UML. Wprowadzenie

Autor: Sinan Si Alhir
Tłumaczenie: Adam Jarczyk
ISBN: 83-7361-327-7
Tytuł oryginału: Learning UML
Format: B5, stron: 252

W książce „UML. Wprowadzenie" Sinan Si Alhir przedstawia UML i jego znaczenie, a następnie prowadzi w kierunku mistrzowskiego opanowania języka. Najpierw dowiesz się, jak UML wykorzystywany jest do modelowania struktury systemu. W rozdziale poświęconym diagramom klas i diagramom obiektów przedstawiono wiele pojęć związanych z UML-em: ogólnych (klasy) i szczegółowych (obiekty). Następnie dowiesz się, jak za pomocą diagramów przypadków użycia modelować funkcjonalność systemu. Na koniec zobaczysz, w jaki sposób za pomocą diagramów komponentów i wdrażania modeluje się sposób wdrożenia systemu w środowisku fizycznym.

Nauczysz się, jak posługiwać się diagramami sekwencji i kolaboracji, jak modelować interakcje pomiędzy składnikami systemu, jak za pomocą diagramów stanów opisywać cykle życiowe składników systemu i jak dokumentować czynności przepływów sterowania i zakresy odpowiedzialności.

Od pierwszej do ostatniej strony książki Sinan Si Alhir koncentruje się na UML-u jako języku i unika zaplątania się w metodologii. Jego wywody są jasne i zwięzłe. Każdy rozdział kończy się zestawem ćwiczeń, które pozwolą Ci sprawdzić Twoją coraz większą znajomość języka UML. Pod koniec książki (a nawet wcześniej), powinieneś zauważyć swoją rosnącą sympatię do prostego, acz wyrazistego języka, jakim jest UML i zacząć stosować go do efektywnego i profesjonalnego przekazywania wszelkich aspektów projektowania systemów.

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

1

Wprowadzenie

W tym rozdziale przedstawię język UML (ang. *Unified Modeling Language* — ujednolicony język modelowania). Omówię powody, dla których UML jest ważny, i jak można się go nauczyć, koncentrując się na paradygmacie obiektowym, technikach modelowania strukturalnego i behawioralnego oraz innych możliwościach UML-a. Powodów do nauczenia się i korzystania z języka UML jest wiele. Krótko mówiąc, UML jest *lingua franca* systemów informacyjnych i branż technologicznych. Bardziej formalnie, UML jest językiem ogólnego zastosowania, a zarazem standardem branżowym o szerokich zastosowaniach, powszechnie obsługiwanym przez narzędzia obecne na rynku.

Konstruowanie systemów polega na tworzeniu ich zgodnie z wymaganiami, z zastosowaniem przy ich rozwoju procesu cyklu życia. Wymagania są zasadniczo problemami do rozwiązania, system jest rozwiązaniem tych problemów, zaś konstruowanie systemu jest procesem rozwiązywania problemów, w skład którego wchodzą: rozpoznanie problemu, rozwiązanie problemu i implementacja rozwiązania. Do opisywania wymogów służą języki naturalne. Języki programowania, a w szerszym kontekście — oparte na technologii języki implementacji, np. XML, SQL, Java, C# itp., służą do przekazywania (opisywania) szczegółów systemu. Ponieważ języki naturalne są mniej precyzyjne od języków programowania, w procesie rozwiązywania problemów do przekraczania przepaści pomiędzy wymogami i systemem służą języki modelowania, takie jak UML.

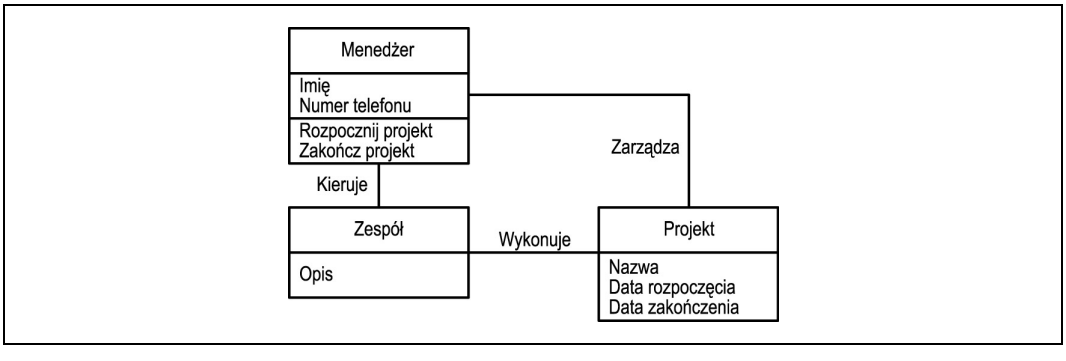
Język ogólnego zastosowania, np. UML, może być stosowany w całym procesie tworzenia systemu, od gromadzenia wymogów, aż po implementację systemu. Ponieważ UML jest językiem o szerokim zakresie zastosowań, można go używać w różnych typach systemów, dziedzin i procesów. Możemy dzięki temu użyć UML-a do opisu systemów programowych i nieprogramowych (tzw. systemów biznesowych) w różnych dziedzinach i branżach, np. w produkcji, bankowości, handlu elektronicznym itd. Co więcej, możemy zastosować UML do dowolnego procesu lub metody rozwiązania. Język ten obsługuje wielu producentów narzędzi, które są standardami branżowymi; nie jest to zastrzeżony lub zamknięty język modelowania.

Co to jest UML?

UML jest po prostu językiem wizualnym, służącym do modelowania i opisywania systemów za pomocą diagramów i dodatkowego tekstu. Na przykład, rysunek 1.1 przekazuje następujące informacje:

- Menedżer przewodzi zespołowi, który wykonuje projekt.
- Każdy menedżer ma imię i numer telefonu, i może zainicjować lub zakończyć (przerwać) projekt.
- Każdy projekt ma nazwę, datę rozpoczęcia i datę ukończenia.
- Każdy zespół ma opis, i tylko to nas w nim interesuje.

Proszę się nie przejmować, jeśli rysunek 1.1 jest nie do końca zrozumiały. Zajmiemy się tym rysunkiem i UML-em bardziej szczegółowo w rozdziale 2.



Rysunek 1.1. Menedżerowie, projekty i zespoły

Trzy aspekty UML-a

Jak już wiemy, UML jest skrótem od Unified Modeling Language (ujednolicony język modelowania). Każde z tych trzech słów mówi o innym ważnym aspekcie UML-a. W kilku następnych podpunktach omówimy te aspekty, rozwiązując skrót od końca.

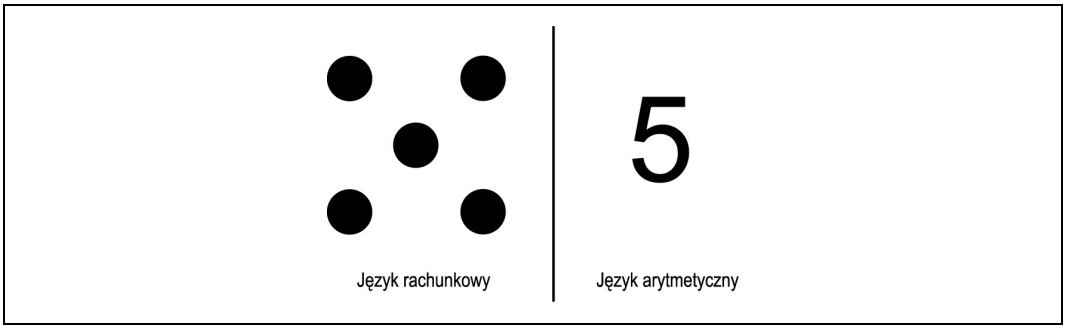
Language (język)

Język pozwala nam porozumiewać się na temat danego podmiotu. W konstruowaniu systemu podmiotem może być wymóg lub system. Bez języka trudno o komunikację pomiędzy członkami zespołu i o współpracę, pozwalającą z powodzeniem opracować system.

Języki w ogólnym znaczeniu nie zawsze składają się z zapisanych wyrazów. Na przykład, powszechnie używamy „języka rachunkowego", aby uczyć dzieci liczenia i arytmetyki. Dziecko otrzymuje ileś jabłek, pomarańczy, kamyków lub innych obiektów, które reprezentują sobą liczbę. Dla dziecka przedstawieniem liczby 5 może być pięć jabłek. Działania

dodawania i odejmowania sa reprezentowane przez fizyczną czynność dodawania lub zabierania obiektów ze zbioru dziecka. My, dorośli, preferujemy zamiast tego język arytmetyki, w którym określoną liczbę reprezentuje łańcuch cyfr arabskich, a dodawanie i odejmowanie reprezentowane są przez operatory + i −.

Rysunek 1.2 zestawia język rachunkowy dziecka z bardziej abstrakcyjnym językiem arytmetycznym, używanym przez dorosłych.



Rysunek 1.2. Wartość pięć w dwóch „językach"

Weźmy teraz możliwość przedstawienia w tych dwóch językach określonej liczby dni w projekcie. Do zamodelowania i wyrażenia wartości „pięć" język rachunkowy używa pięciu obiektów, zaś język arytmetyczny łańcucha „5". Do modelowania i wyrażania większych wartości, np. 365, język rachunkowy wymaga 365 obiektów (co może być niepraktyczne), zaś język arytmetyczny stosuje ciąg „365". Aby zamodelować i wyrazić wartość cztery i pół język rachunkowy wymaga czterech i pół obiektu (ponownie, pół obiektu niekoniecznie jest praktycznym rozwiązaniem), zaś arytmetyczny używa łańcucha „4,5". Ponieważ arytmetyka pozwala łatwiej i bardziej praktycznie niż język rachunkowy przedstawiać wartości z szerszego zakresu, mówimy, że język arytmetyczny jest bardziej *ekspresywny* od rachunkowego. Oprócz tego liczbę możemy wyrazić za pomocą arytmetyki bardziej ściśle niż w języku rachunkowym. Stosując bardziej ekspresywne języki, możemy przekazywać w sposób bardziej zwięzły złożone informacje o złożonych podmiotach.

Formalizując nieco, UML jest językiem służącym do specyfikacji, wizualizacji, konstrukcji i dokumentacji artefaktów *procesu zorientowanego na system*. Proces zorientowany na system oznacza podejście koncentrujące się na systemie, łącznie z etapami tworzenia i utrzymania systemu, zgodnie z wymogami, jakie ten system musi spełnić. Specyfikacja obejmuje tworzenie modelu opisującego system. W procesie wizualizacji model jest tworzony i opisywany za pomocą diagramów (model jest ideą, a diagramy wyrażeniem tej idei). Konstrukcja oznacza wykorzystanie tego wizualnego obrazu do zbudowania systemu, podobnie jak plany techniczne są używane przy wznoszeniu budynku. Dokumentacja wykorzystuje modele i diagramy do zarejestrowania naszej wiedzy o wymogach i systemie w obrębie całego procesu.

UML sam w sobie nie jest procesem. Proces polega na zastosowaniu zbioru kroków, opisanych przez *metodologię*, do rozwiązania problemu i stworzenia systemu spełniającego wymogi użytkownika. *Metoda* zajmuje się jedynie częścią procesu tworzenia systemu, na

przykład gromadzeniem wymogów, analizą, projektowaniem itp., natomiast metodologia dotyczy całego procesu, od gromadzenia wymogów aż do udostępnienia systemu użytkownikom. Różnorodne metody gromadzenia i wykorzystywania wymogów, analizowania wymogów, projektowania systemu itp. noszą nazwę *technik*. *Artefakty* (*wytwory*) są produktami pracy, tworzonymi i wykorzystywanymi w procesie; należy do nich również dokumentacja służąca do komunikacji pomiędzy stronami pracującymi nad systemem i samym fizycznym systemem. Wszystkie typy diagramów UML nazywane są również *technikami modelowania*.

Model

Model jest reprezentacją podmiotu. Na przykład, jak już wspomniano, do modelowania i wyrażenia wartości „pięć" język rachunkowy używa pięciu obiektów, zaś język arytmetyczny ciągu „5". Model rejestruje zbiór związanych z podmiotem idei, zwanych *abstrakcjami*. Bez modelu bardzo trudno jest osiągnąć porozumienie pomiędzy członkami zespołu na temat wymogów i systemu oraz analizować wpływ zmian wprowadzanych podczas tworzenia systemu.

Jeśli przy tworzeniu modelu będziemy próbowali przedstawić jednocześnie wszystkie informacje o danym podmiocie, z łatwością przytłoczy nas objętość informacji. Ważne jest więc skoncentrowanie się na rejestrowaniu liczących się informacji, niezbędnych do zrozumienia danego problemu, znalezienia rozwiązania tego problemu i zaimplementowania rozwiązania, a zarazem wykluczaniu wszelkich nieistotnych informacji, które mogą spowolnić nasze postępy. Decydując, które abstrakcje składają się na model, ustalając poziom ich szczegółowości i etapy, na których będą rejestrowane w procesie konstruowania systemu, możemy lepiej zapanować nad ogólną złożonością tworzenia systemu.

Unified (ujednolicony)

Pojęcie „ujednolicony" bierze się z faktu, iż Object Management Group (OMG) — organizacja tworząca standardy powszechnie przyjmowane w przemyśle — razem z Rational Software Corporation stworzyły UML w celu połączenia ze sobą najlepszych metod stosowanych w systemach informacyjnych i w branży technologicznej. Do praktyk tych zalicza się stosowanie technik, które pozwalają z większym powodzeniem konstruować systemy. Bez wspólnego języka, nowym członkom zespołów trudno jest szybko osiągać produktywność i wnosić swój wkład do tworzenia systemu.

Zadania i zakres

Poznanie zadań i zakresu stosowania UML, wytyczonych przez OMG, pozwoli nam lepiej zrozumieć motywy, które doprowadziły do powstania tego języka. Według założeń OMG, UML ma być:

- gotowy do użytku,
- ekspresywny,

- prosty,
- precyzyjny,
- rozszerzalny,
- niezależny od implementacji,
- niezależny od procesu.

Ponieważ UML jest gotowy do użytku, ekspresywny, prosty i precyzyjny, język ten można od razu zastosować w projekcie rozwojowym. Aby umożliwić tworzenie precyzyjnych modeli, organizacja OMG stworzyła język OCL (Object Constraint Language — język zawężania obiektów), który jest podjęzykiem służącym do dołączania warunków, jakie elementy modelu muszą spełnić, aby model można było uznać za poprawny (inaczej *well--formed* — poprawnie zbudowany). OCL został omówiony w rozdziale 10.

Język rozszerzalny pozwala definiować nowe pojęcia, co przypomina wprowadzanie nowych słów i rozszerzanie słownictwa języka naturalnego. Rozszerzalność została omówiona w rozdziale 9. Język niezależny od implementacji może być używany niezależnie od konkretnej technologii implementacji, takiej jak Java lub .NET. Język niezależny od procesu może być stosowany do różnych typów procesów.

Zdefiniowany przez OMG podczas tworzenia UML zakres stosowania tego języka obejmował połączenie trzech najbardziej znaczących metod konstruowania systemów: metody *Booch '93* Grady'ego Boocha, metody *Object Modeling Technique (OMT) -2* Jamesa Rumbaugha oraz metody *OODE* (ang. *Object-Oriented Software Engineering*) Ivara Jacobsona, z zalecanymi praktykami inżynierskimi dla systemów informatycznych i branży technologicznej. Oddzielnie są one jedynie metodami, lecz razem tworzą dość kompletną metodologię.

Historia

W historii UML-a możemy wyróżnić pięć okresów. Poznanie tych okresów pozwoli nam zrozumieć, z jakich powodów powstał UML i jak wciąż ewoluuje.

Okres fragmentacji

Od połowy lat 70. do połowy lat 90. organizacje zaczęły zdawać sobie sprawę z tego, jak cenne jest oprogramowanie dla biznesu, lecz dysponowały tylko niepełnymi zbiorami technik tworzenia i utrzymania oprogramowania. Spośród różnych powstających wówczas technik i metod, koncentrujących się na bardziej wydajnym tworzeniu i utrzymaniu oprogramowania (każda miała własne języki modelowania), wyróżniały się trzy:

- Metoda Grady'ego Boocha *Booch '93* (powstała z *Booch '91*) kładła nacisk na projektowanie i tworzenie systemów oprogramowania.
- Metoda Jamesa Rumbaugha OMT-2 (ang. *Object Modeling Technique* — technika modelowania obiektów, powstała z OMT-1) kładła nacisk na analizę systemów oprogramowania.

- Metoda Ivara Jacobsona *OODE* (ang. *Object-Oriented Software Engineering* — obiektowa technika programowania) kładła nacisk na modelowanie biznesowe i analizę wymagań.

W miarę ewoluowania metod strukturalnych w kierunku metod obiektowych branża podzieliła się, na zwolenników tych trzech (i innych) metod. Wyznawcy jednej metody mieli kłopoty ze zrozumieniem produktów entuzjastów innych metod. Oprócz tego mieli problemy z przenoszeniem się z jednej organizacji do drugiej, ponieważ taki ruch często oznaczał konieczność nauczenia się nowej metody. Co więcej, obsługa przez narzędzia wahała się od zerowej do minimalnej, ponieważ metod było tak wiele. Z tych powodów zbyt wysokie koszty uniemożliwiały często użycie jakiejkolwiek metody.

Okres jednoczenia

Od połowy lat 90. do roku 1997 wyłonił się język UML 1.0. James Rumbaugh, a po nim Ivar Jacobson dołączyli do Grady'ego Boocha w firmie Rational Software Corporation, aby połączyć swoje metody podejścia. Wysiłki na rzecz unifikacji przyniosły im określenie „Trzej Amigo". Gdy organizacje zaczęły doceniać wartość UML-a, zespół Object Analysis and Design Task Force z OMG wydał dokument RFP (ang. *Request for Proposal*) w celu utworzenia standardu, definiującego znaczenia pojęć technologii obiektowej dla narzędzi, które obsługują analizę i projektowanie zorientowane obiektowo. Razem z różnymi innymi organizacjami firma Rational Software Corporation utworzyła UML Partners Consortium i partnerzy ci zgłosili do OMG wersję 1.0 UML-a jako jedną z wielu wstępnych odpowiedzi na dokument RFP.

Okres standaryzacji

W drugiej połowie 1997 roku pojawił się UML 1.1. Wszystkie odpowiedzi na RFP włączono w tę wersję UML-a. W listopadzie 1997 organizacja OMG zaadoptowała UML i wzięła na siebie odpowiedzialność za dalszy rozwój standardu.

Okres korekt

Po przyjęciu UML-a 1.1 pojawiły się różne wersję języka UML. OMG wyznaczyła grupę zajmującą się korektami (RTF — ang. *revision task force*), której zadaniem było przyjmowanie publicznych komentarzy dotyczących UML-a i dokonywanie pomniejszych zmian redakcyjnych i technicznych w standardzie. Wielu różnych dostawców produktów i usług zaczęło wspierać i promować UML poprzez narzędzia, usługi doradcze, książki itp. Aktualną wersją UML-a jest 1.4, a OMG pracuje obecnie nad wersją UML 2.0.

Okres wdrożenia

Równolegle z korektami OMG zgłasza standard UML do przyjęcia jako standard międzynarodowy poprzez organizację ISO (ang. *Organization for Standardization*) w formie dostępnej publicznie specyfikacji — PAS (ang. *Publicly Available Specification*). Najbardziej aktualna wersja specyfikacji UML jest dostępna w serwisie OMG *http://www.omg.org*.

UML i proces

Mimo że UML jest niezależny od procesu, jego autorzy promują proces, który jest sterowany przypadkami użycia, skoncentrowany na architekturze, iteracyjny i przyrostowy. Poznając relacje pomiędzy UML-em i procesem oraz typem procesu promowanym przez autorów UML-a, możemy lepiej zrozumieć, jakie podejście do nauki UML-a będzie najlepsze. Mimo to z języka UML może korzystać proces dowolnego typu, nawet nie posiadający tych cech.

Ogólnie mówiąc, proces cyklu życia rozwoju każdego systemu obejmuje następujące czynności:

- Gromadzenie wymogów definiujących, co system powinien robić.
- Analizę pozwalającą zrozumieć wymogi.
- Projektowanie — ustalenie, w jaki sposób system będzie spełniał narzucone wymogi.
- Implementację — budowanie systemu.
- Testowanie — weryfikacja, czy system spełnia wymogi.
- Wdrożenie, udostępniające system użytkownikom.

Do wykonania tych czynności w celu stworzenia systemu można podchodzić na wiele sposobów. Tradycyjnie stosowana była metoda kaskadowa (inaczej wodospadowa). W chwili obecnej częściej spotyka się podejście iteracyjne.

Stosowanie metody kaskadowej

Przy stosowaniu *metody kaskadowej* (ang. *waterfall approach*) czynności związane z cyklem życia systemu wykonywane są w pojedynczej, liniowej sekwencji dla wszystkich wymogów. Prowadzi to często do odkrywania podczas testów, gdy integrowane są poszczególne fragmenty systemu, problemów związanych z jakością, które pozostawały w ukryciu podczas czynności projektowania i implementacji. Ponieważ problemy takie są odkrywane na późnych etapach procesu rozwoju, może być za późno na ich rozwiązanie lub koszty rozwiązania mogą być zbyt wysokie. Na przykład odkrycie, iż dany system zarządzania bazą danych ma za małą wydajność dla korzystających z niego aplikacji, gdy aplikacje zostały już opracowane, stanowi kolosalny problem.

Rozważmy projekt obejmujący 10 wymogów, na przykład generowanie 10 różnych typów raportów, z których każdy bierze się z innego wymogu. Przy podejściu kaskadowym wszystkie wymogi są rejestrowane i analizowane, a cały system jest projektowany, implementowany, testowany i wdrażany w jednej liniowej sekwencji. W takim przypadku UML może z łatwością posłużyć do przekazywania wymogów i opisu systemu. Ponieważ jednak czynności wykonywane są dla wszystkich wymogów w jednej liniowej sekwencji, modele UML na każdym kolejnym kroku muszą być dość kompletne. Taki poziom kompletności często trudno jest zmierzyć lub osiągnąć, ponieważ wprawdzie UML jest bardziej

precyzyjny od języków naturalnych, lecz zarazem mniej precyzyjny od języków programowania. Zamiast więc koncentrować się na systemie, zespoły korzystające z UML-a przy podejściu kaskadowym poświęcają swój czas na ustalenie, czy ich modele UML są wystarczająco kompletne.

Stosowanie metody iteracyjnej

Gdy stosujemy *metodę iteracyjną*, wszystkie podzbiory czynności w cyklu życia są wykonywane kilkakrotnie, aby lepiej zrozumieć wymogi i stopniowo opracować bardziej niezawodny system. Każde przejście cyklu tych działań lub ich podzbioru nosi nazwę *iteracji*, a seria iteracji w końcu krok po kroku doprowadza do ostatecznego systemu. Pozwala to lepiej zrozumieć wymogi i stopniowo stworzyć bardziej odpowiedni system przez kolejne ulepszenia i przyrostowe zwiększanie szczegółowości w miarę kolejnych iteracji. Na przykład, możemy zbadać wydajność określonego systemu zarządzania bazami danych i odkryć, iż będzie niewystarczająca dla korzystających z niego aplikacji, jeszcze zanim aplikacje te zostaną w pełni skonstruowane. Pozwoli to wprowadzić odpowiednie zmiany w aplikacjach lub zbadać inny system zarządzania bazą danych, zanim będzie na to za późno lub stanie się to zbyt kosztowne.

Rozważmy projekt, który obejmuje generowanie 10 różnych typów raportów. W podejściu iteracyjnym możliwy jest następujący ciąg iteracji:

1. Identyfikujemy pięć wymogów (nazwanych od W1 do W5) i analizujemy trzy z nich (np. W1, W3 i W5).
2. Rejestrujemy pięć kolejnych wymogów (nazwanych od W6 do W10), analizujemy dwa nie przeanalizowane w poprzedniej iteracji (W2 i W4) oraz projektujemy, implementujemy i testujemy system, który spełnia trzy wymagania przeanalizowane w poprzedniej iteracji (W1, W3 i W5) oraz dwa przeanalizowane w tej iteracji (W2 i W4), lecz nie wdrażamy systemu, ponieważ w bieżącej iteracji nie przeznaczyliśmy na tę czynność wystarczająco dużo czasu).
3. Wdrażamy system spełniający pięć wymogów przetestowanych w poprzedniej iteracji (od W1 do W5) i kontynuujemy pracę nad pozostałymi (od W6 do W10).
4. Dalej pracujemy nad systemem, lecz musimy zająć się zmianami w jednym z wdrożonych już wymogów (np. W3), zmianami w innych wymogach, które nie zostały jeszcze wdrożone (np. W6 i W10) oraz innymi technicznymi zmianami w systemie.

Ten ciąg iteracji może sprawiać wrażenie chaotycznego, jednakże podejście iteracyjne jest tylko ideą, a UML tylko językiem, więc do zastosowania UML-a w rzeczywistym projekcie niezbędna jest metodologia. Iteracje wykorzystywane przez metodologię nie są chaotyczne, lecz zorganizowane i dość dynamiczne w ramach kontekstu tej metodologii.

Podejście iteracyjne do konstruowania systemu przynosi następujące korzyści:

- Możemy lepiej radzić sobie ze złożonością, budując system małymi przyrostowymi porcjami, a nie cały od razu.
- Możemy lepiej radzić sobie ze zmianami w wymogach, rozkładając zmiany na cały proces, a nie usiłując zarejestrować i wprowadzić wszystkie zmiany na raz.
- Możemy udostępniać użytkownikom fragmentaryczne rozwiązania w trakcie procesu, a nie kazać im czekać na ukończenie procesu, kiedy otrzymaliby kompletny system i być może stwierdzili, że nie tego oczekiwali.
- Możemy zwrócić się do użytkowników z prośbą o uwagi dotyczące opracowanych już części systemu, co pozwoli wprowadzać zmiany i tak kierować postępami w pracach, by stworzyć bardziej solidny system spełniający potrzeby użytkowników.

Proces iteracyjny jest przyrostowy, ponieważ nie przerabiamy jedynie raz za razem tych samych wymogów w kolejnych iteracjach, lecz zajmujemy się w nich coraz większą liczbą wymogów. Oprócz tego w jednej iteracji mogą odbywać się *równoległe* czynności, jeśli koncentrują się na różnych częściach systemu i nie kolidują ze sobą. Wobec tego, chociaż podejście to określane jest często iteracyjnym i przyrostowym, w rzeczywistości jest iteracyjne, przyrostowe i równoległe. Ponieważ w tej metodzie system tworzony jest przez kolejne udoskonalenia i zwiększaną przyrostowo liczbę szczegółów, łatwiej jest nam ustalić wystarczający poziom kompletności modelu UML niż w podejściu kaskadowym. Na przykład, jeśli mamy pytanie lub problem, którym należy się zająć, i jeśli nie jesteśmy w stanie użyć do tego posiadanego modelu UML, to być może musimy rozbudować model; w przeciwnym razie możemy iść dalej bez poświęcania dodatkowych nakładów czasu i pracy na rozwój modeli UML.

Jak można organizować i motywować działania w celu spełnienia wymogów przy takim dynamicznym podejściu, w którym iteracje odbywają się równolegle a system jest tworzony przyrostowo? Jak skupić się na systemie i uniknąć tworzenia systemu, który będzie trudny do utrzymania i rozbudowy, ponieważ będzie jedynie zbiorem elementów posklejanych ze sobą bez obejmującego je schematu? Którymi wymogami mamy zająć się na początku, i które części systemu implementować w pierwszej kolejności? Odpowiedzi na te pytania są elementami podejścia iteracyjnego, w których przypadki użycia, architektura i zarządzanie ryzykiem są krytyczne.

Przypadki użycia

Przypadek użycia (ang. *use case*) jest wymogiem funkcjonalnym opisanym z perspektywy użytkowników systemu. Na przykład, do wymogów funkcjonalnych w większości systemów zalicza się funkcjonalność zabezpieczeń, pozwalająca użytkownikom logować się do systemu i wylogowywać, wprowadzać i przetwarzać dane, generować raporty itd. Przypadki użycia są tematem rozdziału 4.

Proces kierowany przypadkami użycia to taki proces, w którym możemy wykorzystać przypadki użycia do planowania i przeprowadzania iteracji. Pozwala nam to zorganizować działania i skoncentrować się na implementowaniu wymogów wobec systemu.

Inaczej mówiąc, rejestrujemy i analizujemy przypadki użycia, projektujemy i implementujemy system spełniający ich potrzeby, testujemy i wdrażamy system oraz planujemy następne iteracje. Wobec tego przypadki użycia wiążą ze sobą wszystkie czynności w danej iteracji.

Architektura

Architektura obejmuje elementy składające się na system i sposób, w jaki współpracują one ze sobą w celu zapewnienia wymaganej funkcjonalności systemu. Na przykład, większość systemów zawiera elementy obsługujące funkcjonalność zabezpieczeń, wprowadzania i przetwarzania danych, generowania raportów itp. Elementy i relacje pomiędzy nimi noszą nazwę *struktury* systemu. Modelowanie struktury systemu nosi nazwę *modelowania strukturalnego* i jest tematem części II niniejszej książki. Elementy, ich interakcje i współpraca noszą nazwę *zachowania* systemu. Modelowanie zachowań systemu nosi nazwę *modelowania behawioralnego* i jest tematem części III niniejszej książki. Różne typy elementów składających się na architekturę systemu, zarówno na strukturę, jak i zachowanie, są ustalane przez paradygmat obiektowy. Zasady i pojęcia paradygmatu obiektowego będą tematem rozdziału 2.

Proces *architekturo-centryczny* koncentruje się w kolejnych iteracjach na architekturze systemu. Dzięki temu możemy z większą pewnością zagwarantować, że otrzymany system nie będzie mieszaniną elementów, trudną do zintegrowania, utrzymania i ulepszania. Oznacza to, że architektura wiąże ze sobą wszystkie elementy systemu podczas jego konstruowania w kolejnych iteracjach.

Ryzyko

Ryzyko oznacza dowolną przeszkodę lub niewiadomą, która może zniweczyć nasze wysiłki. Na przykład, ryzykiem przy konstruowaniu systemu mogą być niewystarczające fundusze, nie przeszkoleni członkowie zespołu odgrywający krytyczną rolę lub niestabilne technologie.

Aby ustalić, jakie przypadki użycia powinny motywować daną iterację, i na których elementach architektury mamy się w danej iteracji skoncentrować, najpierw identyfikujemy zagrożenia dotyczące projektu. Następnie zajmujemy się tymi przypadkami użycia, które związane są z najwyższym ryzykiem i tymi elementami architektury, które po skonstruowaniu rozwiążą najpoważniejsze zagrożenia. Podejście takie często nazywane jest *konfrontowaniem ryzyka*.

Wróćmy jeszcze raz do projektu, w którym generowanych jest 10 typów raportów. Powiedzmy, że trzy z nich (np. W1, W3 i W5) wymagają znaczącego dostępu do bazy danych, a cztery (np. W3, W6, W8 i W10) wymagają znaczącego wkładu użytkownika. Ryzyka mogą być np. dwa: brak wystarczająco intuicyjnego interfejsu użytkownika (o nazwie R1) i niewystarczająco wydajny system zarządzania bazą danych (nazwane R2). Z powyższego opisu wiemy, że W1, W3 i W5 są związane z ryzykiem R1, a W3, W6, W8 i W10 z ryzykiem R2. Jeśli ryzyko R1 jest bardziej krytyczne dla naszego projektu i ma większe

prawdopodobieństwo wystąpienia lub poważniejszy wpływ na projekt, w miarę możliwości powinniśmy najpierw zająć się wymogami W1, W3 i W5 lub możliwie największą liczbą z nich, ponieważ spotykają się z ryzykiem R1. Jeśli ryzyko R2 jest dla naszego projektu bardziej krytyczne i ma większe prawdopodobieństwo wystąpienia lub poważniejszy wpływ na projekt, w miarę możliwości powinniśmy najpierw zająć się wymogami W3, W6, W8 i W10 lub możliwie największą liczbą z nich, ponieważ spotykają się z ryzykiem R2. Jednakże w obu przypadkach powinniśmy zacząć od W3, ponieważ wiąże się z obydwoma ryzykami.

UML udostępnia techniki modelowania strukturalnego i behawioralnego, które mogą być używane w procesie krokowym sterowanym przez wymogi, koncentrującym się na stworzeniu systemu o zdrowej architekturze spełniającej nasze wymogi i pozwalającym na konfrontacje z ryzykami w całym procesie konstruowania systemu.

Nauka UML-a

Proces nauki UML-a może być przytłaczający, biorąc pod uwagę zakres i głębię tego języka oraz brak procesu, jeśli nie wiemy, na której części UML-a mamy się skoncentrować. Lecz jeśli rozumiemy relacje pomiędzy UML-em i procesem, to wiemy, że należy skupić się na:

- Paradygmacie obiektowym, ponieważ tworzy podstawy języka UML.
- Modelowaniu strukturalnym i behawioralnym, ponieważ pozwalają zrozumieć wymogi i architekturę.
- Innych możliwościach UML-a.

Oprócz tego przy nauce UML-a ważne jest, by skoncentrować się na podstawach i zrozumieć, jak skutecznie i z powodzeniem stosować UML do modelowania systemów, zamiast ugrzązć w nauce każdego z możliwych aspektów języka.

W całej książce będę posługiwał się praktycznym przykładem systemu zarządzania projektami, który pomoże Czytelnikowi w nauce odczytywania, pisania oraz skutecznego stosowania języka UML. Celem nie jest tu stworzenie pełnego lub szczegółowego modelu, na podstawie którego będzie można zaimplementować system, lecz zapoznanie się z praktycznym przykładem i nauka, jak efektywnie i z powodzeniem wykorzystać UML do komunikacji podczas konstrukcji rzeczywistego systemu. Do rozdziałów dołączyłem ćwiczenia, które pozwolą na trening i sprawdzenie umiejętności Czytelnika.

Ogólnie mówiąc, system zarządzania projektami opisywany w przykładzie zapewnia funkcjonalność zarządzania projektami i zasobami oraz samym systemem. Zdefiniowane są w nim następujące role:

Menedżer projektu

Odpowiedzialny za zapewnienie, by projekt dał produkt o odpowiedniej jakości, w ramach ustalonego czasu, kosztu i zasobów.

Menedżer zasobów

Odpowiedzialny za udostępnienie dla projektu wykwalifikowanych i wyszkolonych zasobów ludzkich.

Zasób ludzki

Odpowiedzialny za utrzymanie kwalifikacji i wykonanie dla projektu prac o odpowiedniej jakości.

Administrator systemu

Odpowiedzialny za udostępnienie dla projektu systemu zarządzania projektem.

Dalsze szczegóły przykładu praktycznego będą wprowadzane w stosownych miejscach w pozostałej części książki. Rozmyślnie nie podaję wszystkich szczegółów na raz, dzięki czemu Czytelnik będzie mógł zobaczyć, jak różne informacje są używane w różnych technikach modelowania UML-a. To z kolei pomoże zrozumieć, kiedy należy poszukiwać takich szczegółów i dobrze ilustruje sposób, w jaki podejście iteracyjne pozwala stopniowo gromadzić informacje w miarę tworzenia projektu.

Zamiast wprowadzać jakąś naciąganą pseudometodologię lub proces, skoncentruję się na sposobach wykorzystania różnorodnych diagramów UML-a i ich elementów zależnie od tego, co każdy z diagramów ma przekazać i na co kładzie nacisk. Pozwoli to nam skupić się na tym, jak poszczególne elementy języka pasują do siebie, a nie traktować UML-a jako zbioru różnorodnych typów diagramów, których nie łączy żaden wspólny schemat, dzięki czemu Czytelnik będzie mógł skuteczniej i z większym powodzeniem stosować język UML. Rozdział 2. koncentruje się na paradygmacie obiektowym i zależnościach pomiędzy różnymi elementami języka. Część II skupia się na wymogach funkcjonalnych i modelowaniu struktury systemu zarządzania projektami za pomocą technik modelowania strukturalnego UML-a. Część III koncentruje się na modelowaniu zachowań systemu za pomocą technik modelowania behawioralnego UML-a. Część IV przedstawia pewne inne możliwości UML-a, w tym OCL i mechanizmy rozszerzania języka.