

IDŹ DO

PRZYKŁADOWY ROZDZIAŁ



SPIS TREŚCI

KATALOG KSIĄŻEK

KATALOG ONLINE

ZAMÓW DRUKOWANY KATALOG

TWÓJ KOSZYK

DODAJ DO KOSZYKA

CENNIK I INFORMACJE

ZAMÓW INFORMACJE
O NOWOŚCIACH

ZAMÓW CENNIK

CZYTELNIA

FRAGMENTY KSIĄŻEK ONLINE

UML dla każdego

Autor: Joseph Schmuller

Tłumaczenie: Krzysztof Maślowski

ISBN: 83-7361-107-X

Tytuł oryginału: [Teach Yourself UML in 24 Hours](#)

Format: B5, stron: 372



UML jest jednym z najbardziej ekscytujących narzędzi do tworzenia obiektowo zorientowanych systemów. Jest to język modelowania wizualnego, pozwalający budowniczym systemów na tworzenie planów, w których ich wizje zostaną uchwycone i wyrażone w standardowy, łatwy do zrozumienia sposób. UML dostarcza też mechanizmy ułatwiające efektywną wymianę informacji i przekazywanie projektów innym.

Podczas 24 sesji zawartych w książce „UML dla każdego” nauczysz się tworzenia diagramów za pomocą zunifikowanego języka modelowania. Proste wyjaśnienia i metoda prowadzenia za rękę krok po kroku w każdym rozdziale, pozwalają na poznanie podstaw tego języka i zrozumienie jego zastosowań.

Poznasz:

- Podstawy projektowania obiektowego, związki UML-a z projektowaniem obiektowym
- Powiązania, agregacje, agregacje całkowite, interfejsy i realizacje
- Przypadki użycia i diagramy przypadków użycia
- Diagramy: stanów, przebiegu, kooperacji, czynności, komponentów, wdrożenia
- Tworzenie systemów za pomocą UML-a
- Studium przypadku, pokazujące kolejne etapy tworzenia złożonego systemu
- Przyszłość i rozszerzenia języka UML

Autor Joseph Schmuller, ekspert w tej dziedzinie, tłumaczy UML-a za pomocą odpowiednio dobranych i interesujących przykładów. Dzięki tej książce w szybkim tempie opanujesz sztukę operowania językiem UML i przekonasz się, jak bardzo porządkuje on proces projektowania oprogramowania.



Spis treści

O Autorze	15
Wstęp	17
Część I Zaczynamy	19
Rozdział 1. Co to jest UML	21
Dodanie metody do szkieletu	22
Jak się narodził UML	23
Komponenty UML-a	24
Diagram klas	24
Diagram obiektów	25
Diagram przypadków użycia	26
Diagram stanów	26
Diagram przebiegu	27
Diagram czynności	27
Diagram kooperacji	28
Diagram komponentów	29
Diagram wdrożenia	29
Kilka innych składników	29
Pakiety	29
Notatki	30
Stereotypy	30
Po co tyle różnych diagramów?	31
Podsumowanie	31
Warsztaty	31
Test	32
Ćwiczenia	32
Rozdział 2. Co to jest obiektowość	33
Obiekty, obiekty, wszędzie obiekty	34
Kilka pojęć	35
Abstrakcja	35
Dziedziczenie	36
Polimorfizm	37
Kapsułkowanie (hermetyzacja)	38
Wysyłanie komunikatów	39
Powiązania	39
Agregacja	41

Korzyści	43
Podsumowanie	43
Warsztaty.....	44
Test	44
Rozdział 3. UML i obiektowość	45
Wizualizacja klas	45
Atrybuty	46
Operacje	47
Atrybuty, operacje i wizualizacja.....	48
Zobowiązania i ograniczenia	50
Dołączone notatki	51
Klasy — do czego służą i gdzie ich szukać	51
Podsumowanie	53
Warsztaty.....	54
Test	54
Ćwiczenia	54
Rozdział 4. Związki.....	55
Powiązania.....	55
Ograniczenia powiązań.....	57
Klasy powiązań.....	57
Wiązania	58
Liczebność	58
Powiązania kwalifikowane.....	60
Powiązanie zwrotne	60
Dziedziczenie i uogólnienie	61
Poznanie dziedziczenia.....	62
Klasy abstrakcyjne	62
Zależności.....	63
Podsumowanie	64
Warsztaty.....	64
Test	65
Ćwiczenia	65
Rozdział 5. Agregacje, agregacje całkowite, interfejsy i realizacje	67
Agregacje.....	67
Ograniczenia agregacji	68
Agregacje całkowite.....	69
Otoczenia.....	69
Interfejsy i realizacje	71
Widoczność.....	72
Zasięg.....	73
Podsumowanie	73
Warsztaty.....	74
Test	74
Ćwiczenia	74
Rozdział 6. Informacje wstępne o przypadkach użycia.....	75
Przypadki użycia — co to takiego?	76
Przypadki użycia — dlaczego są ważne?.....	76
Przykład: automat do sprzedaży napojów gazowanych.....	77
Przypadek użycia „Kup napój”.....	77
Dodatkowy przypadek użycia	78

Zawieranie przypadków użycia	79
Rozszerzanie przypadków użycia	80
Rozpoczęcie analizy przypadków użycia	80
Podsumowanie	81
Warsztaty	81
Test	81
Ćwiczenia	82
Rozdział 7. Diagramy przypadków użycia	83
Prezentacja modelu przypadków użycia	84
Wracamy do przypadku automatu do sprzedaży napojów gazowanych	84
Śledzenie kroków scenariuszy	84
Związki między przypadkami użycia	86
Zawieranie	86
Rozszerzenie	86
Uogólnienie	88
Grupowanie	88
Stosowanie przypadków użycia w procesie analizy	89
Stosowanie modeli przypadków użycia — przykład	89
Poznanie domeny	89
Zrozumienie użytkowników	90
Zrozumienie przypadków użycia	90
Drażąc w głąb	91
Remanent rzeczy poznanych	93
Elementy strukturalne	94
Związki	94
Grupowanie	94
Przypisy	95
Rozszerzenie	95
I inne	95
Obraz ogólny	95
Podsumowanie	96
Warsztaty	96
Test	96
Ćwiczenia	97
Rozdział 8. Diagramy stanów	99
Diagram stanów	100
Zestaw symboli	100
Podawanie szczegółów w ikonie stanu	101
Dodawanie szczegółów transmisji — zdarzenia i akcje	101
Dodawanie szczegółów transmisji — warunki dozoru	103
Podstany	103
Podstany sekwencyjne	104
Podstany współbieżne	104
Stany wznowienia	105
Komunikaty i sygnały	106
Znaczenie diagramów stanów	107
Obraz ogólny UML-a	108
Podsumowanie	109
Warsztaty	109
Test	109
Ćwiczenia	110

Rozdział 9. Diagramy przebiegu.....	111
Co to jest diagram przebiegu?	111
Obiekty	112
Komunikaty.....	112
Czas	112
GUI.....	113
Kolejność	113
Pokazanie kolejności na diagramie przebiegu	114
Przypadek użycia	114
Automat do sprzedaży napojów — diagramy przebiegu: egzemplarzowy i ogólny	115
Egzemplarzowy diagram przebiegu	116
Ogólny diagram przebiegu.....	116
Tworzenie obiektów podczas przebiegu	119
Rekurencja.....	121
Obraz ogólny UML-a	121
Podsumowanie	121
Warsztaty.....	123
Test	123
Ćwiczenia	123
Rozdział 10. Diagramy kooperacji	125
Co to jest diagram kooperacji?	126
GUI.....	126
Zmiany stanu.....	127
Automat do sprzedaży napojów	128
Tworzenie obiektów.....	129
Kilka dodatkowych koncepcji	131
Obiekty wielokrotne	131
Zwracanie wyniku.....	131
Obiekty aktywne.....	132
Synchronizacja	132
Obraz ogólny UML-a	133
Podsumowanie	133
Warsztaty.....	134
Test	135
Ćwiczenia	135
Rozdział 11. Diagramy czynności	137
Co to jest diagram czynności?	137
Decyzje.....	138
Ścieżki współbieżne (rozwidłone).....	139
Sygnały.....	140
Stosowanie diagramów czynności	140
Operacja: wyrazy ciągu Fibonacciego	140
Proces: tworzenie dokumentu.....	141
Tory.....	143
Diagramy hybrydowe.....	143
Obraz ogólny UML-a	144
Podsumowanie	145
Warsztaty.....	146
Test	147
Ćwiczenia	147

Rozdział 12. Diagramy komponentów	149
Co to jest komponent?.....	149
Komponenty i interfejsy.....	150
Zastępstwo i wielokrotne użycie.....	151
Rodzaje komponentów.....	151
Co to jest diagram komponentów?.....	152
Reprezentacja komponentu.....	152
Sposoby przedstawiania interfejsów.....	153
Zastosowanie diagramów komponentów.....	154
Strona WWW z apletem Java.....	154
Strona WWW z kontrolkami ActiveX.....	155
PowerToys.....	156
Diagramy komponentów w ogólnym obrazie UML-a.....	157
Podsumowanie.....	158
Warsztaty.....	159
Test.....	159
Ćwiczenia.....	159
Rozdział 13. Diagramy wdrożenia	161
Czym jest diagram wdrożenia?.....	161
Stosowanie diagramów wdrożenia.....	163
Domowy system komputerowy.....	163
Sieć token-ring.....	164
Sieć ARCnet.....	164
Cienki Ethernet.....	166
Sieć bezprzewodowa Metricom Ricochet.....	167
Diagramy wdrożenia w ogólnym obrazie UML-a.....	168
Podsumowanie.....	168
Warsztaty.....	169
Test.....	169
Ćwiczenia.....	169
Rozdział 14. Podstawowe koncepcje UML-a	171
Struktura UML.....	172
Spojrzenie z bliska na warstwę metamodelu.....	173
Pakiet Podstawy.....	174
Pakiet Elementy zachowania.....	175
Pakiet Model zarządzania.....	176
Rozszerzanie UML-a.....	176
Stereotypy.....	177
Zależność.....	177
Klasyfikator.....	178
Klasa.....	178
Uogólnienie.....	178
Pakiety.....	179
Komponent.....	179
Kilka dodatkowych stereotypów.....	179
Stereotypy graficzne.....	179
Ograniczenia.....	180
Metki.....	181
Podsumowanie.....	181
Warsztaty.....	182
Test.....	182

Rozdział 15. UML jako narzędzie tworzenia systemów	183
Metodologie: stara i nowa.....	184
Stara metoda	184
Nowa metoda	185
Co trzeba zrobić podczas tworzenia oprogramowania?	185
GRAPPLE.....	187
RAD ³ : struktura GRAPPLE.....	187
Zbieranie wymagań.....	188
Analiza	191
Projekt.....	192
Budowa kodu	193
Wdrożenie.....	194
Spojrzenie wstecz na GRAPPLE.....	195
Podsumowanie	195
Warsztaty.....	196
Test.....	196
Część II Studium przypadku	197
Rozdział 16. Studium przypadku — informacje wstępne.....	199
GRAPPLE-owanie problemu.....	200
Poznanie procesów biznesowych.....	200
Obsługa klienta.....	201
Przygotowanie posiłku.....	206
Sprzątanie stołu.....	208
Czego się nauczyliście?.....	210
Podsumowanie	212
Warsztaty.....	212
Test	213
Ćwiczenia	213
Rozdział 17. Analiza domeny.....	215
Analiza wywiadu dotyczącego procesów biznesowych.....	215
Tworzenie wstępnego diagramu klas.....	217
Grupowanie klas	217
Tworzenie powiązań	219
Powiązania z klientem	219
Powiązania z kelnerem	223
Powiązania z szefem kuchni.....	225
Powiązania z sprzątaczem.....	225
Powiązania menadżera	225
Dygresja.....	226
Tworzenie agregacji i agregacji całkowitych	227
Wypełnianie klas	227
Klient.....	227
Pracownik.....	229
Rachunek	230
Ogólne zagadnienia modelowania	231
Słownik modelu.....	231
Organizacja diagramu.....	231
Przerobiony materiał	231
Podsumowanie	232
Warsztaty.....	232
Test	232
Ćwiczenia	233

Rozdział 18. Poznanie wymagań systemu	235
Tworzenie wizji.....	237
Zanim zaczniemy zbierać wymagania systemu	244
Sesja JAD poświęcona wymaganiom systemu	245
Rezultaty.....	248
Co dalej?.....	251
Podsumowanie	251
Warsztaty.....	251
Test	251
Ćwiczenie	251
Rozdział 19. Praca nad przypadkami użycia.....	253
Efektywne wykorzystanie przypadków użycia.....	254
Analiza przypadku użycia.....	254
Pakiet Kelner.....	255
Przyjmij zamówienie	256
Przełącz zamówienie do kuchni.....	256
Zmień zamówienie.....	257
Śledź stan realizacji zamówienia.....	258
Informuj szefa o statusie klientów.....	259
Wystaw rachunek	260
Drukuj rachunek	261
Wezwij pomocnika.....	262
Pozostałe przypadki użycia	263
Komponenty systemu.....	263
Podsumowanie	264
Warsztaty.....	264
Test	264
Ćwiczenia	264
Rozdział 20. Poznanie interakcji i zmian stanu	265
Robocze części systemu	265
Pakiet Kelner.....	266
Pakiet SzeFKuchni.....	266
Pakiet Sprzątacze	267
Pakiet Pomocnik Kelnera.....	267
Pakiet Pomocnik Szefa Kuchni.....	267
Pakiet Barman.....	267
Pakiet Szatniarz.....	268
Interakcje występujące w systemie	268
Przyjmij zamówienie.....	268
Zmień zamówienie.....	271
Śledź stan realizacji zamówienia.....	272
Implikacje	273
Podsumowanie	274
Warsztaty.....	274
Test	275
Ćwiczenia	275
Rozdział 21. Ogólne spojrzenie na projektowanie i wdrażanie.....	277
Kilka ogólnych zasad projektowania GUI	277
Sesja JAD poświęcona GUI.....	280
Od przypadków użycia do interfejsów użytkownika.....	281
Diagramy UML-owe projektowanych GUI.....	283

Planowanie wdrażania systemu	284
Sieć	284
Węzły i diagram wdrożenia	285
Następne kroki	286
...i kilka słów od zleceniodawcy	287
Zwiększanie zdolności sprzedaży	287
Ekspansja restauracyjnego imperium	288
Podsumowanie	289
Warsztaty	290
Test	290
Ćwiczenia	290
Rozdział 22. Wzorce projektowe	291
Parametryzacja	291
Wzorce projektowe	293
Łańcuch odpowiedzialności	294
Łańcuch odpowiedzialności: domena restauracji	295
Łańcuch odpowiedzialności: Modele zdarzeń przeglądarki sieciowej	297
Nasze własne wzorce projektowe	298
Korzyści ze stosowania wzorców	300
Podsumowanie	300
Warsztaty	301
Test	301
Ćwiczenia	301
Część III Spojrzenie w przyszłość	303
Rozdział 23. Modelowanie systemów wbudowanych	305
Potrzeba jest matką wynalazków	306
Narodziny PoprawiaczaChwytu	307
Czym jest system wbudowany?	309
Pojęcia dotyczące systemów wbudowanych	309
Czas	310
Wątki	310
Przerwania	310
System operacyjny	311
Modelowanie PoprawiaczaChwytu	313
Klasy	314
Przypadki użycia	315
Interakcje	316
Ogólne zmiany stanu	317
Wdrożenie	318
Poprawianie mięśni	318
Podsumowanie	319
Warsztaty	320
Test	320
Ćwiczenia	320
Rozdział 24. Co dalej z UML-em?	321
Rozszerzenia dla biznesu	321
Nauki płynące z rozszerzeń biznesowych	323
Interfejsy graficzne użytkownika	323
Połączenie z przypadkami użycia	323
Modelowanie GUI	324

Systemy ekspertowe	325
Komponenty systemu ekspertowego	325
Przykład	327
Modelowanie bazy wiedzy	328
Aplikacje sieci WWW	331
To już wszystko	333
Podsumowanie	333
Warsztaty	334
Test	334
Ćwiczenia	334
Dodatki	335
Dodatek A Rozwiązania testów	337
Dodatek B Narzędzia modelowania w UML-u	349
Cechy wspólne	349
Rational Rose	350
Select Enterprise	352
Visual UML	354
Idealne narzędzie do tworzenia modeli	355
Dodatek C Podsumowanie w rysunkach	357
Diagram czynności	357
Diagram klas	358
Diagram kooperacji	359
Diagram komponentów	360
Diagram wdrożenia	360
Diagram przebiegu	360
Diagram stanów	361
Diagram przypadków użycia	361
Skorowidz	363

Rozdział 1.

Co to jest UML

UML (ang. *Unified Modeling Language* — zunifikowany język modelowania) jest jednym z najbardziej ekscytujących narzędzi do tworzenia obiektowo zorientowanych systemów. Dlaczego? Ponieważ UML jest językiem modelowania wizualnego, pozwalającym budowniczym systemów na tworzenie planów, na których ich wizje zostają uchwycone i wyrażone w standardowy, łatwy do zrozumienia sposób. Dostarcza też mechanizmów ułatwiających efektywną wymianę informacji i przekazywanie projektów innym.

W tym rozdziale dowiemy się:

- ◆ dlaczego UML jest nieodzowny,
- ◆ jak powstał UML,
- ◆ czym są diagramy UML-a,
- ◆ dlaczego ważne jest stosowanie kilku rodzajów diagramów.

Najważniejsze jest przekazanie wizji innym. Przed pojawieniem się UML-a rozwój systemów był zawsze mniej lub bardziej udaną próbą trafienia do celu. Analitycy systemowi starali się ocenić potrzeby klientów, następnie tworzyli analizę wymagań i warunków zapisaną w notacji jasnej dla nich, ale nie zawsze zrozumiałej dla klientów. Przekazywali tę analizę zespołowi programistów i mieli nadzieję, że produktem końcowym będzie system, którego klient oczekiwał.

Kilka pojęć:

W tej książce przez *system* rozumiemy taką kombinację oprogramowania i sprzętu, która jest rozwiązaniem problemu biznesowego. *Budowanie systemu* to tworzenie systemu dla *klienta*, czyli osoby, która ma problem do rozwiązania. *Analityk* tworzy dokumentację problemu klienta i przedstawia ją *deweloperom*, czyli programistom, którzy tworzą oprogramowanie rozwiązujące problem i wdrażają je do użytku na sprzęcie komputerowym.

Ponieważ budowanie systemu jest dziełem ludzi, na każdym etapie działania istnieje potencjalne niebezpieczeństwo powstania błędu. Analitycy mogą źle zrozumieć klienta

lub stworzyć dokument, którego klient nie będzie w stanie pojąć. Wynik pracy analityków może nie być jasny dla programistów, którzy z kolei mogą stworzyć program trudny do obsługi i nierozwiązujący rzeczywistego problemu klienta.

Cóż więc dziwnego, że większość działających od dawna systemów jest źle skonstruowana i trudna do użycia?

Dodanie metody do szaleństwa

We wczesnym okresie komputeryzacji nieliczni wówczas programiści polegałi na ręcznej analizie problemów. Jeżeli w ogóle coś analizowali, robili to zwykle na drugiej stronie serwetki śniadaniowej. Zwykle pisali programy jednym ciągiem, tworząc kod w miarę poznawania problemu. Miało to swą romantyczną atmosferę i śmiałość spojrzenia, ale całkowicie nie pasowało do obecnego, stawiającego wysokie wymagania, świata biznesu.

Dzisiaj dobrze przemyślany plan to podstawa. Klient musi zrozumieć, co robi grupa tworząca oprogramowanie, i musi umieć wskazać, w którym miejscu realizujący zamówienie nie w pełni zrozumieli jego potrzeby (może to być także konieczne, gdy klient zmienił swoje żądania). Ponadto tworzenie systemu jest wysiłkiem grupowym, więc każdy członek grupy musi wiedzieć, którą część całości tworzy (i jak wielka jest ta całość).

W miarę jak świat staje się coraz bardziej skomplikowany, systemy będące częścią tego świata i korzystające z komputerów również stają się bardziej skomplikowane. W skład takich systemów wchodzi często wiele elementów sprzętowych, różnych programów, sieci rozciągnięte na duże odległości, połączenia z bazami danych, w których zostały zmagazynowane całe góry informacji. Jeżeli chcecie, aby Wasze systemy sprawnie i skutecznie działały, musicie się pogodzić z tym, że są one skomplikowane.

Kluczem do osiągnięcia sukcesu jest odpowiednie zorganizowanie procesu projektowania, by analitycy, klienci i programiści zaangażowani w tworzenie systemu rozumieli się nawzajem i potrafili się pogodzić. UML zapewnia taką organizację.

Tak jak nie można zbudować skomplikowanej struktury biurowca bez wcześniejszego nakreślenia szczegółowego rysunku technicznego, tak bez stworzenia szczegółowego projektu nie można stworzyć systemu, który ma w tym biurowcu funkcjonować. Jego plan musi być tak czytelny dla klienta jak projekt architektoniczny dla zleceniodawcy budowy. Musi on być rezultatem szczegółowej analizy potrzeb klienta.

Krótkie terminy wykonania są kolejnym problemem, z którym należy sobie radzić przy tworzeniu współczesnych systemów. Jeżeli terminy „depczą sobie po piętach”, solidne projektowanie jest absolutną koniecznością.

Kolejnym aspektem współczesnego życia wymagającym porządnego projektowania są przejęcia firm. Jeżeli jedna firma przejmuje inną, może dokonać ważnych zmian we własnie realizowanym projekcie (zmienić narzędzia implementacyjne, język ko-

dowania i wiele innych rzeczy). Dobrze skonstruowany plan będzie „odporny na uszkodzenia” i ułatwi dokonywanie niezbędnych poprawek. Jeżeli jest solidny, zmiany implementacyjne przebiegną gładko.

Potrzeba solidnego projektowania zrodziła potrzebę istnienia notacji, którą analitycy, twórcy oprogramowania i klienci zaakceptują jako standardową — tak jak została powszechnie zaakceptowana inżynierska notacja używana przy kreśleniu diagramów obwodów elektronicznych oraz notacja diagramów Feynmana powszechnie przyjęta przez fizyków. UML jest taką notacją.

Jak się narodził UML

UML został wymyślony przez Grady’ego Boocha, Jamesa Rumbaugh’a i Ivara Jacobsona. Ci trzej dżentelmeni, nazwani „Three Amigos”¹, w latach 80 i 90 XX wieku pracowali w trzech różnych organizacjach, niezależnie rozwijając własne metodologie obiektowo zorientowanej analizy i projektowania. Ich osiągnięcia przewyższyły jakością dzieła licznych współzawodników. W połowie lat 90 zaczęli od siebie nawzajem pożyczać pomysły, aż wreszcie zdecydowali się połączyć swoje wysiłki.



Rozdział 2 „Co to jest obiektowość” i rozdział 4 „Związki” dotyczą spraw obiektowości. W naszej książce te koncepcje odgrywają podstawową rolę.

W roku 1994 Rumbaugh przeniósł się do Rational Software Corporation, gdzie Booch już wcześniej pracował. Po roku dołączył do nich Jacobsen.

Reszta, jak mówią, jest historią. Szkicowe wersje UML-a zaczęły krążyć w świecie informatycznym i spotkały się z odzewem, pod wpływem którego wersja ostateczna uległa znacznym zmianom. Wiele korporacji uznało, że UML może dobrze służyć ich strategicznym celom, więc powstało UML-owe konsorcjum.

Wśród jego członków znalazły się takie firmy jak DEC, Hewlett-Packard, Intellicorp, Microsoft, Oracle, Texas Instruments, Rational i inne. Dzięki temu współdziałaniu w roku 1997 stworzono wersję 1.0 UML-a i przedstawiono ją do oceny OMG (ang. *Object Management Group*)² — organizacji, która sformułowała żądania stworzenia standardowego języka modelowania.

¹ „Three Amigos” to nazwa parodii westernu z roku 1986, w której trzech patalachów przeżywa na preri przerożne przygody. Nazwa ta rzeczywiście przylgnęła do trójki twórców UML-a, o czym można się przekonać choćby na stronie internetowej <http://c2.com/cgi/wiki?ThreeAmigos> — przyp. tłum.

² OMG (Object Management Group) to organizacja utworzona w 1989 r. przez 13 liczących się przedsiębiorstw z branży software’owej. Jej celem jest promowanie teorii oraz praktyki technologii obiektowych. Obecnie do OMG należy ponad 750 firm-producentów oprogramowania oraz sprzętu komputerowego. Organizacja zajmuje się opracowywaniem standardów pomagających w tworzeniu aplikacji obiektowych — przyp. tłum.

Konsorcjum rozwinęło się i wyprodukowało wersję 1.1, która — w końcu roku 1997 poddana ocenie OMG — uzyskała aprobatę tej organizacji. OMG przejęła trud rozwijania UML-a, czego rezultatem było wypuszczenie dwóch kolejnych wersji w roku 1998. UML stał się de facto standardem w przemyśle informatycznym i wciąż jest rozwijany.

Komponenty UML-a

UML zawiera wiele elementów graficznych grupowanych w postaci diagramów. Ponieważ jest językiem, określa zasady łączenia tych elementów. Zamiast wyliczania elementów i zasad, lepiej przejdźmy od razu do diagramów, ponieważ właśnie one będą Wam służyć do analizowania systemów.



To podejście jest zbliżone do uczenia się obcego języka przez praktykę, zamiast wkuwania teorii. Jeżeli przez pewien czas będziecie używać obcego języka, łatwiej Wam będzie potem pojąć jego zasady gramatyczne.



Celem diagramów jest pokazanie wielu perspektyw systemu; ten zestaw perspektyw to *model*. UML-owy *model* systemu to coś w rodzaju wykonanego w skali modelu budynku wraz z jego artystyczną interpretacją. Należy podkreślić, że model opisuje, *co* system ma robić, ale nie określa, *jak* system ten ma zostać zaimplementowany.

W kolejnych podrozdziałach zostaną krótko opisane najpowszechniej używane diagramy UML-owe i koncepcje, które reprezentują. Dalej w części pierwszej każdemu z nich przyjrzymy się dokładniej. Pamiętajcie, że jest możliwe tworzenie diagramów hybrydowych i że UML dostarcza sposobów organizowania i rozszerzania diagramów.

Modele:

Model jest pojęciem przydatnym w nauce i inżynierii. W najogólniejszym sensie, tworząc model, używamy czegoś, co dobrze znamy, do zrozumiałego objaśnienia czegoś, o czym wiemy niewiele. W niektórych dziedzinach wiedzy modelem może być układ równań, w innych model to symulacja komputerowa. Istnieje wiele różnych modeli.

Dla nas modelem będzie zestaw diagramów UML-owych, które będziemy mogli sprawdzać, oceniać i modyfikować w celu poznania i rozwinięcia systemu.

Diagram klas

Pomyślcie o czymś należącym do otaczającego Was świata (przynaję, że to dość niesprecyzowane żądanie, ale spróbujcie). Większość rzeczy ma swoje atrybuty (właściwości) i zachowuje się w szczególny sposób. O tych zachowaniach możemy myśleć jako o zbiorze operacji.

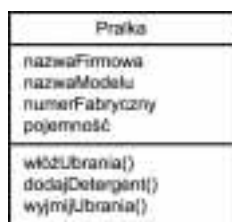


Zauważycie także, że rzeczy w sposób naturalny można przypisać do różnych kategorii (samochody, meble, pralki...). Te kategorie nazywamy klasami. *Klasa* to kategoria lub grupa rzeczy, które mają podobne atrybuty i wspólne zachowania. Oto przykład. Cokolwiek należącego do klasy pralek ma takie atrybuty jak nazwa firmowa, model, numer fabryczny i pojemność. Czynności właściwe dla rzeczy z tej klasy to między innymi działania opisane przez polecenia „włóż ubrania”, „dodaj detergent”, „włącz” i „usuń ubrania”.

Na rysunku 1.1 mamy przykład notacji UML-owej. Jest to prostokątna ikona reprezentująca klasę. W jej trzech polach oprócz nazwy klasy zostały zapisane jej atrybuty i działania. W części najwyższej mamy nazwę klasy, w środkowej — atrybuty, a w najniższej — działania. Diagram klas składa się z pewnej liczby takich prostokątów połączonych liniami wskazującymi zależności między klasami.

Rysunek 1.1.

Ikona klasy



Dlaczego w ogóle mamy się zajmować klasami rzeczy, ich atrybutami i działaniami? W celu osiągnięcia interakcji ze złożonym światem zewnętrznym większość programów symuluje niektóre jego aspekty. Doświadczenie dziesiątków lat wskazuje, że łatwiej jest stworzyć oprogramowanie, gdy reprezentuje ono klasy rzeczy ze świata realnego. Diagramy klas dla tworzących oprogramowanie są taką reprezentacją.

Diagramy klas ułatwiają również wykonywanie analizy. Umożliwiają rozmawianie z klientami ich językiem, za pomocą przyjętych przez nich określeń. Pozwala to na odkrycie ważnych szczegółów problemu, który należy rozwiązać.

Diagram obiektów



Obiekt to egzemplarz należący do klasy — szczególna rzecz, która ma szczególne atrybuty i operacje. Na przykład Wasza pralka może mieć nazwę firmową Laundatorium, nazwę modelu Washmeister, numer fabryczny: GL57774 i pojemność 5 kg.

Na rysunku 1.2 widzimy UML-ową reprezentację obiektu. Zauważ, że ikona jest prostokątna — jak w przypadku klasy, ale nazwa jest podkreślona. Nazwa egzemplarza jest podana przed dwukropkiem, a nazwa klasy — po dwukropku.

Rysunek 1.2.

UML-owa ikona obiektu



Diagram przypadków użycia

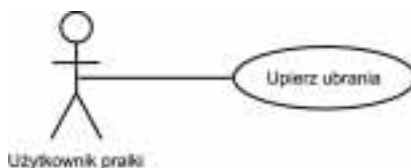


Przypadek użycia to opis zachowania systemu z punktu widzenia użytkownika. Dla tworzących oprogramowanie jest to bardzo przydatne narzędzie: służy do uchwycenia metodą prób i błędów założeń systemu z punktu widzenia użytkownika. Jest to istotne, jeżeli system ma służyć zwykłym ludziom, a nie „cyborgom”.

O przypadkach użycia będziemy szczegółowo mówić w dalszej części książki. Teraz tylko jeden przykład. Pralki używasz, oczywiście, do prania ubrań. Na rysunku 1.3 zostało to pokazane na diagramie UML.

Rysunek 1.3.

Diagram przypadków użycia



Mała schematyczna figurka, w tym przykładzie odpowiadająca użytkownikowi pralki, to *aktor*. Elipsa reprezentuje przypadek użycia. Zauważ, że aktor inicjuje przypadek użycia. Może to być osoba lub inny system.

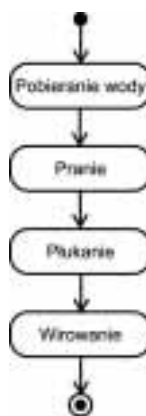
Diagram stanów

Obiekt zawsze jest w jakimś stanie. Człowiek może być noworodkiem, niemowlęciem, dzieckiem, nastolatkiem lub dorosłym. Mamy tu do czynienia ze wznoszeniem, zatrzymaniem lub spadkiem. Pralka może pobierać wodę, prac, płukać, wirować lub być wyłączona.

UML-owy diagram stanów pokazany na rysunku 1.4 reprezentuje ten fragment rzeczywistości. Na rysunku zostały pokazane przejścia od jednego stanu do drugiego.

Rysunek 1.4.

Diagram stanów



Symbol na górze reprezentuje stan początkowy, a symbol na dole — stan końcowy.

Diagram przebiegu

Diagram klas i diagram obiektów przedstawiają informację statyczną. Jednakże w działającym systemie obiekty wpływają na siebie wzajemnie i trwa to w określonym czasie. Diagram przebiegu ukazuje dynamikę interakcji w zależności od czasu.

Wracając do przykładu z pralką, zauważmy, że jej komponenty to rura (do doprowadzenia wody), bęben (część, w której umieszczamy ubrania) i rura odprowadzająca wodę. Oczywiście, jest tam wiele innych obiektów (jak widzicie, jeden obiekt może się składać z wielu innych).

Co się stanie, gdy przystąpicie do realizacji przypadku użycia „Upierz ubrania”? Założmy, że wykonaliście wszystkie operacje „włóż ubrania”, „dodaj detergent” i „włącz”. Kolejność następných kroków powinna być następująca:

1. Woda wpływa do bębna przez rurę.
2. Bęben pozostaje nieruchomy przez pięć minut.
3. Woda przestaje wpływać do bębna.
4. Bęben obraca się tam i z powrotem przez piętnaście minut.
5. Woda z mydłem spływa do ścieku.
6. Woda zaczyna ponownie napływać.
7. Bęben ponownie obraca się tam i z powrotem.
8. Woda przestaje napływać.
9. Woda spłukująca spływa do ścieku.
10. Bęben zaczyna coraz szybciej obracać się w jedną stronę i wiruje przez pięć minut.
11. Bęben przestaje się obracać — pranie jest skończone.

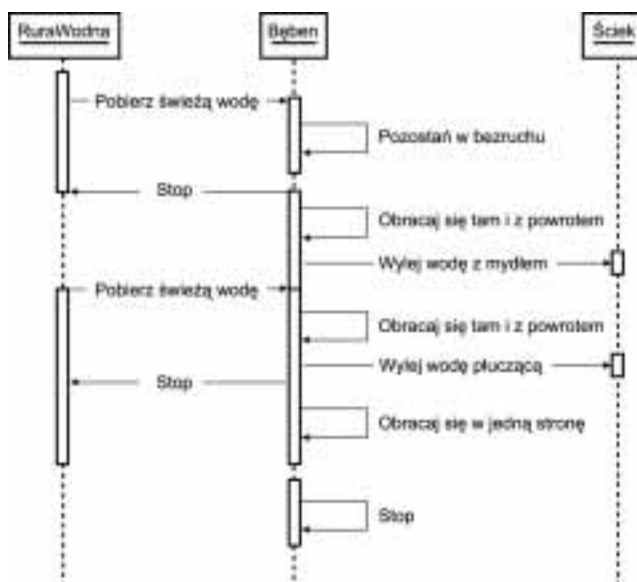
Na rysunku 1.5 widzimy diagram przebiegu ukazujący interakcje między napływającą wodą, bębniem i ściekiem (reprezentowanymi przez prostokąty pokazane na górze) następujące w czasie. Oś czasu na tym diagramie jest skierowana z góry na dół.

A wracając do stanów — kroki 1 i 2 możemy określić jako stan namaczania, kroki 3 i 4 — jako stan prania, kroki 5 i 6 — jako stan płukania, zaś kroki od 8 do 10 — jako stan wirowania.

Diagram czynności

Czynności, które występują w przypadku użycia lub jako operacje obiektu, zwykle następują w określonej kolejności — jak w jedenastu krokach w poprzednim podrozdziale. Kroki od 4 do 6 na rysunku 1.6 zostały pokazane w postaci diagramu czynności.

Rysunek 1.5.
Diagram przebiegu



Rysunek 1.6.
Diagram czynności

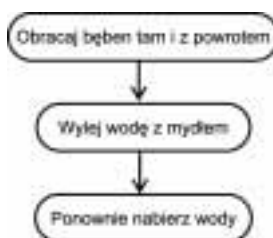


Diagram kooperacji

Elementy systemu współpracują ze sobą w celu zrealizowania celów i język modelowania musi to wyrazić. W UML-u służy temu diagram kooperacji pokazany na rysunku 1.7. Na tym rysunku do zbioru klas tworzącego pralkę został dodany zegar wewnętrzny. Po upływie odpowiednio długiego czasu zegar zatrzymuje nabieranie wody i nakazuje rozpoczęcie obracania bębna tam i z powrotem.

Rysunek 1.7.
Diagram kooperacji

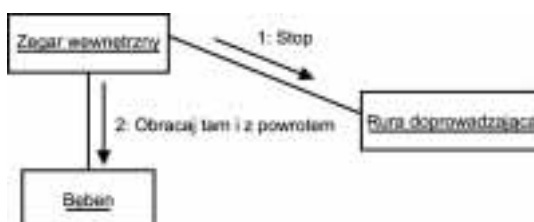


Diagram komponentów

Ten i następny diagram nie mają nic wspólnego z prakkami, ponieważ diagram komponentów i diagram wdrożenia dotyczą systemów komputerowych.

W procesie tworzenia współczesnego oprogramowania używamy komponentów, co jest szczególnie ważne przy oparciu pracy na działaniu grup. Nie będziemy się zbytnio rozwódzić nad tym zagadnieniem. Diagram komponentów został pokazany na rysunku 1.8.

Rysunek 1.8.

Diagram komponentów

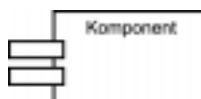
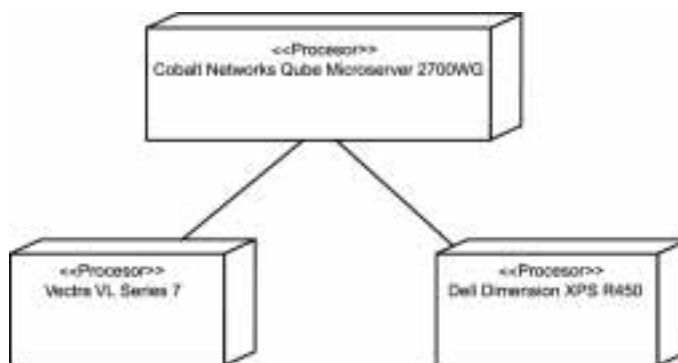


Diagram wdrożenia

Diagram wdrożenia pokazuje fizyczną architekturę systemu komputerowego (rysunek 1.9). Może obrazować komputery i inne przyrządy oraz połączenia między nimi i oprogramowanie na nich zainstalowane. Każdy komputer jest reprezentowany przez prostopadłościan, a połączenia między komputerami są widoczne w postaci linii łączących te prostopadłościany.

Rysunek 1.9.

Diagram wdrożenia



Kilka innych składników

Wspomniałem poprzednio, że UML dostarcza składników służących do organizowania i rozszerzania diagramów.

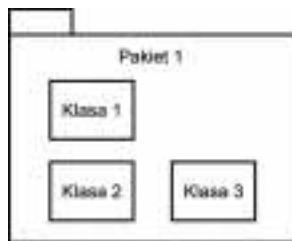
Pakiety



Czasami zechcecie połączyć elementy diagramu w grupę. Na przykład kilka klas lub kilka komponentów tworzy podsystem. Do tego służy grupowanie w *pakiecie* reprezentowanym przez rysunek teczek z zakładką. Patrz rysunek 1.10.

Rysunek 1.10.

Pakiet pozwala na zgrupowanie elementów diagramu



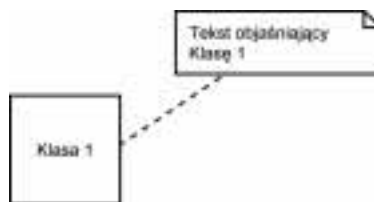
Notatki



Czasami nie jest jednoznacznie określone, dlaczego jakiś element diagramu znajduje się dokładnie w tym miejscu lub co należy z nim robić. W takim przypadku pomocne jest użycie *notatki*. Uważaj notatkę za odpowiednik żółtej karteczki z objaśnieniem przyklepianej gdzieś w widocznym miejscu. Reprezentuje ją ikona w postaci prostokąta z zagiętym rogiem. Wewnątrz prostokąta zapisujemy tekst objaśniający. Przykład został pokazany na rysunku 1.11. Notatkę dodajemy do diagramu, łącząc ją przerywaną linią z elementem, do którego się odnosi.

Rysunek 1.11.

Za pomocą notatki możesz umieścić objaśnienia na dowolnym diagramie



Stereotypy



UML dostarcza wielu pożytecznych elementów, ale nie jest to zestaw wyczerpujący możliwości. Wcześniej czy później będziecie projektować system wymagający użycia bloków konstrukcyjnych specjalnie przystosowanych do jego potrzeb. *Stereotypy* pozwalają na przekształcenie istniejących elementów UML-owych w inne. To tak jakby kupować gotowy garnitur po to, by go przerobić na własny rozmiar (zamiast uszyć nowy z kuponu materiału). Uważajcie stereotyp za przykład tego rodzaju zmiany. Jest przedstawiany w postaci nazwy zawartej w podwójnych nawiasach kątowych zwanych francuskimi.



Dobrym przykładem jest interfejs. *Interfejs* obejmuje tylko operacje, bez atrybutów. Jest to zestaw operacji, które chcecie wielokrotnie wykonywać w modelu. Zamiast wymyślać nowy element reprezentujący interfejs, możecie użyć ikony klasy z napisem «*Interfejs*» umieszczonym nad nazwą klasy. Zostało to pokazane na rysunku 1.12.

Rysunek 1.12.

Stereotyp pozwala na tworzenie nowych elementów na podstawie istniejących



Po co tyle różnych diagramów?

Jak widzicie, diagramy UML-a pozwalają spojrzeć na system z wielu różnych punktów widzenia. Warto zauważyć, że nie wszystkie diagramy muszą pojawiać się we wszystkich modelach. Jednakże większość modeli zawiera diagramy, które zostały wyliczone.



Do czego są potrzebne różne perspektywy systemu? Zwykle w procesie tworzenia bierze udział wielu *uczestników* — ludzi zainteresowanych różnymi aspektami systemu. Wróćmy do naszego przykładu z pralką. Z innej perspektywy patrzycie na system, projektując silnik pralki, z innej, gdy piszecie instrukcję jej używania. Zupełnie inaczej patrzycie na pralkę, projektując jej wygląd, inaczej, gdy chcecie uprać ubranie.

Sumienne projektowanie wymaga spojrzenia na system ze wszystkich możliwych punktów widzenia, a każdy diagram UML-a pokazuje system z innej perspektywy. Celem jest usatysfakcjonowanie każdego uczestnika procesu.

Podsumowanie

Systemy budują ludzie. Bez użycia łatwej do zrozumienia notacji proces tworzenia systemów zawiera wiele potencjalnych możliwości popełnienia błędów.

UML to notacja systemowa, która stała się światowym standardem stosowanym w procesach tworzenia systemów. Jest to rezultat pracy Grady'ego Boocha, Jamesa Rumbaugh'a i Ivara Jacobsona. Korzystanie z diagramów UML-a pozwala analitykom budować plany ukazujące różne oblicza systemu — zrozumiałe dla klientów, programistów i wszystkich innych osób związanych z procesem twórczym. Tworzenie wszystkich tych diagramów jest konieczne, ponieważ każdy z nich przeznaczony jest dla innego typu uczestników owego procesu.

Model przygotowany za pomocą UML-a pokazuje, *co* system ma robić, ale nie wyjaśnia, *jak* to ma być wykonane.

Warsztaty

Wiecie już, czym jest UML. Czas na sprawdzenie nabytej wiedzy o tym doskonałym narzędziu. W tym celu wykonajcie ćwiczenia i odpowiedzcie na pytania zawarte w treści. Rozwiązania znajdziecie w dodatku A „Rozwiązania testów”.

Test

1. Dlaczego do tworzenia modeli systemów potrzebujemy tak wielu rodzajów diagramów?
2. Które diagramy przedstawiają statyczny obraz systemu?
3. Które diagramy przedstawiają dynamiczny obraz systemu (tzn. które ilustrują zmiany zachodzące w czasie)?

Ćwiczenia

1. Załóżmy, że budujesz system komputerowy, który ma grać w szachy z użytkownikiem. Których diagramów powinieneś użyć i dlaczego?
2. Przygotuj listę pytań, które zadałbyś potencjalnemu użytkownikowi systemu z poprzedniego ćwiczenia? Zastanów się, dlaczego właśnie te pytania powinieneś zadać.