Algorytm wykonywania zadań przez zespół projektowy

```
nazwa: wykonaj_zadania
parametry wejściowe: cv – bieżące niezakończone działania
```

```
1. for each P_i ∈ Z
2.   begin
3.     for each tz_j ∈ cv
4.       begin
5.         if tz_{j,r} ∈ P_{i,r} ∩ ¬ζ_{tz_j}
6.         then wykonaj_zadanie( tz_j);
7.       end;
8.    end;
```

gdzie:

P_i – i–ta osoba z zespołu projektowego,

Z – zespół projektowy,

tz_j – j–ty typ zadania,

cv – bieżące niezakończone działania,

$tz_{j,r}$ – rola wymagana do wykonania j–tego typu zadania,

ζ_{tz_j} – funkcja sprawdzająca, czy typ zadania jest zakończony

$P_{i,r}$ – zbiór ról projektowych przypisanych i–tej osobie.

Wykonywanie zadań klasy uogólniającej

W zadaniach klasy uogólniającej artefakty wyjściowe są tworzone tylko raz. W repozytorium jest jedna instancja dla każdego typu artefaktu wyjściowego. Przykładem jest artefakt typu słownik pojęć, który jest tworzony w jednym egzemplarzu na podstawie pojęć zamieszczonych we wszystkich artefaktach różnych typów w projekcie.

Algorytm wykonania zadania dla zadań klasy uogólniającej jest przedstawiony w Alg. 3.3. Dla wszystkich typów artefaktów wyjściowych umieszczonych w zbiorze tz_{aw} tworzone są instancje artefaktów za pomocą funkcji repozytorium γ i umieszczane w zbiorze aop. Utworzone artefakty są automatycznie zapisane w repozytorium. Te artefakty wyjściowe są łączone z instancjami artefaktów wejściowych pobieranymi za pomocą funkcji ω z repozytorium. Artefakty wejściowe stają się poprzednikami, a artefakty wejściowe następnikami w relacji wynikania instancji artefaktów. Artefakty wejściowe oznaczone są jako przetworzone przez wykonywany typ zadania tz.

Alg. 3.3 Algorytm wykonania zadania klasy uogólniającej

```
nazwa: wykonaj_zadanie_uog
parametry wejściowe: tz – typ wykonywanego zadania
```

```
1. for each aw_i ∈ tz_aw
2.   begin
3.     for each aow_j ∈ ω_{aw_i,tz,t}
4.       begin
5.         for each ap_k ∈ tz_ap
6.           begin
7.             aop_k := γ_{ap_k};
8.             aop_{k,s} := aop_{k,s} ∪ aow_j;
9.             aow_{j,p} := aow_{j,p} ∪ aop_k;
10.          end;
11.       end;
12.   end;
```

gdzie:
tz – typ wykonywanego zadania, zmienna wejściowa;
aop – zbiór instancji artefaktów produkowanych przez algorytm, zmienna wyjściowa;
aw_i – i–ty typ artefaktu wejściowego typu zadania tz,
tz_{aw} – zbiór typów artefaktów wejściowych zadania tz,
aow_j – j–ta instancja nieprzetworzonego artefaktu wejściowego,
$\omega_{aw_{i,a},tz,t}$ – funkcja repozytorium zwracająca zbiór artefaktów typu aw_i nieprzetworzonych przez zadanie typu tz w czasie t;
ap_k – k–ty typ artefaktu produkowany przez typ zadania tz,
tz_{ap} – zbiór typów artefaktów produkowanych przez typ zadania tz,
γ_{ap_k} – funkcja repozytorium szukająca instancji artefaktu typu ap_k i tworząca nową, gdy nie znajdzie.

W celu zwiększenia czytelności w podanym algorytmie zastosowano następujące uproszczenia:

- przetwarzane są wszystkie instancje artefaktów wejściowych, w rzeczywistości liczba przetworzonych artefaktów zależy od celu dyscypliny w bieżącej iteracji.

Wykonywanie zadań klasy iteracyjnej

Klasa zadań iteracyjnych jest odmianą klasy zadań uogólniających. W zadaniu klasy uogólniającej artefakty wejściowe są tworzone tylko raz, natomiast w zadaniu iteracyjnym artefakty tworzone są raz na iterację. Przykładem są różnego rodzaju przeglądy: wymagań, projektu, kodu itd. W każdej iteracji wykonywana jest praca i powstają kolejne artefakty związane z przeglądami. Algorytm zadania klasy iteracyjnej jest oparty na Alg. 3.3. Dla klasy iteracyjnej nowe instancje artefaktów wyjściowych są tworzone również wtedy, gdy istniejący artefakt wyjściowy został utworzony w iteracji innej niż bieżąca.

Wykonywanie zadań klasy modelującej

Zadanie klasy modelującej dla artefaktów wejściowych tworzy artefakty elementarne, będące składowymi modelu – artefaktu, który je wszystkie łączy. Przykładem zadania klasy modelującej jest znajdowanie przypadków użycia. Dla wszystkich nieprzetworzonych artefaktów żądań udziałowców tworzone są artefakty przypadków użycia i aktorów składających się na jeden

grupowy artefakt – model przypadków użycia. Wykonanie zadania przebiega zgodnie z Alg. 3.4.

Alg. 3.4 Algorytm wykonania zadania klasy modelującej

```
nazwa: wykonaj_zadanie_model
parametry wejściowe: tz – typ wykonywanego zadania
```

1. for each $aw_i \in tz_{aw}$
2. begin
3. $aom := \gamma_{tz_{am}}$;
4. for each $aow_j \in \omega_{aw_i,tz,t}$
5. begin
6. for each $ap_k \in tz_{ap}$
7. begin
8. $aop_k := \gamma_{ap_k}$;
9. $aop_{k,s} := aop_{k,s} \cup aow_j$;
10. $aow_{j,p} := aow_{j,p} \cup aop_k$;
11. $aop_{k,\mathrm{p}} := aop_{k,\mathrm{p}} \cup aom$;
12. $aom_s := aom_s \cup aop_k$;
13. end;
14. end;
15. end;

gdzie:
tz – typ wykonywanego zadania, zmienna wejściowa;
aop – zbiór instancji artefaktów produkowanych przez algorytm, zmienna wyjściowa;
aw_i – i–ty typ artefaktu wejściowego typu zadania tz,
tz_{aw} – zbiór typów artefaktów wejściowych zadania tz,
aom – instancja artefaktu – modelu,
aow_j – j–ta instancja nieprzetworzonego artefaktu wejściowego,
$\omega_{aw_{i,a},tz,t}$ – funkcja repozytorium zwracająca zbiór artefaktów typu aw_i nieprzetworzonych przez zadanie typu tz w czasie t;
ap_k – k–ty typ artefaktu produkowany przez typ zadania tz,
tz_{ap} – zbiór typów artefaktów produkowanych przez typ zadania tz,
γ_{ap_k} – funkcja repozytorium tworząca instancję artefaktu typu ap_k.

W celu zwiększenia czytelności w podanym algorytmie zastosowano następujące uproszczenia:

- przetwarzane są wszystkie instancje artefaktów wejściowych, w rzeczywistości liczba przetworzonych artefaktów zależy od celu dyscypliny w bieżącej iteracji.
- dla każdego artefaktu wejściowego tworzony jest jeden artefakt wyjściowy, w rzeczywistości ten stosunek zależy od typów artefaktów i jest określony parametrami.

Wykonywanie zadań klasy zespołowej

Zadania klasy zespołowej nie przetwarzają artefaktów wejściowych. Artefakty wyjściowe są tworzone dzięki współpracy osób z zespołu projektowego z przedstawicielami klienta.

Alg. 3.5 Algorytm wykonania zadania klasy zespołowej

```
nazwa: wykonaj_zadanie_zesp
parametry wejściowe: tz – typ wykonywanego zadania
1. for each ap_k ∈ tz_ap
2.   begin
3.     τ_ap_k;
4.   end;
```

gdzie:
tz – typ wykonywanego zadania, zmienna wejściowa;
ap_k – k–ty typ artefaktu produkowany przez typ zadania tz,
tz_{ap} – zbiór typów artefaktów produkowanych przez typ zadania tz,
τ_{ap_k} – funkcja repozytorium tworząca nową instancję artefaktu typu ap_k.

W celu zwiększenia czytelności w podanym algorytmie zastosowano następujące uproszczenia:

- wytwarzana jest jedna instancja artefaktów wyjściowych, w rzeczywistości liczba produkowanych artefaktów jest zależna od parametrów modelu.

3.2.3. Rozszerzenia rdzenia modelu

W poprzednim podrozdziale zostały przedstawione elementy wewnętrzne rdzenia modelu i ich interakcje pozwalające na odtworzenie procesu wytwarzania z reprezentacją cech i zjawisk niezbędnych do symulacji. W tym podrozdziale zaproponowano rozszerzenia dla rdzenia modelu o kolejne cechy i zachowania, połączone tematycznie w trzech grupach. Pierwsza z nich opisuje rozszerzenia dotyczące zespołu projektowego, druga rozszerzenia dotyczące procesu wytwarzania oprogramowania. Trzecia grupa to podmodel systemu zapewniania jakości. W każdym z proponowanych rozszerzeń korzyści płynące z ich implementacji ujęto w formie pytań, na które rozszerzony model mógłby odpowiedzieć lub przynajmniej pomóc w udzieleniu odpowiedzi.

Rozszerzenia modelu dotyczące zespołu projektowego

Rozszerzenia dotyczące zespołu projektowego obejmują możliwość podziału zespołu na grupy robocze, uwzględnienie kompetencji członków zespołu projektowego podczas wykonywania zadań, modelowanie uczenia się i nabywania kompetencji, współpracę i komunikację między osobami w jednym i wielu zespołach projektowych, wpływ lokalizacji zespołów na możliwości komunikacji oraz wpływ cech osobowości na komunikację i współpracę.

Grupy robocze

Grupy robocze składają się z członków zespołu projektowego. Możliwość podzielenia zespołu projektowego na grupy robocze została wprowadzona po to, aby przyspieszyć prace nad projektem. W warsztatach pozyskiwania wymagań uczestniczy tylko część zespołu projektowego np. analitycy, architekt, kierownik zespołu i przedstawiciele klienta. Każda grupa posiada swojego lidera, który prowadzi warsztaty oraz uczestniczących w warsztatach członków grupy. W utworzonych grupach określone są role osób, których uczestnictwo jest niezbędne i role osób, których obecność na warsztatach jest opcjonalna.

Grupy robocze pomagają zrównoleglić prace nad projektem informatycznym przez podzielenie go na poddziedziny i prace nad każdą z poddziedzin w grupach. Prace mogą również dotyczyć

takich dyscyplin, jak projektowanie, implementacja i testowanie, nie tylko wymagań. Zastosowanie rozszerzenia pozwoli użyć modelu do odpowiedzi na pytania:

1. Jak podzielić prace na poddziedziny, aby prace mogły być prowadzone równolegle?
2. Jaki powinien być skład grup, żeby nie tworzyły się wąskie gardła w procesie wytwarzania?
3. Jak podzielić zespół projektowy na grupy, aby uzyskać dużą efektywność pracy (żeby wszyscy członkowie zespołu projektowego mieli przydzielone zadania)?

Kompetencje a wydajność pracy

Każda osoba zatrudniona w projekcie może mieć inne cechy osobowości i kompetencje technologiczne i społeczne. Te różnice indywidualne wpływają na wydajność pracy, współpracę między członkami zespołu projektowego i na komunikację między nimi. Wszystkie kompetencje, które mogą być użyte do opisu członków zespołu projektowego są określone zbiorem kompetencji KM.

$$KM = \{km_i | i = 1 \dots lkm\} \tag{3.5}$$

gdzie:
KM – zbiór kompetencji,
km_i – i–ta kompetencja zbioru KM,
lkm – liczba kompetencji.

Każdej osobie z zespołu projektowego P_i powinny być przypisane wartości sekwencja składająca się z przypisanych jej ról projektowych oraz zbioru sekwencji określające jej kompetencje i ich wartości:

$$P_i = (\{r_{i,k} | k = 1 \dots lr_i\}, \{(km_n, kmw_n) | n = 1 \dots lkm_i\}) \tag{3.6}$$

gdzie:
P_i – i–ta osoba zespołu projektowego,
$r_{i,k}$ – k–ta rola projektowa i–tego członka zespołu projektowego, $r_{i,k} \in R$ zbioru ról projektowych metodyki wytwarzania M,
lr_i – liczba ról i–tej osoby zespołu projektowego Z,
km_n – n–ta kompetencja i–tej osoby, $km_n \in KM$,
kmw_n – wartość n–tej kompetencji i–tej osoby, $kmw_n \in \{1,2,3,4,5\}$
lkm_i – liczba kompetencji i–tej osoby.

Tab. 3.1 Wartości zmiennych do oceny kompetencji

Źródło: Opracowanie własne

Wartość zmiennej poziom kompetencji	Ocena lingwistyczna
1	brak
2	niskie
3	średnie
4	wysokie
5	bardzo wysokie

Czas potrzebny na wykonywanie zadania typu tz_i jest określony przez wartość zmiennej tzd_i (3.4). Wartość tej zmiennej jest przechowywana w parametrach wewnętrznych modelu, które reprezentują zmierzone średnie czasy wykonywania zadań danego typu. Czas wykonywania zadania powinien zależeć od tego, na jakim poziomie są kompetencje osoby wykonującej zadanie.

$$tw_i = tzd_i * \prod_{j=1}^{lkm_n} \frac{3}{kmw_j} \tag{3.7}$$

gdzie:
tw_i – wynikowy czas wykonania i–tego typu zadania,
tzd_i – średni czas wykonania i–tego typu zadania, wynikający z parametrów wewnętrznych modelu,
kmw_j – j–ta wartość kompetencji n–tej osoby,
lkm_n – liczba kompetencji n–tej osoby.

Uczenie się osób z zespołu projektowego

Rozszerzeniem powyższej koncepcji kompetencji jest przypisanie typom zadań kompetencji wymaganych do ich wykonania. Dzięki temu rozszerzeniu można podczas obliczania czasu wykonania zadania brać pod uwagę tylko istotne kompetencje oraz zrealizować w modelu koncepcję uczenia się osób wykonujących zadania. Kompetencje osoby wykonującej zadania o określonych i wymaganych kompetencjach do wykonania zadania danego typu powinny wzrastać wraz z liczbą wykonanych zadań tego typu.

Współpraca członków zespołu projektowego

Współpraca osób wchodzących w skład zespołu projektowego może się odbywać na trzy sposoby. Pierwszy z nich to konsultacje, których udziela osoba mająca wysoki poziom kompetencji w pewnej dziedzinie osobie, która ma niski poziom kompetencji. Jest to sytuacja, kiedy osoba mniej doświadczona prosi doświadczonego współpracownika o pomoc. Drugi sposób współpracy, to konsultacje, których udziela twórca zadania jego wykonawcy. Ma to miejsce, np. gdy tester prosi analityka systemu o wyjaśnienie pewnych zawiłości w sformułowanych przez analityka wymaganiach funkcjonalnych. Trzeci sposób współpracy ma miejsce podczas warsztatów pozyskiwania wymagań, gdy grupa członków zespołu projektowego spotyka się z przedstawicielami klienta, aby zebrać wymagania dotyczące projektowanego systemu lub podczas wywiadów, gdy przedstawiciel klienta udziela informacji na temat wymagań analitykowi.

Komunikacja między osobami

Skuteczność współpracy między osobami zależy od wybranego przez nie kanału komunikacji, dostępności czasowej osób oraz kombinacji ich cech osobowości. Jeżeli współpracujące osoby pracują w jednej lokalizacji, to mogą użyć najwydajniejszej metody komunikacji – bezpośredniej rozmowy. Ten sposób komunikacji jest zalecany i praktykowany w zwinnych metodykach wytwarzania [86]. Jednak w tradycyjnych metodykach często zdarza się, że osoby pełniące różne role projektowe pracują w różnych działach i różnych lokalizacjach, miastach, a czasami krajach lub kontynentach. Taka organizacja pracy prowadzi do używania nieinteraktywnych

strategii komunikacji np. formalnych dokumentów lub poczty elektronicznej, których efektywność jest niska. Badania nad efektywnością strategii komunikacji [92], które są oparte na teorii bogactwa przekazu (ang. *Media Richness Theory*) [22] i pracach A. Cockburna [21], doprowadziły do ilościowego określenia współczynnika efektywności strategii komunikacji. Wyniki badań ankietowych dotyczących postrzegania efektywności strategii komunikacyjnych [91] pokazano na Rys. 3.4, a opracowane na ich podstawie liczbowe wskaźniki efektywności o wartościach od –5 (bardzo nieefektywne) do +5 (bardzo efektywne) zamieszczono w Tab. 3.2.

Rys. 3.4 Efektywność strategii komunikacyjnych
Źródło: Opracowanie własne na podstawie [92]

W rozszerzonym modelu czas wykonania zadania t_c powinien składać się z czasu rzeczywistej pracy nad zadaniem t_z i czasem wymaganym na komunikację z twórcą zadania lub ekspertem dziedzinowym t_k.

$$t_c = t_z + t_k \tag{3.8}$$

gdzie:
t_c – całkowity czas wykonania zadania,
t_z – czas wykonania zadania właściwego,
t_k – czas trwania komunikacji.

Czas trwania komunikacji powinien zależeć od wybranej strategii komunikacji. Strategia komunikacji zależy od lokalizacji komunikujących się osób. Analiza komunikacji w zespołach byłaby pełniejsza, gdyby rozszerzyć ją o wiele zespołów projektowych z możliwością różnej lokalizacji zespołów. Współczynniki efektywności podane w literaturze i zamieszczone w Tab. 3.2 powinny zostać przeskalowane. Po przeskalowaniu współczynnik ez przyjmuje symbol ezs i wartości z przedziału $< \frac{1}{5}, 2\frac{1}{5} >$.

$$ezs = \frac{\mathrm{ez} + 6}{5} \tag{3.9}$$

gdzie:
ez – współczynnik efektywności strategii komunikacyjnych wewnątrz zespołu,
ezs – przeskalowany współczynnik efektywności strategii komunikacyjnych wewnątrz zespołu.

W najprostszym modelu czas wymagany na komunikację jest zależny od parametru modelu określającego średni czas wymagany na komunikację t_{sk} i przeskalowanego współczynnika efektywności:

$$t_k = \frac{t_{sk}}{ezs} \tag{3.10}$$

gdzie:
t_k – czas wymagany na komunikację,
t_{sk} – parametru modelu określającego średni czas wymagany na komunikację,
ezs – przeskalowany współczynnik efektywności.

Tab. 3.2 Współczynniki efektywności strategii komunikacyjnych
Źródło: Opracowanie własne na podstawie [92]

Strategia komunikacji	Efektywność	
	Wewnątrz zespołu ***ez***	Z udziałowcami ***eu***
Bezpośrednia (F2F)	4,25	4,06
F2F przy białej tablicy	4,24	3,46
Poglądowe diagramy	2,54	1,89
Komunikatory	2,10	0,15
Poglądowa dokumentacja	1,84	1,86
Telekonferencje	1,42	1,51
Wideokonferencje	1,34	1,62
Poczta elektroniczna	1,08	1,32
Szczegółowa dokumentacja	–0,34	0,16

Rozszerzenie modelu o analizę strategii komunikacyjnych w zespole i z udziałowcami pozwoli odpowiedzieć na następujące pytania:

1. Jak rozmieszczenie zespołów projektowych wpływa na efektywność produkcji oprogramowania?
2. Jakie ryzyko jest związane z uszkodzeniem sprzętu do komunikacji na odległość?
3. Czy dane rozmieszczenie osób pracujących w projekcie w zespołach i lokalizacjach jest właściwe, odpowiednie i sprzyja efektywnej komunikacji?

Wiele zespołów projektowych

Nowoczesne, zwinne metodyki wytwarzania oprogramowania preferują małe zespoły projektowe, których optymalna wielkość to 7–10 osób. Skalowanie zwinnych metodyk do wykorzystania w dużych projektach informatycznych wymusza użycie wielu zespołów, aby zakończyć projekt w założonym terminie. Rdzeń modelu został ograniczony do jednego zespołu. Użycie modelu do reprezentacji procesów przebiegających zgodnie ze zwinnymi metodykami wymaga rozszerzenia go o możliwość definiowania wielu zespołów projektowych. Poniżej przedstawiono kolejny aspekt analizy, w której można wykorzystać to rozszerzenie: wpływ położenia geograficznego zespołów na komunikację i współpracę.

Rozproszone zespoły projektowe

Duży wpływ na skuteczność komunikacji ma rozproszenie geograficzne współpracujących zespołów. W projektach, w których zespoły są rozlokowane na różnych kontynentach pojawia się problem pracy w różnych strefach czasowych, co zmniejsza liczbę godzin, w których obydwa zespoły są na raz dostępne. Np. jeżeli jeden zespół projektowy pracuje w Stanach Zjednoczonych, a drugi w Polsce i obydwa rozpoczynają pracę o godzinie 8 rano i kończą o godzinie 16, to z powodu 6–cio godzinnego przesunięcia czasowego pracownicy mają tylko 2 godziny dziennie na komunikację interaktywną. Analiza związana z zespołami rozproszonymi pozwoli odpowiedzieć na pytania:

1. Jak zaplanować komunikację zespołów rozproszonych?
2. Czy włączenie kolejnego zespołu przyspieszy prace nad projektem?

Dostępność członków zespołu projektowego i przedstawicieli klienta

Osoby należące do zespołu projektowego mogą być częściowo zajęte pracą w innych projektach. Taka sytuacja nie jest zalecana, ale w praktyce jest spotykana, ponieważ wpływa negatywnie na możliwość współpracy i komunikacji. Podobna sytuacja dotyczy ograniczenia dostępności przedstawicieli klienta. Są oni zajęci pracą w organizacji klienta i czasami są nakładane duże ograniczenia na liczbę godzin, którą mogą przeznaczyć na współpracę z zespołem projektowym [8].

Dostępność członków zespołu projektowego zależy również od rozłożenia świąt państwowych w kraju, w którym pracuje zespół. Część świąt ma stałe daty, a część jest ruchoma. Co prawda święta wpływają w jednakowy sposób na cały zespół projektowy, ale mają duży wpływ na współpracę i komunikację, gdy zespoły są rozmieszczone w różnych krajach.

Kolejnymi czynnikami wpływającymi na współpracę i komunikację są urlopy i różnego rodzaju zwolnienia z pracy. Liczba dni urlopu w roku jest ograniczona i z góry znana, wiadome są również plany urlopowe każdego z pracowników. Natomiast liczba dni zwolnień okolicznościowych i chorobowych jest nieznana, lecz jest bliska wartości średniej znanej z roczników statystycznych.

Uwzględnienie dostępności osób pracujących nad projektem pozwoli uzyskać bardziej realistyczne wyniki, które będzie można porównać z rzeczywistymi. Jest to szczególnie ważne przy użyciu modelu do wygenerowania wzorca procesu zgodnego z metodyką i planem projektu. Porównanie rzeczywistego przebiegu projektu z wzorcem nie uwzględniającym weekendów, świąt, i urlopów zaniżałoby ocenę procesów wytwórczych organizacji. Ponadto rozszerzenie pozwalające uwzględnić dostępność osób, umożliwia odpowiedź na pytania:

1. Na jakim etapie planowanego projektu osoby częściowo pracujące w innym projekcie powinny być w pełni włączone do planowanego projektu?
2. Jaki wpływ na wykonanie projektu ma dostępność członków zespołu projektowego?
3. Jaki wpływ na wykonanie projektu mają ograniczenia dostępności przedstawicieli klienta?
4. Jakie jest ryzyko związane z okresem urlopowym lub okresem zwiększonych zachorowań?
5. Jak rozplanować urlopy członków zespołu projektowego?
6. Jaki wpływ będą miały zaplanowane urlopy i święta na produkcję?

Cechy osobowości członków zespołu projektowego

Cechy osobowości mają duży wpływ na współpracę i komunikację w zespole projektowym. Koncepcja modelowania tego wpływu na proces wytwarzania jest oparta na badaniach i współpracy z Ireną Bach–Dąbrowską [6, 69]. Koncepcja bazuje na jednym z najbardziej znanych na świecie modeli osobowości Myers–Briggs Type Indicator (MTBI) [37]. Osobom wchodzącym w skład zespołu projektowego przypisany jest jeden z szesnastu profili osobowości MTBI. Analiza komunikacji i współpracy, bazująca na zależności pracochłonności od kanałów komunikacji i dostępności, powinna być rozszerzona o czynnik komunikacji zależny od typów osobowości komunikujących się osób. Dzięki temu rozszerzeniu modelu można uzyskać odpowiedzi na następujące pytania:

1. Czy przyjęcie osoby o określonym typie osobowości do zespołu projektowego zwiększy jego wydajność?
2. Czy przypisanie roli projektowej osobie o określonym typie osobowości polepszy komunikację w zespole?

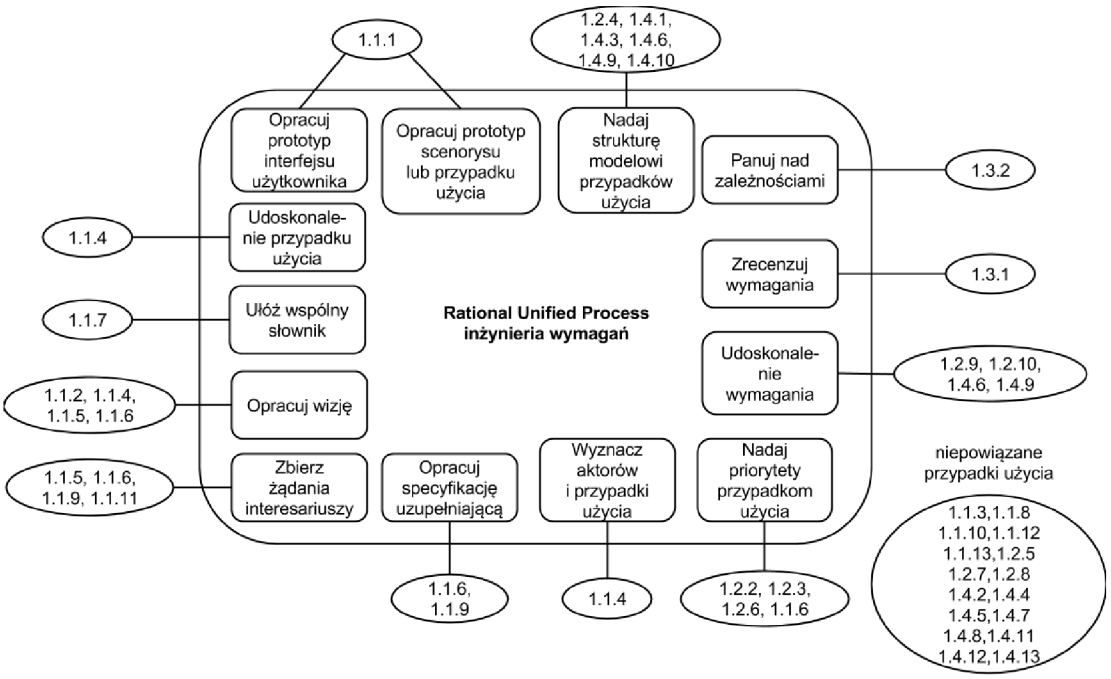
Rozszerzenia modelu dotyczące procesu wytwarzania

Rozszerzenia modelu zgrupowane wokół tematyki procesu wytwarzania zawierają: modelowanie narzędzi i technologii, zmiany w parametrach wejściowych modelu podczas prac nad projektem, możliwość nakładania się iteracji, modelowanie metodyk zwinnych i możliwość dopasowania metodyki do wytwarzanego projektu.

Narzędzia i technologie wspierające proces wytwarzania oprogramowania

To rozszerzenie pozwoli określić wpływ użytych narzędzi na prace zespołu projektowego. Dzięki temu uzyskana zostanie możliwość doboru narzędzi do metodyki, specyfiki projektu i składu zespołu projektowego. Koncepcja rozszerzenia wymaga, aby członkowie zespołu mieli przypisane określone kompetencje posługiwania się różnymi narzędziami lub technologiami wspierającymi proces wytwarzania oprogramowania.

Analiza przydatności narzędzi do modelowanego projektu, metodyki i zespołu projektowego jest oparta na wspólnej pracy z Bartoszem Chrabskim, nad powiązaniem działań metodyki RUP z przypadkami użycia narzędzi Rational firmy IBM oraz narzędzi typu Open Source [36]. Przykładowe powiązanie działań dyscypliny wymagań metodyki RUP z przypadkami użycia oprogramowania IBM Rational Requirements Composer zostały pokazane na Rys. 3.5. Badania nad narzędziami zostały rozszerzone na inne metodyki niż RUP, w tym metodyki zwinne.



Rys. 3.5 Powiązanie działań dyscypliny wymagań metodyki RUP z przypadkami użycia oprogramowania IBM Rational Requirements Composer.

Źródło: Opracowanie własne

Ocena przydatności narzędzia polegałaby na zliczeniu liczby przypadków użycia wspierających zadania wykonane przez zespół projektowy w badanym projekcie. Ocena przydatności pozwoliłaby na odpowiedni dobór narzędzi do zespołu projektowego, metodyki i projektu oraz umożliwiłaby dobór darmowych narzędzi wspierających wytwarzanie oprogramowania zastępujących rozwiązania komercyjne, z możliwością porównania ich przydatności do przydatności rozwiązań komercyjnych. Takie porównanie mogłoby umożliwić dobór darmowych narzędzi zastępujących narzędzia komercyjne.

Modelowanie zmian w procesie wytwarzania

W trakcie trwania procesu wytwarzania oprogramowania często zachodzą zmiany w wartościach parametrów wejściowych. Najważniejszym parametrem, który ulega zmianie, to skład zespołu projektowego. Doświadczeni pracownicy odchodzą z pracy i na ich miejsce są zatrudniane nowe osoby, które dopiero wdrażają się w projekt i nabywają kompetencji dziedzinowych i technologicznych.

W procesie wytwarzania oprogramowania może zmienić się także zakres projektu informatycznego, plan projektu i metodyka wytwarzania. Zmiany parametrów wejściowych w modelu są reprezentowane przez zapisanie w odpowiednim rejestrze parametrów wejściowych RPW nowych wartości zmiennych wejściowych i daty, od której obowiązują.

$$RPW = \{(pd_i, pz_i, pw_i) | i = 1 \dots lrpw\} \tag{3.11}$$

gdzie:

RPW – rejestr parametrów wejściowych,
pd_i – data i czas i–tej zmiany w umownych jednostkach czasu,
pz_i – zmienna wejściowa i–tej zmiany,

pw_i – nowa wartość zmiennej wejściowej pz_i i–tej zmiany,
$lrpw$ – liczba zmian wartości zmiennych wejściowych modelu.

Nakładanie iteracji w projektach RUP

W standardowym przebiegu procesu RUP analitycy systemowi pracujący nad wymaganiami po zakończeniu prac w danej iteracji czekają na wykonanie prac przez resztę zespołu projektowego i rozpoczęcie nowej iteracji. W tym czasie oczekiwania mogliby rozpocząć prace nad wymaganiami zaplanowanymi na kolejną iterację. To podejście do wykonywania RUP jest nazwane nakładającymi się iteracjami (ang. *overlapping iterations*) [3]. W modelu można odtworzyć to podejście przez umożliwienie uruchomienia kolejnych działań z grafu przepływu, kiedy wykonywane zadania bieżących działań osiągnęły określony procent zaawansowania. Np. przy określonej wartości 50% ukończonych działań z obszaru projektowania, programiści mogą rozpocząć implementowanie projektu i następnie działania z obszaru projektowania i implementacji będą przebiegać równolegle.

Metodyki zwinne

Rozszerzenie modelu o możliwość reprezentacji zwinnych procesów wytwarzania jest niezwykle istotna z powodu ich coraz większej popularności. Rozszerzenie mogłoby zostać również wykorzystane do modelowania procesów hybrydowych, składających się z praktyk tradycyjnych i zwinnych. Jego potencjalne wykorzystanie to tworzenie modeli procesów wytwarzania organizacji w trakcie zwinnej transformacji.

Proponowane podejście do rozszerzenia modelu o metodyki zwinne polega na rozbudowie dotychczasowego układu eksperymentu. Plan projektu złożony z sekwencyjnie wykonywanych iteracji należących do etapów jest dostosowany do opisu procesu wykonywanego zgodnie z metodykami tradycyjnymi. Metodyki zwinne wymagają rozbudowy planu projektu, ponieważ występujące w nich iteracje są z góry ograniczone czasowo, natomiast tradycyjne iteracje mają założony cel do osiągnięcia. W metodykach zwinnych występują również zdarzenia codzienne, które powinny być uruchamiane o określonej godzinie. W metodykach tradycyjnych zdarzenia codzienne nie występują. Przed rozpoczęciem wykonywanie działań przepływu zdarzenia codziennego aktualnie wykonywane działania powinny być zawieszone, a zadania wykonywane przez członków zespołu projektowego powinny być zakończone. Po zakończeniu wykonywania działań zdarzenia codziennego uprzednio wstrzymane działania powinny być przywrócone.

Dopasowanie metodyki do charakterystyki projektu

Metodyki są zbiorem dobrych praktyk stosowanych w procesie wytwarzania oprogramowania. Dokumentacje i podręczniki przedstawiające metodykę zwykle opisują najbardziej obszerną, złożoną jej wersję i starają się zaprezentować wszystkie możliwości. Do kierownika projektu należy dobranie odpowiednich praktyk, poziomu formalności i ilości dokumentacji do specyfiki projektu. Prosty projekt małych rozmiarów nie potrzebuje tak dużo dokumentacji, jak projekt duży i złożony. Na przykład w metodyce RUP stosowanie wszystkich jej artefaktów i praktyk w pojedynczym projekcie, bez analizy przydatności, jest traktowane jako błąd [58].

To rozszerzenie powinno zapewnić zmianę ilości wymaganej dokumentacji, dodawanie i usuwanie typów zadań i wytwarzanych artefaktów. Można to osiągnąć przez definiowanie własnych typów zadań, które wytwarzają uproszczone wersje artefaktów, wymagające mniej czasu, niż pełne wersje artefaktów.

Dopasowanie metodyki w połączeniu z modelowaniem systemu zapewniania jakości, pozwala zaprojektować proces wytwarzania oprogramowania o odpowiedniej jakości produktu końcowego.

Rozszerzenia modelu dotyczące systemu zapewniania jakości

Możliwość modelowania działania systemu zapewniania jakości, przez wprowadzenie odpowiednich działań, mających na celu wyeliminowanie błędów oraz przyspieszenie procesu wytwarzania oprogramowania, ma bardzo istotne znaczenie dla planowania, wyznaczenia kosztów oraz dla oceny ukończonego procesu wytwarzania oprogramowania. Podstawowe założenie systemu zapewniania jakości jest takie, że im wcześniej w procesie wytwarzania błąd jest wykryty, tym mniejszy jest koszt jego naprawy. Największe koszty generują błędy wykryte w trakcie wdrożenia gotowego systemu u klienta, ponieważ często trzeba wrócić do początkowych faz wytwarzania i poprawić wymagania, a co za tym idzie wszystkie zależne od nich artefakty utworzone w kolejnych etapach: projekt, bazę danych, interfejs użytkownika, implementację i testy. Błędy wykryte w przeglądzie wymagań powodują kilkukrotnie mniejsze koszty niż błędy wykryte w późniejszych etapach.

Powstawanie błędów

Modelowanie jakości oprogramowania zostało oparte na pojęciach potencjału błędu oprogramowania i usuwaniu błędów. Pojęcie potencjału błędu zostało wprowadzone w roku 1973 przez firmę IBM. Oznacza ono sumę błędów wszystkich klas, które mogą wystąpić podczas: pozyskiwania wymagań, projektowania, kodowania, tworzenia dokumentacji, bazy danych, testowania i naprawiania błędów [49]. Potencjał błędu jest wyrażany jako liczba błędów na jednostkę rozmiaru oprogramowania. W Tab. 3.3 pokazano średnią wartość potencjału błędu przypadającą na liczbę punktów przypadków użycia w zależności od pochodzenia.

Tab. 3.3 Potencjał błędu wg pochodzenia

Źródło: Opracowanie własne na podstawie [49]

Źródło błędów	Liczba błędów na UCP	Procent wszystkich błędów
Wymagania	1,15	9,58%
Architektura	0,25	2,08%
Projektowanie	1,50	12,50%
Kodowanie	2,00	16,67%
Testy	1,85	15,42%
Dokumentacja	0,75	6,25%
Baza danych	2,75	22,92%
Strony internetowe	1,75	14,58%
Razem	**12,00**	**100,00%**

Modelowanie potencjału błędu polega na oznaczeniu, w momencie utworzenia, odpowiedniej liczby instancji artefaktu jako zawierających błąd. Artefakty, których źródłowe artefakty są obarczone błędem, również są traktowane jako zawierające błąd. Liczba artefaktów zawierających błąd zależy od dyscypliny, do której należy wykonywane zadanie i od zakresu projektu

informatycznego ZP. Definicję instancji artefaktu oa_i (3.2) należało rozszerzyć o informację o błędzie er_i, która została umieszczona w sekwencji:

$$oa_i = (a_i, s_i, p_i, er_i) \tag{3.12}$$

gdzie:
er_i – informacja o błędzie, $er_i \in \{0,1\}$.

Usuwanie błędów

Kolejnym elementem modelowania systemu zapewnienia jakości jest usuwanie powstałych błędów. Pojęcie usuwania błędów jest związane z metodami zapobiegania błędom, działaniami wspierającymi usuwanie ich przed fazą testów i usuwaniem ich podczas testowania. Każde z tych działań ma określoną skuteczność usuwania, której wartość określa procent usuniętych błędów. Największą skuteczność usuwania błędów uzyskuje się przez łączenie różnych metod. Zadania usuwające błędy, opierając się na danych o skuteczności zamieszczonych w literaturze [46–49] i przedstawionych w Tab. 3.4, usuwają z repozytorium artefakty zawierające błąd. Usunięcie artefaktu zawierającego błąd usuwa z repozytorium także wszystkie pochodne artefakty, które powstały na jego podstawie. Powoduje to konieczność ponownego utworzenia usuniętych artefaktów oraz artefaktów od nich pochodnych. W ten sposób modelowana jest proporcjonalna zależność między czasem wykrycia błędu a kosztem jego naprawienia. Jeżeli błąd w artefakcie zostanie wykryty szybko, to niewiele pochodnych artefaktów zostanie usuniętych, więc mniej pracy będzie potrzebne do ich odtworzenia. Na przykład zadanie formalnej inspekcji wymagań, przeprowadzone przed rozpoczęciem etapu projektowania należy do działań wspierających usuwanie błędów przed fazą testów i ma efektywność od 50% do 90%.

Tab. 3.4 Efektywność usuwania błędów

Źródło: Opracowanie własne na podstawie [49]

Przedtestowe usuwanie błędów	Minimalna	Średnia	Maksymalna
Formalna inspekcja projektu	65,00%	87,00%	97,00%
Formalna inspekcja kodu	60,00%	85,00%	96,00%
Analiza statyczna	65,00%	85,00%	95,00%
Formalna inspekcja wymagań	50,00%	78,00%	90,00%
Programowanie w parach	40,00%	55,00%	65,00%
Koleżeński przegląd kodu	35,00%	50,00%	60,00%
Sprawdzanie biurkowe	25,00%	45,00%	55,00%
Średnia	48,57%	69,29%	79,71%
Usuwanie błędów podczas testów	**Minimalna**	**Średnia**	**Maksymalna**
Testowanie bezpieczeństwa	50,00%	65,00%	80,00%
Testowanie procedur	27,00%	45,00%	60,00%
Testowanie systemu	27,00%	42,00%	55,00%
Zewnętrzne beta testy	30,00%	40,00%	50,00%
Testy wydajności	30,00%	40,00%	45,00%
Testowanie funkcjonalności	33,00%	40,00%	55,00%
Testy jednostkowe automatyczne	20,00%	40,00%	50,00%
Testy jednostkowe ręczne	15,00%	38,00%	50,00%
Testy regresyjne	35,00%	35,00%	45,00%
Niezależna weryfikacja	20,00%	35,00%	47,00%
Metodyka clean–room	20,00%	35,00%	50,00%
Testy akceptacyjne	15,00%	35,00%	40,00%
Niezależne testowanie	15,00%	35,00%	42,00%
Średnia	25,92%	40,38%	51,46%

Pomiar jakości wytworzonego oprogramowania

Model podstawowy umożliwia określenie dokładnej liczby błędów w wytworzonym oprogramowaniu. Jest to suma liczby artefaktów w repozytorium oznaczonych jako zawierające błąd i liczby artefaktów od nich pochodnych. Procentowy stosunek liczby artefaktów niezawierających błędu do liczby wszystkich artefaktów określa jakość wytworzonego oprogramowania. Jest to wypadkowa wszystkich działań skierowanych na zapewnienie jakości, która umożliwia dostosowanie metodyki procesu wytwarzania do potrzeb jakościowych projektu. Dobrze zaprojektowana kombinacja działań zapobiegającym błędom i usuwających błędy pozwala zmniejszyć koszty wytwarzania oprogramowania o 50% [49].

3.3. Analiza dziedzinowa modelu podstawowego

Przedstawiony model podstawowy jest obszerny i obejmuje wiele zagadnień, które nie są niezbędne z punktu widzenia przeznaczenia modelu, którym jest ocena i planowanie procesów wytwarzania przeprowadzonych zgodnie z tradycyjnymi metodykami wytwarzania. Elementy modelu podstawowego zostaną przeanalizowane pod kątem przydatności do oceny procesu wytwarzania w kolejności, w jakiej zostały opisane. Elementy, które są zbędne w przyjętym układzie eksperymentu zostaną uproszczone lub zredukowane i w ten sposób zostanie utworzona uproszczona wersja modelu podstawowego.

Opis metodyki wytwarzania jest niezbędny do oceny procesu wytwarzania. Co więcej, metodyka powinna być opisana w sposób umożliwiający dostosowanie jej do specyfiki wytwarzanego projektu. Metodyka opisana przy pomocy typów zadań i typów artefaktów połączonych w skierowany graf działań pozwala na elastyczne dopasowanie modelu do przyjętej metodyki oraz w przyszłości daje możliwość zastosowania modelu w szerszej ramie układów eksperymentu.

Część modelu opisująca system zapewniania jakości z punktu widzenia przydatności w układzie eksperymentu jest zbędna. Jest to właściwie osobny podmodel luźno sprzężony z modelem głównym, który można usunąć z wynikowego modelu uproszczonego bez utraty spójności modelu podstawowego.

Zmienna wejściowa plan projektu reprezentuje podział procesu wytwarzania projektu na etapy, iteracje i dyscypliny, oparty na opisie procesu RUP. Plan projektu steruje przebiegiem prac w modelu reprezentującym implementację wybranej metodyki. Wynikowy przebieg prac jest wzorcem, który jest porównywany z badanym przebiegiem. Z punktu widzenia układu eksperymentu plan projektu jest niezbędny, więc zostanie wykorzystany w modelu uproszczonym.

Modelowanie zmian parametrów wejściowych w procesie wytwarzania oprogramowania nie jest istotne w przyjętym układzie eksperymentu, ponieważ model będzie odtwarzał zakończone procesy wytwarzania i w chwili odtwarzania parametry wejściowe będą znane. Jedynym wyjątkiem jest zmienna wejściowa określająca skład zespołu projektowego, który w rzeczywistości często się zmienia. Możliwość określania zmiany w okresach zatrudnienia członków zespołu projektowego oraz ich zaangażowania w prace w projekcie powinna być zachowana w modelu uproszczonym.

Prace nad modelem podstawowym w części dotyczącej zespołu projektowego są bardzo rozbudowane. Z punktu widzenia przydatności tych elementów modelu w przyjętym układzie eksperymentu większość z nich jest nadmiarowa. Zespół projektowy można potraktować jako zbiór osób z przypisanymi rolami projektowymi zdefiniowanymi w przyjętej metodyce wytwarzania. Kompetencje technologiczne i społeczne, cechy osobowości, strategie komunikacyjne, rozmieszczenie zespołów mogą być zredukowane i usunięte z modelu uproszczonego. Istotnym elementem modelu podstawowego jest możliwość odwzorowania świąt, urlopów i nieobecności pracowników, ponieważ wzorzec procesu utworzony przez model będzie porównywany z rzeczywistym procesem, w którym nieobecności pracowników często występują. Możliwość odtworzenia nieobecności pracowników zostanie zatem zachowana w modelu uproszczonym.

Model uproszczony uzyskany przez usunięcie złożonego podmodelu systemu jakości, podmodelu kompetencji, współpracy i komunikacji w zespole oraz podmodelu zmian parametrów procesu jest elastycznym, kompletnym i w pełni funkcjonalnym modelem procesu wytwarzania oprogramowania, którego zmienna wyjściowa może stanowić wzorzec do oceny rzeczywistych procesów wytwarzania.

3.4. Dobór metody modelowania

Po przeprowadzeniu analizy dziedzinowej i ograniczeniu modelu podstawowego do granic wytyczonych przez przyjęty układ eksperymentu, co określiło zakres modelu uproszczonego, można dobrać odpowiednią dla modelu metodę modelowania. Mimo silnego ograniczenia mo-

delu, usunięte elementy nadal będą rozważane w doborze metody modelowania, ponieważ model podstawowy ma duży potencjał badawczy, praktyczny i edukacyjny. Dobór metody modelowania rozpocznie się od przeglądu znanych metod modelowania i badań dotyczących zastosowań ich w modelowaniu procesu wytwarzania oprogramowania. Następnie zostanie wybrana taka metoda modelowania, która pozwoli zaimplementować model uproszczony, jednocześnie nie wykluczając rozbudowy o elementy modelu podstawowego. Dla wybranej metody modelowania zostanie opisana koncepcja zastosowania jej do realizacji modelu uproszczonego.

3.4.1. Przegląd metod modelowania

Po określeniu modelu uproszczonego należy wybrać odpowiednie metody modelowania. Badania nad metodami modelowania są bardzo rozległą dziedziną wiedzy, dlatego dokonano wstępnego doboru metod pod kątem zastosowań w modelowaniu procesu wytwarzania. Poniżej zostały przedstawione w skrócie wybrane metody modelowania, zastosowania w modelowaniu procesu wytwarzania oprogramowania oraz dotychczasowe badania.

Modele analityczne

Idea tworzenia modeli w celu lepszego zrozumienia procesu wytwarzania oprogramowania nie jest nowa. Prace nad modelami analitycznymi [2, 12] rozpoczęły się pod koniec lat 70–tych i na początku lat 80–tych XX wieku. Te modele składały się z równań matematycznych, które reprezentowały związki między różnymi elementami procesu. Przykładami tych modeli są:

- model punktów funkcyjnych [2], służący do oszacowania wielkości produktu i związanej z nim pracochłonności,
- model COCOMO[12] służący do oszacowania harmonogramu i pracochłonności procesu wytwarzania
- model Rayleigha [74], pomagający oszacować profil personelu, oraz
- modele wzrostu niezawodności [29], w celu oszacowania pracochłonności testowania.

Mimo ich użyteczności w określonych zastosowaniach, modele analityczne posiadały wady: oddzielały atrybuty procesu od ich efektów i nie były w stanie odtworzyć dynamicznego oddziaływania między czynnikami procesu [24]. Z powodu tych wad zaczęto używać modeli symulacyjnych, które zostaną przedstawione w kolejnych podrozdziałach.

Modele ciągłe w czasie

Modele ciągłe w czasie są oparte na teorii przestrzeni stanów i reprezentują działanie systemu za pomocą równania różniczkowego lub układu równań różniczkowych nazywanych układem równań stanu. To podejście jest nazywane metodą dynamiki systemów i zostało wprowadzone pod koniec lat 50 XX w. Pierwsze szeroko znane zastosowanie dynamiki systemów do opisu procesu wytwarzania oprogramowania zostało zaprezentowane przez Abdela–Hamida [1]. Dynamika systemów umożliwia ujęcie najistotniejszych właściwości procesu wytwarzania, produktu, personelu, które są powiązane z planowaniem, śledzeniem i kontrolą składającą się na zarządzanie procesem. Wytyczne dotyczące tworzenia i walidacji modeli dynamiki systemów można znaleźć w [81, 102].

Na bazie badań systemów społecznych [32], wykorzystując dynamikę systemów, przeprowadzono badania długoterminowej ewolucji oprogramowania [62, 80]. Dynamika systemów była stosowana do badania problemów związanych z pracochłonnością i jakością oprogramowania

[1]. Były one podstawą do kolejnych badań m.in. nad: planowaniem i kontrolą w zakresie projektu [75], wybranymi częściami cyklu życia oprogramowania [30, 38], utworzeniem środowiska edukacyjnego [64].

Modele dyskretnych zdarzeń

Model dyskretnych zdarzeń reprezentuje działanie systemu jako sekwencję dyskretnych zdarzeń w czasie. Każde zdarzenie występuje w określonym czasie i powoduje zmianę stanu systemu. W czasie między zdarzeniami nie następują zmiany stanu systemu, więc symulacja może przechodzić bezpośrednio od poprzedniego do następnego zdarzenia. Proces wytwarzania oprogramowania jest reprezentowany jako ciąg obiektów przetwarzanych przez sekwencję działań. Każdy obiekt może być opisany przez różne wartości atrybutów. Zmiany wartości atrybutów przez działania mogą posłużyć do modelowania zmiany stanu systemu. Pracochłonność lub czas trwania każdego z działań może być wyznaczana za pomocą rozkładu losowego, co umożliwia zamodelowanie niepewności procesu. To podejście pozwala uchwycić podczas symulacji skutki zmian w przetwarzanych obiektach (takich jak wielkość, złożoność, liczba błędów, rodzaje błędów i tak dalej), w każdym z działań.

Symulacja modelu dyskretnych zdarzeń pozwala analizować różne próbki wartości parametrów wejściowych w różnych procesach wytwarzania oprogramowania. To pozwala potraktować symulowane procesy jak sieci transportowe których przepustowość, natężenie ruchu i zatory mogą być oceniane zgodnie z alternatywą (heurystyczną lub statystyczną) otrzymując w wyniku czasy przyjazdu i odstępy serwisowe. Eksperci zarządzania procesami uważają, że korzystanie z symulacji modelu dyskretnych zdarzeń ułatwia wykrycie i obserwację wąskich gardeł procesu oraz wybór najlepszej z istniejących alternatyw [24, 42, 79].

Modele stanowe

Model stanowy (ang. *state–based simulation model*) jest oparty na idei diagramów stanu, reprezentuje działania procesu i dynamikę procesu przez przejścia między stanami wyzwalane zdarzeniami. To podejście zapewnia graficzną reprezentację procesu wytwarzania oprogramowania, która jest bardziej formalna i dopracowana, niż diagram przepływu danych i zbliżona do sieci Petriego. Modele stanowe łatwo przedstawiają działania prowadzone równolegle, dzięki czemu stają się atrakcyjnym narzędziem do analizy problemu „wąskich gardeł". Na przykład zakończenie projektowania może wyzwolić jednocześnie rozpoczęcie działań kodowania i testów jednostkowych. Raffo używał modelu stanowego do oszacowania czasu, kosztu i jakości w szczegółowym modelu zastosowanym w praktyce [78]. Model stanowy został również zastosowany do przewidywania konsekwencji możliwych zmian w procesie wytwarzania oprogramowania [79].

Modele oparte na wiedzy

Modelowanie oparte na wiedzy jest procesem tworzenia komputerowego, interpretowalnego modelu wiedzy lub standardowych specyfikacji o procesie, obiekcie lub produkcie. W modelach opartych na wiedzy stany procesu wytwarzania mogą być zadeklarowane jawnie lub domyślnie. Domyślna reprezentacja oznacza, że stany procesu odpowiadają chwilowym wartościom bazy wiedzy zbudowanej z wzajemnie powiązanych obiektów, atrybutów i relacji wykorzystywanych do rejestracji i zarządzania symulacją. Model oparty na wiedzy pozwala także

symbolicznie oceniać klasę procesu bez wartości instancji. Wreszcie, symulacja oparta na wiedzy wykorzystuje rozumowanie w postaci wzorców reguł wnioskowania, który jest realizowany za pośrednictwem bazy reguł produkcji [65].

Modele agentowe

Model agentowy reprezentuje system rzeczywisty przy pomocy autonomicznych agentów i ich interakcji ze środowiskiem i ze sobą nawzajem. Model agentowy jest rodzajem modelu mikroskalowego [60, 70], który symuluje wiele zachodzących naraz zdarzeń i interakcji między agentami w celu odtworzenia i możliwości przewidywania złożonych zjawisk. Ten typ modelu może być użyty do rozwiązania problemu, które jest trudny lub niemożliwy do rozwiązania przy użyciu pojedynczego agenta lub modelu jednolitego. Model agentowy powinny charakteryzować następujące cechy:

- agenty dysponują niepełną informacją o całym systemie i mają ograniczone możliwości działania,
- sterowanie systemem jest zdecentralizowane,
- rozproszenie danych,
- asynchroniczne obliczenia [44]

Agent jest jednostką obliczeniową posiadającą zdolność do samodzielnych, elastycznych działań w typowym dynamicznym i niedeterministycznym środowisku. Postrzega środowisko, w którym jest umieszczony i potrafi na nie oddziaływać. Jego działania powinna charakteryzować:

- reaktywność – agenty powinny obserwować środowisko i reagować w odpowiednim czasie na zachodzące w nim zmiany
- proaktywność – agenty powinny nie tylko działać w odpowiedzi na zmiany w środowisku, ale powinny wykazywać skierowane na cel zachowanie i podejmować inicjatywę w odpowiednich przypadkach
- społeczność – agenty powinny współdziałać w stosownych przypadkach z innymi agentami i ludźmi w celu osiągnięcia swoich własnych celów, lub pomocy innym w osiągnięciu ich celów [44]

Prace nad zastosowaniem modelu agentowego do opisu procesu wytwarzania oprogramowania rozpoczęto na początku lat 90–tych [65]. Mimo, że jak wykazano, model agentowy „jest naturalnym sposobem opisu zarówno komunikacji między osobami, charakterystyk osób i dyskretnych decyzji podejmowanych podczas negocjacji” [105], to ilość badań nie jest duża. Rozszerzony, systematyczny przegląd publikacji dotyczących modeli procesów wytwarzania oprogramowania przeprowadzony w roku 2010 wykazał tylko 5 publikacji na temat zastosowania modeli agentowych [111]. Jedna z nich dotyczy zastosowania modelu agentowego do dokładniejszego planowania prac zespołu projektowego [51], inna kładzie nacisk na wykorzystanie czynniki jakościowe do opisu zespołu projektowego, np. złożoność projektu lub motywacja deweloperów [80]. W kolejnych publikacjach z tego okresu również podkreślana jest istotność czynników jakościowych, takich jak: techniczne i społeczne umiejętności i kompetencje, oraz zdolność uczenia się deweloperów oraz ich wpływ na dynamikę procesu wytwarzania i efektywność zespołu projektowego [23]. W późniejszych publikacjach znajdujemy zastosowanie modelu agentowego do oszacowania wydajności pracy i zachowań programistów [96].

3.4.2. Koncepcja zastosowania wybranej metody modelowania

Z przedstawionych metod modelowania symulacyjnego najmniej spójną z koncepcjami rozważanymi w modelu podstawowym jest modelowanie wykorzystujące dynamikę systemów. Użycie dynamiki systemów oznacza posługiwanie się równaniami, które w swojej istocie uśredniają wartości, atrybuty i charakterystyki pojedynczych elementów procesu. Jest to model na poziomie skali makro, podczas gdy w modelu podstawowym uwaga skierowana jest na analizę wpływu wartości atrybutów, własności i zachowań na poziomie mikroskali na własności, cechy i dynamikę procesu wytwarzania oprogramowania na poziomie makro.

Metoda modelowania dyskretnych zdarzeń pasuje do podejścia przedstawienia procesu wytwarzania oprogramowania jako opisanego przez metodykę składającą się z podstawowych typów zadań oraz przez plan projektu opisującego kolejne etapy procesu. Proces w takim ujęciu składa się z ciągu zadań wykonywanych przez członków zespołu projektowego. Wykonanie zadania jest dyskretnym zdarzeniem, które trwa pewną liczbę skwantowanych jednostek czasu i kończy się wyprodukowaniem lub przetworzeniem pewnej liczby dyskretnych jednostek informacji.

Podejście oparte na modelowaniu stanów przełączanych zdarzeniami jest spójne z modelem podstawowym. Stanem modelu są w tym podejściu kolejne etapy i iteracje planu projektu, natomiast przełączającymi zdarzeniami jest osiągnięcie celu iteracji, czyli odpowiednia zawartość repozytorium lub zdarzenia czasowe w przypadku zwinnych metodyk wytwarzania.

Podejściem najwięcej wnoszącym do koncepcji realizacji modelu podstawowego jest model oparty na systemie wieloagentowym. System wieloagentowy pozwala w prosty sposób zamodelować zespół projektowy za pomocą autonomicznych, komunikujących się ze sobą i współpracujących agentów. Podejście agentowe wspiera analizy oparte na komunikacji, kompetencjach technologicznych i społecznych oraz cechach osobowości. Każdy z agentów może mieć zestaw cech i atrybutów na dowolnym poziomie szczegółowości, co pozwoli przeprowadzać symulacje i analizy oparte na podmiotowym traktowaniu reprezentowanych przez agentów uczestników modelowanego procesu. Agenty jako byty aktywne, obdarzone autonomią i swobodą działania mogą być obdarzone inteligencją, która pomaga im reagować na zmiany zachodzące w środowisku i podejmować decyzję. Możliwości związane z inteligencją agentów rozciągają się od prostych automatów skończonych [87] przez wnioskowanie regułowe [33, 116] do modeli odtwarzających racjonalne działanie (model BDI) [35, 104]. To spektrum możliwości pozwala na modelowanie kompetencji, emocji, motywacji i komunikacji członków zespołu projektowego. Obok inteligencji jednostek ważne jest też odtworzenie relacji między jednostkami, które składa się z protokołów prostej wymiany informacji, konkurowania, współpracy, negocjacji, licytacji [27, 76]. Podejście agentowe pozwala rozważać i budować modele relacji społecznych, które są istotne dla pracy w zespole.

Wynikowy model został skonstruowany przy użyciu wyżej wymienionych metod modelowania. Jest to podejście agentowe oraz obiektowe z elementami modelu dyskretnego i modelu bazującym na stanach. Elementy aktywne modelu są reprezentowane przez agenty umieszczone w środowisku agentowym. Elementy pasywne modelu są reprezentowane przez obiekty umieszczone w środowisku, do których agenty mają dostęp, mogą z nich odczytywać informacje i wykonywać na nich operacje. Osoby wchodzące w skład zespołu projektowego są reprezentowane przez agenty w relacji 1:1, czyli jedna osoba jest reprezentowana przez jednego agenta. Oprócz agentów – osób w modelu występuje symulator – agent nadzorujący przebieg

symulacji. Czas w procesie symulacji przebiega w sposób dyskretny, oparty na skwantowanej jednostce. Przebieg czasu jest elementem metody modelowania dyskretnych zdarzeń. Nie zastosowano jednak metody wyznaczania kolejnych zdarzeń, aby ominąć momenty, w których zdarzenia nie występują. Zegar symulacji jest zrealizowany jako agent modelu, który odlicza kolejne takty bez przeskoków, ponieważ agenty – osoby prawie zawsze wykonują zadania, a ponadto często utrzymują ze sobą komunikację. Zastosowanie kolejkowania zdarzeń nie przyspieszyłoby realizacji symulacji modelu.

3.5. Analiza możliwości weryfikacji zasadności modelu uproszczonego

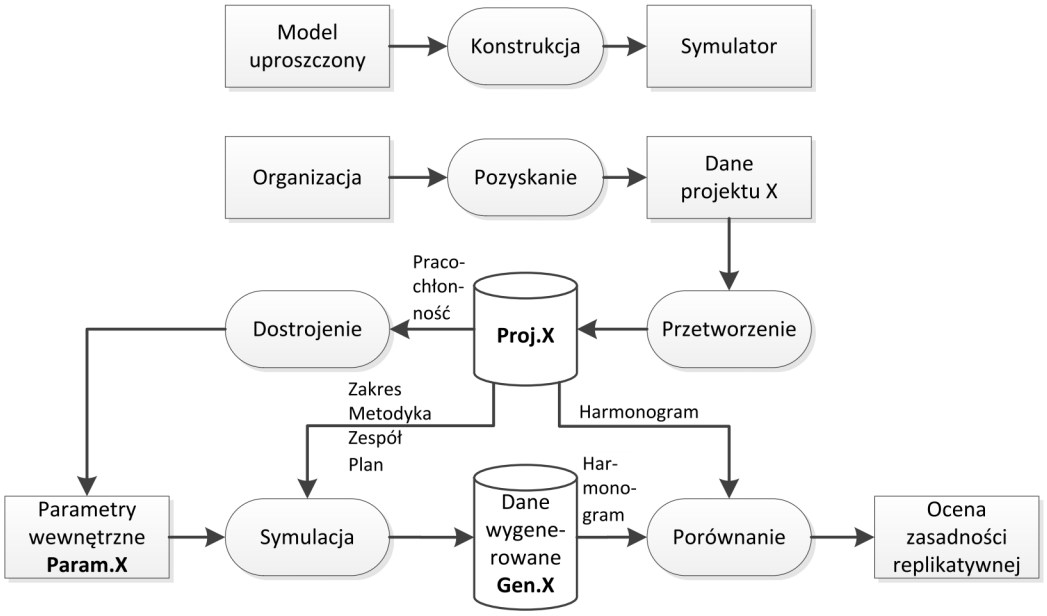
Po nakreśleniu koncepcji modelu, dostosowania go do przyjętego układu eksperymentu i wybrania metody, według której będzie konstruowany, należy rozważyć możliwości weryfikacji zasadności modelu uproszczonego. Zasadność (ang. *validity*) modelu jest podstawową relacją modelowania i odnosi się do relacji między systemem rzeczywistym, modelem i układem eksperymentu. Według sformułowania Zeiglera:

Definicja 3.1 Koncepcja zasadności odpowiada na pytanie, czy jest niemożliwe odróżnienie systemu rzeczywistego od modelu w przyjętym układzie eksperymentu [110].

Zasadność jest mierzona stopniem zgodności między danymi z systemu rzeczywistego a danymi generowanymi przez model. Istnieją stopnie wielkości tej zasadności. Najbardziej podstawowa koncepcja zasadności replikatywnej jest potwierdzona, jeżeli dla wszystkich eksperymentów, możliwych w przyjętym układzie eksperymentu, zachowanie modelu i systemu rzeczywistego jest zgodne w przyjętych granicach tolerancji. Od tego warunku silniejszy jest warunek, po spełnieniu którego model jest zasadny predykcyjnie. Następuje to wtedy, gdy można zapewnić zgodność obu grup przed uzyskaniem danych z systemu rzeczywistego (lub przynajmniej przed widzeniem ich przez model). Poniżej przedstawiono koncepcję weryfikacji replikatywnej oraz prognostycznej modelu uproszczonego.

3.5.1. Weryfikacja replikatywna

Konstruowany model jest modelem symulacyjnym, więc by zweryfikować jego zasadność należy najpierw zbudować symulator, który wykona instrukcje wymagane przez model. Użycie symulatora pozwoli na uzyskanie danych wygenerowanych przez model i porównanie ich z danymi rzeczywistymi. Pozyskanie danych dotyczących przebiegu procesu wytwarzania jest bardzo kłopotliwe. Organizacje zajmujące się wytwarzaniem oprogramowania strzegą tych informacji. Są to dane niejawne i zabezpieczone klauzulą poufności. Do zweryfikowania zasadności replikatywnej potrzebne są szczegółowe dane dotyczące projektu. Przede wszystkim są potrzebne dokładne wartości zmiennych wejściowych modelu, a więc: zakres projektu, metodyka wytwarzania, plan projektu i informacje o zespole projektowym. Potrzebne są również szczegółowe dane o przebiegu całego procesu wytwarzania oprogramowania. Niezbędna jest informacja, jakie zadania były wykonywane, przez kogo, kiedy i jak długo trwały. Wymagania dotyczące szczegółowości danych sprawiają, że pozyskane dane zawierają bardzo dużo informacji, która powinna być wstępnie przetworzona, dostosowana i umieszczona w bazie danych. Następnym krokiem jest dostrojenie parametrów wewnętrznych symulatora do danych projektu rzeczywistego. Dostrojony symulator generuje dane, które po przetworzeniu mogą być porównane z danymi z systemu rzeczywistego. Na podstawie porównania przeprowadzona zostanie ocena zasadności replikatywnej systemu. Schemat działań składających się na weryfikację zasadności replikatywnej modelu został przedstawiony na Rys. 3.6.



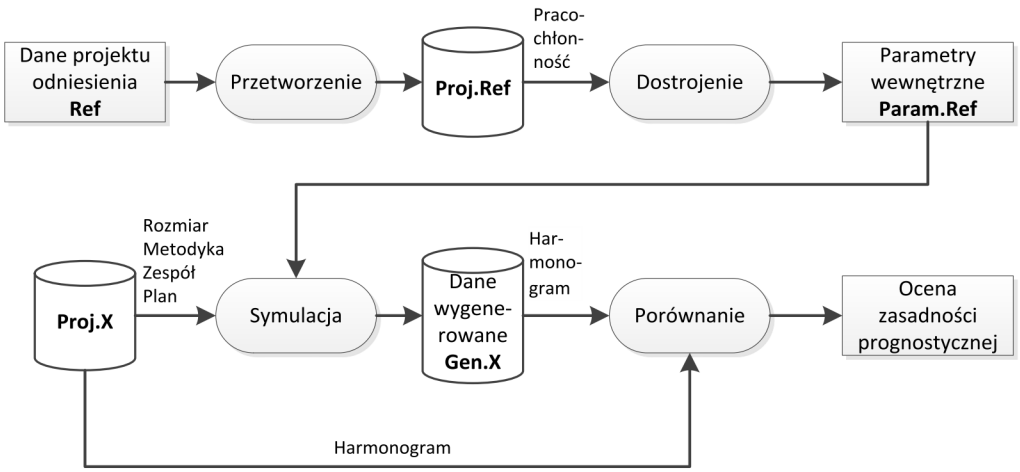
LEGENDA:
Projekt X – badany projekt,
Proj.X – baza danych prac badanego projektu,
Param.X – parametry wewnętrzne modelu dostrojone do badanego projektu,
Gen.X – wynik symulacji badanego projektu, wykonane prace i ich harmonogram

Rys. 3.6 Schemat działań celem oceny weryfikacji replikatywnej modelu
Źródło: Opracowanie własne

Na podstawie modelu uproszczonego budowany jest symulator modelu, następnie z organizacji zajmującej się wytwarzaniem oprogramowania zostają pozyskane dane projektu o nazwie X. Po przetworzeniu danych projektu zostaje utworzona jednolita baza danych Proj.X. która służy jako źródło danych w dalszych działaniach. Model zostaje dostrojony do danych z bazy Proj.X, następnie otrzymane wartości parametrów wewnętrznych oraz wartości zmiennych wejściowych są użyte do symulacji modelu i zostają wygenerowane dane harmonogramu Gen.X, które zostają porównane z rzeczywistym harmonogramem pochodzącym z bazy Proj.X. Na podstawie wyniku porównania zostanie przeprowadzona ocena zasadności replikatywnej modelu.

3.5.2. Weryfikacja prognostyczna

W celu weryfikacji zasadności predykcyjnej należy sprawdzić zgodność danych rzeczywistych projektu z danymi wygenerowanymi przez model bez wiedzy o systemie rzeczywistym. Wiedza o systemie rzeczywistym jest przechowywana jako wartość parametrów wewnętrznych modelu, odpowiadających za pracochłonność elementarnych typów zadań. Do wygenerowania danych przez model zostaną użyte parametry wewnętrzne otrzymane przez dostrojenie modelu do standardowego, referencyjnego projektu. Dzięki użyciu referencyjnych wartości parametrów wewnętrznych, mimo użycia wartości zmiennych wejściowych dobranych dla projektu X, można zaplanować proces wytwarzania projektu X i następnie na podstawie porównania planu procesu z rzeczywistym procesem ocenić zasadność prognostyczną modelu. Koncepcję przeprowadzenia weryfikacji prognostycznej przedstawiono w postaci graficznej na Rys. 3.7.



LEGENDA:
Ref – projekt odniesienia,
Proj.Ref – baza danych zadań projektu odniesienia,
Param.Ref – parametry wewnętrzne modelu dostrojone do projektu odniesienia,
Proj.X – planowany projekt,
Gen.X – wynik symulacji, harmonogram planowanego projektu.

Rys. 3.7 Koncepcja działań planowanych w celu weryfikacji prognostycznej modelu
Źródło: Opracowanie własne

Dane projektu referencyjnego Ref zostają wstępnie przetworzone, do jednolitej postaci bazy danych Proj.Ref, aby można było dostroić do niego model. Parametry wewnętrzne Param.Ref uzyskane w procesie dostrojenia modelu do projektu Ref oraz zmienne wejściowe modelu o wartościach odpowiadających badanemu projektowi Proj.X zostają użyte do przeprowadzenia symulacji, której wynikiem jest harmonogram Gen.X. Porównanie wygenerowanego harmonogramu Gen.X z danymi rzeczywistymi Proj.X pozwoli ocenić stopień zasadności prognostycznej modelu.

3.6. Podsumowanie

W rozdziale 3 przedstawiono model podstawowy, czyli pasujący do wielu różnych układów eksperymentu. Prace rozpoczęto od rozbudowy układu eksperymentu o elementy wewnętrzne, tworząc rdzeń modelu reprezentujący podstawy procesu wytwarzania oprogramowania. Na tym rdzeniu oparto rozszerzenia i podmodele związane z różnymi aspektami procesu wytwarzania skupionymi wokół zespołu projektowego, procesu wytwarzania i zapewniania jakości. Każdą propozycję rozbudowy opatrzono zestawem pytań, na które tak rozszerzony model mógłby udzielić odpowiedzi. Model podstawowy składa się z szerokiego wachlarza tematów i aspektów, które nie tylko były opisywane w jako koncepcje w artykułach, ale dla których sporządzano prototypowe rozwiązania w postaci arkuszy kalkulacyjnych lub kodu programu. Analiza dziedzinowa modelu podstawowego pozwoliła wyodrębnić te elementy modelu podstawowego, które są niezbędne do realizacji celu, dla osiągnięcia którego model jest budowany. W ten sposób powstał model uproszczony - wersja modelu podstawowego dostosowana do układu eksperymentu opisanego w rozdziale 2. Po przeprowadzeniu przeglądu metod modelowania i zastosowania ich do modelowania procesu wytwarzania oprogramowania zostały wy-

brane podejścia odpowiednie do realizacji przedstawionego modelu oraz zaprezentowano koncepcję budowy takiego modelu przy użyciu wybranych metod i podejść. Dla tak przedstawionej koncepcji modelu uproszczonego zbadano możliwości weryfikacji i zaproponowano przeprowadzenie weryfikacji zasadności replikatywnej oraz zasadności prognostycznej.

W następnym rozdziale model uproszczony zostanie opisany w sposób formalny i zostanie przedstawiona jego implementacja w systemie wieloagentowym.

4. Budowa uproszczonego modelu procesu wytwarzania oprogramowania

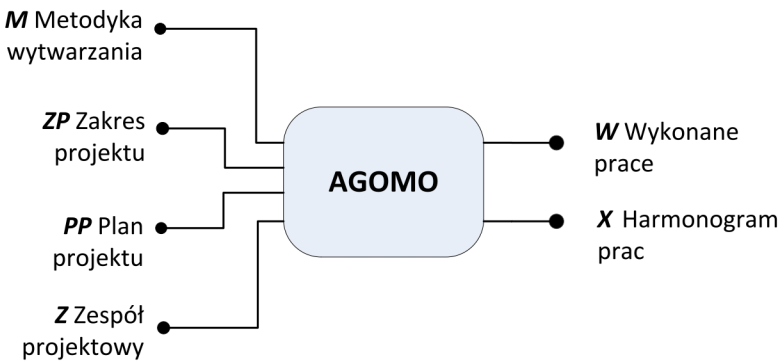
W poprzednim rozdziale opisano model podstawowy procesu wytwarzania oprogramowania, przedstawiono jego rdzeń i rozszerzenia reprezentujące wybrane aspekty procesu wytwarzania. Następnie, w celu realizacji, po analizie dziedzinowej i dostosowaniu modelu do układu eksperymentu, jako metodę modelowania wybrano podejście agentowo–obiektowe z dyskretną reprezentacją czasu. Opis formalny elementów modelu, w różnych wersjach, już został zamieszczony we wcześniejszych rozdziałach. Celem ujednolicenia, zwiększenia precyzji i czytelności pracy w bieżącym rozdziale zostanie przedstawiony w formie dostosowanej do podejścia agentowego. AGentowo–Obiektowy MOdel procesu wytwarzania oprogramowania (AGOMO) najpierw zostanie opisany na poziomie ogólnym jako czarna skrzynka, a następnie zostaną wyróżnione w nim bloki funkcjonalne. W dalszej części rozdziału zostanie przedstawiony szczegółowy opis, który omówi elementy budowy wewnętrznej i ich współpracę. Następnie zostanie przedstawiona metoda AGOMAS (ang. *AGent–Object Model software process ASsessment method*) przeznaczona do oceny procesów RUP organizacji, oparta na opracowanym modelu. Rozdział kończy się podsumowaniem prac i oceną modelu AGOMO.

4.1. Model procesu wytwarzania w widoku ogólnym

Widok ogólny modelu procesu wytwarzania zostanie przedstawiony w postaci czarnej skrzynki, czyli zostaną opisane zmienne wejściowe i wyjściowe modelu uproszczonego bez informacji budowie wewnętrznej. Następnie zostanie przedstawiona struktura wewnętrzna modelu w postaci bloków funkcjonalnych opisanych w Rozdziale 3 – rdzenia i połączonych z nim rozszerzeń. Szczegółowy opis elementów modelu zostanie przedstawiony w kolejnym podrozdziale.

4.1.1. Model uproszczony w postaci czarnej skrzynki

Model uproszczony w postaci czarnej skrzynki został przedstawiony na Rys. 4.1. Umieszczone na rysunku zmienne wejściowe i wyjściowe modelu są zgodne z opisanymi w poprzednich rozdziałach układem eksperymentu oraz modelem podstawowym. Zakładamy, że na wejścia modelu i systemu rzeczywistego (badanego procesu wytwarzania oprogramowania) podawane są te same wartości zmiennych wejściowych. Wprowadzono jednak ograniczenia w strukturę zmiennych wejściowych, spowodowane decyzją o wspieraniu przez model tylko jednej, tradycyjnej metodyki wytwarzania Rational Unified Process (RUP). Poniżej został zamieszczony opis zmiennych modelu AGOMO.



Rys. 4.1 Model AGOMO w postaci czarnej skrzynki
Źródło: Opracowanie własne

Zmienna wejściowa metodyka wytwarzania

Metodyka wytwarzania wykorzystywana w modelowanym procesie jest ustalona przez zmienną wejściową M, której wartość stanowi opis standardowej metodyki RUP, sporządzony na podstawie literatury [58, 59] oraz dokumentacji dostarczanej z oprogramowaniem Rational Method Composer produkcji IBM [114].

$$M = (D, E, K, O, R, TZ, A) \tag{4.1}$$

gdzie:
D – zbiór dyscyplin metodyki,
E – zbiór etapów metodyki,
K – zbiór kategorii działań metodyki,
O – zbiór działań metodyki,
R – zbiór ról projektowych metodyki,
TZ – zbiór elementarnych typów zadań metodyki,
A – zbiór typów artefaktów metodyki.

Zmienna M i jej składowe zostały szczegółowo opisane w podrozdziale 2.3.2, równania (2.2–2.5). Zbiór elementarnych typów metodyki został opisany w podrozdziale 3.2.1, równanie (3.4).

Zmienna wejściowa zakres projektu

Natomiast zakres projektu, którego proces wytwarzania jest modelowany, ustala zmienna wejściowa ZP. Jej wartość określa rozmiar projektu oszacowany w punktach przypadków użycia AGOMO–UCP. Metoda szacowania AGOMO–UCP została opracowana na podstawie metody szacowania UCP, przez dostosowanie jej do potrzeb modelu AGOMO. Metoda szacowania zakresu projektu UCP jest historycznie i pojęciowo bliska metodyce RUP przez powiązanie z przypadkami użycia, które są częścią technik modelowania UML, stosowanych łącznie z metodyką RUP. Opis metody AGOMO–UCP znajduje się w Dodatku A do rozprawy.

Określenie zakresu projektu powinno być na tyle uniwersalne, aby umożliwiać modelowanie procesu wytwarzania zarówno nowych projektów – tworzonych od podstaw, jak też projektów polegających na rozbudowie istniejącego systemu oraz projektów tworzonych na podstawie systemów poprzedniej generacji. W tym celu wartość zmiennej wejściowej określającej zakres projektu ZP została wyrażona w postaci pary liczb:

$$ZP = (zn, zo) \tag{4.2}$$

gdzie:
zn – zakres nowej części projektu w punktach AGOMO–UCP, wymagania do pozyskania w trakcie warsztatów i wywiadów z udziałowcami, $zn \in N$,
zo – zakres projektu w punktach AGOMO–UCP do odzyskania z poprzedniej wersji systemu, wymagania odzyskiwane z oprogramowania odziedziczonego, $zo \in N$.

Zmienna wejściowa plan projektu

Kolejna zmienna wejściowa modelu pozwala określić plan projektu, którego proces wytwarzania ma zostać zamodelowany. Użyta jest do tego zmienna wejściowa PP, która pozwala ustalić procentowe cele do wykonania w każdej iteracji projektu, osobno dla każdej z ośmiu dyscyplin metodyki RUP.

$$PP = (I_{1,} \dots, I_{li}) \tag{4.3}$$

gdzie:
I_i – *i*–ta iteracja planu *PP,*
li – liczba iteracji planu projektu *PP.*

Iteracja składa się z określenia etapu i procentowego zakresu prac do wykonania.

$$I_i = (e_i, z_i) \tag{4.4}$$

gdzie:
e_i – etap *i*–tej iteracji planu *PP,* $e_i \in E$ metodyki wytwarzania M,
z_i – zbiór celów do zrealizowania w i–tej iteracji, określonych jako procent zakresu projektu.

$$z_i = \{zd_{i,d} | d \in D\} \tag{4.5}$$

gdzie:
$zd_{i,d}$ – wartość procentowa 0–100% dla dyscypliny d, odnosząca się do zakresu projektu ZP

Zmienna wejściowa zespół projektowy

Ostatnia zmienna wejściowa modelu – Z pozwala ustalić zespół projektowy składający się z osób wykonujących zadania składające się na modelowany proces wytwarzania oprogramowania.

$$Z = \{P_i | i = 1 \dots lp\} \tag{4.6}$$

gdzie:
Z – zespół projektowy,
P_i – *i*–ty członek zespołu projektowego Z
lp – liczba osób w zespole projektowym.

Kompetencje każdej z osób wchodzących w skład zespołu projektowego opisano za pomocą jednej lub więcej ról projektowych pochodzących z metodyki RUP.

$$P_i = \{r_{i,k} | k = 1 \dots lr_i\} \tag{4.7}$$

gdzie:
$r_{i,k}$ – k–ta rola projektowa i–tego członka zespołu projektowego, $r_{i,k} \in R$ zbioru ról projektowych metodyki wytwarzania M,
lr_i – liczba ról *i*–tej osoby zespołu projektowego Z.

Zmienne wyjściowe – wykonane prace i ich harmonogram

Zmienne wyjściowe modelu zawierają informacje o wykonanych pracach i ich harmonogramie. Zmienna opisująca wykonane prace jest zdefiniowana następująco:

$$W = \{tz_i | i = 1 \dots lws\} \tag{4.8}$$

gdzie:
o_i – typ i–tego zadania wykonanych prac na wyjściu modelu,
lws – liczba zadań na wyjściu modelu.

Z tak opisaną zmienną wyjściową związana jest druga, zawierająca harmonogram wykonanych prac na poziomie działań:

$$X = \{(t_i, d_i) | i = 1 \dots lws\} \tag{4.9}$$

gdzie:
X – harmonogram wykonanych prac na wyjściu modelu,
t_i – czas rozpoczęcia i–tego zadania w u.j.t,
d_i – czas trwania i–tego zadania w u.j.t.

4.1.2. Model uproszczony w postaci białej skrzynki

W poprzednim podrozdziale został przedstawiony model procesu wytwarzania w postaci wejść i wyjść. W tym podrozdziale przedstawiamy zarys struktury wnętrza modelu AGOMO, składający się z wybranych elementów modelu podstawowego, opisanych w rozdziale 3.

Rys. 4.2 Model AGOMO jako biała skrzynka
Źródło: Opracowanie własne

Rdzeń modelu uproszczonego ograniczony do RUP

W modelu zastosowano rdzeń modelu podstawowego opisany w podrozdziale 3.2.1. Algorytmy tam przedstawione zostały dostosowane do agentowej architektury modelu. Za wykonanie algorytmu realizacji planu projektu odpowiada wyspecjalizowany agent symulator. Natomiast za wykonanie algorytmów wypełniających zadania odpowiadają agenty reprezentujące członków zespołu projektowego. Funkcjonalność rdzenia została ograniczona do modelowania procesów wytwarzania realizowanych wg metodyki RUP. W wyniku tego ograniczenia plan projektu został podzielony na etapy i iteracje, dla których zakres prac jest określony przez procent zakresu projektu. Nie wprowadzono możliwości definiowania iteracji o ograniczonym czasie trwania oraz zdarzeń codziennych. Rdzeń modelu został również ograniczony do jednego zespołu projektowego, co jest spowodowane dostosowaniem do potrzeb modelowania metodyki RUP.

Rozszerzenia zastosowane w modelu uproszczonym

We wcześniej prezentowanym modelu podstawowym, w celu uzyskania jak najpełniejszego opisu procesu wytwarzania oprogramowania, zamieszczono rozszerzenia reprezentujące różne aspekty modelowanego procesu wytwarzania. W modelu uproszczonym, dostosowanym do oceny i planowania procesu wytwarzania, zastosowano rozszerzenia niezbędne do wygenerowania przez model wzorca procesu możliwego do porównania z badanym procesem. Na przykład, gdyby w modelu agenty pracowałyby bez dni wolnych, to prace w wygenerowanym przez model wzorcu procesu przebiegałyby w szybszym tempie niż w rzeczywistości. W takim przypadku, przy ocenie badanego procesu współczynnik zgodności byłby zaniżony. Implementację modelu AGOMO dla zwiększenia realizmu wyników zaopatrzono również w możliwość pracy grupowej, nakładanie prac w iteracji, zmiany zatrudnienia w zespole projektowym oraz generator świąt, urlopów i zwolnień, które zostaną omówione poniżej.

Podział zespołu projektowego na grupy

Rozszerzenie modelu o możliwość pracy w grupach pozwala podzielić wytwarzany projekt informatyczny na kilka podprojektów. Prace nad tymi podprojektami i prowadzenie w grupach, składających się z członków zespołu projektowego. Dzięki możliwości podziału zespołu projektowego na grupy, prace analityczne nad podprojektami mogą odbywać się równolegle. Każda osoba z zespołu projektowego może należeć do wielu grup. Każda grupa ma swojego lidera, osobę, która jest odpowiedzialna za organizowanie warsztatów wymagań.

Nakładanie prac w iteracji

Rozszerzenie umożliwiające nakładanie prac w iteracji zostało wprowadzone, aby zamodelować praktyki zwiększające efektywność procesu wytwarzania. Graf przepływu działań w etapie metodyki może zawierać działania wykonywane sekwencyjnie albo równolegle. Bez możliwości nakładania prac w działaniach wykonywanych sekwencyjnie, aby uruchomić prace w zakresie następnego działania, wszystkie prace poprzedniego działania musiały zostać ukończone. W rzeczywistości unika się tego rodzaju sztywnej realizacji metodyki RUP, ponieważ zmniejsza to efektywność procesu wytwarzania [3]. Rozwiązaniem jest stosowanie praktyk nazywanych nakładaniem działań w iteracji, która polega na tym, że kolejne działania startują po wykonaniu określonego procentu prac poprzedniego działania. Celem zwiększenia realizmu modelu zaimplementowano możliwość określenia progu procentowego ukończenia bieżącego działania, po którego osiągnięciu uruchamiane są kolejne działania sekwencyjne.

Zmiany w zespole projektowym

Rozszerzenie umożliwiające zmiany w zespole projektowym ma na celu odwzorowanie częstych sytuacji, gdy pracownicy zmieniają miejsce zatrudnienia. W poprzednim rozdziale opisano rozszerzenie w postaci rejestru zawierającego zmiany parametrów wejściowych modelu. W przypadku użycia modelu do oceny zakończonego procesu wytwarzania nie ma konieczności implementowania tego rejestru. W procesie wytwarzania oprogramowania jednak zespół projektowy ulega częstym zmianom. Powodem jest przechodzenie osób między projektami, zwalnianie i przyjmowanie nowych pracowników. W zrealizowanym modelu umożliwiono wskazanie okresów zatrudnienia osoby w projekcie i jej zaangażowanie w projekt jako liczbę godzin w tygodniu oraz możliwość wskazania, czy osoba pracuje w nadgodzinach.

Dni wolne od pracy

Rozszerzenie uwzględniające weekendy, święta, urlopy, nadgodziny zostało wprowadzone, aby można było porównać badany proces wytwarzania z zamodelowanym. Implementacja modelu AGOMO zawiera generator weekendów, świąt państwowych stałych i ruchomych, urlopów w zadanym wymiarze przyznawanych w standardowych okresach – wakacje, ferie, święta oraz generator sezonowych i losowych zwolnień z powodu choroby. Opracowany moduł umożliwia także wskazanie, które osoby mogą pracować w nadgodzinach, o ile zachodzi taka potrzeba. Nadgodziny są rozliczane w innych stawkach godzinowych niż zwykłe godziny pracy, co ma wpływ na koszty projektu.

4.2. Model procesu wytwarzania w widoku szczegółowym

Po opisaniu modelu AGOMO na poziomie ogólnym, w bieżącym rozdziale zostanie szczegółowo omówiona budowa wewnętrzna modelu złożonego ze agentów i środowiska agentowego.

Wewnątrz modelu dynamiczne elementy składowe reprezentowane są przez agenty umieszczone w środowisku agentowym, zawierającym pasywne obiekty, do których agenty mają dostęp i na których wykonują operacje składające się na odwzorowanie procesu wytwarzania oprogramowania. Agenty można przybliżyć jako autonomiczne programy komputerowe, działające niezależnie od siebie, mające możliwość komunikacji między sobą, badania środowiska, w którym się znajdują, oraz podejmowania decyzji i oddziaływania na środowisko, co zostało przedstawione schematycznie na Rys. 4.3.

Rys. 4.3 Agent obserwujący stan środowiska i generujący akcje, które na nie wpływają

Źródło: Opracowanie własne

W kolejnych podrozdziałach zostaną omówione elementy środowiska agentowego, typy agentów oraz współpraca między agentami różnych typów w modelowaniu procesu wytwarzania oprogramowania.

4.2.1. Środowisko agentowe i jego elementy

W środowisku agentowym zostały umieszczone zmienne wejściowe, parametry wewnętrzne oraz repozytorium i rejestr wykonanych zadań. Zawartość repozytorium są to zmienne stanu modelu, natomiast zawartość rejestru wykonanych zadań jest zmienną wyjściową modelu. Zatem w środowisku agentowym umieszczone są zmienne wejściowe i zmienna wyjściowa modelu oraz stałe w postaci parametrów modelu, wszystkie dostępne jako obiekty.

Zmienne wejściowe

Wszystkie zmienne wejściowe modelu są umieszczone w środowisku agentowym, a ich wartości są dla agentów dostępne jako obiekty.

Repozytorium

Repozytorium jest odpowiedzialne za przechowywanie instancji artefaktów wg typów, nadawanie artefaktom unikalnych identyfikatorów, tworzenie instancji artefaktów o podanym typie i wyszukiwanie instancji artefaktów wg podanego typu. Repozytorium przechowuje i udostępnia również informacje, czy dana instancja artefaktu została przetworzona przez zadanie określonego typu.

W modelu możliwość realizacji zadań elementarnych tz_i jest uwarunkowana dostępem do wymaganych artefaktów typu a_i. Typy i liczba dostępnych artefaktów w modelu jest określana w postaci repozytorium, definiowanym jako sekwencja:

$$RP = (A, \varphi, \omega, \gamma, \tau) \tag{4.10}$$

gdzie:
A – zbiór typów artefaktów środowiska projektowego,
$\varphi(a,t) = \varphi_{a,t}$ – funkcja określająca liczbę instancji artefaktów typu $a \in A$ w dyskretnej chwili czasu t,
$\omega(a, tz, t) = \omega_{a,tz,t}$ – funkcja wyszukująca instancji artefaktów typu $a \in A$ nieprzetworzonych przez zadania typu $tz \in T$ w dyskretnej chwili czasu t,
$\gamma(a,t) = \gamma_{a,t}$ – funkcja wyszukująca w repozytorium instancji artefaktu typu $a \in A$ i tworząca instancję, jeśli nie została znaleziona,
$\tau(a,t) = \tau_{a,t}$ – funkcja tworząca w repozytorium instancję artefaktu typu $a \in A$.

Zadania produkujące instancje artefaktów nadają im unikalne identyfikatory i umieszczają utworzone instancje w repozytorium, gdzie są dostępne dla innych zadań i po wyszukaniu stają się ich artefaktami wejściowymi.

Rejestr wykonanych zadań

Rejestr wykonanych zadań jest odpowiedzialny za przechowywanie informacji o tym kto, kiedy i jak długo pracował nad zadaniem podanego typu.

$$RZ = (TZ, \rho, \sigma) \tag{4.11}$$

gdzie:
RZ – rejestr wykonanych zadań,
TZ – zbiór typów zadań metodyki,
$\rho(p, tz, t, d) = \rho_{p,tz,t,d}$ – funkcja zapisująca w rejestrze RZ: p – członka zespołu projektowego, tz – typ wykonanego zadania, t – czas rozpoczęcia wykonywania zadania, d – czas trwania zadania,
σ – funkcja odczytująca z rejestru zbiór sekwencji zawierających zapisaną informację w postaci $\{(p, tz, t, d)\}$.

Rejestr wykonanych działań

Rejestr wykonanych działań przechowuje informacje zmianach iteracji i bieżących działań, które są przełączane przez agenta symulatora. Informacja z rejestru działań udostępniania jest na wyjściu modelu w postaci zmiennych W^S i X^S.

$$RO = (O, \mu, \pi) \tag{4.12}$$

gdzie:
RRO – rejestr wykonanych działań,
O – zbiór działań metodyki,
$\mu(i,o,t) = \mu_{i,o,t}$ – funkcja zapisująca w rejestrze RO: i – iterację, o – działanie, t – czas rozpoczęcia działania,
π – funkcja odczytująca z rejestru zbiór sekwencji zawierających zapisaną informację w postaci $\{(i,o,t)\}$.

Parametry wewnętrzne modelu
Parametry wewnętrzne modelu zawierają stałe, których wartości określają czas przetworzenia jednego artefaktu wejściowego przez typy zadań elementarnych lub czas produkowania artefaktu wyjściowego dla typów zadań grupowych.

4.2.2. Agenty

W modelu występują trzy typy agentów współpracujących ze sobą w odtworzeniu procesu wytwarzania oprogramowania. Agenty osoby wykonują zadania członków zespołu projektowego, agent symulator kontroluje stan prac i zarządza realizacją planu projektu, a agent zegar symulacji synchronizuje prace wszystkich agentów i zachowań w systemie.

Agenty reprezentujące osoby z zespołu projektowego
Osoby wchodzące w skład zespołu projektowego w modelu AGOMO zostały odwzorowane przy pomocy agentów, nazwanych w skrócie „agent osoba". Jednej osobie odpowiada jeden agent osoba, który posiadając informację o przypisanych rolach projektowych, wykonuje zadania członka zespołu projektowego przy pomocy mechanizmu zachowań. Na początku symulacji dla każdej osoby z zespołu projektowego tworzony jest agent osoba z odpowiednimi zachowaniami.

Model zespołu projektowego jest zbiorem agentów reprezentujących osoby z zespołu projektowego:

$$G = \{g_i | i = 1 \dots lp\} \tag{4.13}$$

gdzie:
G – zbiór agentów,
g_i – i–ty agent zbioru G,
lp – liczba osób w zespole projektowym.

Agent osoba jest scharakteryzowany przez sekwencję, która określa osobę z zespołu projektowego i zbiór zachowań agenta:

$$g_i = \left(P_i, \{b_{i,k} | k = 1 \dots \mathrm{lgb_i}\}\right) \tag{4.14}$$

gdzie:
g_i – i–ty agent zbioru G,
P_i – i–ta osoba w zespole projektowym,
$b_{i,k}$ – k–te zachowanie i–tego agenta,
lgb_i – liczba zachowań i–tego agenta.

W skład zespołu projektowego wchodzą osoby aktywnie wykonujące zadania i osoby pasywne – pomagające innym w wykonywaniu przez nich zadań. Pasywni uczestnicy projektu to udziałowcy z organizacji zamawiającej system informatyczny, czyli zarząd, kierownicy i zwykli użytkownicy. Uczestniczą oni w różnych formach działań mających na celu zebranie wymagań dla projektu informatycznego. Celem tej grupy jest pomaganie osobom aktywnym w wykonywaniu ich zadań.

Głównym celem aktywnych uczestników projektu, takim jak analitycy systemowi, architekci, projektanci, programiści i testerzy jest aktywne wyszukiwanie zadań do wykonania i wykonywanie ich zgodnie z przyjętą metodą wytwarzania oprogramowania.

Agenty wymieniają informacje przy pomocy komunikatów, natomiast swoje cele realizują używając mechanizmu zachowań. Agent może mieć wiele zachowań aktywnych równocześnie. Każde z zachowań agenta jest osobnym automatem skończonym. Stan agenta jest określony przez zbiór stanów jego zachowań. Zachowanie agenta jest automatem skończonym (ang. *finite state automaton*) zdefiniowanym przez uporządkowaną piątkę:

$$b = (S, s_0, L, T, F) \tag{4.15}$$

gdzie:
b – zachowanie agenta,
S – jest skończonym zbiorem nazywanym stanem,
s_0 – jest stanem początkowym, $s_0 \in S$;
L – jest skończonym zbiorem etykiet,
T – jest zbiorem przejść, $T \subseteq (S \times L \times S)$,
F – jest zbiorem stanów końcowych, $F \subseteq S$.

Cel agentów polegający na współpracy i pomaganiu innym agentom został zrealizowany przez zachowanie *HelperBehaviour*, które jest przypisane wszystkim (aktywnym i pasywnym) agentom – uczestnikom projektu.

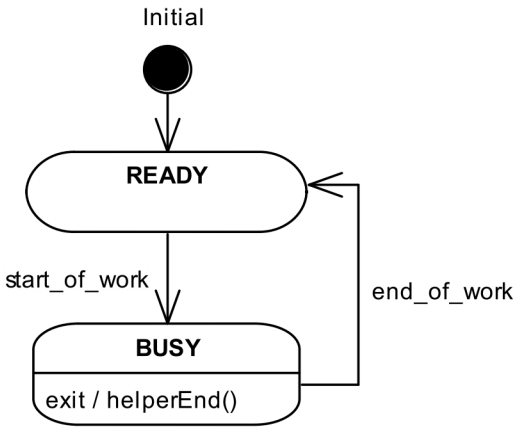
Cel aktywnych uczestników projektu – wyszukiwanie i wykonywanie zadań są realizowane przez zachowanie *TaskManagerBehaviour*.

Zachowanie HelperBehaviour

Wszyscy uczestnicy projektu mają uruchomione zachowanie *HelperBehaviour*, odpowiedzialne za współpracę z innymi agentami i wspólne wykonywanie zadań.

Zachowanie *HelperBehaviour* jest dwustanowym automatem o stanach READY i BUSY, którego diagram stanów pokazano na Rys. 4.4. W początkowym stanie READY zachowanie oczekuje na propozycję współpracy od agenta potrzebującego pomocy, nazwanego agentem głównym. Jeżeli uda się nawiązać współpracę, co jest sygnalizowane etykietą **start_of_work**, to zachowanie przechodzi w stan BUSY, w którym oczekuje na informację od agenta głównego o zakończeniu wykonywania wspólnego zadania. Agent główny po zakończeniu wykonywania zadania zapisuje informacje o typie i czasie zadania w rejestrze wykonanych zadań RZ. Następnie wysyła do agentów pomocniczych komunikat o zakończeniu współpracy endOfHelp(). Agenty pomocnicze po odebraniu komunikatu również zapisują zadanie, nad którym pracowały, w rejestrze RZ. W zachowaniu *HelperBehaviour* agenta pomocniczego odebranie komu-

nikatu endOfHelp() jest sygnalizowane etykietą **end_of_work**, powodującą przejście zachowania w stan READY. Wyjście ze stanu READY powoduje wysłanie komunikatu helperEnd(), co przełącza zachowanie *TaskManagerBehaviour* ze stanu HELP na FREE.



Rys. 4.4 Diagram stanów automatu *HelperBehaviour*
Źródło: Opracowanie własne

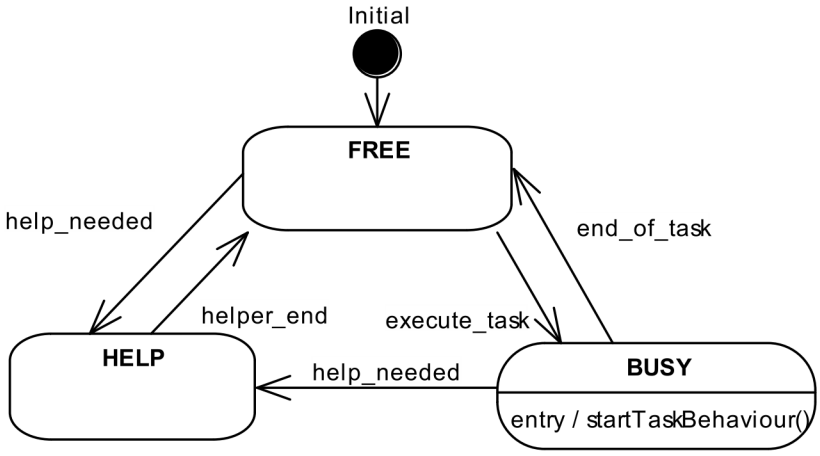
Zachowanie TaskManagerBehaviour

Aktywni uczestnicy projektu należący do zespołu projektowego mają uruchomione dwa zachowania: *HelperBehaviour* i *TaskManagerBehaviour*. To drugie jest odpowiedzialne za wyszukanie zadań możliwych do wykonania przez agenta i uruchomienie odpowiedniego zachowania wykonującego wybrane zadanie.

Zachowanie *TaskManagerBehaviour* jest trzystanowym automatem o stanach FREE, HELP i BUSY, którego diagram stanów pokazano na Rys. 4.5. W początkowym stanie FREE wykonywane jest sprawdzenie, czy inny agent wymaga pomocy w wykonaniu zadania. Jeżeli tak, to zachowanie przechodzi w stan HELP i pozostaje w nim, dopóki zachowanie *HelperBehaviour* współpracuje z agentem głównym. Zakończenie współpracy oznaczone jest etykietą **helper_end**. Następnie zachowanie na podstawie ról projektowych pełnionych przez uczestnika wyszukuje typu zadania do wykonania. Zachowanie po kolei sprawdza, czy typ zadania można uruchomić i jeśli można, to je uruchamia. Zadanie można uruchomić, jeśli spełnione są trzy warunki:

- istnieją nieprzetworzone przez zadanie artefakty wejściowe,
- cel iteracji dla dyscypliny, do której należy typ zadania, nie został zrealizowany,
- czas pozostały do końca dnia roboczego wystarczy do wykonania zadania, jeśli zadanie ma określony minimalny czas wykonania (np. Warsztaty wymagań).

Po znalezieniu typu zadania, które można wykonać – przejście oznaczone etykietą **execute_task**, zachowanie przechodzi w stan BUSY i uruchomiona zostaje metoda startTaskBehaviour(). Metoda tworzy zachowanie *TaskBehaviour* odpowiednie dla wykonywanego zadania. Nowe zachowanie zostaje dodane do zachowań agenta i uruchomione. Jeżeli w trakcie wykonywania zadania inny agent potrzebuje pomocy i zadanie może zostać przerwane, to wykonywanie zadania zostaje zakończone i zachowanie przechodzi ze stanu BUSY w stan HELP - przejście oznaczone etykietą **help_needed**.



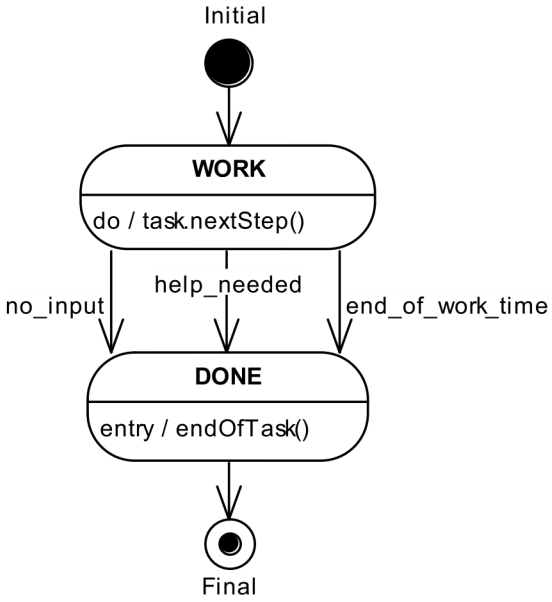
Rys. 4.5 Diagram stanów automatu *TaskManagerBehaviour*
Źródło: Opracowanie własne

Zachowanie TaskBehaviour

Zachowanie TaskBehaviour jest zachowaniem agenta odpowiedzialnym za wykonywanie zadań. Zachowanie jest dwustanowym automatem skończonym o stanach WORK i DONE, którego diagram stanów pokazano na Rys. 4.6. Stanem początkowym zachowania jest stan WORK, dla którego w każdym cyklu zegara symulacji uruchamiany jest kolejny krok wykonywanego zadania. Zachowanie przechodzi ze stanu WORK w stan DONE, gdy jest spełniony jeden z warunków:

- **is_not_needed** – zadanie zostało zakończone, czyli wszystkie artefakty wejściowe zostały przetworzone lub został zrealizowany cel iteracji,
- **help_needed** – rozpoczęła się współpraca z innym uczestnikiem projektu, a bieżące zadanie jest wykonywane samodzielnie,
- **end_of_work_time** – zakończył się dzienny czas pracy agenta.

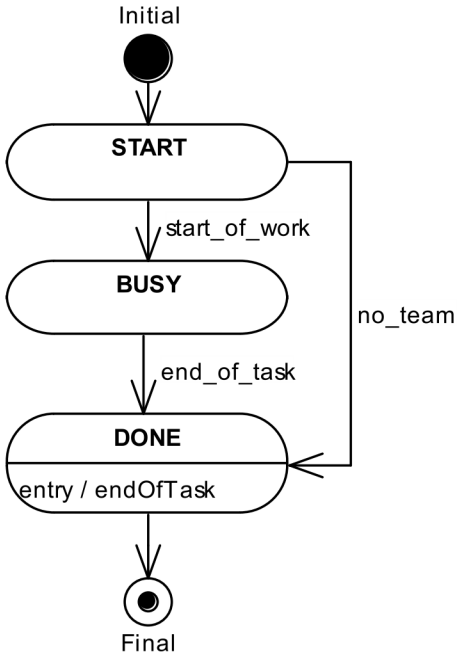
Przejście zachowania TaskBehaviour do stanu DONE, wysyła do agenta komunikat o zakończeniu zadania endOfTask(), który powoduje przełączenie zachowania *TaskManagerBehaviour* ze stanu BUSY w stan FREE. Jeżeli zadanie zostało zakończone z powodu rozpoczęcia współpracy z innym agentem, to zachowanie *TaskManagerBehaviour* przechodzi ze stanu BUSY w stan HELP. W końcowym stanie DONE, zachowanie jest usuwane ze zbioru aktywnych zachowań agenta.



Rys. 4.6 Diagram stanów automatu *TaskBehaviour*
Źródło: Opracowanie własne

Zachowanie WorkshopBehaviour

Zachowanie WorkshopBehaviour służy do uruchomienia zadania ElicitStakeholderRequestsOnWorkshop, którego celem jest zorganizowanie warsztatów wymagań, w skład których wchodzą agenty – aktywni uczestnicy projektu (analitycy systemowi, architekci, specyfikatorzy wymagań) i pasywni uczestnicy projektu (udziałowcy ze strony klienta, w tym użytkownicy powstającego systemu). Zachowanie jest trzystanowym automatem skończonym o stanach START, BUSY i DONE, którego diagram stanów pokazano na Rys. 4.7. W początkowym stanie START zachowanie zbiera informacje o agentach gotowych do współpracy przy zadaniu. Jeżeli liczba agentów pełniących wymagane role przekroczy określone minimum – przejście oznaczone etykietą **start_of_work**, to zachowanie przechodzi w stan BUSY, w którym pozostaje do czasu zakończenia warsztatów wymagań. Przejście zachowania *WorkshopBehaviour* w stan BUSY powoduje zmianę stanu zachowania *HelperBehaviour* współpracujących agentów pomocniczych na stan BUSY. W przypadku niezebrania zespołu warsztatowego – przejście oznaczone etykietą **no_team**, zachowanie przechodzi w stan DONE. Zakończenie zadania przeprowadzenia warsztatów również powoduje przejście zachowania w stan DONE. W stanie DONE zachowanie wysyła do agenta komunikat o zakończeniu wykonywania zadania endOfTask(), co przełącza zachowanie *TaskManagerBehaviour* w stan FREE. W końcowym stanie zachowanie jest usuwane ze zbioru aktywnych zachowań agenta.



Rys. 4.7 Diagram stanów automatu *WorkshopBehaviour*
Źródło: Opracowanie własne

Agent symulator zarządzający procesem wytwarzania

Agent symulator steruje wykonaniem planu projektu, jest odpowiedzialny za:

- zainicjowanie elementów środowiska agentowego i utworzenie agentów
- w trakcie trwania symulacji za przełączanie działań i iteracji w planie projektowym
- zapisywanie informacji o zmianach bieżących działań i iteracji w rejestrze wykonanych działań
- zakończenie symulacji, usunięcie agentów oraz zapisanie przebiegu i końcowego stanu symulacji

Agent zegar symulacji synchronizujący działanie agentów

Agent zegar symulacji odmierza czas symulacji i synchronizuje działanie wszystkich agentów w systemie. Standardowo jeden takt zegara symulacji odpowiada 15 minutom czasu projektu. Taka wartość została przyjęta, ponieważ najkrótszy czas pracy przeznaczony na wykonanie zadania trwa około kwadransa. Aby uniknąć wpływu chaosu wynikającego z nieprzewidywalnego porządku uruchamiania zachowań agentów, wynikającego z wielowątkowej pracy systemu wieloagentowego, wprowadzono podział jednego taktu zegara symulacji na cztery osobne fazy, co nie będzie szerzej omawiane w niniejszej rozprawie.

Współpraca między agentami

Na Rys. 4.8 przedstawiono uproszczony diagram komunikacji agentów i obiektów w modelu. Pominięto działanie zegara symulacji, ponieważ wprowadzenie szczegółów dotyczących taktów zegara uczyniłoby rysunek mniej czytelnym. Komunikacja między agentami a środowiskiem również została uproszczona. Nie uwzględniono współpracy agentów osób z innymi agentami i nie pokazano inicjowania i zamknięcia systemu przez agenta symulatora.

Działania agenta symulatora

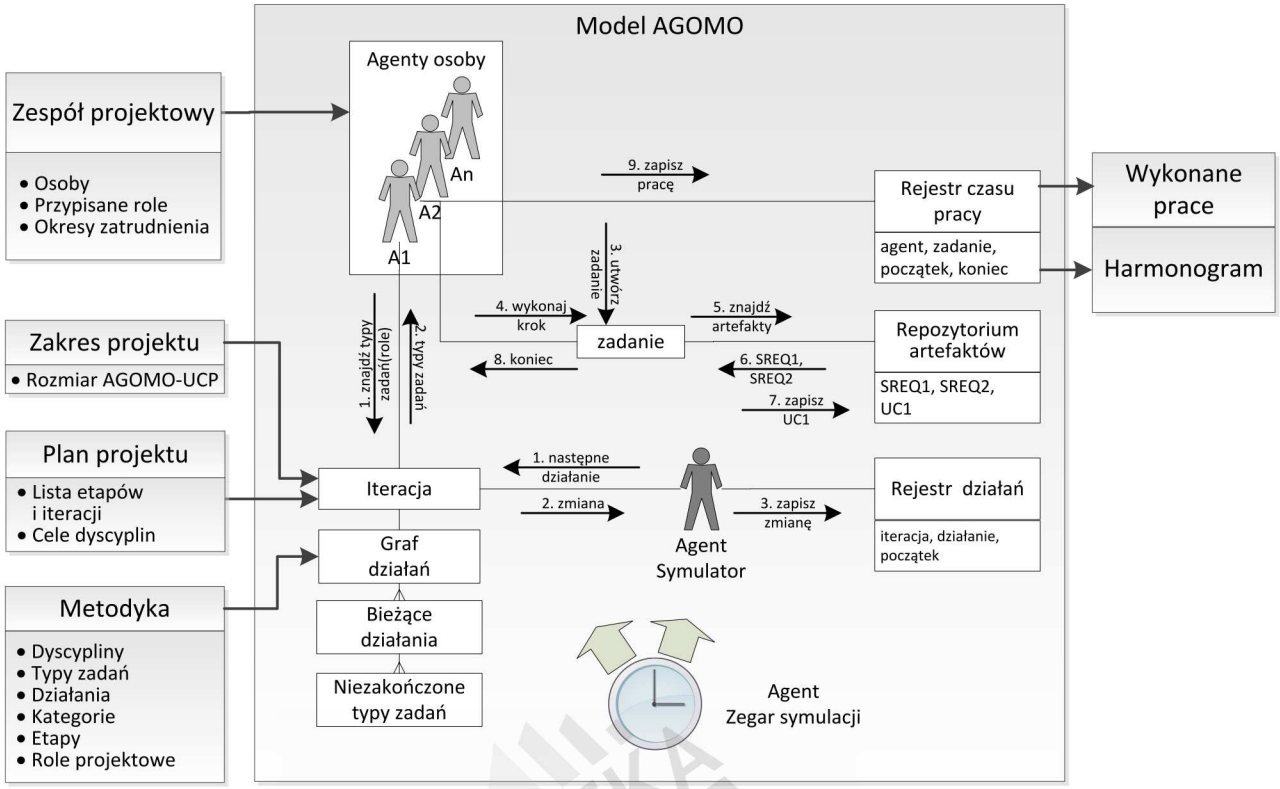
Agent Symulator wysyła do obiektu Iteracja komunikat *1. następna czynność*. Obiekt Iteracja przesyła komunikat dalej do bieżących działań przepływu. Działania przepływu sprawdzają, czy typy zadań, z których się składają zostały zakończone. Typ zadania jest zakończony, gdy wszystkie artefakty wejściowe z repozytorium zostaną przetworzone na artefakty wyjściowe. Kiedy wszystkie typy zadań działania są zakończone, to działanie również jest zakończone. Zakończone działania są usuwane z działań bieżących. W działaniach szeregowych oznacza to, że do bieżących działania zostaje dołączona następne działanie. Po zakończeniu działania lub dołączeniu nowego, obiekt Iteracja wysyła komunikat *2. zmiana* do agenta Symulatora. Agent Symulator wysyła komunikat *3. zapisz zmianę* do obiektu Rejestr zmian działania, który zapisuje zmianę działania.

Działania agenta osoby

Na diagramie komunikacji Rys. 4.8 dane wejściowe dotyczące zespołu projektowego są odwzorowane przez zespół agentów $g_1 \ldots g_{lp}$. Współpraca została przedstawiona dla przykładowego agenta g_1. Agent g_1 wyszukuje odpowiedniego dla swoich ról swoich ról w projekcie typów zadań, które należy wykonać w bieżącej iteracji. Wysyła w tym celu komunikat *1. znajdź typy zadań(role)* do obiektu Iteracja. Obiekt Iteracja wyszukuje wśród niezakończonych typów zadań takich, które mogą być wykonane przez podane role projektowe. Po znalezieniu typów zadań wysyła komunikat do agenta g_1 *2.typy zadań*. Agent g_1 wybiera spośród typów typ zadania do wykonania. Tworzy zadanie wybranego typu przez wysłanie komunikatu *3. utwórz zadanie*. Do utworzonego zadania wysyła komunikat *4. Wykonaj krok*. Zadanie po odebraniu komunikatu od agenta wyszukuje w repozytorium wejściowych artefaktów wysyłając komunikat *5. znajdź artefakty*. Repozytorium odpowiada komunikatem *6. SREQ1, SREQ2* wysyłając do zadania dwa znalezione artefakty wejściowe – żądania udziałowcy. Zadanie przetwarza w kolejnych krokach wykonania, nie pokazanych na diagramie, żądania udziałowcy na artefakt wyjściowy – przypadek użycia. Artefakt wyjściowy jest komunikatem *7. zapisz UC1* wysyłany do repozytorium, które go zapisuje. Po przetworzeniu wszystkich artefaktów wejściowych zadanie jest zakończone, o czym informuje agenta $g_1$wysłaniem komunikatu *8. koniec*. Agent g_1 wysyła do rejestru czasu pracy komunikat *9. zapisz pracę*, co powoduje zapisanie informacji o wykonaniu przez agenta g_1 zadania wybranego typu, które trwało od taktu n, do taktu n + czas_wykonania zegara symulacji.

Działania agenta zegar symulacji

Zegar symulacji wysyła komunikaty o kolejnych taktach zegara symulacji do wszystkich agentów w systemie.



Rys. 4.8 Diagram komunikacji agentów i obiektów modelu AGOMO

Źródło: Opracowanie własne

4.3. Implementacja modelu AGOMO w systemie wieloagentowym

Dla implementacji systemu agentowo – obiektowego wybrano JADE – napisany w języku Java system wieloagentowy typu open source na licencji GNU. Powodem wyboru właśnie tego, spośród wielu systemów wieloagentowych, były następujące cechy:

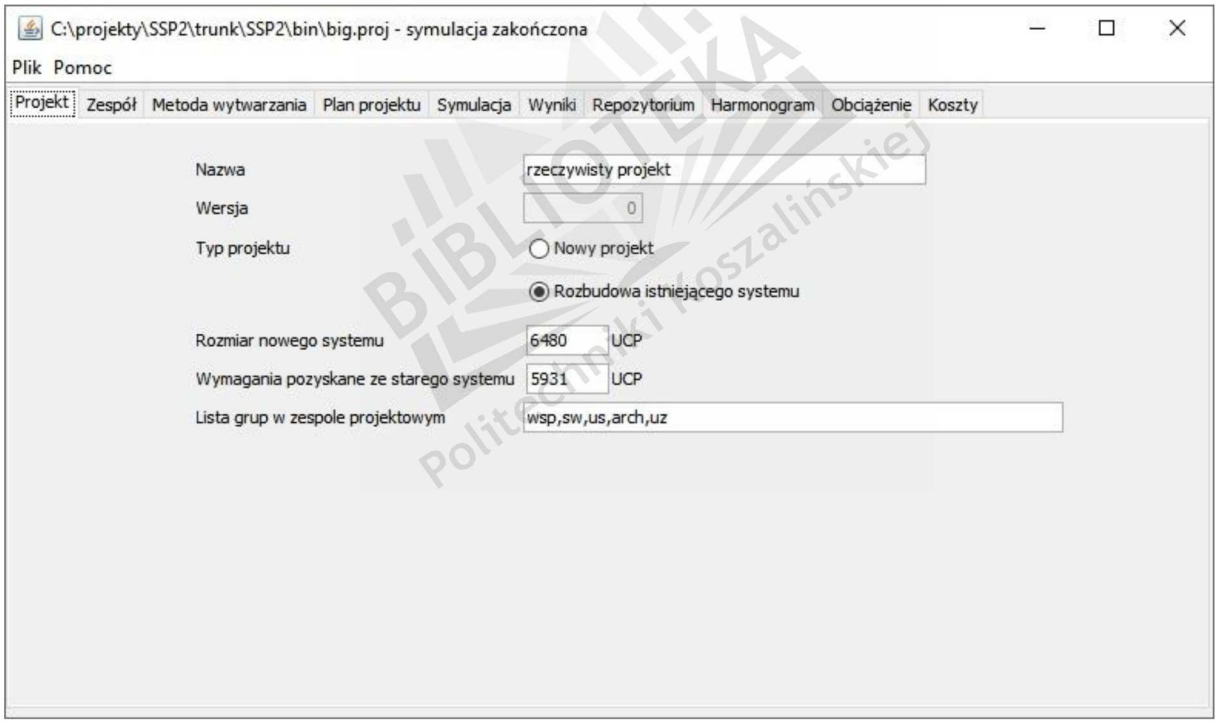
- współdziałanie – Jade jest zgodne ze standardami FIPA, agenty systemu Jade mogą komunikować z agentami innych systemów zgodnymi ze standardami [120],
- jednolitość i przenośność – Jade dostarcza jednolitego interfejsu API bez względu na rodzaj sieci i wersję środowiska Java,
- łatwość użycia – złożone mechanizmy środowiska rozproszonego są dostępne przez prosty i intuicyjny interfejs programisty API,
- płacisz, gdy używasz – programiści nie muszą znać wszystkich cech środowiska rozproszonego, mogą wykorzystywać tylko jego część i tylko ta część będzie obciążać obliczeniowo system,
- możliwość integracji z regułowym silnikiem decyzyjnym Jess [33, 116].

Istotny wpływ na wybranie systemu Jade miało również to, że system jest aktywnie rozwijany przez firmę Telecom Italia i oraz międzynarodową społeczność [115, 120], co zapewnia czynne wsparcie ze strony deweloperów, gwarantuje prace nad dalszym rozwojem systemu i dostarcza wielu źródeł informacji, zarówno o samym systemie, jak i sposobach radzenia sobie z najczęściej występującymi problemami.

System wieloagentowy musiał zostać dostosowany do użycia jako środowisko uruchomieniowe symulacji wieloagentowej (MABS) [103]. W systemie JADE brakuje rozwiązania problemu

czasu symulacji. Wobec tego zastosowano autorskie rozwiązanie centralnego odmierzania czasu symulacji o stałym kroku, będące rozszerzeniem koncepcji zegara symulacji pakietu Jadex [14]. Zegar symulacji został rozszerzony o możliwość ustalenia kolejności wysyłania informacji o czasie do grup agentów, dzięki wprowadzeniu wielofazowego taktu zegara. To rozwiązanie pozwoliło zmniejszyć entropię i uzyskać lepszą powtarzalność wyników symulacji procesów RUP.

Symulator modelu AGOSIM–APP został wyposażony w graficzny interfejs użytkownika zbudowany w oparciu o środowisko Java [89], biblioteki Swing [26, 83] i jfreechart [117]. System pozwala na określenie wartości zmiennych wejściowych modelu, uruchomienie symulacji, obejrzenie wyników symulacji w postaci harmonogramu wykonanych prac, zawartości repozytorium, raportu obciążenia zadaniami zespołu projektowego i raportu kosztów projektu. Dane wprowadzone do aplikacji i wyniki symulacji można zapisać do pliku projektu, który może być odczytany przez symulator lub narzędzia wspierające metodę AGOMAS do oceny procesu wytwarzania. Wygląd okna aplikacji został pokazany na Rys. 4.9. Wartości zmiennych wejściowych i wyniki symulacji zostały umieszczone w osobnych zakładkach okna aplikacji. Aplikacja została opisana szczegółowo w dodatku B.

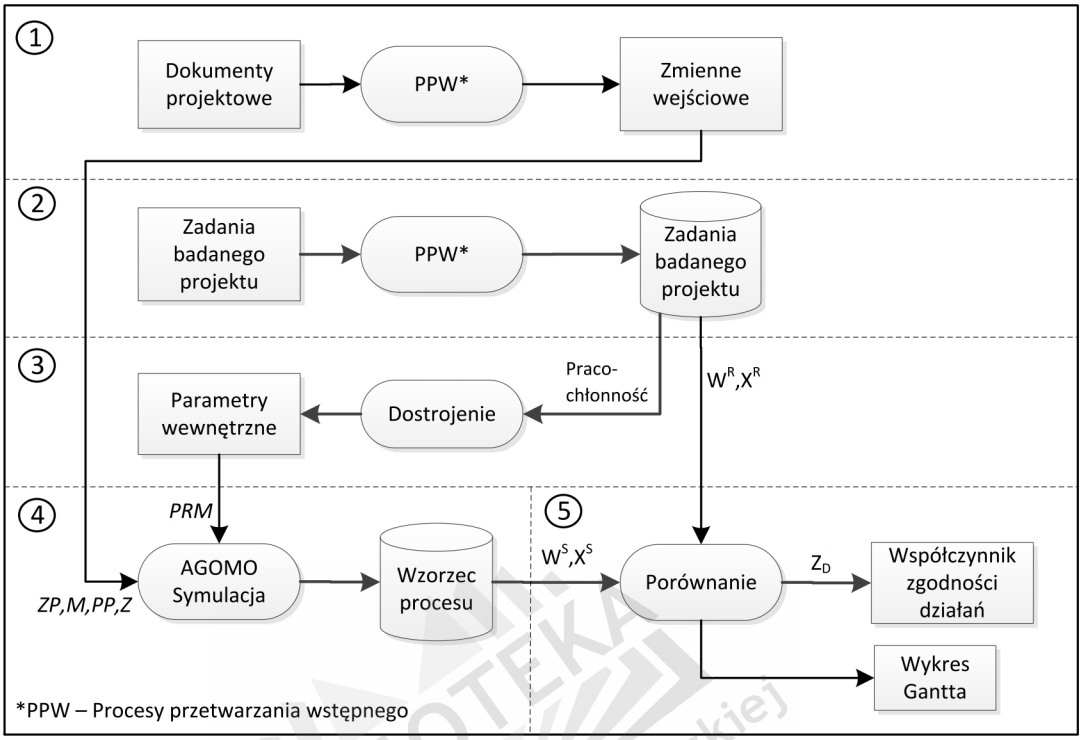


Rys. 4.9 Interfejs graficzny systemu AGOSIM–APP, symulatora modelu AGOMO
Źródło: Opracowanie własne

4.4. Metoda oceny procesów RUP

Ocena procesów wytwarzania oprogramowania jest podstawowym zastosowaniem modelu AGOMO. Polega na porównaniu zadań badanego procesu wytwarzania z wzorcowym harmonogramem wygenerowanym przez model na podstawie opisu środowiska projektowego: składu zespołu, zakresu i planu projektu oraz metodyki wytwarzania oprogramowania. Wynik porównania jest przedstawiony w postaci procentowej, pozwalającej ocenić proces, co pozwala wnioskować o stanie gotowości organizacji do zmiany wykorzystywanej metodyki wytwarzania.

Koncepcja zastosowania modelu AGOMO do oceny procesu wytwarzania oprogramowania zgodnego z metodyką RUP została pokazane na Rys. 4.10.



Rys. 4.10 Etapy metody AGOMAS służącej do oceny procesu RUP przy użyciu modelu AGOMO

Źródło: Opracowanie własne

Metoda oceny dojrzałości AGent–Object Model ASsessment method (AGOMAS) składa się z następujących etapów, pokazanych na Rys. 4.10, które zostaną opisane szczegółowo w kolejnych podrozdziałach:

1. Ustalenie wartości zmiennych wejściowych: zakres projektu, plan projektu, konfiguracji metodyki RUP, skład zespołu projektowego i przypisane role na podstawie dokumentów badanego projektu.
2. Przetworzenie informacji o zadaniach wykonanych w badanym projekcie na jednolitą bazę danych zawierającą: kategorię, wykonawcę, czas rozpoczęcia i czas trwania zadania.
3. Skalibrowanie modelu: obliczenie wartości parametrów wewnętrznych modelu na podstawie pracochłonności badanego projektu.
4. Wykonanie symulacji procesu wytwarzania i zapisanie wyniku jako wzorca procesu.
5. Porównanie zadań badanego procesu z wzorcem procesu i obliczenie wartości wskaźnika zgodności działań. Przedstawienie wyniku porównania w postaci diagramu Gantta.

Metoda AGOMAS może być stosowana dla projektów wytwarzanych zgodnie z procesem RUP. Istnieje możliwość zastosowania metody AGOMAS dla projektów z dowolną metodyką wspieraną przez model AGOMO, jednak nie zostało to zweryfikowane w niniejszej rozprawie.

Metoda AGOMAS jest dostosowana do oceny projektów informatycznych o średnim i dużym rozmiarze, co wynika z rozbudowanej wersji metodyki RUP wbudowanej w model AGOMO. Opis metodyki RUP może być dowolnie dostosowany do projektu, ponieważ jest to zmienna wejściowa modelu AGOMO. Dotychczasowe prace nad weryfikacją polegały na symulacji dużych projektów informatycznych i do nich dostosowano metodykę RUP zastosowaną w modelu AGOMO.

4.4.1. Etap pierwszy – określenie wartości zmiennych wejściowych modelu

Do określenia wartości zmiennych wejściowych modelu możliwe są dwa podejścia: pierwsze polega na analizie dokumentów z obszaru zarządzania powstałych przed rozpoczęciem lub na początku procesu wytwarzania oprogramowania, drugie na analizie danych z zakończonego procesu wytwarzania. W praktyce zastosowano obydwa podejścia w różnych proporcjach, zależnie od zmiennych wejściowych, co zostanie przedstawione poniżej.

Wartość zmiennej wejściowej ZP – zakres projektu

Głównym źródłem informacji o zakresie projektu jest dokument oszacowania rozmiaru projektu, który jest zwykle dołączony do umowy między organizacją wytwarzającą oprogramowanie a organizacją je zamawiającą. Jeśli nie ma dostępu do tego dokumentu, np. z powodu niejawności umów handlowych, to można posłużyć się dokumentem wizji systemu lub dokumentami z późniejszego okresu, powstałymi w trakcie trwania procesu wytwarzania. Takim dokumentem jest model przypadków użycia, na podstawie którego można w dosyć łatwy sposób obliczyć rozmiar oprogramowania w punktach przypadków użycia AGOMO–UCP (patrz Dodatek A. Rozmiar oprogramowania). Podejście oparte na wykorzystaniu dokumentów lub innych artefaktów wytworzonych w procesie wytwarzania ma tą zaletę, że jest ostateczną i niezmienną wartością, natomiast wstępne oszacowania są często zmieniane i korygowane. W modelu nie zaimplementowano rejestru parametrów wejściowych, który umożliwiałby zamodelowanie tych korekt i ich wpływu na proces wytwarzania, wobec czego preferowanym podejściem jest oparcie wartości zmiennych wejściowych na analizie artefaktów.

Wartość zmiennej wejściowej M – metodyka wytwarzania

Po zawężeniu akceptowanych przez model metodyk wytwarzania do jednej, wartość zmiennej metodyka wytwarzania została opracowana w wersji dla projektów średniego rozmiaru i powinna być dostosowana do specyfiki organizacji i projektu. Informacje dotyczące metodyki stosowanej przez organizację można odnaleźć w opisach procedur wytwarzania oprogramowania.

Wartość zmiennej wejściowej PP – plan projektu

Wartość zmiennej plan projektu ma również dwa źródła. Pierwsze źródło to odtworzenie oryginalnego planu projektu na podstawie dokumentów z zakresu zarządzania projektem. Drugim źródłem informacji o planie projektu jest analiza zapisów umieszczonych w Systemie Zarządzania Zmianą (SZZ). Analiza SZZ wymaga wykonania dodatkowej pracy, lecz otrzymujemy plan rzeczywiście zrealizowany, a nie tylko zakładany. SZZ może być zrealizowany za pomocą różnych systemów informatycznych, jak ClearCase, JIRA, Bugzilla i inne. Formaty zapisów informacji w tych systemach różnią się między sobą, jednak zwykle występuje tam data wykonania i iteracja, w ramach której zadanie jest zgłaszane i realizowane. Z tych danych można odtworzyć zrealizowany plan projektu.

Wartość zmiennej wejściowej ZP – zespół projektowy

Zespół projektowy jest zmienną wejściową modelu, której wartość często zmienia się w czasie. W tym przypadku najlepszym źródłem informacji jest analiza zadań wykonywanych w badanym projekcie przez członków zespołu projektowego. Uważna analiza powinna pozwolić na uzyskanie informacji nie tylko o okresach zatrudnienia poszczególnych osób, ale również o stopniu zaangażowania w projekt (0–100%) i przepracowanych nadgodzinach.

4.4.2. Etap drugi – przetworzenie informacji o wykonanych zadaniach

Dalszą częścią analizy wykonanych zadań na potrzeby ustalenia wartości zmiennych wejściowych jest przygotowanie danych do uzyskania niezbędnych informacji o badanym procesie. Na wyjściu badanego procesu otrzymujemy niepełne dane o wykonanych zadaniach, które trzeba wstępnie przetworzyć, aby z pewnym prawdopodobieństwem odtworzyć niezbędne informacje. Głównym problemem jest brak precyzyjnej informacji o wykonanym typie zadania. Istnieje jedynie opis zadania w postaci tekstu i na jego podstawie trzeba odtworzyć potrzebną informację. Zdecydowano, że na podstawie informacji uzyskanych z badanego projektu, to jest opisu tekstowego zadania oraz ról projektowych osób tworzących, wykonujących i weryfikujących zadanie, zostanie odtworzona informacja o kategorii zadania. Problemem mniejszej wagi jest brak informacji o przebiegu realizacji zadania. Znana jest tylko sumaryczna liczba godzin przeznaczona na wykonanie zadania. Do odtworzenia przebiegu zadania założono, że prace nad zadaniem rozpoczynają się, gdy pracownik zakończył inne zadania i trwają nieprzerwanie aż do przepracowania określonej liczby godzin. W procesie wytwarzania odtworzonym w ten sposób, osoba wykonuje zadania sekwencyjnie, jedno po drugim.

4.4.3. Etap trzeci – dostrojenie modelu do badanego projektu

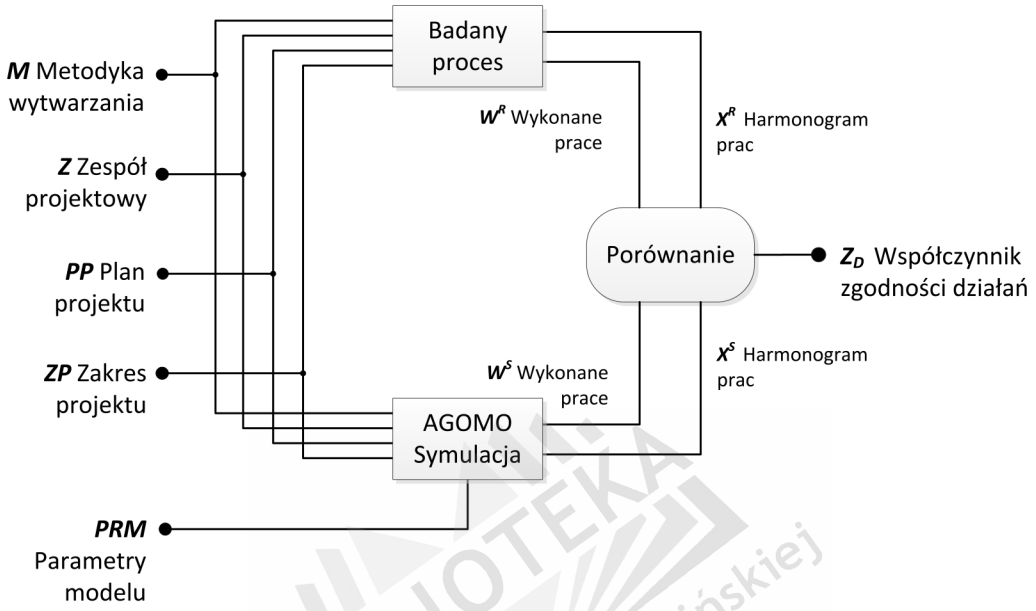
Dostrojenie modelu do badanego projektu polega na obliczeniu wartości parametrów wewnętrznych modelu tak, aby odtworzyć pracochłonność badanego projektu. Parametry wewnętrzne modelu określają czas potrzebny na przetworzenie jednego artefaktu w przypadku zadań przetwarzających artefakty elementarne lub czas wytworzenia jednego artefaktu grupowego dla zadań je wytwarzających. W modelu ograniczonym do metodyki Rational Unified Process występuje 77 typów zadań. Zmienna wejściowa metodyka definiuje 14 kategorii, dla których jest obliczana sumaryczna pracochłonność. Na podstawie pracochłonności zsumowanej według kategorii zadań należy wyznaczyć wartość parametrów wewnętrznych dla elementarnych typów zadań. Każda z kategorii ma przypisany zbiór typów zadań, z których się składa. Typy zadań mają różny wkład w sumaryczną pracochłonność kategorii, zatem zostały im przypisane wagi, zgodnie z którymi wartość pracochłonności jest rozdzielana.

4.4.4. Etap czwarty – symulacja procesu wytwarzania

Symulacja procesu wytwarzania przy wykorzystaniu modelu AGOMO odbywa się w dedykowanej aplikacji symulatora AGOSIM–APP, która umożliwia wprowadzenie wartości zmiennych wejściowych, przeprowadzenie symulacji, przedstawienie i zapisanie danych wynikowych oraz sporządzenie raportów i zestawień. Pliki projektu utworzone przez aplikację są odczytywane przez aplikację AGOCOMP–APP wspierającą metodę AGOMAS.

4.4.5. Etap piąty – ocena procesu wytwarzania oprogramowania

Ostatnim krokiem metody jest ocena procesu wytwarzania oprogramowania, polegająca na porównaniu działań wykonanych w badanym procesie wytwarzania z wzorcem procesu wygenerowanym przez model AGOMO. Wynikiem przeprowadzonej oceny jest procentowy wskaźnik zgodności działań.



Rys. 4.11 Zastosowanie modelu AGOMO do oceny dojrzałości procesów RUP organizacji
Źródło: Opracowanie własne

Porównanie badanego procesu z wzorcem

Na wyjściu badanego procesu znajdują się zmienne zawierające zbiór wykonanych prac i związany z nim harmonogram. Wykonane prace są opisane za pomocą kategorii zdefiniowanych przez metodykę wytwarzania jako zbiory działań (2.18). Kategorie grupują typy zadań z większą dokładnością niż dyscypliny. Ich liczba jest wynikiem kompromisu między potrzebą uzyskania jak najpełniejszej informacji o badanym procesie a możliwością jej uzyskania na podstawie niejednoznacznych opisów tekstowych wykonanych zadań. Zatem prace wykonane w badanym procesie W^R są przedstawione jako zbiór kategorii wykonanych zadań:

$$W^R = \{k_i^R | i = 1 \dots lwr\} \tag{4.16}$$

gdzie:
W^R – prace wykonane w badanym procesie,
k_i^R – kategoria i–tego zadania badanego procesu, $k_i^R \in K$,
lwr – liczba wykonanych zadań w badanym procesie.

Z wykonanymi pracami badanego projektu związany jest ich harmonogram:

$$X^R = \{(t_i^R, d_i^R) | i = 1 \dots lwr\} \tag{4.17}$$

gdzie:
X^R – harmonogram wykonanych prac,

t_i^R – czas rozpoczęcia i–tego zadania w u.j.t,
d_i^R – czas trwania i–tego zadania w u.j.t
lwr – liczba wykonanych prac w badanym procesie.

Porównywanie prac badanego procesu i modelu na poziomie zadań wykonywanych przez poszczególne osoby nie jest miarodajne dla oceny procesu wytwarzania, ponieważ jest to porównanie zbyt szczegółowe. Nawet zakładając, że badany proces jest całkowicie zgodny z metodyką wytwarzania, jest bardzo małe prawdopodobieństwo, że członek rzeczywistego zespołu projektowego w tym samym czasie wykona identyczne zadania, jak odpowiadający mu agent w modelu AGOMO. Zdecydowano zatem, że porównanie badanego procesu z wzorcem odbędzie się na wyższym poziomie, niż typy zadań i osoby, czyli na poziomie przepływu działań i ról projektowych. Wobec tego na wyjściu modelu typy zadań wykonanych prac są zagregowane do działań i zmienna wyjściowa jest zdefiniowana następująco:

$$W^S = \{o_i^S | i = 1 \dots lws\} \tag{4.18}$$

gdzie:
o_i^S – działanie i–tego zadania wykonanych prac na wyjściu modelu,
lws – liczba zadań na wyjściu modelu.

Z tak zapisaną zmienną wyjściową związana jest druga, zawierająca harmonogram wykonanych prac na poziomie działań:

$$X^S = \{(t_i^S, d_i^S) | i = 1 \dots lws\} \tag{4.19}$$

gdzie:
X^S – harmonogram wykonanych prac na wyjściu modelu,
t_i^R – czas rozpoczęcia i–tego działania w u.j.t,
d_i^R – czas trwania i–tego działania w u.j.t.

Wskaźnik oceny procesu i jego interpretacja

Wskaźniki oceny procesu powstają w wyniku porównania wartości przedstawionych w poprzednim punkcie zmiennych wyjściowych badanego procesu ze zmiennymi wyjściowymi modelu. Kategorie, za pomocą których opisane są prace wykonane w badanym procesie W^R można łatwo porównać z działaniami, za pomocą których opisane są prace wykonane w wirtualnym środowisku modelu W^S, ponieważ kategorie składają się z działań. Dla każdego zadania wykonanego w badanym procesie sprawdzane są działania wykonywane w odpowiadającym czasie w modelu. Jeżeli zbiór działań wykonywanych w modelu ma część wspólną ze zbiorem działań składających się na kategorie zadań wykonywanych w badanym procesie, to zadanie jest oceniane jako zgodne z metodyką wytwarzania i oznaczane przez ≡.

$$W_j^R \equiv W_i^S \tag{4.20}$$
$$jeżeli\ (\exists\ o_i^S \in W^S)\ \wedge\ (\exists k_j^R\ \in W^R),$$
$$takie, że\ (o_i^S \subset k_j^R) \wedge \left(\ t_j^R \leq (t_i^S + d_i^S)\right) \wedge (t_i^S \leq (t_j^R + d_j^R))$$

gdzie:
≡ – zgodność zadania badanego projektu z zadaniami otrzymanymi z modelu,
W_j^R – j–te zadanie badanego projektu,
W_i^S – i–te zadanie symulowanego projektu,

o_i^S – i–te działanie symulowanego projektu,
k_j^R – kategoria j–tego zadania badanego projektu,
t_j^R – czas rozpoczęcia j–tego zadania badanego projektu,
d_j^R – czas trwania j–tego zadania badanego projektu,
t_i^S – czas rozpoczęcia i–tego zadania symulowanego projektu,
d_i^R – czas rozpoczęcia i–tego zadania symulowanego projektu.

Stosunek procentowy liczby zadań zgodnych do liczby wszystkich zadań został nazwany wskaźnikiem zgodności działań.

$$Z_D = \frac{L_Z}{L_W} * 100\% \tag{4.21}$$

gdzie:
Z_D – wskaźnik zgodności działań,
L_Z – liczba zgodnych działań,
L_W – liczba wszystkich działań.

Zgodnie z koncepcją metody przedstawioną w podsumowaniu rozdziału 1 i przyjętymi tam założeniami, wartość wskaźnika zgodności może być interpretowana jako odpowiednik poziomu dojrzałości organizacji w modelu CMMI o wartości od 1 do 3, co może być związane z poziomem ryzyka zwinnej transformacji organizacji. Propozycja interpretacji wartości współczynnika Z_D została umieszczona w Tab. 4.1.

Tab. 4.1 Proponowana interpretacja wartości wskaźnika zgodności działań Z_D
Źródło: Opracowanie własne

Wartość Z_D	Poziom CMMI	Ryzyko transformacji
0–50%	Poziom 1 – początkowy	Bardzo wysokie
50%–75%	Poziom 2 – zarządzany	Średnie
powyżej 75%	Poziom 3 – zdefiniowany	Niskie

Narzędziem wspierającym metodę AGOMAS jest aplikacja AGOCOMP–APP, która porównuje przebieg badanego projektu z wzorcem procesu, na tej podstawie oblicza wskaźnik zgodności działań i przedstawia wynik porównania jako diagram Gantta. Interaktywny diagram Gantta umożliwia powiększanie wybranych obszarów, przesuwanie diagramu i wyświetlanie szczegółowych informacji na temat zadań podświetlonych przez kursor urządzenia wskazującego. Wygląd interfejsu graficznego aplikacji pokazano na Rys. 4.12. Aplikacja jest szczegółowo opisana w dodatku C (patrz Aplikacja AGOCOMP–APP).

Rys. 4.12 Aplikacja AGOCOMP–APP wspierająca metodę AGOMAS

Źródło: Opracowanie własne

4.5. Podsumowanie i ocena konstrukcji modelu uproszczonego oraz opracowanej metody

W rozdziale przedstawiono konstrukcję i implementację uproszczonego modelu procesu wytwarzania oprogramowania. Model został najpierw przedstawiony w ujęciu ogólnym, opisano jego zmienne wejściowe i wyjściowe, a następnie strukturę wewnętrzną na poziomie modułów funkcjonalnych. W dalszej części przedstawiono model szczegółowy – elementy wewnętrzne i ich interakcje w przy zastosowaniu podejścia wieloagentowego do modelowania. Agenty i środowisko agentowe opisano w sposób formalny, za pomocą równań. Zachowania agentów realizujące algorytmy podane w rozdziale 3 opisano jako zbiór automatów skończonych. Wykonano w pełni funkcjonalną implementację modelu w środowisku Java w postaci systemu zawierającego symulator modelu, który została opisany w skrócie w tekście rozdziału. Pełny opis systemu AGOMO–APP znajduje się w dodatku B. Na podstawie modelu AGOMO opracowano metodę AGOMAS do oceny procesów RUP organizacji, która została przedstawiona w końcowych podrozdziałach.

Zbudowany model uproszczony i oparta na nim metoda, wraz z przygotowanymi narzędziami w postaci symulatora i aplikacji do porównania badanego projektu z wzorcem procesu pozwalają na przeprowadzenie zarówno weryfikacji zasadności replikatywnej modelu, jak i weryfikacji zasadności predyktywnej modelu, co zostanie zrealizowane w kolejnych rozdziałach.

5. Weryfikacja replikatywna modelu AGOMO

5.1. Wprowadzenie

Według sformułowania B. P. Zeiglera[110] „Zasadność modelu odpowiada na pytanie, czy nie można rozróżnić modelu od systemu rzeczywistego, w zaplanowanym układzie eksperymentu. Najbardziej podstawową koncepcją jest zasadność replikatywna modelu, która jest potwierdzona, jeżeli dla wszystkich eksperymentów możliwych w układzie eksperymentu zachowanie systemu rzeczywistego i modelu jest zgodne w akceptowalnych granicach tolerancji."

Celem weryfikacji replikatywnej modelu AGOMO zaprojektowano eksperyment, którego plan został oparty na metodzie AGOMAS i rozszerzony o etap weryfikacji replikatywnej. Eksperyment został zaplanowany w pięciu etapach, których przebieg zostanie opisany szczegółowo w następnym podrozdziale, a następnie podsumowany.

5.2. Eksperyment badawczy dla weryfikacji zasadności replikatywnej modelu

Celem eksperymentu była weryfikacja zasadności replikatywnej modelu oraz weryfikacja poprawności metody AGOMAS. Poniżej zostanie przedstawiona organizacja IT i projekt, którego dane użyto do eksperymentu, następnie zostanie przedstawiony plan eksperymentu i opisana realizacja jego etapów.

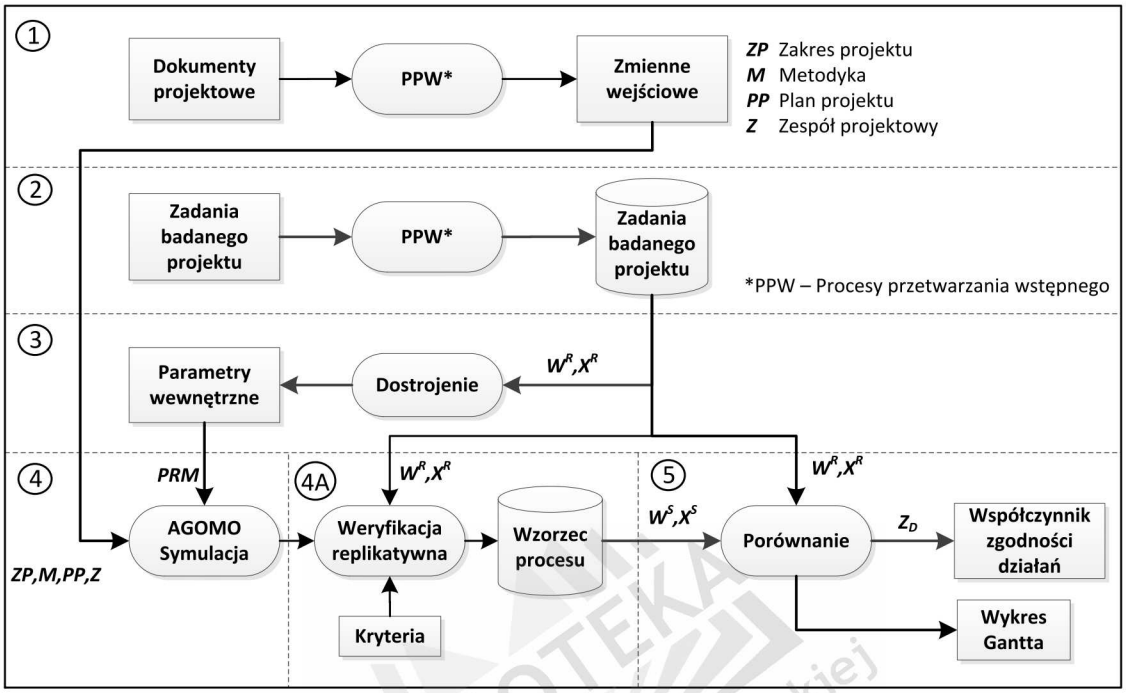
5.2.1. Badana organizacja IT i projekt

Do przeprowadzenia eksperymentu weryfikującego zasadność replikatywną niezbędne były dane projektu informatycznego, który model miałby odtworzyć. Dane dotyczące przebiegu procesu wytwarzania są trudne do uzyskania, ponieważ organizacje informatyczne chronią dane projektów informatycznych i nie udostępniają ich na zewnątrz. Podobnie zachowała się średniej wielkości organizacja IT, która pracowała nad projektem wykorzystanym w eksperymentach. Taka postawa organizacji IT jest bardzo poważną przeszkodą w prowadzeniu badań naukowych, która została ominięta w ten sposób, że autor w był zatrudniony danej organizacji i pracował wiele lat jako deweloper w zespole projektowym, nad projektem wykorzystanym w badaniach. Projektowi temu, ze względu na dziedzinę zastosowania, została nadana nazwa kodowa UBEZP. Dziedziną projektu UBEZP są ubezpieczenia społeczne. System powstawał jako rozbudowa poprzedniej wersji systemu z dużymi zmianami architektury i funkcjonalności. Proces wytwarzania był oparty na metodyce RUP i trwał 5 lat. Skład zespołu projektowego często się zmieniał, liczba osób wynosiła od 30 do ok. 140 osób. Liczba zadań wykonanych podczas wytwarzania systemu to ponad 30 000. Czas wykonania symulacji projektu wyniósł około 45 minut.

5.2.2. Plan eksperymentu weryfikacji zasadności replikatywnej modelu

Eksperyment został zaplanowany w sześciu etapach według metody AGOMAS opisanej w podrozdziale 4.4. Między etapami 4 i 5 metody AGOMAS został dodany etap 4A, którego celem jest weryfikacja replikatywna modelu. Plan eksperymentu został zamieszczony na Rys.

5.1. Niektóre etapy metody zostały rozszerzone na kilka faz z powodu dużej ilości przetwarzanych źródeł informacji o projekcie. Poniżej zostaną przedstawione wyniki kolejnych faz eksperymentu.



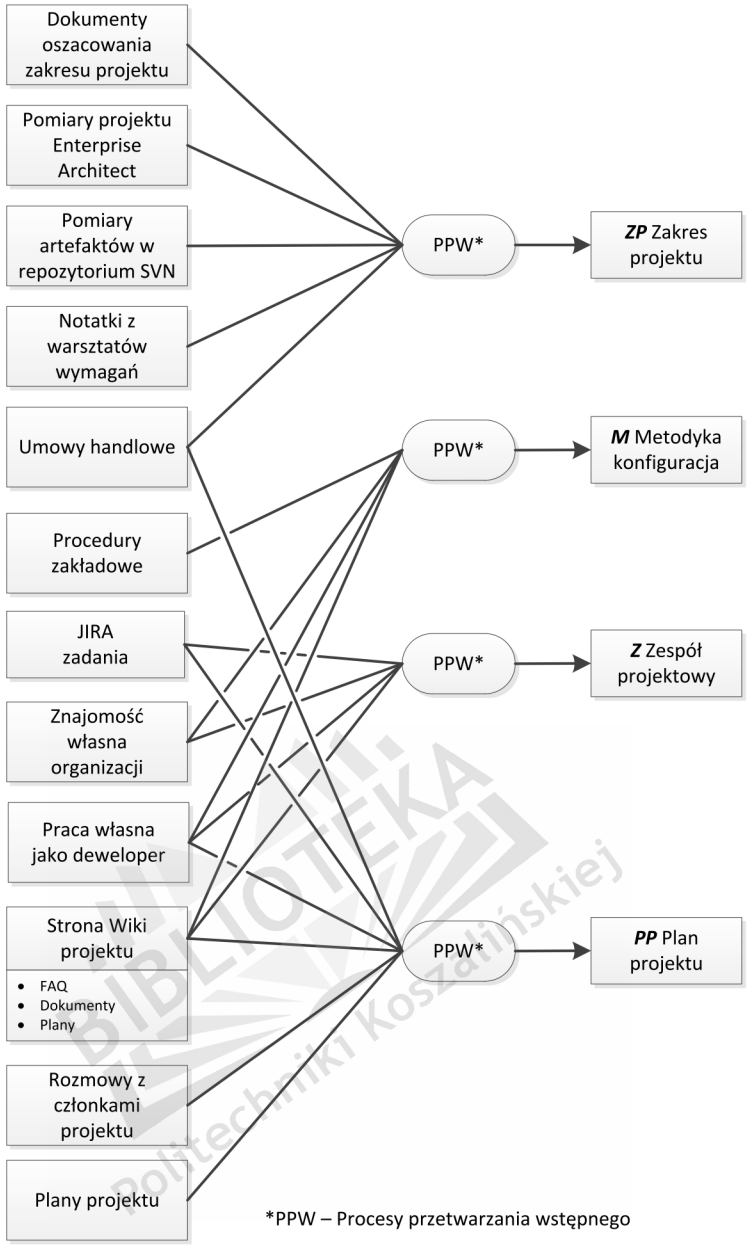
Rys. 5.1 Plan eksperymentu weryfikacji zasadności replikatywnej modelu
Źródło: Opracowanie własne

5.2.3. Realizacja etapu pierwszego – ustalenie wartości zmiennych wejściowych

Celem pierwszego etapu eksperymentu jest uzyskanie wartości zmiennych wejściowych modelu. Źródła danych, które zostały przeanalizowane i wstępnie przetworzone oraz ich powiązania ze zmiennymi wejściowymi przedstawiono na Rys. 5.2.

Ustalenie wartości zmiennej wejściowej ZP – zakres projektu

Wstępne szacowanie rozmiaru projektu w celu uzgodnienia zakresu, terminu i budżetu były przeprowadzone przy użyciu metody Use Case Points i zostały zawarte w załącznikach do umowy handlowej. Zgodnie z przeprowadzonymi obliczeniami, całkowity rozmiar systemu wynosi 7 413 [AGOMO–UCP]. Na podstawie analizy notatek powstałych po przeprowadzeniu spotkań i warsztatów wymagań oszacowano, że rozmiar modernizacji systemu wynosi $zn = 549$ [AGOMO–UCP]. W związku z tym większość wymagań dotyczących systemu pozyskano z poprzedniej wersji oprogramowania i rozmiar ten wynosi $zo = 6\,864$ [AGOMO–UCP]. Wartość tak obliczonego zakresu projektu umieszczono w Tab. 5.1.



Rys. 5.2 Źródła informacji użyte do określenia wartości zmiennych wejściowych
Źródło: Opracowanie własne

Dodatkowo potwierdzono wyliczony rozmiar projektu przez obliczenie rozmiaru projektu na podstawie przypadków użycia zawartych w projekcie sporządzonym przy użyciu oprogramowania Enterprise Architect. Uzyskane dane są spójne z liczbą klas umieszczonych w repozytorium kodu źródłowego. Całkowity rozmiar systemu zgadza się z policzoną pracochłonnością prac, która wynosi 347 685 roboczogodzin. Wyliczony z tych danych współczynnik liczby roboczogodzin na jeden punkt AGOMO–UCP wynosi 47, co odpowiada współczynnikowi 9,4 dla metody UCP i jest wartością prawidłową dla doświadczonych zespołów projektowych. Uzyskano zatem spójne dane, które uzupełnione o plan projektu i informacje o zespole projektowym pozwoliły wykonać symulację badanego projektu.

Tab. 5.1 Wartość zmiennej wejściowej ZP – zakres projektu
Źródło: Opracowanie własne

Zmienna wejściowa	Wartość
Całkowity rozmiar projektu według metody AGOMO–UCP	7 412
Rozmiar rozbudowy systemu AGOMO–UCP, zn	549
Rozmiar wymagań odzyskanych z poprzedniej wersji systemu w AGOMO–UCP, zo	6 863

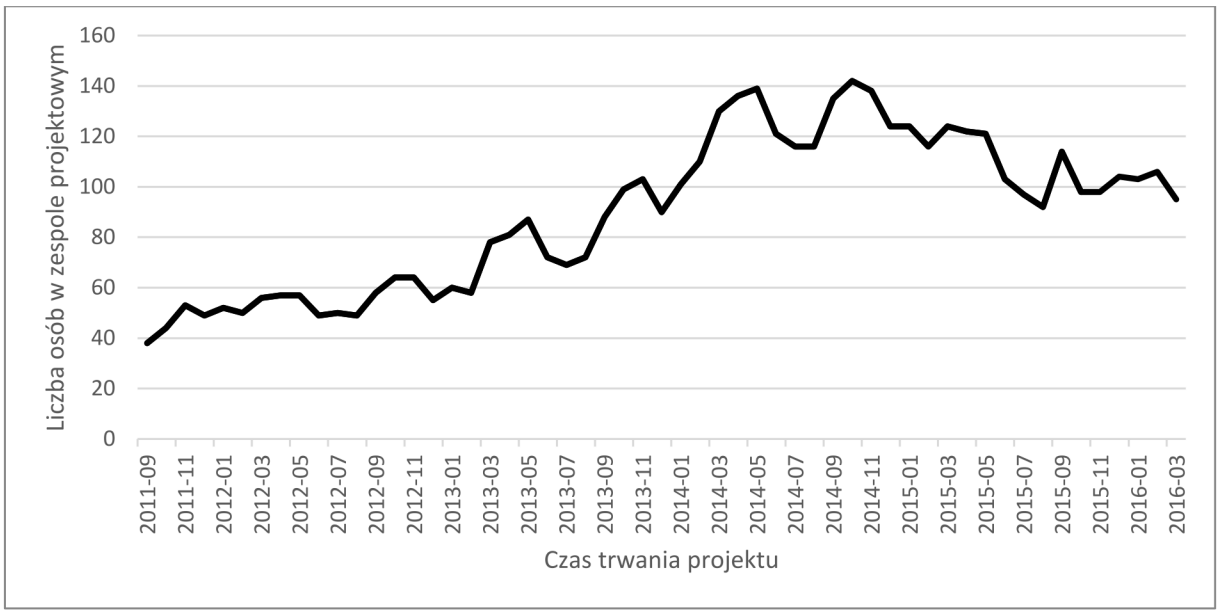
$$ZP = (549, 6863) \tag{5.13}$$

Ustalenie wartości zmiennej wejściowej M – metodyka wytwarzania

Konfiguracja metodyki wytwarzania RUP użyta w badanym projekcie została opracowana na podstawie literatury na temat metodyki [58, 59, 61] oraz na podstawie informacji zawartych w procedurach zakładowych, stronie internetowej wiki projektu oraz doświadczenia autora z pracy w projekcie UBEZP w roli dewelopera. Konfiguracja metodyki RUP zawiera etapy od rozpoczęcia do wdrożenia. W przepływach działań pominięto działania analizy biznesowej, ponieważ takie działania nie występowały w badanym projekcie.

Ustalenie wartości zmiennej wejściowej Z – zespół projektowy

Skład zespołu projektowy ulegał częstym zmianom w trakcie trwania procesu wytwarzania. W początkowym etapie projektu pracowało około 30 osób, głównie analitycy systemowi odtwarzający przypadki użycia z poprzedniej wersji systemu. Następnie zespół rozrastał się, aby zrealizować cele zaplanowane w kolejnych iteracjach powstawania systemu informatycznego. Niestety nie istniał żaden dokument lub źródło informacji, które zawierałoby informacje niezbędne do ustalenia wartości zmiennej wejściowej modelu. Lista osób, które pracowały w zespole powstała głównie na podstawie zadań systemu JIRA. Pełna lista członków zespołu liczy ponad 200 osób, jednak zespół nigdy się nie składał z tak dużej liczby osób, ponieważ w zespole następowało dużo zmian. Średnia miesięczna liczba osób w zespole podczas procesu wytwarzania wzrosła do około 140 osób, co zostało pokazane na Rys. 5.3. Role projektowe przypisane członkom zespołu projektowego zostały odtworzone na podstawie informacji umieszczonych na portalu wiki projektu oraz na podstawie zadań systemu JIRA związanych z daną osobą. Okresy zatrudnienia i zaangażowanie w projekt oraz informacja, czy dana osoba wykonywała nadgodziny i w jakich okresach zostały odtworzone na podstawie informacji o wykonanych zadaniach z systemu JIRA.



Rys. 5.3 Średnia miesięczna liczba osób w zespole projektowym

Źródło: Opracowanie własne

Ustalenie wartości zmiennej wejściowej *PP* – plan projektu

Plan projektu kilka razy ulegał zmianom w trakcie procesu wytwarzania. Powstawały nowe dokumenty planów projektu, poprzednie plany zostały porzucone i nie były uaktualniane. Również w przypadku planu projektu nie było jednego miejsca, w którym zgromadzono by spójne i aktualne dane. Wobec tego, plan został odtworzony głównie z informacji o zadaniach przechowywanych w systemie JIRA. Informacje te zostały uzupełnione o dane z portalu Wiki projektu, rozmowy z uczestnikami projektu i wiedzę uzyskaną podczas pracy autora w projekcie w roli dewelopera. Zawartość tak odtworzonego planu projektu została zamieszczona w Tab. 5.2. Dla dyscyplin wymagań, analizy i projektowania oraz implementacji przyjęto, że prace zostaną zakończone w etapie konstrukcji.

Tab. 5.2 Odtworzona zawartość zmiennej *PP* – plan projektu

Źródło: Opracowanie własne

Dyscyplina	Cele Iteracji [%]						
	I1	C1	C2	C3	C4	C5	T1
Wymagania	35	50	55	65	80	100	100
Analiza i projektowanie	10	45	55	70	80	100	100
Implementacja	0	15	25	40	65	100	100
Testowanie	10	15	25	40	60	95	100
Wdrożenie	0	0	0	0	0	5	100
Zarządzanie konfiguracją i zmianą	0	5	10	20	25	35	100
Zarządzanie projektem	40	70	70	75	80	95	100
Środowisko	40	50	60	75	85	95	100

5.2.4. Realizacja etapu drugiego – przetworzenie zadań projektu UBEZP

Drugi etap eksperymentu miał na celu utworzenie spójnej i kompletnej bazy danych zadań wykonanych w projekcie UBEZP. Była ona potrzebna do obliczenia pracochłonności według kategorii zadań, celem dostrojenia parametrów wewnętrznych modelu, przeprowadzenia weryfikacji zasadności replikatywnej modelu i oceny dojrzałości procesu. Aby spełnić swe zadanie, baza zawierała informacje: do jakiej kategorii należy zadanie, kto je wykonał, czas rozpoczęcia i trwania. Dostępne zapisy dotyczące wykonanych prac miały tylko opisy tekstowe. Na podstawie opisów określono kategorie zadań, które zostały wykorzystane do określenia struktury pracochłonności. To pozwoliło na dostrojenie wartości parametrów wewnętrznych modelu. Wprowadzono 14 kategorii zadań projektowych i według nich określono strukturę pracochłonności procesu wytwarzania projektu UBEZP.

Informacje z systemu JIRA nie zawierały szczegółowych informacji o tym, kiedy rozpoczęto i ile czasu trwało wykonanie zadania. Te informacje zostały odtworzone z sumarycznej liczby godzin przy założeniu, że osoba pracowała nad jednym zadaniem na raz i dopiero po jego zakończeniu rozpoczynała kolejne według daty zgłoszenia. Zadania projektu UBEZP zostały zaimportowane do bazy danych Oracle i przetworzone automatycznie za pomocą procedur napisanych w języku PL/SQL.

5.2.5. Realizacja etapu trzeciego – dostrojenie modelu

Celem dostrojenia modelu było odtworzenie za pomocą modelu, jak najdokładniej procesu projektu UBEZP, aby mógł być użyty jako wzorzec do porównania z badanym procesem. Dostrojenie modelu polegało ma dobraniu wartości parametrów wewnętrznych modelu. Pierwszą fazą dostrojenia było odtworzenie, przy pomocy modelu, artefaktów o typach i liczbie zbliżonej po artefaktów projektu UBEZP. Drugą fazą dostrojenia było uzyskanie pracochłonności zadań, pogrupowanych według kategorii, o wartościach jak najbardziej zbliżonych do wartości obliczonych dla projektu UBEZP. Dwie fazy dostrojenia zostały opisane w kolejnych podrozdziałach.

Dostrojenie liczby artefaktów modelu

Czas wykonania i pracochłonność zadań w symulowanym procesie zależy od liczby artefaktów wejściowych. Za proporcje między liczbą wejściowych artefaktów a liczbą wyprodukowanych artefaktów wyjściowych odpowiadają parametry wewnętrzne. Dlatego celem pierwszej fazy dostrajania było dobranie parametrów wewnętrznych symulacji tak, aby uzyskać liczbę artefaktów utworzonych w symulowanym procesie, zbliżoną do rzeczywistej. Wartości parametrów wewnętrznych to liczby naturalne, ponieważ liczba artefaktów jest wartością dyskretną. Na przykład parametr *ClassDesign.inPerOut* o wartości 3 oznacza, że zadanie *ClassDesignTask* wyprodukuje jedną klasę projektową *DesignClass* dla trzech przetworzonych wejściowych klas analitycznych *AnalysisClass*. Za wyprodukowanie klas projektowych odpowiadają trzy różne typy zadań, więc dobierając dla nich wartości parametrów wewnętrznych, można ustalić proporcje między liczbą przetwarzanych przez nie artefaktów wejściowych a liczbą wytworzonych klas projektowych, co pozwala określić całkowitą liczbę klas projektowych wyprodukowanych przez model.

Dostrojenie pracochłonności zadań według kategorii

Celem drugiej fazy dostrojenia było odtworzenie pracochłonności projektu UBEZP. Zostało to osiągnięte przez dobranie wartości parametrów wewnętrznych odpowiadających za czas przetwarzania artefaktów wejściowych na wyjściowe dla wszystkich typów zadań metodyki RUP. W pierwszej fazie uzyskano liczbę artefaktów wytworzonych przez model zbliżoną do liczby artefaktów badanego projektu. Dzięki temu parametry wewnętrzne odpowiadające za czas przetwarzania typów zadań wskazują na średni czas przetwarzania pojedynczych artefaktów. Parametry czasowe mają wartości uśrednione dla całego zespołu projektowego.

Do dostrajania wykorzystano informacje o kategorii i czasie trwania zadań badanego projektu utworzone w fazie drugiej eksperymentu. Podczas dostrajania każdej kategorii zadań badanego projektu przypisano odpowiadające jej typy zadań modelu. Na podstawie sumarycznych pracochłonności, pogrupowanych według kategorii i wag przypisanym typom zadań, wyznaczono pracochłonności typów zadań i dzięki temu obliczono wartości parametrów czasowych typów zadań modelu. Parametry wewnętrzne określające czas wykonania typów zadań zostały dobrane tak, aby w symulacji uzyskać pracochłonność zbliżoną do rzeczywistej.

5.2.6. Realizacja etapu czwartego – symulacja procesu wytwarzania i weryfikacja replikatywna modelu

W etapie czwartym przeprowadzono symulację procesu wytwarzania oprogramowania opartą na modelu AGOMO, podając na wejście modelu wartości zmiennych otrzymane w pierwszym etapie eksperymentu. Parametrom wewnętrznym modelu nadano wartości otrzymane w trzecim etapie eksperymentu. W celu zweryfikowania zdolności replikatywnej modelu harmonogram procesu otrzymany na wyjściu modelu został porównany z harmonogramem projektu UBEZP. Wyniki symulacji i przebieg weryfikacji przedstawiono w podrozdziałach.

Symulacja procesu wytwarzania oprogramowania

Celem otrzymania wynikowego harmonogramu prac, przeprowadzono symulację procesu wytwarzania oprogramowania opartą na modelu AGOMO. Dla symulacji przyjęto wartości zmiennych wejściowych ustalone w pierwszym etapie eksperymentu i wartości parametrów wewnętrznych dostrojone do badanego projektu UBEZP w etapie trzecim eksperymentu.

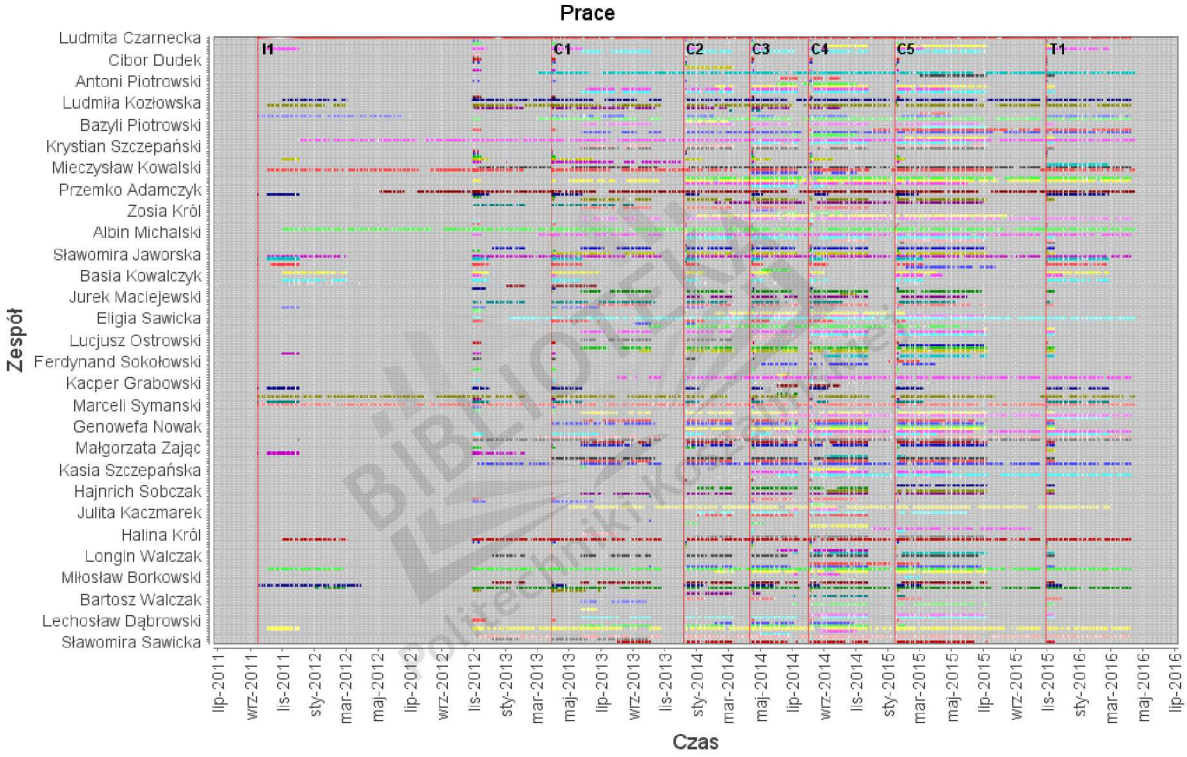
Wyniki porównania liczby artefaktów wyprodukowanych przez model z liczbą artefaktów projektu UBEZP umieszczono w Tab. 5.3. Zamieszczono w niej wybrane artefakty badanego projektu, ich liczbę, odpowiadające im artefakty modelu, liczbę artefaktów wyprodukowanych przez model oraz błąd względny między wartościami rzeczywistymi a wynikiem symulacji procesu wytwarzania oprogramowania opartą na modelu AGOMO. Błąd względny jest poniżej 10%, co uznano za wartość wystarczającą, biorąc pod uwagę ograniczenia modelu, w którym proporcje między liczbami przetwarzanych a produkowanych artefaktów są wyrażone za pomocą liczb naturalnych. Artefakty projektu UBEZP wybrano ze względu na łatwość dostępu do informacji o ich liczbie. Liczbę plików źródłowych kodu w języku java obliczono na podstawie zawartości odpowiednich katalogów repozytorium projektu. Liczbę klas projektowych i interfejsów uzyskano ze statystyk projektu wykonanego w systemie Enterprise Architect.

Tab. 5.3 Wyniki symulacji – porównanie liczby artefaktów

Źródło: Opracowanie własne

Artefakty projektu UBEZP	Artefakty modelu	Badany projekt	Wynik symulacji	Błąd względny
Pliki java	ImplementationElement	15517	14343	8%
Klasy projektowe	DesignClass	10402	10206	2%
Interfejsy	Interface	1940	2069	6%

Wynikowy diagram Gantta harmonogramu prac uzyskanego w symulacji został przedstawiony na Rys. 5.4. Pionowymi liniami oddzielono iteracje procesu wytwarzania, kolorami oznaczono osobę wykonującą zadania.



Rys. 5.4 Diagram Gantta przedstawiający harmonogram otrzymany w wyniku symulacji

Źródło: Opracowanie własne

Weryfikacja zdolności replikatywnej

Po przeprowadzeniu symulacji, uzyskane wyniki posłużyły do weryfikacji zdolności replikatywnej modelu. W celu weryfikacji replikatywnej modelu należy przyjąć kryteria porównania zmiennych wyjściowych modelu ze zmiennymi wyjściowymi systemu rzeczywistego i określić dopuszczalną granicę błędu. Podstawowym parametrem określającym przebieg prac jest pracochłonność. Jako dopuszczalną granicę błędu wyznaczono 1% wartości pracochłonności. Określona granica błędu jest dla projektu UBEZP równoznaczna z pięcioma dniami roboczymi zespołu o liczebności 89 osób, czyli wartości średniej dla projektu. Z praktycznego punktu widzenia opóźnienie o tydzień prac nad projektem trwającym kilka lat wydaje się być w zupełności do przyjęcia. Rozważano dwa podejścia do porównywania pracochłonności. Pierwszym z nich jest pracochłonność sumaryczna w podziale na kategorie zadań. To podejście zostało

użyte do dostrojenia wartości parametrów wewnętrznych modelu. Wyniki symulacji modelu, zamieszczone w

Tab. 5.4, pokazują wysoką zgodność zachowania modelu i systemu rzeczywistego. Błąd względny sum pracochłonności dla wprowadzonych kategorii jest dużo niższy niż 1%, a dla większości kategorii jest niższy niż 5%, tylko dla dwu kategorii błąd względny jest większy, jednak pracochłonności tych kategorii są niskie w porównaniu do pracochłonności sumarycznej. Błąd względny dla pracochłonności sumarycznej jest bardzo niski i jest bliski zeru. Te wyniki potwierdzają zasadność replikatywna modelu.

Tab. 5.4 Wyniki symulacji – porównanie pracochłonności wg kategorii zadań
Źródło: Opracowanie własne

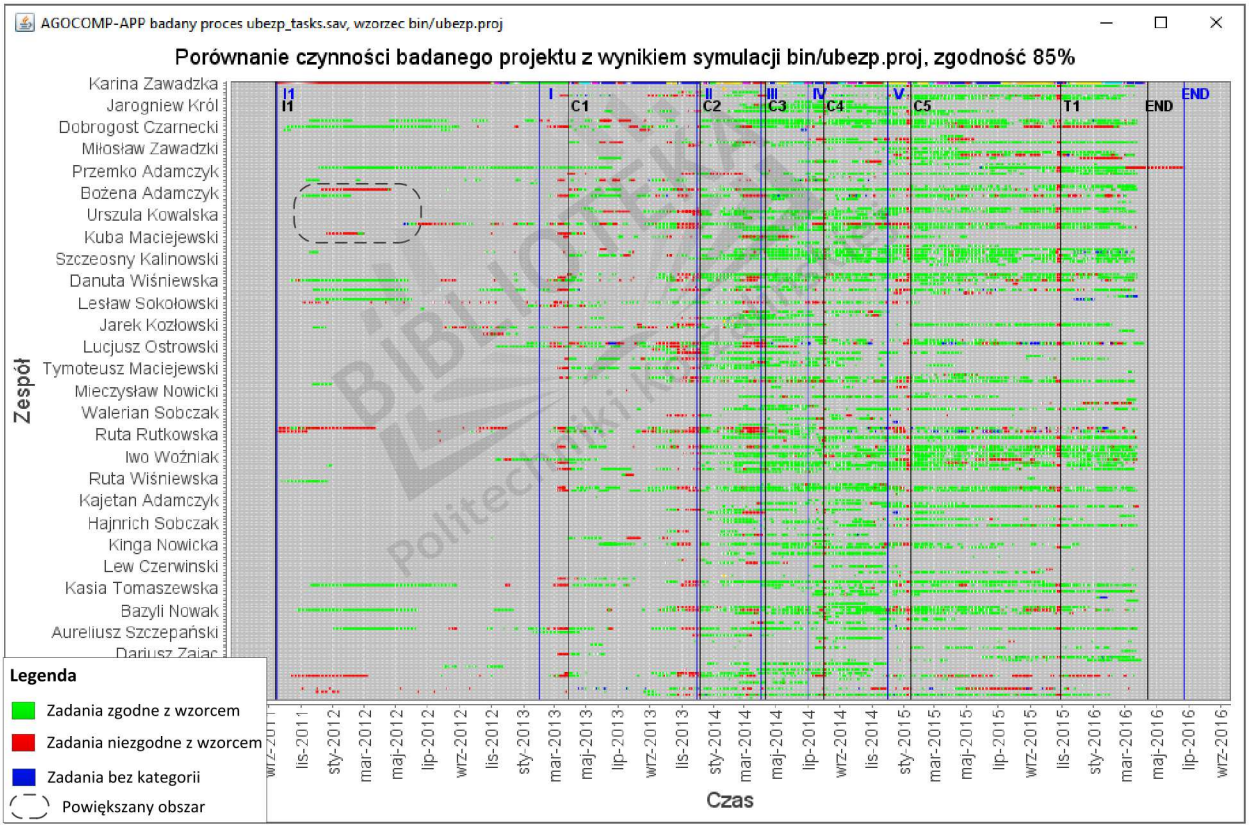
Kategoria	Badany projekt [RBH]	Model [RBH]	Błąd względny
Projekt architektury	1 301	1 298	0,23%
Prototyp architektury	3 062	3 063	0,04%
Integracja systemu	2 978	2 671	10,31%
Zarządzanie konfiguracją	5 198	5 548	6,31%
Implementacja	126 331	126 028	0,24%
Projektowanie	17 667	17 507	0,91%
Projekt analityczny	8 577	8 340	2,77%
Projekt bazy danych	5 967	5 932	0,59%
Projekt interfejsu graficznego	7 644	7 321	4,23%
Testowanie	85 630	86 106	0,55%
Model przypadków użycia	33 488	33 704	0,64%
Warsztaty wymagań i wywiady	4 330	5 020	13,74%
Zarządzanie projektem	17 215	17 282	0,39%
Razem	**348 308**	**348 273**	**0,01%**

Drugim podejściem do porównywania pracochłonności prac w modelu i systemie rzeczywistym jest porównanie pracochłonności sumarycznej pogrupowanej według kolejnych iteracji procesu wytwarzania. To podejście nie było stosowane w przypadku weryfikacji replikatywnej, ponieważ rozkład prac w czasie zależy od zmiennej wejściowej plan projektu oraz od metodyki wytwarzania oprogramowania, która też jest zmienną wejściową modelu. Poza tym harmonogram prac na wyjściu modelu nie zawsze są zgodne z harmonogramem prac w systemie rzeczywistym. Przyczyną różnicy jest to, że na wyjściu modelu otrzymujemy harmonogram wzorcowy, którego przebieg prac wynika z przyjętej metodyki wytwarzania i planu projektu. Czasy trwania iteracji w symulacji są zbliżone do czasów trwania iteracji w rzeczywistym projekcie, co widać na Rys. 5.5 przedstawiającym porównanie harmonogramu wynikowego symulacji, czyli wzorcowego z rzeczywistym przebiegiem badanego procesu wytwarzania.

Harmonogram otrzymany na wyjściu modelu jest dobrze dopasowany do badanego przebiegu, więc można go zastosować jako wzorzec procesu do porównania z badanym projektem i ustalenia współczynnika zgodności w piątym etapie eksperymentu.

5.2.7. Realizacja etapu piątego eksperymentu – porównanie badanego procesu z wzorcem

W etapie piątym eksperymentu wykorzystano wyniki symulacji procesu RUP jako wzorzec do porównania z przebiegiem badanego projektu. Porównanie zostało przeprowadzone za pomocą aplikacji AGOCOMP–APP, wspierającej metodę AGOMAS. Aplikacja jest opisana w dodatku C rozprawy. Wynik porównania w postaci diagramu Gantta zamieszczono na Rys. 5.5. Diagram pokazuje zadania rzeczywistego procesu wytwarzania. Kolor zadań jest zależny od zgodności z wzorcem projektu. Zadania zgodne z wzorcem są oznaczone kolorem zielonym, zadania niezgodne są w kolorze czerwonym, natomiast zadania o bez przypisanej kategorii są wyświetlone w kolorze niebieskim. Na rysunku widać duży zielony obszar, który oznacza wysoką zgodność badanego procesu z wzorcem. Jest przerwany nielicznymi czerwonymi pionowymi polami. Te czerwone skupiska są wynikiem niezgodności zadań z wzorcem, spowodowaną różnicami w czasie trwania iteracji wzorca i iteracji projektu rzeczywistego. Obszar wykresu zaznaczony linią przerywaną został powiększony i przedstawiony na następnym rysunku

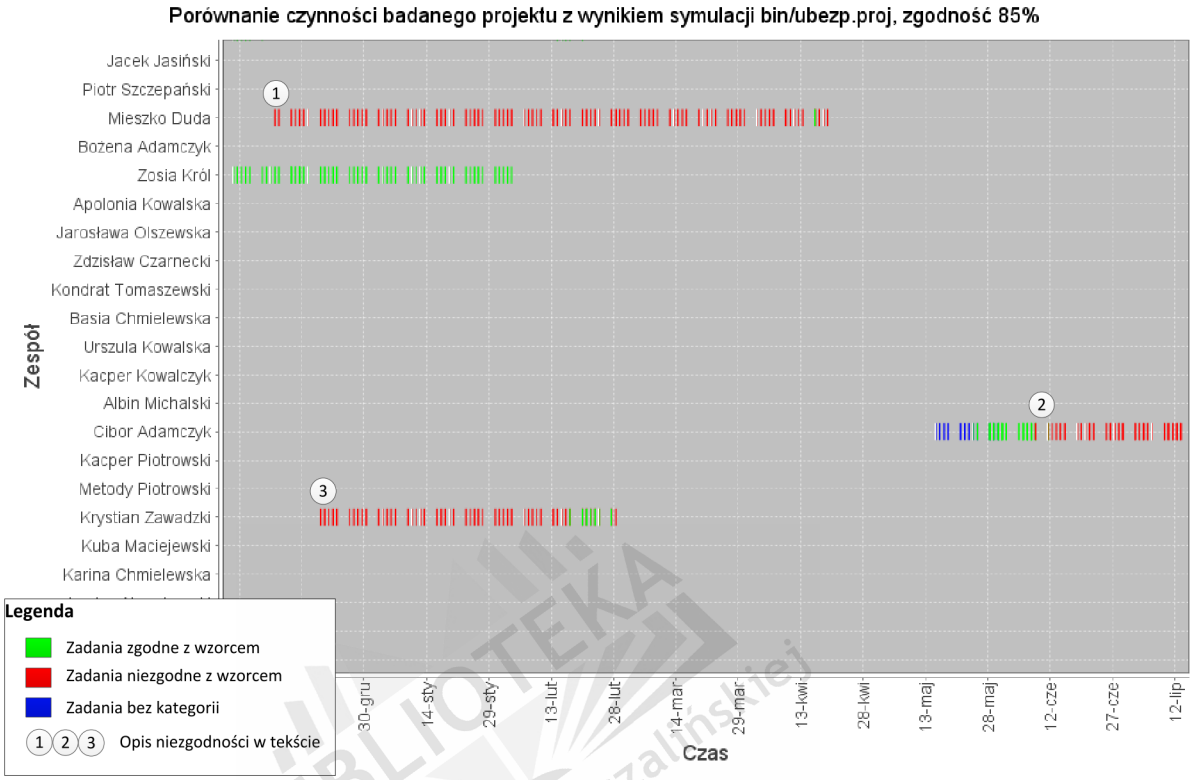


Rys. 5.5 Aplikacja AGOCOMP–APP – wynik porównania wzorca procesu z przebiegiem badanego projektu

Źródło: Opracowanie własne

Na Rys. 5.6 powiększono obszar etapu rozpoczęcia, na którym widać trzy czerwone zadania niezgodne z wzorcem procesu. Zadanie nr 1 należy do kategorii programowanie, zadanie nr 2 to prace nad wykonaniem projektu analitycznego, zadanie nr 3 to prace nad prototypem architektury. Wszystkie te zadania wystąpiły zbyt wcześnie, ponieważ w tym momencie etapu rozpoczęcia są pozyskiwane żądania udziałowców i przetwarzane w wymagania i przypadki uży-

cia. Właściwy czas na wykonanie prototypu architektury (3) jest w późniejszej fazie etapu rozpoczęcia, projekt analityczny i programowanie powinny być realizowany w kolejnym etapie – konstrukcji.



Rys. 5.6 Aplikacja AGOCOMP–APP – powiększony obszar wyniku porównania wzorca procesu z przebiegiem badanego projektu

Źródło: Opracowanie własne

Liczbowe wyniki porównania oraz wynikowy współczynnik zgodności działań Z_D obliczony wg wzoru (4.21) zostały umieszczone w Tab. 5.5. Zgodność działań na poziomie 85% zinterpretowana zgodnie ze wskazówkami zawartymi Tab. 4.1 oznacza równoważność z osiągnięciem poziomu 3 – zdefiniowanego w modelu CMMI co może oznaczać gotowość do rozpoczęcia procesu zwinnej transformacji organizacji.

Tab. 5.5 Wyniki porównania badanego procesu z wzorcem procesu

Źródło: Opracowanie własne

Zmienna	Opis	Wartość
L_Z	Liczba zadań zgodnych	54 793
L_W	Liczba wszystkich zadań	64 740
Z_D	Współczynnik zgodności działań	85%

5.3. Podsumowanie – znaczenie wyników eksperymentu i procesów weryfikacji replikatywnej

W rozdziale przedstawiono plan eksperymentu oparty na metodzie AGOMAS, rozszerzony o etap 4A, przeznaczony do weryfikacji replikatywnej modelu. W ramach przeprowadzonego

eksperymentu przedstawiono badaną organizację i scharakteryzowano badany projekt, a następnie przeprowadzono pięć etapów eksperymentu. Pierwszy etap eksperymentu, przeznaczony na ustalenie wartości zmiennych wejściowych modelu był bardzo pracochłonny, ponieważ wymagał zebrania danych z wielu rozproszonych źródeł. Podobnie pracochłonny był drugi etap, w którym uzupełniono i przetworzono informacje o zadaniach badanego projektu na format niezbędny do przeprowadzenia dalszych etapów metody. Przetwarzanie informacji było zrealizowane automatycznie przy użyciu, opracowanych do tego celu, procedur bazodanowych. Podstawowymi danymi, na których były oparte dwa pierwsze etapy były dane z systemu śledzenia błędów i systemu zarządzania projektami JIRA. Istnieje możliwość zautomatyzowania dwu pierwszych etapów metody AGOMAS, ponieważ system JIRA umożliwia dodanie dodatkowych pól do przechowywanych informacjach o zadaniach oraz posiada mechanizm wtyczek, który umożliwia łatwe tworzenie aplikacji zintegrowanych z systemem przez API, usługi sieciowe oraz wspólną bazę danych. Trzeci etap eksperymentu zawierał dwustopniowe dostrajanie parametrów wewnętrznych modelu, które również może zostać zautomatyzowane i przeniesione do dedykowanej aplikacji. W czwartym etapie przeprowadzono symulację modelu z wartościami zmiennych wejściowych i parametrów wewnętrznych ustalonych we wcześniejszych etapach eksperymentu. Porównanie pracochłonności harmonogramu otrzymanego na wyjściu modelu z pracochłonnością badanego projektu pozwoliło na weryfikację zasadności replikatywnej modelu. Można powiedzieć, że model odtworzył przebieg badanego projektu informatycznego z dużą dokładnością, ponieważ sumaryczny błąd względny wyniósł 0,01%. W etapie piątym harmonogram działań – zmienna wyjściowa modelu została użyta jako wzorzec procesu do porównania z harmonogramem badanego projektu. Efektem tego porównania była wysoka wartość wskaźnika zgodności, co zostało zinterpretowane jako poziom 3 dojrzałości organizacji według modelu CMMI, z czym może być związane niskie ryzyko zwinnej transformacji organizacji. Wysoki poziom dojrzałości procesu wytwarzania i niski poziom ryzyka zostały potwierdzone przez udaną transformację badanej organizacji. Zatem cele eksperymentu zostały osiągnięte, co oznacza potwierdzenie zarówno zdolności modelu do odtworzenia przebiegu rzeczywistego procesu wytwarzania oprogramowania, jak i przydatność metody AGOMAS do oceny procesów wytwarzania oprogramowania, której znaczenie rośnie wraz z upowszechnianiem zwinnych metodyk.

Zakres stosowania metody AGOMAS może obejmować również porównywanie bieżącego stanu projektu – harmonogramu wykonanych prac z wzorcowym – zaplanowanym uprzednio harmonogramem. Wynik porównania w postaci współczynnika zgodności działań może zostać zinterpretowany jako zgodność procesu wytwarzania z metodyką i planem wytwarzania, więc może być wykorzystany do kontrolowania bieżącego stanu procesu wytwarzania. Narzędzia do analizy graficznej różnic między porównanymi planowanym i rzeczywistym procesami pozwalają poznać przyczyny różnic, co pozwoli na osiągnięcie wysokiej zgodności oraz, co za tym idzie dojrzałości procesu wytwarzania oprogramowania. Metoda AGOMAS w tym szerszym ujęciu, wraz z narzędziami graficznymi do analizy różnic, mogłaby znaleźć zastosowanie jako narzędzie diagnostyczne do kontroli przebiegu procesu wytwarzania w systemach wsparcia zarządzania procesem wytwarzania oprogramowania, co istotnie zwiększyłoby znaczenie i rozpowszechniłoby zarówno metodę AGOMAS, jak i model AGOMO. W następnym rozdziale zostanie opracowana metoda planowania procesu wytwarzania oprogramowania opartą na modelu AGOMO i następnie, stosując tą metodę, zostanie przeprowadzona weryfikacja zasadności prognostycznej modelu.

6. Weryfikacja prognostyczna modelu

6.1. Wprowadzenie

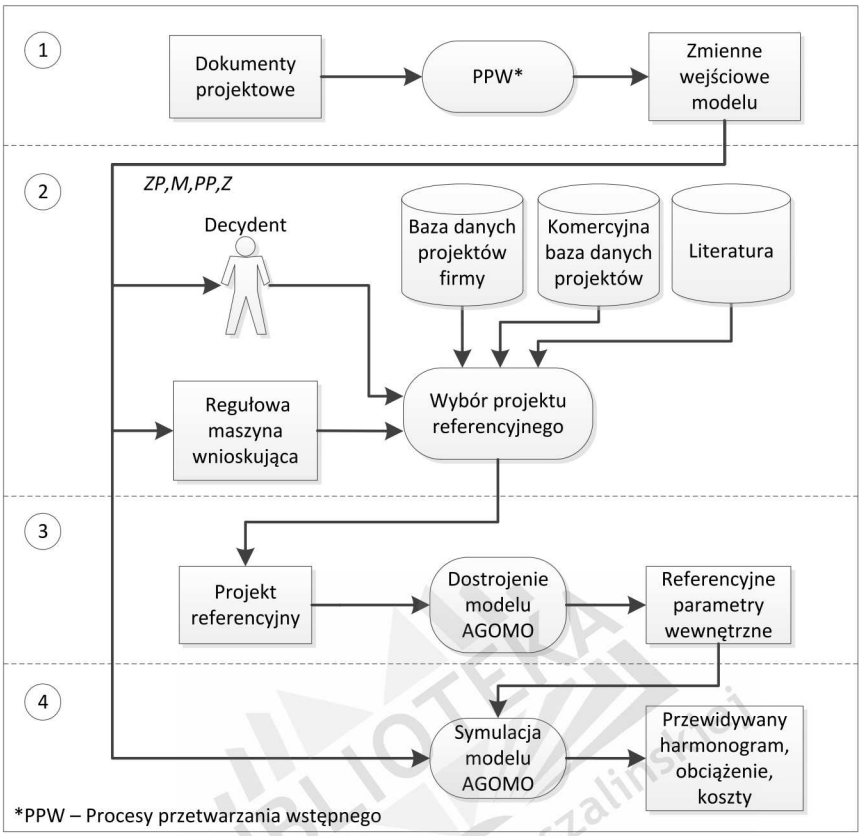
Weryfikacja zasadności predykcyjnej modelu jest drugim i końcowym etapem sprawdzenia poprawności modelu AGOMO. B. P. Zeigler scharakteryzował ten etap następująco: „w weryfikacji predykcyjnej wymagana jest nie tylko zdolność modelu do odtworzenia zachowania systemu rzeczywistego, ale również zdolność przewidywania, nieznanego do tej pory, zachowania systemu rzeczywistego. Aby tą zdolność osiągnąć model powinien być w stanie odpowiadającym stanowi systemu rzeczywistego" [110]. Wiedza modelu AGOMO o modelowanym systemie jest zawarta w jego parametrach wewnętrznych, odpowiadających za czas wykonywania atomowych zadań przyjętej metodyki. Celem sprawdzenia zdolności predykcyjnej modelu użyto parametrów wewnętrznych o wartościach dostrojonych do projektu referencyjnego. To podejście zastosowano w opisanej poniżej, czteroetapowej metodzie planowania procesów RUP – AGOPLAN. Następnie przedstawiono plan eksperymentu oparty na metodzie AGOPLAN, badania i wyniki kolejnych etapów eksperymentu oraz podsumowanie i ocenę możliwości wykorzystania zdolności predykcyjnych modelu AGOMO.

6.2. Metoda planowania procesów RUP

Oprócz zastosowania do oceny procesów wytwarzania oprogramowania oraz kontroli zgodności procesu z planowanym, model AGOMO może być stosowany we wstępnych fazach projektu do planowania przebiegu procesu wytwarzania, odpowiedniego dobrania składu zespołu projektowego, dostosowania metodyki wytwarzania do projektu, oszacowania terminu zakończenia i kosztów projektu. Do planowania procesu wytwarzania przy pomocy modelu AGOMO wykorzystuje się dokumenty projektowe, informacje o zespole projektowym i stosowanej metodyce oraz dane referencyjnego projektu wytwarzania. Najkorzystniejszym źródłem referencyjnego projektu wytwarzania jest wewnętrzna baza ukończonych projektów informatycznych organizacji. Wynikiem planowania procesu wytwarzania jest harmonogram prac, raport obciążenia zespołu projektowego w czasie oraz raport kosztów wynikającego z zatrudnienia zespołu projektowego, w tym z planowanych i wykorzystanych nadgodzin. Symulację procesu można wielokrotnie powtarzać modyfikując wartości zmiennych wejściowych do otrzymania zakładanych, właściwych wyników. Koncepcja zastosowania modelu AGOMO do planowania procesu wytwarzania oprogramowania zgodnego z metodyką RUP została pokazana na Rys. 6.1. Opierając się na niej, opracowano metodę planowania procesu RUP AGent–Object model process PLANning method (AGOPLAN), która składa się z następujących etapów:

1. Ustalenie wartości zmiennych wejściowych: zakres projektu, plan projektu, konfiguracji metodyki RUP oraz skład zespołu projektowego i role przypisane członkom, na podstawie dokumentów badanego projektu.
2. Wybranie projektu referencyjnego z bazy danych projektów na podstawie zgromadzonych w etapie pierwszym informacji o planowanym projekcie.
3. Dostrojenie modelu AGOMO do wybranego projektu referencyjnego.
4. Wykonanie symulacji procesu wytwarzania oprogramowania opartej na modelu AGOMO, z parametrami wejściowymi ustalonymi w pierwszym etapie i parametrami

wewnętrznymi o wartościach odpowiadających projektowi referencyjnemu, o wartościach ustalonych w trzecim etapie metody. Wynikiem przeprowadzonej symulacji jest planowany harmonogram prac w procesie RUP.



Rys. 6.1 Etapy metody AGOPLAN – planowania procesu RUP przy użyciu modelu AGOMO
Źródło: Opracowanie własne

6.2.1. Etap pierwszy metody – ustalenie wartości zmiennych wejściowych modelu

Zmienne wejściowe modelu AGOMO są wspólne dla metody planowania AGOPLAN i metody oceny dojrzałości AGOMAS. Sposoby ustalenia wartości zmiennych wejściowych, opisane w pkt. 4.4.1, mogą być zastosowane w metodzie AGOPLAN. Warto jednak zaznaczyć, że w metodzie AGOPLAN podstawowym źródłem danych do wyznaczania wartości zmiennych wejściowych powinny być wstępne wersje dokumentów projektowych. Danych pochodzących z Systemu Zarządzania Zmianą (SZZ) można użyć dopiero po rozpoczęciu procesu wytwarzania – podczas przeprowadzania kolejnych, dokładniejszych planowań.

6.2.2. Etap pierwszy metody – wybranie projektu referencyjnego

Bardzo ważnym elementem metody jest właściwy dobór referencyjnego projektu informatycznego. Zmiennymi decyzyjnymi dla dokonania doboru jest przede wszystkim rozmiar wytwarzanego oprogramowania, następnie metodyka wytwarzania, wykorzystane dobre praktyki, techniki zapewniania jakości i technologie. Najlepszym źródłem danych do wyboru projektu referencyjnego jest baza danych projektów organizacji, która planuje proces wytwarzania. In-

nymi źródłami są np. komercyjne bazy danych projektów informatycznych lub informacje zamieszczone w literaturze. Wybór jest dokonywany przez decydenta, którym może być osoba (domyślnie kierownik projektu) lub regułowa maszyna wnioskująca.

6.2.3. Etap trzeci metody – dostrojenie modelu do projektu referencyjnego

Na podstawie danych wybranego projektu referencyjnego określane są zmienne wejściowe modelu AGOMO i przeprowadzane jest dostrojenie modelu. Dostrojenie modelu polega na dobraniu wartości parametrów wewnętrznych modelu tak, aby uzyskać zgodność struktury pracochłonności projektu referencyjnego i harmonogramu utworzonego przez model AGOMO. Zgodność struktury pracochłonności oznacza, że błąd względny sumy pracochłonności pogrupowanych wg kategorii zadań jest mniejszy lub równy 10%, natomiast błąd względny pracochłonności sumarycznej jest mniejszy lub równy 1%. Wynikiem prac w tym etapie są referencyjne wartości parametrów wewnętrznych modelu, które zostaną wykorzystane w następnym etapie metody.

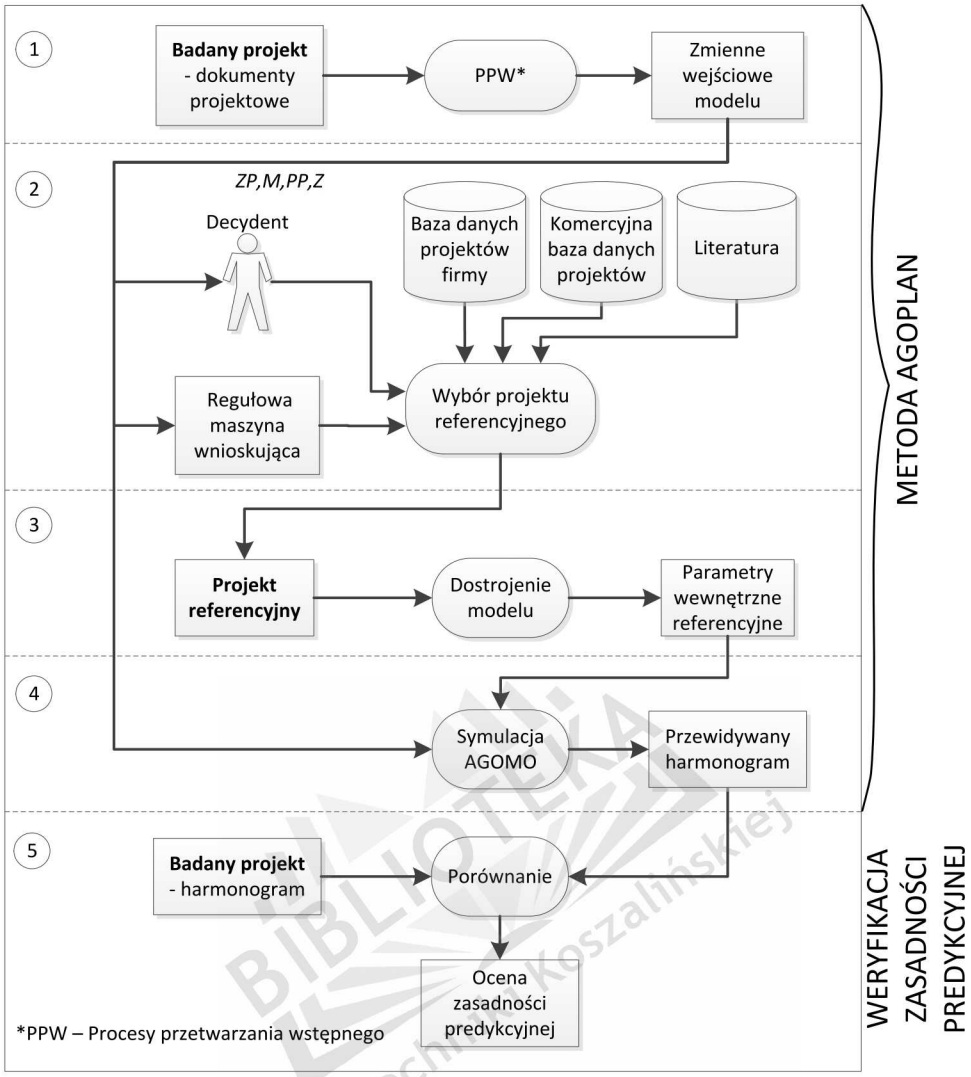
6.2.4. Etap czwarty metody – Symulacja procesu RUP

Symulacja procesu wytwarzania realizowanego zgodnie z metodyką RUP jest końcowym etapem metody AGOPLAN. Model AGOMO wykorzystuje wyznaczone na wcześniejszych etapach wartości zmiennych wejściowych dla planowanego projektu (zakres projektu, metodykę, plan projektu i skład zespołu projektowego) oraz referencyjne wartości parametrów wewnętrznych modelu. Wynikiem symulacji przeprowadzonej w środowisku wieloagentowym jest przewidywany harmonogram zadań dla zespołu projektowego, z którego można wnioskować o wąskich gardłach w procesie wytwarzania wynikających z nieodpowiedniego doboru zespołu projektowego. Analiza harmonogramu jest wspierana przez raport przewidywanego obciążenia pracą osób z zespołu projektowego oraz przez raport miesięcznych kosztów zatrudnienia zespołu projektowego, w tym przewidywanych nadgodzin.

6.3. Eksperyment badawczy w procesie weryfikacji prognostycznej

Celem zaplanowanego eksperymentu badawczego jest sprawdzenie zdolności predykcyjnej modelu AGOMO. Potwierdzenie tej zdolności jest równocześnie weryfikacją dla metody AGOPLAN – planowania procesów RUP.

Eksperyment został zaplanowany zgodnie z metodą AGOPLAN, przeznaczoną do wstępnego planowania procesów RUP i przedstawioną w poprzednim podrozdziale. Plan i etapy eksperymentu zostały pokazane na Rys. 6.2. Etapy 1–4 należą do metody AGOPLAN, dodatkowy etap 5 służy ocenie zasadności predykcyjnej modelu. Jako źródło danych do eksperymentu został użyty projekt UBEZP, wykorzystany w poprzednim rozdziale do przeprowadzenia weryfikacji zasadności replikatywnej modelu AGOMO. Badania prowadzone w kolejnych etapach i uzyskane wyniki zostały opisane w następnych podrozdziałach.



Rys. 6.2 Plan eksperymentu weryfikacji zasadności predykcyjnej modelu AGOMO

Źródło: Opracowanie własne

6.3.1. Etap pierwszy eksperymentu – określenie wartości zmiennych wejściowych modelu

Zmienne wejściowe modelu AGOMO są wykorzystane w eksperymencie do wyboru odpowiedniego projektu referencyjnego oraz do przeprowadzenia symulacji procesu wytwarzania oprogramowania w oparciu o model AGOMO. W pierwszym etapie ustalono wartości następujących zmiennych wejściowych modelu: zakresu projektu, metodyki wytwarzania, zespołu projektowego, planu projektu. Określenie wartości zmiennych wejściowych modelu przeprowadzono w sposób zbliżony, jak w eksperymencie weryfikacji replikatywnej modelu AGOMO. Jednak, ponieważ charakter eksperymentu wymaga, aby rzeczywisty przebieg przewidywanego projektu nie był znany, jako źródło danych wykorzystane zostały jedynie wstępne dokumenty projektowe, utworzone przed rozpoczęciem procesu wytwarzania. Wyjątkiem jest zmienna wejściowa zespół projektowy, która zawiera informacje o wszystkich zmianach w zaangażowaniu w projekt, odejściach z zespołu i przyjęciach nowych osób do zespołu projektowego.

6.3.2. Etap drugi eksperymentu – wybór projektu referencyjnego

Wiedza modelu na temat symulowanego projektu jest zawarta w parametrach wewnętrznych, określających czas przetwarzania artefaktów przez elementarne typy zadań. W celu sprawdzenia możliwości prognozowania przebiegu procesu wytwarzania, wartości parametrów wewnętrznych nie mogą być dostrojone do symulowanego projektu. Byłoby to sprzeczne z założeniami, ponieważ model powinien przewidzieć przebieg procesu wytwarzania bez żadnej wstępnej wiedzy o nim. Jak widać na Rys. 6.2 – etap 2, istnieją trzy źródła danych o projektach referencyjnych. Najlepszym rozwiązaniem byłoby użycie wartości parametrów wewnętrznych dostrojonych do innego projektu informatycznego, wykonanego w tej samej organizacji. Niestety, nie jesteśmy w posiadaniu takich danych, ponieważ organizacja dopiero od niedawna zaczęła w sposób gromadzić informacje o wykonanych projektach informatycznych, aby użyć ich do analizy celem przewidywania, ulepszania i rozwoju procesów wytwarzania i zarządzania. Drugim w kolejności możliwym rozwiązaniem problemu, jest wybór projektu referencyjnego z jednej z komercyjnych baz danych projektów lub z artykułów naukowych i dostrojenie wartości parametrów wewnętrznych modelu na ich podstawie.

W celu wybrania projektu informatycznego, który mógłby służyć jako referencyjny projekt informatyczny, wykorzystano bazę danych projektów informatycznych zbudowaną przez organizację ISBSG (International Software Benchmarking Standards Group) [119]. Baza danych ISBSG w wersji R12 zawiera dane 6006 projektów informatycznych z całego świata. Została zakupiona na zasadach współpracy badawczej, aby wesprzeć procesy weryfikacji modelu AGOMO. Początkowo jako kryterium wyszukania projektu referencyjnego przyjęto metodykę wytwarzania. Jednak w bazie ISBSG znajduje się jedynie 16 projektów wytworzonych w oparciu o metodykę RUP. Są to projekty informatyczne o małym rozmiarze, o pracochłonności kilkunastokrotnie mniejszej niż badany projekt. Eksperyment wykonany z jednym z tych projektów, przyjętym jako referencyjny, zakończył się fiaskiem. W kolejnym etapie wyszukiwania kryterium wyboru projektu referencyjnego zostało zmienione z metodyki wytwarzania na rozmiar projektu. Z bazy danych projektów informatycznych został wybrany największy dostępny projekt. Dane projektu ISBSG 17563 zostały przedstawione w Tab. 6.1. Pracochłonność wybranego projektu referencyjnego jest ponad dwukrotnie niższa niż pracochłonność badanego projektu, zespół projektowy jest również dwukrotnie mniej liczny, co jednak oznacza podobną skalę przedsięwzięcia i lepsze dopasowanie do badanego projektu. Pozostałe dane dotyczące projektu są również zbieżne. Generacja użytego języka oprogramowania jest identyczna, jak w badanym projekcie, co skutkuje podobną wydajnością pracy zespołu projektowego. Brak jest danych dotyczących zastosowanej metodyki wytwarzania, jednak rok rozpoczęcia projektu i jego rozmiar wskazują na to, że była to jedna z metodyk tradycyjnych.

Tab. 6.1 Podstawowe dane projektu ISBSG 17563 – projektu referencyjnego
Źródło: Opracowanie własne na podstawie [119]

Atrybut	Wartość
Identyfikator ISBSG	17563
Rok rozpoczęcia	2003
Sektor przemysłu	Finansowy
Typ aplikacji	Brak informacji
Metodyka	Brak informacji
Typ języka programowania	3GL
Podstawowy język	Visual Basic
Rozmiar [PF]	1803
Rozmiar [AGOMO–UCP]	1789
Pracochłonność [RH]	134 211
Wielkość zespołu [os.]	61
Czas trwania [mies.]	41

6.3.3. Etap trzeci eksperymentu – dostrojenie modelu AGOMO do wybranego projektu referencyjnego

Dostrojenie modelu do projektu referencyjnego wynika z potrzeby ustalenia parametrów wewnętrznych modelu AGOMO, niezbędnych do przeprowadzenia symulacji. W wybranym projekcie referencyjnym, podobnie jak w przypadku innych projektów z tej bazy, dostępna jest jedynie sumaryczna pracochłonność pogrupowana według dyscyplin. Na tej podstawie dostrojono parametry wewnętrzne modelu tak, aby w symulacji odtworzyć proces wytwarzania dla projektu referencyjnego ISBSG 17563. Porównanie pracochłonności otrzymanej w wyniku symulacji procesu wytwarzania oprogramowania w oparciu o model AGOMO z rzeczywistą pracochłonnością projektu ISBSG 17563, zsumowanej według dyscyplin umieszczono w Tab. 6.2. Jak widać w tabeli, w wyniku symulacji uzyskano wartości pracochłonności bardzo bliskie rzeczywistym, spełniające kryteria metody AGOPLAN.

Tab. 6.2 Porównanie wyników symulacji projektu ISBSG 17563 z danymi rzeczywistymi
Źródło: Opracowanie własne

Dyscyplina	Rzeczywistość [RBH]	Symulacja [RBH]	Błąd względny
Wymagania	19 408	19 469	0%
Projekt	23 436	23 556	1%
Program	46 109	46 090	0%
Testy	32 444	32 528	0%
Wdrażanie	10 742	10 391	3%
Zarządzanie	2 072	2 147	3%
razem	**134 211**	**134 181**	**0%**

6.3.4. Etap czwarty eksperymentu – wykonanie symulacji procesu wytwarzania oprogramowania opartego na modelu AGOMO

Celem wykonania symulacji procesu wytwarzania oprogramowania opartego na modelu AGOMO było uzyskanie harmonogramu planowanego procesu RUP. Do przeprowadzenia symulacji użyto zmiennych wejściowych modelu, ustalonych w pierwszym etapie eksperymentu oraz parametrów wewnętrznych dostrojonych do wybranego, referencyjnego projektu ISBSG 17563. Wynikiem przeprowadzonej symulacji jest harmonogram prac w projekcie, który w następnym etapie zostanie porównany z rzeczywistym procesem wytwarzania projektu UBEZP.

6.3.5. Etap piąty eksperymentu – porównanie wyniku symulacji z przebiegiem badanego projektu

Celem ostatniego etapu eksperymentu jest porównanie zaplanowanego procesu RUP z rzeczywistym procesem wytwarzania projektu UBEZP, aby umożliwić wnioskowanie, co do zasadności predykcyjnej modelu AGOMO. Porównanie zostało przeprowadzone w trzech aspektach: czasu trwania procesu, pracochłonności sumarycznej w kategoriach i przebiegu prac w poszczególnych kategoriach w czasie.

Porównanie czasu trwania procesu

Wynikiem porównania czasu trwania procesu jest zestawienie czasów trwania rzeczywistego procesu wytwarzania projektu UBEZP i procesu będącego wynikiem symulacji procesu wytwarzania, umieszczone w Tab. 6.3. Czasy trwania obydwu procesów są zbliżone do siebie. Różnica wynosi dwadzieścia dni, czyli błąd względny wynosi około 1%. Przyczyną tak małej różnicy w czasie trwania projektu symulowanego i rzeczywistego jest użycie jako zmiennej wejściowej dla symulacji procesu wytwarzania rzeczywistych informacji o zespole projektowym, wraz ze wszystkimi zmianami w liczebności i zaangażowaniu w prace projekcie.

Tab. 6.3 Porównanie czasów trwania badanego projektu z wynikiem symulacji
Źródło: Opracowanie własne

Proces	Data rozpoczęcia	Data zakończenia	Czas trwania	
			lata	mies.
Badany projekt UBEZP	2011–09–14	2016–03–23	4	5
Symulacja z param. ISBSG 14325	2011–09–14	2016–03–03	4	5

Porównanie pracochłonności sumarycznej w kategoriach

Drugim aspektem jest porównanie pracochłonności sumarycznej według kategorii. Porównanie pracochłonności planowanego procesu RUP z pracochłonnością projektu UBEZP, zostało zamieszczone w Tab. 6.4. W tabeli zestawiono pracochłonności sumaryczne zakończonego projektu, pogrupowane według kategorii. Dwie pierwsze kolumny zawierają wartości dla badanego projektu i wyniku symulacji. W trzeciej kolumnie umieszczono wartość procentową błędu względnego dla wartości zamieszczonych w pierwszych dwu kolumnach. W ostatnim wierszu tabeli zamieszczono sumaryczną pracochłonność dla wszystkich kategorii.

Błąd względny dla pracochłonności sumarycznej wynosi 1,88%. Główną przyczyną tak wysokiej różnicy w wartości pracochłonności jest odmienna klasa projektu referencyjnego i bada-

nego projektu UBEZP. Projekt referencyjny był projektem nowym, rozpoczętym od zera. Natomiast projekt UBEZP był tworzony na podstawie odziedziczonej wersji systemu zaimplementowanej w przestarzałym środowisku informatycznym o odmiennej architekturze. Prace w projekcie UBEZP rozpoczęły się od odtworzenia przypadków użycia, cech i logiki biznesowej z odziedziczonego, działającego systemu. W projekcie referencyjnym nie prowadzono tego typu prac, stąd różnice w pracochłonności sumarycznej.

Tab. 6.4 Wyniki symulacji procesu wytwarzania projektu UBEZP, przy użyciu parametrów wewnętrznych referencyjnego projektu ISBSG 17563

Źródło: Opracowanie własne

Kategoria	Badany projekt [RBH]	Symulacja [RBH]	Błąd względny [%]
Projekt architektury	1 291	1 184	8,29%
Prototyp architektury	3 038	2 781	8,46%
Integracja systemu	2 955	2 177	26,33%
Zarządzanie konfiguracją	5 158	7 357	29,89%
Implementacja	125 355	124 311	0,83%
Projektowanie	17 531	27 319	35,83%
Projekt analityczny	8 511	12 974	34,40%
Projekt bazy danych	5 921	7 413	20,13%
Projekt interfejsu graficznego	7 585	13 978	45,74%
Testowanie	84 969	88 959	4,49%
Model przypadków użycia	33 229	32 181	3,15%
Warsztaty wymagań i wywiady	4 297	5 364	19,89%
Zarządzanie projektem	17 082	5 308	68,93%
Wdrożenie systemu	28 789	21 046	26,90%
Razem	**345 711**	**352 352**	**1,88%**

Porównanie przebiegu prac w poszczególnych kategoriach

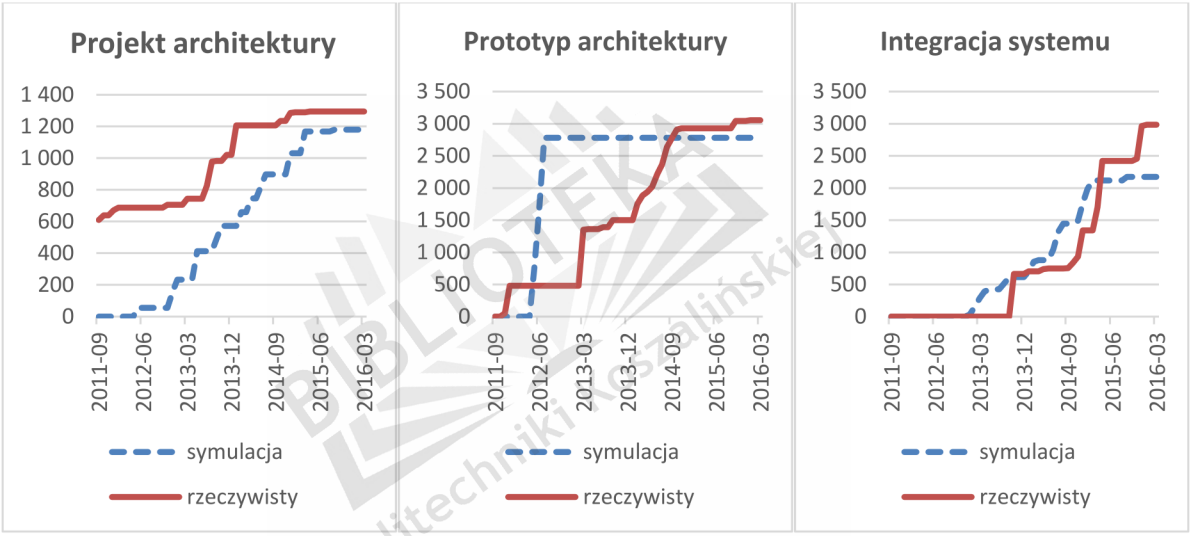
Trzecim aspektem porównania jest porównanie przebiegu prac w poszczególnych kategoriach w czasie. Największe różnice w pracochłonności występują w kategorii zarządzania projektem i projektowaniu. Pracochłonność zadań w kategorii zarządzania jest ponad trzykrotnie wyższa w rzeczywistym projekcie UBEZP niż w planach wynikających z przeprowadzonych symulacji. Tak duża różnica prawdopodobnie wynika z dużo większego zakresu projektu UBEZP od zakresu projektu referencyjnego. Prawdopodobnym jest, że wysiłek włożony w zarządzanie projektem nie wykazuje liniowej zależności od zakresu projektu. Przy dużych projektach informatycznych mogą występować problemy, których nie ma w projektach o mniejszej skali.

Kolejną kategorią, co do wielkości błędu względnego w pracochłonności, jest projektowanie. Zaplanowana pracochłonność zadań związanych z projektowaniem jest półtora razy większa niż rzeczywista pracochłonność projektowania w projekcie UBEZP. Występują dwie przyczyny tej różnicy. Pierwszą jest to, że projekt wydruków i formularzy był odzyskany z poprzed-

niej wersji oprogramowania. Drugą przyczyną jest fakt, że system informatyczny był przygotowywany przez konsorcjum dwóch organizacji, z których ta druga przygotowywała część projektu i integrowała całość systemu.

Ostatnią kategorią obciążoną dużym błędem względnym jest wdrożenie systemu. W tym wypadku plany wyznaczone przez symulację modelu są niedoszacowane względem rzeczywistej pracochłonności. Pracochłonność w badanym projekcie jest wyższa, ponieważ wdrożenie odbywało się na działającym systemie informatycznym klienta i błędy w funkcjonalności nowego systemu powodowały uszkodzenia złożonych struktur danych w bazach danych klienta, których naprawa była niezbędna i zarazem wymagała dużych nakładów pracy.

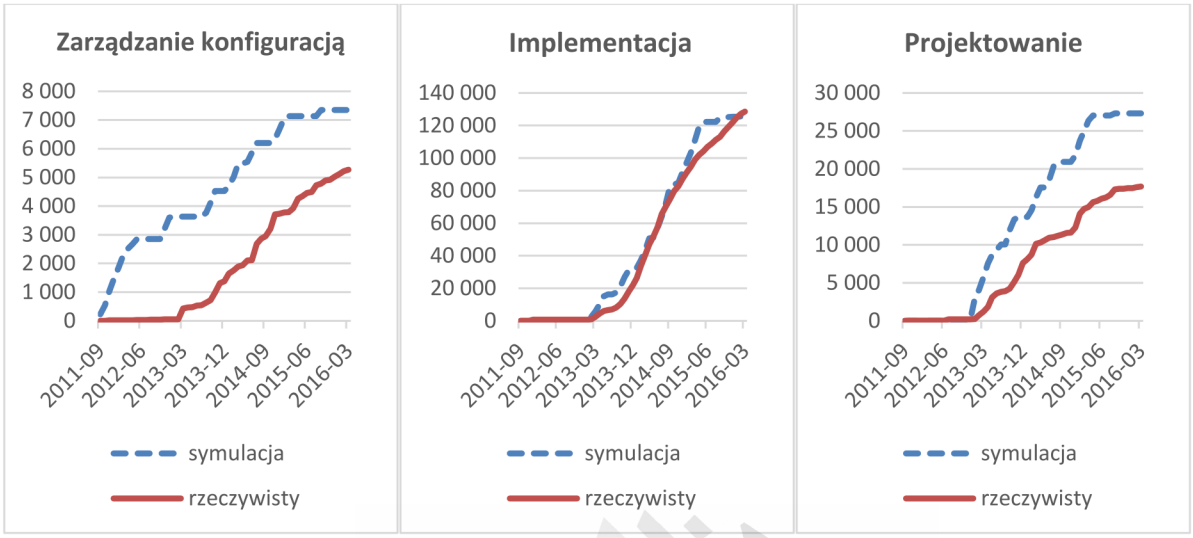
Poniżej, na serii pięciu rysunków, zaprezentowano porównanie rzeczywistej pracochłonności w czasie trwania procesu wytwarzania z pracochłonnością zaplanowaną dzięki symulacji procesu wytwarzania w oparciu o model AGOMO. Wartość pracochłonności na wykresach jest podana w roboczogodzinach.



Rys. 6.3 Porównanie przyrostów pracochłonności w czasie, według kategorii, część 1
Źródło: Opracowanie własne

W pierwszym wykresie umieszczonym na Rys. 6.2 widzimy, że prace nad projektem architektury w rzeczywistym procesie wytwarzania rozpoczynają się od początku trwania procesu. W symulacji prace nad projektem architektury rozpoczynają się dopiero po kilku miesiącach trwania projektu. Jest to zgodne z procesem RUP, w którym projekt architektury wymaga rozpoznania kluczowych przypadków użycia i wymagań niefunkcjonalnych systemu informatycznego. W rzeczywistości jednak, ponieważ głowne funkcjonalności systemu są znane w ogólnym zarysie, a nowa wersja systemu jest tworzona głównie z potrzeby unowocześnienia architektury, prace na projektem architektury rozpoczynają na początku trwania projektu. Podobna sytuacja występuje na kolejnym wykresie porównującym prace nad prototypem architektury. W symulacji prace nad prototypem są odsunięte w czasie o kilka miesięcy, do momentu pozyskania odpowiedniej ilości wymagań, natomiast w rzeczywistości prace rozpoczęły się na początku procesu wytwarzania. W symulacji prace nad prototypem są wykonane jednorazowo w początkowym etapie procesu wytwarzania, w rzeczywistości jednak prace były prowadzone przyrostowo. Prototyp był udoskonalany podczas całego cyklu życia oprogramowania. Rzeczywista i zaplanowana sumaryczna pracochłonność zadań związanych

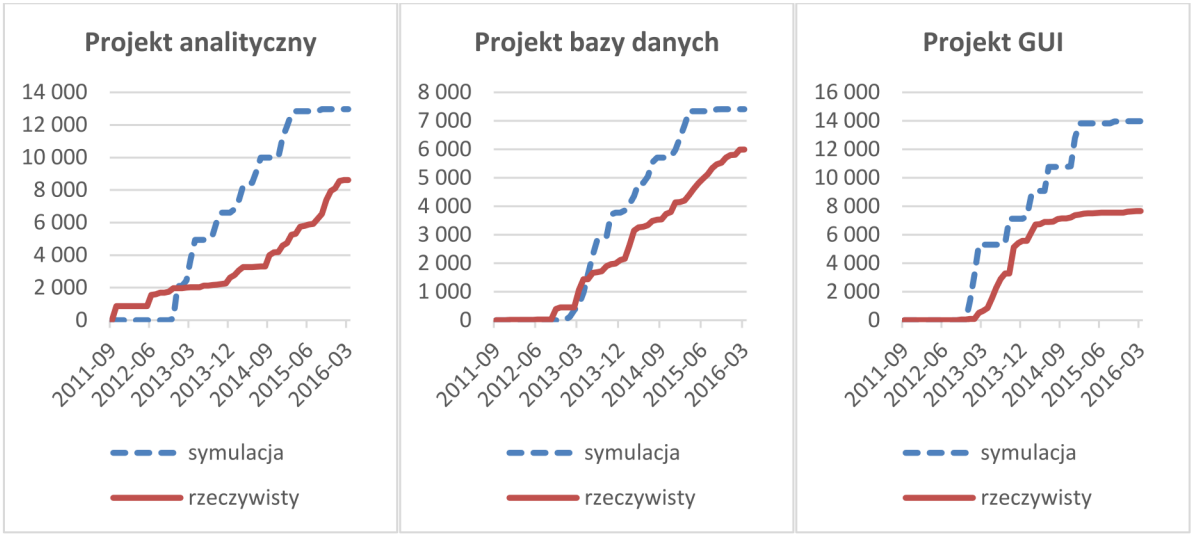
z opracowaniem architektury jest zbliżona do siebie. Na ostatnim wykresie umieszczonym na Rys. 6.2 zamieszczono wykres prac w kategorii integracji systemu, które w rzeczywistości są zbliżone wartością i przebiegiem do zaplanowanych.



Rys. 6.4 Porównanie przyrostów pracochłonności w czasie, według kategorii, część 2

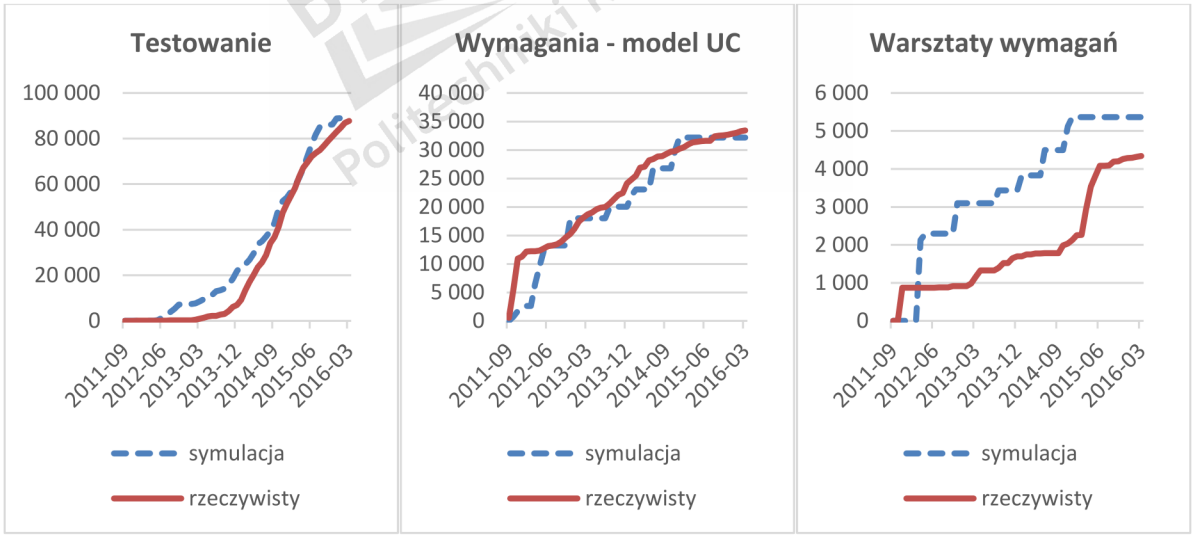
Źródło: Opracowanie własne

Drugą część porównania przepływu prac wg kategorii pokazano na Rys. 6.4. Jak widać na pierwszym wykresie, prace nad zarządzaniem konfiguracją w rzeczywistym procesie wytwarzania rosną bardzo wolno i rozpoczynają się z rocznym opóźnieniem względem początku projektu. Zaplanowana pracochłonność zadań w kategorii implementacji zarówno narasta, jak i osiąga wartość końcową zbliżoną do rzeczywistej. Ma to bardzo istotne znaczenie, ponieważ pracochłonność implementacji ma najwyższą wartość ze wszystkich kategorii, co mocno wpływa na zgodność planu z rzeczywistymi pracami, a co za tym idzie na zasadność predykcyjną modelu AGOMO. Prace zaplanowane w kategorii projektowania mają w dużo większą pracochłonność, niż wykonane w rzeczywistości. Przyczyny takiego stanu rzeczy – odzyskiwanie części projektu z poprzedniej wersji systemu i przygotowanie projektu przez zewnętrzną firmę – zostały omówione w opisie wyników zamieszczonych w Tab. 6.4.



Rys. 6.5 Porównanie przyrostów pracochłonności w czasie, według kategorii, część 3
Źródło: Opracowanie własne

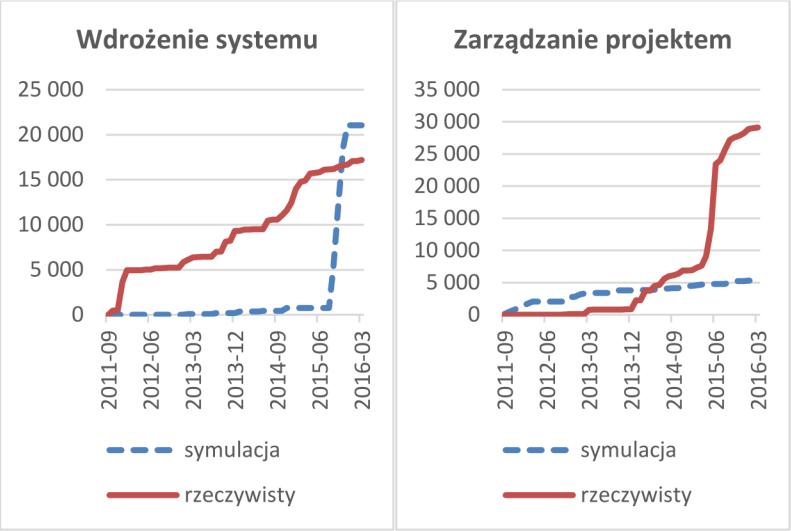
W trzeciej części porównania przepływu prac między planem projektu a rzeczywistym przebiegiem, pokazanym na Rys. 6.5 znalazły się trzy wykresy szczegółowych kategorii związanych z projektowaniem. Są to projektowanie analityczne, projektowanie bazy danych oraz interfejsu użytkownika. Na wszystkich trzech wykresach widać, że zaplanowana pracochłonność jest dużo wyższa, niż rzeczywista, jednak kształt wykresu, a więc też wykonywanie prac planowanych i rzeczywistych są do siebie zbliżone. Rzeczywiste projektowanie analityczne zaczyna się wcześniej niż planowane, które rozpoczyna się dopiero po zebraniu odpowiedniej ilości wymagań.



Rys. 6.6 Porównanie przyrostów pracochłonności w czasie, według kategorii, część 4
Źródło: Opracowanie własne

W czwartej części porównania zaplanowanej pracochłonności z rzeczywistą, na Rys. 6.6, umieszczono wykresy porównania pracochłonności w kategoriach testowania oraz pozyskiwania wymagań. Zaplanowana pracochłonność, a co za tym idzie przebieg prac, zarówno w kategorii testowania, jak i pozyskiwania wymagań są zbliżone do rzeczywistych. Jest to bardzo

istotne, ponieważ prace w tych kategoriach mają istotny wpływ na dokładność przewidywania ze względu na wysokie wartości pracochłonności.



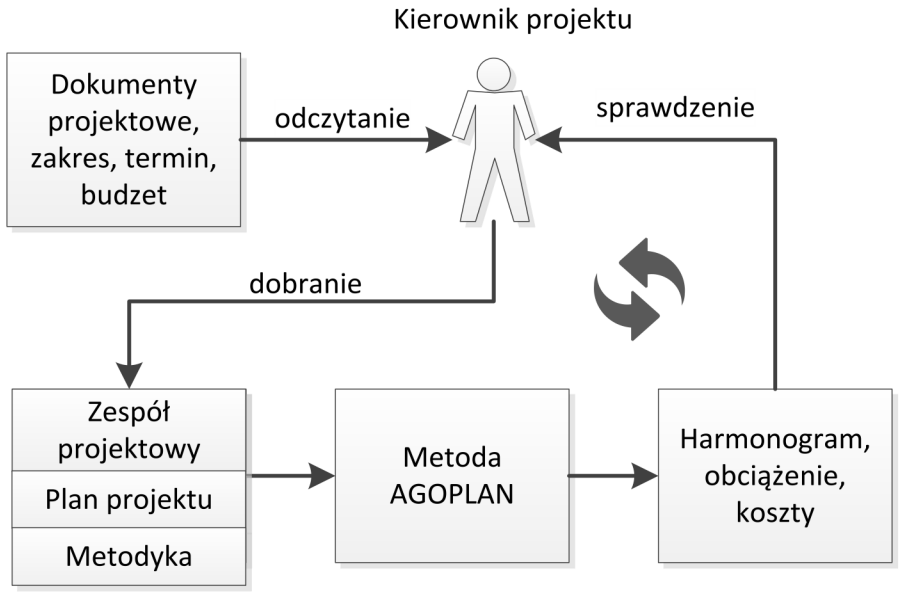
Rys. 6.7 Porównanie przyrostów pracochłonności w czasie, według kategorii, część 5
Źródło: Opracowanie własne

W ostatniej części porównania pracochłonności zaplanowanej z rzeczywistą, umieszczonej na Rys. 6.7, znajdują się wykresy porównania dla kategorii wdrożenia systemu i zarządzania projektem. Zaplanowana pracochłonność, wynikająca z planowanego przebiegu prac w kategorii wdrożenia systemu mocno różni się od rzeczywistego przebiegu prac w projekcie UBEZP. Plan prac zakładał, zgodnie z metodyką RUP, że testy alfa i beta systemu odbędą się dopiero w etapie wdrożenia systemu. W rzeczywistym projekcie prace wdrożeniowe odbywały się we wszystkich iteracjach procesu wytwarzania. Końcowa wartość zaplanowanej pracochłonności w jest w stosunku do rzeczywistej obarczona błędem względnym o wartości 27%. Taka rozbieżność sugeruje konieczność lepszego dostosowania metodyki RUP do projektu. Dużo większym błędem względnym jest obarczona pracochłonność w kategorii zarządzania projektem. Rzeczywista pracochłonność jest około sześciokrotnie większa niż zaplanowana. Ta rozbieżność, jak zostało to omówione wcześniej, może wynikać z nieliniowości w zależności pracochłonności od zakresu/rozmiaru projektu. Projekt referencyjny, według którego zaplanowano proces wytwarzania projektu UBEZP, jest od niego dwukrotnie mniejszy.

6.4. Podsumowanie rozdziału – znaczenie procesów weryfikacji prognostycznej modelu AGOMO

W rozdziale opisano proces weryfikacji zasadności prognostycznej modelu AGOMO. Weryfikacja miała na celu sprawdzenie zdolności modelu AGOMO do prognozowania procesu wytwarzania oprogramowania nieznanego modelowi systemu rzeczywistego. Eksperyment mający na celu weryfikację został zaplanowane w oparciu o AGOPLAN – metodę planowania procesów RUP opisaną w podrozdziale 6.2. Plan eksperymentu został rozszerzony względem metody o dodatkowy etap, w którym dokonano porównania zaplanowanego procesu wytwarzania z rzeczywistym procesem wytwarzania projektu UBEZP. Podczas eksperymentu wybrano odpowiedni projekt referencyjny, do którego dostrojono model, aby uzyskać referencyjne wartości parametrów wewnętrznych modelu. Zmienne wejściowe o wartościach określonych na podstawie dokumentów projektowych i referencyjne parametry wewnętrzne modelu zostały

użyte do wykonania symulacji, w wyniku której uzyskano plan procesu wytwarzania. Następnie plan został porównany w trzech aspektach z rzeczywistym procesem wytwarzania projektu UBEZP. Czas trwania zaplanowanego i rzeczywistego procesu różniły się jedynie o 1%. Całkowita pracochłonność zaplanowanego i rzeczywistego procesu obarczona była błędem względnym 1,88%. Przepływ prac w planowanym i rzeczywistym procesie wytwarzania w najbardziej znaczących kategoriach – tych o największej pracochłonności, były bardzo do siebie zbliżone. Wyniki porównania planu procesu wykonanego w oparciu o metodę AGOPLAN z rzeczywistym procesem wytwarzania prowadzą do wniosku o potwierdzeniu przez eksperyment zasadności predyktywnej modelu AGOMO. Powyższy wniosek potwierdza również przydatność metody AGOPLAN do planowania procesów RUP dla projektów informatycznych.
Planowanie procesów RUP ma duże znaczenie praktyczne i potencjał komercyjny. Metoda AGOPLAN została zweryfikowana przez powyższy eksperyment dla metodyki RUP, ze względu na ograniczony dostęp do danych projektów informatycznych oraz ograniczony czas badań i pracy nad modelem AGOMO. Prace nad modelem mogą i powinny być kontynuowane. Wprowadzony w modelu AGOMO opis metodyki w postaci grafów działań, składających się z atomowych zadań jest zapisem ogólnym i uniwersalnym, umożliwiającym nie tylko konfigurację i dostosowanie metodyki RUP, ale także użycie modelu AGOMO do planowania procesów wytwarzania prowadzonych według innych metodyk, również zwinnych.
Koncepcja użycia metody AGOPLAN do planowania procesu wytwarzania i dobrania elementów środowiska projektowego została przedstawiona na Rys. 6.8. Planowanie przy użyciu metody AGOPLAN jest wykonywane przez kierownika projektu. Kierownik projektu po odczytaniu dokumentów projektowych odpowiednio dobiera zespół projektowy, określa plan projektu i dostosowuje metodykę wytwarzania do specyfiki projektu. Następnie, po tak przeprowadzonym ustaleniu wartości zmiennych wejściowych modelu, przeprowadzana jest symulacja procesu wytwarzania. Wynikiem symulacji jest harmonogram planowanego procesu wytwarzania, informacja o obciążeniu zespołu projektowego pracą i koszty zatrudnienia pracowników, w tym koszty przepracowanych nadgodzin. Po sprawdzeniu wyników symulacji i porównaniu z ograniczeniami projektu, zawartymi w dokumentach projektowych, kierownik projektu odpowiednio zmienia i dobiera wartości zmiennych wejściowych modelu przed kolejnym uruchomieniem symulacji. Praca nad planowaniem procesu wytwarzania odbywa się przez wykonywanie cyklu dobierania wartości, symulacji i porównywania. Cykle wykonywane są przez kierownika projektu do uzyskania satysfakcjonującego wyniku.



Rys. 6.8 Zastosowanie metody AGOPLAN do planowania procesu wytwarzania i doboru środowiska projektowego

Źródło: Opracowanie własne

Przy użyciu cykli planowania można dobrać osoby wchodzące w skład zespołu projektowego pod względem kompetencji i przypisać im odpowiednie role tak, aby obciążenie pracą było zbliżone do wymaganego i nie powstawały wąskie gardła w procesie wytwarzania. Raport kosztów procesu wytwarzania pozwala na sprawdzenie, czy nie zostanie przekroczony budżet przedsięwzięcia. Wynikowy harmonogram nie tylko pozwala kontrolować termin zakończenia projektu, ale razem z raportem obciążenia pracą, może być podstawą do zsynchronizowania dwóch lub więcej procesów wytwarzania w organizacji. Przedstawione cykle planowania są przydatne nie tylko przed rozpoczęciem procesu wytwarzania oprogramowania. Dodatkowe cykle planowania można wykonać po istotnych zmianach w zakresie projektu, wymaganiach, składzie zespołu projektowego lub innych.

Istniejący system umożliwiający zastosowanie metody oceny i planowania środowiska wytwarzania oprogramowania i procesu wytwarzania oprogramowania dzięki użyciu modelu AGOMO może zostać zintegrowany z systemami zarządzania projektami informatycznymi. Integracja z istniejącymi systemami zapewniłaby dostęp do baz danych zakończonych projektów jako źródła danych referencyjnych dla metody AGOPLAN. Większość informacji dotyczących badanego / planowanego projektu również możne zostać zaczerpniętych z bazy danych głównego systemu zarządzania projektami informatycznymi. Udostepnienie modelu AGOMO i utworzonych na jego podstawie metod przy pomocy aplikacji zintegrowanej z systemem zarządzania procesami wytwarzania oprogramowania zapewni ciągły rozwój modelu i metod przez zwiększenie przepływu informacji i zapewnienie współpracy między jednostkami badawczymi a przemysłem.

7. Podsumowanie i wnioski

Celem badawczym niniejszej rozprawy była budowa modelu obiektowo–agentowego do symulacji procesu wytwarzania oprogramowania oraz opracowanie metod oceny i planowania procesu wytwarzania oprogramowania oraz środowiska projektowego.

Opracowanie modelu rozpoczęto od analizy systemu rzeczywistego w celu ustalenia zmiennych wejściowych i wyjściowych opisujących układ eksperymentu. Analiza procesów wytwarzania realizowanych według różnych metodyk wykazała, że można zdefiniować zmienną wejściową określającą plan projektu tak, aby obejmowała plany projektów realizowanych przy pomocy różnych metodyk wytwarzania. Podobnie reprezentacja zmiennej wejściowej metodyka wytwarzania, przy pomocy grafu skierowanego, którego węzły są działaniami składającymi się z elementarnych typów zadań, umożliwiła opisanie dowolnej z rozpatrywanych metodyk. Po analizie obydwu koncepcji wykorzystania modelu, ustalono zmienną wyjściową w postaci harmonogramu prac wykonanych w procesie wytwarzania. Dzięki temu opracowany układ eksperymentu może być wykorzystany zarówno do oceny, jak i do planowania procesu wytwarzania oprogramowania.

Ustalenie wspólnego układu eksperymentu umożliwiło opracowanie modelu podstawowego, który składa się z rdzenia realizującego podstawowe funkcje oraz rozszerzeń modelujących dodatkowe aspekty procesu wytwarzania oprogramowania. Dla elementów rdzenia modelu określono strukturę w zapisie formalnym oraz przeanalizowano występujące interakcje między elementami i podano dla nich algorytmy sterujące w postaci pseudokodu. Analiza dziedzinowa ograniczyła model podstawowy do zakresu ustalonego przez ramy układu eksperymentu, co dało podstawy do budowy modelu uproszczonego. Na podstawie przeglądu metod modelowania oraz analizy wymagań modelu podstawowego wybrano metodę agentowo–obiektową jako najlepiej spełniającą wymagania modelu uproszczonego i nie ograniczającej możliwości zmiany układu eksperymentu w zakresie przedstawionego modelu podstawowego. Przeprowadzone rozważania dotyczące zakresu i przeznaczenia modelu, potwierdziły zdolność modelu agentowo–obiektowego do wieloaspektowego odwzorowania pracy zespołu projektowego i przebiegu procesu wytwarzania. Następnie przedstawiono koncepcje weryfikacji modelu, które posłużyły w dalszych częściach pracy do przeprowadzenia eksperymentów weryfikujących zasadność modelu.

Aby umożliwić konstrukcję, model uproszczony został przedstawiony najpierw ogólnie, a następnie szczegółowo za pomocą opisu formalnego. Elementy aktywne, jak osoby wchodzące w skład zespołu projektowego są reprezentowane przez agenty. Agenty zostały opisane jako zbiory automatów skończonych o zawartości zmieniającej się w zależności od potrzeb. Automaty przedstawiono w postaci diagramów stanów UML. Do sterowania automatami zostały wykorzystane algorytmy opracowane w rozdziale trzecim dla rdzenia modelu podstawowego. Elementy pasywne, umieszczone w środowisku agentowym, również zostały przedstawione w zapisie formalnym. Opis formalny modelu został zweryfikowany przez implementację w systemie wieloagentowym JADE, co pokazuje możliwość opisu agentów za pomocą dynamicznych zbiorów automatów skończonych przetwarzających artefakty. Wynikowy, agen-

towo–obiektowy model został nazwany AGOMO. System wieloagentowy JADE, wraz ze środowiskiem i agentami, został wykorzystany jako element systemu umożliwiającego zastosowanie metody oceny i planowania środowiska wytwarzania oprogramowania i procesu wytwarzania oprogramowania. Głównym elementem systemu jest aplikacja AGOSIM–APP, zbudowana, aby wprowadzać wartości zmiennych wejściowych, przeprowadzać symulację modelu oraz prezentować wyniki w postaci diagramów, wykresów i raportów. Aplikacja została opisana w dodatku B. Na podstawie tak zweryfikowanego modelu uproszczonego i przedstawionych w rozdziale pierwszym koncepcji, opracowano metodę oceny procesów wytwarzania oprogramowania wg metodyki RUP, nazwaną AGOMAS.

Aby zbadać zasadność replikatywną modelu AGOMO zaprojektowano i przeprowadzono eksperyment. Plan eksperymentu został oparty na metodzie AGOMAS, rozszerzonej o etap weryfikacji replikatywnej. W ramach przeprowadzonego eksperymentu przedstawiono badaną organizację i scharakteryzowano badany projekt, a następnie przeprowadzono pięć etapów eksperymentu. W wyniku eksperymentu model AGOMO z dużą dokładnością odtworzył przebieg badanego projektu informatycznego, co potwierdziło zasadność replikatywną modelu. Utworzony przez model harmonogram działań został użyty w metodzie AGOMAS jako wzorzec procesu do porównania z harmonogramem badanego projektu. W wyniku przeprowadzonej oceny procesu wytwarzania uzyskano wysoką wartość wskaźnika zgodności. Zgodnie z wcześniejszymi założeniami, otrzymana wartość wskaźnika została zinterpretowana jako wysoka dojrzałość procesu wytwarzania oprogramowania. Otrzymane wyniki oceny mogą sugerować gotowość organizacji do zwinnej transformacji. Rzeczywista gotowość organizacji do transformacji została później potwierdzona przez udaną transformację badanej organizacji.

Celem weryfikacji zdolności modelu AGOMO do prognozowania procesów wytwarzania oprogramowania, opracowano metodę AGOPLAN, przeznaczoną do planowania procesów RUP. Następnie, stosując przedstawioną metodę, przeprowadzono eksperyment mający na celu weryfikację zasadności predykcyjnej modelu AGOMO i poprawności opracowanej metody. Wyniki porównania planu procesu wykonanego w oparciu o metodę AGOPLAN z rzeczywistym procesem wytwarzania prowadzą do wniosku o potwierdzeniu przez eksperyment zasadności predyktywnej modelu AGOMO. Powyższy wniosek potwierdza również przydatność metody AGOPLAN do planowania procesów RUP projektów informatycznych.

Poniżej przedstawiono wady i zalety modelu AGOMO.

Zalety modelu

1. Szczegółowość i dokładność wynikająca z mikroskali. Rdzeń modelu odtwarza rzeczywisty proces wytwarzania na poziomie mikromodelu i mikrosymulacji. Symulacja polega na wykonywaniu elementarnych, atomowych zadań procesu wytwarzania. Prowadzi to do pojawiania się, na zasadzie emergencji, w wynikach symulacji złożonych zjawisk, które odpowiadają zjawiskom rzeczywistym.
2. Niewielka zależność od arbitralnych, analitycznych wzorów i modeli.
3. Oparty na standardowej metodzie szacowania. Metoda Use Case Points jest rozpowszechniona od połowy lat 90–dziesiątych i chętnie stosowana do szacowania zakresu projektów stosujących technologie obiektowe.

4. Uniwersalny opis metodyki wytwarzania. Opis metodyki pozwala na modelowanie procesu RUP dostosowanego do badanego projektu. W założeniu zastosowany opis metodyki pozwala także na modelowanie innych niż RUP metodyk wytwarzania, w tym metodyk zwinnych.
5. Zastosowanie technologii agentowej. Technologia agentowa pozwala na analizę wpływu cech osobowości członków zespołu projektowego na komunikację i wydajność zespołu projektowego i co za tym idzie procesu wytwarzania.
6. Skalowalność implementacji. Zastosowanie technologii agentowej wiąże się również z przetwarzaniem wielowątkowym i możliwością wykonywania symulacji w postaci obliczeń rozproszonych w klastrach komputerowych o architekturze GRID.

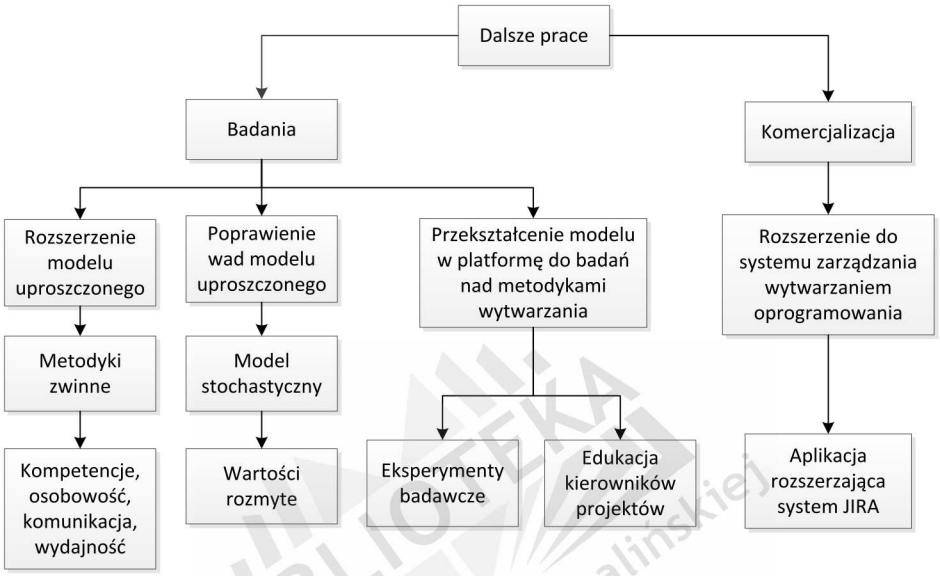
Wady modelu:

1. Brak odwzorowania zależności między elementami składowymi projektu informatycznego, np. projekt bazy danych modułu powinien powstać przed warstwą interfejsu do bazy danych i przed warstwą logiki biznesowej, która przetwarza dane. Podobnie w modelu brakuje reprezentacji zależności między modułami projektu. Moduły, które są podstawą do działania innych modułów są w rzeczywistości projektowane i konstruowane przed modułami, które z nich korzystają.
2. Niewykorzystane możliwości agentów do modelowania cech odróżniających osoby z zespołu projektowego, jak kompetencje i cechy osobowości, wpływające na skuteczność komunikacji i wydajność pracy.
3. Determinizm modelu. Determinizm opiera się na uśrednionych wartościach wydajności pracy dla całego zespołu projektowego, bez względu na okoliczności. Mimo przyjętego założenia o determinizmie, wyniki symulacji różnią się między sobą z powodu uruchamiania agentów jako osobne wątki w systemie przetwarzania równoległego. Dzięki zaawansowanym mechanizmom synchronizacji, różnice między wynikami symulacji są zostały zmniejszone do ok. 1 promila, bez znaczącej utraty wydajności.
4. Brak analizy statystycznej. Wydajność symulacji procesów wytwarzania oparciu o model AGOMO jest niewystarczająca do przeprowadzenia analizy statystycznej. Symulacja projektu UBEZP na średnie klasie komputerze osobistym trwa ok. jednej godziny.

7.1. Dalsze prace nad modelem

Dalsze prace prowadzone na podstawie niniejszej rozprawy mogą rozwijać się w stronę dalszych badań nad modelem lub zmierzać do utworzenia komercyjnego produktu wspierającego zarządzanie wytwarzaniem oprogramowania, co pokazano na Rys. 7.1. Przedstawione tam kierunki dalszych badań mają trzy źródła. Pierwsze źródło to badania nad włączaniem do modelu uproszczonego elementów rozpoznanych podczas prac nad modelem pełnym. Jest to przede wszystkim rozszerzenie opisu metodyki w sposób, który umożliwiałby symulację procesów zwinnych metodyk wytwarzania. Za tym rozszerzeniem modelu uproszczonego mogłoby pójść odwzorowanie cech indywidualnych osób wchodzących w skład zespołu projektowego. Włączenie w analizę cech osobowości, miękkich i twardych kompetencji, umożliwiłoby prace nad odpowiednim doborem składu zespołu projektowego w celu usprawnienia komunikacji, współpracy i poprawienia wydajności zespołów. Drugie źródło tematów dalszych badań jest motywowane usunięciem wad modelu,

zamieszczonej na liście wad. Badania prowadzone w tych kierunkach miałyby na celu usunięcie opisanych wad. Najciekawszą drogą dalszych prac w tym kierunku byłoby zmiana deterministycznego modelu w stochastyczny dzięki użyciu wartości rozmytych do określenia czasów wykonywania elementarnych zadań, co pozwoliłoby na przeprowadzenie analizy statystycznej wyników symulacji procesu wytwarzania w oparciu o model. Dzięki tej analizie kierownik projektu uzyskałby informacje m.in. o prawdopodobieństwie ukończenia prac w terminie, możliwych różnicach w kosztach zaplanowanych prac, co ułatwiłoby zarządzanie ryzykiem w projekcie.



Rys. 7.1 Wskazane kierunki dalszych prac nad modelem AGOMO

Źródło: Opracowanie własne

Trzecią drogą rozwoju prac badawczych nad modelem jest przekształcenie modelu w wirtualne środowisko do przeprowadzenia eksperymentów z dziedziny zarządzania procesami wytwarzania. Eksperymenty przeprowadzone w organizacjach niosą za sobą zbyt duże ryzyko niepowodzenia, co blokuje rozwój badań w tej dziedzinie. Wirtualne środowisko badań nad procesami wytwarzania umożliwiłoby prace nad nowymi podejściami do wytwarzania oprogramowania oraz otwierałoby drogę do kształcenia nowych menedżerów dla przemysłu.

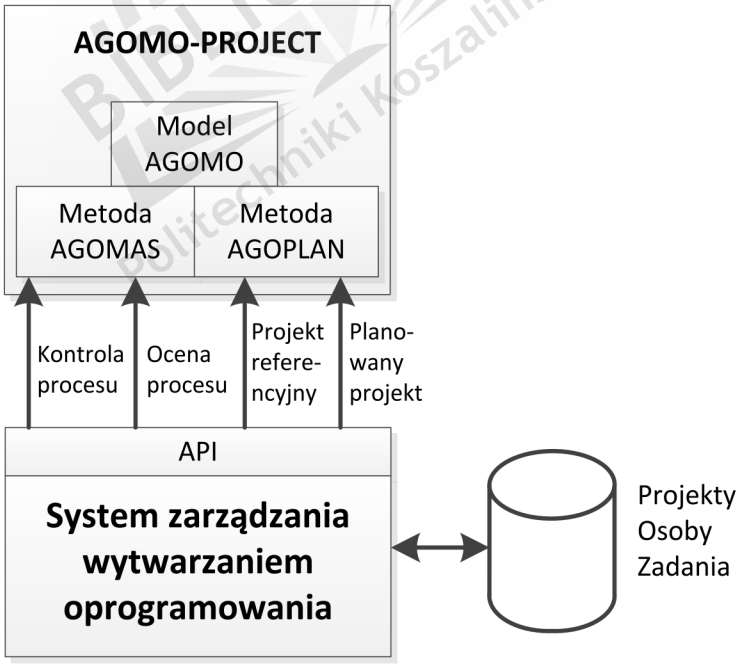
Opracowany model AGOMO oraz oparte na modelu metody AGOMAS i AGOPLAN mogą zostać wykorzystane do zarządzania dojrzałością procesu wytwarzania oprogramowania, co prowadzi do uzyskania wysokiej gotowości organizacji do zwinnej transformacji. Zarządzanie dojrzałością procesu polega na użyciu opracowanych metod we wszystkich fazach życia oprogramowania. Koncepcja użycia metod została przedstawiona w Tab. 7.1.

Tab. 7.1 Zastosowanie metod AGOMAS i AGOPLAN do wsparcia zarządzania projektami informatycznymi

Źródło: Opracowanie własne

Faza projektu	Metoda	Cel i wynik
Planowanie	AGOPLAN	Użycie metody AGOPLAN do odpowiedniego dobrania zespołu projektowego, planu projektu do danego trójkąta ograniczeń. Wynikiem jest plan procesu wytwarzania.
Wytwarzanie	AGOMAS	Sprawdzenie zgodności badanego procesu wytwarzania z przyjętym planem. Wynikiem jest wartość współczynnika zgodności. Zapewnia możliwość analizy przyczyn niezgodności.
Wytwarzanie	AGOPLAN	W przypadku zmiany zespołu projektowego, zakresu projektu lub konfiguracji metodyki umożliwia korektę procesu. Wynikiem jest nowy plan procesu wytwarzania.
Zakończenie	AGOMAS	Ocena procesu wytwarzania, która ma potencjalny wpływ na ocenę gotowości organizacji do zwinnej transformacji.

Koncepcja zarządzania dojrzałością procesów wytwarzania oprogramowania może znaleźć zastosowanie komercyjne. Komercjalizacja modelu AGOMO jest rozważana jako umieszczenie implementacji modelu AGOMO, razem z modułami implementującymi metodę oceny dojrzałości projektu AGOMAS i metodę planowania procesu RUP AGOPLAN w jednym systemie o nazwie AGOMO–PROJECT, który powinien być połączony z systemem zarządzania wytwarzaniem oprogramowania. Koncepcja połączenia jest pokazana na Rys. 7.2.



Rys. 7.2 System AGOMO–PROJECT połączony z systemem zarządzania wytwarzaniem oprogramowania

Źródło: Opracowanie własne

Dostęp systemu AGOMO–PROJECT do bazy danych systemu zarządzania zostanie zrealizowany przez interfejs API, co pozwoli na ocenę zakończonych projektów, wybór odpowiedniego projektu referencyjnego do planowania projektu informatycznego oraz dobór środowiska pro-

jektowego i planowanie procesu wytwarzania wybranego projektu. Dzięki bezpośredniemu dostępowi do bazy danych systemu nie będzie potrzeby wprowadzania lub importowania dużej ilości danych. Podstawowymi funkcjami systemu będą:

- ocena dojrzałości zakończonych projektów informatycznych z możliwością analizy zjawisk prowadzących do wyznaczenia danego poziomu oceny.
- planowanie procesu wytwarzania oprogramowania z możliwością doboru zakresu projektu, składu zespołu i konfiguracji metodyki, czyli środowiska projektowego.
- zmiana planowanego procesu wytwarzania, aby dostosować go do zmieniających się warunków (zmiana wymagań, zmiana zakresu projektu, zmiany osobowe w zespole projektowym)
- ocena zgodności bieżącego projektu z planowanym przebiegiem procesu z możliwością analizy przyczyn niezgodności.

Prace na modelem AGOMO i zastosowaniem opartych na nich metod ma duży potencjał badawczy oraz komercyjny. Co więcej, uruchomienie zastosowania komercyjnego mogłoby spowodować wzrost przepływu informacji między jednostkami badawczymi i przemysłem, co pozwoliłoby przełamać impas wynikający z niechęci organizacji do udostępniania danych dotyczących procesów wytwarzania oprogramowania. Współpraca z jednostką badawczą, dotycząca zastosowania modelu usprawniającego funkcjonowanie organizacji mogłaby zmienić to nastawienie.

8. Spis literatury

1. Abdel–Hamid, T., Madnick, S.E.: Software Project Dynamics: An Integrated Approach. Prentice–Hall, Inc., Upper Saddle River, NJ, USA (1991).
2. Albrecht, A.: Measuring Application Development Productivity. In: Press, I. (ed.) Proc. of IBM Application Development Symp. pp. 83–92 (1979).
3. Anthony Crain: Overlapping iterations in a RUP–based project. IBM Dev. (2005).
4. Armentrout, A.W., Trujillo, R.M.: Function Points, Use Case Points, Story Points: Observations From. CrossTalk. 23 (2013).
5. Aydin, M.N. et al.: On the Adaptation of an Agile Information Systems Development Method: J. Database Manag. 16, 4, 25–40 (2005).
6. Bach–Dąbrowska, I., Wojnar, J.: Role patterns in it projects teams: Design of a selection module using fuzzy logic techniques. Found. Manag. 5, 1, 7–20 (2013).
7. Beck, K.: Embracing change with extreme programming. Computer. 32, 10, 70–77 (1999).
8. Beck, K., Andres, C.: Extreme programming explained: embrace change. Addison–Wesley, Boston, MA (2005).
9. Begel, A., Nagappan, N.: Usage and Perceptions of Agile Software Development in an Industrial Context: An Exploratory Study. In: Proceedings of the First International Symposium on Empirical Software Engineering and Measurement. pp. 255–264 IEEE Computer Society, Washington, DC, USA (2007).
10. Boehm, B.: A spiral model of software development and enhancement. ACM SIGSOFT Softw. Eng. Notes. 11, 4, 22–42 (1986).
11. Boehm, B.W. et al.: Software Cost Estimation with Cocomo II with Cdrom. Prentice Hall PTR, Upper Saddle River, NJ, USA (2000).
12. Boehm, B.W.: Software Engineering Economics. Prentice Hall PTR, Upper Saddle River, NJ, USA (1981).
13. Boehm, B.W., Turner, R.: Balancing agility and discipline: a guide for the perplexed. Addison–Wesley, Boston (2004).
14. Braubach, L., Pokahr, A.: The Jadex Project: Simulation. In: Ganzha, M. and Jain, L.C. (eds.) Multiagent Systems and Applications. pp. 107–128 Springer Berlin Heidelberg, Berlin, Heidelberg (2013).
15. Cao, L. et al.: A framework for adapting agile development methodologies. Eur. J. Inf. Syst. 18, 4, 332–343 (2009).
16. Cataldo, M., Nambiar, S.: On the Relationship Between Process Maturity and Geographic Distribution: An Empirical Analysis of Their Impact on Software Quality. In: Proceedings of the the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering. pp. 101–110 ACM, New York, NY, USA (2009).
17. Cezary Orłowski et al.: Model służący kształtowaniu procesu zarządzania projektem informatycznym dla podniesienia gotowości do zwinnej transformacji organizacji informatycznej. Presented at the XIX Konferencja Innowacje w Zarządzaniu i Inżynierii Produkcji , Zakopane 02–01.03 (2016).
18. Chrissis, M.B. et al.: CMMI for development: guidelines for process integration and product improvement. Addison–Wesley, Upper Saddle River, NJ (2011).
19. Clemmons, R.K.: Project estimation with use case points. J. Def. Softw. Eng. 18–22 (2006).
20. CMMI Product Team: CMMI for Development, Version 1.3, http://www.sei.cmu.edu/library/abstracts/reports/10tr033.cfm.
21. Cockburn, A.: Agile software development. Addison–Wesley, Boston (2002).

22. Daft, R.L., Lengel, R.H.: Organizational Information Requirements, Media Richness and Structural Design. Manag. Sci. 32, 5, 554–571 (1986).
23. Dong, S., Hu, B.: Multi–agent based simulation of team effectiveness in team's task process: a member–task interaction perspective. Int. J. Simul. Process Model. 4, 1, 54–68 (2008).
24. Donzelli, P., Iazeolla, G.: Hybrid simulation modelling of the software process. J. Syst. Softw. 59, 3, 227–235 (2001).
25. Eason, O.K.: Information Systems Development Methodologies Transitions: An Analysis of Waterfall to Agile Methodology. University of New Hampshire (2016).
26. Elliott, J. et al.: Java Swing, Second Edition. O'Reilly Media, Sebastopol, CA (2002).
27. Endriss, U. et al.: Logic–Based Agent Communication Protocols. In: Dignum, F. (ed.) Advances in Agent Communication. pp. 91–107 Springer Berlin Heidelberg (2003).
28. Esfahani, H.C.: Transitioning to Agile: A Framework for Pre–adoption Analysis using Empirical Knowledge and Strategic Modeling. (2012).
29. Fenton, N.E., Bieman, J.: Software metrics: a rigorous and practical approach. CRC Press, Taylor & Francie Group, Boca Raton (2014).
30. Ferreira, S. et al.: Understanding the Effects of Requirements Volatility in Software Engineering by Using Analytical Modeling and Software Process Simulation. J Syst Softw. 82, 10, 1568–1577 (2009).
31. Feyerabend, P.: Against method. Verso, London ; New York (2010).
32. Forrester, J.W.: Principles of systems. Pegasus Communications, Inc., Waltham, MA (1999).
33. Friedman–Hill, E.: Jess in action: rule–based systems in Java. Manning, Greenwich, CT (2003).
34. Fritzsche, M., Keil, P.: Agile methods and CMMI: compatibility or conflict? E–Inform. Softw. Eng. J. 1, 1, (2007).
35. Georgeff, M. et al.: The Belief–Desire–Intention Model of Agency. In: Müller, J.P. et al. (eds.) Intelligent Agents V: Agents Theories, Architectures, and Languages. pp. 1–10 Springer Berlin Heidelberg (1998).
36. Górski, J., Orłowski, C.: Integracja systemów informatycznych: nowe wyzwania. Pomorskie Wydawnictwo Naukowo–Techniczne PWNT, Gdańsk (2011).
37. Hannay, J.E. et al.: Effects of Personality on Pair Programming. IEEE Trans. Softw. Eng. 36, 1, 61–80 (2010).
38. Hanne, T., Neu, H.: Simulating human resources in software development processes. Berichte Fraunhofer ITWM. 64, (2004).
39. Henderson–Sellers, B., Serour, M.K.: Creating a Dual–Agility Method: The Value of Method Engineering. J. Database Manag. 16, 4, 1–24 (2005).
40. Herbsleb, J. et al.: Software quality and the capability maturity model. Commun. ACM. 40, 6, 30–40 (1997).
41. Hillel Glazer (Entinex, Inc.) Jeff Dalton (Broadsword Solutions Corporation) David Anderson (David J. Anderson & Associates, Inc.) Michael D. Konrad Sandra Shrum: CMMI or Agile: Why Not Embrace Both!
42. Höst, M. et al.: Exploring Bottlenecks in Market–Driven Requirements Management . . . (2000).
43. Jakobsen, C.R., Johnson, K.A.: Mature Agile with a Twist of CMMI. Presented at the (2008).
44. Jennings, N.R. et al.: A Roadmap of Agent Research and Development. Auton. Agents Multi–Agent Syst. 1, 1, 7–38 (1998).
45. Jiang, J.J. et al.: An exploration of the relationship between software development process maturity and project performance. Inf. Manage. 41, 3, 279–288 (2004).
46. Jones, C.: Estimating Software Costs: Bringing Realism to Estimating. McGraw–Hill Companies, New York (2007).

47. Jones, C.: Software Assessments, Benchmarks, and Best Practices. Addison Wesley, Boston, Mass (2000).
48. Jones, C.: Software Engineering Best Practices: Lessons from Successful Projects in the Top Companies. McGraw–Hill, New York (2010).
49. Jones, C., Bonsignour, O.: The Economics of Software Quality. Addison–Wesley Professional (2011).
50. Jones, M.R., Karsten, H.: Giddens's Structuration Theory and Information Systems Research. MIS Q. 32, 1, 127–157 (2008).
51. Joslin, D., Poole, W.: Agent–based simulation for software project planning. In: Proceedings of the Winter Simulation Conference, 2005. p. 8 pp.– (2005).
52. J.T.A. Jones: Software Development Life Cycle Model, http://www.slideshare.net/J.T.A.JONES/software–development–life–cycle–model–1392777.
53. Kan, S.H., Kosowski, A.: Metryki i modele w inżynierii jakości oprogramowania. Wydawnictwo Naukowe PWN, Warszawa (2006).
54. Kerzner, H.: Project Management: A Systems Approach to Planning, Scheduling, and Controlling. Wiley, Hoboken, New Jersey (2013).
55. Khan, A.I. et al.: A Comprehensive Study of Commonly Practiced Heavy and Light Weight Software Methodologies. (2011).
56. Krasteva, I. et al.: Experience–Based Approach for Adoption of Agile Practices in Software Development Projects. In: Pernici, B. (ed.) Advanced Information Systems Engineering. pp. 266–280 Springer Berlin Heidelberg, Berlin, Heidelberg (2010).
57. Krishnan, M.S. et al.: An Empirical Analysis of Productivity and Quality in Software Products. Manage Sci. 46, 6, 745–759 (2000).
58. Kroll, P. et al.: Rational unified process od strony praktycznej: [RUP]. Wydawnictwa Naukowo–Techniczne, Warszawa (2007).
59. Kruchten, P., Waligórski, S.: Rational unified process od strony teoretycznej: [RUP]. Wydawnictwa Naukowo–Techniczne, Warszawa (2007).
60. Kunz, R. et al.: Coupling of mesoscale and microscale models—an approach to simulate scale interaction. Environ. Model. Softw. 15, 6–7, 597–602 (2000).
61. Larman, C. et al.: UML i wzorce projektowe: analiza i projektowanie obiektowe oraz iteracyjny model wytwarzania aplikacji. Helion, Gliwice (2011).
62. Lehman, M.M. et al.: The Impact of Feedback in the Global Software Process. J. Syst. Softw. 46, 123–134 (1998).
63. McMahon, P.E.: Integrating CMMI and agile development: case studies and proven techniques for faster performance improvement. Addison–Wesley, Upper Saddle River, NJ (2011).
64. Merrill, D., Collofello, J.S.: Improving software project management skills using a software project simulator. Presented at the (1997).
65. Mi, P., Scacchi, W.: A Knowledge–Based Environment for Modeling and Simulating Software Engineering Processes. IEEE Trans Knowl Data Eng. 2, 3, 283–294 (1990).
66. Mikulenas, G., Kapocius, K.: An Approach for Prioritizing Agile Practices for Adaptation. In: Song, W.W. et al. (eds.) Information Systems Development. pp. 485–498 Springer New York, New York, NY (2011).
67. Nageswaran, S.: Test effort estimation using use case points. In: Quality Week. pp. 1–6 (2001).
68. O'Connor, C.P.: Anatomy and Physiology of an Agile Transition. In: Agile Conference (AGILE), 2011. pp. 302–306 (2011).
69. Orłowski, C. et al.: Hybrid Fuzzy–Ontological Project Framework of a Team Work Simulation System. Procedia Comput. Sci. 35, 1175–1184 (2014).
70. Osterweil, L.J.: Unifying Microprocess and Macroprocess Research. In: Li, M. et al. (eds.) Unifying the Software Process Spectrum. pp. 68–74 Springer Berlin Heidelberg (2005).

71. Paulk, M.: Capability maturity model for software. Encycl. Softw. Eng. (1993).
72. Paulk, M.: Using the Software CMM with good Judgment. Inst. Softw. Res. 12 (1999).
73. Paulk, N.C.: Extreme programming from a CMM perspective. IEEE Softw. 18, 6, 19–26 (2001).
74. Pfahl, D. et al.: A CBT module with integrated simulation component for software project management education and training. J. Syst. Softw. 59, 3, 283–298 (2001).
75. Pfahl, D., Lebsanft, K.: Using simulation to analyse the impact of software requirement volatility on project performance. Inf. Softw. Technol. 42, 14, 1001–1008 (2000).
76. Pitt, J., Mamdani, A.: Communication Protocols in Multi–agent Systems: A Development Method and Reference Architecture. In: Dignum, F. and Greaves, M. (eds.) Issues in Agent Communication. pp. 160–177 Springer Berlin Heidelberg (2000).
77. Qumer, A., Henderson–Sellers, B.: A framework to support the evaluation, adoption and improvement of agile methods in practice. J. Syst. Softw. 81, 11, 1899–1919 (2008).
78. Raffo D, Kellner M.,: Predicting the impact of potential process changes: a quantitative approach to process modeling. In: Elements of Software Process Assessment and Improvement. .
79. Raffo, D.M. et al.: Software process simulation to achieve higher CMM levels. J. Syst. Softw. 46, 2, 163–172 (1999).
80. Ramil, J.F., Smith, N.: Qualitative simulation of models of software evolution. Softw. Process Improv. Pract. 7, 3–4, 95–112 (2002).
81. Richardson, G.P., Pugh, A.L.: Introduction to System Dynamics Modeling with Dynamo. MIT Press, Cambridge, MA, USA (1981).
82. Rising, L., Janoff, N.S.: The Scrum software development process for small teams. IEEE Softw. 17, 4, 26–32 (2000).
83. Robinson, M., Vorobiev, P.A.: Swing. Manning, Greenwich (2003).
84. Rohunen, A. et al.: Approaches to Agile Adoption in Large Settings: A Comparison of the Results from a Literature Analysis and an Industrial Inventory. In: Ali Babar, M. et al. (eds.) Product–Focused Software Process Improvement. pp. 77–91 Springer Berlin Heidelberg, Berlin, Heidelberg (2010).
85. Royce, W.W.: Managing the development of large software systems: concepts and techniques. Presented at the Proceedings of the 9th international conference on Software Engineering March 1 (1987).
86. Rubin, K.S.: Essential Scrum: a practical guide to the most popular agile process. Addison–Wesley, Upper Saddle River, NJ (2012).
87. Sakellariou, I.: Agent based modelling and simulation using state machines. In: SIMULTECH, SciTePress. p. 270 (2012).
88. Satzinger, J.W. et al.: Systems Analysis and Design in a Changing World. Course Technology, Boston, MA (2015).
89. Schildt, H.: Java. The complete reference. McGraw–Hill Education, New York (2017).
90. Schwaber, K., Beedle, M.: Agile software development with Scrum. Prentice Hall, Upper Saddle River, NJ (2002).
91. Scott Ambler: Agile Practices and Principles Survey Results: July 2008, http://www.ambysoft.com/surveys/practicesPrinciples2008.html.
92. Scott Ambler: Communication on Agile Software Teams, http://agilemodeling.com/essays/communication.htm.
93. Sidky, A. et al.: A disciplined approach to adopting agile practices: the agile adoption framework. Innov. Syst. Softw. Eng. 3, 3, 203–216 (2007).
94. Sidky, A., Arthur, J.: Determining the Applicability of Agile Practices to Mission and Life–Critical Systems. In: Proceedings of the 31st IEEE Software Engineering Workshop. pp. 3–12 IEEE Computer Society, Washington, DC, USA (2007).
95. Soares, F.S.F., de Lemos Meira, S.R.: An agile strategy for implementing CMMI project management practices in software organizations. Presented at the June (2015).

96. Spasic, B., Onggo, B.S.: Agent–based simulation of the software development process: a case study at AVL. In: Simulation Conference (WSC), Proceedings of the 2012 Winter. pp. 1–11 (2012).
97. Stoica, M. et al.: Software Development: Agile vs. Traditional. Inform. Econ. 17, 4/2013, 64–76 (2013).
98. Sureshchandra, K., Shrinivasavadhani, J.: Adopting Agile in Distributed Development. Presented at the August (2008).
99. Sutherland, J. et al.: Scrum and CMMI Level 5: The Magic Potion for Code Warriors. Presented at the August (2007).
100. Syed–Abdullah, S. et al.: The Impact of an Agile Methodology on the Well Being of Development Teams. Empir. Softw. Eng. 11, 1, 143–167 (2006).
101. Turner, R., Jain, A.: Agile Meets CMMI: Culture Clash or Common Cause? In: Wells, D. and Williams, L. (eds.) Extreme Programming and Agile Methods — XP/Agile Universe 2002. pp. 153–165 Springer Berlin Heidelberg, Berlin, Heidelberg (2002).
102. Tvedt, J.D.: An Extensible Model for Evaluating the Impact of Process Improvements on Software Development Cycle Time. Arizona State University (1996).
103. Uhrmacher, A., Weyns, D. eds: Multi–agent systems: simulation and applications. CRC Press/Taylor & Francis, Boca Raton (2009).
104. Vikhorev, K. et al.: Agent Programming with Priorities and Deadlines. In: The 10th International Conference on Autonomous Agents and Multiagent Systems – Volume 1. pp. 397–404 International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC (2011).
105. Wickenberg, T., Davidsson, P.: On multi agent based simulation of software development processes. In: Multi–Agent–Based Simulation II. pp. 171–180 Springer (2003).
106. Williams, L.: The xp programmer: the few–minutes programmer. IEEE Softw. 20, 3, 16–20 (2003).
107. Wysocki, W. et al.: Efficiency and Maturity Assessment Model of RUP Process in IT Organizations. In: Wilimowska, Z. et al. (eds.) Information Systems Architecture and Technology: Proceedings of 37th International Conference on Information Systems Architecture and Technology – ISAT 2016 – Part IV. pp. 209–219 Springer International Publishing (2017).
108. Wysocki, W. et al.: Model of RUP Processes Maturity Assessment in IT Organizations. In: Madeyski, L. et al. (eds.) Software Engineering: Challenges and Solutions. pp. 187–199 Springer International Publishing (2017).
109. Yin, A. et al.: Scrum Maturity Model. In: Sixth International Conference on Software Engineering Advances (ICSEA 2011). pp. 20–29 IARIA (2011).
110. Zeigler, B.P. et al.: Theory of modeling and simulation: integrating discrete event and continuous complex dynamic systems. Academic Press, San Diego (2000).
111. Zhang, H. et al.: Software process simulation modeling: an extended systematic review. In: New Modeling Concepts for Today's Software Processes. pp. 309–320 Springer (2010).
112. agile42 | Introducing the agile42 Enterprise Transition Framework, http://www.agile42.com/en/blog/2014/02/10/enterprise–transition–framework/?utm_source=agile42&utm_medium=referral&utm_content=entry_list_block_homepage.
113. Enterprise Architect – UML Design Tools and UML CASE tools for software development, http://www.sparxsystems.com/products/ea/index.html.
114. IBM developerWorks : IBM Rational Method Composer, http://www.ibm.com/developerworks/downloads/r/rup/index.html.
115. Jade – Java Agent DEvelopment Framework, http://jade.tilab.com/.
116. Jess, the Rule Engine for the Java Platform, http://www.jessrules.com/.
117. JFreeChart, http://www.jfree.org/jfreechart/index.html.
118. Manifest programowania zwinnego, http://www.agilemanifesto.org/iso/pl/.

119. Software Project Benchmarking – Home Page, http://isbsg.org/.
120. Welcome to the Foundation for Intelligent Physical Agents, http://www.fipa.org/.

Spis rysunków

Rys. 1.1 Fazy metodyki kaskadowej 21
Rys. 1.2 Struktura procesu wytwarzania modelu spiralnego 22
Rys. 1.3 Fazy i kamienie milowe modelu RUP. 23
Rys. 1.4 Fazy cyklu życia projektu XP 27
Rys. 1.5 Podstawowe elementy metodyki Scrum 28
Rys. 1.6 Ocena dojrzałości organizacji w modelu CMMI 39
Rys. 1.7 Koncepcja oceny procesu wytwarzania oprogramowania organizacji oparta na modelu 40
Rys. 1.8 Koncepcja użycia modelu procesu wytwarzania do planowania. 41
Rys. 2.1 System rzeczywisty 43
Rys. 2.2 Dobór elementów środowiska projektowego jako źródło wiedzy o zmiennych wejściowych systemu rzeczywistego 44
Rys. 2.3 Zidentyfikowane zmienne wejściowe i wyjściowe procesu wytwarzania oprogramowania oraz przyjęty układ eksperymentu 46
Rys. 2.4 Diagram aktywności dyscypliny wymagań metodyki RUP 56
Rys. 2.5 Przepływ działań dyscypliny wymagań w postaci grafu skierowanego 56
Rys. 2.6 Elementy składowe zmiennej metodyka wytwarzania 57
Rys. 2.7 Idea zróżnicowanej wagi czynności poszczególnych dyscyplin w kolejnych iteracjach procesu wytwarzania. 58
Rys. 2.8 Przykład zbioru kategorii zadań wykonanych prac 62
Rys. 2.9 Powiązania między zmiennymi opisującymi system rzeczywisty w przyjętym układzie eksperymentu 63
Rys. 3.1 Koncepcja modelu podstawowego zbudowanego z uniwersalnego rdzenia i opcjonalnych rozszerzeń modelu 66
Rys. 3.2 Typ zadania, rola projektowa i artefakty Źródło: Opracowanie własne 69
Rys. 3.3 Cztery klasy zadań z przykładami 70
Rys. 3.4 Efektywność strategii komunikacyjnych 79
Rys. 3.5 Powiązanie działań dyscypliny wymagań metodyki RUP z przypadkami użycia oprogramowania IBM Rational Requirements Composer. 83
Rys. 3.6 Schemat działań celem oceny weryfikacji replikatywnej modelu 94
Rys. 3.7 Koncepcja działań planowanych w celu weryfikacji prognostycznej modelu 95
Rys. 4.1 Model AGOMO w postaci czarnej skrzynki 98
Rys. 4.2 Model AGOMO jako biała skrzynka 100
Rys. 4.3 Agent obserwujący stan środowiska i generujący akcje, które na nie wpływają 102
Rys. 4.4 Diagram stanów automatu *HelperBehaviour* 106
Rys. 4.5 Diagram stanów automatu *TaskManagerBehaviour* 107
Rys. 4.6 Diagram stanów automatu *TaskBehaviour* 108
Rys. 4.7 Diagram stanów automatu *WorkshopBehaviour* 109
Rys. 4.8 Diagram komunikacji agentów i obiektów modelu AGOMO 111
Rys. 4.9 Interfejs graficzny systemu AGOSIM–APP, symulatora modelu AGOMO 112

Rys. 4.10 Etapy metody AGOMAS służącej do oceny procesu RUP przy użyciu modelu AGOMO 113
Rys. 4.11 Zastosowanie modelu AGOMO do oceny dojrzałości procesów RUP organizacji 116
Rys. 4.12 Aplikacja AGOCOMP–APP wspierająca metodę AGOMAS 119
Rys. 5.1 Plan eksperymentu weryfikacji zasadności replikatywnej modelu 122
Rys. 5.2 Źródła informacji użyte do określenia wartości zmiennych wejściowych 123
Rys. 5.3 Średnia miesięczna liczba osób w zespole projektowym 125
Rys. 5.4 Diagram Gantta przedstawiający harmonogram otrzymany w wyniku symulacji .. 128
Rys. 5.5 Aplikacja AGOCOMP–APP – wynik porównania wzorca procesu z przebiegiem badanego projektu 130
Rys. 5.6 Aplikacja AGOCOMP–APP – powiększony obszar wyniku porównania wzorca procesu z przebiegiem badanego projektu 131
Rys. 6.1 Etapy metody AGOPLAN – planowania procesu RUP przy użyciu modelu AGOMO 134
Rys. 6.2 Plan eksperymentu weryfikacji zasadności predykcyjnej modelu AGOMO 136
Rys. 6.3 Porównanie przyrostów pracochłonności w czasie, według kategorii, część 1 141
Rys. 6.4 Porównanie przyrostów pracochłonności w czasie, według kategorii, część 2 142
Rys. 6.5 Porównanie przyrostów pracochłonności w czasie, według kategorii, część 3 143
Rys. 6.6 Porównanie przyrostów pracochłonności w czasie, według kategorii, część 4 143
Rys. 6.7 Porównanie przyrostów pracochłonności w czasie, według kategorii, część 5 144
Rys. 6.8 Zastosowanie metody AGOPLAN do planowania procesu wytwarzania i doboru środowiska projektowego 146
Rys. 7.1 Wskazane kierunki dalszych prac nad modelem AGOMO 150
Rys. 7.2 System AGOMO–PROJECT połączony z systemem zarządzania wytwarzaniem oprogramowania 151
Rys. B.1 Okno aplikacji AGOSIM–APP po uruchomieniu 167
Rys. B.2 Operacje na plikach projektów 168
Rys. B.3 Parametry wejściowe – zakres projektu 169
Rys. B.4 Parametry wejściowe – zespół projektowy 169
Rys. B.5 Parametry wejściowe – metodyka wytwarzania 170
Rys. B.6 Parametry wejściowe – plan projektu 171
Rys. B.7 Uruchomienie symulacji procesu wytwarzania w oparciu o model AGOMO 171
Rys. B.8 Sumaryczne wyniki symulacji 173
Rys. B.9 Wyniki symulacji – zawartość repozytorium 174
Rys. B.10 Wyniki symulacji – harmonogram prac 175
Rys. B.11 Wyniki symulacji – obciążenie pracą członków zespołu projektowego 176
Rys. B.12 Wyniki symulacji – zestawienie kosztów prac w projekcie 177
Rys. C.1 Interfejs graficzny aplikacji AGOCOMP–APP 179
Rys. D.1 Struktura menu opcji dodatkowych interaktywnego wykresu 182

Spis tabel

Tab. 1.1 Porównanie cech metodyk zwinnych i tradycyjnych 30
Tab. 1.2 Poziomy dojrzałości wraz z odpowiadającymi im obszarami procesowymi modelu CMMI 35
Tab. 1.3 Zależność między poziomem dojrzałości organizacji a ryzykiem zwinnej transformacji 38
Tab. 2.1 Przykładowy plan projektu realizowanego zgodnie z metodyką kaskadową w przypadku wykrycia błedów projektu 48
Tab. 2.2 Przykładowy plan projektu procesu wytwarzania zrealizowanego zgodnie z metodyką spiralną. 49
Tab. 2.3 Przykładowy plan projektu procesu wytwarzania zrealizowanego zgodnie z metodyką RUP 50
Tab. 2.4 Przykładowy plan projektu procesu wytwarzania zrealizowanego zgodnie z metodyką XP 51
Tab. 2.5 Przykładowy plan projektu procesu wytwarzania zrealizowanego zgodnie z metodyką Scrum 52
Tab. 3.1 Wartości zmiennych do oceny kompetencji 77
Tab. 3.2 Współczynniki efektywności strategii komunikacyjnych 80
Tab. 3.3 Potencjał błędu wg pochodzenia 85
Tab. 3.4 Efektywność usuwania błędów 87
Tab. 4.1 Proponowana interpretacja wartości wskaźnika zgodności działań $\boldsymbol{ZD}$ 118
Tab. 5.1 Wartość zmiennej wejściowej $\boldsymbol{ZP}$ – zakres projektu 124
Tab. 5.2 Odtworzona zawartość zmiennej $\boldsymbol{PP}$ – plan projektu 125
Tab. 5.3 Wyniki symulacji – porównanie liczby artefaktów 128
Tab. 5.4 Wyniki symulacji – porównanie pracochłonności wg kategorii zadań 129
Tab. 5.5 Wyniki porównania badanego procesu z wzorcem procesu 131
Tab. 6.1 Podstawowe dane projektu ISBSG 17563 – projektu referencyjnego 138
Tab. 6.2 Porównanie wyników symulacji projektu ISBSG 17563 z danymi rzeczywistymi. 138
Tab. 6.3 Porównanie czasów trwania badanego projektu z wynikiem symulacji 139
Tab. 6.4 Wyniki symulacji procesu wytwarzania projektu UBEZP, przy użyciu parametrów wewnętrznych referencyjnego projektu ISBSG 17563 140
Tab. 7.1 Zastosowanie metod AGOMAS i AGOPLAN do wsparcia zarządzania projektami informatycznymi 151
Tab. A.1 Klasyfikacja złożoności przypadków użycia i przypisane wagi 164
Tab. A.2 Czynniki złożoności technicznej 164
Tab. A.3 Czynniki złożoności środowiska 165

Dodatek A. Rozmiar oprogramowania

W niniejszym dodatku przedstawiona została metoda szacowania rozmiaru oprogramowania UCP (ang. *Use Case Points*) dostosowana do modelu AGOMO. Oryginalna metoda UCP została zmieniona celem dopasowania wartości oszacowania UCP do liczby artefaktów typu przypadek użycia w repozytorium modelu. Dostosowana metoda została nazwana AGOMO–UCP, wartość oszacowanego rozmiaru oprogramowania jest również oznaczona jako AGOMO–UCP.

Metoda Use Case Point została opracowana w roku 1993 przez Gustawa Karnera na podstawie metody Function Points, przez dostosowanie jej do paradygmatu obiektowego, przez zastąpienie ekranów i elementów architektury przypadkami użycia i aktorami. Oryginalna metoda UCP polega na wyliczeniu następujących elementów:

- UUCW (ang. *Unadjusted Use Case Weight*) – waga przypadków użycia – jest to rozmiar oprogramowania w punktach, na który składa się liczba i złożoność przypadków użycia,
- UAW (ang. *Unadjusted Actor Weight*) – waga aktorów – jest to rozmiar oprogramowania w punktach, na który składa się liczba i złożoność aktorów,
- TCF (ang. *Technical Complexity Factor*) – czynnik, który jest używany do uwzględnienia złożoności technicznej systemu,
- ECF (ang. *Environmental Complexity Factor*) – czynnik, który jest używany do uwzględnienia złożoności środowiska systemu.

W metodzie AGOMO–UCP występują trzy elementy używane do obliczenia wartości rozmiaru oprogramowania: AGOMO–UUCW, AGOMO–TCF, AGOMO–ECF. Element odpowiadający wadze aktorów został pominięty z powodu małego wpływu złożoności aktorów na wynik szacowania. Wzorem dla tego uproszczonego podejścia była metoda szacowania wzorowana na UCP, włączona w system modelowania oprogramowania Enterprise Architect firmy Sparx [113].

1. Obliczanie wartości AGOMO–UCP na podstawie modelu przypadków użycia

Czynnik AGOMO–UUCW

AGOMO–UUCW jest podstawowym czynnikiem, który ma wpływ na oszacowany rozmiar oprogramowania do wytworzenia. Jest on obliczany na podstawie liczby i złożoności przypadków użycia systemu. Aby wyliczyć wartość AGOMO–UUCW, każdy z przypadków użycia musi być zaliczony do jednej z kategorii złożoności: prosty, średni albo złożony. Każda z kategorii ma przypisaną odpowiednią wagę. Wartości wag w metodzie AGOMO–UCP różnią się od wartości wag w standardowej metodzie UCP. Po zaklasyfikowaniu przypadków do kategorii złożoności, obliczenie wartości AGOMO–UUCW polega na zsumowaniu odpowiednich wag dla przypadków użycia, które zostały umieszczone w Tab. A.1.

Tab. A.1 Klasyfikacja złożoności przypadków użycia i przypisane wagi
Źródło: opracowane na podstawie [19]

Klasyfikacja przypadku użycia	Liczba transakcji	Waga
prosty	od 1 do 3	1
średni	od 4 do 7	2
złożony	8 i więcej	3

$$AGOMO–UUCW = liczba\ prostych\ p.u. + (liczba\ średnich\ p.u.*2) + (liczba\ średnich\ p.u.*3) \quad \text{(Dodatek A.1)}$$

Czynnik AGOMO–TCF

AGOMO–TCF jest to czynnik, który został włączony w oszacowanie rozmiaru oprogramowania w celu uwzględnienia technicznej złożoności systemu. Jest on określony przez przypisanie odpowiedniej liczy punktów od 0 (czynnik jest nieistotny) do 5 (czynnik jest niezbędny) trzynastu technicznym czynnikom zamieszczonych w

Tab. A.2. Liczba punktów jest następnie mnożona przez wagę czynnika. Suma wyliczonych wartości jest to czynnik techniczny AGOMO–TF. Wartość czynnika AGOMO–TCF jest wyliczana według następującego wzoru:

$$AGOMO–TCF = 0,6 * AGOMO–TF \quad \text{(Dodatek A.2)}$$

Tab. A.2 Czynniki złożoności technicznej
Źródło: opracowane na podstawie [19]

Czynnik	Opis	Waga
T1	System rozproszony	2,0
T2	Wydajność	1,0
T3	Wydajność dla użytkownika końcowego	1,0
T4	Złożone przetwarzanie wewnętrzne	1,0
T5	Ponowne użycie	1,0
T6	Łatwość w instalacji	0,5
T7	Łatwość użycia	0,5
T8	Przenośność	2,0
T9	Łatwość wprowadzania zmian	1,0
T10	Współbieżność	1,0
T11	Specjalne zabezpieczenia	1,0
T12	Zależność od zewnętrznych bibliotek	1,0
T13	Dodatkowe szkolenia dla użytkowników	1,0

Czynnik AGOMO–ECF

AGOMO–ECF jest to czynnik, który został włączony w oszacowanie rozmiaru oprogramowania, aby uwzględnić złożoność środowiskową systemu. Jest on określony przez przypisanie odpowiedniej liczy punktów od 0 (brak doświadczenia) do 5 (ekspert) ośmiu czynników środo-

wiska zamieszczonych w Tab. A.3. Liczba punktów jest następnie mnożona przez wagę czynnika. Suma wyliczonych wartości jest to czynnik środowiska AGOMO–EF. Wartość czynnika AGOMO–ECF jest wyliczana według następującego wzoru:

$$AGOMO–ECF=1,4 + (– 0,03 * AGOMO–EF) \quad \text{(Dodatek A.3)}$$

Tab. A.3 Czynniki złożoności środowiska

Źródło: opracowane na podstawie [19]

Czynnik	Opis	Waga
E1	Zaznajomienie z projektem	1,5
E2	Doświadczenie w tworzeniu aplikacji	0,5
E3	Doświadczenie w projektowaniu aplikacji zorientowanych obiektowo	1,0
E4	Umiejętności głównego analityka	0,5
E5	Motywacja	1,0
E6	Stabilność wymagań	2,0
E7	Pracownicy pracują w niepełnym wymiarze	1
E8	Trudność języka programowania	1,0

Wynik szacowania zgodnie z metodą AGOMO–UCP jest obliczany na podstawie wyznaczonych czynników, według następującego wzoru:

$$AGOMO–UCP = AGOMO–UUCW * AGOMO–TCF * AGOMO–ECF \quad \text{(Dodatek A.4)}$$

Dodatek B. Aplikacja AGOSIM–APP

1. Instalacja aplikacji

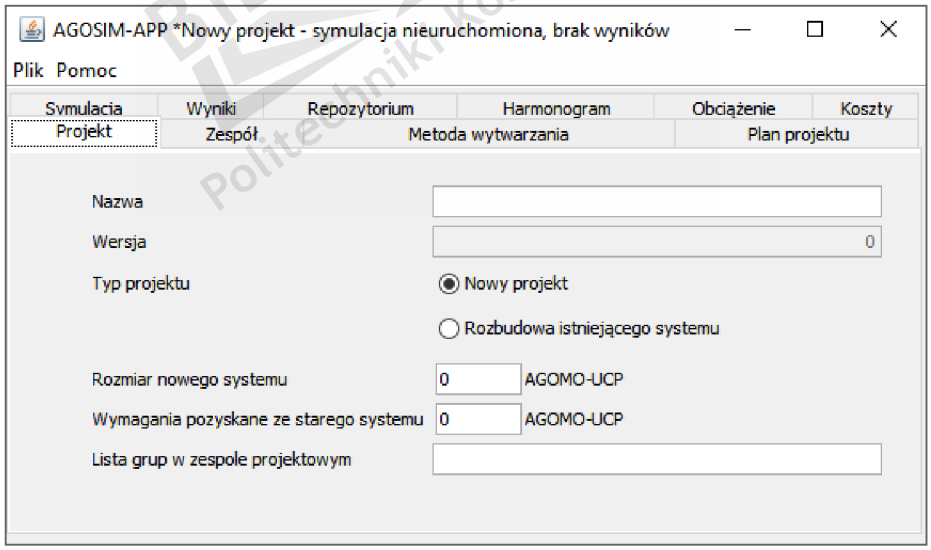
W celu zainstalowania aplikacji należy zainstalować w systemie środowisko deweloperskie języka Java wersja jdk 1.7.55. Aplikacja jest dostarczona w postaci pliku agosim.zip, który należy umieścić w katalogu na dysku twardym komputera, który po instalacji staje się katalogiem głównym aplikacji. Po rozpakowaniu archiwum zostaje utworzona odpowiednia struktura podkatalogów, niezbędna do działania aplikacji.

2. Uruchomienie

Aplikację uruchamia się przez wpisanie następującego polecenia w linię poleceń otwartą w katalogu głównym aplikacji:

```
katalog-główny>agosim
```

Po uruchomieniu aplikacji AGOSIM–APP pojawia się główne okno aplikacji z menu i zakładkami, pokazane na Rys. B.1. Funkcje aplikacji umieszczone w menu i zakładkach zostaną opisane poniżej.



Rys. B.1 Okno aplikacji AGOSIM–APP po uruchomieniu

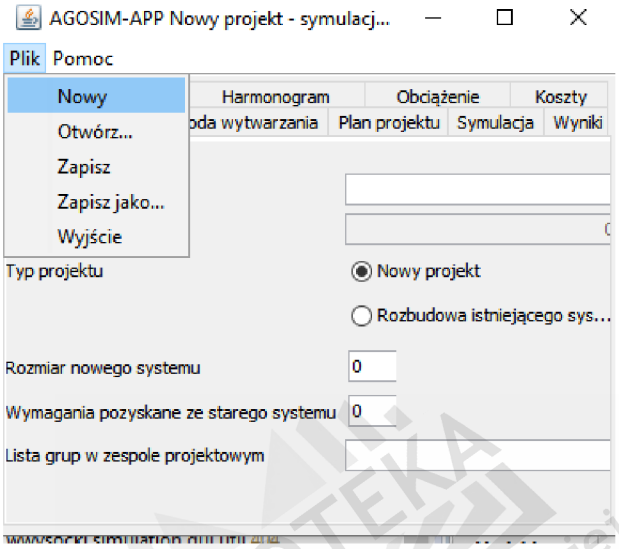
Źródło: Opracowanie własne

3. Funkcje aplikacji

Poniżej przedstawiono główne funkcje aplikacji i rysunki zawierające odpowiadające im elementu interfejsy graficznego. Aplikacja zawiera implementację modelu uproszczonego rozszerzoną o możliwości opisane w podrozdziale 0.

Praca z projektami

Aplikacja umożliwia standardowe operacje na projektach: utworzenie nowego projektu, odczytanie z pliku, zapisanie do pliku oraz zapisanie pod inną nazwą. Projekt zawiera zestaw danych jednego projektu informatycznego: parametry wejściowe modelu oraz wynik symulacji. Nazwa pliku projektu i jego stan – informacja czy symulacja została wykonana i czy wynik symulacji jest aktualny (czy zostały zmienione parametry wejściowe) są umieszczone w pasku tytułowym aplikacji.

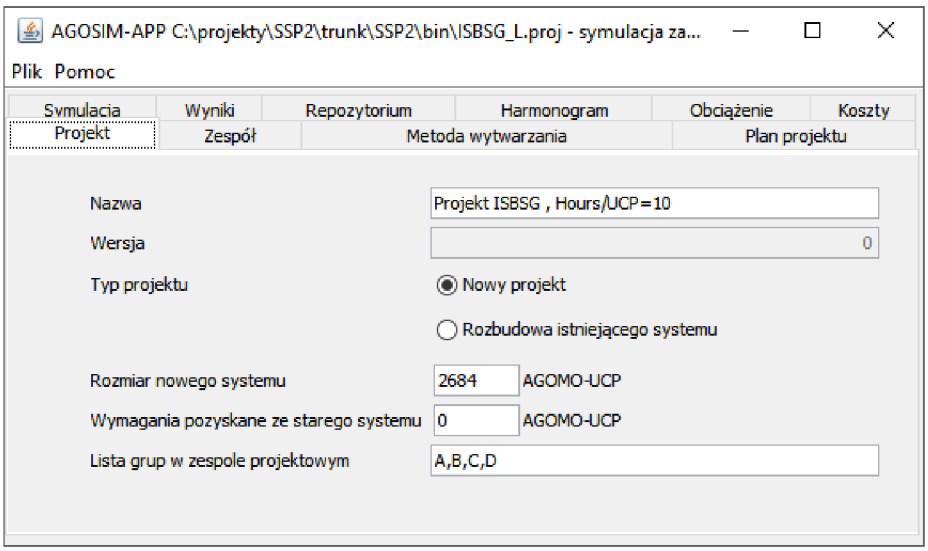


Rys. B.2 Operacje na plikach projektów

Źródło: Opracowanie własne

Wprowadzanie parametrów wejściowych symulacji

Przed rozpoczęciem symulacji projektu należy wprowadzić wartości parametrów wejściowych modelu. Z powodu dużej liczby elementów składowych, parametry wejściowe zostały umieszczone w osobnych zakładkach okna aplikacji. Wartości zmiennej wejściowej *ZP* – Zakres projektu zostały umieszczone w zakładce Projekt, pokazanej na Rys. B.3.

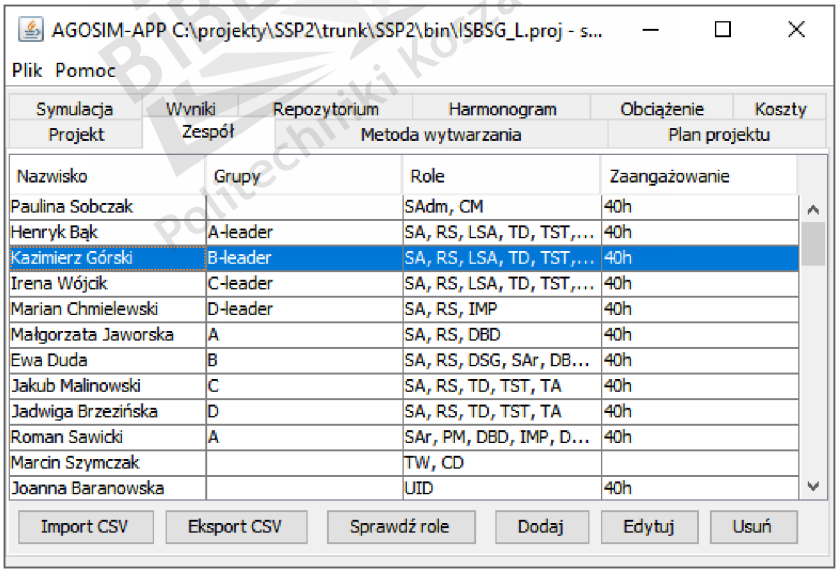


Rys. B.3 Parametry wejściowe – zakres projektu

Źródło: Opracowanie własne

W zakładce można określić nazwę projektu, typ – czy jest to projekt nowy, czy rozbudowa poprzedniej wersji systemu oraz oszacowany rozmiar części nowej i odzyskanej w AGOMO–UCP. Poniżej można określić nazwy grup w zespole projektowym. Możliwość określenia grup została opisana w podrozdziale 0. Pole zawiera nazwy grup oddzielone przecinkami.

Wartość zmiennej *Z* – Zespół projektowy można określić w zakładce Zespół, której zawartość pokazano na Rys. B.4.



Rys. B.4 Parametry wejściowe – zespół projektowy

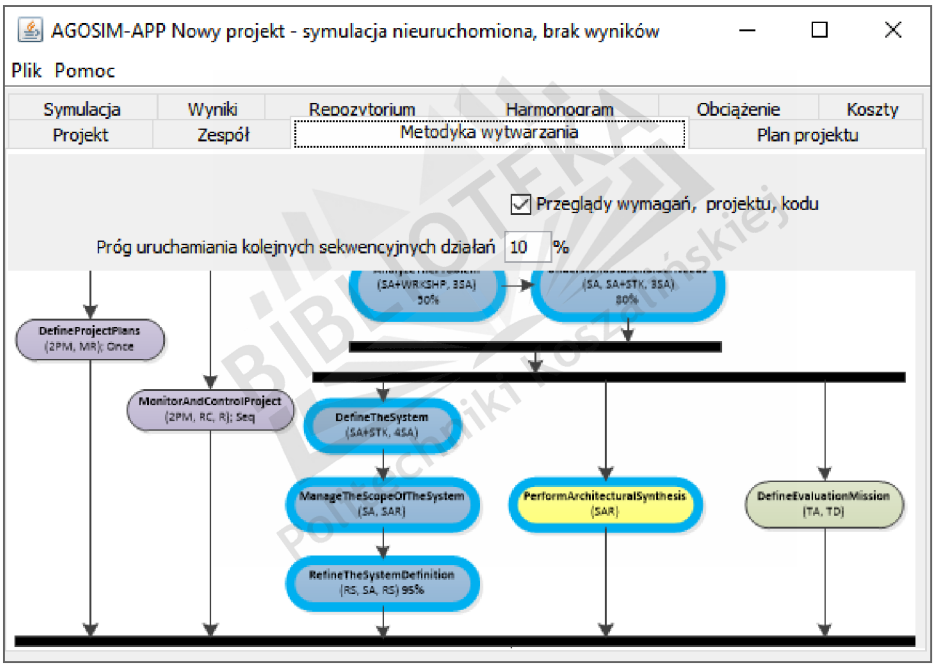
Źródło: Opracowanie własne

Zawartość zakładki to tabela przedstawiająca osoby wchodzące w skład zespołu projektowego, ich role i cechy oraz przyciski umożliwiające następujące działania na zespole projektowym lub podświetlonym wierszu tabeli:

- **Import CSV** – umożliwia odczytanie składu zespołu projektowego z zewnętrznego pliku zapisanego w formacie CSV

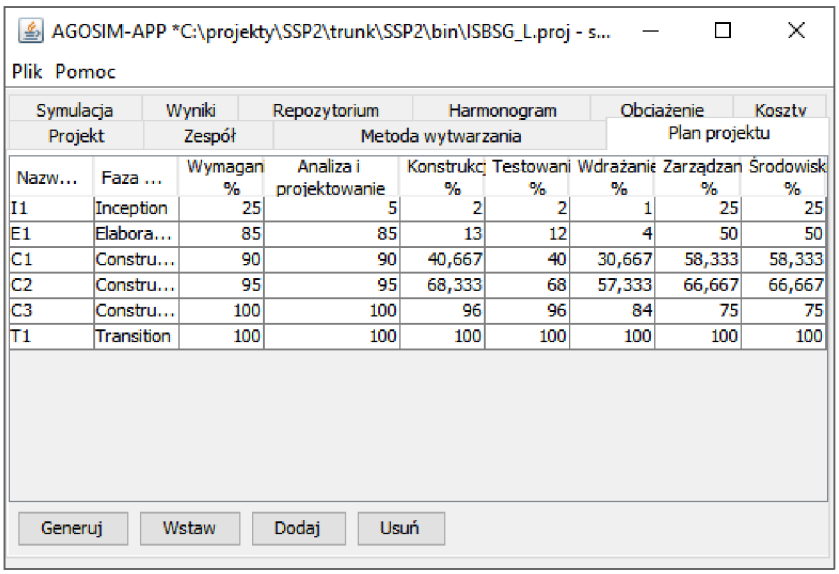
- **Eksport CSV** – umożliwia zapisanie składu zespołu projektowego do zewnętrznego pliku w formacie CSV
- **Sprawdź role** – przycisk uruchamia sprawdzenie czy członkowie zespołu projektowego mają przydzielone wszystkie role wymagane przez metodykę RUP
- **Dodaj** – przycisk umożliwia dodanie i edycję nowego członka zespołu projektowego
- **Edytuj** – przycisk umożliwia edycję wybranego członka zespołu projektowego
- **Usuń** – przycisk umożliwia usunięcie wybranego członka zespołu projektowego.

Wartość zmiennej wejściowej *M* jest umieszczona w zakładce Metoda wytwarzania. Znajduje się na niej włącznik zadań polegających na przeglądach wymagań, projektu i kodu oraz pole liczbowe określające, po osiągnięciu jakiej wartości procentowej wykonania zadania należącego do działania sekwencyjnego należy uruchomić następne zadanie. Poniżej elementów służących do konfiguracji metodyki wytwarzania umieszczono okno z diagramem aktywności metodyki RUP zaimplementowanej w modelu AGOMO. Diagram można przesuwać wewnątrz okna poruszając wskaźnikiem, przy wciśniętym lewym przycisku urządzenia wskazującego.



Rys. B.5 Parametry wejściowe – metodyka wytwarzania
Źródło: Opracowanie własne

Wartość zmiennej wejściowej *PP* – Plan projektu została umieszczona w zakładce Plan projektu. Jak widać na Rys. B.6, w kolejnych wierszach tabeli umieszczono informacje o celach dyscyplin dla poszczególnych iteracji. Nowy plan projektu może zostać utworzony po wciśnięciu przycisku Generuj, na podstawie wprowadzonej liczby iteracji etapów procesu wytwarzania. Procentowe cele do osiągnięcia w utworzonych iteracjach są równomiernie rozłożone w cyklu życia oprogramowania.

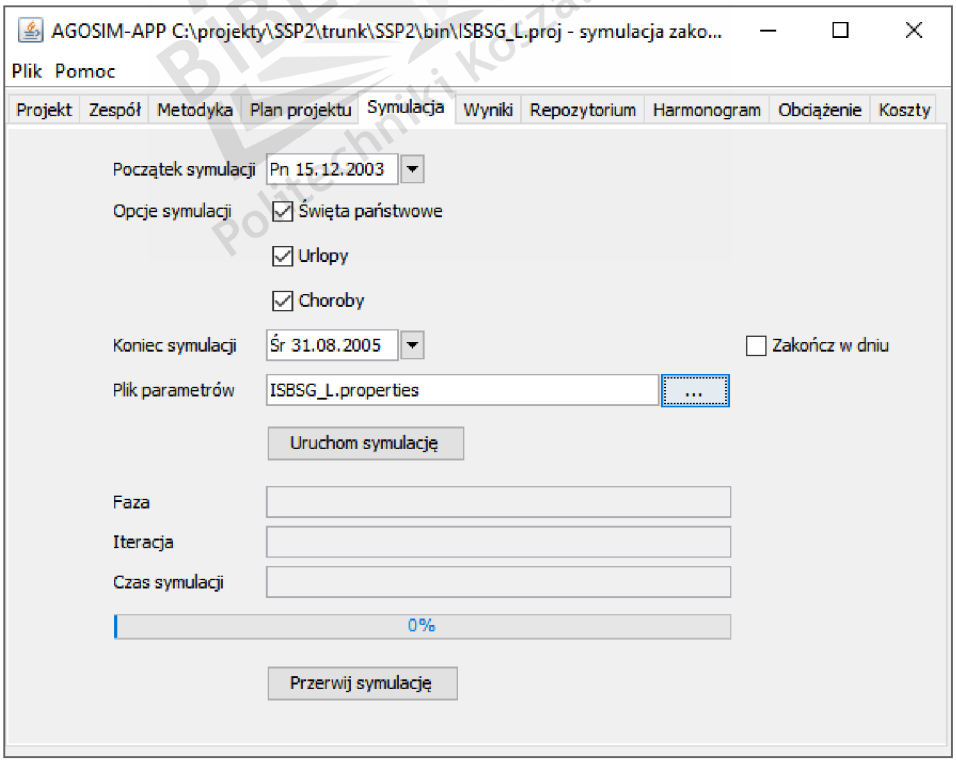


Rys. B.6 Parametry wejściowe – plan projektu

Źródło: Opracowanie własne

Uruchomienie symulacji

Po ustaleniu wartości zmiennych wejściowych modelu, w omówionych powyżej zakładkach można uruchomić symulację procesu wytwarzania w oparciu o model AGOMO. Symulacja jest uruchamiana po utworzeniu odpowiednich agentów i obiektów w systemie wieloagentowym JADE. Uruchomienie i kontrola symulacji są możliwa dzięki zakładce Symulacja, której zawartość została przedstawiona na Rys. B.7.



Rys. B.7 Uruchomienie symulacji procesu wytwarzania w oparciu o model AGOMO

Źródło: Opracowanie własne

W górnej części zakładki znajduje się pole, w którym można określić datę początkową symulowanego procesu wytwarzania. Poniżej znajdują się opcje modułu kalendarza. Pierwsza opcja

pozwala włączyć generator weekendów oraz świąt państwowych stałych i ruchomych. Wyłączenie opcji spowoduje pracę zespołu projektowego przez 8 godzin, niezależnie od dnia tygodnia, przez 365 dni w roku. Opcja Urlopy steruje włączeniem generatora urlopów. Generator urlopów przydziela członkom zespołu projektowego 26 dni urlopu losowo w częściach, podzielone na krótsze ferie zimowe, dłuższe letnie wakacje i reszta dni przypadkowo w ciągu roku. Generator chorób jest włączany i wyłączany przez opcję Choroby. Włączenie generatora chorób powoduje zwolnienie chorobowe każdego pracownika przez losowo wybrane 12 dni w ciągu roku pracy. Umieszczone poniżej pole z datą o nazwie Koniec symulacji pozwala wymusić zakończenie symulacji w wybranym dniu. Potwierdzenie zakończenia symulacji w wybranym dniu powinno być potwierdzone ustawieniem opcji Zakończ w dniu. Pole Plik parametrów pozwala ustalić plik konfiguracyjny, w którym znajdują się parametry wewnętrzne symulacji, opisane w podrozdziale 0 Typy zadań. Wybranie pliku parametrów następuje po wciśnięciu przycisku […].

Uruchomienie symulacji procesu wytwarzania następuje po wciśnięciu przycisku Uruchom symulację. Poniżej przycisku znajdują się pola, w których symulator wyświetla aktualną fazę, iterację oraz czas symulacji liczony w dniach, godzinach i minutach od podanej daty początku symulacji. Pod tymi informacjami został umieszczony pasek postępu pokazujący procent zaawansowania symulacji. Przycisk Przerwij symulację pozwala zatrzymać symulację w dowolnym momencie. Po zakończeniu lub przerwaniu symulacji aktualizowany jest opis stanu projektu umieszczony w pasku tytułowym okna aplikacji. Część opisu poświęcona wizualizacji otrzymanych wyników umieszczono w kolejnym podrozdziale.

Analiza wyników

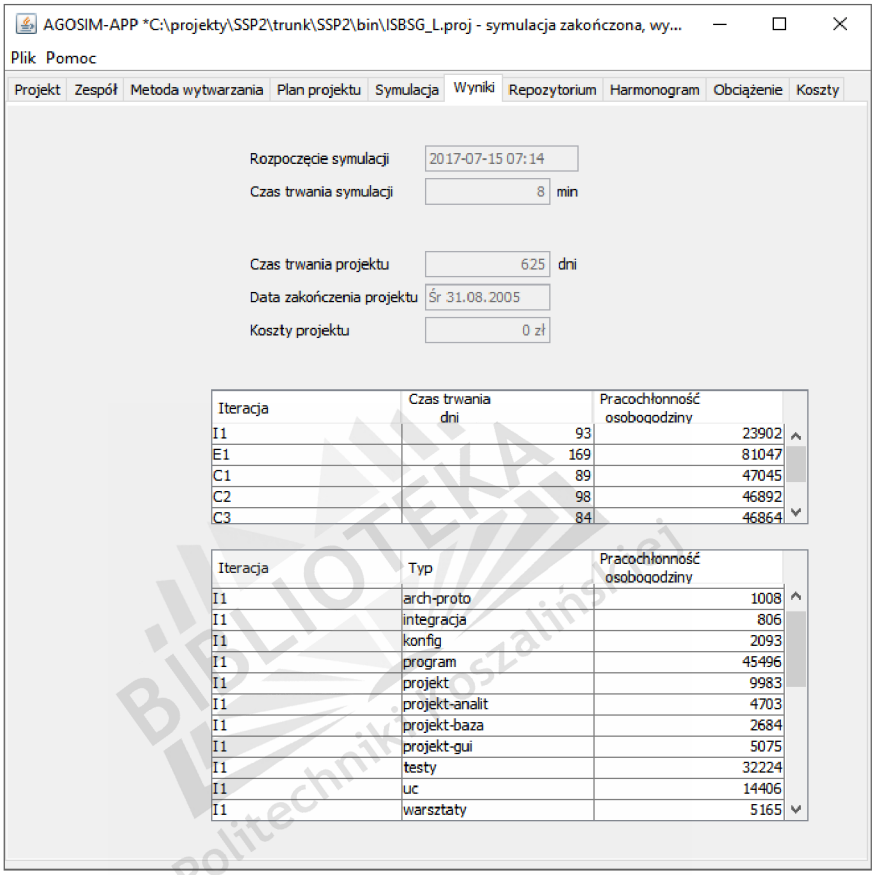
Po zakończeniu lub przerwaniu symulacji otrzymane wyniki można obejrzeć w pięciu zakładkach:

- Wyniki – zawierają informacje o czasie trwania projektu, podsumowanie pracochłonności i kosztów,
- Repozytorium – zawiera informację o liczbie artefaktów wszystkich typów w chwili przerwania lub zakończenia symulacji,
- Harmonogram – zawiera interaktywny wykres Gantta harmonogramu prac w projekcie wyznaczonych przez model AGOMO,
- Obciążenie – zawiera interaktywny wykres obciążenia godzinowego wybranych lub wszystkich członków zespołu projektowego.
- Koszty – zawiera zestawienie sumarycznych kosztów projektu wynikających z zatrudnienia zespołu projektowego.

Zawartość zakładek zostanie omówiona w kolejnych podrozdziałach.

Wyniki sumaryczne

Wygląd zakładki zawierającej podsumowanie wyników symulacji przedstawiono na Rys. B.8. Główne informacje to liczba dni symulowanego procesu wytwarzania, data zakończenia projektu i koszty zatrudnienia zespołu projektowego. Poniżej znajdują się dwie tabele. Pierwsza z nich zawiera liczbę dni i pracochłonność w osobogodzinach pogrupowane według iteracji. Druga tabela zawiera bardziej szczegółowe zestawienie pracochłonności w podziale na iteracje i kategorie zadań.

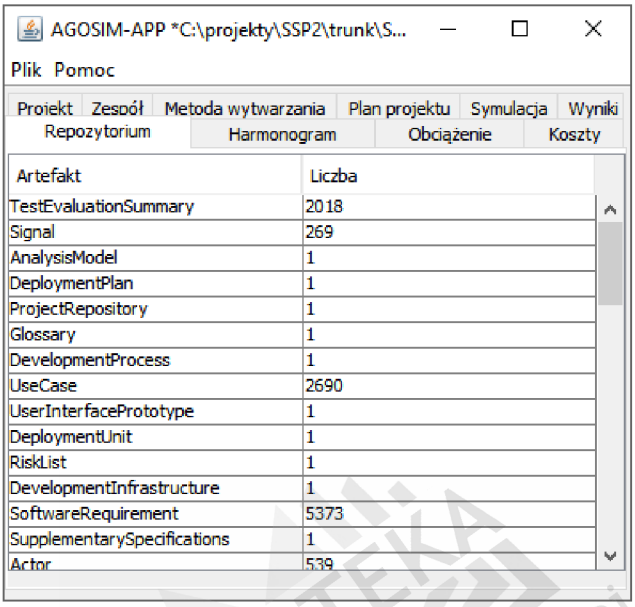


Rys. B.8 Sumaryczne wyniki symulacji

Źródło: Opracowanie własne

Zawartość repozytorium

Przedstawiona na Rys. B.9 zakładka z zawartością repozytorium składa się z tabeli z typami artefaktów umieszczonych w repozytorium i liczby instancji danego typu. Pokazuje stan repozytorium w chwili przerwania lub zakończenia procesu wytwarzania oprogramowania.

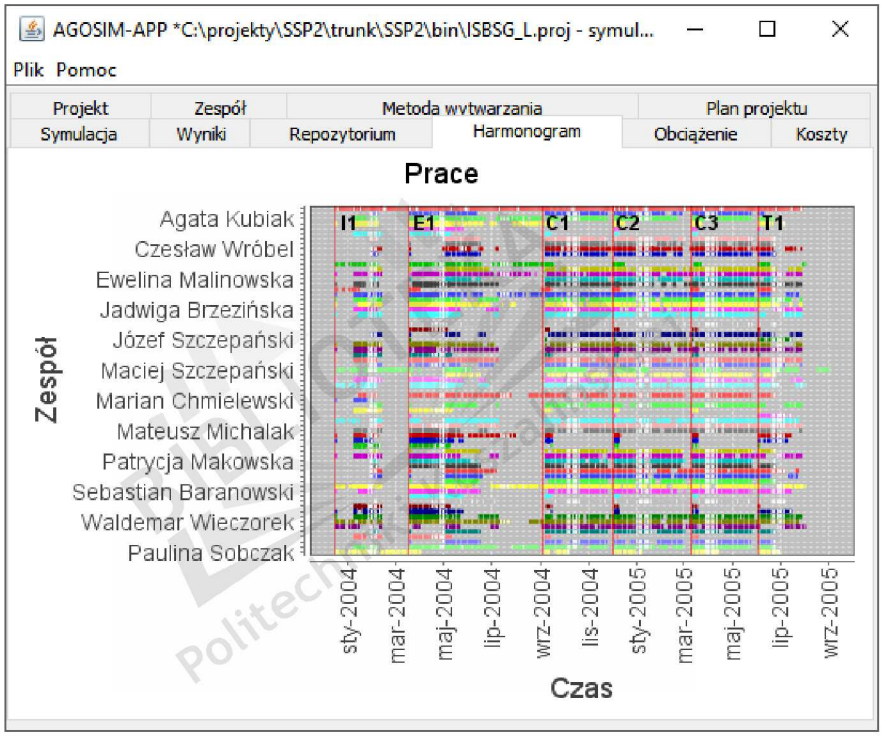


Rys. B.9 Wyniki symulacji – zawartość repozytorium

Źródło: Opracowanie własne

Harmonogram prac

Szczegółowy harmonogram prac, będący wynikiem symulacji modelu AGOMO, jest przedstawiony w postaci interaktywnego diagramu Gantta. Diagram został umieszczony w osobnej zakładce Harmonogram aplikacji AGOSIM–APP, którą pokazano na Rys. B.10. Na osi poziomej umieszczono czas trwania projektu, na osi pionowej skład zespołu projektowego. Pierwszym, licząc od góry wykresu, przebiegiem prac są @Działania. Zawierają one działania procesu wytwarzania, których zadania reprezentowane są poniżej. Pionowymi liniami oddzielono iteracje procesu wytwarzania, kolorami oznaczono osobę wykonującą zadania. Umieszczenie wskaźnika urządzenia wskazującego na zadaniu przedstawionym na wykresie bez ruchu, powoduje wyświetlenie okienka z informacją kto wykonywał zadanie, jakiego typu było to zadanie, kiedy się rozpoczęło i ile trwało. Dodatkowo funkcje zapewniane przez interaktywne wykresy aplikacji i sposób pracy z nimi zamieszczono w Dodatku D rozprawy.



Rys. B.10 Wyniki symulacji – harmonogram prac

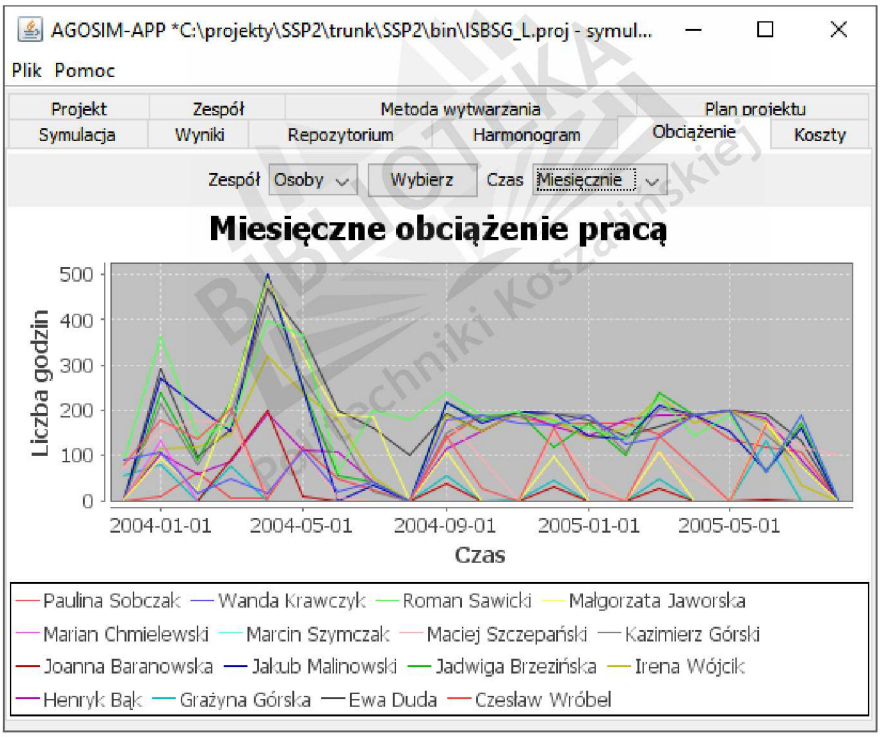
Źródło: Opracowanie własne

Obciążenie pracą

Wykres przedstawiający obciążenie pracą zespołu projektowego umieszczono w osobnej zakładce aplikacji, której wygląd przedstawia Rys. B.11. W górnej części zakładki umieszczono pola pozwalające wybrać typ wykresu:

- Zespół – lista pozwala wybrać, czy mają być pokazywane obciążenia pojedynczych osób, czy ról projektowych,
- Wybierz – po wciśnięciu przycisku można wybrać członków zespołu projektowego, dla których zostanie sporządzony wykres,
- Czas – lista pozwala wybrać okres sumowania obciążenia. Możliwe wartości to dziennie, tygodniowo, miesięcznie lub za cały projekt.

Funkcje zapewniane przez interaktywne wykresy aplikacji i sposób pracy z nimi zamieszczono w Dodatku D rozprawy.

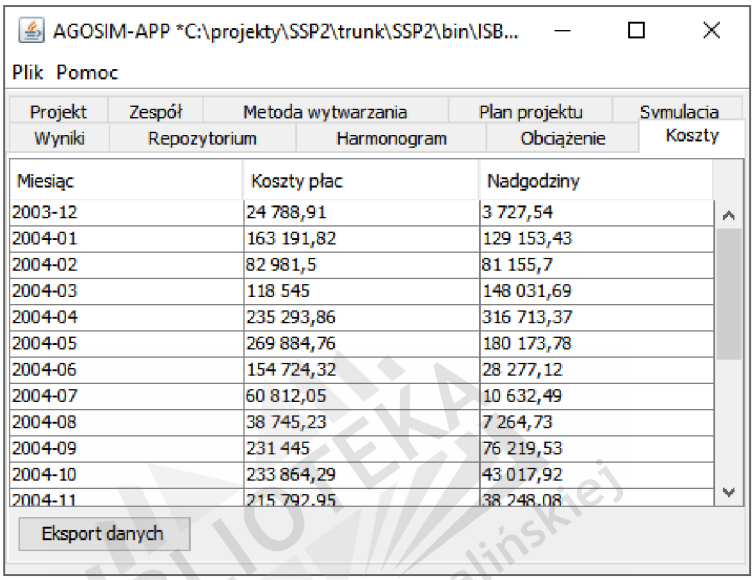


Rys. B.11 Wyniki symulacji – obciążenie pracą członków zespołu projektowego

Źródło: Opracowanie własne

Koszty prac w projekcie

Zestawienie kosztów prac w projekcie jest wyświetlane jako tabela w osobnej zakładce w oknie aplikacji, co pokazuje Rys. B.12. Jest ono obliczane jako sumy miesięczne za okres trwania projektu na podstawie zaplanowanego harmonogramu prac, informacji o zaangażowaniu w prace zespołu projektowego oraz stawek godzinowych. Koszty prac osób, które mają zaangażowanie o wartości ponad 40 godzin tygodniowo i w zaplanowanym harmonogramie przepracowały nadgodziny są ujęte w kolumnie Nadgodziny.



Rys. B.12 Wyniki symulacji – zestawienie kosztów prac w projekcie

Źródło: Opracowanie własne

Dodatek C. Aplikacja AGOCOMP–APP

1. Instalacja aplikacji

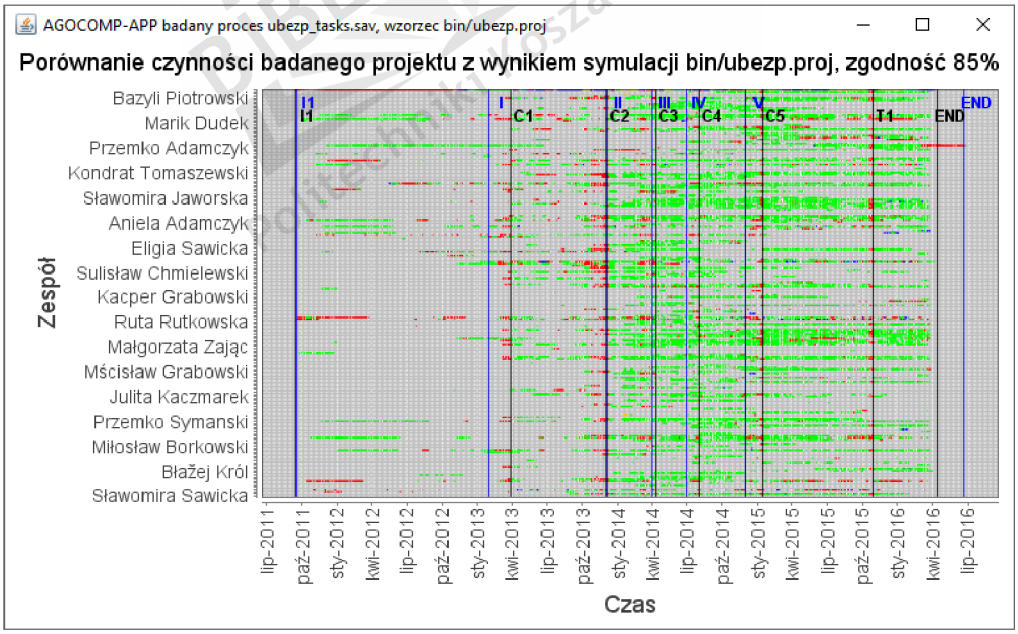
W celu zainstalowania aplikacji należy zainstalować w systemie środowisko deweloperskie języka Java wersja jdk 1.7.55.Aplikacja jest dostarczona w postaci pliku agocomp.zip, który należy umieścić w katalogu na dysku twardym komputera, który po instalacji staje się katalogiem głównym aplikacji. Po rozpakowaniu archiwum zostaje utworzona odpowiednia struktura podkatalogów, niezbędna do działania aplikacji.

2. Uruchomienie

Aplikację uruchamia się przez wpisanie następującego polecenia w linię poleceń otwartą w katalogu głównym aplikacji:

```
katalog-główny>agocomp
```

Po uruchomieniu aplikacji AGOCOMP–APP pojawia się główne okno aplikacji, pokazane na Rys. C.1.



Rys. C.1 Interfejs graficzny aplikacji AGOCOMP–APP

Źródło: Opracowanie własne

3. Funkcje aplikacji

Główną funkcją aplikacji jest odczytanie badanego przebiegu projektu z pliku ubezp_tasks.sav i wzorca projektu z pliku ubezp.proj, porównanie ich ze sobą w sposób opisany w podrozdziale 4.4. Wynik porównania w postaci wskaźnika zgodności działań jest umieszczony w tytule wykresu. Szczegóły porównania pokazano poniżej w postaci diagramu Gantta. Diagram

pokazuje zadania rzeczywistego, badanego procesu wytwarzania. Kolor zadań jest zależny od zgodności z wzorcem projektu. Zadania zgodne z wzorcem są oznaczone kolorem zielonym, zadania niezgodne są w kolorze czerwonym, natomiast zadania o bez przypisanej kategorii są wyświetlone w kolorze niebieskim. Umieszczenie wskaźnika urządzenia wskazującego na zadaniu przedstawionym na wykresie bez ruchu, powoduje wyświetlenie okienka z informacją kto wykonywał zadanie, jakiego typu było to zadanie, kiedy się rozpoczęło i ile trwało. Dodatkowo funkcje zapewniane przez interaktywne wykresy aplikacji i sposób pracy z nimi zamieszczono w Dodatku D rozprawy.

Dodatek D. Interaktywne wykresy

Interaktywne wykresy, które zostały wykorzystane do przedstawienia diagramów Gantta i wykresu obciążenia w aplikacjach AGOSIM–APP i AGOCOMP–APP powstały dzięki wykorzystaniu darmowej biblioteki do tworzenia wykresów JFreeChart [117]. Wykresy zostały omówione we wspólnym dodatku, ponieważ funkcje udostępniane przez wykresy i sposób ich użycia jest identyczny w obu aplikacjach.

1. Powiększenie obszaru wykresu

Powiększenie obszaru wykresu następuje po wskazaniu na lewy górny wierzchołek powiększanego obszaru, następnie wciśnięcie lewego przycisku urządzenia wskazującego i przeciągnięciu z wciśniętym lewym przyciskiem do prawego, dolnego wierzchołka powiększanego obszaru. Po zwolnieniu lewego przycisku wybrany obszar zostanie powiększony, a zakres oznaczeń na osiach automatycznie dostosowany.

2. Przesuwanie powiększonego obszaru po wykresie

Przesuwanie powiększonego obszaru po wykresie odbywa się przez przeciąganie wskaźnikiem urządzenia wskazującego przy wciśniętym klawisza Ctrl.

3. Zmiana powiększenia

Zmiana powiększenia jest możliwa dzięki użyciu rolki urządzenia wskazującego lub w opcjach dodatkowych

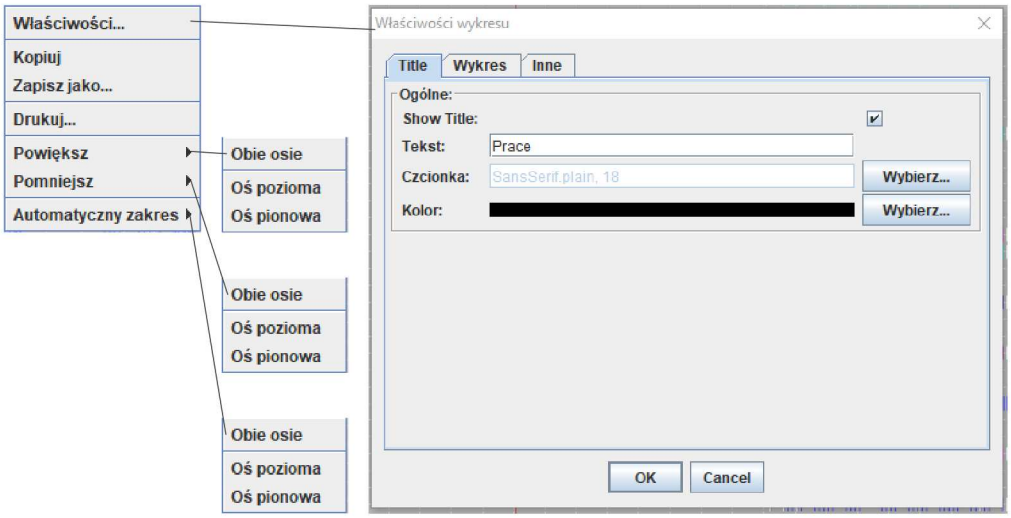
4. Przywrócenie początkowego widoku

Przywrócenie początkowego widoku następuje po wciśnięciu lewego przycisku urządzenia wskazującego i przeciągnięciu z wciśniętym lewym przyciskiem w lewo i w górę. Po zwolnieniu lewego przycisku początkowy widok wykresu zostanie przywrócony.

5. Opcje dodatkowe

Po wciśnięciu prawego przycisku urządzenia wskazującego, gdy wskaźnik znajduje się na wykresie, pojawia się menu o strukturze pokazanej na Rys. D.1. Menu umożliwia

- określenie właściwości wykresu, w tym czcionek opisów i etykiet osi wykresu, kolorów i innych,
- skopiowanie wykresu do schowka systemu,
- zapisanie wykresu do pliku graficznego PNG,
- powiększanie, pomniejszanie, ustalanie automatycznego zakresu osi.



Rys. D.1 Struktura menu opcji dodatkowych interaktywnego wykresu

Źródło: Opracowanie własne

Streszczenie rozprawy doktorskiej

mgr inż. Włodzimierz Wysocki

pt.: *„Agentowo–obiektowy model wsparcia procesu wytwarzania oprogramowania”*

Promotor: prof. Cezary Orłowski

Promotor pomocniczy: dr hab. prof. nadzw. PK Grzegorz Bocewicz

Ocena procesu wytwarzania oprogramowania ma zasadnicze znaczenie zarówno dla ulepszania samego procesu, jak i poprawy jakości wytworzonego oprogramowania. Tradycyjne podejścia, polegające na ręcznej ocenie jakościowej, są nieefektywne, ponieważ są pracochłonne, ograniczone przez kompetencje ekspertów i nacechowane subiektywizmem. Ponadto nie biorą pod uwagę wpływu struktury zespołu projektowego na proces wytwarzania oprogramowania. Model procesu wytwarzania oprogramowania uwzględniający cechy projektu i metodykę wytwarzania wymaga użycia tradycyjnego, obiektowego paradygmatu. Natomiast uwzględnienie wpływu struktury zespołu projektowego, w tym współpracy i komunikacji członków zespołu na proces wytwarzania można uzyskać dzięki rozszerzeniu modelu o paradygmat agentowy. Dzięki połączeniu obydwu paradygmatów modelowania, wynikowy agentowo–obiektowy model pozwoli na zautomatyzowanie oceny, która uwzględni wpływ wszystkich elementów środowiska projektowego na proces wytwarzania oprogramowania. W związku z tymi ograniczeniami zaproponowano podejście oparte na zastosowaniu agentowo-obiektowego modelu procesu wytwarzania oprogramowania. Celem badawczym pracy jest zatem opracowanie metod oceny i planowania procesu wytwarzania i środowiska projektowego, bazujących na agentowo–obiektowym modelu procesu wytwarzania oprogramowania zbudowanym przez autora. W tym celu przeprowadzono analizę środowiska projektowego i procesów wytwarzania według kluczowych metod. Następnie opracowano układ eksperymentu, który posłużył do zbudowania modelu podstawowego w postaci rdzenia i rozszerzeń, reprezentującego wiele aspektów modelowanego procesu, z którego następnie, po przeprowadzeniu analizy dziedzinowej, skonstruowano model uproszczony AGOMO (ang *AGent-Object software process MOdel*). Model ten, przy zastosowaniu podejścia agentowo-obiektowego, został opisany za pomocą równań, algorytmów i diagramów stanu. W celu weryfikacji został zaimplementowany w systemie wieloagentowym JADE, co pozwoliło na przeprowadzenie eksperymentów weryfikujących zasadność replikatywną i predyktywną modelu. Na podstawie modelu opracowano metody: AGOMAS (ang. *AGent-Object Model software process ASsessment method*) do oceny procesu wytwarzania oprogramowania i środowiska projektowego oraz AGOPLAN (ang *AGent-Object model software process PLANning method)* do planowania procesu wytwarzania oprogramowania i środowiska projektowego. Metody zostały zweryfikowane w eksperymentach przeprowadzonych z wykorzystaniem danych dużych projektów informatycznych pochodzących z przemysłu IT. Prace zostały podsumowane oraz przedstawiono wady i zalety modelu. Zamieszczono również plan dalszych prac zarówno badawczych, jak i komercyjnych. Cel utylitarny pracy został zrealizowany w postaci systemu informatycznego umożliwiającego planowanie i ocenę procesu wytwarzania i środowiska projektowego, opartego na skonstruowanym modelu i opracowanych metodach.

Abstract of the doctoral dissertation

MSc Włodzimierz Wysocki

"Agent-object model of support for the software development process"

Evaluation of the software development process is essential both for improving the process itself and for improving the quality of the software produced. Traditional approaches, based on manual qualitative assessment, are ineffective because they are labour-intensive, limited by the competences of experts and characterized by subjectivism. In addition, they do not take into consideration the impact of the project team's structure on the software development process. The model of the software development process taking into account the characteristics of the project and the methodology of development process requires the use of a traditional, object-oriented paradigm. However, accounting for the influence of the structure of the project team, including cooperation and communication of team members on the production process can be obtained by extending the model to an agent paradigm. By combining both modelling paradigms, the resulting agent-object model will allow the automation of the assessment, which will include the impact of all elements of the design environment on the software development process. To overcome the above limitations, an approach based on the use of an agent-object model of the software development process was proposed. The aim of the research work is therefore to develop methods of evaluation and planning of the development process and the project environment, based on the agent-object model of the software development process originated by the author. To this end, an analysis of the design environment and manufacturing processes was carried out according to key methods. Next, an experiment system was developed that served to build a basic model in the form of a core and extensions, representing many aspects of the modelled process, from which then, after field analysis, a simplified model AGOMO (ang AGent-Object software process MOdel) was constructed. This model, using the agent-object approach, was described using equations, algorithms and state diagrams. For verification it was implemented in the multi-agent JADE system, which allowed for conducting experiments verifying the replicative and predictive validity of the model. Based on the model were developed methods: AGOMAS (AGent-Object Model software process ASsessment method) to assess the software process and the design environment, and AGOPLAN (AGent-Object model software process PLANning method) for planning the software process and the project environment. The methods were verified in experiments carried out using data from large industrial software projects. The work has been summarized, the advantages and disadvantages of the model has been presented. A plan for further research and commercial work has been provided also. The utilitarian purpose of the work was implemented in the form of an IT system enabling planning and evaluation of the software process and the project environment based on the constructed model and developed methods.