AUTOMATYKA • 2005 • Tom 9 • Zeszyt 3

Tomasz Serafiński*, Edgar Głowacki**

Śledzenie zmian w oprogramowaniu wywoływanych pojawieniem się nowego wymagania

1. Wstęp

Oprogramowanie jest najbardziej skomplikowanym produktem wytwarzanym przez człowieka. Ta złożoność jest podstawową przyczyną, dla której rozwój oprogramowania jest powolny, a produkt finalny zazwyczaj zawiera liczne błędy. Równie skomplikowany i trudny do przewidzenia jest sam proces wytwarzania oprogramowania – SDP (*Software Development Process*). Jego specyficzną odmianą jest proces mający na celu wprowadzenie zmiany do istniejącego już oprogramowania. Jak pokazuje praktyka, wprowadzenie zmiany jest zazwyczaj bardziej kłopotliwe niż wytworzenie nowego produktu (oprogramowania). W przypadku gdy proces wytwórczy poprzedniej wersji został zakończony – wyjątkowe problemy nastręcza odtworzenie tegoż procesu oraz odtworzenie środowiska wytwórczego, w którym miał miejsce. Dlatego w praktyce zakończenie procesu wytwórczego oprogramowania jest jednoznaczne z wycofaniem się z danego projektu informatycznego.

Najczęściej więc proces wytwarzania oprogramowania jest ciągły. Jego celem są wydania kolejnych wersji produktu – zawierającego mniej błędów oraz ulepszoną funkcjonalność. Mówimy wtedy o rozwoju oprogramowania. Rozwój oprogramowania prowadzi do zwiększania się wartości oprogramowania. Przez zwiększanie wartości rozumiemy ulepszanie istniejącej lub dodawanie nowej funkcjonalności przy jednoczesnym zwiększaniu niezawodności. To może stanowić o podniesieniu konkurencyjności produktu na rynku – co w konsekwencji znajdzie przełożenie na rentowność danego projektu informatycznego.

Czynnikami sukcesu projektu informatycznego są zatem:

- funkcjonalność odpowiadająca potrzebom i wymaganiom użytkownika,
- efektywny proces wytwarzania oprogramowania.

Podstawową miarą jego efektywności jest ilość czynności powodujących dodanie wartości do produktu w stosunku do tych, które wartości nie dodają. Minimalizowanie tych ostatnich jest celem optymalizacji procesu wytwarzania oprogramowania. Ma to szczególne znaczenie, gdy wystąpi konieczność dokonania zmiany. Nawet jeśli proces wytwórczy

* Katedra Informatyki Stosowanej, Politechnika Łódzka, Łódź
** Polsko-Japońska Wyższa Szkoła Technik Komputerowych, Warszawa

przebiega sprawnie i bez zakłóceń, wprowadzanie zmiany do oprogramowania powoduje wzrost ilości czynności nieprowadzących bezpośrednio do dodawania wartości do produktu. Jedną z typowych bolączek inżynierów oprogramowania jest trudność w oszacowaniu zakresu zamiany w chwili, gdy znają jedynie wymaganie (*use-case*). To prowadzi do konieczności podejmowania dodatkowych działań nieobjętych harmonogramem prac oraz konieczności modyfikacji fragmentów oprogramowania, w których zmiany nie były pierwotnie planowane.

Poniżej zaprezentowano metodę rejestracji przebiegu procesu wytwarzania oprogramowania oraz sposoby wnioskowania na podstawie tak zgromadzonych danych o koniecznych do podjęcia w przyszłości działaniach oraz rozprzestrzenianiu się zmiany w produkcie informatycznym. Na wstępie naszych rozważań zakładamy, że proces wytwórczy jest udokumentowany, a w jego kolejnych etapach wytwarzane są modele analityczne i projektowe. Całość artefaktów wytwarzanych w czasie rozwoju oprogramowania jest umieszczana w repozytorium. Dodatkowym założeniem jest to, że przynajmniej jedno wydanie zostało ukończone.

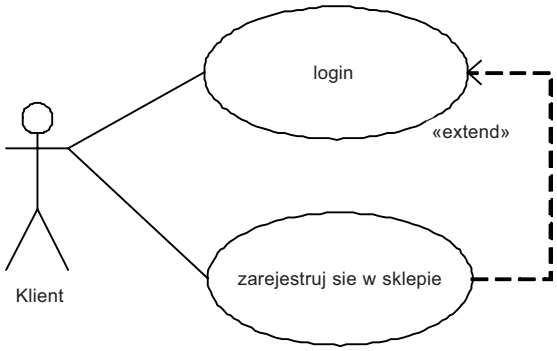
2. Model przypadków użycia

Konieczność zmiany wynika z samego procesu rozwoju produktu informatycznego. Środowisko, w którym pracuje oprogramowanie, podlega nieustannym zmianom. Musi ono nadążać za tymi zmianami. Dlatego mówimy, że zmiana jest nieodłącznie związana z oprogramowaniem [1]. Zmiany te są związane zazwyczaj z postępem technologicznym, zmianami w przepisach i normach, które musi spełniać oprogramowanie lub dodawaniem wartości do produktu. W początkowej fazie wprowadzania zmiany pojawia się więc wymaganie przedstawiane jako przypadek użycia. Ponieważ pojawienie się nowego wymagania jest zazwyczaj związane z analizami strategicznymi, pożądana jest możliwie dokładana wiedza o procesie wytwórczym, aby móc przewidzieć wpływ zmiany na całość oprogramowania. Jest to kluczowe dla oszacowania zasobów potrzebnych do wprowadzenia zmiany, co bezpośrednio przekłada się na koszty. Dlatego szczególnie ważne jest wykonanie odpowiednio szczegółowej dokumentacji już w fazie strategicznej i analitycznej procesu wprowadzania zmiany [2]. Diagram przypadków użycia opisuje dodatkowo aktorów, czyli zewnętrzne interakcje z systemem. Przypadki użycia mogą pozostawać ze sobą w związkach, które również należy udokumentować.

Rysunek 1 przedstawia przykładowy diagram przypadków użycia. Model przypadków użycia jest zazwyczaj uzupełniany przez:

- opis wymagań,
- ograniczenia,
- scenariusze.

Aby mówić o kompletności modelu przypadków użycia, musi on zawierać opis wszystkich przypadków użycia, których implementacja znalazła się w finalnym produkcie.



Rys. 1. Diagram przypadków użycia

Opis wymagań – zawiera formalne wymagania funkcjonalne, jakie musi spełniać oprogramowanie. Mówimy, że wymaganie stanowi rodzaj kontraktu, który musi wypełnić przypadek użycia celem osiągnięcia założonego celu.

Ograniczenia – formalne reguły i ograniczenia, przy których zachodzi przypadek użycia.

Wyróżniamy ograniczenia:

wstępne – opisujące warunki niezbędne, aby dany przypadek użycia mógł wystąpić;
końcowe – pojawią się, gdy przypadek użycia zostanie zakończony;
pośrednie – występujące w czasie, gdy przypadek użycia jest aktywny.

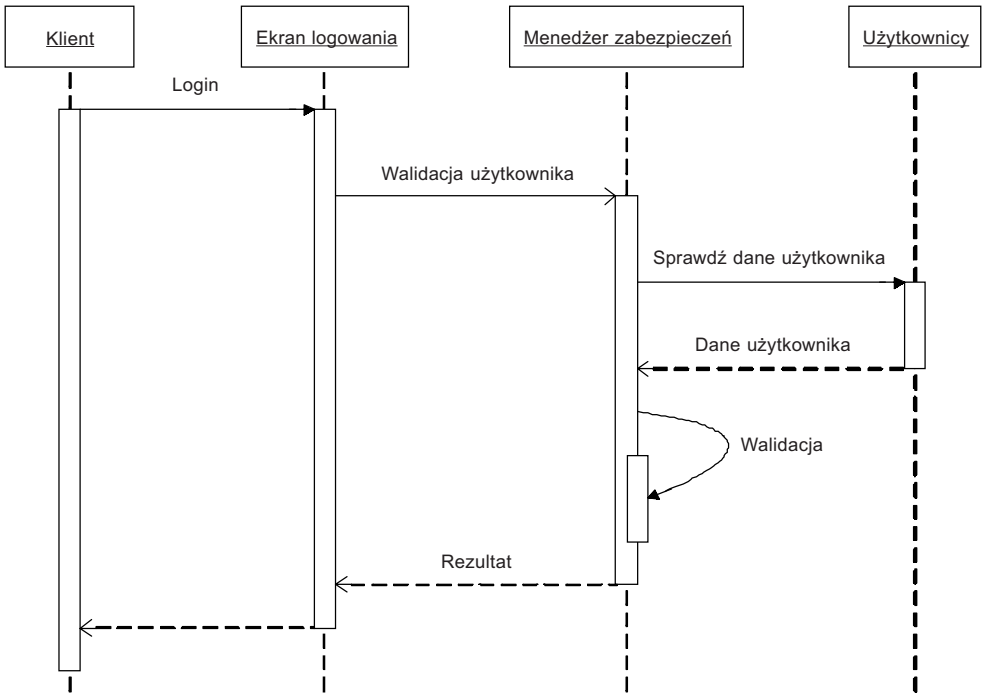
Scenariusze – określają w formalny sposób ciąg zdarzeń, jaki zachodzi w czasie, gdy przypadek użycia jest aktywny. Scenariusze mogą być uzupełniane diagramem sekwencji.

Kolejnym artefaktem, który należy wytworzyć, jest związany model sekwencyjny. Zazwyczaj każdemu scenariuszowi odpowiada jeden diagram sekwencji. Przedstawiona jest na nim kolejność występowania zdarzeń opisanych w scenariuszu. Diagramy te doskonale nadają się do udokumentowania scenariuszy przypadku użycia. Dodatkowo pozwalają wcześnie w fazie analitycznej uchwycić wymagane (konieczność wytworzenia) lub zmieniane obiekty, a także zweryfikować ich późniejsze użycie, co ma szczególne znaczenie podczas fazy projektowej.

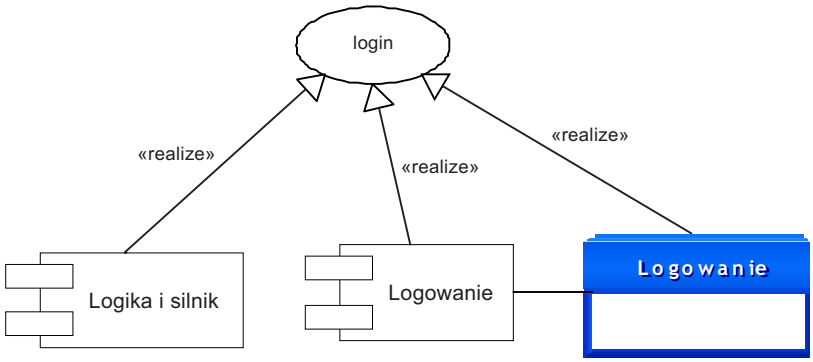
Przykładowy diagramu sekwencji znajduje się na rysunku 2.

Jeżeli model analityczny zawiera statyczny diagram klas, obiekty przedstawiane na diagramie sekwencji powinny reprezentować instancje klas przewidzianych w projekcie.

Całość modelu powinna być jeszcze uzupełniona diagramem implementacyjnym. Stanowi on formalny opis funkcjonalności skonstruowanego systemu. Niesie on informacje o tym, jakie klasy, czy komponenty (artefakty) zostały wytworzone w celu implementacji danego przypadku użycia. Zapewnia to wysoką zdolność do śledzenia zmiany w oprogramowaniu (*traceability*) [3]. Diagram implementacji (rys. 3) pokazuje, które elementy systemu zawierają realizacje poszczególnych przypadków użycia.



Rys. 2. Diagram sekwencji związany ze scenariuszem przypadku użycia



Rys. 3. Przykładowy diagram implementacji przedstawiający komponenty oprogramowania stanowiące realizacje danego przypadku użycia

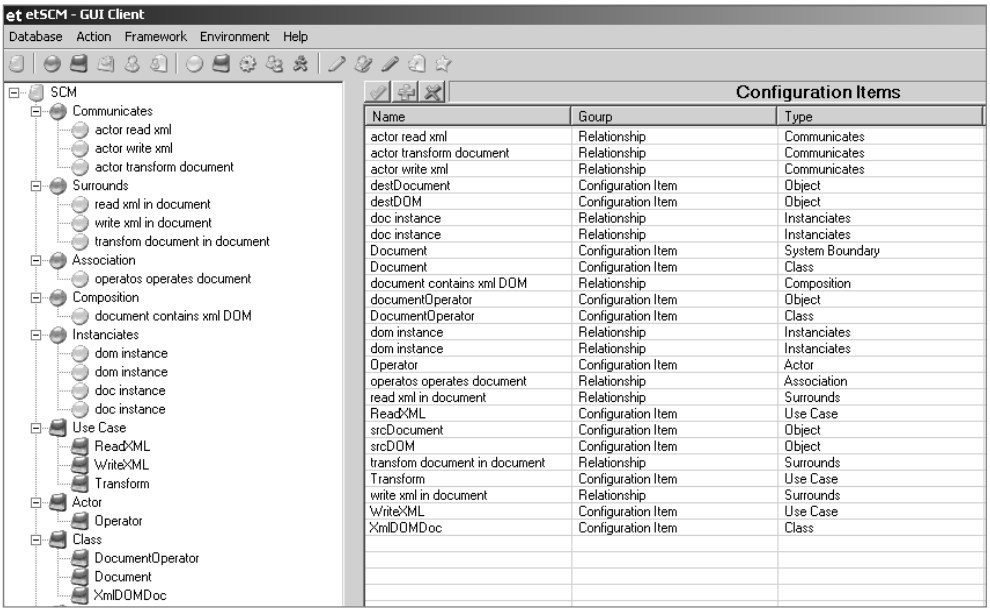
Dzięki wytworzeniu szeregu diagramów analitycznych otrzymujemy model, który we wszechstronny sposób przedstawia nie tylko interakcje systemu ze środowiskiem zewnętrznym, w tym z użytkownikami, ale również pozwala zobaczyć, które elementy składowe produktu finalnego biorą w nich udział. Na potrzeby dalszych rozważań zakładamy, że na

badany proces wytwarzania oprogramowania jest nałożona pewna dyscyplina. Dyscyplina ta polega na ścisłym stosowaniu się do określonych reguł i zasad. Przede wszystkim jednak zakładamy, że dla wymagań systemowych wytwarzane są modele analityczne. Mogą one mieć postać taką, jak przedstawiony wcześniej przykład, ale nie jest to konieczne. Nadrzędnym celem modelu analitycznego jest wytworzenie artefaktów opisujących wymagania i ich późniejsze odniesienie do elementów rzeczywistego systemu. Stanowią one również fundament dla modelu projektowego, który opisuje system w terminologii implementacyjnej i powinien uwzględniać specyfikę środowiska wytwórczego. Wszystkie artefakty powstające w procesie rozwoju oprogramowania są umieszczane w repozytorium i podlegają zarządzaniu konfiguracją oprogramowania (*Software Configuration Management*) [4]. Bardzo ważne jest, aby proces nakładał dyscyplinę wymuszającą spójność modeli znajdujących się w dokumentacji ze stanem faktycznym. Jednym ze sposobów osiągnięcia tego stanu jest taka strategia zarządzania konfiguracją oprogramowania, która wiąże ze sobą w repozytorium modele i projekty artefaktów z ich realizacjami. Dzięki temu możemy w łatwy sposób zlokalizować brakujące elementy, ocenić stan zaawansowania prac (postęp), oraz śledzić przebieg procesu. Dane potrzebne w tym celu będą w praktyce zapisane w repozytorium i mogą w razie potrzeby zostać stamtąd pozyskane. Warunek stanowi jednak zgranie dwóch elementów. Pierwszy z nich to odpowiednio zdefiniowany proces wytwarzania oprogramowania, wymuszający wytwarzanie odpowiednich artefaktów. Drugi, równie ważny, to zarządzanie konfiguracją oprogramowania, które zapewni przechowanie i identyfikowalność wytwarzanych artefaktów, a także pozwoli na zachowanie spójności z projektem. Dzięki temu informacje o przebiegu implementacji przypadków użycia w poprzednich wersjach mogą posłużyć jako dane porównawcze do szacowania zakresu i kosztów zmiany, gdy pojawi się nowe wymaganie (*use-case*) stawiane oprogramowaniu.

3. Rejestracja przebiegu procesu wytwarzania oprogramowania

W niniejszej pacy zakładamy że rejestracja procesu wytwarzania jest zintegrowana z repozytorium, którego oryginalna struktura jest szczegółowo omawiana w [5]. Rozwiązanie to spełnia wymagania stawiane w poprzednim rozdziale niniejszej pracy. Warunkiem skuteczności jest jednak nadzór nad prawidłowością przebiegu całego procesu. Sam proces rejestracji polega na rejestrowaniu wraz z artefaktem informacji o relacjach, w jakie wchodzi z innymi artefaktami. Dodatkowo rejestrowane są zasoby użyte do jego wytworzenia. Założeniem jest pozostawienie użytkownikowi dowolności w wyborze procesu wytwórczego. Dlatego przedstawiany model zawiera jedynie rodzaj szablonu, który musi zostać uformowany poprzez wypełnienie go metadanymi opisującymi definiowany proces wytwarzania oprogramowania. Następnie w trakcie przebiegu samego procesu szablon ten zostaje wypełniony konkretnymi danymi. W praktyce tworzona jest złożona sieć powiązań pomiędzy artefaktami (pozycjami konfiguracji). W miarę postępu procesu dodawane są coraz to nowe pozycje konfiguracji. Ich typ oraz rodzaje rejestrowanych powiązań są definiowane w szablonie wraz z procesem wytwórczym.

Rysunek 4 przedstawia fragment ekranu eksperymentalnej aplikacji realizującej rejestrację danych opisujących przebieg procesu wytwarzania i rozwoju oprogramowania.



Rys. 4. Ekran eksperymentalnej aplikacji umożliwiającej rejestrację danych dotyczących przebiegu procesu wytwarzania oprogramowania. Widać zarejestrowane pozycje konfiguracji o określonych typach

Osoby znajdujące się w zarządzie projektu, jak kierownik projektu lub kierownik systemu zarządzania konfiguracją oprogramowania, powinny zdefiniować odpowiedni szablon [6]. Szablon może zostać zdefiniowany na rozmaite sposoby tak, aby był dostosowany do procesu wytwórczego. Na decyzję dotyczącą rodzaju rejestrowanych pozycji konfiguracji i zależności pomiędzy nimi powinno mieć wpływ to, jakiego rodzaju wielkości (w jaki sposób) zamierzamy mierzyć lub szacować. Oczywiste staje się zatem, że pierwsze definicje szablonu muszą być nieco nadmiarowe i podlegać korekcie po kolejnych wydaniach produktu.

4. Analiza zebranych danych

Analiza zebranych danych jest zadaniem dość trudnym. Składa się na to kilka czynników. Po pierwsze informacje zarejestrowane w trakcie trwania procesu wytwórczego są bardzo złożone, a ich ilość jest ogromna. W praktyce oznacza to, że ich analizowanie bez wspomagania komputerowego jest niemożliwe. Kolejną trudność stanowi fakt, że każdy typ zarejestrowanego artefaktu i każde zarejestrowane powiązanie pomiędzy pozycjami konfiguracji niesie ze sobą inny rodzaj informacji. Dlatego dane, które możemy pozyskać z opisywanego repozytorium, zależą w dużej mierze od zdefiniowanego szablonu. Widać zatem, że jest to jedna z kluczowych czynności, która – aby efekty były zadowalające – wymaga doświadczenia od osoby wykonującej ją. Dlatego zakładamy, że użytkownicy opi-

sywanego tutaj rozwiązania posiadają profesjonalną wiedzę na temat wytwarzania oprogramowania i są w stanie określić potrzebne im w fazie strategicznej informacje. Co za tym idzie, są w stanie prawidłowo zaprojektować szablon, który zostaje wypełniany danymi podczas rozwoju oprogramowania. Jednak ten element jest zależny od specyfiki projektu informatycznego i dlatego nie powinien być narzucany inżynierom. Sprawia to, że podejmowane rozwiązanie jest możliwie generyczne i daje się łatwo zaadoptować do różnych, indywidualnych potrzeb.

Problemem, który pozostaje do rozwiązania, jest dostarczenie inżynierom oprogramowania (użytkownikom proponowanego rozwiązanie) sprawnego i elastycznego narzędzia do pozyskiwania danych z repozytorium. Ideą przyświecającą przy konstruowaniu takiego narzędzia było niewprowadzanie dodatkowych typów bytów czy typów pozycji konfiguracji, które nie byłyby związane z wykonywanymi zadaniami. Dlatego podczas opracowania przyjęto, że mechanizm pozyskiwania danych musi się zasadzać na już istniejących regułach i być całkowicie spójny z modelem analitycznym repozytorium. Aby to osiągnąć, dodano dwa typy pozycji konfiguracji do szablonu rejestracji danych. Jeżeli korzystający z przedstawianego rozwiązania inżynierowie zdecydują się wyzyskać proponowaną metodę, muszą do projektowanego przez siebie szablonu dodać dwa typy pozycji konfiguracji: „granice" oraz relacje „zawiera się w granicy". Różnica polega na tym, że o ile instancje poszczególnych typów są powoływane do życia (rejestrowane są pozycje konfiguracji określonych typów) w czasie działania projektu informatycznego, o tyle instancje typów, granica i zawierania mogą być utworzone w momencie, gdy zachodzi potrzeba pozyskania określonych danych. W szczególności mogą być powoływane w dowolnym momencie, a nawet modyfikowane później. Aby pozyskanie danych było możliwe, potrzebne jest dodatkowo zdefiniowanie uniwersalnej reguły mówiącej, które typy pozycji konfiguracji będą się zawierały w określonych granicach. Czyli w rzeczywistości, pomiędzy którymi pozycjami konfiguracji a instancją określonej granicy zostanie zarejestrowane powiązanie typu „zawiera się w granicy". Należy zwrócić uwagę, że użytkownik może zdefiniować wiele typów granic oraz powoływać do życia wiele instancji jednego typu granicy. W efekcie, w momencie analizy danych sprawdzamy jedynie, które pozycje konfiguracji zostały objęte odpowiednimi granicami. Dzięki temu możemy śledzić, np. przebieg rozwoju oprogramowania w czasie lub pozyskać informacje, które pozycje konfiguracji uległy zmianie w momencie wprowadzenia i zaimplementowanie nowego przypadku użycia reprezentującego wymaganie systemowe. Efekty działania opisanego mechanizmu mogą zostać nakładane na siebie, np. na podzbiór pozycji konfiguracji powstałych z zastosowania odpowiedniej granicy zostanie nałożony kolejny filtr stanowiący zbiór pozycji konfiguracji z innej granicy (innych reguł). Dzięki temu otrzymujemy sprawne i elastyczne narzędzie, które do modelu repozytorium nie wprowadza nowych bytów, a jedynie wykorzystuje już istniejące.

5. Wnioski i dalszy rozwój

Proponowane rozwiązanie może być dostosowane do prawie każdego procesu wytwórczego oprogramowania. Szczególne korzyści można odnieść, integrując je ze stosowanym systemem zarządzania konfiguracją oprogramowania. Dzięki temu proponowane rozwiązanie oferuje całą swoją funkcjonalność, jednocześnie nie narzucając użytkownikowi

niczego poza dyscypliną rejestrowania artefaktów. Właściwe korzyści z zarejestrowanych danych odnosimy dopiero wtedy, gdy możemy je porównać z ich poprzednimi wersjami. Dlatego w praktyce, rozwiązanie może przynieść korzyści dopiero przy drugiej iteracji danego cyklu życiowego oprogramowania. Pierwsza iteracja – najczęściej wytworzenie pierwszej wersji oprogramowania – jest niejako cyklem rozruchowym. Kolejne iteracje będą mogły być analizowane z rosnącą dokładnością, ponieważ ilość danych porównawczych będzie się zwiększała. Ma to znaczenie, jeśli uwzględnimy fakt, że najczęściej zostaje wykonane kilka iteracji, zanim dojdzie do pierwszego wydania produktu informatycznego. Oznacza to, że w praktyce już prace nad wytworzeniem pierwszego wydania produktu mogą być wspierane przez proponowane narzędzie.

Dalszy rozwój prac kierowany jest na rozwijanie metod pozyskania danych. Prowadzone są również eksperymenty w celu wytworzenia definicji procesu wytwórczego, który poprzez używanie odpowiednich typów pozycji konfiguracji byłby w stanie zapewnić, do pewnego stopnia, możliwość ekstrapolacji swojego przebiegu. Dodatkowo proces ten musi spełniać wymagania wydajnościowe takie, aby zysk z możliwości przewidzenia zakresu i kosztów zmiany nie został pochłonięty przez dodatkowe czynności – nie dodające wartości do produktu.

Literatura

[1] Larman C.: *Agile and Iterative Development: A Manager's Guide.* Addison Wesley, August 2003
[2] Onoma A.K., Komuro M., Suganuma H.: *Object-Oriented Software Engineering – A Use Case Driven Approach.* Addison-Weseley, 1992
[3] Głowacki E., Serafinski T., Subieta K.: *Dependencies Supporting Assessment of the Scope of Software Change.* Communications, 31st Conference on Current Trends in Theory and Practice of Computer Science, Sofsem 2005, ISBN 80-969255-4-7
[4] Hass A.: *Configuration Management Principles and Practice.* Addison-Wesley, January 2003
[5] Serafiński T., Głowacki E., Subieta K.: *Zależności pomocne przy szacowaniu zakresu zamiany w oprogramowaniu.* Półrocznik AGH Automatyka, t. 8, z, 3, 2004, ISSN 1429-3447
[6] Humphrey W.: *Introduction to the Team Software Process.* Addison-Wesley Longman, 2000