ZESZYTY NAUKOWE
WYDZIAŁU NAUK EKONOMICZNYCH

Tomasz SOBESTIAŃCZYK*

STANDARD UML 2.3 W ZARZĄDZANIU WYTWARZANIEM OPROGRAMOWANIA

Zarys treści: W pracy podjęto próbę przedstawienie UML 2.3 jako metody w zarządzaniu wytwarzaniem oprogramowania oraz scharakteryzowanie słabych i mocnych stron. UML służy do modelowania dziedziny problemu (opisywania-modelowania fragmentu istniejącej rzeczywistości – na przykład modelowanie tego, czym zajmuje się jakiś dział w firmie) – w przypadku stosowania go do analizy oraz do modelowania rzeczywistości, która ma dopiero powstać – tworzy się w nim głównie modele systemów informatycznych.
Słowa kluczowe: UML, zarządzanie wytwarzaniem oprogramowania, modelowanie, notacja, wzorzec projektowy, diagram, klasa, system, programowanie.

Wprowadzenie do UML

UML (*ang*. Unified Modeling Language) jest graficznym językiem modelowania pozwalającym na tworzenie modeli systemów (nie tylko informatycznych, lecz głównie do tego celu jest on używany) za pomocą diagramów i słów. Pozwala obrazować, specyfikować i dokumentować elementy systemu oraz korzysta z zalet programowania obiektowego[1].

UML nie jest językiem programowania, choć możliwe jest generowanie kodu na podstawie niektórych diagramów np. generowanych z modeli w narzędziu Enterprise Architekt. UML jest językiem do:

1. Obrazowania
- utrwalanie ulotnych pomysłów (rozwiązań) projektantów systemu,
- przedstawienie projektu w sposób czytelny dla pozostałych członków zespołu,
- przejrzystość projektu.

2. Specyfikowania
- UML wspomaga specyfikowanie wszystkich ważnych decyzji analitycznych, projektowych i implementacyjnych.

* Uniwersytet Jana Kochanowskiego, Kielce
[1] http://wymagania.net/baza-wiedzy/36-baza-wiedzy/80-uml-modelowanie-procesow [2011].

3. Tworzenia
- modele z języka UML można wprost powiązać ze zorientowanymi obiektowo językami programowania (np. Java, C++),
- wsparcie zarówno dla inżynierii do przodu (forward engineering) jak i inżynierii wstecz (reverse engineering).

4. Dokumentowania
- UML pozwala udokumentować każdy etap wytwarzania oprogramowania[2].

UML definiuje dwie podstawowe składowe: notację poszczególnych elementów używanych na diagramach, a z drugiej strony – ich semantykę, czyli tzw. metamodel. Z punktu widzenia analityka istotniejsze jest czytelne i jednoznaczne opisanie modelu tak, aby inne osoby mogły zrozumieć jego znaczenie. Zatem ważniejsza dla niego jest notacja, zaś metamodel powinien być zrozumiały intuicyjnie. Z kolei przy generowaniu kodu i przejściu do implementacji ważniejsze jest ścisłe rozumienie znaczenia poszczególnych elementów, tak aby możliwa była automatyczna konwersja modelu do innego formalizmu.

W języku UML wyróżniamy następujące rodzaje elementów.

1. Strukturalne – statyczne części modelu, reprezentujące składniki pojęciowe lub fizyczne. Wyróżniamy następujące elementy strukturalne:
- klasa,
- interfejs,
- przypadek użycia,
- klasa aktywna,
- komponent,
- węzeł.

2. Czynnościowe – dynamiczna część modelu, wyrażona czasownikami opisującymi zachowanie w czasie i przestrzeni. Wyróżniamy następujące rodzaje elementów czynnościowych:
- interakcja,
- maszyna stanowa.

3. Grupujące – pełnią rolę organizacyjną oraz odpowiadają blokom, na które dany model może zostać rozłożony. Wśród elementów grupujących wyróżniamy następujące elementy:
- komentujące – są to elementy, które pełnią rolę objaśniającą. Są to adnotacje, których można użyć w celu opisania, uwypuklenia lub zaznaczenia dowolnych składników modelu[3].

[2] http://www.staff.amu.edu.pl/~josinski/dapoli0.htm [2011].
[3] http://www.staff.amu.edu.pl/~josinski/dapoli0.htm [2011].

W języku UML wyróżniamy także związki – czyli podstawowe bloki konstrukcyjne UML, służące do łączenia elementów. Wyróżniamy następujące rodzaje związków w języku UML:

- zależność – związek znaczeniowy między dwoma elementami (zmiany dokonane w definicji jednego z elementów mogą mieć wpływ na znaczenie drugiego);
- powiązanie – związek strukturalny, który określa zbiór wiązań między obiektami;
- uogólnienie – związek między dwoma bytami – ogólnym (przodek) i szczegółowym (potomek);
- realizacja – związek znaczeniowy między klasyfikatorami, z których jeden określa kontrakt, a drugi zapewnia wywiązanie się z niego (najczęściej interfejs-klasa)[4].

Diagramy w UML

Diagram jest schematem przedstawiającym zbiór bytów. Najczęściej ma postać grafu, w którym wierzchołkami są elementy, a krawędziami związki. Diagram to swego rodzaju rzut systemu. Tylko w przypadku najprostszych systemów diagram przedstawia pełny obraz bytów wchodzących w skład systemu. Ten sam byt może się pojawić na wszystkich diagramach, jednak najczęściej występuje tylko na niektórych. W wyjątkowych przypadkach dany byt może nie wystąpić na żadnym diagramie[5]. W UML (wersja 2.3) wyróżnia się 14 diagramów głównych oraz 3 abstrakcyjne (struktur, zachowań i interakcji). Diagramy struktur (diagram abstrakcyjny):

- klas (*ang*. class diagram),
- obiektów (*ang*. object diagram),
- komponentów (*ang*. component diagram),
- wdrożenia (*ang*. deployment diagram),
- struktur złożonych (*ang*. composite structure diagram),
- pakietów (*ang*. package diagram),
- profili (*ang*. profile diagram).

Diagramy zachowań (diagram abstrakcyjny):

- czynności (*ang*. activity diagram),
- przypadków użycia (*ang*. use case diagram),
- maszyny stanów (*ang*. state machine diagram).

Diagramy interakcji (diagram abstrakcyjny):

[4] http://stud.ics.p.lodz.pl/~collina/case/ [2011].

[5] *„Język UML – opis notacji”*, Paweł Gryczon, Paweł Stańczuk, Politechnika Warszawska, Warszawa, grudzień 2002 r.

- komunikacji (ang. communication diagram),
- sekwencji (ang. sequence diagram),
- czasowe (ang. timing diagram),
- przeglądu interakcji (ang. interaction overview diagram).

W przypadku modelowania biznesowego wykorzystywane są modyfikacje powyższych diagramów – np. w diagramie biznesowych przypadków użycia, wyznacznikiem tego, że modelujemy biznesowe przypadki użycia wykonywane przez aktorów biznesowych, jest charakterystyczna cięciwa dla symbolów aktora i przypadku użycia.

W praktyce niezwykle rzadko tworzy się wszystkie diagramy. W większości przypadków korzysta się z mniej niż połowy wyżej wymienionych. W niektórych projektach stosowane są np. tylko diagramy przypadków użycia, aktywności i klas[6].

Diagram klas (ang. object diagram)

Diagram klas – cechy:

- zawiera informacje o statycznych związkach między elementami (klasami),
- klasy są ściśle powiązane z technikami programowania zorientowanego obiektowo,
- są jednymi z istotniejszych diagramów w UML.

Symbol klasy:

- symbolem klasy jest prostokąt, zwykle podzielony poziomymi liniami na trzy sekcje: nazwy, atrybutów, operacji. W razie potrzeby może zostać uzupełniony dodatkowymi sekcjami (np. wyjątków).

Klasa – notacja:

- trzy pola: nazwy, atrybutów, metod – możliwe są różne poziomy szczegółowości.

Diagram obiektów (ang. object diagram)

Na diagramie obiektów przedstawia się obiekty, czyli konkretne instancje klas i związki między nimi. Diagram wyobraża statyczny rzut pewnych egzemplarzy elementów występujących na diagramie klas[7].

Diagram obiektów jest wizualizacją hipotetycznego stanu systemu podczas jego działania. Służy do tworzenia przykładów pomagających zrozumieć diagram klas a przede wszystkim powiązań w nim występujących. Z punktu

[6] http://wymagania.net/baza-wiedzy/36-baza-wiedzy/80-uml-modelowanie-procesow [2011].
[7] http://brasil.cel.agh.edu.pl/~09sbfraczek/diagram-obiektow,1,12.html [2011].

widzenia notacji diagramy obiektów używają elementów zapożyczonych z diagramów klas, chociaż często używają prostszej notacji. Skupiają się na obiektach a nie na związkach pomiędzy klasami. Większość z nich używa wyłącznie obiektów i asocjacji[8].

Diagram komponentów (ang. component diagram)

Diagramy komponentów (component diagram) pokazują podział systemów programowych na mniejsze podsystemy.

Komponent to wymienialny, wykonywalny fragment systemu, z ukrytymi szczegółami implementacyjnymi (np. plik .dll, podprogram).

Komponent – reprezentuje jeden fizyczny moduł kodu. Często jest to jeden pakiet, ale nie zawsze istnieje taka jednoznaczna odpowiedniość. Komponent musi posiadać nazwę, wyróżniającą go spośród pozostałych (gdy jest poprzedzony nazwą otaczającego pakietu, wtedy jest to nazwa ścieżkowa, wpp nazwa prosta). Można je grupować w pakiety[9].

Diagram wdrożenia (ang. deployment diagram)

Diagramy wdrożenia przedstawiają powiązania między oprogramowaniem (artefaktami) i sprzętem (węzłami). Są stosowane przy modelowaniu dużych systemów. Diagram wdrożenia (ang. deployment diagram) odzwierciedla fizyczną strukturę całego systemu, z uwzględnieniem oprogramowania i sprzętu.

Jednostki oprogramowania są reprezentowane przez artefakty (czyli skompilowane wersje komponentu, który można uruchomić), dane i biblioteki. Stronę sprzętową reprezentują węzły, czyli poszczególne urządzenia obliczeniowe, komunikacyjne i przechowujące, powiązane ścieżkami komunikacyjnymi (np. połączeniem TCP/IP). Diagramy te są rzadko używane przy modelowaniu mniejszych i średnich systemów, dlatego zwykle ich rola jest ograniczona. Ponieważ posługują się zaledwie kilkoma symbolami, dlatego kluczową rolę odgrywają stereotypy nadawane poszczególnym elementom. Pozwalają one doprecyzować znaczenie i funkcję oprogramowania oraz sprzętu. Diagramy wdrożenia istotną rolę odgrywają przy wdrażaniu dużych, rozproszonych systemów[10].

Diagram struktur złożonych (ang. composite structure diagram)

[8] http://www.ii.pwr.wroc.pl/~kazienko/projektowanie/Diagramy%20klas.pdf [2011].
[9] http://www.mimuw.edu.pl/~janusz/ [2011].
[10] http://www.cs.put.poznan.pl/bwalter/gniezno/03-uml/io-6-wyk.ppt [2011].

Diagram jest schematem przedstawiającym zbiór bytów. Najczęściej ma postać grafu, w którym wierzchołkami są elementy, a krawędziami związki. Diagram to swego rodzaju rzut systemu. Tylko w przypadku najprostszych systemów diagram przedstawia pełny obraz bytów wchodzących w skład systemu. Ten sam byt może się pojawić na wszystkich diagramach, jednak najczęściej występuje tylko na niektórych. W wyjątkowych przypadkach dany byt może nie wystąpić na żadnym diagramie[11].

Diagram pakietów (ang. package diagram)

Diagram jest schematem przedstawiającym zbiór bytów. Najczęściej ma postać grafu, w którym wierzchołkami są elementy, a krawędziami związki. Diagram to swego rodzaju rzut systemu. Tylko w przypadku najprostszych systemów diagram przedstawia pełny obraz bytów wchodzących w skład systemu. Ten sam byt może się pojawić na wszystkich diagramach, jednak najczęściej występuje tylko na niektórych. W wyjątkowych przypadkach dany byt może nie wystąpić na żadnym diagramie[12].

Diagram profili (ang. profile diagram)

Profile UML są narzeczami języka UML przystosowanymi do modelowania pewnej dziedziny zastosowań. Wśród tych mechanizmów najważniejszym są profile, zawierające kompletny i spójny zestaw elementów dedykowanych do modelowania określonej dziedziny, np. systemów czasu rzeczywistego, baz danych, logiki biznesowej etc. Zdefiniowane i zaakceptowane profile pozwalają uniknąć kaskady różnorodnych rozszerzeń dokonywanych przez użytkowników na własną rękę, co znacznie zmniejszało czytelność i komunikatywność modeli.

Diagram czynności (ang. activity diagram)

Diagram czynności w języku UML służy do modelowania czynności i zakresu odpowiedzialności elementów bądź użytkowników systemu. Jest niejako podobny do diagramu stanu, jednak w odróżnieniu od niego nie opisuje działań związanych z jednym obiektem a wieloma, pomiędzy którymi może występować komunikacja przy wykonywaniu czynności[13].

Główne cechy:

[11] *„Język UML – opis notacji”,* Paweł Gryczon, Paweł Stańczuk, Politechnika Warszawska, Warszawa, grudzień 2002 r.
[12] http://www.blaze.pl/go/pliki/inne/opis_uml.pdf [2011].
[13] http://brasil.cel.agh.edu.pl/~09sbfraczek/diagram-aktywnosci,1,10.html [2011].

- diagramy aktywności łączą idee pochodzące z trzech źródeł: diagramów zdarzeń J.Odell'a, technik modelowania stanów i sieci Petriego. Są szczególnie użyteczne przy modelowaniu przepływów operacji czy też opisie zachowania z przewagą przetwarzania współbieżnego,
- diagramy aktywności z zasady nie pokazują wszystkich szczegółów przetwarzania,
- pokazują aktywności bez pokazywania bytów, realizujących daną aktywność i dlatego z reguły używane są jako punkt startowy dla procesu modelowania zachowań,
- dla skompletowania projektu każda aktywność powinna być rozpisana na szereg operacji (akcji), z których każdą trzeba będzie na późniejszym etapie przydzielić do odpowiedniej klasy.

Diagram przypadków użycia (ang. use case diagram)

Diagram przypadków użycia (pot. z ang. use case) – graficzne przedstawienie przypadków użycia, aktorów oraz związków między nimi, występujących w danej dziedzinie przedmiotowej.

Diagram przypadków użycia w języku UML służy do modelowania funkcjonalności systemu. Tworzony jest zazwyczaj w początkowych fazach modelowania. Diagram ten stanowi tylko przegląd możliwych działań w systemie, szczegóły ich przebiegu są modelowane za pomocą innych technik[14].

Diagram przypadków użycia przedstawia usługi, które system świadczy aktorom, lecz bez wskazywania konkretnych rozwiązań technicznych.

Cele stosowania diagramów przypadków użycia:

- identyfikacja oraz dokumentacja wymagań,
- umożliwiają analizę obszaru zastosowań, dziedziny przedmiotowej,
- pozwalają na opracowanie projektu przyszłego systemu,
- stanowią przystępną i zrozumiałą platformę współpracy i komunikacji twórców systemu, inwestorów i właścicieli,
- są rodzajem umowy, kontraktu pomiędzy udziałowcami co do zakresu i funkcjonalności przyszłego systemu,
- stanowią podstawę testowania funkcji systemu na dalszych etapach jego cyklu życia[15].

Diagram maszyny stanów (ang. state machine diagram)

Diagram maszyny stanów – diagram używany przy analizie i projektowaniu oprogramowania. Pokazuje przede wszystkim możliwe stany

[14] http://brasil.cel.agh.edu.pl/~10smbaster/ [2011].
[15] http://www.ee.pw.edu.pl/~ksiezyk/teksty/dydaktyka/AiPSI/DW.pdf [2011].

obiektu oraz przejścia, które powodują zmianę tego stanu. Można też z niego odczytać, jakie sekwencje sygnałów (danych) wejściowych powodują przejście systemu w dany stan, a także jakie akcje są podejmowane w odpowiedzi na pojawienie się określonych stanów wejściowych.

Diagram komunikacji (ang. communication diagram)

Diagram komunikacji znany również jako diagram kolaboracji. Jest to rozszerzona wersja diagramu współdziałania znanego z UML 1.x. Przedstawia sposób wymiany informacji pomiędzy obiektami (aktorami, klasami), które wchodzą ze sobą w interakcję.

Diagram komunikacji (ang. communication diagram) jest rozszerzoną i przemianowaną wersją diagramu współdziałania znanego z UML 1.x. Skupia się on na obiektach wchodzących w skład interakcji i wymienianymi przez nie komunikatach, natomiast w mniejszym stopniu niż diagram sekwencji (choć nadal obecnym) wskazuje na aspekt czasowy. Z tego powodu obiekty na diagramie komunikacji są umieszczone tak, aby łatwo można było opisać ich relacje pomiędzy sobą. Komunikacje są przedstawiane za pomocą linii łączących obiekty, natomiast przesyłane między obiektami komunikaty i dane są umieszczane obok tych linii. Każdy komunikat jest opatrzony etykietą liczbową, wskazującą na kolejność ich wysyłania. Etykieta ta ma postać liczb oddzielonych kropkami[16].

Diagram sekwencji (ang. sequence diagram)

Diagram sekwencji jest kolejnym elementem modelu zorientowanego obiektowo. Diagram przedstawia obiekty (instancje klas), stanowiące składowe jakiegoś systemu oraz komunikaty wymieniane pomiędzy nimi w celu realizacji danego zadania. Diagram może, ale nie musi, zawierać również aktorów, oraz opisywać ich interakcję z systemem.

Diagram sekwencji reprezentuje zachowanie systemu pod kątem interakcji. Uzupełnia diagram klas, który reprezentuje statyczną strukturę systemu. Diagram sekwencji jest dynamiczny: opisuje zachowanie klas, interfejsów oraz możliwe zastosowanie ich operacji (metod)[17].

Diagram tego typu precyzuje role obiektów w układzie chronologicznym w oparciu o zdarzenia. Tworzenie modelu systemu rozpoczyna się zwykle od diagramu przypadków użycia, na którego podstawie identyfikuje się klasy, wykorzystywane później w diagramie klas. Następnie tworzone są diagramy

[16] http://smurf.mimuw.edu.pl/external_slides/UML_cz.II/UML_cz.II.html [2011].
[17] http://wneiz.as.prv.pl/IO/IO4.pdf [2011].

sekwencji, których celem jest wskazanie interakcji pomiędzy obiektami biorącymi udział w przypadku użycia[18].

Diagram czasowy (ang. timing diagram)

Diagram czasowy jest rodzajem diagramu interakcji, reprezentującym na osi czasu zmiany dopuszczalnych stanów, jakie może przyjmować instancja klasyfikatora uczestnicząca w interakcji.

Diagramy te stosuje się w celu sporządzenia harmonogramów interakcji, a więc specyfikacji interakcji instancji klasyfikatorów w aspekcie zmian czasu trwania ich stanów. Punktem wyjścia tych diagramów są podstawowe kategorie diagramów sekwencji oraz maszyn stanowych. Terminowa realizacja interakcji wymaga niekiedy wielkiej dokładności czasowej. W związku z tym na diagramach harmonogramowania można przedstawiać kolejność występowania stanów instancji klasyfikatorów oraz czas ich trwania. Wprowadzenie diagramów harmonogramowania w UML 2.0 jest istotną zmianą w stosunku do poprzednich wersji. Ich tworzenie i użytkowanie jest szczególnie zalecane w systemach o rozbudowanej dynamice[19].

Diagram przeglądu interakcji (ang. interaction overview diagram)

Diagram przeglądu interakcji w języku UML służy do opisu zależności przy przesyłaniu komunikatów pomiędzy obiektami, czyli zależności w przepływie sterowania pomiędzy obiektami. Diagram ułatwiający zrozumienie zależności w przepływie sterowania. Metody realizujące to sterowanie są rozproszone w wielu klasach, co powoduje trudności ze zrozumieniem ich wzajemnej zależności i interakcji. Jest to powód sporządzania diagramów interakcji. Służą one do opisu zależności przy przesyłaniu komunikatów dla pewnej grupy obiektów. Często stanowią bardziej precyzyjny opis pojedynczego przypadku użycia. Istnieje wiele wariantów diagramów interakcji o różnych odmianach syntaktycznych. UML wprowadza dwa rodzaje takich diagramów: diagramy sekwencji i diagramy kolaboracji (współpracy)[20]. Przeglądowe diagramy interakcji (ang. interaction overview diagrams) udostępniają wysokiego poziomu widok wzajemnej współpracy kilku interakcji wykorzystywanych w celu implementacji pewnej części systemu, na przykład danego przypadku użycia.

[18] http://wneiz.as.prv.pl/IO/IO4.pdf [2011].
[19] http://brasil.cel.agh.edu.pl/~09sbfraczek/diagram-czasowe,1,15.html [2011].
[20] http://www.pjwstk.dyski.one.pl [2011].

Podsumowanie

UML w krótkim czasie stał się bardzo popularny. Należy oczekiwać, że przez wiele lat będzie dominował w obszarze analizy i projektowania, zwłaszcza, że stał się składową standardu OMG (CORBA). W odróżnieniu od innych tego rodzaju propozycji, UML nie ma ambicji być metodyką projektowania. Jest to raczej zestaw pojęć, oznaczeń i reguł syntaktycznych, który może być użyty w dowolnej metodyce. Notacja ta opiera się o podstawowe pojęcia obiektowości. UML wprowadza pojęcia i diagramy, które w założeniu mają przykryć większość aspektów modelowanych systemów. Jednakże kwestia semantyki tej notacji pozostaje mglista, co jest prawdopodobnie nieuchronnym skutkiem sprzeczności pomiędzy naturą procesów twórczych, wysokim poziomem abstrakcji i precyzyjną formalizacją. Równie poważnym problemem jest pragmatyka użycia tej notacji (tj. reguły dopasowania notacji do konkretnej sytuacji występującej w procesie projektowym). Niektóre rozwiązania zastosowane w UML (w szczególności, opis jego semantyki, tzw. metamodel) są przedmiotem krytyki, co jest prawdopodobnie nieuchronne, zważywszy na wagę tematu dla biznesu i konkurencję, która nie jest zainteresowana powodzeniem UML.

Dlaczego warto stosować UML:

- oprogramowanie jest profesjonalnie zaprojektowane i udokumentowane, przed rozpoczęciem etapu kodowania. Dokładnie wiadomo co system będzie „robił”;
- większa efektywność i niższy koszt implementacji;
- błędy „logiczne” projektu wychwycone na etapie modelu;
- projekt systemu narzuca implementację, a nie odwrotnie (co się często zdarza);
- zmiany w istniejącym systemie wprowadzane są łatwiej, gdy istnieje odpowiednia dokumentacja UML;
- zarządzanie implementacją (zmiana organizacji wśród programistów) jest łatwiejsze, gdy istnieje dokumentacja UML;
- dokumentacja UML ułatwia komunikację, zarówno na poziomie wykonawca – klient, jak i w ramach struktury wykonawcy.

Największą zaletą UML jest to, iż jest to notacja graficzna. Rozpoznawanie symboli jest silną stroną ludzkiego mózgu, dlatego oglądając prostokąty i linie na diagramach łatwo zrozumieć zależności między obiektami. Co więcej, tak jak każda notacja graficzna, UML pozwala skupić uwagę na najważniejszych koncepcjach lub pomysłach i pominąć bądź ukryć nieistotne szczegóły.

UML nie jest – w założeniu swoich twórców – wysoce formalnym językiem do przedstawiania czy udowadniania nowych teorii. Ma to być przede

wszystkim uniwersalny język modelowania ogólnego zastosowania, przeznaczony do wykorzystania tam, gdzie tworzy się systemy oprogramowania.

Zalety UML:

- jest notacją systemową, która stała się światowym standardem w procesach tworzenia systemów (nie tylko informatycznych);
- pozwala opisać system z różnych punków widzenia (klienta, analityka, integratora, programisty, i innych);
- 13 rodzajów diagramów – w konkretnym projekcie nie wszystkie muszą wystąpić;
- model UML pokazuje, co system ma robić, ale nie wyjaśnia, jak to ma być wykonane.

Wady UML:

- duża złożoność języka;
- podobieństwo niektórych symboli i rodzajów linii powodujące trudności z rozróżnieniem znaczenia „na pierwszy rzut oka";
- wada pośrednia – bywa nadużywany (budowa wspaniałego modelu kosztem rzeczywistej realizacji zadania).

Wyliczając zalety i zalety nie wolno zapomnieć, że UML to tylko notacja. Sama znajomość notacji i ewentualnie opanowanie jakiegoś narzędzia CASE (Computer Aided Software Engineering) nie wystarczy żeby zostać analitykiem, projektantem lub programistą. To tak samo jak znajomość języka polskiego i umiejętność posługiwania się edytorem tekstu nie czyni od razu pisarzem.

Bibliografia

1. *„Infrastructure and Superstructure"*, UML 2.3, *http://www.omg.org*.
2. *„Język UML – opis notacji"*, Paweł Gryczon, Paweł Stańczuk, Politechnika Warszawska, Warszawa, grudzień 2002 r. – fragment pracy dyplomowej magisterskiej pt. *„Obiektowy system konstrukcji scenariuszy przypadków użycia"*.
3. http://case-tools.org.
4. http://www.uml.com.pl.
5. http://www.staff.amu.edu.pl.
6. http://www.wymagania.net.
7. http://www.stud.ics.p.lodz.pl.
8. http://www.brasil.cel.agh.edu.pl.
9. http://www.ii.pwr.wroc.pl.
10. http://www.informatycy.slask.pl.

11. http://www.tempus.metal.agh.edu.pl.
12. http://www.mimuw.edu.pl.
13. http://www.cs.put.poznan.pl.
14. http://www.ee.pw.edu.pl.
15. http://www.wneiz.as.prv.pl.
16. http://www.pjwstk.dyski.one.pl.

UML 2.3 STANDARD IN SOFTWARE DEVELOPMENT MANAGEMENT

This publication describes standard: UML 2.3. Unified Modeling Language (UML) is a standardized general-purpose modeling language in the field of object-oriented software engineering.
Keywords: UML, management software development, modeling, notation, a design pattern, diagram, class, system, programming.