Rozdział IV

Zwinna specyfikacja wymagań

Lech Madeyski, Marcin Kubasiak
Wydziałowy Zakład Informatyki, Wydział Informatyki i Zarządzania
Politechnika Wrocławska
{Lech.Madeyski, Marcin.Kubasiak}@pwr.wroc.pl

We want to restore a balance. We accept modeling, but not in order to file some diagram in a dusty corporate repository. We accept documentation, but not hundreds of pages of never-maintained and rarely used tomes. We plan, but recognize the limits of planning in a turbulent environment."

--- Jim Highsmith

Streszczenie

Rozdział dotyczy propozycji sprawnej i wykonywalnej[1] specyfikacji wymagań osadzonej na gruncie metodyki XP (ang. eXtreme Programming). Przedstawiono i porównano zarówno stosowane wcześniej przez autorów podejście do specyfikacji wymagań wykorzystujące przypadki użycia (ang. use cases), jak i nowe podejście bazujące na kartach wymagań, szkicach ekranów, testach akceptacyjnych oraz otwartym formacie przechowywania wymagań - XML. Jednym z rezultatów proponowanego podejścia jest uzyskanie wykonywalnej specyfikacji wymagań. Testy akceptacyjne okazały się niemal odpowiednikiem wykonywalnych przypadków użycia. Fakt ten, wydaje się mieć duży wpływ na efektywność procesu wytwarzania oprogramowania.

[1] Przez wykonywalną specyfikację wymagań nie rozumiemy specyfikacji zapisanej w wykonywalnym języku specyfikacji wymagań tylko specyfikację, która umożliwia ciągłą i automatyczną walidację tworzonego systemu w kontekście jego zgodności z wymaganiami klienta oraz dostarcza informację zwrotną na temat postępów w realizacji systemu. Wykonywalność specyfikacji wynika z możliwości wykonania testów akceptacyjnych, zapisanych w postaci skryptów XML, przez odpowiednie narzędzie. W proponowanym podejściu testy akceptacyjne nie są opracowywane na podstawie specyfikacji wymagań, lecz są jej integralną częścią (ang. *Test-Driven Requirements Specification*). Jest to analogia do programowania/projektowania poprzez pisanie testów (ang. *Test-Driven Development*) przeniesionego na grunt specyfikacji wymagań.

1. Wstęp

W czerwcu 2002 roku na Wydziale Informatyki i Zarządzania Politechniki Wrocławskiej pojawił się pomysł realizacji serwisu e-Informatyka (www.e-informatyka.pl). Przedsięwzięcie to miało na celu przygotowanie i wypromowanie liczącego się, w pełni profesjonalnego czasopisma o charakterze naukowym i popularno-naukowym, które będzie skupiało polskie środowisko informatyczne. Założeniem było, iż czasopismo będzie posiadało zarówno formę drukowaną, jak i elektroniczną.

Wyłoniona została wówczas grupa sześciu studentów, której zadaniem było przygotowanie wizji systemu oraz, na tej podstawie opracowanie dokumentu specyfikacji wymagań tak, aby począwszy od października 2002 roku, system mógł być realizowany przez większą, sześćdziesięcioosobową grupę studentów w ramach przedmiotu „Technologie Biznesu Elektronicznego" prowadzonego przez jednego z autorów.

W niniejszym rozdziale przedstawimy tradycyjne podejście do specyfikacji wymagań wykorzystane przez studentów na początku przedsięwzięcia, wpływ tego podejścia na dalszą realizację projektu oraz nowe podejście do specyfikacji wymagań, które autorzy stosują realizując drugie wydanie serwisu e-Informatyka.

2. Tradycyjne podejście do specyfikacji wymagań

Specyfikując wymagania dla pierwszego wydania serwisu e-Informatyka, studenci wykorzystali popularną technikę przypadków użycia (ang. *use cases*). Aby zapewnić spójność i kompletność opisów poszczególnych przypadków użycia, studenci rozpoczęli pracę od ustalenia standardowego szablonu opisu pojedynczego przypadku użycia. Szablon ten zawierał następujące pola:

- aktorzy;
- cele;
- priorytet;
- warunki początkowe;
- scenariusz bazowy;
- scenariusze alternatywne;
- wyjątki;
- warunki końcowe;
- wymagania niefunkcjonalne.

Następnie, spotykając się wielokrotnie, studenci dyskutowali o możliwej funkcjonalności systemu i na podstawie notatek ze spotkań, indywidualnie dokumentowali kolejne

wymagania wobec systemu. Opis jednego z przypadków użycia, „Zalogowanie się na konto”, przedstawiamy poniżej:

UC #008: Zalogowanie się na konto

Aktorzy pierwszoplanowi:
Zarejestrowany Użytkownik

Aktorzy drugoplanowi:
-

Cele:
- Dostęp do pełnych wersji artykułów.

Priorytet:
Wysoki

Warunki początkowe:
Zarejestrowany Użytkownik nie jest zalogowany.

Scenariusz bazowy:
1. Zarejestrowany Użytkownik wybiera łącze umożliwiające zalogowanie się na konto.
2. System wyświetla formularz logowania. Formularz logowania powinien zawierać następujące pola: login, hasło, łącze umożliwiające przypomnienie zapomnianego hasła (patrz UC #009: Przypomnienie zapomnianego hasła).
3. Zarejestrowany Użytkownik wprowadza w formularzu wymagane dane.
4. Zarejestrowany Użytkownik zatwierdza podane dane.
5. Zarejestrowany Użytkownik przesyła zatwierdzone dane do Systemu.
6. System weryfikuje kompletność i poprawność danych przesłanych przez Zarejestrowanego Użytkownika.

Scenariusz alternatywny nr 1:
6a1. Dane przesłane przez Zarejestrowanego Użytkownika w p. 5 są niekompletne lub niepoprawne.
6a2. System ponownie wyświetla formularz logowania wraz z notką wyjaśniającą przyczynę odrzucenia podanych przez Zarejestrowanego Użytkownika danych.
6a3. Dalej wg scenariusza bazowego p. 3.

Warunki końcowe:
Zarejestrowany Użytkownik jest zalogowany.

Wyjątek nr 1:
Jeżeli w p. 4 Zarejestrowany Użytkownik nie zatwierdza podanych w p. 3 danych, wówczas:
1. Wykonanie przypadku użycia jest przerywane.

Zarejestrowany Użytkownik nie jest zalogowany.

Wyjątek nr 2:
Jeżeli w p. 6 System nie może zweryfikować kompletności i poprawności danych przesłanych przez Zarejestrowanego Użytkownika w p. 5, wówczas:
1. System wyświetla informację o zaistniałym problemie.
2. Wykonanie przypadku użycia jest przerywane.

Zarejestrowany Użytkownik nie jest zalogowany.

Wymagania niefunkcjonalne:
- Wygenerowanie strony logowania przez serwer nie powinno trwać dłużej niż 1 sekundę.

W wyniku podjętych prac, niedługo po rozpoczęciu semestru, gotowy był 80-stronicowy dokument opisujący ponad 40 możliwych przypadków użycia systemu. Opracowany dokument specyfikacji wymagań stał się podstawą pracy dla wspomnianej wcześniej, większej grupy studentów, którzy mieli w ciągu semestru zrealizować system w ramach przedmiotu „Technologie Biznesu Elektronicznego".

2.1. Zagrożenia projektu i podjęte środki zapobiegawcze

Zadanie wydawało się bardzo trudne do wykonania ze względu na następujące fakty:

1. Technologie i narzędzia, które planowano wykorzystać do budowy systemu (platforma Java 2 Enterprise Edition, XML/XSLT, szkielet publikacji Apache Cocoon, Web Services), miały być przez większość realizatorów dopiero poznane w ramach przedmiotu, przy czym literatura w języku polskim nie była dostępna.
2. Studenci biorący udział w projekcie musieli równolegle realizować inne przedmioty, często równie czasochłonne.
3. Bardzo trudno jest zarządzać tak dużą grupą.

Zagrożenie związane z pkt. 1. było trudne do usunięcia, ponieważ ideą przedmiotu, w ramach którego realizowano projekt, jest właśnie poznawanie najnowocześniejszych technologii internetowych w praktyce.

Zagrożenie związane z pkt. 2. było jeszcze trudniejsze do wyeliminowania. Realizatorzy nie mieli żadnego wpływu na zmiany w siatce godzin.

Problem związany z zarządzaniem bardzo dużym zespołem (pkt. 3.) postanowiono rozwiązać w ten sposób, że cały projekt zdekomponowano na 10 podprojektów. Szefowie podprojektów tworzyli rdzeń zespołu (ang. *core team*). Ich zadaniem było kierowanie pracą poszczególnych zespołów oraz zapewnienie komunikacji pomiędzy podprojektami. W tym celu utworzono listy dyskusyjne: jedną dla szefów podprojektów, drugą dla wszystkich uczestników projektu. Listy dyskusyjne sprawdziły się doskonale i były intensywnie wykorzystywane.

2.2. Problemy związane ze specyfikacją wymagań

W trakcie realizacji systemu bardzo szybko okazało się, że:

1. Pomimo dużej szczegółowości opisów przypadków użycia nie precyzują one jednak wszystkich szczegółów wymaganych do realizacji systemu.
2. Opisy przypadków użycia są różnie interpretowane przez różne osoby realizujące system.
3. Duża objętość i formalność dokumentu specyfikacji wymagań zniechęca osoby realizujące system do pełniejszego zapoznania się z jego treścią.

Problemy z pkt. 1. i pkt. 2. spowodowały pojawianie się propozycji ulepszeń i uwag do pierwotnie wyspecyfikowanych wymagań. Nanoszenie poprawek w dokumencie specyfikacji wymagań okazało się bardzo czasochłonne. Z kolei zaniechanie pielęgnacji tego dokumentu oznaczałoby jego stopniową dezaktualizację i wystąpienie problemów w komunikacji między grupami realizującymi poszczególne części systemu (każda z grup miałaby inny obraz systemu).

Pomocą w rozwiązaniu problemu braku jednoznaczności opisów przypadków użycia (pkt. 2.) okazało się sporządzanie obrazujących je szkiców ekranów. Idea ta została jednak rozwinięta i w pełni zastosowana dopiero podczas dalszych prac nad systemem.

Duża objętość i formalność dokumentu specyfikacji (pkt. 3.) w połączeniu z faktem, iż większa część studentów realizujących system nie brała udziału w procesie specyfikacji wymagań, doprowadziły do sytuacji, w której osoby implementujące system nie miały jego całościowego i spójnego obrazu.

Ponadto pod koniec projektu okazało się, że studenci nie zdołali zrealizować wszystkich wyspecyfikowanych na wstępie przypadków użycia (wg autorów głównym powodem takiego stanu rzeczy była decyzja o zastosowaniu do realizacji systemu nowych, nieznanych większości studentom technologii oraz bardzo duże obciążenie studentów innymi zajęciami). Niezależnie od powodów, z punktu widzenia pierwszego wydania e-Informatyki, można stwierdzić, że czas i zasoby poświęcone na szczegółowe wyspecyfikowanie niezrealizowanych przypadków użycia zostały wykorzystane nieefektywnie.

3. Propozycja nowego podejścia do specyfikacji wymagań

Z edukacyjnego punktu widzenia przedmiot zakończył się sukcesem. Studenci mieli okazję zapoznać się w praktyce z najnowszymi technologiami internetowymi (Java, Apache Cocoon, XML, XSLT, Web Services), pracować wspólnie nad dużym projektem w dużym zespole, praktycznie wykorzystywać narzędzia do wersjonowania oprogramowania – CVS (ang. *Concurrent Versioning System*) czy automatycznej integracji – Apache Ant.

Z biznesowego punktu widzenia pierwsze wydanie e-Informatyki należy jednak uznać za niepowodzenie. Pomimo wysiłku sześćdziesięcioosobowego zespołu otrzymaliśmy zaledwie wykonywalny prototyp systemu, weryfikujący założenia poczynione na etapie specyfikacji wymagań, a nie działający system nadający się do wdrożenia w produkcji.

Aby nie powtórzyć wcześniejszych błędów, podczas prac przygotowawczych do rozpoczęcia realizacji wydania drugiego stwierdzono, że:

1. W projekcie brało udział zbyt wiele osób, co utrudniało komunikację w zespole. W związku z tym dalsze prace będą kontynuowane w znacznie węższym gronie, na zasadach wolontariatu przy jednoczesnej starannej selekcji. Niewielki, ale bardzo dobry zespół jest w stanie zrobić więcej niż zespół duży, ale przeciętny.

2. Zbyt dużo czasu poświęcono szczegółowej specyfikacji wymagań nie sprawdzając jednocześnie w praktyce przyjmowanych założeń (brak informacji zwrotnej o systemie). Autorzy zwrócili swoje zainteresowanie w kierunku **metodyk zwinnych** (ang. *agile methodologies*), takich jak FDD (ang. *Feature Driven Development*), SCRUM a w szczególności w kierunku metodyki XP (ang. *eXtreme Programming*).

Drugie wydanie e-Informatyki realizowane jest przez zespół ośmioosobowy, który stosuje nowe podejście do realizacji projektu bazujące na zwinnej metodyce XP. W dalszej części rozdziału opisane zostanie stosowane przez autorów nowe podejście do specyfikacji wymagań. Kluczowymi elementami tego podejścia są:

- karty wymagań (ang. *user stories*),
- szkice ekranów,
- testy akceptacyjne specyfikowane przed implementacją będące integralną częścią specyfikacji wymagań,
- forum projektu.

3.1. Karty wymagań

Zespół zbiera się w pełnym składzie w pomieszczeniu wyposażonym w tablicę. Dodatkowo należy przygotować plik czystych kartek, kilka pisaków i cyfrowy aparat fotograficzny. Ważne jest, aby każdy z członków zespołu brał aktywny udział w procesie specyfikacji wymagań tak, aby wszyscy mieli pełną i spójną wizję mającego powstać systemu. Unikamy w ten sposób sytuacji z realizacji pierwszego wydania e-Informatyki, kiedy osoby implementujące system nie miały jego całościowego obrazu.

Zespół zaczyna dyskutować o pożądanej funkcjonalności systemu. Na początku należy koncentrować się na funkcjach o największej wartości biznesowej. Nazwę każdej z proponowanych funkcji zespół zapisuje na osobnej karcie wymagania. Karty wymagań zawierają nazwę, właściwą treść i oszacowania [BECK2000]. Po zidentyfikowaniu kluczowych dla danego wydania systemu funkcji zespół przystępuje do dyskusji nad każdą z nich z osobna. Jedna z osób próbuje ująć dyskutowaną funkcję w kilku zdaniach, zapisując je na odpowiedniej karcie wymagania. Następnie odczytuje głośno zapisane zdania. Jeżeli zespół ma zastrzeżenia co do opisu wymagania, wówczas karta jest odrzucana i ta sama lub inna osoba próbuje ponownie opisać dyskutowane wymaganie uwzględniając uwagi zgłoszone przez zespół. Procedura ta powtarzana jest tak długo, aż cały zespół uzna opis wymagania znajdujący się na karcie za właściwy.

Ważne jest, aby opisy na kartach wymagań nie były zbyt szczegółowe (nie chcemy powtarzać błędu polegającego na całościowym, szczegółowym specyfikowaniu wymagań na początku wydania, ponieważ nie mamy pewności, że wszystkie z nich zostaną zrealizowane w danym wydaniu systemu). Zazwyczaj wystarczy kilka zdań. Karty wymagań mają bowiem reprezentować a nie dokumentować wymagania. Przykład karty opisującej funkcję „Zalogowania się na konto” przedstawiamy poniżej:

Nazwa: Zalogowanie się na konto
Treść: Zarejestrowany Użytkownik loguje się na konto podając swój login i hasło.
Oszacowanie: 2 dni

Z kolei karta wymagania opisująca funkcję „Przypomnienia zapomnianego hasła" wygląda następująco:

Nazwa: Przypomnienie zapomnianego hasła
Treść: Jeżeli Zarejestrowany Użytkownik zapomni swoje hasło, może poprosić System o jego przypomnienie. Zarejestrowany Użytkownik podaje swój adres e-mail a System wysyła hasło Zarejestrowanego Użytkownika na wskazany adres.
Oszacowanie: 2 dni

Na podstawie powyższych opisów nie można jeszcze przystąpić do realizacji systemu. Karty wymagań pozostawiają zbyt wiele pytań bez odpowiedzi, np. jak powinien zareagować system, gdy podczas logowania użytkownik poda zły login, czy też jak powinien zareagować system, gdy podczas przypominania zapomnianego hasła współpracy odmówi serwer pocztowy? Opis przypadku użycia uwzględniałby odpowiedzi na takie pytania. Ale karty wymagań są tylko jednym z trzech elementów specyfikacji wymagań. Uzupełniamy je o szkice ekranów i testy akceptacyjne.

W dalszej części rozdziału postaramy się wykazać, że karty wymagań, szkice ekranów i testy akceptacyjne są efektywniejszą metodą specyfikacji wymagań niż przypadki użycia.

Warto w tym miejscu podkreślić jeszcze jeden fakt. Mimo, że karty wymagań nie precyzują wszystkich szczegółów opisywanej funkcjonalności systemu, to dają programistom dobre pojęcie odnośnie tego, co ma zostać zrealizowane. Mała ziarnistość funkcji opisywanych przez karty wymagań umożliwia programistom dokładne szacowanie czasu ich realizacji. Jest to szczególnie istotne, gdyż dokładne szacunki programistów umożliwiają z kolei precyzyjne planowanie (więcej na ten temat można znaleźć w [BECK1999, JEFF2000, BECK2000] pod hasłem „Gra w planowanie", „Planowanie wydania", „Planowanie iteracji"). Tak więc karty wymagań pełnią również bardzo istotną rolę w procesie planowania.

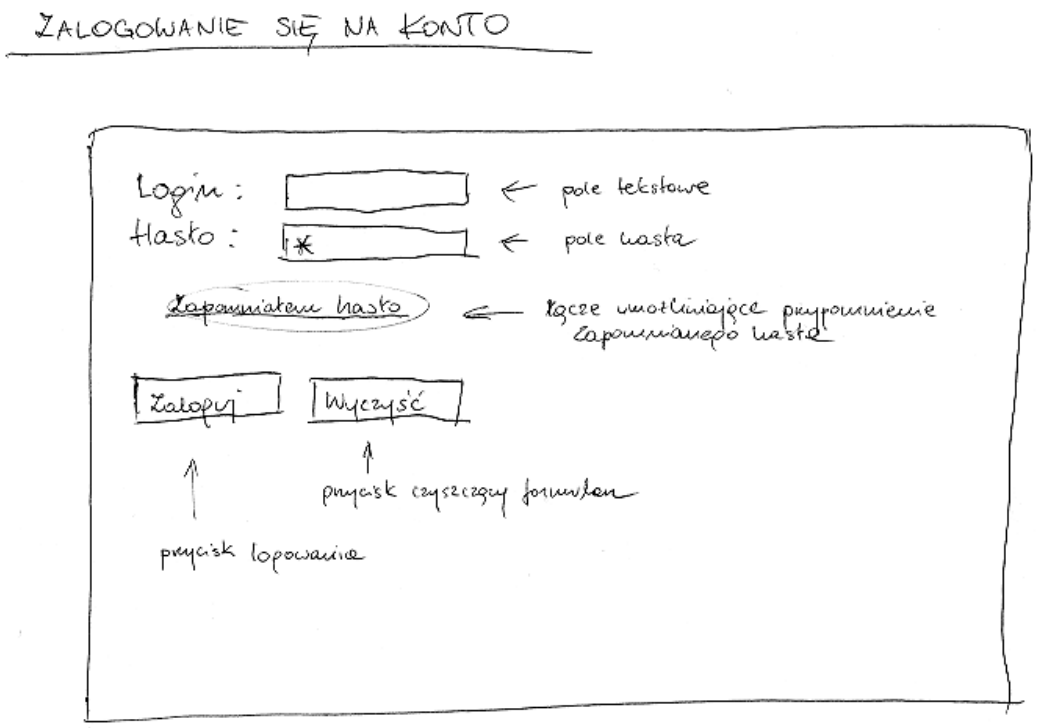
3.2. Szkice ekranów

Wspomnieliśmy już wcześniej o problemie, który pojawił się w trakcie realizacji pierwszego wydania e-Informatyki i dotyczył rozbieżności w interpretacji opisów

przypadków użycia. Niejednoznaczność interpretacji wymagań prowadziła do nieporozumień, co w efekcie negatywnie wpłynęło na efektywność pracy całego zespołu. W zaistniałej sytuacji pomocną techniką okazało się sporządzanie szkiców ekranów obrazujących przypadki użycia systemu.

Technikę tę stosujemy obecnie jako integralny element procesu specyfikacji wymagań. Równolegle z formułowaniem opisu funkcji systemu na karcie wymagania, zespół szkicuje na tablicy ekrany związane z daną funkcją. Po zatwierdzeniu opisu dyskutowanej funkcji i związanych z nią szkiców ekranów, szkice ekranów zapisywane są za pomocą cyfrowego aparatu fotograficznego.

Szkice ekranów nie powinny być zbyt szczegółowe – obrazują jedynie najważniejsze elementy funkcjonalne wynikające z opisu słownego na karcie wymagania. Przykład szkicu ekranu dla karty „Zalogowanie się na konto” przedstawiamy poniżej:



Rys. 1. Szkic ekranu związany z kartą wymagania „Zalogowanie się na konto”

Zauważmy również, że szkice ekranów dają nam pierwszą, podstawową informację zwrotną o tworzonym systemie. Co więcej, zbiór wszystkich szkiców stanowi doskonały punkt wyjścia do projektowania właściwego interfejsu użytkownika systemu.

3.3. Testy akceptacyjne

Karty wymagań i związane z nimi szkice ekranów wciąż nie opisują wystarczającej liczby szczegółów wymaganych do implementacji systemu[2]. Świadomie opóźniamy jednak specyfikowanie szczegółów do momentu, w którym są one naprawdę potrzebne.

Wszystkie prace składające się na realizację kolejnego wydania systemu przebiegają w rytmie około 1, 2-tygodniowych iteracji. Każda iteracja rozpoczyna się od wybrania kart wymagań, które będą implementowane w ramach danej iteracji. Przy wyborze kart, z jednej strony kierujemy się ich wartością biznesową, z drugiej zaś, oszacowanym wcześniej kosztem realizacji (więcej na ten temat można znaleźć w [BECK1999, JEFF2000, BECK2000] pod hasłem „Planowanie iteracji"). Dopiero po wybraniu kart, które będą realizowane w ramach danej iteracji przystępujemy do uzupełniania ich o wymagane szczegóły. Warto podkreślić zalety takiego podejścia – opóźniając specyfikowanie szczegółów do momentu, w którym są one naprawdę potrzebne, otwieramy się na możliwość uwzględnienia w nich informacji zwrotnej z już zaimplementowanej części systemu. Jest to podejście adaptacyjne charakterystyczne dla zwinnych metodyk wytwarzania oprogramowania. W efekcie dokładnie specyfikujemy tylko to, co rzeczywiście będzie implementowane i ma największą wartość biznesową.

Po wybraniu kart wymagań, które będą realizowane w ramach danej iteracji, zespół ustala szczegóły wymagane do ich implementacji. Następnie do każdej z kart zgłasza się osoba, która będzie odpowiedzialna za wyspecyfikowanie ustalonych dla danej karty szczegółów w formie testów akceptacyjnych. Testy akceptacyjne specyfikujemy w pierwszej kolejności, przed przystąpieniem do realizacji pozostałych zadań związanych z implementacją wybranych kart. Specyfikowanie testów w pierwszej kolejności prowadzi do lepszego zrozumienia funkcji, które mają być zrealizowane oraz odkrycia ewentualnych szczegółów, które nie zostały jeszcze ustalone. Specyfikowanie testów akceptacyjnych powinno przebiegać zatem w miarę sprawnie. W przypadku projektu e-Informatyka wykorzystujemy do tego narzędzia takie jak: JUnit [WWW2003a], HttpUnit [WWW2003b], DBUnit [WWW2003c] i XmlUnit [WWW2003d].

Szczególnie obiecujące, m. in. pod względem sprawności specyfikowania, wydaje się jednak zapisywanie testów akceptacyjnych w postaci skryptów XML [WWW2003e]. Testy akceptacyjne zapisywane w postaci skryptów XML są bardziej zwięzłe i czytelne niż odpowiadający im kod w języku programowania. W związku z tym pisze się je szybciej.

Skrypty XML same w sobie nie reprezentują wykonywalnych programów. Służą raczej do reprezentacji danych wejściowych i wyjściowych oraz definiują na nich odpowiednie asercje, które weryfikowane są dopiero przez odpowiednie narzędzie.

Przykład takiego skryptu, definiującego szczegóły dla karty wymagania „Zalogowanie się na konto", przedstawiamy poniżej:

[2] Implementacja i projekt systemu w metodykach zwinnych są ze sobą ściśle powiązane.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--+
    | Test akceptacyjny dla karty wymagania "Zalogowanie się na konto".
    +-->
<testSpec application="e-Informatyka" baseURL="http://localhost:8080/e-Informatyka/">
    <!--+
        | W pierwszej kolejności specyfikujemy strukturę strony logowania.
        | Strona logowania składa się z formularza logowania i łącza umożliwiającego przypomnienie
zapomnianego hasła.
        | Formularz logowania zawiera z kolei: pole tekstowe 'login', pole hasła 'password', przycisk
'submit' i przycisk 'reset'.
        +-->
    <definePage name="login" url="account/login">
        <form name="form" method="post" action="do-login">
            <input name="login" type="text" set="#{login}"/>
            <input name="password" type="password" set="#{password}"/>
            <input name="submit" type="submit"/>
            <input name="reset" type="reset"/>
        </form>
        <link name="forgotten-password" href="password-reminder"/>
    </definePage>
    <!--+
        | Następnie definiujemy poszczególne testy.
        +-->
    <steps>
        <!--+
            | Test pierwszy sprawdza, czy struktura strony logowania zwracanej przez System zgadza się
ze strukturą zdefiniowaną na początku skryptu.
            +-->
        <suite name="testPageStructure">
            <with page="login">
                <fetch/>
                <verify/>
            </with>
        </suite>
        <!--+
            | Użytkownik logując się musi podać zarówno swój login jak i hasło.
            | Jeżeli dane przesłane przez Użytkownika są niekompletne, wówczas System ponownie
wyświetla formularz logowania wraz z notką informującą o tym, że zarówno pole 'login' jak i 'hasło' są
wymagane.
            | Powyższe warunki sprawdza następujący test:
            +-->
        <suite name="testRequiredFields">
            <with page="login">
                <fetch/>
                <set name="login" value=""/>
                <set name="password" value=""/>
                <submit/>
            </with>
            <with page="do-login">
                <verify contenttype="text/xml">
                    <element xpath="//login[@status='failed']"/>
                    <element xpath="//errors/error[@field='login' &amp;&amp; @type='required']"/>
                    <element xpath="//errors/error[@field='password' &amp;&amp; @type='required']"/>
                </verify>
        </with>
        </suite>
        <!--+
```

```
            | Kolejny test sprawdza sytuację, w której Użytkownik podaje nieistniejący login.
            | System powinien ponownie wyświetlić formularz logowania i poinformować Użytkownika o
zaistniałym problemie.
            +-->
        <suite name="testInvalidLogin">
            <with page="login">
                <fetch/>
                <set name="login" value="invalid-login"/>
                <set name="password" value="password"/>
                <submit/>
            </with>
            <with page="do-login">
                <verify contenttype="text/xml">
                    <element xpath="//login[@status='failed']"/>
                    <element xpath="//errors/error[@field='login' &amp;&amp; @type='invalid']"/>
                </verify>
            </with>
        </suite>
        <!--+
            | Użytkownik może również podać złe hasło do istniejącego konta.
            | Również i w tej sytuacji System powinien ponownie wyświetlić formularz logowania i
poinformować Użytkownika o pomyłce.
            +-->
        <suite name="testInvalidPassword">
            <with page="login">
                <fetch/>
                <set name="login" value="login"/>
                <set name="password" value="invalid-password"/>
                <submit/>
            </with>
            <with page="do-login">
                <verify contenttype="text/xml">
                    <element xpath="//login[@status='failed']"/>
                    <element xpath="//errors/error[@field='password' &amp;&amp; @type='invalid']"/>
                </verify>
            </with>
        </suite>
        <!--+
            | Ostatecznie podejmujemy próbę pomyślnego zalogowania się do Systemu.
            | O pomyślnym zalogowaniu się do Systemu świadczy wartość 'success' atrybutu 'status'
elementu 'login' w odpowiedzi generowanej przez System.
            +-->
        <suite name="testSuccessfulLogin">
            <with page="login">
                <fetch/>
                <set name="login" value="login"/>
                <set name="password" value="password"/>
                <submit/>
            </with>
            <with page="do-login">
                <verify contenttype="text/xml">
                    <element xpath="//login[@status='success']"/>
                </verify>
            </with>
        </suite>
        <!--+
            | Mając definicje wymaganych testów możemy je wykonać.
            +-->
        <run suite="testPageStructure"/>
```

```
        <run suite="testRequiredFields"/>
        <run suite="testInvalidLogin"/>
        <run suite="testInvalidPassword"/>
        <run suite="testSuccessfulLogin"/>
    </steps>
</testSpec>
```

Testy akceptacyjne pozwalają wyspecyfikować wszystkie szczegóły wymagane do rozpoczęcia implementacji systemu, których brak w zwięzłych opisach funkcji na kartach wymagań.

Co więcej, testy akceptacyjne są wykonywalną postacią specyfikacji wymagań w znaczeniu przytoczonym we wstępie rozdziału. Integrując je z procesem budowania (ang. *build*) systemu i często budując system (praktyka ciągłej integracji z XP) będziemy otrzymywać ciągłą i konkretną informację zwrotną o naszych postępach w realizacji projektu.

3.4. Forum projektu

Każdy uczestnik projektu powinien mieć bezpośredni dostęp do wyspecyfikowanych wymagań (wartość komunikacji podkreślana w XP) oraz możliwość łatwej ich modyfikacji (praktyka kolektywnego prawa do zmian zapożyczona z XP). Realizując e-Informatykę, korzystamy w tym celu z forum projektu bazującego na pomyśle Wiki zaproponowanym przez Warda Cunninghama [WWW2003f]. Wiki umożliwia uczestnikom projektu szybkie tworzenie nowych i modyfikowanie istniejących specyfikacji wymagań. Głównymi zaletami Wiki są prostota i kontrola wersji.

Na forum projektu prezentowane są również wyniki procesu ciągłej integracji tworzonego systemu. W przypadku realizowanego przez nas projektu obejmuje to:

- listę ostatnich modyfikacji bazy kodu w CVS,
- logi z procesu budowania systemu: kompilacji, pakowania i rozmieszczania komponentów aplikacji na serwerze,
- raporty z wykonania zautomatyzowanych testów akceptacyjnych i jednostkowych.

4. Porównanie obu podejść do specyfikacji wymagań

Zestawmy obie techniki specyfikacji wymagań – przypadki użycia oraz proponowane podejście:

1. Zarówno przypadki użycia jak i karty wymagań mają identyfikującą je nazwę (np. „Zalogowanie się na konto”).
2. Przypadek użycia w sposób jawny wyraża cele związane z jego wykonaniem (np. „Dostęp do pełnych wersji artykułów”). Podczas wstępnego specyfikowania funkcjonalności systemu za pomocą kart wymagań również rozważamy cele użycia

proponowanych funkcji. Funkcje dla których nie można określić celów użycia nie są brane pod uwagę.

3. Przypadki użycia mają określony priorytet. Podobnie karty wymagań mają określoną wartość biznesową, którą szacujemy na początku każdej iteracji (ponieważ wartość biznesowa karty może zależeć od już zrealizowanej funkcjonalności systemu).
4. Przypadki użycia definiują warunki początkowe i końcowe (np. warunek początkowy: „Zarejestrowany Użytkownik nie jest zalogowany"). Również testy akceptacyjne mogą zawierać kroki weryfikujące warunki początkowe i końcowe, np. dla przykładowego warunku początkowego sekwencja kroków w teście akceptacyjnym byłaby następująca: (1) zalogowanie Zarejestrowanego Użytkownika; (2) próba ponownego logowania, która powinna zakończyć się błędem.
5. Przypadki użycia definiują scenariusze (bazowy, alternatywne i wyjątkowe) składające się z sekwencji interakcji pomiędzy zidentyfikowanymi wcześniej aktorami a rozważanym systemem. Testy akceptacyjne specyfikuje się na tej samej zasadzie. Symulują one działania aktorów i weryfikują zachowania systemu w tych samych sytuacjach, co scenariusze przypadku użycia.
6. Przypadki użycia, oprócz wymagań funkcjonalnych, umożliwiają również specyfikowanie wymagań niefunkcjonalnych (np. „Wygenerowanie strony logowania przez serwer nie powinno trwać dłużej niż 1 sekundę). W testach akceptacyjnych także można wyrażać proste wymagania niefunkcjonalne. Wymagania niefunkcjonalne nie związane z żadną kartą wymagania specyfikujemy w osobnym teście akceptacyjnym.

Powyższe porównanie pokazuje, że w sensie funkcjonalnym karty wymagań połączone z testami akceptacyjnymi odpowiadają przypadkom użycia. Również wysiłek związany z przygotowaniem przypadków użycia i testów akceptacyjnych jest podobny, przy założeniu, że do specyfikowania testów akceptacyjnych wykorzystujemy skrypty XML.

W czym zatem tkwi różnica? Dlaczego proponujemy i w swojej praktyce stosujemy zaprezentowane w tym rozdziale podejście do specyfikacji wymagań?

4.1 Zalety zwinnego specyfikowania wymagań

Zdaniem autorów proponowane podejście wydaje się efektywniejsze niż technika przypadków użycia z następujących powodów:

1. Testy akceptacyjne są wykonywalną postacią specyfikacji wymagań. Możemy zintegrować je z procesem budowania systemu i, stosując praktykę ciągłej integracji znaną z XP, będziemy otrzymywać ciągłą i konkretną informację zwrotną o postępach w realizacji projektu.
2. Ponieważ testy akceptacyjne są niemal odpowiednikiem wykonywalnych przypadków użycia, zatem praca nad jednym a potem nad drugim artefaktem

osobno wydaje się autorom marnowaniem zasobów. Mogłoby się wydawać, że opisy przypadków użycia są bardziej czytelne, ale specyfikacja testu akceptacyjnego w postaci skryptu XML może być równie czytelna i, co więcej, w prosty sposób transformowana za pomocą XSL (ang. *eXtensible Stylesheet Language*) na dokumentację systemu w dowolnej, dostosowanej do potrzeb użytkownika postaci i formacie (np. HTML czy PDF).

3. W proponowanym podejściu, przy porównywalnym wysiłku, oprócz specyfikacji wymagań otrzymujemy jednocześnie obszerny, budowany od samego początku projektu zestaw zautomatyzowanych testów akceptacyjnych. W efekcie pokrycie systemu testami będzie lepsze niż w podejściu klasycznym. Dodatkowo automatyzacja testów akceptacyjnych znacznie przyspiesza sam proces testowania. Dzięki temu pozostaje więcej czasu na ręczne testowanie, co w przypadku średnich i dużych systemów jest niezmiernie czasochłonne. Ręcznie testujemy głownie to, czego nie da się zautomatyzować (np. spójność graficzną poszczególnych stron portalu).
4. Sytuacja, w której wymagania nie ulegają zmianie i są jasno określone na początku projektu należy do rzadkości. W przypadku klasycznego podejścia, każda zmiana wymagań sprawia, że modyfikowane są dokumenty specyfikujące przypadki użycia i istniejące testy akceptacyjne. W przypadku zaproponowanego podejścia i realizowanych bądź już zrealizowanych funkcji systemu, wymagania zawarte są w testach akceptacyjnych, przypadki użycia nie są potrzebne a co za tym idzie zakres modyfikowanych dokumentów jest mniejszy. Dodatkowo rozbudowany zestaw automatycznych testów gwarantuje wychwycenie ewentualnych problemów, które mogą pojawić się po wprowadzeniu zmian w już zaimplementowanej części systemu. W przypadku funkcji, które dopiero będą realizowane, testy akceptacyjne jeszcze nie istnieją - nie ma potrzeby modyfikowania jakichkolwiek dokumentów.
5. W proponowanym podejściu szczegółowe wymagania specyfikowane są dopiero wtedy, gdy są potrzebne – nie wcześniej. Specyfikujemy tylko to, co jest naprawdę potrzebne, ma największą wartość biznesową i powinno być zrealizowane w pierwszej kolejności. Dodatkowo, specyfikując szczegóły dopiero wtedy, gdy są potrzebne - możemy uwzględnić w nich informację zwrotną z już zaimplementowanej części systemu.
6. Karty wymagań z definicji są ograniczone czasem realizacji, co umożliwia dokładne planowanie i kompletną implementację kart wymagań w pojedynczej iteracji. Przypadki użycia są zwykle bardziej gruboziarniste, gdyż są niezależne od ograniczeń czasowych.
7. Specyfikacja wymagań jest łatwo dostępna w formie elektronicznej w formacie XML, może być zdalnie modyfikowana, wersjonowana (z wykorzystaniem np. CVS) i prezentowana w różnych formatach wyjściowych (np. HTML, PDF, PS).

Zdaniem autorów z powyższego porównania wynika, że proponowane podejście do specyfikacji wymagań może być znacznie efektywniejsze - oferując wykonywalną specyfikację wymagań, ciągłą informację zwrotną, obszerny, budowany od początku projektu zestaw testów, jak również pozwalając uniknąć marnowania czasu na czynności

związane z tworzeniem i uaktualnianiem powiązanych ze sobą artefaktów (przypadków użycia i testów akceptacyjnych).

Za szczególnie istotne uważają autorzy również:

1. Zaangażowanie całego zespołu w specyfikację wymagań, co skutkuje pełną i spójną wizją systemu u każdego członka zespołu.
2. Wykorzystanie szkiców ekranów, które zmniejszają niejednoznaczności w interpretacji wymagań, dają pierwszą informację zwrotną o systemie i stanowią punkt wyjścia dla projektowania właściwego interfejsu użytkownika.

4.2. Ograniczenia proponowanego rozwiązania

Oczywiście proponowane przez autorów rozwiązanie nie jest doskonałe. Dostrzegamy następujące związane z nim ograniczenia:

1. Aktualnie pojawiające się pierwsze rozwiązania skryptowe bazujące na XML są funkcjonalnie bardzo proste i nie pozwalają m. in. na łatwe i elastyczne ustawianie i testowanie stanu bazy danych, pełną obsługę formularzy (np. sprawdzanie typów pól obecnych w pobranym formularzu) czy też asercje dotyczące wydajności (np. czasu odpowiedzi serwera na żądanie). W związku z tym, na chwilę obecną, pisanie testów akceptacyjnych wymaga od zespołu bardzo dobrej znajomości narzędzi takich jak: JUnit, HttpUnit, DBUnit czy XmlUnit.
2. Ograniczenia wynikające z osadzenia proponowanego rozwiązania na gruncie metodyki XP (np. nie wszyscy klienci mogą być przygotowani do określania wymagań o ziarnistości wymaganej przez karty wymagań [WAGN2001]).

5. Podsumowanie

Testy akceptacyjne zapisane w XML są wykonywalną postacią specyfikacji wymagań. Można powiedzieć, że są niemal odpowiednikiem wykonywalnych przypadków użycia. Fakt ten warto wykorzystać, by zoptymalizować proces wytwarzania oprogramowania i uzyskać wcześniejszą informację zwrotną nt. realizowanego systemu. Taki jest też zamiar autorów, którzy w przyszłości chcą opisać podstawowe założenia metodyki wytwarzania aplikacji internetowych bazującej na wykonywalnej specyfikacji wymagań za pomocą testów akceptacyjnych. Zdaniem autorów metodyka taka w połączeniu z odpowiednio dobraną architekturą szkieletu tworzonych aplikacji internetowych (ang. *application framework*) pozwoli na bardzo sprawne realizowanie projektów.

Bibliografia

[BECK1999] Beck K., *Extreme Programming Explained: Embrace Change*. Addison-Wesley, 1999.

[BECK2000] Beck K., Fowler M., *Planning Extreme programming*. Addison-Wesley, 2000.

[JEFF2000] Jeffries R., Anderson A., Hendrickson Ch, *Extreme Programming Installed*. Addison-Wesley, 2000.

[WAGN2001] Wagner L., *Extreme Requirements Specification*, Cutter IT Journal, Vol. 14, No. 12, December 2001.http://www.cutter.com/itjournal/itj0112f.html

[WWW2003a] JUnit – szkielet umożliwiający testowanie jednostkowe aplikacji pisanych w Javie, http:// http://junit.sourceforge.net

[WWW2003b] HTTPUnit – rozszerzenie JUnit o możliwość pisania testów akceptacyjnych dla aplikacji webowych, http://httpunit.sourceforge.net

[WWW2003c] DBUnit – rozszerzenie JUnit o możliwość ustawiania i testowania stanu bazy danych, http://dbunit.sourceforge.net

[WWW2003d] XmlUnit – rozszerzenie JUnit o możliwość definiowania asercji dotyczących danych XML, http://xmlunit.sourceforge.net

[WWW2003e] XmlTestSuite – narzędzie umożliwiające specyfikowanie testów akceptacyjnych dla aplikacji webowych w postaci skrytpów XML, http://xmltestsuite.sourceforge.net

[WWW2003f] http://www.c2.com/cgi/wiki?WikiWikiWeb