ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1999
Seria: INFORMATYKA z. 36 Nr kol. 1414

Anna BOBKOWSKA, Henryk KRAWCZYK
Politechnika Gdańska, Wydział Elektroniki, Telekomunikacji i Informatyki

WYTWARZANIE SIECIOWYCH APLIKACJI UŻYTKOWYCH Z WYKORZYSTANIEM NOTACJI UML[1]

Streszczenie. Jedną z cech współczesnych procesów projektowych jest ich konfigurowalność. Przedstawiono elastyczny proces wytwarzania użytkowych aplikacji sieciowych ze śledzeniem wybranych parametrów jakościowych w poszczególnych fazach jego cyklu życia. Zaprezentowano przydatność notacji UML do opisu aplikacji sieciowych oraz prognozowania charakterystyk jakościowych. Zasygnalizowano potrzebę budowy środowiska wspierającego taką metodę wytwarzania aplikacji sieciowych.

NETWORK APPLICATIONS DEVELOPMENT WITH UML NOTATION

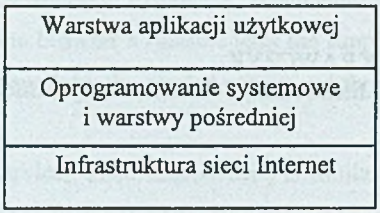
Summary. One of the features of contemporary software development processes is their configurability. In the paper, flexible development process for network applications is proposed. It emphasises quality criteria traceability in each phase of the application development. Accuracy of the UML notation for modelling network applications and its suitability for prediction of the quality criteria is presented. A need for creation of the environment supporting such development strategy is also suggested.

1. Wprowadzenie

Wytwarzane współcześnie aplikacje są coraz bardziej złożone i coraz bardziej rozproszone. Duże nowe możliwości daje infrastruktura sieci Internet wraz z oprogramowaniem systemowym. Dzięki gwałtownemu rozwojowi technologii informatycznych wytwarzanie aplikacji najczęściej sprowadza się do oprogramowania

[1] Realizacja projektu w ramach grantu KBN Nr 08 T11C 12 043 „Metodyka oceny jakości oprogramowania systemów informatycznych".

najwyższej warstwy systemu sieciowego [7]. Rys. 1 przedstawia kontekst wytwarzania aplikacji w sieci Internet.



Rys. 1. Schemat architektury warstwowej aplikacji sieciowych
Fig. 1. Schema of the layer architecture of network applications

Ważne jest, aby powstające oprogramowanie miało odpowiednią dokumentację i było wykonane zgodnie ze standardem gwarantującym wysoką jakość rozwiązania. Taki standard powinien obejmować zarówno proces wytwarzania, definicję produktów powstających w wyniku każdej fazy wytwarzania aplikacji, jak również zestaw charakterystyk jakości produktu końcowego.

1.1. Współczesne metody wytwarzania aplikacji

W obiektowych metodach wytwarzania oprogramowania pierwszej generacji (Coad-Yourdon, OMT [11]) próbowano zaproponować ogólną metodę postępowania przy wytwarzaniu oprogramowania. Jednakże - jak wykazywały studia przypadków - takie metody były albo nadmiarowe, albo niedostateczne w zależności od typu systemu. Dlatego współczesne procesy wytwarzania oprogramowania, chociaż często sugerują pewne szablony procesów, to jednak nie określają dokładnie procesu, a tylko podają pewne cechy charakterystyczne. Współczesne metody (Objectory, Catalysis [4]) cechuje:

- wykorzystanie technologii obiektowej, między innymi obiektowych notacji, do modelowania systemu;
- iteracyjność i inkrementacyjność procesu wytwarzania - np. w Objectory są wyróżnione cykle, fazy, iteracje i punkty końcowe, przy czym ich wykonanie jest planowane i kontrolowane;
- zarządzanie wymaganiami i sterowanie wprowadzaniem zmian (zbieranie wymagań, zarządzanie, komunikacja);
- zapewnienie elastyczności i modyfikowalności poprzez silną koncentrację na architekturze - proponowane są wzorce dla poszczególnych typów aplikacji;
- wspomaganie rozwoju aplikacji polegającego na dołączaniu komponentów;

- konfigurowalność procesu pod względem czynności, przepływu pracy, ilości wykonawców, komponentów procesu.

W metodzie Objectory podstawową funkcję pełnią przypadki użycia. Przypadki użycia służą do specyfikacji wymagań użytkownika, są podstawą opisu zachowania i specyfikacji przypadków testowych, a także na ich podstawie można wygenerować szkielet dokumentacji użytkownika. Umożliwiają łatwe dodanie nowych funkcji poprzez uwzględnienie dalszych przypadków użycia. Dzięki nim możliwe jest śledzenie wymagań oraz zachowanie spójności i elastyczności.

Metoda Catalysis koncentruje się na sprawdzaniu semantyki i spójności, łączeniu formalnych i wizualnych metod opisu, co pozwala na automatyczne sprawdzanie poprawności. Daje również możliwość opisu aplikacji na różnych poziomach abstrakcji i uszczegółowienia, a także zapewnia wspomaganie reinżynieringu.

W procesie projektowania ważne są także aspekty pozatechniczne, aspekty menedżersko-biznesowe, organizacyjne, kwestie skrócenia czasu dostarczenia produktu, wykonania projektu w rozproszeniu, a także aspekty związane z zapewnieniem jakości produktu. Na przykład, głównym celem metody ASP (Agile Software Process [1]) jest skrócenie czasu dostarczenia produktu na rynek poprzez skrócenie cyklu wytwórczego, szybką adaptację zmieniających się wymagań i planu dostarczania oraz indywidualizację procesów projektowych. Chociaż cykl wytwarzania pojedynczego modułu bazuje na modelu kaskadowym, cała architektura procesu jest iteracyjna i inkrementacyjna.

Podejmowane są próby połączenia metodologii z wytycznymi dotyczącymi zapewnienia jakości, których przykładem jest model CMM. W procesie Objectory opartym na przypadkach użycia możliwa jest do spełnienia (a nawet wspomagane jest spełnienie) dużej części wytycznych dla procesów będących na 2 i 3 poziomie modelu CMM [3, 10].

1.2. Notacja UML

UML (Unified Modelling Language [9]) jest notacją służącą do obiektowego modelowania systemów. Nie wspomaga ona konkretnego procesu projektowego, ale - zgodnie z intencjami autorów - ma być podstawą wszystkich procesów i stanowić standard dokumentacji. UML w wersji 1.1 jest wynikiem uzgodnień pomiędzy większością dużych firm wytwarzających oprogramowanie w technologii obiektowej, uwzględnia koncepcje z wielu obiektowych metod modelowania systemów i obecnie podlega standaryzacji w OMG (Object Management Group). Ponieważ założeniem tego standardu jest pokrycie prawie wszystkich koncepcji modelowania, zbiór notacji jest nieco nadmiarowy, szczególnie w części dotyczącej modelowania zachowania - tak więc projektant procesu ma możliwość wyboru odpowiednich środków wyrazu w zależności od potrzeby i charakteru modelowanego

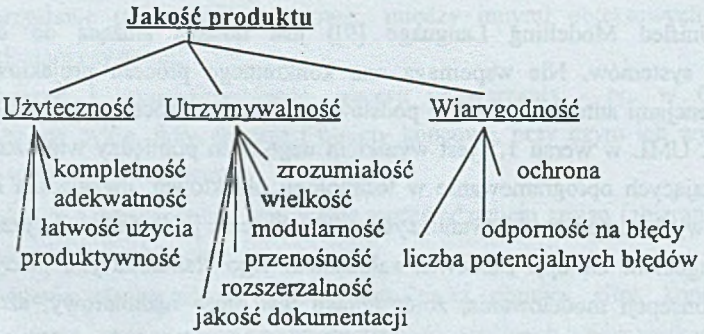
obiektu. UML udostępnia szereg diagramów, które znajdują zastosowanie zarówno w fazie analizy, jak i w fazie projektowania. Modele różnią się zakresem i stopniem szczegółowości. Wśród diagramów wyróżniono:

- diagramy struktury statycznej: diagram klas (ang. class diagram) i diagram obiektów (ang. object diagram);
- diagram przypadków użycia (ang. use case diagram);
- diagramy interakcji: diagram ciągów zdarzeń (ang. sequence diagram) i diagram współpracy (ang. collaboration diagram);
- diagramy oparte na stanach: rozszerzony diagram przejść stanów (ang. statechart diagram) i diagram działania (ang. activity diagram);
- diagramy implementacyjne: diagram komponentów i diagram alokacji podsystemów i procesów do procesorów (ang. deployment diagram).

Oprócz modelowania za pomocą diagramów konieczne jest dodanie opisów w formie tekstowej lub tabelarycznej. Jeżeli tylko jest to możliwe, proponowane jest użycie szablonów. Diagramy dają całościowy pogląd na system, co stanowi wielką zaletę, zwłaszcza w przypadku bardzo złożonych systemów, natomiast pomijają z konieczności szczegółowe opisy. Tak więc formy notacyjna i opisowa wzajemnie się uzupełniają.

1.3. Charakterystyki jakościowe dla użytkowych aplikacji sieciowych

Charakterystyki jakościowe określają cechy wytwarzanego produktu, które powinny być wzięte pod uwagę przy ocenie gotowej aplikacji [5]. Niektóre charakterystyki, np. kompletność, adekwatność, określają funkcjonalność systemu, natomiast inne, jak np. wydajność, wiarygodność, są związane z atrybutami pozafunkcjonalnymi. Zbiór atrybutów i charakterystyk jakości, modelowany na drzewie jakości, jest uzależniony od typu aplikacji. Przykładowe drzewo jakości określone dla użytkowych aplikacji sieciowych pokazane jest na rys. 2.

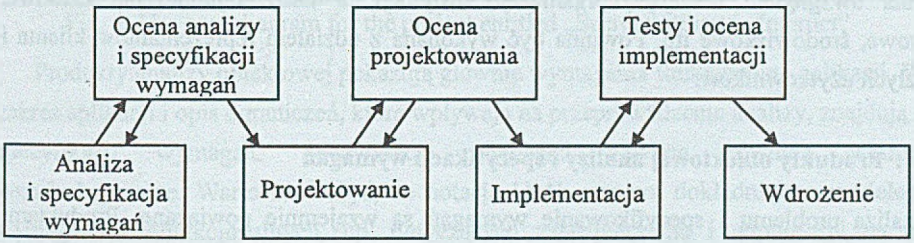


Rys. 2. Część drzewa jakości niezmienna dla wszystkich faz wytwarzania oprogramowania
Fig. 2. A part of quality tree that is valid through all the phases of software development

Część charakterystyk, np. kompletność, adekwatność, zrozumiałość, ma charakter ogólny, tzn. są one istotne dla wszystkich typów aplikacji; natomiast inne, np. ochrona, odporność na błędy, uzyskują większe znaczenie w związku z wprowadzeniem rozproszenia. Takie drzewko jakości zdefiniowane w fazie specyfikacji wymagań określa podstawowe wymagania dotyczące jakości aplikacji. Zakres dopuszczalnych wartości charakterystyk i metryk oraz ich konkretna postać powinny być wynegocjowane z potencjalnym użytkownikiem. We wszystkich fazach procesu wytwarzania oprogramowania o wysokiej jakości powinno być kontrolowane uwzględnienie wymagań zarówno funkcjonalnych, jak i pozafunkcjonalnych.

1.4. Zakres pracy

W pracy przyjęto założenie, że proces wytwarzania będzie iteracyjny, tzn. po każdej fazie wytwarzania oprogramowania będzie następowała dodatkowa faza oceny dokumentacji i prognozowania jakości produktu końcowego [2, 6, 8]. Podstawą oceny będą pomiary i oceny dokonywane przez osobę nie biorącą udziału w pracach analityczno-projektowych. Na rys. 3. przedstawiony jest schemat iteracji w jednym cyklu wytwarzania produktu. Po każdej fazie następuje ocena jakości i w przypadku, gdy nie jest ona zadowalająca, następuje powrót do tej fazy w celu wprowadzenia poprawek.



Rys. 3. Schemat iteracji w cyklu wytwarzania oprogramowania
Fig. 3. Schema of iterations in the cycle of the software development

Celem pracy jest wykazanie, że modele UML są przydatne do analizy i projektowania aplikacji sieciowych oraz że na ich podstawie możliwe jest śledzenie wybranych parametrów jakościowych. Metoda UML szczególnie przydaje się do opisu złożonych aplikacji. Wynika to z wyodrębniania komponentów i dokładnego określania interfejsów pomiędzy nimi, co umożliwia ukrycie szczegółów standardowej infrastruktury sieciowej, możliwości opisu różnych mechanizmów komunikacyjnych i specyfikowania rozmieszczenia komponentów na poszczególne węzły sieci oraz łatwości rozbudowy i modyfikowalności dzięki wprowadzeniu przypadków użycia. Występowanie tych cech w aplikacjach sieciowych sugeruje naturalne wykorzystanie notacji UML, która z założenia ma umożliwiać opis szerokiej gamy klas

aplikacji. W rozdziale 2 zaprezentowano fazę analizy i specyfikacji wymagań oraz odpowiadającą im fazę oceny, natomiast w rozdziale 3 - fazę projektowania oraz fazę oceny projektowania i implementacji. Rozdział 4 został poświęcony omówieniu koncepcji systemu wspomagania wytwarzania aplikacji metodą iteracyjną. Przedstawione treści zilustrowano na przykładzie aplikacji zrealizowanej tą metodą przez grupę studentów ze specjalności „Przetwarzanie równoległe i rozproszone" kierunku Informatyka na wydziale Elektroniki, Telekomunikacji i Informatyki Politechniki Gdańskiej. Aplikacja dotyczy realizacji tajnego głosowania w Internecie przez przedstawicieli rozproszonej instytucji. Aplikacja powinna podawać tematy głosowania, uwierzytelniać użytkowników, zbierać i podliczać głosy oraz informować o wynikach głosowania. Projekt zakłada istnienie użytkownika o specjalnych uprawnieniach, który określa tematy głosowania i terminy zakończenia głosowania oraz listy użytkowników uprawnionych do głosowania na dany temat. Może on również orzec nieważność głosowania z przyczyn niezależnych od systemu.

2. Analiza problemu i specyfikacja wymagań

Celem fazy analizy jest określenie, jaka aplikacja jest użyteczna dla potrzeb danej organizacji, tzn. jakie są konkretne wymagania funkcjonalne i pozafunkcjonalne. Analiza powinna uwzględnić istniejącą organizację instytucji, a także ograniczenia czasowe, finansowe, środowiskowe itp. Powinna być wykonana z udziałem reprezentantów klienta i przyszłych użytkowników.

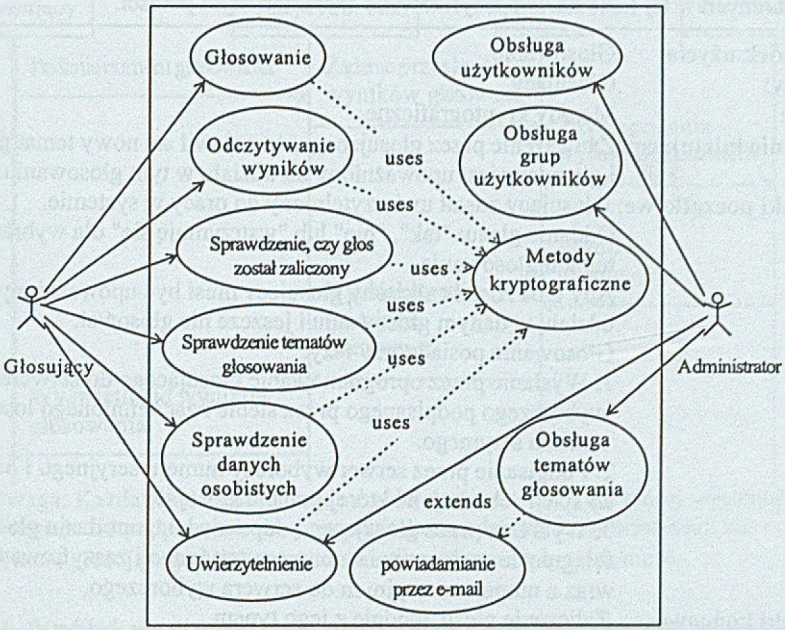
2.1. Produkty obiektowej analizy i specyfikacji wymagań

Analiza problemu i specyfikowanie wymagań są wzajemnie powiązane. Produktami obiektowej analizy są następujące diagramy wykonane w notacji UML:

- diagram przypadków użycia (ang. use cases diagram) wraz z opisem w formie szablonu;
- diagram klas (ang. class diagram) z opisem;
- diagramy ciągów zdarzeń (ang. sequence diagram) pokazujące sytuacje normalne i wyjątkowe.

Podobnie jak w metodzie Objectory, w prezentowanej metodzie podstawą jest diagram przypadków użycia, który pokazuje, jakie usługi system udostępnia i reprezentuje punkt widzenia klienta. Następnie interakcje z systemem są modelowane na diagramach ciągów zdarzeń, przy czym każdemu powiązaniu typu „komunikuje się" na diagramie przypadków użycia odpowiada co najmniej jeden diagram ciągu zdarzeń. Przykłady diagramu przypadków

użycia, szablonu i diagramu ciągu zdarzeń dla rozpatrywanego projektu znajdują się na rys. 4, 5 i 6.



Rys. 4. Diagram przypadków użycia dla projektu pt. „Tajne głosowanie przez Internet"
Fig. 4. Use case diagram for the project entitled „Secret voting via Internet"

Produkty analizy obiektowej pokazują głównie wymagania funkcjonalne aplikacji. Wizja, zakres aplikacji i opis ograniczeń, które wpływają na przeprowadzenie analizy, znajdują się w specyfikacji wymagań. Specyfikacja wymagań zawiera również wymagania pozafunkcjonalne. Warto zauważyć, że notacja UML wspiera dokładniejsze modelowanie wymagań pozafunkcjonalnych, np. dotyczących zabezpieczenia i odporności na błędy. Dostępna na diagramie klas informacja o klasach obiektów może być wykorzystana do określenia, które obiekty powinny być chronione i w jaki sposób. Znajomość profili użytkowników systemu - aktorów na diagramie przypadków użycia ułatwia specyfikowanie mechanizmów uwierzytelniania, które należy wprowadzić. Zdarzenia dotyczące błędnego zachowania aplikacji są specyfikowane jako sytuacje wyjątkowe na diagramie ciągu zdarzeń, co stanowi szkielet analizy odporności systemu na błędy.

2.2. Ocena analizy i specyfikacji wymagań

Po przeprowadzeniu analizy i wyspecyfikowaniu wymagań następuje ocena dokumentacji wykonana przez osobę spoza zespołu wykonującego prace analityczne. Faza oceny ma na celu znalezienie błędów i niedoskonałości i usunięcie ich jak najwcześniej i możliwie

najniższym kosztem. Pewnym problemem jest to, że metryki fazy analizy nie mierzą w bezpośredni sposób charakterystyk jakościowych, dlatego potrzebne jest przełożenie metryk i ocen zebranych w tej fazie na wyspecyfikowane charakterystyki jakości.

Przypadek użycia:	Głosowanie
Aktorzy:	Głosujący
Używa:	Metody kryptograficzne.
Zdarzenie inicjujące:	Zauważenie przez głosującego, że pojawił się nowy temat głosowania i głosujący jest upoważniony do udziału w tym głosowaniu.
Warunki początkowe:	Głosujący został uwierzytelniony do pracy w systemie.
Opis:	Oddanie głosu: "tak", "nie" lub "wstrzymuję się" dla wybranego tematu głosowania. Aby głos został zaliczony, głosujący musi być upoważniony do udziału w danym głosowaniu i jeszcze nie głosował. Głosowanie posiada trzy fazy: 1. Wysłanie przez oprogramowanie głosującego do serwera wyborczego podpisanego przez siebie i zaciemnionego losowego numeru seryjnego. 2. Podpisanie przez serwer wyborczy numeru seryjnego i odesłanie go do stacji roboczej, na której pracuje głosujący. 3. Wybranie przez głosującego odpowiedniego rodzaju głosu, ściągnięcie zaciemnienia z numeru seryjnego i zaszyfrowanego głosu wraz z numerem seryjnym do serwera wyborczego.
Warunki końcowe:	Zaliczenie głosu zgodnie z jego typem.
Wyjątki:	1. Głosujący już głosował lub nie jest upoważniony do głosowania. 2. Zestaw głosów bez prawidłowego podpisu głosującego. 3. Oddany głos bez prawidłowego podpisu serwera wyborczego. Akcja: Wyświetlenie komunikatu o rodzaju błędu.

Rys. 5. Szablon opisu przypadku użycia z przykładem opisu „tajnego głosowania"
Fig. 5. Use case template with fillings for „Secret voting"

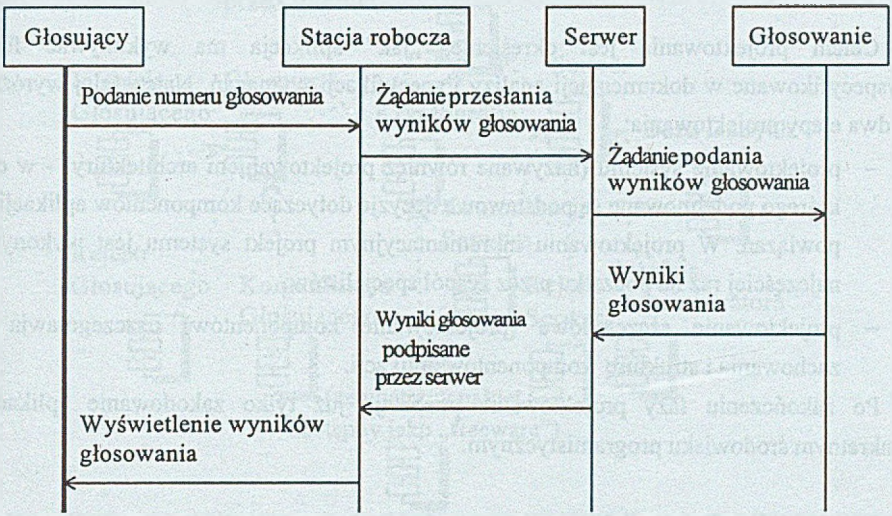
Oceny jakości dotyczą:

- stopnia spełnienia wymagań;
- podania oczekiwanych wartości metryk, charakterystyk i atrybutów;
- wskazania źródeł błędów i metod ich usunięcia;
- określenia zależności pomiędzy metrykami i ocenami a charakterystykami jakości.

Ocena jakości może być przeprowadzona za pomocą ankiety lub pakietu wspomagającego ocenę. Można w nich wyróżnić następujące elementy:

- część dotyczącą zbierania danych i sprawdzania spójności;
- część wnioskowania o charakterystykach i atrybutach jakości;
- część generowania porad dotyczących poprawy jakości.

Sprawdzenie wyników głosowania



Uwaga: Każda wiadomość od serwera do stacji roboczej, dla której występuje niezgodność podpisu, powoduje wyświetlenie odpowiedniej informacji na ekranie głosującego i dalsze oczekiwanie na poprawną wiadomość.

Rys. 6. Przykładowy diagram ciągu zdarzeń dla sprawdzenia wyników głosowania
Fig. 6. Example of a sequence diagram for finding out voting results via Internet

W ramach oceny analizy problemu i specyfikacji wymagań można sprawdzić:

- czy zostały wyspecyfikowane wszystkie funkcje i czy są one adekwatne;
- czy zostały uwzględnione wszystkie adekwatne klasy obiektów;
- czy usługi będą łatwo dostępne;
- czy dokumentacja jest zrozumiała, jednoznaczna, spójna, bez nadmiarowej złożoności i weryfikowalna;
- czy dokumentacja nie jest zbyt małej wielkości, co może wskazywać na niewystarczające zaangażowanie analityka i małą dokładność opisu;
- czy zostały wyspecyfikowane wszystkie możliwe sytuacje błędne i odpowiednie zachowanie systemu w razie ich wystąpienia;
- czy jest wyspecyfikowana odpowiednia ochrona zasobów systemu;
- jaka jest liczba potencjalnych błędów i jaka jest ich waga.

Na tej podstawie następuje wnioskowanie o atrybutach jakości oraz o wykonalności aplikacji w ramach znanych ograniczeń czasowych poprzez porównywanie uzyskanych danych z wartościami oczekiwanymi, a następnie formułowane są zalecenia co do poprawy jakości produktów.

3. Projektowanie i implementacja

Celem projektowania jest określenie, „jak" aplikacja ma wykonywać funkcje wyspecyfikowane w dokumentacji analizy i specyfikacji wymagań. Najczęściej wyróżniane są dwa etapy projektowania:

- projektowanie systemu (nazywane również projektowaniem architektury) - w czasie którego podejmowane są podstawowe decyzje dotyczące komponentów aplikacji i ich powiązań. W projektowaniu inkrementacyjnym projekt systemu jest wykonywany najczęściej raz na początku przez zespół specjalistów;
- projektowanie szczegółowe (projektowanie komponentów) uszczegóławia opis zachowania i strukturę komponentów aplikacji.

Po zakończeniu fazy projektowania pozostaje już tylko zakodowanie aplikacji w konkretnym środowisku programistycznym.

3.1. Produkty fazy projektowania

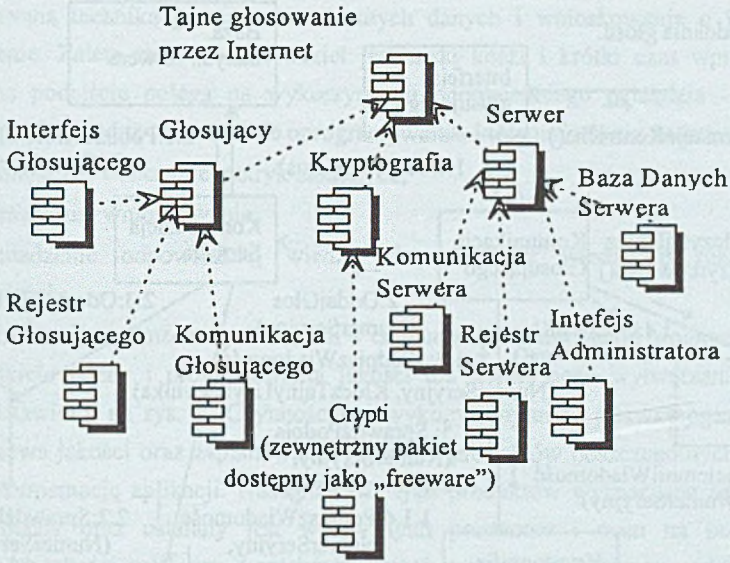
Produktami fazy projektowania są projekt systemu i projekty szczegółowe dla pakietów. W ramach dokumentacji projektu systemu identyfikowane są następujące zagadnienia:

- założenia dotyczące architektury (podział aplikacji na pakiety (ang. packages), interfejsy pomiędzy pakietami, topologia);
- decyzje dotyczące kształtu aplikacji (wybór środowiska implementacji, identyfikacja i uszczegółowienie zagadnień rozproszenia i równoległości, opis warunków granicznych, priorytetów wykonania i innych aspektów istotnych dla projektu);
- projekt interfejsów człowiek-komputer;
- diagramy implementacyjne: diagram komponentów i alokacji podsystemów do procesorów (ang. deployment diagram).

Na rys. 7 przedstawiony jest przykładowy diagram pakietów dla rozpatrywanej aplikacji.

Natomiast w ramach projektu szczegółowego dla każdego z pakietów przedstawiane są:

- implementacyjne diagramy klas z uwzględnieniem architektury - wydzielane są rodzaje obiektów i dodawane są klasy konstrukcyjno-implementacyjne, opis klas obiektów jest uszczegóławiany (typy danych, wywołania metod, cechy charakterystyczne) oraz określana jest reprezentacja związków;
- uszczegółowienie metod - w zależności od typu metody, klasy obiektów i skomplikowania zastosować można diagram stanów (ang. statechart), diagram czynności (ang. activity diagram), diagram współpracy (ang. collaboration diagram) lub pseudokod.

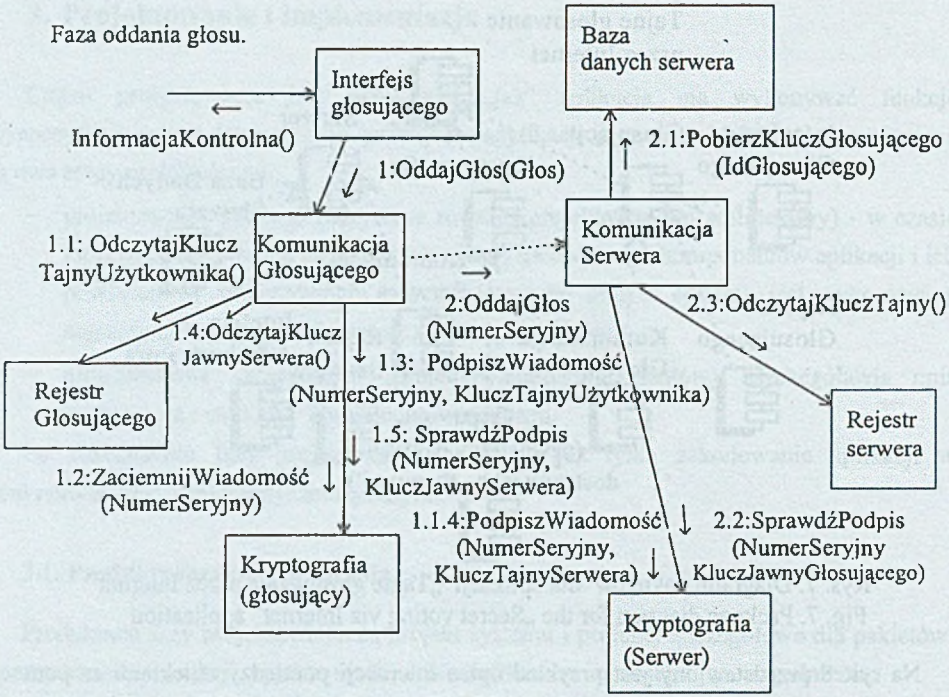


Rys. 7. Diagram pakietów dla aplikacji „Tajne głosowanie przez Internet"
Fig. 7. Package diagram for the „Secret voting via Internet" application

Na rys. 8 przedstawiony jest przykład opisu interakcji pomiędzy obiektami za pomocą diagramu współpracy. Ilustruje on przyjętą metodę uwierzytelniania przekazywanej wiadomości.

3.2. Ocena produktów fazy projektowania i implementacji

Podejście do oceny fazy projektowania i implementacji jest podobne jak w przypadku oceny fazy analizy. W fazie oceny implementacji zapisywane są dodatkowo przebieg i wyniki testów. Na podstawie projektu systemu sprawdzana jest głównie kompletność i adekwatność pakietów i interfejsów, modularność architektury oraz przenośność i rozszerzalność. Na podstawie prototypu interfejsów pomiędzy człowiekiem a aplikacją oceniana jest łatwość użycia i produktywność. Projekt szczegółowy dostarcza informacji na temat kompletności funkcjonalnej i adekwatności komponentów. Tu również można sprawdzić odporność na błędy, jak i przyjęte mechanizmy ochrony. Jakość dokumentacji podlega tym samym kryteriom oceny co w fazie oceny analizy, z uwzględnieniem spójności z poprzednią częścią dokumentacji i śladowości (ang. traceability). Część ocen fazy implementacji ma charakter prognozowania, gdyż niektóre charakterystyki będzie można zmierzyć ostatecznie dopiero w przyjętym środowisku podczas działania aplikacji.



Rys. 8. Diagram współpracy dla oddania głosu w tajnym głosowaniu
Fig. 8. Collaboration diagram for secret voting via Internet

4. System wspomagający iteracyjne wytwarzanie aplikacji

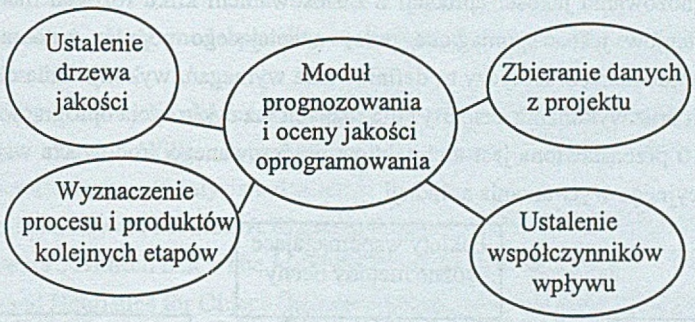
W rozdziałach 2.2 i 3.2 zostały zasygnalizowane procedury oceny poszczególnych faz wytwarzania oprogramowania i metody dokonywania prognoz jakości gotowej aplikacji. W ogólnym przypadku na takie metody mają wpływ:

- zdefiniowane drzewo jakości - zakłada się, że część drzewa jakości reprezentująca charakterystyki i atrybuty jakości dla danego projektu jest stała, niezależna od fazy wytwarzania oprogramowania. Produkty każdej fazy dostarczają innych danych i czasami nie jest dostępna wystarczająca ilość danych do wnioskowania o wszystkich charakterystykach i atrybutach przedstawionych na drzewie jakości;
- rozpatrywana faza projektu - ten sam typ diagramu w różnych fazach wytwarzania oprogramowania będzie podlegał innym procedurom oceny;
- produkty danej fazy - w zależności od tego, jakie są produkty danej fazy, stanowiące dane wejściowe procesu oceny, inaczej będą skonstruowane metody dokonywania pomiarów, oceny i przełożenia na charakterystyki.

Proponowaną techniką gromadzenia istotnych danych i wnioskowania o jakości jest ankietyzowanie. Zaletą zastosowania ankiet jest niski koszt i krótki czas wprowadzenia. Alternatywne podejście polega na wykorzystaniu odpowiedniego narzędzia - aplikacji wspomagającej iteracyjne wytwarzanie oprogramowania, która umożliwia dodatkowo:

- automatyczne obliczanie metryk złożonych;
- automatyczne wnioskowanie;
- zgromadzenie odpowiedniej wiedzy eksperta oraz wiedzy o realizowanych projektach;
- analizowanie zgromadzonych danych w celu udoskonalenia metod prognozowania.

Kontekst pakietu oceny i prognozowania jakości dla iteracyjnego wytwarzania aplikacji został przedstawiony na rys. 9. Czynnościami wykonywanymi w pierwszej kolejności są ustalenie drzewa jakości oraz zaplanowanie procesu i produktów poszczególnych faz, które stanowią dokumentację aplikacji. Następnie dla tych produktów wyznaczane są konkretne pomiary i oceny oraz ustalany jest wpływ tych pomiarów i ocen na poszczególne charakterystyki jakości. W tym kontekście można zastosować różne procedury oceny i prognozowania jakości.



Rys. 9. Kontekst pakietu oceny i prognozowania jakości w systemie wspomagajacym iteracyjne wytwarzanie aplikacji

Fig. 9. Context of the package for evaluation and predicting in the system of iterative application development

Warto zauważyć, że z prognozowaniem jakości związana jest duża ilość niepewności. Istnienie elementów niepewnych (obserwator nie jest pewny co do informacji) i rozmytych (tych, które z natury nie są ściśle ustalone) ma swoje źródło:

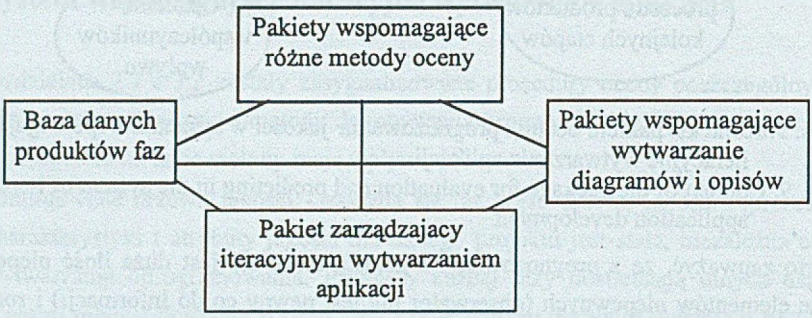
- w specyfice modelu - model ze swojej natury nie przedstawia wszystkich informacji dotyczących danego systemu;
- w niepewności wnioskowania - trudno jest określić, na ile dany aspekt wpływa na ocenę danej charakterystyki;

- w niepewności (rozmytości) prawidłowego rozwiązania - wynikającego z niepełnych wymagań pozafunkcjonalnych podanych przez klienta, wobec czego wizja oczekiwanego rozwiązania zależy od projektanta i oceniającego;
- w niepewności obserwacji - dotyczy stopnia, w jakim oceniający dobrze zrozumiał analizowaną dokumentację.

Ponadto naturalna wydaje się ocena poszczególnych atrybutów i charakterystyk w ankiecie nie według skali binarnej (np. użyteczny albo nieużyteczny), ale według skali wielowartościowej (np. nieużyteczny, słabo, średnio, raczej tak, zdecydowanie użyteczny). Uwzględnienie elementów, których dotycząca wiedza jest rozmyta lub niepewna, prowadzi do modelowania opartego na logice rozmytej. Jest to uzupełniające podejście do takiego, które zakłada eliminację możliwych do uniknięcia niepewności poprzez np. precyzyjne zdefiniowanie pojęć dotyczących kryteriów jakościowych, zapewnienie dokładności opisów systemu i stosowanie standardów.

Na powyższych przesłankach opiera się propozycja prognozowania jakości bazująca na logice zbiorów rozmytych. Obecnie trwają prace nad wytworzeniem prototypu systemu oceny jakości oprogramowania, którego podsystemami są zarówno pakiet pomiarów, jak i pakiety prognozowania jakości aplikacji z zastosowaniem kilku różnych metod. Zadaniem tych podsystemów jest wspomaganie pracy oceniającego poprzez automatyzację wielu czynności pomocniczych. Dotyczy to definiowania wymagań, wykonywania eksperymentów pomiarowych oraz wykonania ekspertyz dla różnych faz cyklu życia oprogramowania.

Na rys. 10 przedstawiona jest architektura zintegrowanego środowiska wspomagającego metodę iteracyjnego wytwarzania aplikacji.



Rys. 10. Architektura środowiska wspomagającego iteracyjne wytwarzanie aplikacji
Fig. 10. Architecture of the environment supporting iterative software development

Składa się ono z czterech klas komponentów: bazy danych zawierającej informacje o projektach zrealizowanych i aplikacjach aktualnie wytwarzanych, pakietów wspomagających tworzenie dokumentacji projektów (np. narzędzi CASE), pakietów kontroli i oceny jakości produktów oraz pakietu zarządzającego, który umożliwia integrację i koordynację prac projektowych.

5. Uwagi końcowe

W pracy zaprezentowano metodę wytwarzania użytkowych aplikacji sieciowych z dokonywaniem iteracji na każdym etapie wytwarzania oprogramowania. Każdy etap dostarcza różnych informacji o aplikacji, zatem wymaga innych procedur wspomagających wnioskowanie o charakterystykach jakości. Zasygnalizowano zastosowanie modeli bazujących na logice rozmytej dla poprawy skuteczności procedur wnioskowania o charakterystykach jakości. Wskazano, jakie produkty częściowe opisywane przez notację UML powinny być brane pod uwagę przy prognozowaniu jakości. Podkreślono potrzebę budowy zintegrowanego środowiska wspomagającego wytwarzanie aplikacji sieciowych. Prace nad prototypem takiego systemu są kontynuowane w ramach grantu KBN.

LITERATURA

1. Aoyama M.: Web-based Agile Software Development. IEEE Software, Nov/Dec 1998, str. 56-61.
2. Bobkowska A.: UML models quality evaluation. Proceedings of ACS'98, Szczecin 1998, str. 558-559.
3. Capability Maturity Model for Software: CMU/SEI-93-TR-24, February 93.
4. D'Souza D., Wills A.: Catalysis - Practical Rigor and Refinement, www.iconcomp.com/catalysis/index.html
5. Fenton N.: Software Metrics. A Rigorous Approach, Chapman&Hall, 1991.
6. Grotehen T., Dittrich K.R.: The MeTHOOD Approach Measures, Transformation Rules, and Heuristics for Object-Oriented Design, raport, grotehen@ifi.unizh.ch.
7. Krawczyk H., Wiszniewski B.: Analysis and Testing of Distributed Software Applications, Wiley & Sons, 1998.
8. Lorenz J., Kidd J.: Object-Oriented Software Metrics, Prentice-Hall, 1994.
9. Rational Software Corporation: UML Notation Guide, www.rational.com
10. Rational Software Corporation - Reaching CMM levels 2 and 3 with Rational's Objectory Process, www.rational.com
11. Rumbaugh J. et al.: Object-Oriented Modelling and Design, Prentice-Hall, 1991.

Recenzent: Dr inż. Przemysław Szmal

Wpłynęło do Redakcji 26 kwietnia 1999 r.

Abstract

Applications are becoming more complex and more distributed nowadays (see Fig.1). Several software development processes are proposed to assure software quality. The common features of them include: use of the object-oriented technology; iterative and incremental development process; strong emphasis on architecture and component-based development; support for requirements management and change control; and configurability of the process. The method for iterative network application development presented in this paper bases on UML - standard object-oriented notation, and a quality tree presented in Fig. 2. It assumes that after each of the development phases there is a phase of inspection which aims to evaluate the products of the given phase and to predict quality of the application, as it is shown in Fig. 3.

Products of the phase of analysis include: use case diagram with template descriptions; class diagram with the data dictionary; and sequence diagrams for normal and exceptional situations. Requirements specification defines the vision of the system (scope and constraints) and non-functional requirements. Requirements specification and analysis influence each other. The functionality of the system is modelled with the diagrams within the scope defined in the specification, and non-functional requirements can be more-detailed expressed while they are based on the analysis diagrams. Examples of the diagrams taken from project entitled ``Secret voting via Internet'' are presented in Fig. 4, 5, 6.

The inspection procedure consist of the following parts:

- data collection from measurement and simple evaluations;
- reasoning about quality criteria and factors;
- generation of advices for quality improvement.

The system design covers the decisions about architecture (packages, interfaces, topology), environment, boundary conditions and priorities, and detailed design refines classes and their behaviour within packages. The examples are presented in Fig. 7 and 8.

The inspections can base on the questionnaire technique, which has low cost and short time of introduction. However a system that supports iterative application development (see Fig. 9, 10) will enable:

- automation of simple activities like measurement,
- better support for reasoning about quality,
- collection of the experts' knowledge as well as knowledge taken from previous projects,
- analysis of the collected data.

The design of integrated environment supporting such a development strategy is proposed.