

Specyfikacja wymagań

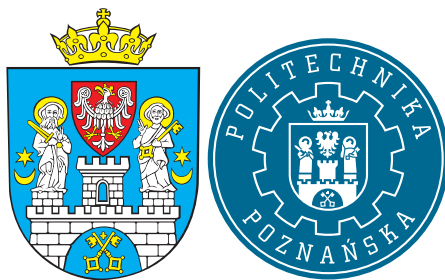
dla

Systemów informatycznych

Instrukcja

Jakub Jurkiewicz, Mirosław Ochodek, Bartosz Walter
opieka merytoryczna Jerzy Nawrocki

Urząd Miasta Poznania
Instytut Informatyki Politechniki Poznańskiej



Skrócony spis treści

Skrócony spis treści · ii

Spis treści · iii

I Wprowadzenie · 1

II Rola wymagań w projektach informatycznych · 4

III Szablon specyfikacji wymagań · 12

IV Instrukcje do szablonu specyfikacji wymagań · 14

A Przypadki użycia · 45

B Język UML · 55

C Modelowanie danych · 61

Bibliografia · 64

Skorowidz · 66

Spis treści

Skrócony spis treści	ii
Spis treści	iii
I Wprowadzenie	1
I.1 Cel instrukcji	1
I.2 Przyjęte zasady typograficzne	1
I.3 Zakres opracowania	1
I.3.1 Organizacja rozdziałów instrukcji 2, I.3.2 Sposoby korzystania z opracowania 2	
II Rola wymagań w projektach informatycznych	4
II.1 Problemy związane z wymaganiami	4
II.2 Rodzaje wymagań	5
II.3 Charakterystyka dobrych wymagań	5
II.3.1 Poprawność 6, II.3.2 Jednoznaczność 6, II.3.3 Kompletność 6, II.3.4 Spójność 7, II.3.5 Uporządkowanie według ważności 7, II.3.6 Weryfikowalność 7, II.3.7 Modyfikowalność 7, II.3.8 Możliwość śledzenia powiązań 7, II.3.9 Wykonalność 8	
II.4 Poziomy wymagań	8
II.4.1 Poziom biznesowy 8, II.4.2 Poziom użytkownika 9, II.4.3 Poziom pojedynczych funkcji 9	
II.5 Priorytetyzacja wymagań	9
II.6 Zalecenia lingwistyczne	10
III Szablon specyfikacji wymagań	12
IV Instrukcje do szablonu specyfikacji wymagań	14
IV.1 Wprowadzenie	14
IV.1.1 Cel dokumentu 14, IV.1.2 Przyjęte zasady w dokumencie 15, IV.1.3 Zakres produktu 15, IV.1.4 Literatura 16	
IV.2 Opis ogólny	16
IV.2.1 Perspektywa produktu 16, IV.2.2 Funkcje produktu 18, IV.2.3 Ograniczenia 18, IV.2.4 Dokumentacja użytkownika 19, IV.2.5 Założenia i zależności 20	

IV.3	Model procesów biznesowych	21
	<i>IV.3.1 Aktorzy i charakterystyka użytkowników 22, IV.3.2 Obiekty biznesowe 24, IV.3.3 Procesy biznesowe 25, IV.3.4 Reguły biznesowe 25</i>	
IV.4	Wymagania funkcjonalne	28
	<i>IV.4.1 Nazwa modułu funkcjonalnego 28</i>	
IV.5	Charakterystyka interfejsów	29
	<i>IV.5.1 Interfejsy użytkownika 29, IV.5.2 Interfejsy zewnętrzne 32</i>	
IV.6	Wymagania pozafunkcjonalne	33
	<i>IV.6.1 Funkcjonalność 34, IV.6.2 Niezawodność 37, IV.6.3 Wydajność 38, IV.6.4 Łatwość utrzymania 39, IV.6.5 Przenośność 41</i>	
IV.7	Inne wymagania	43
IV.8	Dodatek A: Słownik pojęć i terminów	44
IV.9	Dodatek B: Słownik danych	44
IV.10	Dodatek C: Modele analizy	44
IV.11	Dodatek D: Lista spraw otwartych	44
A	Przypadki użycia	45
A.1	Czym są przypadki użycia	45
A.2	Poziomy przypadków użycia	45
A.3	Identyfikacja przypadków użycia	46
	<i>A.3.1 Diagram przypadków użycia 46</i>	
A.4	Podstawowe elementy przypadku użycia	48
A.5	Zapisywanie przypadków użycia	48
	<i>A.5.1 Prezentowanie scenariuszy przypadków użycia 48, A.5.2 Szablony dokumentacji przypadków użycia 50</i>	
A.6	Relacje pomiędzy przypadkami użycia	50
	<i>A.6.1 Relacja zawierania 51, A.6.2 Relacja rozszerzania 52, A.6.3 Relacja specjalizacji 52</i>	
A.7	Modelowanie procesów biznesowych z wykorzystaniem przypadków użycia	53
B	Język UML	55
B.1	Diagram klas	55
B.2	Diagram stanu	57
B.3	Diagram czynności	58
C	Modelowanie danych	61
C.1	Notacja oparta na tekście strukturalnym	61
C.2	Modelowanie danych za pomocą języka UML	62
	Bibliografia	64
	Skorowidz	66



Wprowadzenie

I.1 CEL INSTRUKCJI

Celem instrukcji jest przedstawienie zasad tworzenia specyfikacji wymagań dla systemów informatycznych. Ilustracją do instrukcji jest przykładowa specyfikacja wymagań stworzona dla rzeczywistego systemu informatycznego.

Dokument przeznaczony jest dla analityków wymagań oraz osób odpowiedzialnych za tworzenie specyfikacji wymagań dla produktów informatycznych.

I.2 PRZYJĘTE ZASADY TYPOGRAFICZNE

Dla ułatwienia, w instrukcji potencjalne błędy oraz zalecenia i dobre praktyki są dodatkowo wyeksponowane z wykorzystaniem poniższych konwencji typograficznych.

#n: Potencjalny problem

W ten sposób przedstawione zostaną najczęściej popełniane błędy w specyfikacjach wymagań.

#n: Dobra praktyka

W ten sposób przedstawione zostaną zalecenia, rady i dobre praktyki.

I.3 ZAKRES OPRACOWANIA

Niniejsza instrukcja stanowi jeden z trzech dokumentów wchodzących w skład opracowania pomagającego przy tworzeniu specyfikacji wymagań dla systemów informatycznych:

- **instrukcja** – zawiera wskazówki dotyczące zbierania wymagań oraz ich przedstawiania w formie specyfikacji; instrukcja posiada odwołania do poszczególnych sekcji proponowanego szablonu specyfikacji,
- **edytowalny szablon specyfikacji wymagań** – to szablon specyfikacji w postaci wzorca dokumentu do użycia w praktyce,
- **przykładowa specyfikacja wymagań** – to specyfikacja wymagań stworzona z wykorzystaniem szablonu, obrazuje praktyczne wykorzystanie informacji zawartych w instrukcji.

I.3.1 Organizacja rozdziałów instrukcji

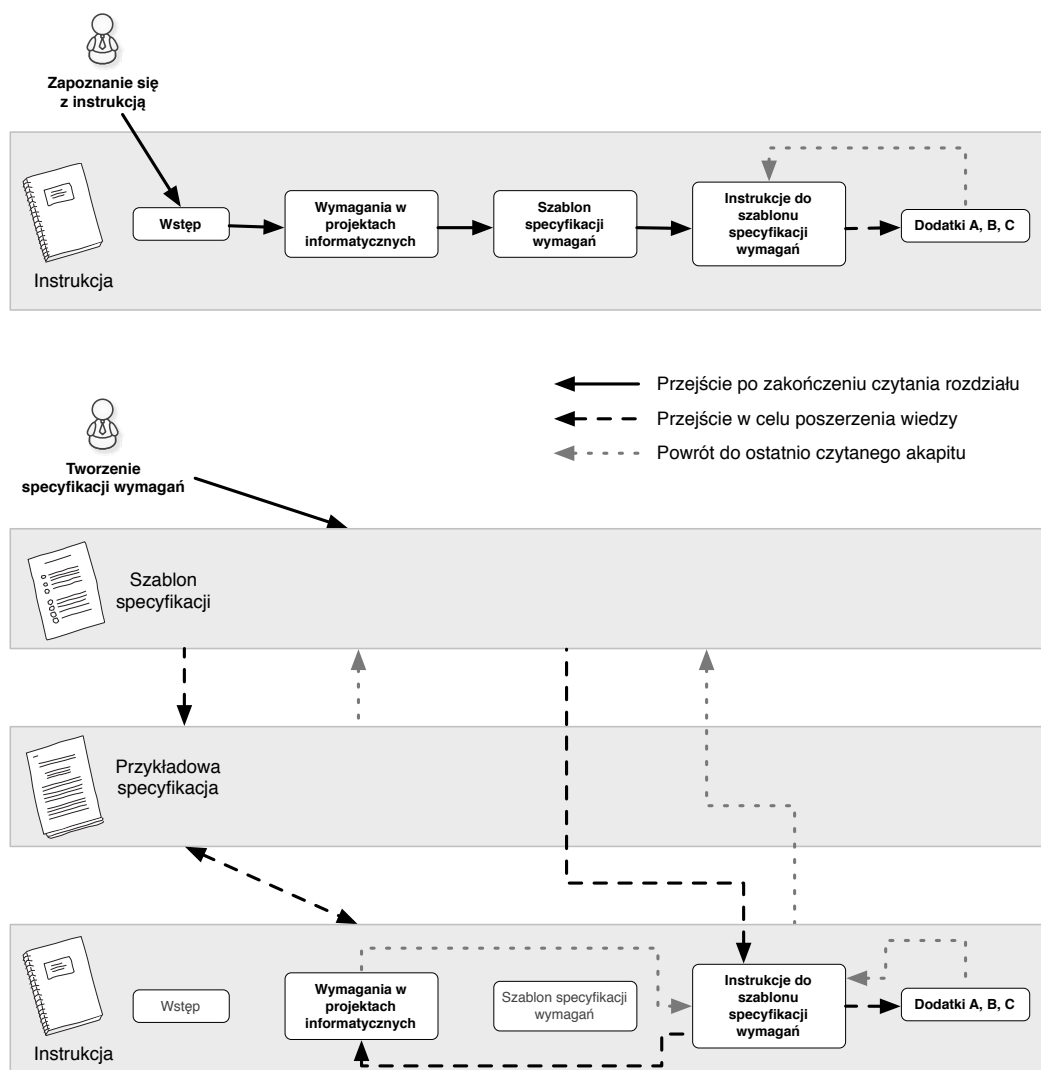
W rozdziale II przedstawiono podstawowe informacje dotyczące roli wymagań dla systemów informatycznych i sposobów ich specyfikowania. W rozdziale III przedstawiono proponowany spis treści specyfikacji wymagań bazujący na standardzie IEEE 830-1998 [iee98b] oraz normie ISO 9126 [iso04]. Spis ten odzwierciedla układ szablonu specyfikacji stanowiącego element niniejszego opracowania. Instrukcje merytoryczne do poszczególnych rozdziałów szablonu specyfikacji zostały przedstawione w rozdziale IV. Niezbędne informacje dotyczące wskazanych w opisie rozdziałów metod specyfikowania wymagań, zostały przedstawione jako dodatki A, B oraz C wraz z odnośnikami do pozycji literaturowych, które pozwolą zainteresowanym czytelnikom poszerzyć wiedzę.

I.3.2 Sposoby korzystania z opracowania

w celu ułatwienia korzystania z niniejszego opracowania poniżej przedstawiono dwa typowe scenariusze użycia posługiwania się nim.

Jeśli celem czytelnika jest zapoznanie się z metodami tworzenia specyfikacji wymagań, należy rozpocząć lekturę od instrukcji, czytając kolejno rozdziały I, II, IV. W rozdziale IV umieszczone są odwołania do poszczególnych metod opisu wymagań przedstawionych w dodatkach A, B, oraz C.

Natomiast jeżeli celem czytelnika jest opracowanie specyfikacji wymagań, należy rozpocząć pracę od edytowalnego szablonu specyfikacji wymagań. Zawiera on komentarze objaśniające zakres poszczególnych rozdziałów. Dodatkowe informacje na dany temat są umieszczone w instrukcji (rozdziały II, IV, Dodatki A, B, C) oraz przykładowej specyfikacji wymagań.



Rysunek I.1: Scenariusze użycia szablonu specyfikacji, niniejszej instrukcji oraz przykładowej specyfikacji wymagań

Rola wymagań w projektach informatycznych

Aby zbudować system informatyczny, należy najpierw poznać jego przeznaczenie – czyli zidentyfikować, zebrać i opisać wymagania, jakie są przed nim stawiane. Wymagania określają zarówno funkcje i cechy, jakie system powinien posiadać, aby móc realizować cele biznesowe użytkownika, jak również pozafunkcjonalne charakterystyki jakościowe, konieczne, aby m.in. praca z nim była efektywna i wygodna dla użytkownika. Zagadnieniami związanymi z wymaganiami zajmuje się dziedzina nazywana inżynierią wymagań, która opisuje proces zbierania wymagań, ich zapisywanie, zarządzanie nimi oraz ich walidację.

II.1 PROBLEMY ZWIĄZANE Z WYMAGANIAMI

Specyfikacja wymagań stanowi podstawę do zawarcia umowy na budowę systemu informatycznego: określa ona jakie warunki musi spełniać zamawiany produkt. Dlatego jej kompletność i dokładność z jednej strony, i brak w niej błędów z drugiej, są niezwykle ważne dla powodzenia przedsięwzięcia. Błędy popełnione przy tworzeniu specyfikacji mogą prowadzić do sporów między zamawiającym a wykonawcą, opóźnień terminu dostawy, przekroczenia budżetu zamówienia czy nawet zarzucenia jego realizacji. Ogólnie, zwykle przyczynami tych sporów są:

- nieumiejętna komunikacja pomiędzy uczestnikami przedsięwzięcia (przede wszystkim między zamawiającym i wykonawcą),
- zmierność wymagań i brak precyzji w ich opisie,
- niepełność specyfikacji w stosunku do oczekiwań zamawiającego.

Pierwszy rodzaj problemów wynika ze zróżnicowania celów, jakie stawiają sobie osoby pracujące z wymaganiami, oraz sposobów ich interpretacji. Do osób uczestniczących w tworzeniu systemu informatycznego można zaliczyć m.in.: *klienta*, który specyfikuje wymagania, *analityka*, który zbiera i zapisuje wymagania,

kierownika projektu, który kieruje pracami w projekcie, *użytkowników końcowych*, którzy będą pracować z oprogramowaniem oraz *programistów*, którzy będą tworzyć oprogramowanie na podstawie wymagań. Z uwagi na inną perspektywę systemu, inaczej wymagania będzie interpretować osoba znająca dziedzinę, w jakiej ma działać oprogramowanie, ale nie znająca aspektów technicznych, a inaczej programista, który potrafi stworzyć oprogramowanie, lecz niekoniecznie zna dziedzinę działania oprogramowania. Różnica ta często powoduje trudności w komunikacji pomiędzy tymi osobami, a w efekcie przekłada się na źle sformułowane i interpretowane wymagania.

Kolejna grupa problemów dotyczy zmienności wymagań. W praktyce nierzadko zdarza się, że zamawiający, mimo spisanej specyfikacji wymagań, dokonuje reinterpretacji celów stawianych przed tworzeniem systemem i modyfikuje je w trakcie realizacji przedsięwzięcia. Ma to istotne konsekwencje dla harmonogramu przedsięwzięcia i kosztów jego realizacji.

Trzecia kategoria dotyczy kształtu i zakresu samej specyfikacji wymagań. Wiele wymagań jest opisywanych bardzo ogólnie lub nawet jest pomijanych w specyfikacji, ponieważ – zdaniem zamawiającego – są one oczywiste i nie wymagają formalnego sformułowania. Opinia ta niekoniecznie jest podzielana przez wykonawców systemu, co prowadzi do trudnych do rozstrzygnięcia nieporozumień w momencie przekazania wykonanego systemu.

II.2 RODZAJE WYMAGAŃ

Najbardziej ogólna klasyfikacja wymagań dzieli je na funkcjonalne i pozafunkcjonalne. Wymagania funkcjonalne opisują funkcje, które system powinien oferować użytkownikom (a zatem stanowią zawsze najważniejszą część specyfikacji). Wymagania pozafunkcjonalne opisują natomiast ograniczenia oraz atrybuty niezwiązane bezpośrednio z funkcjonalnością oprogramowania, a raczej z jego wydajnością, bezpieczeństwem, ergonomią itd.

II.3 CHARAKTERYSTYKA DOBRYCH WYMAGAŃ

Istnieje wiele określeń, jakie powinna posiadać dobra specyfikacja wymagań. Do najważniejszych z nich można zaliczyć [iee98b, Wie03]:

- poprawność,
- jednoznaczność,
- kompletność,
- spójność,
- uporządkowanie według ważności (priorytetyzacja),
- weryfikowalność,
- modyfikowalność,
- możliwość śledzenia zależności,
- wykonalność.

1: Charakterystyka dobrych wymagań jako lista kontrolna

Aby uniknąć błędów i nie pominąć rzeczy istotnych, warto przy tworzeniu specyfikacji wymagań wspierać się listą kontrolną zawierającą konkretne pytania dotyczące własności dobrych wymagań (poprawności, jednoznaczności, kompletności, spójności, ustalonej ważności, weryfikowalności, modyfikowalności, możliwości śledzenia zależności, osiągalności). Po opracowaniu każdego wymagania można wówczas łatwo sprawdzić, czy posiada ono powyższe cechy.

II.3.1 Poprawność

Pojęcie poprawności wymagania oznacza, że odpowiada ono intencji osoby, które je zdefiniowała. Dlatego poprawność wymagania można określić tylko na podstawie zgodności z jego źródłem.

Do najczęściej spotykanych źródeł wymagań można zaliczyć przyszłego użytkownika końcowego tworzonego systemu, wymagania wyższego poziomu (tzw. biznesowego) oraz normy, które ma spełniać wymaganie. Jeśli wymagania tworzą strukturę hierarchiczną, wówczas wymagania niższego poziomu muszą być zgodne z wymaganiami wyższego poziomu.

II.3.2 Jednoznaczność

Jednoznaczność oznacza, że dane wymaganie może zostać zinterpretowane tylko w jeden sposób.

Podstawowy (i w zasadzie nieusuwalny do końca) problem z utrzymaniem jednoznaczności wiąże się z użyciem do zapisu wymagań języka naturalnego. Aby zminimalizować ryzyko niejednoznaczności, należy przy opisie wymagań używać zdań prostych w trybie orzekającym, zbudowanych w oparciu o jednoznaczne słownictwo (więcej wskazówek lingwistycznych znajduje się w rozdziale II.6). Dodatkowo w utrzymaniu jednoznaczności w zakresie terminologii pomaga kompletny słownik terminów (patrz rozdział IV.8), opis obiektów biznesowych (patrz rozdział IV.3.2) oraz słownik danych (patrz rozdział IV.9).

II.3.3 Kompletność

Wymaganie powinno *w pełni* opisywać funkcjonalność, która ma zostać zaimplementowana. Oznacza to, że opis powinien zawierać *wszystkie* informacje potrzebne osobie tworzącej oprogramowanie do zaprojektowania oraz zaimplementowania określonych funkcji, zapisane *w jednym miejscu*.

Jeśli na danym etapie prac pewne niezbędne informacje nie są dostępne, należy to zaznaczyć w opisie wymagania, używając w tym celu konwencyjnej formy, np. poprzedzając opis elementów do uzupełnienia symbolami /DO (do określenia), /DS (do sprecyzowania), czy angielskiego terminu /TBD (z *ang. to be determined*). W trakcie prac nad specyfikacją warto także umieścić informacje o osobie odpowiedzialnej za uszczegółowienie takiego wymagania.

W odniesieniu do całej specyfikacji pojęcie kompletności oznacza, że wszystkie istotne wymagania funkcjonalne, pozafunkcjonalne, odpowiedzi systemu na sytuacje wyjątkowe oraz opisy interfejsów zostały zawarte w dokumencie.

II.3.4 Spójność

Cecha ta odnosi się do całej specyfikacji wymagań. Spójność oznacza, że wymagania nie są wzajemnie sprzeczne lub wykluczające.

II.3.5 Uporządkowanie według ważności

Ze względów praktycznych każde wymaganie powinno posiadać określony priorytet. Ustalona ważność wymagań pozwala kierownikowi przedsięwzięcia na lepsze zarządzanie pracami. Stanowi ona swego rodzaju zabezpieczenie osiągnięcia najbardziej istotnych z punktu widzenia zamawiającego celów w pierwszej kolejności.

Więcej informacji o ustalaniu priorytetów wymagań przedstawiono w rozdziale II.5.

II.3.6 Weryfikowalność

Wymaganie jest weryfikowalne, jeśli istnieje proces (o sensownym koszcie realizacji) umożliwiający automatyczne lub manualne sprawdzenie, czy stworzony produkt informatyczny spełnia to wymaganie.

Przykładem nieweryfikowalnego wymagania może być stwierdzenie „system musi działać wydajnie”, albo „interfejs użytkownika musi być przejrzysty”. W pierwszym przypadku należy sprecyzować weryfikowalne warunki „wydajności”, czyli na przykład średni /maksymalny czas odpowiedzi systemu w określonej konfiguracji, przy założonej liczbie użytkowników oraz określonych warunkach pomiaru. W przypadku drugiego wymagania należy zdefiniować pojęcie *przejrzystości*, np. stwierdzając, że oznacza ono umieszczenie maksymalnie 10 elementów na jednym ekranie.

Warto pamiętać także o określeniu warunków (m.in. konfiguracji oprogramowania), w których będzie następowała weryfikacja. Brak tego typu informacji zwykle prowadzi do różnic w interpretacji wyników procesu weryfikacji. Na przykład określenie: „pomiędzy startem systemu a wyświetleniem okna logowania nie może upłynąć więcej niż 10 sekund” wymaga doprecyzowania, co należy rozumieć przez pojęcie *startu systemu* oraz w jakiej konfiguracji sprzętowej i programowej ten warunek ma być spełniony.

II.3.7 Modyfikowalność

Modyfikowalność odnosi się do dokumentu specyfikacji wymagań, który powinien ułatwiać użytkownikowi wprowadzanie zmian w dokumencie, na przykład poprzez automatyczne generowanie spisów treści, indeksów.

II.3.8 Możliwość śledzenia powiązań

Możliwość śledzenia powiązań wymagań oznacza, że istnieje możliwość znalezienia źródła danego wymagania, związanych z tym wymaganiem wymagań

wyższego poziomu, oraz że istnieje możliwość powiązania wymagań z innymi artefaktami, np. projektem technicznym, kodem programu czy testami.

Zaleca się utrzymywać dwa typy powiązań:

- **powiązania wstecz** – wskazujące na źródła wymagań (np. odwołania do innych dokumentów, wymagań wyższego poziomu),
- **powiązanie w przód** – pozwalające na odnoszenie się do wymagań w przyszłych dokumentach (najczęściej realizowane poprzez nadawanie wymaganiom unikatowych identyfikatorów).

II.3.9 Wykonalność

Cecha ta oznacza, że dane wymaganie jest technicznie realizowalne. Aby to stwierdzić, w wielu przypadkach niezbędna jest wiedza ekspercka na temat określonego zagadnienia (np. trzeba zasięgnąć opinii osoby z większym doświadczeniem w zakresie wytwarzania oprogramowania) lub przeprowadzenie eksperymentu (np. zbudowanie wycinkowego prototyp danej funkcji). Niezapewnienie wykonalności wymagań może prowadzić do trudno rozstrzygalnych sporów między wykonawcą i zamawiającym.

II.4 POZIOMY WYMAGAŃ

Wymagania funkcjonalne przyjmują różną postać w zależności od szerokości perspektywy użytkowników, którzy je definiują. Z tego powodu są one definiowane na różnych poziomach. Zwykle najwyższy poziom, nazywany poziomem biznesowym, określa, w jaki sposób oprogramowanie będzie wspierało działalność biznesową klienta. Poziom pośredni, czyli poziom użytkownika, opisuje sposób, w jaki użytkownik będzie bezpośrednio korzystał z oprogramowania, natomiast poziom najniższy (poziom pojedynczych funkcji) opisuje metody realizacji wymagań na poziomie technicznym.

Definiowanie wymagań na różnych poziomach pozwala opisać te same wymagania z punktu widzenia różnych uczestników procesu wytwarzania oprogramowania, co w konsekwencji umożliwia lepsze zrozumienie tych wymagań oraz dokładniejszą ich analizę w poszukiwaniu potencjalnych problemów (sprzeczności, ograniczeń itp.)

II.4.1 Poziom biznesowy

Określa, w jaki sposób tworzone oprogramowanie odpowiada na potrzeby wynikające z działalności biznesowej klienta. Wymagania na tym poziomie często określają punkt widzenia osób na wysokich stanowiskach w organizacji klienta. Osoby te mają wizję w jaki sposób oprogramowanie będzie wykorzystywane w ich organizacji oraz jakie cele biznesowe można za pomocą systemu osiągnąć (np. zmniejszyć koszty, ulepszyć obsługę klienta, przyspieszyć realizację zamówień, itp.), natomiast nie interesuje ich sposób, w jaki cele te będą realizowane. Wymagania biznesowe nie dostarczają zatem precyzyjnej wiedzy na temat wszystkich funkcji w systemie, jednak pozwalają lepiej poznać jego wizję, zakres oraz kontekst.

II.4.2 Poziom użytkownika

Określa, z jakich funkcji będzie mógł korzystać dany użytkownik podczas pracy z systemem (np. wysyłanie powiadomienia do innego użytkownika, stworzenie faktury, itp). Wymagania te powinny się wpisywać w wymagania biznesowe, tzn. powinny umożliwiać realizację określonych celów organizacji, dlatego ważne jest opisywanie intencji użytkownika, a nie szczegółów technicznych wymagań.

II.4.3 Poziom pojedynczych funkcji

Podczas gdy na poziomie użytkownika nacisk położony jest na cele i intencje użytkownika, to na poziomie pojedynczych funkcji opisywane są szczegółowe wymagania, włączając w to szczegóły implementacyjne oraz techniczne. Jeśli istnieje taka potrzeba, wówczas poszczególne wymagania z poziomu użytkownika można z większą szczegółowością i precyzją opisać na poziomie pojedynczych funkcji.

II.5 PRIORYTYZACJA WYMAGAŃ

Ważnym, choć w wielu przypadkach pomijanym, elementem procesu zbierania wymagań jest określenie priorytetów, czyli wskazanie, które wymagania są najważniejsze z punktu widzenia klienta i które muszą być zrealizowane jak najszybciej*.

Priorytety mają szczególne znaczenie w trzech sytuacjach:

- Priorytety są zabezpieczeniem dla zamawiającego na wypadek, gdyby czas realizacji projektu się wydłużył poza pierwotnie ustalony termin; Wówczas dostawca oprogramowania jest w stanie dostarczyć klientowi najważniejsze dla niego funkcje zanim całe oprogramowanie będzie gotowe.
- Jeżeli przedsięwzięcie realizowane jest etapami, wówczas priorytety stanowią podstawę przy opracowaniu planu realizacji przedsięwzięcia.
- Priorytety wskazują na elementy, które wymagają szczególnej uwagi, ponieważ są najważniejsze z punktu widzenia zamawiającego.

Decyzja dotycząca priorytetów jest zwykle podejmowana przez zamawiającego, jednak – z powodu ich wpływu na koszt realizacji przedsięwzięcia – warto, o ile jest to możliwe, zaangażować w proces ich ustalania osoby ze strony wykonawcy, które są w stanie określić wstępne koszty oraz ryzyko związane z poszczególnymi wymaganiami.

Do określania priorytetów można zastosować prostą 3-stopniową skalę w postaci:

- **Wysoki priorytet** – priorytet ten powinien być przypisany wymaganiom, które są kluczowe dla misji tworzonego systemu oraz są niezbędne do implementacji pozostałej funkcjonalności, bez implementacji wymagań o wysokim priorytecie nie można przejść do implementacji pozostałej funkcjonalności;

*Warto zaznaczyć, że samo określenie priorytetów nie obliguje wykonawcy oprogramowania do dostarczania funkcjonalności zgodnie z ustaloną istotnością wymagań (stanowi jedynie wskazówkę). Jeśli zamawiający chciałby wymusić na wykonawcy realizację wymagań zgodnie z ich priorytetami, powinien umieścić taki zapis jako ograniczenie projektowe w specyfikacji lub zawrzeć stosowny zapis w umowie.

2: Pytania pomagające ustalić priorytety wymagań

W sytuacji, gdy zamawiający nie jest w stanie określić priorytetów wymagań, można przy ich ustaleniu posiłkować się następującymi pytaniami:

- Czy jest inny sposób realizacji danego celu biznesowego?
- Jakie będą konsekwencje, jeśli dane wymaganie nie zostanie zaimplementowane?
- Jaki wpływ na organizację klienta będzie miała opóźniona implementacja danego wymagania?
- Dlaczego użytkownicy końcowi będą niezadowoleni z braku danego wymagania?

- **Średni priorytet** – priorytet ten powinien być przypisany wymaganiom, które uważamy za istotne dla misji systemu, jednak brak ich implementacji w danym momencie nie blokuje możliwości implementacji innych wymagań.
- **Niski priorytet** – priorytet ten powinien być przypisany wymaganiom, które nie są konieczne do realizacji misji systemu, najczęściej stanowią one dodatki, bez których możliwe jest prawidłowe wykorzystanie funkcjonalności oprogramowania.

Powyższą skalę można dostosowywać do własnych potrzeb, zwiększając lub zmniejszając jej granularność.

Priorytety można nadawać wymaganiom na różnych poziomach, np. na poziomie całego modułu funkcjonalnego, charakterystyki pozafunkcjonalnej, przypadku użycia, poszczególnych elementów przypadku użycia (np. sposób postępowania opisany w sekcji wyjątków może mieć niski priorytet w porównaniu do scenariusza głównego).

II.6 ZALECENIA LINGWISTYCZNE

Ponieważ wymagania są zapisywane za pomocą języka naturalnego, dlatego sposób jego użycia ma bardzo istotny wpływ na ich jakość oraz sposób, w jaki wymagania są interpretowane. Jeśli użyte będą wyrażenia niejednoznaczne, niejasne lub bardzo skomplikowane, to możliwe, że osoba czytająca specyfikację będzie miała problem z ich poprawną interpretacją.

Poniżej zaprezentowane są podstawowe zasady dotyczące używanego języka przy zapisie wymagań:

- Należy używać pełnych zdań, poprawnych pod względem gramatycznym, ortograficznym i interpunkcyjnym.
- Zdania i akapity powinny być krótkie i spójne. Należy unikać zdań wielokrotnie złożonych połączonych wieloma spójnikami.
- Przy opisie akcji należy określać kto ją wykonuje (podmiot) oraz co jest jej przedmiotem.
- Należy bezwzględnie zapisywać każde wymaganie tylko w jednym miejscu, a w pozostałych miejscach umieszczać odniesienia do niego (np. poprzez unikatowy identyfikator wymagania).

- Zawsze należy kłaść nacisk na precyzję opisu wymagań, nawet za cenę powtórzeń terminów i obniżenia jakości stylistycznej tekstu.
- Wymagania, które nie są dostatecznie sprecyzowane lub niedokończone, należy wyraźnie oznaczyć (najlepiej jest stosować w tym celu stałe oznaczenia) i umieścić odniesienie do nich w rozdziale 7 specyfikacji wymagań.
- Wymagania powinny być zapisane w sposób umożliwiający ich późniejszą weryfikację w produkcie końcowym.
- Należy unikać wyrażań wyrażających subiektywną opinię, np. kilka, lepszy, elastyczny, opcjonalnie, wydajny, łatwy, akceptowalny, odpowiedni, powinien, "etc.", może, optymalnie, praktycznie, itp.
- W miarę możliwości należy stosować aktywną stronę czasowników (np. "System wysyła dane") zamiast biernej ("Dane są wysyłane").
- Jeśli do opisu funkcjonalności lub własności systemu używa się stwierdzeń typu: "System powinien / musi ...", warto zwrócić uwagę na to, że słowo *musi* ma mocniejszy wydźwięk niż słowo *powinien*. Dlatego dobrze jest ograniczyć wykorzystanie słowa *musi* do przypadków, w których szczególnie ważne jest podkreślenie istotności danego wymagania czy własności systemu.
- Wszystkie skróty i terminy powinny zostać wyjaśnione w słowniku.
- Należy konsekwentnie korzystać z terminów zdefiniowanych w słowniku, unikając synonimów (szczególnie jeżeli ich znaczenie nie jest identyczne z terminem źródłowym).
- Warto (w miarę możliwości) korzystać z rysunków w celu lepszego opisu rzeczywistości, należy jednak pamiętać o zachowaniu spójności rysunków z opisem słownym.
- Najistotniejsze fragmenty można wyróżniać za pomocą rozmaitych rozwiązań typograficznych np. kolorów, pogrubień, specjalnych czcionek lub oznaczeń; przyjętą konwencję i sposób jej interpretacji należy opisać w rozdziale 1.2 specyfikacji wymagań.

Szablon specyfikacji wymagań

Zaproponowany spis treści specyfikacji wymagań dla systemów informatycznych został opracowany na bazie standardu IEEE 830-1998[iee98b] i normy ISO 9126 [iso04].

1. Wprowadzenie (str.14)
 - 1.1 Cel dokumentu (str.14)
 - 1.2 Przyjęte zasady w dokumencie (str.15)
 - 1.3 Zakres produktu (str.15)
 - 1.4 Literatura (str.16)
2. Opis ogólny (str.16)
 - 2.1 Perspektywa produktu (str.16)
 - 2.2 Funkcje produktu (str.18)
 - 2.3 Ograniczenia (str.18)
 - 2.4 Dokumentacja użytkownika (str.19)
 - 2.5 Założenia i zależności (str.20)
3. Model procesów biznesowych (str.21)
 - 3.1 Aktorzy i charakterystyka użytkowników (str.22)
 - 3.2 Obiekty biznesowe (str.24)
 - 3.3 Procesy biznesowe (str.25)
 - 3.4 Reguły biznesowe (str.25)
4. Wymagania funkcjonalne (str.28)
 - 4.x Nazwa modułu funkcjonalnego (str.28)
 - 4.x.1 Opis i priorytet (str.28)
 - 4.x.2 Przypadki użycia (str.28)
 - 4.x.3 Specyficzne wymagania funkcjonalne (str.28)
5. Charakterystyka interfejsów (str.29)
 - 5.1 Interfejsy użytkownika (str.29)
 - 5.2 Interfejsy zewnętrzne (str.32)
 - 5.2.1 Interfejsy sprzętowe (str.33)
 - 5.2.2 Interfejsy programistyczne (str.33)

5.2.3 Interfejsy komunikacyjne (str.33)

6. Wymagania pozafunkcjonalne (str.33)

7. Inne wymagania (str.43)

Dodatek A: Słownik pojęć i terminów (str.44)

Dodatek B: Słownik danych (str.44)

Dodatek C: Modele analizy (str.44)

Dodatek D: Lista spraw otwartych (str.44)

W stosunku do szablonu zawartego w standardzie IEEE 830-1998 dokonano zmian polegających na reorganizacji struktury spisu treści (zredukowano poziom zagłębień) oraz wzbogacono go o dodatkowe rozdziały.

Dodatkowo rozdział poświęcony wymaganiom pozafunkcjonalnym powstał w oparciu o zbiór charakterystyk jakościowych zawartych w normie ISO 9126.

Instrukcje do szablonu specyfikacji wymagań

Rozdział ten zawiera wskazówki dotyczące zawartości poszczególnych rozdziałów specyfikacji wymagań przedstawionych w rozdziale III.

Należy pamiętać, że w zależności od specyfiki realizowanego przedsięwzięcia informatycznego, część informacji zawartych w rozdziałach szablonu może okazać się zbyteczna lub nieadekwatna. W takim przypadku można pominąć dany fragment w tworzonej specyfikacji wymagań. Jeśli okaże się, że problem dotyczy całego rozdziału, warto (dla uniknięcia niespójności w numeracji rozdziałów) pozostawić nagłówek rozdziału, umieszczając adnotację, że nie dotyczy on opisywanego systemu.

IV.1 WPROWADZENIE

Celem tego rozdziału jest omówienie struktury oraz celu dokumentu specyfikacji wymagań. Po przeczytaniu tego rozdziału czytelnicy powinni lepiej rozumieć dokument oraz będą wiedzieć, w jaki sposób powinni go czytać, aby odnaleźć interesujące ich informacje.

IV.1.1 Cel dokumentu

W tym rozdziale należy określić produkt lub aplikację, której dotyczy ten dokument (wraz z numerem wersji, np. aplikacja XYZ wersja 2.4). Jeśli dokument dotyczy jedynie modułu większego systemu, należy to wyraźnie zaznaczyć.

Jest to również miejsce dla pełnej listy potencjalnych czytelników dokumentu. Należy opisać sposób, w jaki dokument ten jest skonstruowany oraz gdzie należy szukać określonych informacji. Można również zasugerować sekwencję czytania dokumentu, która będzie odpowiednia dla poszczególnych czytelników.

3: Potencjalni czytelnicy specyfikacji

Do potencjalnych czytelników dokumentu specyfikacji wymagań można zaliczyć:

- **Klienta**, który zleca wykonanie oprogramowania oraz określa wymagania wobec niego,
 - **Użytkowników**, którzy będą wykorzystywali oprogramowanie w sposób pośredni lub bezpośredni,
 - **Analityków**, którzy będą zapisywali wymagania,
 - **Programistów**, którzy będą analizowali, projektowali, implementowali oraz utrzymywali oprogramowanie,
 - **Testerów**, którzy będą weryfikowali jakość oprogramowania,
 - **Twórców dokumentacji**, którzy będą tworzyli dokumentację użytkownika, materiały szkoleniowe, pomoc systemową, itp.,
 - **Kierowników projektu**, którzy będą planowali wykonanie projektu i będą kierowali wszystkimi pracami,
 - **Prawników**, którzy będą określali prawny aspekt wymagań,
 - **Sprzedawców, osoby z działu marketingu, dział wsparcia** czyli osoby, które będą pracowały z oprogramowaniem i/lub jego użytkownikami.
-

IV.1.2 Przyjęte zasady w dokumencie

W miejscu tym należy opisać wszelkie standardy oraz konwencje zapisu używane w dokumencie, w tym rodzaje czcionek, pogrubienia, podkreślenia, użycie kolorów, oznaczenia obrazkowe, itp. Na przykład, jeśli nazwy pewnego typu wymagań będą oznaczane wytłuszczoną czcionką, wówczas znaczenie tego zabiegu powinno być wyjaśnione w tym rozdziale.

IV.1.3 Zakres produktu

Rozdział ten ma na celu przybliżyć czytelnikowi ogólną wizję systemu, aby opisywane później poszczególne wymagania umieścić we właściwym kontekście. Z jednej strony rozdział ten powinien dostarczyć ogólną wiedzę na temat projektu osobom, które nie zapoznały się ze szczegółowymi wymaganiami, a chcą mieć ogólny przegląd projektu, a z drugiej – być wstępem dla osób, które będą zainteresowane szczegółowymi informacjami.

W rozdziale tym należy w krótki sposób opisać zakres i cel projektu, podać również jego nazwę, która go będzie identyfikować. Jeśli istnieje taka możliwość, należy odnieść się do celów biznesowych, strategii, problemów organizacji, dla której powstaje opisywane oprogramowanie. Jeśli wizja systemu została już opisana w innym dokumencie, należy umieścić odwołanie do niego. Opisuując zakres produktu można również określić to, co nie jest jego zadaniem, jeśli może być to niejasne dla czytelnika.

IV.1.4 Literatura

W miejscu tym należy zamieścić listę dokumentów lub innych zasobów, do których specyfikacja wymagań się odwołuje (np. w postaci łączy hipertekstowego, jeśli istnieje taka możliwość). Dokumenty, o których warto pamiętać to przede wszystkim:

- standardy, których tworzone oprogramowanie ma przestrzegać,
- kontrakty dotyczące tworzonego systemu,
- inne powiązane specyfikacje wymagań,
- specyfikacje interfejsów,
- książki i artykuły zawierające wiedzę potrzebną przy realizacji projektu,
- przepisy prawne, z którymi oprogramowanie ma być zgodne.

IV.2 OPIS OGÓLNY

Celem tego rozdziału jest zarysowanie wizji produktu, jaką posiada organizacja zamawiająca system, oraz określenie w jakich warunkach tworzone oprogramowanie będzie wykorzystywane.

IV.2.1 Perspektywa produktu

Pierwsza część rozdziału służy przedstawieniu ogólnej perspektywy tworzonego systemu, jego otoczenia oraz genezy (o ile jej znajomość może mieć znaczenie dla prawidłowego poznania celów stawianych przed systemem). Pozwala to na zrozumienie kontekstu, w jakim system będzie funkcjonował, oraz sprzyja prawidłowej (zgodnej z intencjami autorów specyfikacji) interpretacji. Jeżeli system stanowi jedynie część większej całości, należy formułowane wobec niego wymagania umieścić na jej tle.

Wygodnym sposobem przedstawienia otoczenia tworzonego systemu, czyli jego powiązań z różnymi użytkownikami oraz innymi systemami, jest diagram kontekstu.

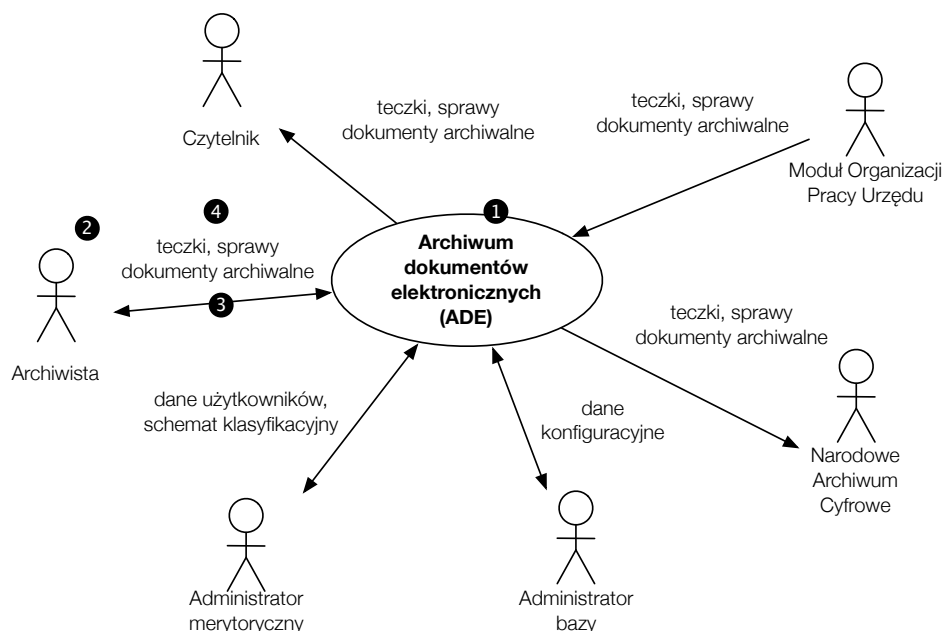
Diagram
kontekstu

4: Korzystaj z diagramu kontekstu do opisu otoczenia systemu

Diagram kontekstu pozwala w skrócie przedstawić otoczenie tworzonego systemu poprzez pokazanie najważniejszych interfejsów komunikacyjnych z innymi systemami lub klasami użytkowników, wraz z informacją o wpływie danych.

Diagram kontekstu (ang. *context diagram*) [Wie03] służy do graficznego przedstawienia otoczenia tworzonego systemu. Pozwala przedstawić granice tworzonego systemu, użytkowników (rozumianych jako role, a nie osoby) oraz systemy z którymi ma współpracować. Przykładowy diagram kontekstu przedstawiono na rysunku IV.1.

W centralnym punkcie diagramu umieszcza się element graficzny symbolizujący tworzony system (1). Warto zwrócić uwagę, że na diagramie kontekstu system ten przedstawiony jest jako tzw. "czarna szkrzynka". Oznacza to, że na tym etapie



Rysunek IV.1: Przykładowy diagram kontekstu

nie są istotne mechanizmy wewnętrzne systemu, ale jego oddziaływanie z otoczeniem.

Następnym etapem tworzenia diagramu jest identyfikacja aktorów (ludzi, systemów) którzy będą współpracować z tworzoną systemem. Można ułatwić sobie to zadanie, udzielając odpowiedzi na następujące pytania [WA02]:

- **Użytkownicy (ludzie)**
 - Kto potrzebuje informacji dostępnych w systemie?
 - Jakich informacji potrzebuje?
 - Czy system udostępnia je na żądanie, okresowo, czy też na oba sposoby?
 - Kto udostępnia informacje systemowi?
 - Jakie informacje ten ktoś udostępnia?
 - Czy system musi obowiązkowo udzielać odpowiedzi na te żądania?
- **Inne systemy**
 - Jakie systemy mogą wywołać reakcję tworzonego systemu?
 - Czy tworzony system może, czy też musi odpowiadać na te żądania?
 - Czy żądania są regularne, czy nieprzewidywalne w czasie?
 - Jakie systemy muszą odpowiadać na żądania wysyłane przez tworzony system?
 - Czy tworzony system wysyła żądania okresowo?

Zidentyfikowanych aktorów umieszcza się na diagramie dookoła symbolu reprezentującego tworzony system (2).

Kolejny krok tworzenia diagramu polega na powiązaniu łukami poszczególnych aktorów z tworzoną systemem (3). Łuki te obrazują kierunek przepływu

informacji. Można je dodatkowo opatrzyć etykietami opisującymi przekazywane dane (4). Z uwagi na czytelność zaleca się, aby na diagramie stosować nazwy logicznych grup danych, a nie pojedynczych atrybutów. Na przykład, jeśli tworzony system komunikuje się z systemem kadrowym w celu pozyskania informacji o pracownikach, to łuk łączący system kadrowy z tworzonym systemem powinien mieć etykietę *dane kadrowe pracowników* (a nie *imię, nazwisko, data urodzenia itd.*).

IV.2.2 Funkcje produktu

Rozdział ten zawiera przegląd i ogólny opis kluczowych funkcji tworzonego produktu. Szczegółowy opis wymagań funkcjonalnych znajduje się w rozdziale 4 specyfikacji, dlatego w tym rozdziale są przedstawione funkcje najważniejsze z punktu widzenia podstawowego przeznaczenia tworzonego systemu produktu oraz najistotniejsze powiązania między nimi.

Rozdział ten można stworzyć na podstawie rozdziału 4 specyfikacji, przedstawiając główne moduły funkcjonalne oraz prezentując diagramy przypadków użycia (patrz rozdział IV.4 instrukcji).

Diagram przypadków użycia

IV.2.3 Ograniczenia

Ten rozdział powinien przedstawiać wszystkie ograniczenia, które zawężają swobodę wykonawcy w zakresie sposobu realizacji produktu.

Ograniczenia można podzielić na dwie zasadnicze grupy: **implementacyjne**, związane z technicznymi aspektami tworzenia oprogramowania i **projektowe**, związane z samym procesem wytwarzania.

Najbardziej typowe ograniczenia implementacyjne dotyczą:

- technologii, narzędzi, języków programowania, baz danych, które muszą być wykorzystane w produkcie – takie informacje mają duże znaczenie dla potencjalnych wykonawców z uwagi na ich specjalizację w tworzeniu oprogramowania z wykorzystaniem konkretnych technologii;
- protokołów komunikacyjnych i standardów wymiany danych – niektóre technologie wykorzystywane przez potencjalnego wykonawcę oprogramowania mogą nie wspierać określonych protokołów albo formatów danych;
- ograniczeń sprzętowych – m.in. konieczności wykorzystania określonych urządzeń, dostępności pamięci operacyjnej, przepustowości łącz, kosztu urządzeń itd.;
- standardów kodowania i dokumentacji kodu – postawienie takiego ograniczenia ma sens wówczas, gdy zamawiający będzie rozwijał dostarczony produkt w przyszłości lub zapewniał jego obsługę. Należy pamiętać, że takie ograniczenia zawsze zwiększają nakład pracy ze strony wytwórcy oprogramowania, a co za tym idzie – koszt wytworzenia systemu. Z drugiej strony, brak takiego ograniczenia może prowadzić w przyszłości do uzależnienia zamawiającego od wybranego wykonawcy;
- konwencji związanych z interfejsem użytkownika,
- zgodności z różnorodnymi politykami i ustawami (np. wewnętrznymi politykami bezpieczeństwa, ustawą o ochronie danych osobowych itd.).

Ograniczenia implementacyjne

Oprócz typowych ograniczeń implementacyjnych w systemie mogą wystąpić także ograniczenia projektowe, związane z realizacją i zarządzaniem projektem, na

Ograniczenia projektowe

przykład dotyczące maksymalnego czasu realizacji projektu, wymaganych form audytu, stosowanego procesu testowania itp.

Szczególną uwagę należy zwrócić na kompletność listy ograniczeń. Nawet pozornie proste zadanie programistyczne może okazać się trudne przy pewnych ograniczeniach, które niekoniecznie są znane *a priori*.

Każde opisywane ograniczenie powinno posiadać krótkie uzasadnienie, wskazujące na przyczynę jego zdefiniowania. Brak tej informacji może prowadzić do błędnej interpretacji poszczególnych ograniczeń lub ich bagatelizacji. Poza tym obecność uzasadnienia (a przynajmniej jego zidentyfikowanie) ma również wartość dla zamawiającego: większość ograniczeń powoduje wzrost kosztów wytworzenia produktu, dlatego w najlepszym interesie zamawiającego jest aby postawione ograniczenia były racjonalne.

5: Pamiętaj o podaniu uzasadnienia ograniczeń

Uzasadnienie ograniczeń dostarczy wykonawcy oprogramowania informacji, które ograniczenia mogą podlegać negocjacji (np. dostarczenie alternatywnego rozwiązania), a które muszą być bezwzględnie respektowane.

IV.2.4 Dokumentacja użytkownika

Dokumentacja użytkownika jest istotną częścią systemu przede wszystkim z punktu widzenia zamawiającego. Natomiast w wielu przypadkach wykonawcy oprogramowania koncentrują się przede wszystkim na zadaniach implementacyjnych, nie biorąc pod uwagę czasu potrzebnego na wytworzenie dokumentacji użytkownika [Jon07]. Dlatego dobrze jest szczegółowo określić w specyfikacji zakres dokumentacji, którą wykonawca będzie zobowiązany dostarczyć razem z produktem.

Dokumentacja
użytkowni-
ka

Wymagania dotyczące dokumentacji powinny zawierać [Wit07]:

- **nazwę/rodzaj dokumentu** – należy nazwać poszczególne dokumenty (np. “pomoc online”, “podręcznik użytkownika”, “instrukcja operacyjna”, “procedury bezpieczeństwa” itp.),
- **opis zawartości** – wymagania określające jakie informacje ma zawierać dany dokument,
- **format** – można określić dokładnie format (np. MS Word 2007, XHTML itd.), lub bardziej ogólnie, jak na przykład forma drukowana, pomoc online, pomoc kontekstowa wbudowana w system itp.,
- **standard** – należy go wskazać, jeśli dokumentacja ma być zgodna z określonym standardem,
- **język dokumentacji** – element istotny w przypadku, gdy dostęp do dokumentacji będą miały osoby różnych narodowości.

Spośród wymienionych atrybutów należy przede wszystkim określić rodzaj i formę dokumentacji (np. dokumentacja użytkownika w formie książki, pomoc kontekstowa, podręcznik online).

IV.2.5 Założenia i zależności

W rozdziale tym należy przedstawić założenia i czynniki mające wpływ na przedstawione w dokumencie wymagania.

Założenie to zasada stanowiąca podstawę dalszych wywodów lub dalszego postępowania. Formułując wymagania ich autor zwykle opiera się na pewnych założeniach. Niespełnienie któregoś z nich jest istotnym czynnikiem ryzyka i wiąże się najczęściej ze zmianą wymagań w późniejszych etapach projektowych.

Przykładem założenia może być stwierdzenie, że wersja określonego systemu operacyjnego będzie dostępna na platformę sprzętową, która ma zostać wykorzystana do budowy produktu. W przypadku niespełnienia tego założenia sformułowane wymagania będą musiały zostać rozszerzone o adaptację innego (dostępnego) systemu operacyjnego, stworzenie go od podstaw, czy też zmianę platformy sprzętowej na obsługiwaną przez system operacyjny. Jak łatwo się domyślić, ma to olbrzymi wpływ na koszt realizacji systemu.

Duże zagrożenie dla sukcesu projektu stanowią *ukryte założenia*, czyli takie, które nie zostały udokumentowane, a stanowiły podstawę specyfikowania wymagań.

Ukryte
założenia

1: Eliminuj ukryte założenia wobec wymagań

Wagę ukrytych założeń prezentuje następujący przykład. Zamawiający chciałby, żeby w zamawianym systemie użytkownicy mieli możliwość wysyłania wiadomości e-mail, tak jak jest to w posiadanym przez niego systemie obsługi obiegu dokumentów. Zapisuje więc ogólne wymaganie o możliwości wysyłania wiadomości e-mail.

Potencjalny dostawca przyjmuje niejawnie założenie, że na komputerach użytkowników będzie zainstalowany klient poczty i nie usiłuje uściślić wymagań dotyczących tej funkcjonalności z zamawiającym.

Podczas wdrożenia systemu okazuje się, że z przyczyn bezpieczeństwa na komputerach użytkowników nie ma zainstalowanego oddzielnego klienta poczty, a jest on wbudowany w system obiegu dokumentów, który wykorzystywany jest w organizacji zamawiającego. Okazuje się także, że jest to rozwiązanie zamknięte i nie ma możliwości integracji z nim nowego systemu.

Taki błąd w komunikacji odnośnie założeń spowodowałby konieczność zdefiniowania dodatkowych (nieprzewidzianych wcześniej) modułów funkcjonalnych i ich implementację.

Aby uniknąć opisanego problemu, należałoby umieścić następujące założenie: *istnieje możliwość integracji z klientem pocztowym, wbudowanym w posiadany przez zamawiającego system obsługi X, w zakresie określonym przez interfejs Y.*

Druga istotna informacja, która powinna zostać zawarta w tym rozdziale, to zależność od *czynników zewnętrznych*. Są to czynniki, które mogą mieć wpływ na zawarte w dokumencie wymagania, ale znajdują się poza kontrolą projektu.

Czynniki
zewnętrzne

Bardzo często zależności zewnętrzne dotyczą sytuacji, w której implementacja wymagań zależy od dostarczenia produktów (komponentów) przez innych dostawców. Przykładowa zależność mogłaby zostać opisana w następujący sposób:

tworzony system ma zostać zintegrowany z istniejącym systemem obiegu dokumentów, poprzez interfejs komunikacyjny X, który zostanie zaimplementowany przez zewnętrzną firmę Y w terminie do Z.

Jeśli zależności zewnętrzne są opisane także w innych dokumentach (np. w planie projektu), należy koniecznie do nich się odwołać.

IV.3 MODEL PROCESÓW BIZNESOWYCH

Proces biznesowy opisuje zbiór czynności w określonym środowisku (biznesowym), które muszą być wykonane, aby osiągnąć istotny z punktu widzenia organizacji cel, uwzględniając powiązania między procesami oraz wykorzystywane zasoby [PE00].

Opis procesów biznesowych stanowi część większego zagadnienia nazywanego modelowaniem procesów biznesowych (*ang. business process modeling*). Stworzenie modelu biznesowego ma na celu ustalenie w jaki sposób działa dana organizacja. W wyniku analizy biznesowej powstaje model organizacji na różnych poziomach szczegółowości: model biznesowy, mapy procesów, przepływy pracy w ramach poszczególnych procesów.

W praktyce modelowanie procesów biznesowych jest często czynnością poprzedzającą specyfikowanie wymagań dla systemów informatycznych. W wielu przypadkach jest ono powiązane z re-inżynierią procesów, czyli próbą zoptymalizowania istniejących procesów, tak aby zwiększyć wydajność, a tym samym zyski organizacji.

Standard IEEE 1362-1998 [iee98a] zaleca przed przystąpieniem do prac nad systemem informatycznym sporządzenie dokumentu nazywanego *Concept of Operations*, który opisuje zastane procesy biznesowe, uzasadnienie zmian oraz proponowany system. Taki opis jest istotny, ponieważ pozwala zrozumieć *dlaczego system musi dostarczać określone funkcje, oraz kto i kiedy będzie z nich korzystał*. Taka informacja jest przydatna, zwłaszcza gdy wymagania zapisane są w postaci stwierdzeń typu: „system powinien ...”. Bez znajomości szerszego kontekstu dostawca oprogramowania, nie będzie w stanie zrozumieć powiązań pomiędzy wymaganiami i roli poszczególnych z nich.

Rola procesów biznesowych

Wystarczający (dla potrzeb specyfikacji wymagań) model biznesowy powinien zawierać*:

- opis aktorów i charakterystykę użytkowników,
- opis głównych obiektów biznesowych,
- opis procesów biznesowych,
- zbiór reguł biznesowych.

Istnieje wiele notacji umożliwiających przedstawianie procesów biznesowych. Do najbardziej popularnych należą notacje graficzne, np. UML [PE00] czy BPMN [Pio07], oraz notacje tekstowe, np. opisane w niniejszej instrukcji przypadki użycia [Jac87].

*Modelowanie procesów biznesowych jest odrębną, złożoną dziedziną i wykracza poza zakres tej instrukcji. Dlatego w ramach instrukcji zawarto zakres wiedzy, niezbędny do stworzenia opisu procesów biznesowych na potrzeby specyfikacji wymagań.

W stosunku do notacji graficznych, zapis w postaci języka naturalnego posiada tę przewagę, że czytelnik nie musi zostać zapoznany z elementami notacji, aby mógł zrozumieć opis procesu biznesowego.

IV.3.1 Aktorzy i charakterystyka użytkowników

Pod pojęciem aktora kryją się ludzie, organizacje, systemy informatyczne oraz urzędnicy, które komunikują się między sobą, aby osiągnąć istotny cel. Pojęcie to zostało już wcześniej użyte jako element diagramu kontekstu.

Na poziomie opisu procesów biznesowych aktorzy reprezentują najczęściej archetyp osoby pełniącej określone funkcje w organizacji. Należy przy tym zaznaczyć, że mowa tutaj o typach uczestników procesu, a nie konkretnych osobach. Dlatego, jeżeli Jan Kowalski jest osobą, która zajmuje się przyjmowaniem dokumentów do archiwum, to jego reprezentacją będzie aktor Archiwista, a nie Jan Kowalski.

Kim/czym
jest aktor?

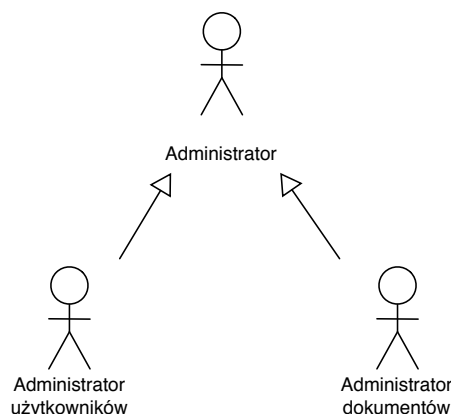
Co więcej, rola aktora w procesie niekoniecznie musi odpowiadać stanowisku w organizacji. Oczywiście, w niektórych przypadkach może wystąpić taka zbieżność (np. w organizacji może istnieć stanowisko Administrator Systemu i w opisie systemu także może wystąpić aktor o takiej nazwie), jednak w ogólności nie jest to regułą. W rzeczywistości ta sama osoba fizyczna bardzo często pełni różne role (wciela się w różnych aktorów) w ramach różnych procesów.

2: Błędy w identyfikacji aktorów

Podczas identyfikacji aktorów należy pamiętać, że:

- aktor odzwierciedla *rolę w procesie* a nie konkretną osobę,
- jedna osoba w organizacji może wcielać się w różnych aktorów,
- nazwa aktora odzwierciedla rolę jaką pełni w procesie, co niekoniecznie musi odpowiadać stanowiskom w organizacji.

Oprócz nazwy, każdy aktor powinien posiadać unikalny identyfikator (nazwa może także pełnić także rolę identyfikatora, o ile pozostaje unikatowa) oraz opis. W opisie należy wyjaśnić rolę danego aktora w procesach i organizacji. Jeśli dany aktor jest powiązany z danym stanowiskiem w organizacji, wówczas w opisie można umieścić stosowną adnotację.



Rysunek IV.2: Przykład relacji specjalizacji pomiędzy aktorami (*Administrator użytkowników* i *Administrator dokumentów* są specjalizacjami aktora *Administrator*).

Przykładowy opis aktora może wyglądać następująco:

AKT_REF: Referent

Opis: Referent to pracownik załatwiający merytorycznie daną sprawę i przecho-
wujący dokumentację sprawy w trakcie jej załatwiania.

Przy identyfikowaniu aktorów istnieje możliwość tworzenia nowego aktora ja-
ko specjalizacji innego, już istniejącego aktora. Na przykład, w systemie istnieje
pewna część zadań przeznaczona dla każdego Administratora. Dodatkowo pewne
funkcje są dostępne dla Administratora użytkowników, a inne dla Administratora
dokumentów. Można w takiej sytuacji stworzyć aktora Administrator i wprowa-
dzić relację specjalizacji pomiędzy tym aktorem oraz Administratorem użytkow-
ników i dokumentów. Aktorzy ci będą posiadać wszystkie cechy Administratora,
jednocześnie uzupełniając je o swoje specyficzne własności. Relację taką można
przedstawić w formie graficznej analogicznej do rysunku IV.2. Relacja wyrażo-
na jest poprzez łuk skierowany z zamkniętym grotem, poprowadzony od aktora
szczegółowego (np. Administratora użytkowników) do aktora bardziej ogólnego
(Administrator).

Oprócz opisów aktorów w rozdziale tym należy także przedstawić charakte-
rystykę poszczególnych grup użytkowników. Pozwoli ona lepiej zrozumieć czy-
telnikowi, kto będzie pracował z tworzonego systemem. Najbardziej przydatne
informacje obejmują poziom wykształcenia przyszłych użytkowników (ze szcze-
gólnym zwróceniem uwagi na przygotowanie do pracy z systemami informa-
tycznymi), ewentualne dysfunkcje uniemożliwiające korzystanie z określonych
urządzeń (np. na niektórych stanowiskach zatrudnione są osoby niepełnosprawne
itp.). Przykładową charakterystykę użytkowników przedstawiono w tabeli IV.1.

Charaketrystyka
użytkowni-
ków

Tabela IV.1: Przykładowa tabela przedstawiająca charakterystykę użytkowników.

Kadra zarządzająca	W urzędzie zatrudnionych jest 5 dyrektorów oraz 30 kierowników. Wszystkie osoby posiadają wyższe wykształcenie. Biegle obsługują pakiet biurowy (arkusz kalkulacyjny, edytor tekstu). Grupa ta będzie wykorzystywać system do zlecania zadań pracownikom oraz akceptacji tworzonych pism.
Referenci	W Urzędzie pracuje 500 referentów. Posiadają oni wyższe wykształcenie (najczęściej o profilu ekonomicznym). Biegle obsługują pakiet biurowy (arkusz kalkulacyjny, edytor tekstu). Referenci stanowią najliczniejszą grupę przyszłych użytkowników. Będą oni korzystać z systemu codziennie, wprowadzając pisma oraz przysyłając wiadomości.
Informatycy	W urzędzie pracuje 5 informatyków, zajmujących się obsługą systemów informatycznych. Wszyscy posiadają wyższe wykształcenie w zakresie informatyki (lub dziedziny pokrewnej). Grupa ta będzie zajmować się utrzymaniem infrastruktury sprzętowej oraz zarządzaniem konfiguracją systemu (w tym zarządzaniem kontami użytkowników).

IV.3.2 Obiekty biznesowe

Każda dziedzina posiada charakterystyczną dla siebie terminologię. W skład tej terminologii wchodzi między innymi rzeczowniki określające byty. Byty istotne z punktu widzenia funkcjonowania organizacji nazywane są obiektami biznesowymi (*ang. business objects*).

Obiekty biznesowe bardzo często tworzą strukturę hierarchiczną. Na przykład, według instrukcji kancelaryjnej „Sprawę” tworzy niepusty zbiór pism.

Obiekty biznesowe posiadają także własności (cechy). Na przykład, *pismo w sprawie* posiada *znak pisma*, *nadawcę*, *adresata* oraz *treść*.

Należy zwrócić uwagę, że rozróżnienie między własnością obiektu a kolejnym poziomem obiektów jest związane z poziomem zagłębienia opisu hierarchii. Na przykład, znak pisma mógłby zostać przedstawiony nie jako własność, tylko jako niezależny obiekt powiązany z dokumentem. Oczywiście, opisując każdy przedmiot wchodzący w skład takiej hierarchii można go „rozbijać” na coraz mniejsze elementy. Trzeba pamiętać, aby pozostawić opis na takim poziomie szczegółowości, na którym poszczególne elementy mają sens z punktu widzenia opisywanej dziedziny. Nadmiernie szczegółowy opis utrudnia czytelnikowi zrozumienie struktury i zależności między obiektami.

Warto także pamiętać, że niektóre pojęcia mogą posiadać synonimy. Na przykład, w przypadku instrukcji kancelaryjnej i teczek spraw jest mowa zamiennie

o woluminach (wolumenach) oraz tomach. Jeśli istnieje wiele określeń dla tego samego bytu, najlepiej jest stosować w specyfikacji konsekwentnie tylko jedno z nich. Dodatkowo warto zaznaczyć w opisie obiektu, że istnieją inne, tożsame określenia, zwłaszcza jeśli czytelnik może je napotkać w innych dokumentach.

Warto zwrócić uwagę, że w proponowanym szablonie specyfikacji (rozdział III) informacje dotyczące modelu danych mogą znajdować się częściowo w rozdziale IV.3.2, a częściowo w słowniku danych stanowiącym dodatek B. Różnica pomiędzy tymi znaczeniem tych rozdziałów jest następująca:

- rozdział “Obiekty biznesowe” prezentuje obiekty, które występują w domenie danego problemu, czyli są częścią świata rzeczywistego. Nie wszystkie obiekty biznesowe muszą znaleźć swoje odzwierciedlenie w modelu danych tworzonego systemu informatycznego. Nawet te obiekty, które zostaną uwzględnione w systemie, mogą zostać odzwierciedlone częściowo (na przykład tylko z niektórymi własnościami),
- “Dodatek B: Słownik danych” zawiera model danych tworzonego systemu informatycznego. W skład słownika danych wchodzi obiekty biznesowe w takiej postaci, w jakiej będą funkcjonowały w systemie, oraz struktury danych będące wynikiem formułowania wymagań. Na przykład, w domenie biznesowej dotyczącej archiwizacji nie występują obiekty typu “Użytkownik”, czy “Prawo dostępu”. Są one wynikiem analizy wymagań odnośnie mechanizmu kontroli dostępu w obrębie tworzonego systemu.

Obiekty
biznesowe
a słownik
danych

Istnieje kilka technik, które mogą posłużyć do przedstawienia struktury obiektów biznesowych. Można do tego celu wykorzystać notacje graficzne, jak na przykład diagramy klas notacji UML [Amb04, Śmi05] lub zapis tekstowy. Przykłady obu podejść zostały przedstawione w dodatku C.

Prezentacja
obiektów
bizneso-
wych

Dodatkowo, jeśli obiekty biznesowe reprezentują dokumenty, wówczas można zamieścić ich wzorce lub dołączyć ich przykładowe egzemplarze.

IV.3.3 Procesy biznesowe

Opis procesu biznesowego polega na określeniu czynności, które trzeba wykonać, aby osiągnąć zamierzony cel biznesowy. Zapis powinien dodatkowo uwzględniać kolejność wykonywania tych czynności oraz aktorów je wykonujących. Szczególnie istotne jest przedstawienie kanałów komunikacyjnych pomiędzy uczestnikami procesu.

Procesy biznesowe można przedstawiać za pomocą notacji graficznych lub w postaci opisu tekstowego. Przykładem popularnej notacji graficznej jest BPMN [Pio07]. Natomiast przykładem notacji tekstowej są przypadki użycia, przedstawione w dodatku A, których można także użyć do prezentowania procesów biznesowych.

IV.3.4 Reguły biznesowe

Reguły biznesowe to stwierdzenia definiujące i/lub ograniczające pewne aspekty biznesu (działalności), którego dotyczy opisywane oprogramowanie. Reguły biznesowe można podzielić na pięć podstawowych typów:

- fakty – proste stwierdzenia, które są zawsze prawdziwe. Fakty często określają powiązania oraz relacje między obiektami biznesowymi. Należy pamię-

tać, że fakty nie zawsze jednoznacznie przekładają się na wymagania, mogą jednak pomóc je zrozumieć. Przykłady:

- Każde zamówienie musi mieć określone koszty dostawy.
- Podatek nigdy nie jest doliczany do kosztów dostawy.
- Każdy pojemnik musi mieć nadany unikalny identyfikator.
- Usuwaniu teczki hybrydowej musi towarzyszyć niszczenie części papierowej tej teczki.
- ograniczenia – ograniczają akcje, które mogą być wykonane przez system i/lub przez użytkownika. Przykłady:
 - Rabaty nie mogą się sumować powyżej 20 procent.
 - Przerwa między zmianami pracownika musi wynosić co najmniej 12 godzin.
 - Student może dostać stypendium naukowe tylko wówczas, gdy jego średnia ocen jest wyższa niż 4,5.
 - Hasło dostępu pracownika musi być zmieniane co 3 miesiące.
 - Teczki mogą być tworzone jedynie na najniższym poziomie klas w schemacie klasyfikacyjnym.
- wyzwalacze – reguły, które wymuszają wykonanie określonej akcji, jeżeli są spełnione określone warunki. Warunki mogą być spowodowane bezpośrednimi operacjami użytkownika, mogą również wynikać pośrednio ze stanu wewnętrznego oprogramowania. Przykłady:
 - Jeśli minęło 30 dni od ostatniej dostawy od danego dostawcy, usuń go z listy ulubionych dostawców.
 - Jeśli została przekroczona data ważności leku, wyślij powiadomienie do administratora.
 - Jeśli użytkownik kupił więcej niż 3 produkty oraz jest zarejestrowany dłużej niż 2 lata, zaproponuj “platynowe członkostwo”.
 - Zmiana klauzuli tajności obiektu powoduje zmianę tajności wszystkich obiektów podrzędnych.
- wnioski – reguły, które generują nową wiedzę (fakty), gdy są spełnione określone warunki. Przykłady:
 - Jeśli sprzedano więcej niż 5000 sztuk towaru, to jest on bestsellerem.
 - Żywność przechowywana dłużej niż 5 lat uznawana jest za przeterminowaną.
- obliczenia - określają obliczenia, które mają być wykonane według ściśle określonego algorytmu. Przykłady:
 - Przy zamówieniach powyżej 200 zł zmniejsz cenę o 10 procent, a przy zamówieniach powyżej 500 zł zmniejsz cenę o 15 procent oraz nie licz kosztów przesyłki.
 - Do znalezienia najkrótszej ścieżki między miastami ma zostać wykorzystany algorytm Dijkstry.
 - Do kosztów podróży należy policzyć cenę wycieczki, cenę lotu, cenę ubezpieczenia oraz cenę opieki na miejscu wypoczynku.

6: Istotność reguł biznesowych typu obliczenia

Często obliczenia są kluczowe dla poprawnego funkcjonowania organizacji (np. wyliczanie podatku w aplikacji księgowej, obliczanie łącznej ceny dostawy w sklepie internetowym, etc.), dlatego też należy dołożyć wszelkich starań, aby zostały one należycie opisane.

7: Zapisywanie reguł biznesowych typu obliczenia

Przy prostych warunkach obliczeń wystarczy opis słowny, natomiast dla obliczeń bardziej złożonych bardziej czytelna może okazać się tabela lub schemat blokowy. Należy pamiętać również o umieszczeniu odnośnika do materiałów, które mogą pomóc zrozumieć, dlaczego oraz w jaki sposób dane obliczenie powinno zostać wykonane.

Przy zapisie reguł biznesowych należy pamiętać o oznaczeniu każdej reguły unikatowym identyfikatorem, który pozwoli na odwoływanie się do tej reguły z innych miejsc w dokumencie specyfikacji wymagań. Warto jest również zapisać typ reguły oraz jej źródło (np. czy wynika z charakterystyki prowadzonej działalności, z aktów prawnych, etc.), a także, czy dana reguła może się zmieniać dynamicznie w czasie, czy też pozostaje niezmienna. Najlepszą formą przechowywania reguł biznesowych jest tabela z odpowiednimi kolumnami.

Przykład:

ID	Definicja reguły	Typ	Zmienność	Źródło
Reg1-abc	Jeśli minęło 30 dni od ostatniej dostawy od danego dostawcy, usuń go z listy ulubionych dostawców	wyzwalacz	dynamiczna	zasady działalności
Reg2-xyz	Wyroby tytoniowe można sprzedawać tylko osobom pełnoletnim	ograniczenie	statyczna	prawo polskie

8: Podział na atomowe reguły biznesowe

W przypadku złożonej reguły biznesowej warto rozważyć możliwość rozbicia jej na kilka mniejszych (atomowych) reguł. Po rozbiciu wynikowe reguły atomowe można połączyć w jedną większą regułę. Przykład: *Jeśli zachodzi [Reg1] oraz [Reg2], to oblicz cenę zgodnie z [Reg5].*

IV.4 WYMAGANIA FUNKCJONALNE

Rozdział ten, stanowiący zwykle najobszerniejszą część specyfikacji, zawiera spis wymagań funkcjonalnych dla tworzonego produktu informatycznego.

Istnieje wiele sposobów ich organizowania. W niniejszej instrukcji zostanie zaprezentowane podejście opierające się na cechach oprogramowania (*ang. features*) lub modułach funkcjonalnych.

Cecha oprogramowania to zespół logicznie powiązanych ze sobą wymagań funkcjonalnych, które umożliwiają użytkownikowi osiągnąć cele biznesowe [Wie03]. Na przykład, można powiedzieć, że tworzony edytor tekstu będzie posiadał cechę w postaci wbudowanego sprawdzania pisowni (*ang. spell checker*). Z taką cechą związanych jest wiele szczegółowych wymagań funkcjonalnych.

Podobną definicja dotyczy modułu funkcjonalnego, a różnice dotyczą sposobu organizacji wymagań wewnątrz modułu i cechy. Ponieważ nie ma to większego znaczenia w przypadku tworzenia specyfikacji wymagań, dlatego w dalszej części pojęcia te będą traktowane równoważnie.

IV.4.1 Nazwa modułu funkcjonalnego

Każdy moduł funkcjonalny najlepiej przedstawić w osobnym podrozdziale, używając nazwy modułu jako tytułu np. “Administracja użytkownikami”, “Obsługa spraw” itd.

Opis i priorytet

Na początku podrozdziału poświęconego danemu modułowi warto umieścić krótki przegląd funkcji wchodzących w jego skład. Pomaga to w szybkim przeglądaniu specyfikacji i wyszukiwaniu w niej poszczególnych wymagań.

Każdemu modułowi należy przypisać priorytet odzwierciedlający jego istotność dla użytkownika. Więcej informacji na ten temat zostało zamieszczonych w rozdziale II.5.

Przypadki użycia

W tej sekcji opisane są przypadki użycia związane z danym modułem. Przypadki użycia są sposobem przedstawiania wymagań z punktu widzenia użytkownika, a nie systemu. Pozwalają one na wierniejsze odtworzenie rzeczywistych oczekiwań przyszłych użytkowników wobec systemu.

Szczegółowe informacje na temat tej techniki zostały przedstawione w dodatku A.

Specyficzne wymagania funkcjonalne

Największą zaletą przypadków użycia jest ich orientacja na cele i przedstawianie systemu z punktu widzenia użytkownika. Sprawia to, że sam model przypadków użycia zawiera znaczną część wymagań funkcjonalnych. Jednak w niektórych przypadkach wymagania zawarte w takim modelu nie są dostatecznie szczegółowe, aby mogły zostać zaimplementowane. Ponadto część wymagań może nie być dostrzegalna z punktu widzenia użytkownika (dotyczyć na przykład przetwarzania danych wewnątrz systemu). Dlatego w wielu przypadkach konieczne jest przedstawienie szczegółowych wymagań funkcjonalnych.

Każde wymaganie powinno posiadać unikatowy identyfikator. Wymagania są zapisywane w formie tekstowej, najczęściej w postaci stwierdzeń "System powinien / musi ...". W związku z tym szczególną uwagę należy zwrócić na użyty język i słownictwo, zgodnie ze wskazówkami zawartymi w rozdziale II.6.

Szczegółowe wymagania funkcjonalne wewnątrz modułu można zidentyfikować na dwa sposoby. Pierwszy polega na przedstawieniu tylko tych wymagań, które nie zostały ujęte w sekcji poświęconej przypadkom użycia. Drugie podejście jest bardziej wyczerpujące i polega na przedstawieniu wszystkich szczegółowych wymagań funkcjonalnych, zarówno tych, które wypływają z modelu przypadków użycia, jak i dodatkowych.

W większości przypadków pierwsze podejście powinno być wystarczające, jednak należy wówczas poinformować czytelnika, że przedstawione w tym rozdziale wymagania stanowią dopełnienie wymagań zapisanych w postaci przypadków użycia.

IV.5 CHARAKTERYSTYKA INTERFEJSÓW

Rozdział ten służy specyfikacji wymagań odnośnie interfejsów, z jakimi tworzone oprogramowanie ma się komunikować. Przez interfejs należy rozumieć tutaj m.in. sprzęt, inne oprogramowanie, biblioteki programistyczne. Rozdział ten określa również wymagania dotyczące interfejsów jakie tworzone oprogramowanie ma udostępniać. Ma to szczególne znaczenie w przypadku, gdy oprogramowanie ma być wykorzystywane przez inny system.

Interfejsy wstępnie opisane są w rozdziale IV.2.1, gdzie przedstawiono ogólną perspektywę i otoczenie tworzonego systemu, czyli m.in. interfejsy programistyczne i sprzętowe, z którymi oprogramowanie ma współpracować.

W przypadku tworzenia oprogramowania składającego się z wielu komponentów komunikujących się za pomocą interfejsów, dobrze jest stworzyć osobny dokument specyfikujący te interfejsy, lub też wykorzystać do tego celu specyfikację architektury systemu (z odpowiednim odniesieniem do tego dokumentu w tym rozdziale). Podobnie należy postąpić z innymi dokumentami opisującymi sprzęt, oprogramowanie, API, itp.

IV.5.1 Interfejsy użytkownika

Rozdział ten służy opisaniu wymagań dotyczących interfejsu użytkownika w tworzonym oprogramowaniu.

Z uwagi na panującą unifikację wyglądu interfejsów użytkownika, przy budowie nowego systemu zwykle ma miejsce jedna z trzech sytuacji:

- Istnieje gotowy i kompletny projekt interfejsu użytkownika – w takim przypadku wystarczy odwołać się do dokumentu z takim projektem.
- Istnieje ogólna koncepcja wyglądu interfejsu użytkownika – należy ją opisać, dołączając szkice ekranów aplikacji (wykonane np. z pomocą programu graficznego). Warto jednak zaznaczyć, że szkice te stanowią jedynie sugestie dla wykonawcy systemu.
- Brak jest specyficznych wymagań wobec interfejsu użytkownika – wówczas warto oprzeć się na ogólnych zasadach i wymaganiach opisanych w stan-

dardzie ISO 9126 dotyczącym użyteczności oraz jakości oprogramowania w użyciu.

Poniżej przedstawiono podstawowe charakterystyki określające jakość interfejsu użytkownika. Można je potraktować jako wytyczne do sformułowania wymagań wobec tego interfejsu.

Zrozumiałość

Definicja:

Określa, do jakiego stopnia użytkownik jest w stanie zrozumieć funkcjonalność zawartą w oprogramowaniu.

Przykład:

- Po 1-dniowym szkoleniu przeprowadzonym przez dostawcę oprogramowania 90% przeszkolonych użytkowników końcowych będzie rozumiało w 95%, w jaki sposób działają poszczególne funkcje dostępne w oprogramowaniu (w celu weryfikacji tego wymagania zostanie przeprowadzona ankieta wśród uczestników szkolenia <podać szczegóły ankiety oraz ocenienia>).
- System musi zapewniać polskojęzyczny interfejs użytkownika oraz umożliwiać wprowadzanie danych w języku polskim.

Łatwość nauczania się

Definicja:

Określa, w jaki sposób, jak szybko i przy wykorzystaniu jakich zasobów użytkownik jest w stanie nauczyć się korzystać z oprogramowania.

Przykład:

- Użytkownik końcowy będzie w stanie poprawnie korzystać z 90% dostępnych funkcji systemu po 3 godzinach pracy z oprogramowaniem bez dodatkowego szkolenia.
- Użytkownik końcowy będzie w stanie korzystać z 80% dostępnych funkcji po 2-dniowym szkoleniu przeprowadzonym przez dostawcę oprogramowania (w tym celu zostanie przeprowadzony test wśród uczestników szkolenia <podać szczegóły testu oraz ocenienia>).

Łatwość operowania

Definicja:

Określa, w jaki sposób oprogramowanie ułatwia użytkownikowi wykorzystywanie funkcjonalności zawartej w oprogramowaniu.

Przykład:

- Użytkownik końcowy będzie w stanie wywołać każdą dostępną funkcję w nie więcej niż 4 kliknięciach myszką.
- Użytkownik końcowy będzie w stanie wywołać każdą dostępną funkcję za pomocą akcji dostępnych w głównym menu systemu.
- Użytkownik końcowy będzie w stanie wywołać 80% dostępnych funkcji za pomocą skrótów klawiaturowych.
- Dostępny będzie mechanizm głosowego wywoływania 50% dostępnych funkcji.

*Atrakcyjność***Definicja:**

Określa, w jakim stopniu interfejs użytkownika jest atrakcyjny dla użytkownika.

Przykład:

- 80% ankietowanych użytkowników końcowych po godzinie korzystania z systemu oceni system jako atrakcyjny (zostanie przeprowadzona ankieta wśród uczestników szkolenia <podać szczegóły ankiety oraz ocenienia>).

*Efektywność***Definicja:**

Określa, w jakim stopniu oprogramowanie umożliwia użytkownikowi osiągnięcie zamierzonych celów, do których wykorzystuje on oprogramowanie, przy utrzymaniu określonej precyzji i kompletności.

Przykład:

- Procent błędnie wprowadzanych metadanych do obiektów archiwum elektronicznego nie może być większy niż 5%. Weryfikacja zostanie przeprowadzona na 10 losowo wybranych użytkownikach systemu (po przejściu szkolenia) i 30 losowo wybranych obiektach archiwum elektronicznego (klasach, teczkach, sprawach, dokumentach archiwalnych) o liczbie pól metadanych od 5 do 20. Łączna liczba popełnionych błędów nie może być większa niż 5% sumy liczb metadanych wylosowanych obiektów.

*Produktywność***Definicja:**

Określa, w jaki sposób tworzone oprogramowanie wpłynie na wydajność pracy osób z niego korzystających. Jest ona wypadkową wielu czynników, ale w dużej mierze zależy od interfejsu użytkownika.

Przykład:

- Średni czas wprowadzenia pojedynczego dokumentu do archiwum elektronicznego przez użytkownika będzie nie większy niż 30 sekund x liczba pól metadanych dokumentu archiwalnego i będzie nie dłuższy niż 1 minuta x liczba pól metadanych dokumentu archiwalnego. Weryfikacja zostanie przeprowadzona na 10 losowo wybranych użytkownikach systemu (po przejściu szkolenia) i 20 losowo wybranych dokumentach o liczbie pól metadanych od 5 do 20.

*Bezpieczeństwo użycia***Definicja:**

Określa, w jaki sposób oprogramowanie zapewnia osiągnięcie akceptowalnego poziomu ryzyka spowodowania szkód osobowych lub rzeczowych. Najczęściej występuje w produktach informatycznych, które obejmują także elementy sprzętowe.

Przykład:

- Urządzenia skanujące linie papilarne będą posiadały mechanizm samoczynnego wyłączania jeśli ich temperatura przekroczy 40 stopni Celsjusza.

- Urządzenia przeznaczone do skanowania dokumentów w trakcie pracy nie mogą emitować dźwięków o natężeniu większym niż 50 dB.

Satysfakcja

Definicja:

Określa, w jakim stopniu oprogramowanie satysfakcjonuje użytkowników wykonującego konkretne zadanie.

Przykład:

- 70% użytkowników wprowadzających dokumenty archiwalne do archiwum elektronicznego (po wprowadzeniu przynajmniej 50 dokumentów archiwalnych) odpowie *tak* albo *zdecydowanie tak* (skala: *zdecydowanie tak, tak, nie, zdecydowanie nie*) na stwierdzenie: *Jestem zadowolony(a) ze sposobu działania funkcji wprowadzania dokumentów archiwalnych do Systemu.*

Ponieważ wymagania dotyczące interfejsu użytkownika są trudno weryfikowalne, należy zwrócić szczególną uwagę na takie ich sformułowanie, aby stwierdzenie spełnienia tych wymagań było w ogóle możliwe. Jednym z sposobów weryfikacji jakości oprogramowania w kontekście powyższych charakterystyk są badania statystyczne na użytkownikach końcowych tworzonego oprogramowania. Na przykład, badanie atrakcyjności może polegać na losowym wyborze 10 użytkowników końcowych, których zadaniem jest praca z oprogramowaniem przez godzinę. Po upływie tego czasu wypełniają oni ankiety z pytaniami dotyczącymi atrakcyjności.

Oczywiście tego typu podejście wiąże się z szeregiem ryzyk związanych z zaangażowaniem użytkowników przyszłego systemu w proces weryfikacji wymagań (np. negatywne nastawienie do tworzonego systemu). Należy jednak zdawać sobie sprawę, że jeśli kryteria weryfikacji wymagań nie zostaną określone w żaden sposób, zamawiający nie będzie miał argumentów w trakcie dyskusji na temat poprawności ich realizacji.

IV.5.2 Interfejsy zewnętrzne

Opis każdego interfejsu powinien zawierać unikatowy identyfikator interfejsu, jego nazwę, cel zastosowania, komponenty, które będą brały udział w interakcji, standard i/lub dokumentację opisującą interfejs oraz technologię, która musi zostać wykorzystana do komunikacji z interfejsem.

Specyfikując interfejsy, z którymi tworzone oprogramowanie ma współpracować, należy zadbać o jak najwyższy stopień precyzji. Jeżeli nie wszystkie informacje na ten temat są dostępne, wówczas należy opisać intencjonalny sposób korzystania z interfejsu oraz jak (subiektywnie) komunikacja z takim interfejsem powinna wyglądać, a także stworzyć odpowiedni zapis w rozdziale poświęconym sprawom otwartym.

Czasami do wykonania określonej operacji tworzone oprogramowanie może wykorzystywać kilka interfejsów, np. wysyłanie danych może być realizowane poprzez wiadomości e-mail oraz poprzez dowolny inny protokół komunikacyjny. W takiej sytuacji najlepiej pominąć fakt ich wspólnego użycia przez jedną z funkcji i każdy z dopuszczalnych protokołów opisać oddzielnie, gdyż każdy będzie wymagał innej implementacji oraz osobnego testowania.

9: Precyzja przy specyfikacji interfejsów

Istotną kwestią przy specyfikacji charakterystyki interfejsów jest zachowanie najwyższego stopnia precyzji. Brak informacji lub niejasność może spowodować, że dostawca oprogramowania dostosuje swój produkt do innego interfejsu, niż jest to wymagane.

W przypadku, gdy tworzone oprogramowanie ma udostępniać interfejs, z którego będą mogły korzystać inne systemy, wówczas w jego opisie warto przedstawić możliwe scenariusze interakcji (włączając w to sytuacje informujące o napotkanych błędach), rodzaje wymienianych danych, typy komunikatów oraz wymagane technologie. Należy się również zastanowić, czy zostały określone związane z interfejsem wymagania dotyczące skalowalności, rozszerzalności, przepustowości, dostępności, bezpieczeństwa i dokumentacji tworzonego interfejsu.

Interfejsy sprzętowe

Jest to miejsce opisu charakterystyki każdego interfejsu na styku sprzęt-oprogramowanie. Opis ten powinien zawierać typy wspieranych urządzeń, typy wymienianych danych, sposób kontroli oraz komunikacji.

Interfejsy programistyczne

W rozdziale tym znajduje się opis komunikacji pomiędzy tworzonym oprogramowaniem a innym systemem komputerowym (identyfikowanym przez nazwę oraz wersję). Należy uwzględnić wymagania dotyczące wykorzystywanych baz danych, systemów operacyjnych, bibliotek programistycznych, przeglądarek, internetowych, komercyjnych komponentów oraz narzędzi i programów, z którym tworzone oprogramowanie ma współpracować. Należy wskazać sposób współpracy, dane (wraz z ich formatem), które będą wymieniane między systemami, rodzaj oraz format komunikatów.

Interfejsy komunikacyjne

Należy określić wymagania dotyczące funkcji komunikacyjnych wykorzystywanych przez tworzone oprogramowanie, np. poczta e-mail, przeglądarka internetowa, protokoły sieciowe, etc.

IV.6 WYMAGANIA POZAFUNKCJONALNE

Oprócz wymagań funkcjonalnych ważnym elementem specyfikacji wymagań są wymagania pozafunkcjonalne, które definiują pewne zasady oraz charakterystyki, z których wartościami powinno być zgodne tworzone oprogramowanie. Podczas gdy weryfikacja wymagań funkcjonalnych jest dość prosta, to w przypadku wymagań pozafunkcjonalnych może sprawić ona trudność, jeśli wymagania te nie zostaną wyrażone w sposób jednoznaczny, mierzalny i realistyczny. Najlepszym rozwiązaniem jest określenie dla każdego wymagania pozafunkcjonalnego w jaki sposób i w jakim środowisku będzie ono weryfikowane.

Poniżej przedstawione zostały charakterystyki i podcharakterystyki dla wymagań pozafunkcyjnych zgodnie ze standardem ISO/IEC 9126-1:2001. Przedstawione charakterystyki zostały opatrzone definicjami, przykładami oraz wzorcami. Należy pamiętać, że zarówno wzorce nie obejmują wszystkich możliwych wymagań, jak i niektóre wymagania mogą odpowiadać wielu charakterystykom, dlatego też mogą one służyć jedynie jako pomoc przy pozyskiwaniu wymagań pozafunkcyjnych, natomiast ewentualna struktura tego rozdziału powinna odpowiadać obszarom tematycznym wymagań.

10: Koszty wymagań pozafunkcyjnych

W przypadku niektórych wymagań pozafunkcyjnych może wydawać się oczywiste podawanie wartości maksymalnych lub zbliżonych do maksymalnych, np. *system ma być dostępny 24 godziny na dobę przez wszystkie dni w roku*, *system ma obsługiwać milion użytkowników w ciągu jednej sekundy*. Należy jednak zauważyć, że niektóre sformułowane w ten sposób wymagania są niemożliwe do zrealizowania, a dodatkowo każde wymaganie pozafunkcyjne powoduje wzrost kosztów wytwarzanego oprogramowania, np. żeby obsłużyć ogromną liczbę użytkowników dostawca oprogramowania będzie musiał prawdopodobnie sięgnąć po kosztowne rozwiązania programistyczne i sprzętowe, co z pewnością odbije się na kosztach końcowych tworzonego systemu.

IV.6.1 Funkcjonalność

Zawiera podcharakterystyki opisujące zdolność oprogramowania do dostarczenia określonej funkcjonalności.

Odpowiedniość

Definicja:

Określa w jakim stopniu oprogramowanie jest zgodne z wyspecyfikowanymi wymaganiami.

Przykład: System musi poprawnie obsługiwać wszystkie czynności związane z procesem weryfikacji bezpiecznego podpisu elektronicznego pomimo odseparowania od sieci Internet.

Dokładność

Definicja:

Określa, w jakim stopniu oprogramowanie w rezultacie wykonywania swoich funkcji dostarcza poprawne i precyzyjne wyniki.

Przykład:

- Cena produktów ma być przedstawiana z dokładnością do drugiego miejsca po przecinku.
- Eksportowane dane obiektów archiwum mają być zapisane w formacie XML zgodnie ze schematem określonym w rozporządzeniu (...).

- System musi umożliwiać prezentację daty w formacie dd-mm-yyyy i czasu w formacie hh-mm-ss.
- System musi umożliwiać określenie okresu przechowywania dokumentów z przedziału od 1 miesiąca do 100 lat z dokładnością do liczby dni.
- Wszystkie identyfikatory obiektów muszą być unikatowe w obrębie całego systemu.

Wzorce:

- Wartości *<wskazać dane>* mają być prezentowane w arytmetyce przedziałowej.
- Jeśli inne wymagania dot. dokładności nie stanowią inaczej, to wartości wyjściowe mają być prezentowane z błędem nie większym niż *<liczba>%*.
- Wartości *<wskazać wyniki>* mają być prezentowane z błędem nie większym niż *<liczba>%*.
- System powinien działać poprawnie dla danych wejściowych *<wskazać dane>* przyjmujących wartości z przedziału od *<liczba>* do *<liczba>* i prezentowanych z dokładnością do *<liczba>* cyfr po przecinku.
- Wartości mają być prezentowane z dokładnością do *<liczba>* miejsc do przecinka.
- *<Dane>* mają być zgodne z *<schemat, standard>*.
- System powinien obsługiwać dane zgodnie z formatem określonym w *<tytuł dokumentu>*.

*Inter-operacyjność***Definicja:**

Określa, w jakim stopniu oprogramowanie prawidłowo współpracuje z innymi systemami (zarówno na poziomie bezpośredniego wywoływania funkcji, jak i wymiany danych).

Przykład:

- Budowany system ma współpracować z przeglądarką internetową Mozilla Firefox w wersji 3.0 lub wyższej.
- Budowany system ma odbierać i zapisywać dane pobrane z systemu XYZ w wersji 1.7.
- System musi umożliwić przyjmowanie, przechowywanie i prezentację dokumentów w następujących formatach: Rich Text Format (RTF) w wersji 1.6, Graphics Interchange Format (GIF) w wersji 98a, Portable Document Format, (PDF) w wersji 1.4.
- Musi istnieć możliwość wprowadzania dodatkowych pozycji do listy formatów plików obsługiwanych przez system.

Wzorce:

- Budowany system ma współdziałać z następującymi systemami: *<wskazać systemy>*.
- Budowany system ma wczytywać pliki w następujących formatach: *<wskazać format pliku>* w wersji *<wskazać wersję formatu>*.

*Bezpieczeństwo***Definicja:**

Określa, w jaki sposób oprogramowanie zapewnia ochronę informacji oraz danych.

Przykład:

- Dostęp do systemu powinien być chroniony hasłem ustawianym indywidualnie przez każdego użytkownika.
- Użytkownik powinien być automatycznie wylogowany z systemu po czasie nieaktywności większym niż 5 minut.
- Użytkownik bez uprawnień administratora powinien mieć dostęp do systemu jedynie z poziomu przeglądarki internetowej.
- Słownik typów dokumentów elementarnych może być zmieniany jedynie przez użytkownika systemu posiadającego odpowiednie uprawnienia.
- Uprawnienia w systemie może zmieniać jedynie Administrator.
- Jeśli system pozwala użytkownikom na podejmowanie niedozwolonych prób dostępu do dokumentów, musi to zapisać w dzienniku kontroli.
- System musi udostępniać użytkownikowi funkcje zgodnie z przyznanym użytkownikowi profilem uprawnień, gwarantując ścisłą kontrolę uprawnień.
- System musi uniemożliwiać wprowadzenie jakichkolwiek zmian w treści dokumentu archiwalnego przez użytkowników oraz przez Administratora.

Wzorce:

- System powinien być odporny na działanie wirusów.
- System powinien być odporny na następujące próby nielegalnego dostępu: *<wskazać typy nielegalnego dostępu>*.
- Dane dotyczące *<wskazać dane>* mają być jawne /dostęp do nich mają wszyscy użytkownicy systemu.
- Stanowisko robocze powinno być zabezpieczone w następujący sposób *<opis>*.
- Dane dotyczące *<wskazać dane>* powinny być szyfrowane *<sposób szyfrowania>*.
- W systemie powinien zostać utworzony następujący system ról i uprawnień *<opis>*.
- Dane *<sprecyzować>* powinny być chronione w następujący sposób: *<opis>*.

*Zgodność funkcjonalności***Definicja:**

Określa, z jakimi standardami, konwencjami oraz uregulowaniami prawnymi (powiązanymi z funkcjonalnością oprogramowania) oprogramowanie musi być zgodne.

Przykład:

- System powinien być zgodny z ustawą o ochronie danych osobowych z dnia 29.08.1997.

Wzorce:

- System powinien być zgodny z *<wskazać dokument>*.

IV.6.2 Niezawodność

Określa, w jakim stopniu oprogramowanie może utrzymywać określony poziom wydajności (w zdefiniowanych warunkach).

Dojrzałość

Definicja:

Określa, w jakim stopniu oprogramowanie jest w stanie unikać awarii.

Przykład:

- Moduł dodawania nowych dokumentów powinien działać prawidłowo w 98% przypadków.
- System musi być dostępny dla wszystkich użytkowników od godziny 6:00 do 22:00 (CET) we wszystkie dni tygodnia przez cały rok.
- System powinien ostrzegać użytkownika przed zapisem plików, które mogą wpłynąć negatywnie na integralność danych przechowywanych w systemie.

Wzorce:

- Funkcja *<wskazać funkcję>* powinna działać prawidłowo w *<podać liczbę>*% przypadków.
- System musi być dostępny dla *<wskazać użytkowników>* w okresie *<podać czas>*.

Tolerancja uszkodzeń

Definicja:

Określa, w jakim stopniu oprogramowanie jest w stanie utrzymać odpowiednią wydajność w przypadku awarii.

Przykład:

- W przypadku wystąpienia zbyt dużego obciążenia serwera zapytaniami użytkowników, system powinien wyświetlić informację o tymczasowym braku dostępu do systemu, powinien również powiadomić administratora, wysyłając wiadomość e-mail na adres administrator@budowany.system.pl.
- System musi zachowywać w sposób ciągły wewnętrzną integralność, bez względu na: czynności serwisowe, inne czynności wykonywane przez użytkownika, awarię poszczególnych komponentów systemu.

Wzorce:

- W przypadku wystąpienia *<opis zdarzenia>* system powinien *<opis zachowania>*.
- Żaden wyjątek wewnętrzny systemu nie może być widoczny dla użytkownika.

Odtwarzalność

Definicja:

Określa, w jakim stopniu oprogramowanie w przypadku awarii jest w stanie powrócić do stanu sprzed awarii.

Przykład:

- W przypadku wystąpienia awarii bazy danych system ma odtworzyć dane sprzed 24 godzin.

- W przypadku awarii modułu wysyłającego wiadomości e-mail system powinien przywrócić zdolność działania tego modułu w czasie nie dłuższym niż 12 godzin.
- System musi umożliwiać odtworzenie stanu systemu z dowolnej chwili czasu za pomocą zawartości kopii zapasowej i dziennika kontroli, przy równoczesnym zachowaniu integralności danych.
- System musi zapewnić funkcjonalność przywracania systemu do stanu sprzed awarii oraz cofnięcia jego stanu w przypadku awarii lub błędu aktualizacji.

Wzorce:

- W przypadku wystąpienia *<opis zdarzenia>* system powinien przywrócić swoją zdolność działania w zakresie *<wskazać funkcjonalność>* w ciągu *<wskazać czas>*.
- W przypadku wystąpienia *<opis zdarzenia>* system powinien przywrócić swoją zdolność działania w pełnym zakresie w ciągu *<wskazać czas>*.
- W przypadku wystąpienia *<opis zdarzenia>* utracie mogą ulec tylko dane dotyczące *<wskazać dane>*.
- System powinien tworzyć kopie zapasowe następujących danych: *<wymienić dane>*.

IV.6.3 Wydajność

Zawiera podcharakterystyki opisujące zdolność oprogramowania do zapewnienia odpowiedniej wydajności w stosunku do ilości wykorzystywanych zasobów (w określonych warunkach).

*Charakterystyka czasowa***Definicja:**

Określa czasy odpowiedzi oraz czasy obliczeń dla poszczególnych funkcjonalności oprogramowania.

Przykład:

- Czas między zatwierdzeniem dodania nowego dokumentu o wielkości 5 MB, a wyświetleniem potwierdzenia systemu o poprawnym dodaniu nie powinien być dłuższy niż 5 minut.
- Czas oczekiwania Administratora na zalogowanie się do systemu w 95% przypadków nie powinien być dłuższy niż 2 minuty.
- Zamknięcie systemu powinno następować w czasie nie dłuższym niż 10 sekund.
- Bez względu na liczbę przechowywanych danych i przy jednoczesnej obsłudze żądań edycji danych przez 200 użytkowników, system powinien zapewnić następujące czasy wykonywania akcji: wyświetlenie okna logowania ≤ 10 sekund, logowanie użytkownika do systemu z wyświetleniem strony powitalnej dla użytkownika ≤ 10 sekund, dodanie nowego obiektu do systemu ≤ 10 sekund, eksport danych ≤ 5 godzin.

Wzorce:

- Czas między *<pobudzenie>* a *<odpowiedź systemu>* nie powinien być nigdy dłuższy niż *<ilość czasu>*.

- Czas między <pobudzenie> a <odpowiedź systemu> w <liczba>% przypadków nie powinien być dłuższy niż <ilość czasu>.
- Czas oczekiwania <wskazać aktora> na realizację <wskazać operację> w <liczba>% przypadków nie powinien być dłuższy niż <ilość czasu>.
- System powinien <opis akcji systemu> co <podać czas>.
- Czas <opis akcji systemu> nie jest istotny, powinien jednak być skończony.

Wykorzystanie zasobów

Definicja:

Określa, w jakim stopniu oprogramowanie wykorzystuje zasoby różnego typu podczas wykonywania swoich funkcji.

Przykład:

- System powinien działać poprawnie przy dostępności 512 MB pamięci operacyjnej.
- System powinien obsługiwać co najmniej 125 użytkowników jednocześnie modyfikujących dane systemu na następującym sprzęcie: 1 procesor RISC 64-bit Power5 1.65 GHZ, 2 GB RAM, 1 wirtualny interfejs sieciowy Ethernet 1Gb/s, zbiór dysków twardych o łącznej pojemności 2TB pracujących w mirroringu (osobne 2TB) w technologii 2Gb/s Fibre Channel.
- Dostęp do systemu powinien być możliwy na następującej minimalnej konfiguracji sprzętowej: procesor Intel Celeron 1,7 GHz, 256MB pamięci RAM, karta sieciowa Etherbet 10/100 Mb/s, wolna przestrzeń dyskowa 2GB.
- System musi zostać zainstalowany na macierzy dyskowej SAN IBM DS4300 z możliwością uruchomienia na dowolnym z dwóch serwerów firmy IBM pSeries p5 570 pracujących pod kontrolą systemu operacyjnego AIX v.5.3.
- System powinien korzystać z infrastruktury sieciowej o przepustowości 10Mb/s i funkcjonującej w oparciu o sieć POZMAN Poznańskiego Centrum Superkomputerowo-Sieciowego.

Wzorce:

- System powinien działać poprawnie przy założeniu, że dostępne są następujące ilości zasobów <typ zasobu> – nie więcej niż <liczba i jednostka miary>.
- System powinien działać na następującej minimalnej konfiguracji sprzętowej <wskazać sprzęt>.
- System powinien działać wykorzystując następującą infrastrukturę sieciową <opisać infrastrukturę sieciową>.

IV.6.4 Łatwość utrzymania

Zawiera podcharakterystyki opisujące zdolność oprogramowania do zapewnienia możliwości jego modyfikacji. Modyfikacje mogą dotyczyć poprawek, udoskonaleń, dostosowywania oprogramowania do innych środowisk i/lub wymagań.

Łatwość analizy

Definicja:

Określa, w jakim stopniu oprogramowanie umożliwia zdiagnozowanie usterek oraz przyczyn awarii oraz w jakim stopniu ułatwia identyfikację miejsca wymagającego poprawy.

Przykład:

- System powinien rejestrować dane dotyczące wszystkich użytkowników w dzienniku systemowym.
- System powinien rejestrować wszystkie operacje na bazie danych w dzienniku bazy danych.
- Powinna być dostępna funkcjonalność umożliwiająca uruchomienie systemu w trybie diagnostycznym, w którym wszystkie operacje wraz z parametrami wykonywane przez system będą zapisywane w dzienniku diagnostycznym.
- System musi zachowywać w dzienniku systemu informacje o wszystkich zmianach ustawień parametrów konfiguracyjnych systemu.
- System musi monitorować dostępne zasoby dyskowe i powiadamiać Administratorów o konieczności podjęcia działania wobec niewystarczających zasobów dyskowych.

Wzorce:

- System powinien rejestrować dane dotyczące *<wskazać dane i zdarzenia podlegające rejestracji w tzw. logu systemowym>*.
- System powinien zawierać testy *<opis testów>*
- Kody źródłowe powinny być udokumentowane w języku *<podać język>* zgodnie ze standardem *<wymienić standard>*.
- Kod źródłowy systemu powinien być utworzony zgodnie ze standardem *<wymienić standard>*.

*Łatwość zmiany***Definicja:**

Określa, do jakiego stopnia oprogramowanie jest łatwe do zmiany.

Przykład:

- System powinien umożliwiać dodanie wcześniej nieuwzględnionego rodzaju dokumentów w czasie nie większym niż 24 godzin.
- Dostosowanie systemu do nowych rozporządzeń prawnych nie powinno trwać dłużej niż 2 dni robocze od momentu uchwalenia rozporządzenia przez odpowiednie organy.

Wzorce:

- Dodanie *<opisać funkcjonalność>* nie powinno zająć więcej niż *<podać zakres czasu>*.

*Stabilność***Definicja:**

Określa, w jakim stopniu oprogramowanie jest odporne na nieprzewidziane efekty wprowadzonych zmian.

Przykład:

- Dodanie nowych atrybutów do istniejącej struktury danych nie powinno wpływać na działanie systemu.
- Import danych z systemów wewnętrznych nie powinien zmieniać przechowywanych danych w systemie.
- Instalacja aktualizacji nie powinna wpływać na przechowywane w systemie dane.

Wzorce:

- Wykonanie *<wymienić operacje>* nie powinno wpływać na *<wskazać część systemu>*.

*Łatwość testowania***Definicja:**

Określa, w jaki sposób oprogramowanie zapewnia możliwość jego przetestowania.

Przykład:

- System powinien posiadać testy umożliwiające automatyczną weryfikację wprowadzonych zmian pokrywającą 95% dostępnej funkcjonalności.
- System powinien posiadać automatyczne testy wydajnościowe symulujące jednoczesną pracę 1000 użytkowników wykonujących losowe operacje (z czego połowa byłaby operacjami modyfikacji danych).

Wzorce:

- System powinien mieć wbudowane testy dotyczące *<wskazać testy, które mają być realizowane na zasadzie built-in-tests>*.

IV.6.5 Przenośność

Zawiera podcharakterystyki opisujące zdolność oprogramowania do pracy po przeniesieniu z jednego środowiska do innego.

*Łatwość adaptacji***Definicja:**

Określa, w jakim stopniu oprogramowanie jest dostosowane do pracy w różnych środowiskach.

Przykład:

- System powinien być w stanie funkcjonować działać w następujących środowiskach: MS Windows Server 2003, Red Hat Linux Enterprise 5.3.
- System powinien być dostosowany do nowej wersji systemu operacyjnego MS Windows Server nie później niż 2 tygodnie po publicznej premierze tego systemu operacyjnego.

Wzorce:

- System powinien móc działać w następujących środowiskach: *<opis środowiska>*.

*Łatwość instalacji***Definicja:**

Określa w jaki sposób oprogramowanie powinno być instalowane w określonym środowisku.

Przykład:

- System powinien instalować się nie dłużej niż 30 minut na następującym sprzęcie: procesor Intel Celeron 1,7 GHz, 256MB pamięci RAM, karta sieciowa Ethernet 10/100 Mb/s, wolna przestrzeń dyskowa 2GB.
- Instalacja systemu nie powinna wymagać od użytkownika więcej niż 6 kliknięć.
- Instalację systemu powinien móc przeprowadzać zwykły użytkownik systemu operacyjnego (bez uprawnień administratora).
- System nie powinien wymagać instalacji żadnego dodatkowego oprogramowania (poza przeglądarką internetową).

Wzorce:

- System powinien się instalować automatycznie i nie powinno to trwać dłużej niż *<liczba i jednostka czasu>*.
- Instalację systemu powinna móc przeprowadzić osoba *<wskazać minimalne kwalifikacje>* i nie powinno jej to zająć dłużej niż *<liczba i jednostka czasu>*.
- System powinien instalować się w następujący sposób *<opis>*.

*Współistnienie***Definicja:**

Określa w jakim stopniu oprogramowanie może bezkolizyjnie działać w tym samym środowisku (wraz ze współdzieleniem zasobów) z innymi aplikacjami.

Przykład:

- System powinien bez problemu równolegle pracować z następującymi aplikacjami: Norton AntyVirus 2009, QK SMTP Server w dowolnej wersji.
- Użytkownicy systemu muszą mieć zagwarantowaną możliwość jednoczesnego i bezkolizyjnego korzystania zarówno z systemu, jak i następujących aplikacji: MS Office 2007, Lotus Notes 7.0, eTrust 7.1.

Wzorce:

- System powinien bez problemu równolegle pracować z następującymi aplikacjami: *<wskazać aplikacje>*.
- System powinien umożliwiać uruchomienie tylko jednej swojej instancji.

*Łatwość zamiany***Definicja:**

Określa możliwości zastąpienia oprogramowanie innym oprogramowaniem posiadającym tę samą lub podobną funkcjonalność.

Przykład:

- Procedura aktualizacji oprogramowania do najnowszej wersji nie powinna trwać dłużej niż 60 minut.
- System powinien sprawdzać przy każdym uruchomieniu, czy są dostępne aktualizacje oprogramowania.
- System powinien obsługiwać wszystkie pliki utworzone za pomocą wszystkich wcześniejszych wersji systemu.
- System powinien umożliwiać swoją aktualizację bez konieczności ponownego uruchomienia systemu.

Wzorce:

- Procedura wymiany oprogramowania powinna trwać nie dłużej niż *<liczba i jednostka czasu>*.
- Interfejs użytkownika nowej wersji systemu powinien być zgodny z poprzednią wersją systemu.
- System powinien umożliwiać wczytanie plików danych *<wyspecyfikować dane>* z poprzedniej/poprzednich wersji *<podać wersję>*.

⚠ # 3: Przykłady źle sformułowanych wymagań pozafunkcjonalnych

- Aplikacja ma zużywać mało pamięci operacyjnej – co oznacza pojęcie “mało”?
- Oprogramowanie ma wytrzymywać duże obciążenia użytkowników – czym jest “duże obciążenie”?
- System musi posiadać elastyczny i parametryzowalny mechanizm wczytywania i eksportowania danych – co należy rozumieć przez “elastyczny i parametryzowalny mechanizm”?
- Użytkownicy systemu muszą mieć możliwość jednoczesnego i bezkolizyjnego korzystania zarówno z systemu jak i z innych aplikacji – o jakie konkretnie “inne aplikacje” chodzi?
- System musi być w stanie zapisać dokumenty o różnych formatach danych i strukturze – jakie konkretnie “formaty i struktury danych” mają być uwzględnione?
- System powinien być obsługiwany za pomocą przeglądarki internetowej – jaka przeglądarka i w jakiej wersji powinna być wzięta pod uwagę?
- System musi być uruchomiony na dowolnym systemie z rodziny MS Windows – czy rzeczywiście chodzi o dowolny system wliczając w to systemy z MS Windows 98, MS Windows 95 i starsze?
- System musi obsługiwać 100 użytkowników na następującym sprzęcie: procesor Intel Pentium 4 2GHz, 1GB pamięci operacyjnej, 10GB wolnej przestrzeni dyskowej, system operacyjny MS Windows XP Professional – jakie konkretnie operacje będą wykonywali użytkownicy (inaczej obciążają system operacje czytania danych, inaczej operacje modyfikacji danych, itp.)?
- Średni czas wykonania operacji powinien być mniejszy niż 10 sekund – o jakie operacje chodzi? Należy zauważyć, że niektóre operacje muszą być wykonane szybko (np. odczytanie danych), inne mogą trwać dłużej (np. eksport danych), dlatego też czas wykonania operacji warto wyspecyfikować osobno dla każdej z operacji.
- System musi być prosty w instalacji – co oznacza pojęcie “prosty w instalacji”?

Jak łatwo zauważyć, wymagania pozafunkcjonalne wyrażone w ten sposób są niemożliwe do zweryfikowania w obiektywny sposób, a więc ich wyegzekwowanie ich spełnienia przez wykonawcę oprogramowania może okazać się trudne.

IV.7 INNE WYMAGANIA

Jeśli w trakcie prac analitycznych pojawią się wymagania, które nie pasują do żadnego z pozostałych rozdziałów, wówczas należy je umieścić w tym miejscu.

Przykładowe wymagania mogą dotyczyć wielojęzyczności oprogramowania (obsługiwane waluty, format daty, kwestie poprawności politycznej itp.), zgodności z aktami prawnymi etc.

IV.8 DODATEK A: SŁOWNIK POJĘĆ I TERMINÓW

Jest to miejsce na zdefiniowanie wszystkich specjalistycznych terminów oraz rozwinięcia używanych skrótów, które potrzebne są czytelnikom do pełnego i jednoznacznego zrozumienia dokumentu specyfikacji wymagań. Trzeba pamiętać, że niektóre terminy i wyrażenia, które są oczywiste dla twórców dokumentu, mogą być niezrozumiałe lub niejednoznaczne dla innych czytelników, dlatego należy więc zadbać, aby zostały one wyjaśnione w sposób niebudzący wątpliwości.

IV.9 DODATEK B: SŁOWNIK DANYCH

Tutaj należy umieścić dokładny słownik danych, widzianych zarówno z perspektywy obiektów informacyjnych narzuconych przez proces biznesowy, jak i dodatkowych obiektów zidentyfikowanych lub stworzonych w następstwie procesu zbierania wymagań.

IV.10 DODATEK C: MODELE ANALIZY

W tym rozdziale można umieścić dodatkowe modele analityczne wspomagające zapis wymagań, np. diagramy przepływu danych, diagramy klas, diagramy stanu lub diagramy związków encji (ERD). Więcej informacji na temat metod analizy z wykorzystaniem języka UML znajduje się w dodatku B.

IV.11 DODATEK D: LISTA SPRAW OTWARTYCH

Jest to miejsce dla dynamicznej listy ciągle otwartych wymagań, kwestii wymagających wyjaśnienia i/lub doprecyzowania. Jeśli brakuje informacji lub decyzji w sprawie jakiegoś wymagania, pojawiły się konflikty w wymaganiach, które należy rozwiązać, wówczas należy te kwestie udokumentować w tym rozdziale. Podczas prac nad dokumentem specyfikacji wymagań należy aktywnie monitorować listę spraw otwartych, aby wszystkie sprawy z tejże listy zostały jak najszybciej wyjaśnione.

Przypadki użycia

A.1 CZYM SĄ PRZYPADKI UŻYCIA

Przypadek użycia opisuje interakcje pomiędzy aktorami (ludźmi lub systemami), którzy dążą do osiągnięcia określonego celu.

Informacje związane z aktorami zostały przedstawione w rozdziale IV.3.1. Warto jednak przypomnieć, że aktor może być utożsamiany z rolą użytkownika. Dla tego jedna osoba w organizacji może pełnić różne role (wcielać się w różnych aktorów) w ramach różnych przypadków użycia.

Zapis w postaci przypadków użycia kładzie nacisk na przekazanie tego *co użytkownik pragnie osiągnąć*, natomiast wymagania typu “system powinien...” obrazują raczej *co system powinien umożliwiać*.

A.2 POZIOMY PRZYPADKÓW UŻYCIA

W rozdziale II.4 zostały omówione poziomy wymagań. Zgodnie z nim, istnieją trzy poziomy wymagań: biznesowy, użytkownika i podfunkcji.

Przypadki użycia mogą dostarczać informacji odnośnie wymagań na wszystkich trzech poziomach.

Poziomem centralnym z perspektywy opisu wymagań dla systemu jest *poziom użytkownika*. Przypadki użycia tego poziomu opisują, w jaki sposób można osiągnąć cele istotne z punktu widzenia użytkownika (cele, które przynoszą korzyść). Na przykład, jeśli modelowanym systemem będzie sklep internetowy, to istotnym celem z punktu widzenia aktora-klienta może być zakupienie produktu.

Bardzo często, aby osiągnąć cel istotny z perspektywy aktora, należy zrealizować kilka mniejszych zadań, które same z siebie nie są znaczące dla użytkownika. O przypadkach opisujących jak osiągnąć takie cele mówimy, że znajdują się na poziomie *podfunkcji*. Na przykład, w sklepie internetowym, aby klient mógł zakupić produkt, musi dokonać rejestracji w systemie, a następnie się do niego zalogować. Realizacja tych celów jest konieczna, ale nie przynosi wymiernych korzyści zauważalnych dla klienta.

Można także zadać pytanie, w jaki sposób dany cel użytkownika wpisuje się w funkcjonowanie organizacji (np. dlaczego jest istotny?). Odpowiedzi na te pytania mogą udzielić przypadki użycia *poziomu biznesowego*. Opisują one interakcje pomiędzy aktorami (w tym przypadku najczęściej ludźmi), przedstawiając w jaki sposób można osiągnąć istotny cel biznesowy. Nawiązując do wcześniejszego przykładu dotyczącego sklepu internetowego, złożenie zamówienia przez klienta stanowi pierwszy krok realizacji całego procesu sprzedaży postrzeganego jako cel biznesowy.

A.3 IDENTYFIKACJA PRZYPADKÓW UŻYCIA

Pierwszym krokiem prowadzącym do identyfikacji przypadków użycia jest sporządzenie diagramu kontekstu (patrz rozdział IV.2.1)). Pozwala on zidentyfikować głównych aktorów oraz ich główne cele komunikacji z tworzoną systemem.

Kolejnym źródłem przypadków użycia poziomu użytkownika mogą być procesy biznesowe. Należy przyrzeć się poszczególnym czynnościom wykonywanym przez aktorów w ramach procesów biznesowych, które mają zostać z informatyzowane. Część z takich czynności można przekształcić w przypadki użycia poziomu użytkownika.

Identyfikacji przypadków użycia można również dokonać prosząc potencjalnych użytkowników o wyszczególnienie *powierzonych im zadań, które ich zdaniem mogłyby być realizowane z wykorzystaniem tworzonego systemu*. Warto zwrócić uwagę, że pytanie powinno dotyczyć tego, *co użytkownik chciałby osiągnąć przy pomocy systemu, a nie funkcji, które system powinien mu udostępniać*.

Podczas identyfikacji wymagań należy kierować się zasadą „najpierw wszcz, potem wgłb”. Podejście „wszcz” oznacza, że w pierwszym rzędzie należy zidentyfikować wszystkie wymagania na danym poziomie szczegółowości zanim będą one uszczegóławiane.

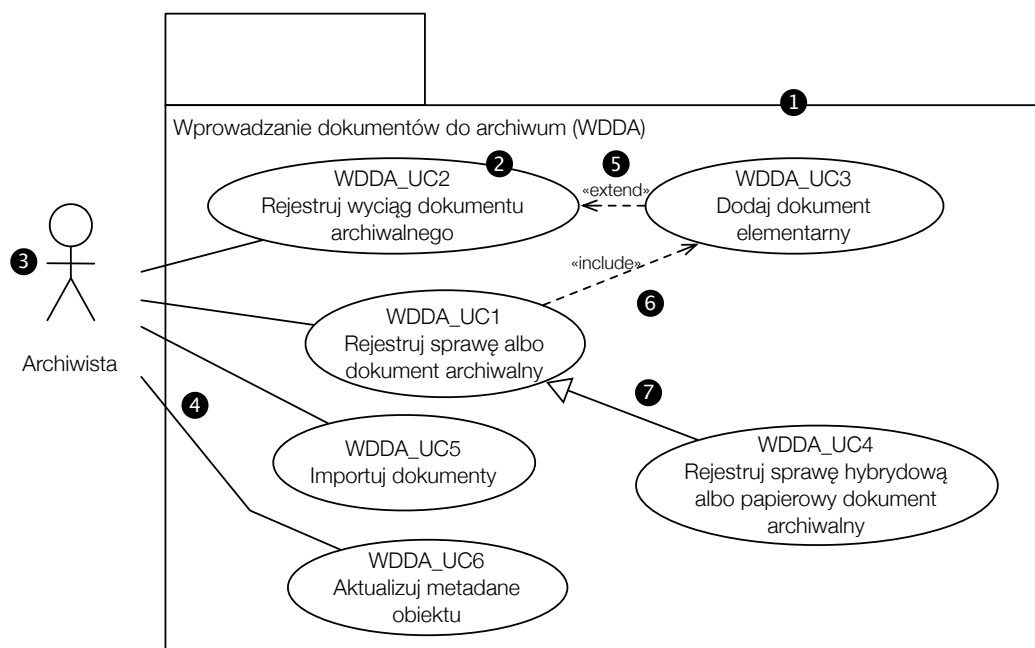
Często popełnianym błędem na wczesnym etapie zbierania wymagań jest działanie odwrotne – przystąpienie do specyfikowania szczegółowych wymagań bez szerszego oglądu całej funkcjonalności systemu. Takie podejście określa się właśnie jako „podejście wgłb”.

11: Najpierw wszcz, potem wgłb

Zanim przystąpisz do szczegółowego opisywania poszczególnych przypadków użycia, postaraj się zidentyfikować jak największą ich liczbę, nadając im identyfikator i tytuł. Zaczynaj od stworzenia diagramu przypadków użycia. Zbyt wczesne skupienie się na szczegółach może spowodować, że znikną z pola widzenia istotne procesy oraz wymagania.

A.3.1 Diagram przypadków użycia

Wygodnym narzędziem pozwalającym usystematyzować pracę z przypadkami użycia jest diagram przypadków użycia, będący częścią notacji UML [ÖP05].



Rysunek A.1: Przykładowy diagram przypadków użycia (1–granica systemu, 2–przypadek użycia, 3–aktor, 4–powiązanie aktora z przypadkiem użycia), 5–relacja «extend», 6–relacja «include», 7–relacja specjalizacji/generalizacji).

Diagram przypadków można sobie wyobrazić jako mapę przedstawiającą relacje pomiędzy przypadkami użycia, oraz ich powiązania z aktorami. Dostarcza on podstawowych informacji o zakresie funkcjonalnym systemu bez konieczności czytania szczegółowego opisu przypadków użycia. Niestety często diagram przypadków użycia jest błędnie utożsamiany z samymi przypadkami użycia, podczas gdy stanowi on jedynie wygodny, ale bardzo ogólny przegląd funkcji dostępnych w systemie.

Przykładowy diagram przypadków użycia został przedstawiony na rysunku A.1. Tworzenie diagramu rozpoczyna się od zarysowania granicy systemu (1). Przypadki użycia, przedstawiane w formie elips (2), znajdujące się wewnątrz obszaru wyznaczonego przez tę granicę, będą realizowane przez tworzony system. Jeśli przypadki realizowane są przez inny system, wówczas powinny zostać umieszczone poza obszarem wyznaczonym przez granice tworzonego systemu.

Dookoła obszaru systemu umieszczeni są aktorzy (3), którzy następnie są łączeni z odpowiednimi przypadkami użycia (4). Połączenie oznacza, że aktor bierze udział w danym przypadku użycia. Jeśli w danym przypadku bierze udział więcej niż jeden aktor, należy poprowadzić łuki od wszystkich tych aktorów do danego przypadku użycia.

Warto zaznaczyć, że diagram przypadków użycia nie służy do przedstawiania kolejności wykonywania czynności. Ta informacja jest przedstawiona w scenariuszach poszczególnych przypadków.

A.4 PODSTAWOWE ELEMENTY PRZYPADKU UŻYCIA

Przypadek użycia jest związany z określonym *celem* istotnym dla aktora i opisuje w jaki sposób ten cel może zostać osiągnięty. Najczęściej istnieje kilka alternatywnych scenariuszy, których wykonanie prowadzi do osiągnięcia takiego celu. Przypadki użycia pozwalają na zgrupowanie wielu scenariuszy w jednej strukturze.

Najbardziej typowe elementy przypadku użycia to:

- **unikatowy identyfikator**,
- **nazwa** – powinna odzwierciedlać *cel*, jaki chce osiągnąć aktor (typowa struktura językowa to czasownik + obiekt, np. Generuj raport),
- **aktorzy** – lista aktorów, którzy są bezpośrednio zainteresowani osiągnięciem celu. Listę aktorów można dodatkowo rozbić na *aktorów głównych*, bezpośrednio zainteresowanych osiągnięciem celu, oraz *aktorów pomocniczych*, którzy wspomagają aktora głównego w osiągnięciu celu,
- **opis** – krótki opis tekstowy streszczający przypadek użycia,
- **scenariusz główny** – zawiera najbardziej typową sekwencję czynności, które należy wykonać, aby osiągnąć cel. Może zostać przedstawiony w postaci opisowej (tekst ciągły), w której poszczególni aktorzy wykonują czynności. Jest to bardzo dobra forma wyjściowa, od której można rozpocząć tworzenie przypadku użycia. Bardziej przejrzystą formą jest scenariusz w postaci listy kroków.
- **scenariusze alternatywne** – inne scenariusze, które prowadzą do osiągnięcia celu przypadku użycia,
- **wyjątki** – opis sytuacji nietypowych, które mogą wystąpić w trakcie wykonania scenariuszy (np. błędy walidacji danych),
- **rozszerzenia** – dodatkowe czynności w trakcie realizacji przypadku użycia (np. przypadek użycia "Generuj raport" może posiadać rozszerzenie opisujące dodatkowe kroki, które należy wykonywać w przypadku generowania szczególnego rodzaju raportu).
- **poziom** – określa, czy przypadek użycia opisuje proces biznesowy, cele istotne dla użytkownika, czy podfunkcje (patrz rozdział A.2),
- **wyzwalacze** – opisują okoliczności w jakich następuje rozpoczęcie wykonania przypadku użycia,
- **warunki początkowe** – jest to lista warunków, które powinny być spełnione przed wykonaniem przypadku użycia,
- **warunki końcowe** – są to warunki, które powinny być prawdziwe po zakończeniu przypadku użycia.

A.5 ZAPISYWANIE PRZYPADKÓW UŻYCIA

A.5.1 Prezentowanie scenariuszy przypadków użycia

Scenariusze opisujące sposoby osiągnięcia celu stanowią najistotniejszą część opisu przypadku użycia.

Najprostszym sposobem przedstawiania scenariusza jest zapis w postaci tek-

Scenariusze

stu ciągłego. Należy zwrócić uwagę, aby dla każdej czynności zidentyfikować aktora, który ją wykonuje. Należy także pamiętać, że zapis scenariuszy przypadków użycia powinny być pozbawiony żargonu technicznego oraz szczegółów związanych z interfejsem użytkownika.

Przykładowo, scenariusz główny przypadku użycia “Wyślij wiadomość e-mail” zapisany w formie tekstu ciągłego mógłby wyglądać następująco:

Użytkownik wybiera opcję wysłania wiadomości e-mail. System prosi o wskazanie adresatów i podanie treści wiadomości. Użytkownik wybiera adresatów z książki kontaktów, podaje treść wiadomości oraz zatwierdza wysłanie wiadomości. System prezentuje informację o pomyślnym wysłaniu wiadomości.

Jednak preferowaną formą prezentacji scenariusza jest lista kroków. Przy tym zapisie każdy krok posiada swój numer, co ułatwia odwołania w tekście. Scenariusz wysłania wiadomości e-mail w formie listy kroków wyglądałby następująco:

1. Użytkownik wybiera opcję wysłania wiadomości e-mail.
2. System prosi o wskazanie adresatów i podanie treści wiadomości.
3. Użytkownik wybiera adresatów z książki kontaktów, podaje treść wiadomości oraz zatwierdza wysłanie wiadomości.
4. System prezentuje informację o pomyślnym wysłaniu wiadomości.

4: Typowe błędy w scenariuszach przypadków użycia

Typowe błędy w scenariuszach przypadków użycia:

- **zbyt długie scenariusze** – prawidłowy scenariusz powinien składać się z 3 do 9 kroków [Coc01]. Jeśli przypadek użycia jest dłuższy, wówczas warto rozważyć rozbięcie go na kilka krótszych przypadków.
- **włączanie opisów interfejsów użytkownika** – przypadki użycia opisują czynności, które musi wykonać użytkownik, aby osiągnąć cel, a nie interfejs użytkownika, który mu to umożliwi. Na przykład sformułowanie “Użytkownik wybiera sprawę” jest poprawne, w przeciwieństwie do “Użytkownik klika tytuł sprawy na liście rozwijalnej”; podobnie krok w postaci “System prezentuje pisma wchodzące w skład sprawy” jest poprawny, w przeciwieństwie do kroku “System prezentuje dokumenty wchodzące w skład sprawy wewnątrz edytowalnej tabeli”,
- **włączanie opisu danych do kroków** – w krokach przypadków użycia można odwoływać się do obiektów zdefiniowanych w słowniku danych, natomiast nie należy ich w tym miejscu definiować. Na przykład, następujący zapis jest niepoprawny: “Użytkownik podaje dane logowania - imię, nazwisko, hasło, hasło powtórzone itp.”,
- **brak wskazania aktora wykonującego dany krok** – pominięcie aktora może spowodować nieporozumienia podczas interpretacji danego kroku.

Oprócz scenariusza głównego przypadek użycia może zawierać scenariusze

Scenariusze alternatywne, wyjątki i rozszerzenia

alternatywne, wyjątki i rozszerzenia. Chociaż ich znaczenie jest różne, sposób prezentacji pozostaje taki sam i składa się z dwóch elementów.

Pierwszy to *zdarzenie*, które opisuje sytuację prowadzącą do wyboru alternatywnej ścieżki. Takie zdarzenie wiąże się z określonym krokiem (krokami), w trakcie realizacji których może wystąpić.

Najczęściej powiązania pomiędzy krokami a zdarzeniem dokonuje się poprzez umieszczenie identyfikatora kroku i dodaniu dodatkowej (kolejnej) litery alfabetu przed opisem zdarzenia. Na przykład, w trakcie realizacji kroku "4. System prezentuje informację o pomyślnym wysłaniu wiadomości" może się okazać, że system nie może połączyć się z serwerem poczty. Można wówczas opisać następujące zdarzenie wyjątkowe: "4.A System nie może nawiązać połączenia z serwerem poczty".

Drugim elementem jest scenariusz wykonywany w przypadku zaistnienia zdarzenia. Zasada pisania alternatywnego scenariusza jest taka sama jak w przypadku scenariusza głównego.

Kontynuując przykład, jeśli nie udało się nawiązać połączenia z serwerem poczty, system powinien poinformować o tym użytkownika. W tym przypadku wyjątek wraz ze scenariuszem wyglądałby następująco:

Wyjątki:

4.A. System nie może nawiązać połączenia z serwerem poczty.

4.A.1. System informuje użytkownika o niepowodzeniu nawiązania połączenia z serwerem poczty.

Przyjęła się konwencja, że po wykonaniu alternatywnego scenariusza należy przejść do kolejnych kroków scenariusza po tym, który spowodował dane zdarzenie. W prezentowanym przypadku użycia zdarzenie takie wystąpiło w kroku 4., dlatego wykonując krok 4.A.1. należy kontynuować wykonanie scenariusza głównego, tzn. zakończyć przypadek użycia. Zasadę domyślnego powrotu do głównego scenariusza można zmienić, umieszczając na końcu scenariusza alternatywnego zapis "Przejdź do kroku X" lub jawnie deklarując zakończenie przypadku. W prezentowanym przykładzie taką informację można umieścić po opisie kroku 4.A.1. albo jako krok 4.A.2.

A.5.2 Szablony dokumentacji przypadków użycia

Przykładowe szablony dokumentowania przypadków użycia, różniące się poziomem szczegółowości, zostały przedstawione na rysunku A.2.

Szablon uproszczony (1) można stosować na wstępnych etapach prac oraz dla mniej skomplikowanych przypadków użycia. Szablon pośredni (2) zawiera zestaw typowych elementów i może być stosowany w większości projektów. Szablon szczegółowy (3) przeznaczony jest dla rozbudowanych przypadków użycia i złożonych projektów.

A.6 RELACJE POMIĘDZY PRZYPADKAMI UŻYCIA

Przypadki użycia, podobnie jak wymagania, tworzą najczęściej strukturę hierarchiczną. Związki wewnątrz hierarchii są oparte na trzech relacjach: zawierania, rozszerzania i specjalizacji.

<p>1</p> <p>ID: UC1 Nazwa: Wyślij wiadomość e-mail</p> <p>Aktorzy główni: Użytkownik Aktorzy pomocniczy: System Poziom: Użytkownika Priorytet: Wysoki</p> <p>Opis: Użytkownika pragnie wysłać wiadomość e-mail. Wybiera opcję wysłania wiadomości, podaje adresatów i treść. System wysyła wiadomość oraz informuje o pomyślnym wysłaniu wiadomości.</p>	<p>3</p> <p>ID: UC1 Nazwa: Wyślij wiadomość e-mail Autor: Jan Kowalski Ostatnio modyfikował: Anna Kowalska Data utworzenia: 01.09.2009 Data ostatniej modyfikacji: 04.09.2009</p> <p>Aktorzy główni: Użytkownik Aktorzy pomocniczy: System Poziom: Użytkownika Priorytet: Wysoki</p> <p>Opis: Użytkownika pragnie wysłać wiadomość e-mail. Wybiera opcję wysłania wiadomości, podaje adresatów i treść. System wysyła wiadomość oraz informuje o pomyślnym wysłaniu wiadomości.</p> <p>Wyzwalacze: 1. Użytkownik pragnie wysłać wiadomość e-mail.</p> <p>Warunki początkowe: 1. Użytkownik posiada skonfigurowane konto pocztowe.</p> <p>Warunki końcowe: 1. Wiadomość e-mail została wysłana do adresatów.</p> <p>Scenariusz główny: 1. Użytkownik wybiera opcję wysłania wiadomości e-mail. 2. System prosi o wskazanie adresatów i podanie treści wiadomości. 3. Użytkownik wybiera adresatów z książki kontaktów, podaje treść wiadomości oraz zatwierdza wysłanie wiadomości. 4. System wysyła wiadomość e-mail do wskazanych adresatów oraz prezentuje informację o pomyślnym wysłaniu wiadomości.</p> <p>Scenariusze alternatywne: 3.A. Użytkownik pragnie wysłać e-mail do adresata spoza książki. 3.A.1. Użytkownik podaje adres e-mail adresata. 3.A.2. Przejdź do kroku 4.</p> <p>Rozszerzenia: 3.A. Użytkownik pragnie dołączyć pliki jako załączniki. 3.A.1. Użytkownik wskazuje pliki, które mają być dołączone do wiadomości.</p> <p>Wyjątki: 4.A. System nie może nawiązać połączenia z serwerem poczty. 4.A.1. System informuje użytkownika o braku możliwości nawiązania połączenia z serwerem poczty.</p> <p>Dodatkowe wymagania: 1. W stopce tworzonej wiadomości powinna zostać automatycznie dołączona informacja o stanowisku osoby wysyłającej e-mail oraz jej dane kontaktowe.</p>
<p>2</p> <p>ID: UC1 Nazwa: Wyślij wiadomość e-mail</p> <p>Aktorzy główni: Użytkownik Aktorzy pomocniczy: System Poziom: Użytkownika Priorytet: Wysoki</p> <p>Wyzwalacze: 1. Użytkownik pragnie wysłać wiadomość e-mail.</p> <p>Warunki początkowe: 1. Użytkownik posiada skonfigurowane konto pocztowe.</p> <p>Warunki końcowe: 1. Wiadomość e-mail została wysłana do adresatów.</p> <p>Scenariusz główny: 1. Użytkownik wybiera opcję wysłania wiadomości e-mail. 2. System prosi o wskazanie adresatów i podanie treści wiadomości. 3. Użytkownik wybiera adresatów z książki kontaktów, podaje treść wiadomości oraz zatwierdza wysłanie wiadomości. 4. System wysyła wiadomość e-mail do wskazanych adresatów oraz prezentuje informację o pomyślnym wysłaniu wiadomości.</p> <p>Scenariusze alternatywne i rozszerzenia: 3.A. Użytkownik pragnie wysłać e-mail do adresata spoza książki. 3.A.1. Użytkownik podaje adres e-mail adresata. 3.A.2. Przejdź do kroku 4. 3.B. Użytkownik pragnie dołączyć pliki jako załączniki. 3.B.1. Użytkownik wskazuje pliki, które mają być dołączone do wiadomości.</p> <p>Wyjątki: 4.A. System nie może nawiązać połączenia z serwerem poczty. 4.A.1. System informuje użytkownika o braku możliwości nawiązania połączenia z serwerem poczty.</p>	

Rysunek A.2: Przykładowe szablony dokumentowania przypadków użycia (1 – szablon uproszczony, 2 – szablon pośredni, 3 – szablon szczegółowy).

A.6.1 Relacja zawierania

Często zdarza się, że w pewnym miejscu jednego przypadku użycia następuje wykonanie czynności będących przedmiotem innego przypadku. Jeśli wykonanie takich czynności *jest konieczne*, to mówimy o relacji zawierania przypadków użycia (*ang. include*).

Na przykład, w systemie dla sklepu internetowego został zidentyfikowany przypadek użycia „Znajdź produkt”. Opisuje on dwa alternatywne scenariusze odnalezienia produktu: poprzez wyszukiwanie oraz przeglądanie struktury kategorii produktów. Następnie został zidentyfikowany kolejny przypadek „Dokonaj zakupu produktu”. Okazało się, że pierwszy krok głównego scenariusza tego przypadku użycia polega na wyszukaniu produktu. Aby uniknąć powtarzania treści przypadku użycia „Znajdź produkt”, wystarczy oba przypadki powiązać relacją zawierania („Zakup produkt” zawiera „Znajdź produkt”).

Relacja
zawierania
«include»

Na diagramie przypadków użycia relacja ta ma postać łuku skierowany (linia przerywana) od przypadku, który wywołuje, do przypadku wywoływanego. Łuk taki opatruje się słowem kluczowym «include».

Na rysunku A.1 przedstawiono diagram przypadków użycia i przykładową releację zawierania (6) pomiędzy przypadkami “Rejestruj sprawę albo dokument archiwalny” oraz “Dodaj dokument elementarny”. Relacja oznacza, że w prezentowanym systemie częścią procesu rejestrowania dokumentu archiwalnego jest dodanie dokumentów elementarnych.

Nie ma określonego formalnego sposobu zapisu tego typu relacji w treści przypadków użycia. Dobrym podejściem jest umieszczenie odwołania w kroku, który stanowi wywołanie zawieranego przypadku użycia. Można na przykład umieścić identyfikator zawieranego przypadku użycia oraz odnośnik do strony w nawiasach kwadratowych.

W przypadku omówionej wcześniej relacji zawierania przedstawionej na rysunku A.1, odwołanie w scenariuszu przypadku użycia “Rejestruj sprawę albo dokument archiwalny” mogłoby wyglądać następująco:

... Archiwista dodaje dokumenty elementarne [WDDA_UC3] ...

A.6.2 Relacja rozszerzania

Związkiem podobnym do relacji zawierania jest relacja rozszerzania (*ang. extend*). Przypadek użycia może rozszerzać scenariusze innego przypadku użycia, włączając do nich swój scenariusz. Główna różnica polega na tym, że włączone czynności są *opcjonalne*.

Na diagramie przypadków użycia relacja ta jest przedstawiana jako łuk skierowany (linia przerywana) od przypadku użycia, który rozszerza (dodaje swój scenariusz) do przypadku użycia, który jest rozszerzany. Dodatkowo łuk jest opatrywany słowem kluczowym «extend».

Na rysunku A.1 przedstawiono graficzną reprezentację takiej relacji na diagramie przypadków użycia (5). Przypadek użycia “Dodaj dokument elementarny” rozszerza możliwości przypadku “Rejestruj wyciąg dokumentu archiwalnego”. Oznacza to, że realizując scenariusz przypadku “Rejestruj wyciąg dokumentu archiwalnego” istnieje *możliwość* dodania dokumentu elementarnego.

Relacja
rozszerzania
«extend»

A.6.3 Relacja specjalizacji

Często zdarza się, że zidentyfikowane przypadki użycia różnią się jedynie częścią scenariusza. Na przykład, po dogłębnej analizie przypadku “Generowanie raportu” okazało się, że system może generować 30 różnych raportów. We wszystkich przypadkach podstawowe czynności są takie same, natomiast w przypadku 10 raportów należy wykonać dodatkowe czynności. W jaki sposób zawrzeć tę informację w modelu przypadków użycia?

Jedną z możliwości jest stworzenie jednego przypadku użycia “Generowanie raportu”, który zawierałby 10 rozszerzeń opisujących dodatkowe czynności, jakie należy wykonać, dla poszczególnych raportów. Takie podejście jest akceptowalne, o ile alternatywnych ścieżek byłoby tylko kilka (3-4). Niestety w tym przypadku 10 rozszerzeń sprawiłoby, że przypadek użycia stałby się mało czytelny.

W takiej sytuacji lepszym rozwiązaniem jest użycie relacji specjalizacji / generalizacji pomiędzy przypadkami użycia. Polega ona powiązaniu przypadku bazowego (ogólnego), który opisuje wspólny zakres czynności, z przypadkiem specjalizowanym, który przejmuje wszystkie własności przypadku bazowego i wzbogaca go o nowe kroki. Relacja ta przedstawiana jest na diagramie przypadków użycia w postaci łuku skierowanego od przypadku wyspecjalizowanego do przypadku bazowego. Grot łuku ma postać trójkąta. Na rysunku A.1 przedstawiono graficzną reprezentację relacji na diagramie przypadków użycia (7), pomiędzy przypadkami "Rejestruj sprawę albo dokument archiwalny" i "Rejestruj sprawę hybrydową albo papierowy dokument archiwalny".

Relacja
specjalizacji

Wykorzystanie tej relacji jest w zasadzie konieczne, gdy występuje dużo zbliżonych wariantów danego przypadku użycia. Niestety, nie ma jednolitej metody przedstawienia tej relacji w opisie tekstowym przypadku użycia. Najczęściej informacja o niej ma charakter adnotacji w specjalizowanym przypadku użycia opisującej zmiany w scenariuszu w stosunku do wersji bazowej, np.:

Pomiędzy krokami 3-4 przypadku bazowego wykonaj ...

Wykonaj kroki 1-4 scenariusza bazowego, a następnie ...

W kroku 3 przypadku bazowego, zamiast usunięcia wpisu dokonaj jego przeniesienia do archiwum.

A.7 MODELOWANIE PROCESÓW BIZNESOWYCH Z WYKORZYSTANIEM PRZYPADKÓW UŻYCIA

Dotychczas przedstawione informacje dotyczące przypadków użycia odnoszą się zarówno do opisu procesów biznesowych, jak i wymagań funkcjonalnych. Podstawowa różnica polega na tym, że w przypadku procesów biznesowych zwykle opisywane są interakcje pomiędzy ludźmi. Inne elementy charakterystyczne dla procesów biznesowych [NNOO06] to :

- **aktorzy zbiorowi** – w przypadku procesów biznesowych często występują grupy aktorów (np. komitet, zarząd itd.),
- **aktorzy tworzeni dynamicznie** – może się zdarzyć, że w trakcie realizacji procedury zostanie powołany nowy aktor, który od momentu swojego stworzenia będzie brał udział w interakcji. Na przykład, zarząd może powołać nowego aktora o nazwie "dyrektor", który będzie brał udział w następnych krokach procesu,
- **metamorfoza aktorów** – w procesach biznesowych zdarza się, że w trakcie realizacji następuje przemiana aktora. Na przykład, w procesie opisującym przeprowadzenie sprawy sądowej, na początku występuje aktor "podejrzany", który z biegiem procesu może zostać "skazanym",
- **prolog** – to sekwencja kroków umieszczona przed scenariuszem głównym, która opisuje czynności poprzedzające wykonanie procesu biznesowego. Umieszczenie prologu może być przydatne, jeżeli istnieją czynności, które nie są częścią opisywanego procesu (być może są nawet częścią innych procesów), ale są z nim związane,
- **dokumenty, wejście/wyjście** – w opisie procesów biznesowych warto dodatkowo opisać wszystkie dokumenty (obiekty biznesowe), które w nim wystę-

ID: BC03 Nazwa: Zakończenie przechowywania dokumentu w archiwum dokumentów elektronicznych Autor: Jan Kowalski Data utworzenia: 01.09.2009 Ostatnio modyfikował: Anna Kowalska Data ostatniej modyfikacji: 04.09.2009
Aktorzy główni: Urząd Miasta Poznania (UMP), System ADE Aktorzy pomocniczy: Archiwista Poziom: Biznesowy Priorytet: Wysoki
Opis: Upłynął termin ważności teczki i należy podjąć decyzję czy teczkę dalej przechowywać, czy dokonać jej zniszczenia.
Wyzwalacze: 1. Upłynął termin ważności teczki zgodnie z kategorią archiwalną teczki (Reg6).
Dokumenty wejściowe: Brak
Dokumenty wyjściowe: 1. Zamknięta teczka
Prolog: brak
Scenariusz główny: 1. System ADE umieszcza teczkę w Procedurze Przeglądu Teczek (PPT). 2. Wyznaczone osoby z UMP przeglądają teczkę i podejmują decyzję o jej usunięciu. 3. System ADE usuwa dane teczki i oznacza teczkę jako usuniętą.
Scenariusze alternatywne i rozszerzenia: 3.A. Teczka jest teczką hybrydową. 3.A.1. Wyznaczone osoby z UMP niszczą papierową część teczki (Reg8). 3.B. Teczka ma kategorię archiwalną A. 3.B.1. Archiwista eksportuje teczkę z ADE. 3.B.2. Archiwista przekazuje wyeksportowane dane do Narodowego Archiwum Cyfrowego.
Wyjątki: brak

Reg6 - każdy dokument archiwalny w teczce musi domyślnie być przechowywany przez okres wynikający z kategorii archiwalnej przypisanej tej teczce. Kategoria archiwalna A - wieczne przechowywanie, BE-n - po n liczbie lat przekazanie teczki do ekspertyzy, B-n - po n latach usunięcie.

Reg8 - w przypadku usuwania teczki hybrydowej musi zostać zniszczona zarówno część elektroniczna teczki jak i część papierowa.

Rysunek A.3: Przypadek użycia poziomu biznesowego – Zakończenie przechowywania dokumentu w archiwum dokumentów elektronicznych.

pują, z podziałem na dokumenty wejściowe (dostępne przed rozpoczęciem procesu) oraz wyjściowe (powstające w wyniku jego realizacji).

Przykładowy proces zakończenia przechowywania dokumentu w archiwum dokumentów elektronicznych, został przedstawiony na rysunku A.3. Na potrzeby opisu procesu biznesowego dokonano adaptacji szablonu przypadku użycia przedstawionego na rysunku A.2.

Język UML

UML (ang. Unified Modeling Language) jest notacją standaryzowaną przez organizację Object Management Group, służącą do modelowania i analizy systemów informatycznych. UML z uwagi na obecnie dominującą pozycję na rynku, stał się także standardem *de facto* oraz swego rodzaju *lingua franca* współczesnej inżynierii oprogramowania.

UML definiuje kilkanaście typów diagramów, na których przedstawiane są rozmaite perspektywy modelowanego systemu. Analityk, przedstawiając pomysł na realizację systemu, modeluje wybrane rozwiązania za pomocą niektórych z nich.

B.1 DIAGRAM KLAS

Najpopularniejszym rodzajem diagramu UML jest diagram klas. Właśnie ze względu na swoją popularność często jest nazywany po prostu "diagramem UMLowym". Służy on do zaprezentowania grupy klas (lub obiektów) oraz ich wzajemnych relacji.

Podstawowymi elementami na tym diagramie są klasy, które można utożsamić np. z obiektami biznesowymi lub elementami modelu danych. Klasy są reprezentowane przez prostokąty podzielone na trzy przedziały. W pierwszym przedziale umieszczana jest nazwa klasy oraz informacje dodatkowe, takie jak pakiet czy stereotyp klasy (nie są one jednak istotne przy modelowaniu wymagań). Drugi przedział zawiera listę atrybutów obiektu, natomiast trzeci opisuje operacje, jakie klasa może wykonać.

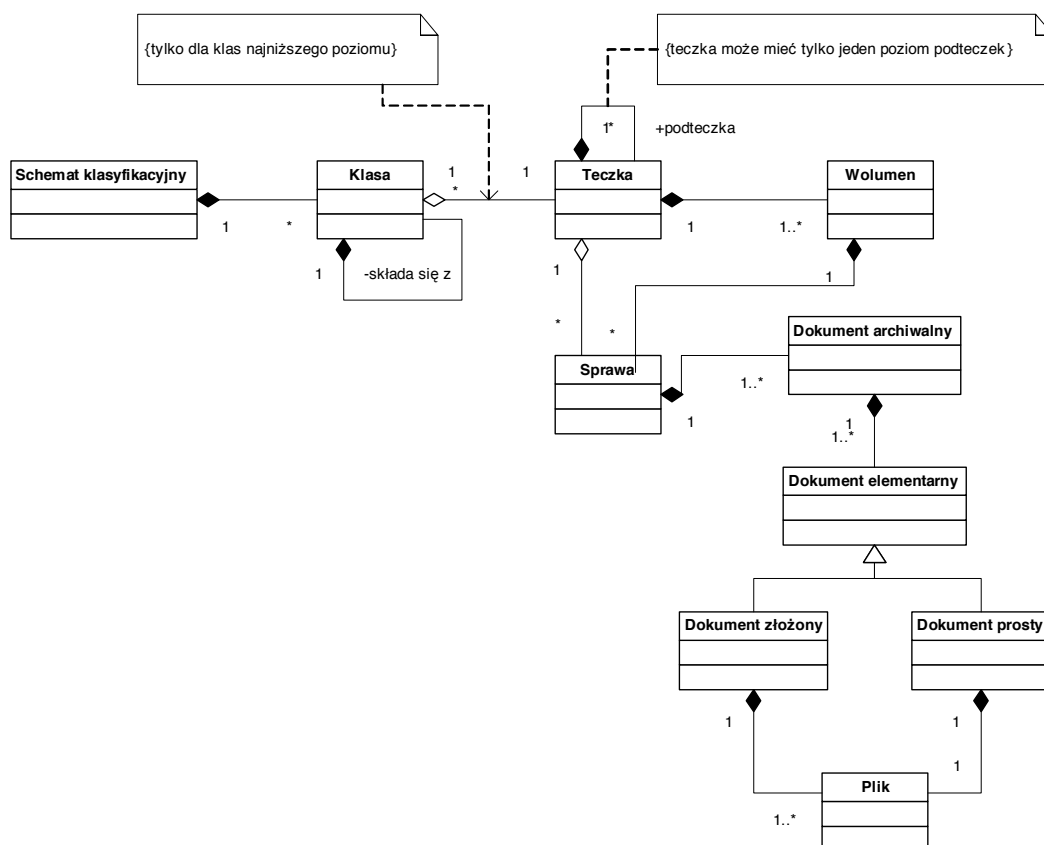
Klasa UML

Obiekty umieszczone na diagramie klas mogą być wchodzić w relacje. UML przewiduje wiele różnych rodzajów relacji, jednak tutaj zostaną omówione tylko najważniejsze z nich.

- **Zależność** to najbardziej ogólna relacja, jaka może wystąpić pomiędzy dwoma obiektami. Oznacza, że jeden obiekt w *jakikolwiek* sposób zależy od drugiego – aby więc przekazać informację dotyczącą natury zależności, często

dodaje się przy niej tzw. słowo kluczowe zapisywane w “pazurkach” (franc. guillemets), np. «refines» oznacza relację uszczegółowienia. Na diagramie relacja zależności jest oznaczana skierowanym łukiem o linio przerywanej ze zwrotem w kierunku od obiektu zależnego do nadrzędnego.

- **Asocjacja** to najczęściej spotykana w praktyce relacja, oznaczająca, że powiązane nią obiekty mogą znaleźć się w związku, który niekoniecznie ma stały charakter oraz nie narusza niezależności każdego z obiektów. Relacja ta jest oznaczana linią ciągłą, może być skierowana lub dwukierunkowa. Asocjacja może posiadać krotność, która wskazuje, ile instancji klas bierze udział w relacji.
- **Dziedziczenie** wiąże obiekty w struktury hierarchiczne, od najbardziej ogólnego do obiektów szczegółowych. Obiekty uczestniczące w takich relacjach opisują zatem ten sam byt, tylko na różnych poziomach abstrakcji, np. klasa *Człowiek* może posiadać podklasy (czyli obiekty dziedziczące) *Mężczyzna* i *Kobieta*. Na diagramach relacja jest reprezentowana poprzez strzałkę z zamkniętym grotem skierowanym ku obiektowi ogólniejszemu (patrz rysunek B.1). W analogiczny sposób przedstawiana była relacja uogólniania na diagramach przypadków użycia.



Rysunek B.1: Przykładowy diagram klas

- **Agregacja i kompozycja** są relacjami o zbliżonej semantyce i służą do reprezentowania związków typu całość-część. Oznacza to, że obiekt jednego typu *składa się* lub *zawiera* grupę obiektów drugiego typu. Relacja to podobna jest do asocjacji, jednak wprowadza nierównowagę między obiektami – rolę znacznie ważniejszą pełni zwykle obiekt zawierający obiekty typu *część*: zwykle zarządza nimi, a w przypadku relacji kompozycji także decyduje o ich powstaniu i usunięciu. Na diagramie relacje te są przedstawiane w postaci linii zakończonej rombem (pustym w przypadku agregacji i wypełnionym dla kompozycji) po stronie obiektu reprezentującego całość.

Dla ułatwienia pracy czytelnikowi diagramu, wszystkie relacje mogą posiadać nazwy (będące zwykle frazami odczasownikowymi), ułatwiające konstruowanie zdań, w których rolę podmiotu i przedmiotu pełnią klasy, a relacja stanowi orzeczenie, np. w zdaniu „Archiwista zapisuje Dokument” *Archiwista* i *Dokument* są klasami powiązanymi relacją opisaną słowem „zapisuje”.

Przykład diagramu klas został przedstawiony na rysunku B.1. Na diagramie tym przedstawiono klasy związane z archiwum dokumentów elektronicznych. *Schemat klasyfikacyjny* składa się z dowolnej liczby *Klas* tworzących hierarchię drzewiastą o wysokości drzewa nie mniejszej niż 1. Z *Klasami* znajdującymi się na najniższym poziomie hierarchii mogą być powiązane *Teczki*. *Teczki* mogą posiadać związane z nimi *podteczki*. *Teczka* jest podzielona na *Woluminy*, zawierające *Sprawy*. *Wolumin* (a przez to i *Teczka*) mogą zawierać dowolną liczbę *Spraw*. *Sprawa* jest niepustym zbiorem obiektów o nazwie *Dokument archiwalny*, z których każdy jest *Dokumentem prostym* (składającym się z jednego *Pliku*) lub *Dokumentem złożonym* (składającym się z wielu *Plików*).

B.2 DIAGRAM STANU

Diagram stanu (ang. *state machine diagram*) reprezentuje zmiany zachowania pewnej części systemu poprzez ukazanie go właśnie jako maszyny stanowej – czyli zbioru stanów, w których ten system może się znaleźć, wraz z opisem okoliczności, w jakich stany te ulegają zmianie. Stan to etap w cyklu życia obiektu, w którym określony warunek ma niezmienną wartość, np. obiekt *Człowiek* znajduje się w stanie *Niepełnoletni* w trakcie, gdy spełniony jest warunek *wiek < 18*.

Podstawowymi elementami diagramu są stany obiektu, reprezentowane przez prostokąty o zaokrąglonych narożnikach, połączone strzałkami przejść. Na strzałkach umieszczone są informacje dotyczące okoliczności zmiany stanu (nazwa przejścia, nazwa zdarzenia wyzwalającego, definicja warunku dopuszczalności przejścia). Dodatkowo mogą pojawić się specjalnie oznaczone stany początkowy i końcowy.

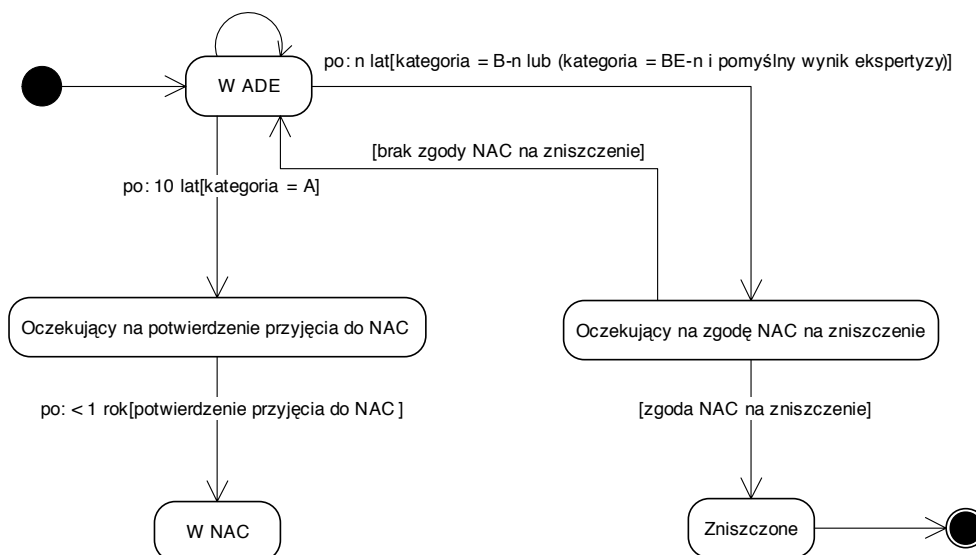
Obiekt (lub grupa obiektów) w momencie utworzenia znajduje się w stanie początkowym, a następnie zmienia stan w reakcji na nadchodzące zdarzenia (o ile spełnione są warunki określone dla każdego przejścia).

Opisując stany i ich przejścia należy zwrócić uwagę na dwie istotne właściwości: *rozłączność przejść* (nie mogą istnieć dwa jednoczesne przejścia prowadzące z jednego stanu) i ich *kompletność* (czyli wszystkie możliwe przejścia powinny być

wyspecyfikowane). Właściwości te decydują o poprawności funkcjonowania maszyny stanowej przedstawionej na diagramie.

Przykład diagramu stanu dla obiektu *Dokument* został przedstawiony na rysunku B.2. Każdy *Dokument* posiada atrybut o nazwie *kategoria archiwalna*, określający sposób, w jaki zmienia się jego stan. Dokumenty o kategorii A są po 10 latach przekazywane do Archiwum Państwowego. Dokumenty o kategorii BE-*n*, gdzie *n* oznacza liczbę lat, są po upływie tego okresu (np. BE-10 oznacza okres 10 lat od wytworzenia dokumentu) poddawane ekspertyzie, która może zadecydować o jego zniszczeniu lub zmianie kategorii (np. na BE-15, co będzie oznaczało przeprowadzenie ekspertyzy po kolejnych 5 latach, tj. po 15 latach od wytworzenia dokumentu). Dokumenty o kategorii B-*n* są traktowane podobnie jak w przypadku kategorii BE-*n*, jednak zniszczenie dokumentu po *n* latach nie wymaga uprzedniej ekspertyzy. Zarówno dla kategorii B-*n* i BE-*n* do zniszczenia dokumentu konieczna jest zgoda Archiwum Państwowego (w tym przypadku Narodowego Archiwum Cyfrowego).

po: *n* lat[kategoria = BE-*n* i negatywny wynik ekspertyzy] / zmiana kategorii



Rysunek B.2: Przykładowy diagram stanu

B.3 DIAGRAM CZYNNOŚCI

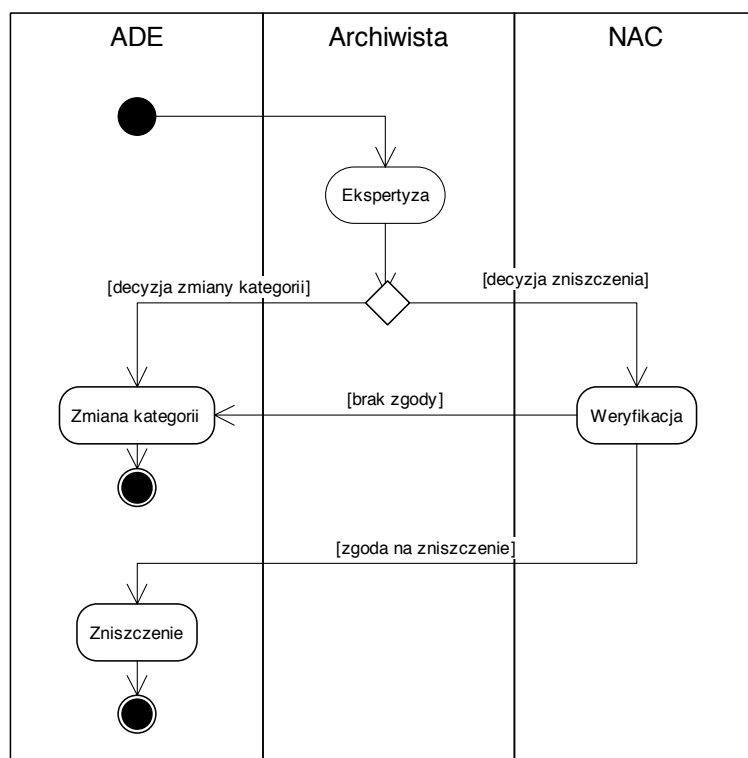
Diagram czynności służy przede wszystkim do przedstawiania złożonych algorytmów, ale można go zastosować również do opisu reguł biznesowych, procesów biznesowych czy scenariuszy przypadków użycia. Na diagramie tym modelowana czynność ma postać grafu skierowanego, w którym węzły reprezentują instrukcje do wykonania, a łuki – przejścia pomiędzy nimi. Punktem rozpoczęcia jest wypełniony okrąg, od którego można w skończony sposób przejść do punktu

zakończenia, reprezentowanego przez okrąg z mniejszym wypełnionym okręgiem w środku. Każda instrukcja algorytmu jest przedstawiana w postaci prostokąta z zaokrąglonymi wierzchołkami, w którym umieszczony jest opis czynności. Wyjątkiem jest instrukcja warunkowa, reprezentowana przez romby, w którym umieszczona jest treść warunku; z rombu poprowadzone są łuki do instrukcji, które należy wykonać w zależności od wartości warunku. Wykonanie przejścia wzdłuż dowolnego łuku może być uzależnione od spełnienia pewnego warunku, zapisywanego nad łukiem w nawiasie kwadratowym. Do czasu spełnienia tego warunku, proces oczekuje i nie wykonuje innych czynności.

Na diagramie tym istnieje także możliwość przedstawienia operacji współbieżnych. Rozdzielenie sterowania jest reprezentowane przez grubą kreskę, z której wychodzą równoległe łuki symbolizujące wykonanie wielu procesów naraz. Procesy te mogą zostać z powrotem złączone (zsynchronizowane) za pomocą analogicznej grubej kreski, do której dochodzą łuki przejść. Tym razem symbolizuje ona oczekiwanie na wszystkie współbieżne czynności. Zasadą (choć nie wymogiem) jest, aby każde rozdzielenie sterowania posiadało odpowiadające sobie złączenie procesów.

Na diagramie czynności nacisk położony jest na zapis schematu algorytmu, natomiast drugorzędną kwestią jest podmiot, który daną czynność wykonuje. Aby jednak umożliwić prezentację takiej informacji, diagram można podzielić na tzw. toru (ang. swimlanes), z których każdy reprezentuje jeden obiekt. Umieszczenie instrukcji w danym torze oznacza, że za jej wykonanie odpowiada obiekt-właściciel tego toru.

Przykładowy diagram czynności, przedstawiający procedurę obsługi akt o kategorii archiwalnej BE-n, znajduje się na rysunku B.3. Punktem wyjścia jest dokonanie przez Archiwistę ekspertyzy dokumentu. Jeżeli w jej wyniku zapadnie decyzja zmiany kategorii, wówczas dokument pozostaje w Archiwum; jeżeli zapadnie decyzja zniszczenia dokumentu, jest ona weryfikowana przez NAC, które udziela zgody na zniszczenie (wówczas dokument jest niszczone w ADE) lub jej odmawia (i wówczas kategoria archiwalna dokumentu ulega zmianie). Procedura ta może być wykonywana cyklicznie, a jej wyzwalaczem jest upływ okresu do przeprowadzenia kolejnej ekspertyzy.



Rysunek B.3: Przykładowy diagram czynności

Modelowanie danych

W dodatku tym zostały zaprezentowane dwie popularne techniki prezentacji modelu danych. Pierwsza wykorzystuje ustrukturalizowany tekst, natomiast druga, będąca przykładem notacji graficznej, wykorzystuje specyficzną postać diagramu klas UML (o diagramie klas można przeczytać w dodatku B).

C.1 NOTACJA OPARTA NA TEKŚCIE STRUKTURALNYM

W tej części jest przedstawione podejście do prezentacji modelu danych wykorzystujące tekst strukturalny [Wie03, DeM79, RR98].

Notacja pozwala przedstawiać *proste elementy danych*, *kompozycje elementów*, *krotności elementów* oraz *wyliczenia*.

Zgodnie z nią, każdy element danych jest opisany za pomocą nazwy umieszczonej po lewej stronie znaku "=", oraz jego definicji po prawej.

Proste elementy danych to obiekty, które nie podlegają dalszej dekompozycji. Zazwyczaj definiowane są w postaci opisu tekstowego zapisywanego pomiędzy znakami "***". Na przykład:

Znak sprawy = * ciąg znaków w postaci XX.JRWA-nr/YY, gdzie XX - symbol literowy komórki organizacyjnej, JRWA – symbol liczbowy hasła wg. JRWA, nr – kolejna liczba, pod którą zarejestrowano sprawę w spisie spraw, YY – dwie ostatnie cyfry roku, w którym sprawę wszczęto *

Jeśli istnieje potrzeba umieszczenia komentarza, poprzedza się go ciągiem znaków złożonym z dwóch symboli gwiazdki – "***".

Kompozycja elementów jest strukturą, która grupuje wiele elementów danych. Poszczególne elementy składowe oddzielamy znakiem "+". Jeśli dany element jest opcjonalny, wówczas umieszcza się go w nawiasach "()". Na przykład:

Pismo = Znak sprawy
 + Nagłówek pisma
 + Znak
 + Odwołanie do pisma źródłowego
 + Data podpisania
 + Odbiorca
 + Treść pisma
 + Podpis
 + Lista adresatów
 + (Liczba załączników)
 + (Termin załatwienia sprawy)
 + (Wskazówki dla kancelarii)

Każdy z elementów składowych (np. Nagłówek pisma) powinien być zdefiniowany w analogiczny sposób.

Krotność elementów określa liczbę elementów wchodzących w skład danego atrybutu. Na przykład, w przedstawionej definicji Pisma wystąpił element nazwany "Listą adresatów". Lista taka składa się z jednego lub więcej adresatów. W zapisie obiektu element występujący wielokrotnie jest umieszczony w nawiasach klamrowych "{}". Dodatkowo przed nawiasem można umieścić minimalną i maksymalną liczbę elementów w formacie *min:max*. Listy adresatów wyglądałyby więc następująco:

Lista adresatów = 1:{Adresat}

W przykładzie celowo nie podano maksymalnej liczby adresatów, ponieważ nie jest ona w żaden sposób ograniczona. Natomiast wiadomo, że lista musi zawierać przynajmniej jednego adresata.

Wyliczenie polega na enumeratywnym określeniu wartości, jakie może przyjmować dana cecha. Listę alternatywnych wartości umieszcza się w nawiasach kwadratowych "[]", a poszczególne alternatywy oddziela się znakiem "|". Na przykład, cecha "Załatwienie sprawy" może przyjmować jedną z dwóch wartości: *tymczasowe* albo *ostateczne*. Definicja tej cechy może wyglądać następująco:

Załatwienie sprawy = [tymczasowe | ostateczne]

C.2 MODELOWANIE DANYCH ZA POMOCĄ JĘZYKA UML

Wprawdzie modelowanie danych nie jest podstawowym zastosowaniem języka UML, jednak jest ono możliwe dzięki obecności w UML specjalnego mechanizmu rozszerzeń – profili. Profil pozwala dostosować znaczenie symboli UML do danej dziedziny zastosowań, zwiększając w ten sposób siłę ekspresji UMLa. Profile definiują zestawy tzw. słów kluczowych (lub stereotypów), które – nałożone na elementy modelu – nadają im inne znaczenie.

Profil do modelowania danych definiuje słowa kluczowe związane przede wszystkim z elementami diagramu klas. Tabela w bazie danych jest reprezentowana za pomocą klasy ze stereotypem «Table» lub «Relational Table». Jej kolumny

to atrybuty klasy. Kolumny wymagające nałożenia dodatkowych ograniczeń również są opisywane za pomocą stereotypów, np. klucz podstawowy jest atrybutem ze stereotypem «PK», klucz obcy – atrybutem ze stereotypem «FK». Jeżeli z ograniczeniem integralnościowym (np. kluczem podstawowym tabeli) jest związane określone zachowanie, wówczas jest ono dodatkowo modelowane jako metoda o tym samym stereotypie (czyli «PK»). , jako operacje wewnątrz klasy, zapisywane są informacje o procedurach składowanych (operacja ze stereotypem «Proc»), wyzwalaczach (operacja ze stereotypem «Trigger») i innych.

Powiązania pomiędzy tabelami są reprezentowane za pomocą zwykłej asocjacji UML (ale opatrzonej odpowiednim stereotypem), która definiuje nazwy ról tabel uczestniczących w powiązaniu oraz ich krotności. W powiązaniu dwóch tabel można zawsze wyróżnić tabelę pełniącą rolę rodzica, definiującego klucz podstawowy, oraz potomka, posiadającego klucz obcy zawierający klucz podstawowy rodzica. Powiązanie takie jest opatrzone stereotypem «identifying», jeżeli klucz obcy potomka jest w całości zależny od klucza podstawowego rodzica, czyli gdy rodzic nadaje tożsamość potomkowi, lub stereotypem «non-identifying», jeżeli potomek jest niezależny od rodzica (nie zawiera w całości jego klucza podstawowego).

Bibliografia

- [Amb04] S.W. Ambler. *The object primer: Agile model driven development with UML 2*. Cambridge University Press, 2004.
- [Coc01] A. Cockburn. *Writing effective use cases*. Addison-Wesley Boston, 2001.
- [DeM79] T. DeMarco. *Structured Analysis and Systems Specification*. New York: YOURDON Press, 1979.
- [iee98a] *IEEE Guide for Information Technology – System Definition – Concept of Operations (ConOps) Document, IEEE Std 1362-1998*, 1998.
- [iee98b] *IEEE Recommended Practice for Software Requirements Specifications, IEEE Std 830-1998*, 1998.
- [iso04] *ISO/IEC 9126: Information technology - Software Product Evaluation - Quality characteristics and guidelines for their use*. International Organization for Standardization, 2001-2004.
- [Jac87] I. Jacobson. Object-oriented development in an industrial environment. *ACM SIGPLAN Notices*, 22(12):183–191, 1987.
- [Jon07] C. Jones. *Estimating Software Costs: Bringing Realism to Estimating*. McGraw-Hill Osborne Media, 2007.
- [NNOO06] Jerzy R. Nawrocki, Tomasz Nędzia, Mirosław Ochodek, and Łukasz Olek. Describing business processes with use cases. In *BIS*, pages 13–27, 2006.
- [ÖP05] G. Övergaard and K. Palmkvist. *Use cases: patterns and blueprints*. Addison-Wesley, 2005.
- [PE00] M. Penker and H.E. Eriksson. *Business Modeling With UML: Business Patterns at Work*. John Wiley & Sons, 2000.
- [Pio07] M. Piotrowski. *Notacja modelowania procesów biznesowych - podstawy*. BTC, 2007.
- [RR98] J. Robertson and S. Robertson. *Complete Systems Analysis: The Workbook, the Textbook, the Answers*. Dorset House Publishing Co., Inc. New York, NY, USA, 1998.
- [WA02] D.R. Windle and L.R. Abreo. *Software requirements using the unified process: a practical approach*. Prentice Hall PTR, 2002.

- [Wie03] K.E. Wiegers. *Software requirements*. Microsoft Press Redmond, WA, USA, 2003.
- [Wit07] Stephen Withall. *Software requirement patterns*. Microsoft Press, Redmond, WA, USA, 2007.
- [Śmi05] M. Śmiałek. *Zrozumieć UML 2.0. Metody modelowania*. Helion, 2005.

Skorowidz

- «extend», 52
- «include», 51
- łatwość adaptacji, 41
- łatwość analizy, 39
- łatwość instalacji, 41
- łatwość nauczania się, 30
- łatwość operowania, 30
- łatwość testowania, 41
- łatwość utrzymania, 39
- łatwość zamiany, 42
- łatwość zmiany, 40

- agregacja, 57
- aktor, 22, 45
- aktor główny, 48
- aktor pomocniczy, 48
- asocjacja, 56
- atrakcyjność, 31

- bezpieczeństwo, 36
- bezpieczeństwo użycia, 31

- cecha oprogramowania, 28
- cel dokumentu, 14
- charakterystyka użytkowników, 23
- Concept of Operations, 21
- czynniki zewnętrzne, 20
- czytelnicy, 14

- diagram czynności, 58
- diagram klas, 55
- diagram kontekstu, 16
- diagram przypadków użycia, 46
- diagram stanu, 57
- dokładność, 34
- dokumentacja użytkownika, 19

- dziedziczenie, 56

- efektywność, 31

- funkcjonalność, 34

- inter-operacyjność, 35
- interfejs, 29, 33
- interfejs użytkownika, 29

- klasa UML, 55
- kompozycja, 57

- literatura, 16

- model danych, 25
- moduł funkcjonalny, 28

- niezawodność, 37

- obiekty biznesowe, 24
- odpowiedniość, 34
- odtwarzalność, 37
- ograniczenia implementacyjne, 18
- ograniczenia projektowe, 18

- poziom biznesowy, 45
- poziom podfunkcji, 45
- poziom użytkownika, 45
- priorytet, 9
- priorytety wymagań, 28
- proces biznesowy, 21
- procesy biznesowe, 25, 53
- produktywność, 31
- profil UML, 62
- przenośność, 41
- przypadki użycia, 28, 45

relacja rozszerzania, 52
relacja specjalizacji, 23, 53
relacja zawierania, 51
rozszerzenia, 49

słownik danych, 25
satysfakcja, 32
scenariusz, 48
scenariusz alternatywny, 49
stabilność, 40

tolerancja uszkodzeń, 37

ukryte założenia, 20
uml, 55

współistnienie, 42
wydajność, 38
wyjątki, 49
wykorzystanie zasobów, 39
wymagania funkcjonalne, 4, 28
wymagania pozafunkcjonalne, 4, 33

założenia, 20
zakres produktu, 15
zależność, 55
zgodność funkcjonalności, 36
zrozumiałość, 30