QUALE

Quale 2/2015

Co jest czym czego – dojrzałość testera

Tomasz Osojca

Testowanie w modelu service delivery

Michał Węgrzyn

Zoom na bramkę jakości we frameworku Scrum – Definition of Done (DoD)

Rafał Stańczak

Testy eksploracyjne w dwóch odsłonach

Spis treści

REDAKCJA

Redaktor naczelny
Karolina Zmitrowicz
karolina.zmitrowicz@quale.pl

Zastępca redaktora naczelnego:
Krzysztof Chytła
krzysztof.chytla@quale.pl

Redaktorzy:
Bartłomiej Prędki
bartlomiej.predki@quale.pl
Michał Figarski
michal.figarski@quale.pl

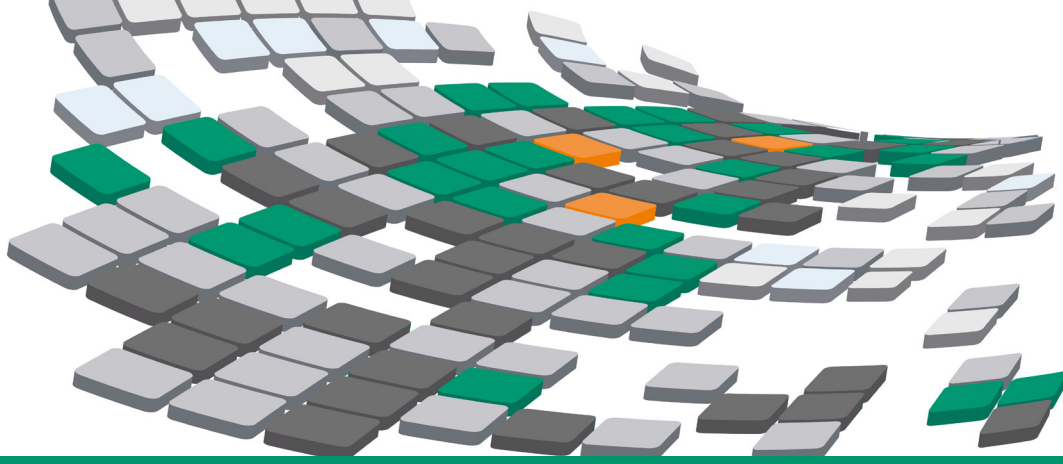
Współpraca:
Tomasz Olszewski
tomasz.olszewski@quale.pl
Piotr Poznański
piotr.poznanski@quale.pl
Zuzanna Gościcka-Miotk
zuzanna.goscicka-miotk@quale.pl

Website:
www.qualemagazine.com (ENG)
www.quale.pl (PL)

Facebook:
http://www.facebook.com/qualemagazine

Spis treści

TestBench



TestBench – The smart solution for all test tasks

imbus TestBench is the working environment for test teams of all sizes, and provides everything that is required of a modern support tool:

Test planning, test design, test automation, test execution and reporting.

TestBench can be integrated seamlessly into your existing tool chain. Its key benefits include intuitive operation, high-performance test functions and a practical model for rights and roles, besides which it also supports structured testing according to the ISTQB® standard.

TestBench provides complete control and transparency in the content, status, progress and results of your software tests – at all times and across all development locations and all releases/versions of your software.

20 years of test management experience are reflected in every aspect of the TestBench.

- Provides fundamental support for all tasks in software testing
- Fully integratable in your test system landscape
- Rapid deployment thanks to various test description methods
- Especially efficient for test design and test specification
- Convenient for test planning and control
- Flexibility in automated test execution
- Extremely good value for money using the renting model

TestBench allows you to concentrate on what is truly important for your company:

- Ensuring that your products and IT systems are of the highest quality through optimum test control throughout their entire life cycle.
- Reducing development costs through efficient working processes and rapid flow of information between teams.
- Shortening your product's time-to-market by eliminating unnecessary bug fixing cycles.

> learn more...

imbus AG, Kleinseebacher Str. 9, 91096 Moehrendorf, Germany
Phone +49 9131 7518-0, testbench@imbus.de, www.imbus.de

Tomasz Osojca

Co jest czym czego* – dojrzałość testera

Egzaminy dojrzałości 2015 za nami (tekst jest pisany w połowie maja), kasztanowce kwitną… a ja zapraszam do analizy Modelu Dojrzałości Testera. Nie procesu testowego, nie organizacji, bo takowe modele istnieją, są dobrze opisane i, mam nadzieję, szeroko stosowane.

Interesuje mnie Model Dojrzałości Testera: testera jako pracownika, członka zespołu wytwarzającego oprogramowanie.

Wprowadzenie

Motywacją do napisania tekstu były nie tylko majowe egzaminy dojrzałości i kasztanowce, ale również moje ostatnie próby znalezienia doświadczonego senior testera do zespołu, którym kieruję. Z żalem muszę stwierdzić, iż bardzo trudno jest znaleźć osobę, która spełniałaby moją definicję dojrzałego, profesjonalnego testera.

Na podstawie powszechnie znanego modelu CMMI opracowałem swój Model Dojrzałości Testera. Skorzystałem również z CIMM [1], mniej znanego modelu, a moim zdaniem nawet bardziej inspirującego, bo pokazującego ewidentne defekty w rozumowaniu ludzi decydujących o kształcie wielu organizacji z branży IT.

Model dojrzałości testera

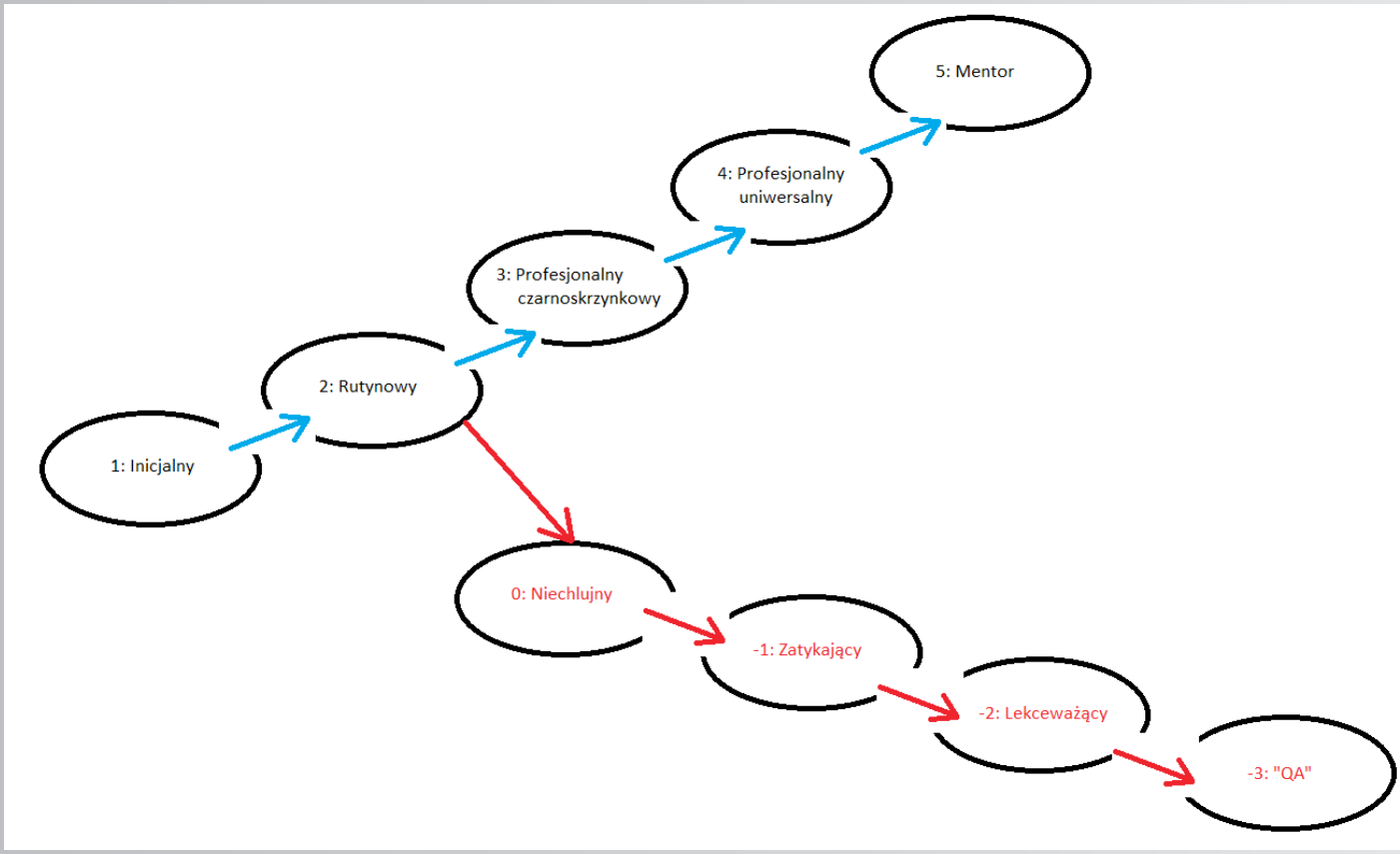
Poziom 1 – Inicjalny

Każdy jest dobrym kandydatem na testera (bo „klikać każdy umie"). Jakkolwiek może to być deprymujące dla „profesjonalistów", prawda jest taka, że w wielu obszarach osoba o świeżym spojrzeniu, bez profesjonalnego doświadczenia, bywa bardzo efektywna. Taki tester charakteryzuje się zaangażowaniem, ciekawością, brakiem nawyków (przede wszystkim tych złych).

Poziom 2 – Rutynowy

„Rutyna" testera na 2. poziomie dojrzałości pochodzi z kilku źródeł. Taką cechę może posiadać:

MODEL DOJRZAŁOŚCI TESTERA



- ekspert dziedzinowy (np. księgowy), który postanowił zostać testerem w projekcie dotyczącym jego specjalizacji;
- osoba, która odpowiednio długo pracuje w jednym projekcie (niekoniecznie jako tester), z jednym produktem lub w jednej branży;
- informatyk – ktoś, kto posiada podstawowe umiejętności i wiedzę (nabytą w trakcie studiów i/lub praktyki zawodowej) z różnych dziedzin inżynierii oprogramowania.

Takie osoby dysponują pewnymi zdolnościami, które mogą być użyteczne w trakcie testowania. Nadal cechują się dużą dozą świeżości i ambicji w nowym dla nich zawodzie. Dzięki połączeniu tych atrybutów mogą osiągać lepsze wyniki od testera inicjalnego.

Poziom 3 – Profesjonalny tester czarnoskrzynkowy

Takim mianem określam testera, który posiada duże doświadczenie, wiedzę i potrafi właściwie zrozumieć zadania oraz rolę

testów w procesie wytwarzania oprogramowania. Zna i stosuje techniki projektowania testów. Testy realizowane przez profesjonalnego testera czarnoskrzynkowego dostarczają jasnej, miarodajnej i wiarygodnej informacji na temat jakości testowanego systemu.

Uwaga! Poziomu 3. nie osiąga się automatycznie po określonej liczbie lat pracy. Istnieje ryzyko, iż tester z kilkuletnim stażem cofnie się do poziomu 0 (patrz *Model Niedojrzałości Testera*).

Poziom 4 – Profesjonalny tester uniwersalny

Jest to profesjonalny tester czarnoskrzynkowy, który dodatkowo posiada umiejętność automatyzacji testów oraz realizacji testów wydajnościowych, zabezpieczeń, użyteczności. Potrafi korzystać z różnorodnych narzędzi, zarówno na poziomie biznesowym, jak i technicznym. Zna i stosuje statyczne metody testowania, poszukuje inspiracji w innych dziedzinach inżynierii i dostosowuje je do potrzeb inżynierii oprogramowania.

Profesjonalny tester uniwersalny posiada szeroki wachlarz umiejętności, a przede wszystkim potrafi świadomie dopasować technikę testów do poziomu ryzyka i typu projektu, w którym uczestniczy.

Poziom 5 – Mentor zespołu testowego

Mentor posiada wszystkie umiejętności wymienione w opisie poziomu 4., a ponadto potrafi przekazywać wiedzę i inspirować zespół do poszukiwania nowych metod i sposobów działania. Jest to osoba łącząca solidne podstawy wiedzy i zdolności z otwartością na nowe, czasami zaskakujące rozwiązania. Zadanie mentora nie polega na zarządzaniu, rozliczaniu z zadań czy sporządzaniu raportów. Powinien on pomagać mniej dojrzałym zawodowo koleżankom i kolegom osiągać kolejne poziomy i strzec ich przed brnięciem w ślepą uliczkę niedojrzałości (patrz niżej).

Model Niedojrzałości Testera

Poziom 0 – Niechlujny

Tester z kilkuletnim stażem. Uczestniczył w szkoleniu, zdobył certyfikat. Niestety, nadal nie rozumie, jak powinno wyglądać profesjonalne testowanie i na czym polega jego – testera – rola w procesie wytwarzania oprogramowania. Taka osoba nie wykorzystuje wiedzy ani technik wypracowanych przez lata w ramach inżynierii testów. Pisze przypadki testowe, zamiast je projektować, szuka błędów zamiast awarii i defektów, „klika", zamiast testować.

Niechlujny tester zazwyczaj okazuje się mniej użyteczny dla projektu niż tester inicjalny, mimo że ma znacznie większe doświadczenie.

DOPISEK

*Tytuł „Co jest czym czego" zapożyczyłem ze *Wstępu do Imaginesko-pii* Śledzia Otrębusa Podgrobelskiego. Z tegoż dzieła pochodzi również poniższy cytat, stanowiący dla mnie nieodmiennie inspirację przy rozwiązywaniu najbardziej nawet złożonych problemów informatycznych:

„twierdzenia (...) o wczuwaniu się jako metodzie poznawczej teoretycznie najsłuszniejszej i metodycznie najbardziej poprawnej wskazują – jeśli uwzględnić niezastąpioną rolę wyobraźni w procesach wczuwania się – na niebezpieczeństwo dysproporcji, rozziewu (hiatus) między wciąż pomnażającą się rzeczywistością a wyobraźnią. Logicznym wnioskiem wypływającym z powyższego musi być zatem pilny postulat powiększenia wyobraźni stosownie do potrzeb poznawczych".

Poziom -1 – Zatykający

To krzykacz, który buduje swoją pozycję w projekcie na konflikcie „tester kontra programista". Czerpie satysfakcję z wytykania innym błędów. Nie stara się dostarczyć użytecznej informacji o jakości i nie dba o zmniejszanie ryzyka produktowego. Zatykający jest dumny z tego, że potrafi wszystko zepsuć. Niestety, często nie zauważa, że jego zachowanie negatywnie wpływa na produktywność zespołu i w konsekwencji na szeroko pojętą jakość finalnego efektu.

Poziom -2 – Lekceważący

Tester, który właśnie nauczył się automatyzacji testów i uważa, że to panaceum na wszystkie problemy. Jedno z najbardziej nieprawdziwych zdań, jakie kiedykolwiek usłyszałem, brzmiało: „Automatyzacja jest najwyższym poziomem umiejętności testera". Dlaczego uważam, że to nieporozumienie? Ponieważ zdecydowanie lepiej jest wykonać ręcznie 10 dobrze zaprojektowanych testów niż 1000 przypadkowych automatycznie. Sztuka polega na opracowywaniu wiarygodnych i miarodajnych testów. Automatyzacja jest rodzajem programowania – zresztą dosyć prostym. Nierzadko osoba, która zdobędzie taką umiejętność, lekceważy „zwykłych" testerów, nieposiadających jej. Zapomina przy tym, że automatyzacja to tylko jeden ze sposobów wykonywania testów – nie zawsze najbardziej efektywny.

Poziom -3 – Quality Assurance Specialist

Tak, to nie pomyłka. Popularny „Qułej" to tester zatykający, który dodatkowo przez siebie i/lub przez organizację, w której pracuje, jest uznawany (nazywany) za specjalistę ds. zapewniania jakości. Przy czym taki QA nie realizuje żadnych zadań związanych z „zapewnianiem", nie usprawnia procesów, nie wprowadza norm, standardów czy choćby dobrych praktyk, nie analizuje danych, nie definiuje rzeczywistych potrzeb użytkowników. Jak napisał Tom Gilb: „False QA is calling your activity 'QA' when in fact you only do testing". Dodałbym od siebie: „Jednocześnie największym błędem testowania jest nazywanie siebie 'QA' w celu ukrycia faktu, iż nie potrafi się przygotowywać dojrzałych, profesjonalnych testów".

Podsumowanie

Maturę zdaje się raz w życiu (czasami w kilku próbach, ale ogólnie rzecz biorąc – raz), natomiast nad dojrzałością w dziedzinie profesjonalnego testowania trzeba pracować nieustannie. Mam nadzieję, że zaprezentowany Model Dojrzałości Testera pomoże w świadomym wkraczaniu na kolejne poziomy zaawansowania w tej trudnej inżynierii.

REFERENCJE

[1] http://en.wikipedia.org/wiki/Capability_Immaturity_Model

AUTOR

Tomasz Osojca

Doświadczony konsultant i trener z zakresu inżynierii testów oraz jakości systemów i procesów. Traktuje „Informatykę" jako „Naukę o Informacji", a nie „naukę o komputerach". Podkreśla, że:

„W projektach IT najważniejsze jest zrozumienie źródeł oraz sposobów przetwarzania informacji w organizacji, a dopiero w drugiej kolejności dostarczenie środków technicznych, czyli kodu źródłowego i serwerów. Innymi słowy, najważniejsza jest wartość użytkowa („Quality in Use" wg ISO/IEC 9126) zbudowana na poprawnej jakości zewnętrznej i wewnętrznej produktu zapewnianej przez efektywność procesów wytwórczych".

W trakcie kariery zawodowej uczestniczył w projektach w roli testera, test managera, automatyzera testów, specjalisty ds. zapewnienia jakości, konsultanta ds. jakości procesów. Zarządzał zespołem specjalistów z zakresu testów jako test manager, kierownik zespołu, dyrektor działu. Realizował projekty zarówno po stronie producenta oprogramowania, jak i organizacji zamawiającej rozwiązanie. Uczestniczył w projektach lokalnych, zespołach rozproszonych geograficznie oraz w projektach międzynarodowych (m.in. dla NATO).

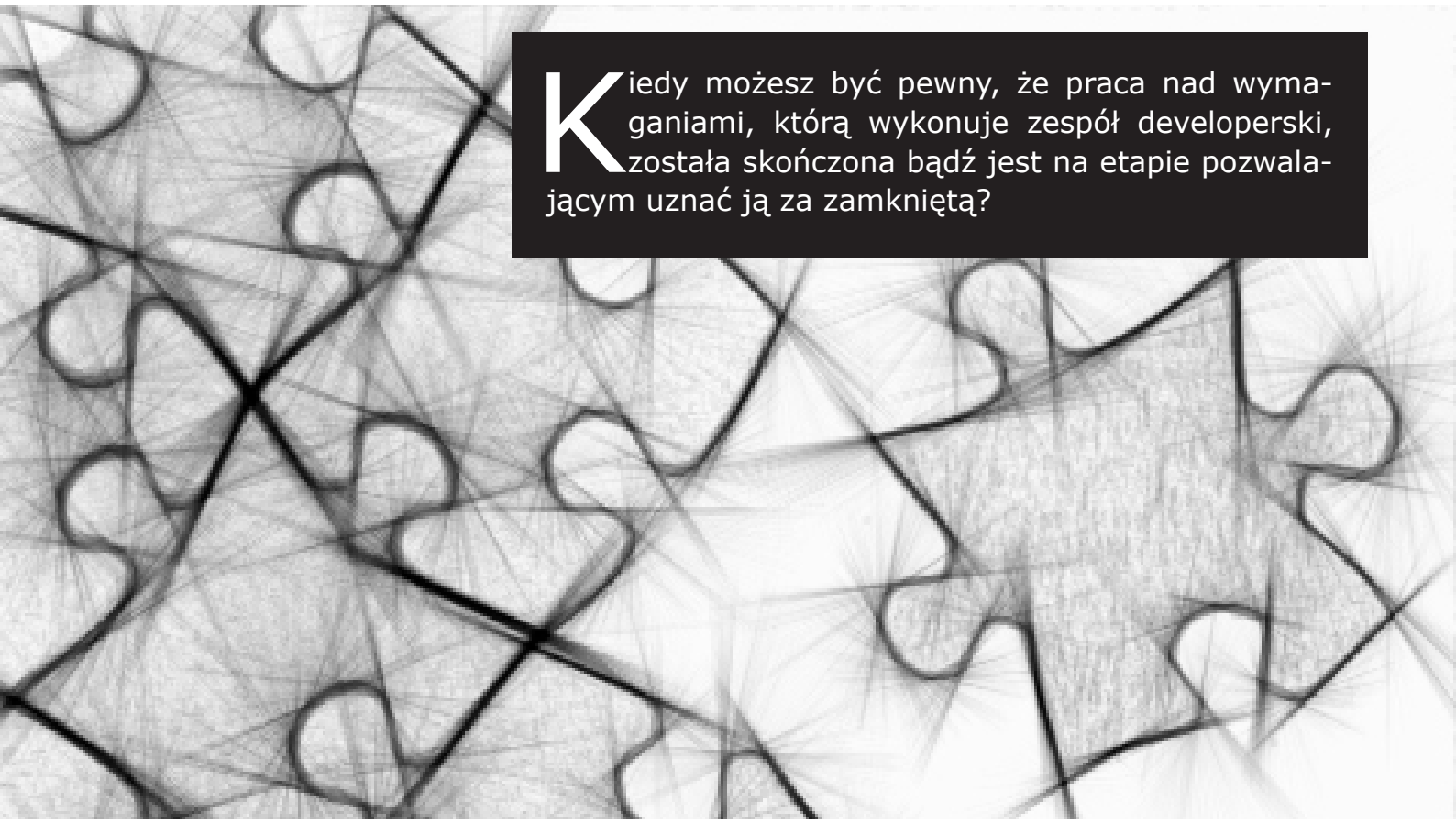
Swoją pasję przekazywania wiedzy realizuje jako trener szkoleń z zakresu testowania i jakości (w tym ISTQB) oraz instruktor ratownictwa górskiego (GOPR).

Prowadzi własną działalność gospodarczą pod nazwą QuICK. Nazwa jest skrótem od Quality by Imagination, Curiosity and Knowledge – co stanowi kwintesencję przymiotów niezbędnych do uzyskiwania odpowienio wysokiego poziomu jakości w branży IT.

Jego idée fixe to wykorzystanie technik sztucznej inteligencji w testach oraz inżynierii jakości systemów informatycznych. Tę ideę realizuje poprzez redagowanie strony www.AI4SE.org – otwartej platformy wymiany myśli o „sztuczno inteligentnej inżynierii oprogramowania".

Rafał Stańczak

Zoom na bramkę jakości we frameworku Scrum – Definition of Done (DoD)



Kiedy możesz być pewny, że praca nad wymaganiami, którą wykonuje zespół developerski, została skończona bądź jest na etapie pozwalającym uznać ją za zamkniętą?

Każdy członek zespołu jest inny i zapewne kieruje się odmiennymi kryteriami w kontekście ukończenia pracy. Jeden developer może pominąć kwestię, która dla drugiego będzie kluczowa. Wobec powyższego widać wyraźnie, że bez ogólnodostępnej, jasnej i przede wszystkim wspólnie stworzonej definicji ukończenia pracy każda funkcjonalność staje się zagrożona i niewykluczone, że nie zostanie dostarczona na koniec trwania iteracji. W takim przypadku konsekwencje mogą być poważne, np. nie otrzymamy od klienta części zakontraktowanej płatności.

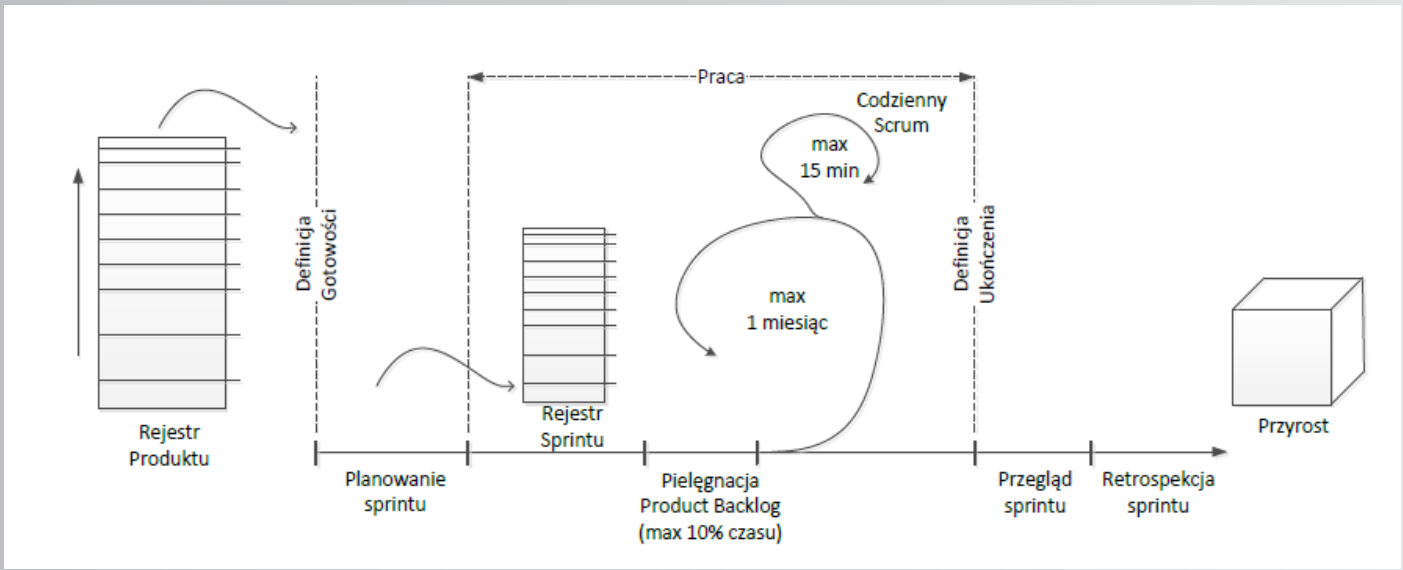
Idea kryjąca się pod pojęciem listy DoD (ang. *Definition of Done*, definicja ukończenia) umożliwia wspólne zrozumienie terminu *Done* (pol. zrobione, ukończone). Definicja ukończenia dostarcza zespołowi niezbędnych informacji, swoistej referencji służącej efektywnej pracy, a co za tym idzie, pozwala na właściwą synchronizację informacji podczas codziennych spotkań. Informacja o statusie ukończonego elementu Rejestru Produktu zawsze niesie za sobą ten sam przekaz. Jak widać na poniższej ilustracji (rys. 1), element zakresu, który spełnił definicję ukończenia, oczekuje tylko na przegląd sprintu.

Poziomy DoD

W zależności od chęci, możliwości i dojrzałości zespołu można definiować ukończenie na wielu poziomach. W artykule skupię się na dwóch najpopularniejszych, choć osobiście uważam, że najważniejsza jest definicja ukończenia dla wydania, ponieważ wówczas nowy inkrement produktu trafia w ręce klienta i/lub użytkownika końcowego. Należą do nich:

- pojedynczy element Rejestru Produktu (user story);

RYSUNEK 1



Rysunek 1. Koncepcja Scrum. Źródło: Krystian Kaczor, *Scrum i nie tylko*, PWN, Warszawa, 2014, s. 114.

PRZYKŁAD

Przykład DoD dla pojedynczego elementu Rejestru Produktu:

- Wszystkie kryteria akceptacji zostały spełnione.
- Kod jest przejrzysty, odpowiednio sformatowany, napisany zgodnie ze standardami programowania.
- Kod przetestowany jednostkowo.
- Kod poddano analizie statycznej.
- Nowo utworzony kod przejrzała doświadczona osoba niebędąca jego autorem.
- Funkcjonalność przetestowana systemowo i eksploracyjnie.
- Narzędzie Continuous Integration nie zwraca błędów integracji.

Przykład DoD dla wydania produktu:

- Przygotowano prezentację nowych funkcjonalności dla interesariuszy projektu.
- Kod przeszedł pomyślnie testy regresji (manualne i automatyczne).
- Kod zapisano w postaci taga i brancha w repozytorium kodu SVN.
- Przygotowano informacje o wydaniu, np. główne zmiany funkcjonalne.
- Dług techniczny nie został powiększony.
- Uzupełniono i zaktualizowano dokumentację funkcjonalną oraz techniczną.
- Kod jest potencjalnie wdrażalny na produkcję.

- wydanie (przykład w ramce).

Jak widać, część elementów listy DoD dla pojedynczego elementu może być wykorzystana z powodzeniem całościowo dla wydania produktu.

Czy lista może się zmieniać??

Definicja ukończenia nie jest statyczną listą tworzoną jednorazowo na początku trwania projektu. Ewoluuje z czasem wraz ze wzrostem umiejętności zespołu i wydajnością organizacji. Podczas trwania iteracji naturalnym momentem, aby dokonać przeglądu aktualnego zakresu listy i przedyskutować ewentualne poprawki, jest retrospektywa.

Oczywiście należy podkreślić, że nie można ingerować w treść definicji w trakcie trwania sprintu.

Czy reguły DoD różnią się pomiędzy zespołami?

Oczywiście tak, jeśli organizacja nie ustali własnego minimalnego zestawu reguł, definicja reguł DoD zależy od produktu i formułuje ją zespół. Gdy powstaje ona „od

zera", zazwyczaj początkowo zawiera kilka najistotniejszych punktów, a następnie – nie częściej niż na koniec każdej iteracji – team stopniowo powinien ją ulepszać. Zespół budując definicję ukończenia od zera, zazwyczaj zaczyna od bardzo prostej definicji kilku najważniejszych punktów, aby później nie częściej niż na koniec każdej iteracji ją ulepszać. Bardzo istotne jest, aby ustalone reguły były zawsze ogólnodostępne i – co niezwykle ważne – właściwie rozumiane przez cały zespół. Nikt nie może mieć wątpliwości, co i jak powinien zrobić. Ta zasada jest silnie powiązana z jednym z trzech filarów frameworku Scrum, a mianowicie *przejrzystością*.

Kilka zespołów pracujących nad jednym produktem

Jeżeli produkt jest złożony i pracuje nad nim wiele zespołów, z praktyki wiem, że często zespoły są dystrybuowane geograficznie, wszystkie teamy muszą koniecznie ustalić wspólną definicję ukończenia, która będzie zapewniała minimalny akceptowalny poziom jakości zintegrowanego rozwiązania.

Oczywiście nic nie stoi później na przeszkodzie, aby poszczególne ekipy stworzyły własne, bardziej wysublimowane DoD dla dostarczanych modułów, komponentów systemu.

Podsumowanie

Odpowiedzmy sobie, zatem na pytanie, dlaczego DoD można traktować, jako bramkę jakości we frameworku Scrum? Otóż utworzenie dedykowanej, dopasowanej do realnych możliwości zespołu i akceptowalnej jakości produktu listy DoD, jest wysoce rekomendowanym rozwiązaniem podczas implementacji dowolnych produktów w podejściu Scrum. To standard, bez którego implementacja zostaje obarczona dużym ryzykiem niepowodzenia. Pierwsza definicja ukończenia jest swoistą bramką jakości, niezmiennym standardem wykonania dla pojedynczych funkcjonalności, ale przede wszystkim przyrostu produktu jako całości.

Ponadto wskazuje developerom rzeczy ważne w procesie wytwórczym, a stara się eliminować straty. Chcemy, aby nasz zespół oraz ich proces wytwórczy był z iteracji na iterację maksymalnie wydajny, aby zawsze niósł za sobą wartość i zakładany poziom jakości. Nie chcemy pozostawiać tego przypadkowi. Możemy to osiągnąć wyłącznie poprzez współpracę pomiędzy zespołem developerskim a właścicielem produktu, głównie w ramach definicji ukończenia i zrozumienia konkretnych praktyk w niej zawartych, które z czasem powinny być ulepszane i wzmacniane.

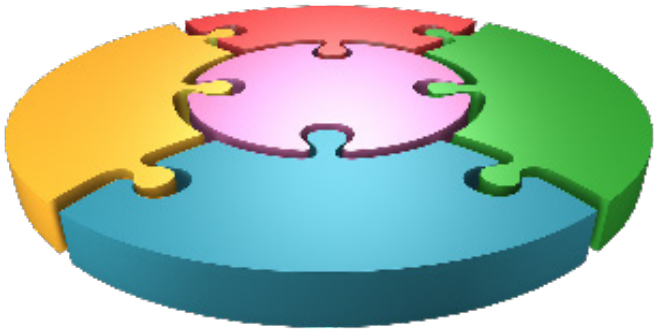
Wówczas członkowie teamu nie będą błądzić, marnotrawiąc cenny czas na tzw. gold platting, czyli próbę ulepszania czegoś, co już jest wystarczająco dobre i mogło być dostarczone bez przekraczania estymacji. Nie popełniajcie tego błędu i jeśli jeszcze nie dysponujecie własną listą, opracujcie ją, uwzględniając możliwości swoich zespołów, a następnie zacznijcie ją ulepszać.

AUTOR

Rafał Stańczak

Konsultant i coach Quality Assurance i Scrum z bogatym doświadczeniem w kilku dużych międzynarodowych projektach z rożnych branż (telekomunikacja, ubezpieczenia, e-commerce). Certyfikowany Scrum Master, założyciel i autor bloga www.scrumdo.pl, w ramach którego dzieli się posiadaną wiedzą ze społecznością polskiego IT.

Aktualnie Scrum Master w dwóch zespołach projektowych oraz Koordynator Quality Assurance portfela produktów w szwajcarskiej firmie z branży bankowej. Biorąc udział w projektach, zawsze koncentruje się na dostarczaniu wysokiej wartości klientom, usuwa przeszkody stojące przed zespołami w drodze do realizacji celu oraz dba o finalną jakość produktu. Członek organizacji Scrum Alliance i Stowarzyszenia Jakości Systemów Informatycznych. Pasjonat i praktyk lekkich metodyk zarzadzania projektami Agile i Lean.



Tom Gilb

Konkurencyjne planowanie Rozdział 1. – Cele

„Cele" to plany, jakie tworzymy, aby uzyskać to, czego chcemy, niezależnie od strategii (środków), jaką możemy później wybrać, aby to osiągnąć.

Wszystkie krytyczne cele można – i trzeba – określić ilościowo

Cel to ocena poziomu wydajności w przyszłości, zazwyczaj poprawa aktualnego stanu.

Fakt, że możemy użyć takich słów jak: *udoskonalony, ulepszony, lepszy*, aby go opisać, to znak, że cele są z natury zmienne i mogą być reprezentowane przez liczby.

Istnieją dwa główne kroki do zamiany na reprezentację ilościową. Najpierw musi zostać określona Skala Pomiarowa. Następnie należy zidentyfikować interesujące punkty na tej skali (takie jak Przeszłość, Cel, Punkt Tolerancji).

Istnieją trzy podstawowe kategorie celów dla wydajności: zdolność do wykonania zadania, oszczędności, atrybuty. Większość zarządzających nie potrzebuje specjalnego instruktażu, aby określić ilościowo dwa pierwsze z nich. Zazwyczaj jednak takie osoby nie potrafią poradzić sobie z drugą stroną niezbilansowanej karty wyników – atrybutami. Opisują one, jak dobrze system (organizacja, proces, projekt, produkt, serwis) działa.

Nie znaleźliśmy żadnych wyjątków: wszystkie cele mogą być mierzone ilościowo.

Gołosłowne cele, „jakość na światowym poziomie" czy „zwiększona zdolność do odpowiedzi na dynamikę rynku" będą niejasne już dla twórcy. Każdy, kto usłyszy lub przeczyta takie sformułowania, zinter-

pretuje je całkiem inaczej. Takie cele są całkowitą stratą czasu.

Używajcie prostego, podstawowego formatu, takiego jak w przykładzie na rysunku 1.

Polityka 1.1: Polityka „Cele czysto ilościowe"

Wszystkie cele dla planów krytycznych będą reprezentowane za pomocą zdefiniowanej Skali Pomiaru i odpowiednich Poziomów Numerycznych.

Dlaczego?

- Zmusimy samych siebie, aby myśleć czysto i głęboko.
- Unikniemy menadżerskich bzdur.
- Pozwoli to nam wziąć odpowiedzialność.
- Możliwe będzie określenie granic odpowiedzialności.

1.2. Ograniczenie: Na dowolnym zadanym poziomie odpowiedzialności wystarczające jest promowanie do dziesięciu krytycznych celów

Istnieje zdecydowanie więcej niż 10 obszarów, które chcielibyśmy usprawnić. Jeśli jednak będziemy próbować zidentyfikować 100 lub więcej zagadnień i pracować nad nimi w tym samym czasie, prawdopodobnie stracimy z oczu bardziej krytyczne cele. Wierzymy i działamy tak, że każdy poziom odpowiedzialności (na przykład menedżer projektów, dyrektor IT, architekt IT) świadomie ogranicza się do garści najbardziej krytycznych celów – na początek. Kiedy zostaną one osiągnięte lub przynajmniej bezpiecznie oddelegowane do innych osób i będą na dobrej drodze do ukończenia, pora zająć się następnymi w kolejce priorytetów.

Na początku najlepiej to zrobić podczas spotkania z ludźmi, którzy muszą wspólnie ustalić, co jest ważne. Prosimy ich, aby wymienili najbardziej krytyczne cele, a następnie zdecydowali, które z nich stanowią pierwszą dziesiątkę (maksimum). Takie spotkanie trwa zazwyczaj około godziny i pozwala dość łatwo dojść do całkiem dobrego porozumienia.

RYSUNEK 1 Jasno wyrażone cele

Reakcja:

Skala: Liczba godzin potrzebna określonym Osobom do Prawidłowej odpowiedzi na dane Sytuacje.
Cel: 24 godziny:

- Kiedy = Koniec następnego roku, Osoby = Poziom kierowniczy, Prawidłowo = Zgodnie z prawem i bez skarg, Odpowiedź = Podjęcie czynności celem rozwiązania Sytuacji.

Nazwanie celów to zaledwie pierwszy etap ich definiowania. Nierozsądnie byłoby spodziewać się poważnego zaangażowania, zanim zostaną one dobrze zdefiniowane i określone ilościowo – tak aby każdy zainteresowany dokładnie wiedział, dlaczego zgodził się nadać im priorytet. Ten proces może zająć resztę dnia ciężkiej równoległej pracy. Wygląda to tak: 3 osoby pracują nad 3 celami, a pod koniec dnia 3 lub 4 zespoły zestawiają przygotowane przez siebie definicje. Takie podejście sprawdza się, stosujemy je podczas „Evo" Startup Planning Week. Osiągamy 10 najważniejszych celów spisanych na jednej stronie – jeśli taką formę przyjmiemy – podczas pierwszego dnia pracy nad projektem.

Bill prywatnie postanowił poświęcić z Kaiem dodatkowy dzień, aby upewnić się, że ich cele są zdefiniowane ilościowo i idealnie dopracowane, zgodnie z życzeniem szefa. We wtorek zaczęliśmy pracować nad 10 strategiami równolegle.

Polityka 1.2: Pierwsze dziesięć krytycznych celów

Podczas pierwszego dnia jakiegokolwiek projektu lub innego większego przedsięwzięcia wybierzemy 10 najbardziej krytycznych celów (na ten dzień). Określimy je ilościowo i zbierzemy na pojedynczej stronie, tak żeby odpowiedni zarząd mógł zatwierdzić całość.

Dlaczego?

Ponieważ cała inna praca (strategia, szacunki) jest niewykonalna bez takiej jasnej podstawy, na której można by się oprzeć. Co w przypadku, gdy musimy zmienić pierwszą dziesiątkę celów?

Zrób to, jeśli pojawi się taka potrzeba, ale nie używaj wymówki, że przyszłe zmiany są nieuniknione, w celu usprawiedliwienia faktu, że na początku wytyczono rozmyte cele.

1.3. Wspieraj przełożonych: Cele na twoim poziomie muszą wspierać cele na poziomie ponad tobą

Ralph Keeney zaproponował doskonały praktyczny pomysł na oddzielenie własnej odpowiedzialności od odpowiedzialności przełożonych czy zespołu, który nas wspiera. Twoje cele (strategiczne) muszą jawnie wspierać osiągnięcie celów osób nad tobą (celów twoich przełożonych, celów „fundamentalnych").

Wszystkie cele, które wspierają twoje strategiczne cele, cele twoich podwładnych czy zespołów, które cię wspierają, są nazywane celami pośrednimi.

Polityka 1.3: Wyjaśnienie odpowiedzialności

Spisana specyfikacja, od razu powiązana z celami, powinna wyjaśnić poziom odpowiedzialności (dla formułowania, zmian i dostarczania rezultatów), jak również wskazywać, co wspiera (zdefiniowane wprost) i przez co jest wspierana (zdefiniowane wprost).

Dlaczego?

Nikt nie powinien mieć wątpliwości w kwestii do swojej odpowiedzialności i jej granic. Ludzie nie mogą mylić celów (priorytetów) ze środkami (znacznie mniejszymi priorytetami).

1.4. Lojalne podporządkowanie: Wszystkie cele twoich podwładnych powinny wprost wspierać twoje cele

Zazwyczaj mamy wiele różnorodnych źródeł, które wspierają nas w osiąganiu celów. Do tego grona należą bezpośredni podwładni, kontraktorzy, konsultanci itd. Każdy byt, który pomaga ci osiągnąć twój cel, nazwijmy zespołem wsparcia.

Zgodzisz się, że wyjaśnienie odpowiedzialności w kwestii tego, w jaki sposób te wszystkie osoby mają przysłużyć się do osiągnięcia twoich celów, jest konieczne. Ma to następujące implikacje:

- Jeśli współpracownicy nie wiedzą, jakie dokładnie są twoje cele, nie mogą efektywnie cię wspierać.
- Jeśli zmodyfikujesz choćby szczegóły swoich celów, powinieneś poinformować o tym wszystkich zainteresowanych, tak aby mogli dostosować się do tych zmian.
- Jeśli zdecydujesz się nie dzielić z zespołem swoimi celami lub sformułujesz je w sposób niejasny i nieprecyzyjny, będziesz odpowiedzialny za brak zdolności współpracowników do efektywnego wspierania cię.
- Jeśli postanowisz powiedzieć ludziom, „co" mają zrobić (przedstawisz środki do osiągnięcia twojego celu), zamiast wybrać sprytniejszą opcję: pokazanie, „jak dobrze to zrobić" (ukazanie twoich celów), skutki tej decyzji spadną na ciebie. Miej to zatem na uwadze i po prostu pozwól zespołowi dojść do „jak"!
- Jeśli jednak wyjaśnisz współpracownikom swoje cele w sposób bardzo przejrzysty, mogą oni (wręcz muszą) być odpowiedzialni za wiele spraw:
  - Powinni się zgodzić (lub w jasny sposób sprzeciwić) na wspieranie niektórych z twoich celów do pewnego stopnia.
  - Powinni być w stanie pokazać wiarygodne (numeryczne, empiryczne, gwarantowane) powiązanie pomiędzy ich czynnościami i planami a nadzieją na pomoc tobie w osiąganiu strategicznych celów.
  - Powinni być w stanie pokazać mierzalny (numerycznie) postęp (przynajmniej używając kluczowych wskaźników), przedstawić dowód na to, że ich plany działają w praktyce.
  - Powinni oczekiwać nagród wynikających nie z tego, co zrobili w dobrej wierze, lecz z tego, jaką wartość dla ciebie ma to, co dostarczyli.
  - Kontraktorzy zewnętrzni powinni być przygotowani na wspieranie twoich celów i na to, że ich zarobki będą uzależnione od rezultatów pracy, a nie tylko od podejmowanego wysiłku.

Polityka 1.4: Polityka odpowiedniego wsparcia

Każda osoba, która wspiera twoje cele, powinna:

- umieć pokazać wprost estymowaną relację do twojego celu wyrażonego numerycznie;
- w razie potrzeby, gdy zmienią się twoje cele, potrafić dostosować się do sytuacji;
- być oceniana pod kątem efektywności kosztowej i czasowej oraz tego, czy pomaga osiągać twoje cele.

Dlaczego?

Abyśmy wiedzieli, czego się spodziewać i kto jest odpowiedzialny za poszczególne obszary.

1.5. Środki do celów: Wszystkie inne plany, niezależnie od ich nazwy, muszą wspierać osiągnięcie twoich celów – na czas

Poniżej znajduje się lista „wszystkich innych planów, niezależnie od ich nazwy”:

- wszystkie plany dla podwykonawców i konsultantów, które są finansowane z twojego budżetu;
- wszystkie kontrakty i umowy;
- wszystkie podprojekty i ich plany;
- wszystkie plany strategiczne;
- wszystkie spotkania i szkolenia;
- wszystkie rekrutacje i zwolnienia w twoim obszarze.

Musisz odpowiedzieć sobie, jaki mają potencjalny wpływ na twoje cele i budżet.

Jeśli uznasz, że ich znaczenie jest „żadne, ale i tak musimy to zrobić”, powód powinien być znany (kwestie prawne, zgodność z zasadami, sprawy wizerunku, polityka firmy) i zaakceptowany.

Jeśli ktoś chce wnieść jakiś wkład do twoich celów, wtedy może rozpocząć się proces trudnych pytań [12 Trudnych Pytań]. Dzięki nim obie strony będą wiedziały, czego oczekują i czy ich założenia są realistyczne, czy może ryzykowne.

Polityka 1.5: Polityka „Skonfrontuj Założenia”

Używaj prostych, jasnych pytań, aby określić, które czynności naprawdę silnie wspierają twoje cele.

Dlaczego?

- Aby pokazać, że poważnie podchodzisz do swoich celów.
- Aby zmotywować swój zespół wsparcia do myślenia lepiej i bardziej celowo.
- Aby stworzyć lepszy zestaw faktów i założeń, które pomogą w procesie zawierania kontraktów.

1.6. Mierz rzeczywistość: Wszystkim celom o zdefiniowanej skali mogą i muszą towarzyszyć odpowiednie metody pomiarowe, które dostarczą informacji o aktualnym poziomie realizacji założeń

W fizyce pierwszym znaczącym krokiem w kierunku poznania nowego przedmiotu jest znalezienie opisu numerycznego i praktycznych metod związanych z mierzeniem rzeczy z nim związanych.

Często mówię, że jeśli możesz zmierzyć to, o czym mówisz, i jesteś w stanie wyrazić to w liczbach, to znaczy, że coś o tym wiesz. Jeśli jednak nie potrafisz tego zmierzyć, nie możesz wyrazić tego numerycznie, twoja wiedza jest skromna i niewystarczająca. Może to być początek wiedzy, ale ty w swoich myślach ledwie wzniosłeś się do stanu Nauki, niezależnie od tego, czego dotyczy temat.

Lord Kelvin, Lecture to the Institution of Civil Engineers, 3 maja 1883, 1893

Wiele osób myli lub łączy koncepcje ujęcia ilościowego i mierzenia. Zazwyczaj używają one łatwej wymówki, że „idealne mierzenie" jest zbyt trudne, ponieważ chcą uniknąć ilościowego ujęcia tematu.

Nielogicznie!

Oczywiście rozróżnienie między budżetem a księgowością, pomiędzy woltem i woltomierzem jest jasne, jednak ludzie – na swoją niekorzyść – ciągle mieszają pojęcia. Tak jak my przez jakiś czas.

Zauważcie, że Kelvin w cytacie powyżej (który określa kierunek naszej pracy zawodowej od około 1965 roku) mierzenie i określanie ilościowe rozróżnia 2 razy w jednym zdaniu, a 3 razy w całej wypowiedzi. To nie jest przypadek.

Określanie ilościowe samo w sobie ma wielkie znaczenie, nawet jeśli nigdy nie zaczniesz niczego mierzyć! Budżetowanie sprawia, że będziesz się zastanawiać, co może się zdarzyć, choćby dane księgowe w rzeczywistości były zupełnie inne. Planowanie finansów przynosi ci ograniczenia, z którymi musisz się liczyć, podczas gdy prawdziwe pomiary grożą pojawieniem się rzeczywistych problemów. To samo rozgraniczenie dotyczy formułowania naukowej lub inżynierskiej hipotezy oraz konsekwentnego eksperymentowania w celu sprawdzenia, czy jest ona prawdziwa, czy nie. Ujęcie ilościowe to – ponad wszystko – użyteczne narzędzie w komunikacji między ludźmi. Liczby rozjaśniają to, co słowa chowają i zaciemniają. Już sama organizacja ujęcia ilościowego (w praktyce: określenie skali pomiarowej i wskazanie na niej interesujących punktów) okazuje się użyteczna. Wiemy też, że zazwyczaj, prędzej czy później, przydatne jest prawdziwe obserwowanie rzeczywistości w liczbach, czyli mierzenie w praktyce. Umożliwia ono prawdziwy kontakt z prawdziwym światem. Jeśli pomiary są częste i zaczynają się wcześnie, możemy dostosowywać nasze plany w taki sposób, żeby lepiej odpowiadały rzeczywistości, a także naszym celom i ograniczeniom.

Mierzenie nie musi być „doskonałe". W rzeczywistości nawet nie może takie być, co przyznają zarówno inżynierowie, jak i naukowcy. Kelvin nie był fanatykiem. A zatem pytanie brzmi: Jaka właściwie jest satysfakcjonująca jakość pomiarów (dokładność, precyzja, wiarygodność) i który proces mierzenia zapewniający tę jakość będzie najtańszy?

Na poszczególnych etapach tworzenia systemu z różnych powodów możemy zdecydować się na inne procesy pomiarowe. Wybór zależy od wielu wymiarów, a więc jest to decyzja związana z projektem inżynierskim.

Polityka 1.6: Planuj pomiary formalnie i integruj je z planami dotyczącymi celów

Pisemne formalne plany, jak mierzyć w praktyce, będą zawarte w specyfikacji celów.

Dlaczego?

Sprawi to, że zastanowimy się, kiedy zamierzamy wykonywać pomiary, a także rozpatrzymy różne poziomy mierzenia wraz z ich kosztami.

Pomoże to uniknąć zbyt wielu pomiarów.

W jaki sposób?

Parametr „Pomiar” będzie mógł być używany w specyfikacji różnych rodzajów procesów pomiarowych.

1.7. Złożoność koncepcyjna: Niektóre cele są złożone, mają wiele wymiarów i w związku z tym wiele skal, które je opisują

Miłość jest wspaniała – mówi stara piosenka. Jednak wysokość i waga mają tylko i wyłącznie jeden wymiar.

Próbując wyjaśnić lub określić ilościowo cele, możesz napotkać pewien problem – brak satysfakcjonującego wymiaru do mierzenia. Tymczasem istnieje ich kilka. „Który jest właściwy?” – aż kusi, żeby zapytać. Prawidłową odpowiedzią może być „Wszystkie z nich, a nawet więcej”. Stary elektroniczny podręcznik rekomenduje dzielenie koncepcji tak długo, aż określenie ilościowe stanie się oczywiste.

René Descartes (1596-1650) rekomendował to samo podejście (*Dyskurs o Metodach*).

Nie uważać za prawdziwe niczego, co nie może być jasno rozpoznane jako takie, czyli ostrożnie unikać pochopności i przesądów w osądach, i nie akceptować niczego ponad to, co zostało zaprezentowane mojemu umysłowi w sposób tak jasny i wyróżniający, że nie będę miał okazji w to zwątpić.

Podzielić wszystkie problematyczne elementy, które badam, na tak wiele części, jak tylko się da i jak to wydaje się właściwe, aby mogły zostać rozwiązane w najlepszy możliwy sposób.

Aby ułożyć moje myśli w należytym porządku: należy radzić sobie z obiektami, które są najprostsze i najłatwiejsze do zrozumienia, tak aby krok po kroku lub stopień po stopniu zdobyć wiedzę o tych najbardziej skomplikowanych, zakładając porządek, nawet fikcyjny, między tymi, które nie podążają za naturalną sekwencją między sobą. We wszystkich wypadkach należy poczynić wyliczenie tak kompletnie, a opinie sformułować na tyle ogólne, żeby mieć pewność, że nie pominęło się nic.

Pewnego razu CEO IBM zdecydował, że *użyteczność* jest falą przyszłości, jeśli chodzi o nowe komputery osobiste. Tom został poproszony o pomoc firmie IBM.

Zasugerował określenie ilościowe *użyteczność*, ale minęły miesiące, nim zrozumieliśmy, że jest tutaj wiele wymiarów, a nie jeden. Ten wielowymiarowy model został następnie zaadaptowany przez IBM.

Jeden z klientów poprosił nas o przeanalizowanie dużego projektu, który się nie powiódł (trwał 8 lat, pochłonął 160 mln dolarów, uczestniczyło w nim 90 osób). CEO w pierwszej kolejności położył nacisk na radykalną poprawę „solidności” głównego produktu. Zawodził on najważniejszych klientów zbyt często, i to od dawna.

Oryginalne początkowe wymaganie, którego nikt nie traktował poważnie przez 8 lat, brzmiało: **Solidny jak skała** (oficjalny nagłówek specyfikacji).

W dalszej części dokumentu wymieniono takie wymagania jak: niepsucie się zbyt często (2 tygodnie) i szybka naprawa (10 minut). To wszytko połączono z długą listą strategii, w jaki sposób można osiągnąć takie efekty.

Sugestia Toma przypominała raczej coś takiego:

Solidny jak skała:

Typ: Złożone wymaganie dotyczące jakości produktu
Zawiera:
{Czas niedostępności oprogramowania,
Szybkość przywrócenia,
Testowalność,
Zdolność zapobiegania błędom,
Zdolność izolowania błędów,
Zdolność do analizowania błędów,
Zdolność do debugowania sprzętu}

Pierwsze trzy wymagania określono ilościowo w ciągu godziny. Wszystkie z nich powinny zostać określone ilościowo i być używane do sterowania projektem 8 lat wcześniej. Stopniowe udoskonalenia celów jakościowych należało wprowadzać przez pierwsze kilka miesięcy trwania projektu.

Polityka 1.7: Dziel złożone cele na mniejsze części

Krytyczne cele najwyższego poziomu powinny zostać rozłożone na elementarne składniki, które można określić ilościowo, jeśli pozwoli to na lepsze zarządzanie celami najwyższego poziomu.

Dlaczego?

- Ponieważ zapewnia to bardziej realistyczne spojrzenie i konsekwentne podejście do podstawowych aspektów problemu.
- Ponieważ takie podejście zmusza do bardziej wnikliwego myślenia i ułatwia realizację celów mierzalnych ilościowo.

1.8. Szybko odzwierciedlaj rzeczywistość. Zmiana specyfikacji celów jest naturalną konieczną odpowiedzią na spostrzeżenia, informacje zwrotne, konkurencje i politykę

To, że cel jest spisany lub określony ilościowo, nie znaczy od razu, iż został „wykuty w kamieniu". W zasadzie jeden z powodów zapisywania założeń to fakt, że później wyraźnie widać jakiekolwiek postępy. Powodem określania ich ilościowo jest chęć lepszego uzmysłowienia sobie, że zmieniła się wartość numeryczna – nieważne w jakim stopniu.

Polityka firmy musi wymagać, aby zmiany były zrozumiałe. Nawet najmniejsza różnica może mieć poważne konsekwencje. Bardzo istotna jest zatem umiejętność wyczuwania zmian i podejmowania szybko odpowiednich czynności.

W jedynym opublikowanym studium przypadku (AT&T, 5ESS system, Communications of ACM) głównym czynnikiem była zmiana dostępności systemu przełączania z 99,90% na 99,98%. Mówimy zatem o różnicy rzędu 00,08% dotyczącej zaledwie jednego czynnika. Koszt zmiany: 8 lat i zaangażowanie 2000-3000 ludzi. Wyobraźcie sobie konsekwencje, gdyby czynnik („najwyższa dostępność") nie był mierzony numerycznie lub nie składał się z 4 cyfr. Stosujemy wiele metod podczas planowania, tak aby wprowadzać dyskretne modyfikacje jaśniej. Wierzymy, że powinno to być określone na poziomie detali (celów), a nie „wielkiego planu", w którym fakt zmian łatwo może zostać przeoczony.

Na liście interesariuszy znajdują się główni gracze zainteresowani jakimikolwiek modyfikacjami. Mogą oni być informowani o zmianach, mają również prawo opiniować

je i zatwierdzać. Właściciel to w tym przypadku właściciel specyfikacji. Nikt poza nim nie może dokonać oficjalnej zmiany w dokumencie. Właściciel jest zobowiązany modyfikować specyfikację odpowiedzialnie, na przykład powinien informować o tym interesariuszy. Może także skonsultować się z ekspertami domenowymi przed opublikowaniem zmiany.

Kontrola wersji, na przykład za pomocą daty modyfikacji lub numeru wersji, pomaga uczulić czytających na różnice między poszczególnymi wydaniami specyfikacji. Data kontroli jakości i status przypominają, że tego typu czynności jeszcze nie przeprowadzono.

Nie wszystkie te szczegóły musisz uwzględniać za każdym razem, kiedy będziesz mówić o danym celu albo go wykorzystywać. Powinny jednak znajdować się w bazie danych z planami, jeśli mają być traktowane poważnie i jeżeli twoja potrzeba szybkiego wprowadzania zmian w innym wypadku mogłaby wprowadzać zamieszanie.

Za tym wszystkim kryje się jedna zasada, którą gorąco polecamy. Brzmi ona: powinna być tylko i wyłącznie jedna prawidłowa wersja – nazwijmy ją Główną Specyfikacją – dla każdej strategii czy celu. To do niej musi się odnosić całe planowanie. Każde inne podejście spowoduje, że w projekcie szybko zapanują chaos i anarchia.

Polityka 1.8: Zmieniaj plany szybko, ale odpowiedzialnie

Odpowiednio zaplanowane cele będą zawierały informacje pozwalające na szybkie modyfikowanie szczegółów w bezpieczny sposób – wszystkie osoby, których dotyczą zmiany, zostaną o tym powiadomione.

Dlaczego?

Jeśli o to nie zadbasz, będziesz używać przestarzałych planów i podejmować decyzje na podstawie nieaktualnych informacji. To niezbyt konkurencyjne podejście.

1.9. Bogata rzeczywistość: Pojedynczy cel może być wyspecyfikowany w każdej użytecznej ilości wymiarów czasu, przestrzeni i zdarzeń

Posiadanie tylko jednej liczby reprezentującej cel może być niebezpieczne.

Jeśli tak jest w twoim przypadku, niestety będzie konieczne logicznie, aby była to największa liczba pokrywająca wszystkie twoje potrzeby – zawsze i w każdych warunkach.

To zły pomysł!

My stosujemy rozróżnienie (coś w rodzaju „segmentacji rynku") celów (które planujemy osiągnąć) i ograniczeń (najgorszych akceptowalnych poziomów). Powód takiego podejścia jest prosty: możemy planować bardziej konkurencyjnie dzięki jasnemu rozdzieleniu sytuacji, które dają dużą wartość w krótkim czasie, od „pozostałych 95%". Pozwala to szybko dostarczać wartość. Na przykład zamiast po prostu określić:

Cel: 20%

wolelibyśmy ustalić różne cele dla konkretnych okresów czasu, środowisk (kto, gdzie) i zdarzeń. Technicznie nazywamy je kwalifikatorami poziomu celu.

To coś w rodzaju:

Cel [Termin = pierwsze wydanie, Rynek = Chiny, Klient = golfista, Założenie = import bez podatku] 20%

Co może również zostać zapisane następująco:

Cel: 20%
Termin = pierwsze wydanie,
Rynek = Chiny,
Klient = golfista,
Założenie = import bez podatku

Wyrażenie:

[Termin = pierwsze wydanie, Rynek = Chiny, Klient = golfista, Założenie = import bez podatku]

jest „kwalifikatorem".

Zawiera ono trzy typy kwalifikatora: *Kiedy, Kto* i *Jeżeli.*

Takich kwalifikatorów może być dowolna potrzebna liczba. Zwłaszcza *Co* ma często od 3 do 6 wymiarów o większej szczegółowości. Wymiaru *Jeżeli* nie zawsze się używa. Natomiast niestosowanie *Kiedy* jest prawie nielogiczne, bo dopuszczałoby osiąganie celu w nieskończoność. Ograniczenia czasowe mają wielki wpływ na wybór środków przybliżających nas do realizowania ustalonych założeń.

Oczywiście w dowolnym momencie procesu planowania możesz mieć tak dużo różnych wyrażeń celu z tak wieloma różnymi kwalifikatorami, jak tylko zechcesz. Ustal dowolny zbiór długich, średnich i krótkich poziomów celu, kiedy tylko odczujesz taką potrzebę.

Planowanie detali może się okazać czynnością równoległą do dostarczania wartości w twoim projekcie. Uszczegółowione wyrażenia czasami są rezultatem informacji zwrotnej z dostarczenia produktu. Masz szansę dowiedzieć się o nowych rynkach i

interesariuszach. Warto się tym zainteresować.

Zazwyczaj predefiniujemy część parametrów, ale niekoniecznie wszystkie, w definicji „Skali". W ten sposób moglibyśmy użyć specyfikacji Skali z „Parametrem Skali", na przykład:

Skala – średni roczny procent sprzedaży w podziale na Rynek i Odbiorcę

To sugeruje, że możemy zdefiniować każdą użyteczną kombinację Rynku i Odbiorcy w funkcji celu (Cel) przy ograniczeniach (Tolerowalnych), a także odnieść to do przeszłości.

Polityka 1.9: Polityka mądrego różnicowania

Określaj cele poprzez uszczegółowianie w jasny sposób, „kiedy", „kto", „co" i „jeżeli" w taki sposób, aby zmaksymalizować krótko- i długoterminową konkurencyjność firmy.

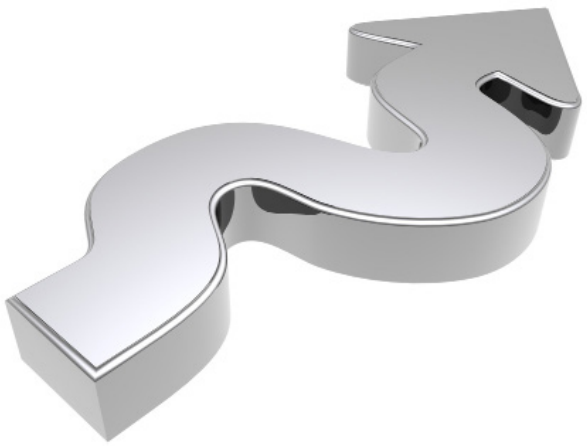
Dlaczego?

- Aby uniknąć opóźnień w przypadku wybranych pilnych interesariuszy.
- Aby wyodrębnić wykonalne akcje krótkoterminowe.
- Aby zmusić się do myślenia w sposób bardziej jasny i szczegółowy o ważnych interesariuszach, jak również o ich prawdziwych potrzebach i o tym, jak szybko powinniśmy dostarczać im gotowe wartości.

1.10 Motyl: Najmniejsza numeryczna lub inna zmiana celu może niespodziewanie wywołać poważne konsekwencje dla koniecznych strategii i ich kosztów

A zatem bądź ostrożny, prosząc o coś. Możesz tego nie potrzebować, niewykluczone też, że zwyczajnie cię na to nie stać.

Jeśli za idealną wartość dla czasu działania systemu bez wyłączenia uważa się 99,998%, ile będzie cię kosztować wymaganie najlepszego, konkurencyjnego 99,999%? Twoja firma okaże się lepsza tylko o 00,001%?



AUTOR

Tom Gilb

Tom jest autorem dziewięciu książek, setek publikacji na tematy związane z jakością w IT i podobnymi.

Tom wykładał gościnnie na uniwersytetach w Wielkiej Brytanii, Chinach, USA i Korei – dodatkowo był głównym prelegentem dziesiątek technicznych konferencji na całym świecie.

www.gilb.com, twitter: @imTomGilb

Po pierwsze, nikt tego nie wie! Jest tylko jeden sposób, żeby się dowiedzieć. Zrobić to.

Jak wiedzą wszyscy inżynierowie, osiągnięcie 100% w skończonym czasie przy znanym koszcie jest niemożliwe. Żaden z nich nie traktuje dążenia do perfekcji na poważnie. A dokładniej: w bardzo konkurencyjnych sytuacjach rozszerzasz granicę, przesuwasz rekord. I nikt nie wie, ile to będzie kosztować, dopóki założenie nie zostanie zrealizowane.

Jak pisaliśmy wcześniej (przypadek AT&T), odpowiedź wydaje się szokująca (24 000 osobolat wysiłku, by zmienić 99,90 na 99,98% dostępności. To może być satysfakcjonujące dla firmy o ogromnym budżecie). Zarząd nie powinien po prostu na poważnie domagać się bezawaryjności „24/7”.

Tak więc w naszym planistycznym repertuarze mamy sposoby na ostrzeganie siebie i na zrozumienie, dlaczego wybraliśmy konkretne poziomy dla celów, jak również nasze Cele i poziomy Tolerancji. Nazywamy to Benchmarkami.

Polityka 1.10: Bądź realistyczny

Opieraj swoje plany na realistycznych informacjach o tym, co jest przyjmowane jako najlepsze rozwiązanie w danej dziedzinie, a także na tym, czym dysponuje konkurencja. Spisz wszystkie te dane w taki sposób, by były połączone z twoimi celami i jak najlepiej aktualizowane.

Dlaczego?

Abyś mógł tworzyć realistyczne i konkurencyjne plany.

- Abyś mógł wyjaśnić i usprawiedliwić swoje cele przed kupującymi i aprobującymi.
- Abyś mógł zapobiec sytuacji, w której oderwani (od rzeczywistości) managerowie będą wymagać od ciebie więcej, niż tak naprawdę potrzebujesz, niż cię na to stać czy więcej, niż jest możliwe.

Quality for IT 2015
Przeżyjmy to jeszcze raz!

QUALITY FOR IT

„Bardzo dobra jakość prezentacji ekspertów /naukowców. Prezentacje wniosły bardzo dużo konkretnych rozwiązań/danych. Warto zapraszać takich ludzi”.

Piotr Bogorodź, Nordea IT Polska

„Konferencja bardzo ciekawa i interesująca, duża różnorodność tematyczna dotycząca jakości w IT. Jedyny minus to długość sesji – są za krótkie”.

Paweł Bielecki, TFS

„Dzięki konferencji zmieniło się moje podejście do jakości. Wcześniej widziałem je jako proces testów. Teraz dostrzegam większość aspektów składających się na jakość końcowego produktu. Wszystkie wykłady były bardzo ciekawe i interesujące.”

Albert Ścibor, CONNECTMEDICA

Michał Węgrzyn

Testowanie w modelu service delivery

Co musisz wiedzieć:

- Obszar, w którym ISTQB spotyka się z ITIL-em, jest dość specyficzny.
- Testowanie aplikacji, która ma już bazę użytkowników posiadających często długoletnie doświadczenie, oznacza nowe wyzwania.
- Wdrożenie testera w środowisku service delivery, kiedy aplikacja „żyje" już od dłuższego czasu, przypomina trochę wsiadanie do jadącego pociągu. Jest ekscytujące, ale i niełatwe.

Software development vs service delivery

Dla mnie idealnym, wymarzonym scenariuszem jest tworzenie oprogramowania od podstaw. Mówię o sytuacji, gdy projekt dopiero się rozpoczyna, zespół zbiera się po raz pierwszy, mamy konkretny zestaw wymagań zebranych w backlogu lub dokumencie Scope of Work. Nie ma długu technologicznego, nie ma elementów systemu, których autorzy dawno już nie pracują w firmie, nie ma starych known issues. Piękna sprawa.

Rzeczywistość jednak jest inna. Ostatnio w czasie rozmowy z szefem dużej firmy outsourcingowej dowiedziałem się, że tylko 20% projektów w jego firmie stanowią nowe przedsięwzięcia. Reszta to utrzymanie i ewentualnie change requesty. Jeśli tak, to znaczy, że w 4 na 5 przypadków testujemy coś, co ma już wyżej wymienione słabości.

Dodajmy do tego fakt, iż ani szkolenia i certyfikaty ISTQB, ani ITIL na poziomie podstawowym nie przygotują nas na to, czego należy się spodziewać w takim otoczeniu.

Specyfika testowania aplikacji będącej serwisem

Zacznę od wypunktowania najważniejszych moim zdaniem charakterystyk testów aplikacji będącej serwisem:

- **Rozmiar** – nasza aplikacja nie jest nowa. Może mieć na karku kilka albo kilkanaście lat. Przez ten czas wprowadzano zmiany, łatano system, dodawano nowe funkcjonalności. Teraz mamy wersję 6.12. Kto z zespołu pamięta zmiany wprowadzone w edycji 2.7? Takich osób prawdopodobnie jest niewiele, zresztą czas robi swoje i nawet one zapewne zapomniały już o szczegółach implementacji.
- **Integracje** – w środowisku korporacyjnym bardzo niewiele aplikacji działa samodzielnie. Od najprostszych integracji z Active Directory, poprzez sieć powiązań, wszystko łączy się ze wszystkim. W ciągu lat działania naszej aplikacji dodano do niej wiele integracji z innymi systemami. One też żyją, zmieniają się, powstają kolejne ich wersje. To wszystko wymusza działania po naszej stronie. Nowe interfejsy, kolejne reguły walidacyjne, inne technologie. Każdą ze zmian trzeba będzie przetestować.
- **Serwis to niekoniecznie jedna aplikacja** – w środowisku korporacyjnym zespoły często utrzymują więcej niż jedną aplikację. Czasami ich programy są połączone, czasami nie. Im więcej aplikacji, tym więcej release'ów i nakładających się terminów. Więcej wszystkiego.
- **Good old „legacy"** – bardzo prawdopodobne, że nasza aplikacja jest w użyciu tak długo, że pojawiają się już plany jej wycofania. Z użycia może również wychodzić część elementów, z którymi się integrujemy. Na ich miejsce powstają nowe, a zatem trzeba będzie się z nimi zintegrować. Testowanie wygaszanej aplikacji jest taką samą procedurą jak przypadku budowanego programu.
- **Historyczne, zapomniane wymagania** – podczas testów obserwujemy nieoczekiwane działanie systemu. Tworzymy raport defektu tylko po to, aby otrzymać rezultat: working as expected. Czemu? Bo kilka lat temu w ramach obejścia problemu X wprowadzono takie właśnie działanie. Oczywiście osoby, które to rozwiązanie implementowały, nie należą już do zespołu.
- **Baza użytkowników** – testerzy są jedną z grup, które dysponują najszerszą wiedzą w zakresie specyfiki aplikacji. W przypadku systemu działającego od lat zajmują jednak drugie miejsce w tej klasyfikacji. W organizacji są bowiem ludzie, którzy pracują z aplikacją od lat, może nawet od początku jej istnienia. Takie osoby nie potrzebują skryptów testowych ani wskazówek. Znają działanie aplikacji i bardzo zależy im na jej wysokiej jakości. Ostatecznie przecież to one będą pracować z systemem.
- **Środowiska testowe, stage'owe, produkcyjne** – jak powszechnie wiadomo, najlepsze testy to te na produkcji :). W przypadku dużych aplikacji, działających od lat, ilość danych produkcyjnych może być duża. Powoduje to konieczność przeprowadzania skomplikowanych i czasochłonnych operacji

tworzenia kopii oraz importowania ich na środowiska testowe i stage'owe. Do tego dochodzi możliwość migracji oraz testowania jakości migracji danych.

- **Model service desk – operations – development** – kolejnym elementem jest udział innych stakeholder'ów w testach. W dobrze zorganizowanym zespole każdy może zgłaszać błędy, pracownik service desku, administator bazy danych z operations czy też architekt. Wyzwaniem jest stworzenie kultury jakości.
- **Globalne otoczenie korporacyjne** – świat IT jest mały. Fakt, że programista pracuje na jednym końcu świata, a użytkownik jego aplikacji na drugim, już nikogo nie powinno dziwić. Zespoły utrzymujące aplikacje również są rozproszone i często funkcjonują w innych strefach czasowych.
- **Wdrożenie nowej osoby** – biorąc pod uwagę wszystkie wyżej wymienione cechy specyficzne, dużym wyzwaniem staje się wdrożenie nowego testera do projektu. Od czego zacząć? Jak ułatwić testerowi przyswojenie takiego ogromu wiedzy?

Przykładowy model

Rysunek 1 przedstawia geograficzny i zadaniowy podział mojego obecnego zespołu, którego zadaniem jest utrzymanie 4 aplikacji.

Pracujemy w różnych strefach czasowych, reprezentujemy inne kultury, pełnimy rozmaite funkcje.

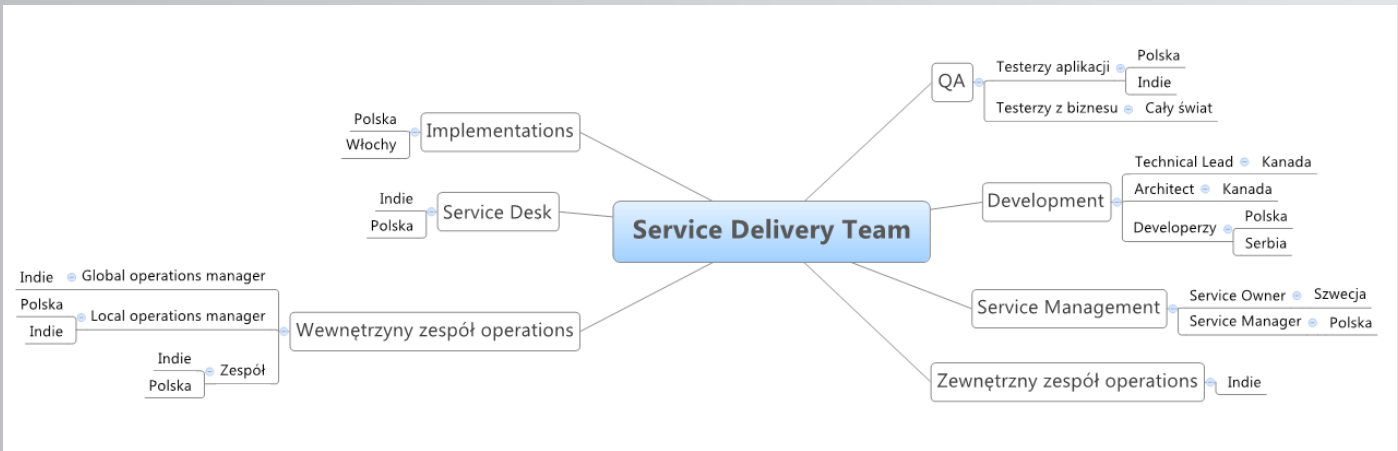
Tester w oku cyklonu

Spójrz jeszcze raz na rysunek 1. Dodajmy teraz kilka integracji i uwzględnijmy fakt, że nasz zespół zajmuje się 4 aplikacjami. W tym wszystkim musi odnaleźć się tester.

Rysunek 2 przedstawia najważniejsze wyzwania stawiane przed testerami w zespołach zajmujących się utrzymaniem i rozwojem aplikacji będącej serwisem.

Wiele z powyższych występuje również w innych kontekstach testowania. Dlatego skupię się na moim zdaniem najważniej-

Rysunek 1. Geograficzny i zadaniowy podział zespołu



Rysunek 2. Najważniejsze wyzwania w zespołach zajmujących się utrzymaniem i rozwojem aplikacji będącej serwisem

szych wyzwaniach:

- **Rozbudowany system** – jak już wspomniałem, bardzo prawdopodobne jest, że od Go-Live naszej aplikacji minęło już wiele lat. Od tego czasu nastąpiło w nich wiele zmian, pojawiło się mnóstwo nowych funkcjonalności oraz poprawiono liczne defekty. Nawet zakładając, że dokumentacja projektowa była prowadzona idealnie (czy ktoś słyszał o takim przypadku?), mamy bardzo, ale to bardzo dużo informacji do przyswojenia. Moim zdaniem w przypadku takiej aplikacji nie da się zrozumieć i nauczyć wszystkiego. Próg naszych zdolności poznawczych, jako gatunku, jest ograniczony. Z tego też powodu uważam, że specjalizacja oraz selektywna nauka są kluczem do efektywnego wykonywania zadań testerskich.
- **Niejasne, często archaiczne wymagania** – to wyzwanie po części wynika z poprzedniego. Pierwsze jego źródło to czas i migracje pracowników. Wiedza, często przenoszona przez żartobliwie zwany „interface białkowy", tzn. pozostająca w głowach analityków i programistów, ginie wraz z ich odejściem. Generuje to luki w rozumieniu, jak aplikacja działa oraz jak musi funkcjonować i dlaczego. Kto, jeśli nie członkowie zespołu, ma wiedzieć, co powinien robić system? Polecam zapytać biznes. W organizacji są ludzie, którzy mogli brać udział w testach danej funkcjonalności. Niewykluczone, że znajdą się nawet osoby, które złożyły na nią zapotrzebowanie. Nie bójmy się pytać.
- **Liczne integracje** – powiedzmy sobie szczerze: jeśli sami niekiedy tracimy kontrolę nad swoją aplikacją, czego możemy oczekiwać od innych zespołów w ramach tej samej organizacji – ludzi mających utrzymywać system, z którym nasz program się integruje? I tu po raz kolejny jasna i przejrzysta komunikacja między teamami pozwoli nam pokonać tę przeszkodę. Klasyczne korporacyjne „odbijanie piłeczki" do niczego nie prowadzi. Spotkajcie się i omówcie to, co wiecie, ustalcie, czego nie wiecie, a przede wszystkim przedstawcie sobie nawzajem swoje oczekiwania. Pozwoli to, po pierwsze, zaoszczędzić mnóstwo czasu i bezproduktywnej wymiany e-maili; po drugie, przy okazji dowiecie się czegoś ciekawego o własnym produkcie. Czegoś, o czym nie wiedzieliście do tej pory. Zrozumienie tematu być może odkryje kilka potencjalnych błędów.

- **Długie przestoje** – cykl życia aplikacji obejmuje wiele faz. Jedną z nich, najmniej angażującą testerów, jest etap „działa". Często bywa tak, iż po ostatnim releasie wiele defektów zostało usuniętych lub też decyzją biznesu aplikacja nie będzie już rozwijana, ponieważ nie jest to uwarunkowane potrzebami użytkowników. Takie momenty trzeba wykorzystywać. Pierwsze miejsce na mojej liście spraw zajmuje uzupełnienie wiedzy, dokumentacji oraz lepsze zrozumienie, jak działa nasz program. Można to osiągnąć na kilka różnych sposobów. Moim zdaniem najlepsza metoda, bo zapewniająca wiele korzyści jednocześnie, to aktualizowanie bazy test case'ów. Nie jestem fanem podejścia polegającego na testowaniu wszystkiego zgodnie ze skryptem. W takiej sytuacji eliminujemy dużą cześć potencjału ludzkiego mózgu i zmieniamy testera w maszynę wykonującą skrypty. Jednak w przypadku dużych systemów sama obecność dobrze napisanych przypadków testowych służy nam dwojako. Po pierwsze, jako checklista tego, co musimy przetestować. Po drugie, znacznie ułatwi wdrożenie nowej osoby, jeśli zajdzie taka potrzeba.

Gdzie szukać usprawnień

- **User Acceptance Testing, czyli biznes prawdę Ci powie** – najważniejszą, absolutnie kluczową sprawą w tworzeniu każdego rodzaju programowania jest feedback od użytkowników końcowych. W naszym przypadku okazuje się to jeszcze bardziej istotne, biorąc pod uwagę fakt, że część z nich to eksperci w swojej dziedzinie, wieloletni interesariusze aplikacji. Im najbardziej zależy na tym, by aplikacja działała poprawnie. Dodajmy do tego fakt, iż takie osoby dysponują wiedzą, dzięki której można dostrzec zagadnienia przeoczone przez zespół projektowy. Nie bójmy się więc poprosić ich o feedback. Dobrą praktyką jest przygotowanie demo na żywo nowych opcji. Połączenie konferencyjne oraz screen sharing to doskonałe narzędzia dla tego typu zastosowań. Ponadto musimy wziąć pod uwagę fakt, iż nasi użytkownicy pracują w różnych strefach czasowych. Spotkanie powinno zatem odbyć się o takiej porze, która gwarantuje udział jak największej liczby interesariuszy. Pozostali będą mogli zapoznać się z nagraniem spotkania (audio i video). W tym przypadku sprawdzi się narzędzie o nazwie Camtasia.
- **Pozwólmy sobie pomóc.** Aby jak najbardziej zmobilizować biznes, musimy zapewnić mu dwie kluczowe rzeczy. Pierwsza to informacje o tym, co należy testować, czyli wcześniej wspomniane demo. Ponadto musimy dać użytkownikom wystarczającą ilość czasu czas na testy. Dlaczego? Ponieważ testowanie naszej aplikacji jest dla nich drugorzędnym zadaniem. Ich priorytetem są codzienne obowiązki. Dlatego też zaproszenie na UAT wysyłajmy użytkownikom z kilkutygodniowym wyprzedzeniem oraz dajmy im przynajmniej kilka dni na testy.
- **Stakeholderzy w kategorii „ulubiony"** – feedback z biznesu będzie różny. Podczas analizowania informacji zwrotnej warto pamiętać o jednej sprawie. Użytkownicy zwykle nie posiadają umiejętności technicznych, często wiedzą niewiele lub nic o procesir wytwarzania oprogramowania. Priorytetem w komunikacji jest nietraktowanie ta-

kich osób z góry. Tłumaczmy wszystko w miarę potrzeb, delikatnie jednak dzieląc testerów UAT na tych bardziej i mniej „ważnych". Niestety, jak zawsze, są plusy i minusy dopuszczenia do testów osób niemających pojęcia o tym procesie. Do wad można zaliczyć: szum komunikacyjny, zgłaszanie usterek, które w rzeczywistości nie są błędami, wynikające z braku zrozumienia aplikacji. Jednak z mojego doświadczenia wynika, że na 10 osób znajdzie się 1-2, których opinię powinniśmy traktować jak świętość. Skupmy się na nich. Jeśli takie osoby uznają, że nasz release jest dobry, możemy uznać ich opinię za rzeczywisty wyznacznik jakości.

- **Macierz specjalizacji** – technika, którą posługuję się przy decydowaniu, co kto powinien umieć, abyśmy jako zespół mogli testować wszystko oraz aby osiągnąć pewny poziom redundancji. Przykład na rysunku 3.

  Do podziału można też dodać testy automatyczne, manualne, performance'owe.

- **Otwarta komunikacja z innymi zespołami odpowiedzialnymi na systemy, z którymi się integrujemy** – wspominałem już o tym w części dotyczącej integracji. Zrozumienie, jak działa nasza aplikacja podczas łączenia się z inną, pozwala głębiej i lepiej zrozumieć mechanizmy jej funkcjonowania, a co za tym, umożliwia lepsze testowanie jej. Dlatego też zamiast nieustannie przerzucać temat lub tygodniami wymieniać się mailami, warto zorganizować spotkanie czy konferencję telefoniczną – cokolwiek, co popchnie sprawę naprzód w stronę wspólnego celu.
- **Utrzymanie dokumentacji testowej w jak najlepszym stanie** – chyba niewielu testerów lubi pisać test case'y. Za to większość powie, że ucząc się nowej aplikacji, dobrze mieć takie dokumenty. Dlatego zachęcam do stworzenia odpowiedniej bazy. Przyda się ona nie tylko nam, ale też innym członkom zespołu. Na przygotowanie dokumentacji wykorzystajmy czas między releasami, kiedy nie ma konkretnych zadań do wykonania.
- **Tester jako wsparcie użytkownika** – innym ciekawym pomysłem na konstruktywne wypełnienie czasu jest zaangażowanie testera w support aplikacji. Z własnego doświadczenia wiem, jak bardzo poszerza to horyzont testowania. Sam zyskałem w ten sposób kilka bardzo ciekawych przypadków testowych i na stałe dołączyłem je do swojego zestawu testów. Wywodzą się one bezpośrednio ze zgłoszeń końcowych użytkowników z pytaniami do suportu. Dodatkowo, jeśli tester jest adwokatem userów, to jak lepiej mógłby wczuć się w rolę takiej osoby niż

Rysunek 3. Macierz specjalizacji

Osoba/Aplikacja	Aplikacja A	Aplikacja B	Aplikacja C
Katarzyna	główna	druga	
Adam		główna	druga
Piotr	druga		główna

poprzez bezpośrednią pomoc w rozwiązywaniu jej codziennych problemów z działaniem aplikacji?

Podsumowanie

Praca testera będącego częścią większego zespołu, którego zadanie polega na utrzymaniu, rozwijaniu i wsparciu aplikacji działającej i mającej swoją bazę użytkowników, jest dość specyficzna.

Otoczenie organizacji oraz fakt, iż program pozostaje w użyciu, stawiają przed takimi osobami kolejne wyzwania.

Z mojego doświadczenia wynika, że warto przeanalizować, kto posiada największą wiedzę o aplikacji oraz kto jest najbardziej zainteresowany zadaniami, jakie wykonujemy jako zespół testerski. Z takimi osobami należy szczególnie blisko współpracować. Ich opinie powinny być dla nas priorytetowe.

Każdemu testerowi na co dzień pracującemu w takim zespole radzę również zapoznać się z metodyką, według której zarządza się service'ami. Warto zdecydować się chociażby na szkolenie z ITIL-a.

AUTOR

Michał Węgrzyn

Doświadczony test engineer, quality assurance specialist, technical writer, trener i team leader. Pełni wiele funkcji w projektach informatycznych, przy czym zawsze kieruje się jednym celem: zadowoleniem użytkownika końcowego.

Pracuje w ABB Polska, gdzie na co dzień zajmuje się testami aplikacji zarówno webowych, jak i desktopowych oraz sprawowaniem pieczy nad procesem kontroli jakości wytwarzanego oprogramowania.

Cele zawodowe autora to ciągłe dążenie do zdobywania nowych umiejętności: technicznych i miękkich oraz coaching młodszych współpracowników.

Michał Węgrzyn jest zapalonym miłośnikiem kulturystyki, literatury fantasy oraz swojego buldoga angielskiego Lebrona.

QUATES

TESTERÓW I PROJEKTY ŁĄCZYMY W UDANE ZWIĄZKI

Akredytowane szkolenia ISTQB
Testowanie oprogramowania
Eksperci na projekty
Oferty pracy dla testerów

www.quates.pl

Grzegorz Dawid

Wycieczki w testach eksploracyjnych

Wprowadzenie

O testach eksploracyjnych do tej pory sformułowano dużo różnych, często sprzecznych wobec siebie opinii.

Twierdzi się np., że takie testy są dodatkową techniką, która „(…) okazuje się najpożyteczniejsza, gdy nie ma specyfikacji, albo jest ona niekompletna czy przestarzała, również w sytuacjach bardzo krótkiego czasu na testy. Przydaje się ona również do wzbogacenia i uzupełnienia bardziej formalnych testów" [1].

Nie brakuje osób, które testom manualnym wieszczą rychły koniec i zastąpienie ich przez zautomatyzowane testy developerskie oraz testy alfa i beta realizowane przez finalnych użytkowników [2, 3].

Jednocześnie można przeczytać, że „testowanie jest równoznaczne z testowaniem eksploracyjnym", a wszystkie inne techniki testowaniem nie są [4,5].

Mnie osobiście najbliżej do ostatniej z przytoczonych opinii, choć nie jestem aż tak radykalny w poglądach. Może wynika to z faktu, że obecnie nie mam możliwości automatyzacji wykonywanych testów – jestem więc skazany na testy manualne. Dlatego też w tym numerze „Quale" podzielę się z Czytelnikami garścią statystyk na temat testów eksploracyjnych, które udało mi się zebrać podczas ostatniego realizowanego projektu. Przedstawię również wnioski, które moim zdaniem wypływają z tych danych.

Projekt

Projekt wdrożeniowy realizowany był w banku. Trwał ok. półtora roku, z czego same testy po stronie banku zajęły mniej więcej 10 miesięcy. Development oraz analizę systemową realizowali zewnętrzni dostawcy (którzy formalnie odpowiadali też za realizację testów developerskich). Przeprowadzenie testów wydajnościowych i integracyjnych było obowiązkiem działu IT banku. Moje zadanie polegało na kierowaniu zespołem testerów w ramach UAT.

Zespół tworzyło ok. 40 członków, których można było podzielić na 3 kategorie.

- 2 osoby zapytane o to, czym się zajmują, bez wahania odpowiedziałyby, że są testerami. To pracownicy z dużym doświadczeniem. Jako jedyni w zespole realizowali głównie testy eksploracyjne.
- 7 osób to pracownicy różnych jednostek banku, oddelegowani na 100% czasu do zespołu testowego. Do tego grona należały osoby posiadające małe lub średnie doświadczenie w testowaniu. Po zakończeniu projektu powróciły do swoich normalnych zajęć w organizacji.
- 31 osób to pracownicy różnych jednostek banku, oddelegowani do zespołu testowego na część etatu. Najczęściej nie posiadali oni doświadczenia w testowaniu.

Testy były realizowane na podstawie przypadków testowych przygotowanych na bazie dokumentów analizy systemowej i weryfikowanych przez dwójkę doświadczonych testerów.

Niestety, nie udało się zamrozić ani wymagań, ani dokumentów analizy systemowej. W efekcie, gdy przystąpiono do testów, scenariusze testowe były już nieaktualne i należało je traktować jedynie jako ogólną sugestię, co trzeba przetestować, a nie jako wyrocznię w kontekście oczekiwanych rezultatów testów.

Wszystkie zgłoszone incydenty przed wysłaniem do IT/zewnętrznych dostawców weryfikowała dwójka doświadczonych testerów. Koncentrowali się oni na eliminacji duplikatów oraz zasadności zgłoszenia (zgodność kwestionowanej funkcjonalności z analizą systemową).

Na koniec projektu statystyka zgłoszonych incydentów prezentowała się zgodnie z tabelą 1 i rysunkiem 1 (szczegółowe dane zaprezentowane tylko dla 15 testerów z największa liczbą zgłoszeń). Dwie osoby na szczycie statystyk to oczywiście doświadczeni testerzy realizujący testy eksploracyjne.

Automatyczny wniosek, który można wysnuć z takich danych, brzmi: zamiast angażować 7 osób na pełen etat i 31 osób na ułamek etatu w celu realizacji testów scenariuszowych, rozsądniej byłoby zatrudnić dodatkowych 3 doświadczonych testerów i realizować wyłącznie testy eksploracyjne. Pozwoliłoby to też uniknąć żmudnego i czasochłonnego przygotowywania i weryfikowania scenariuszy testowych.

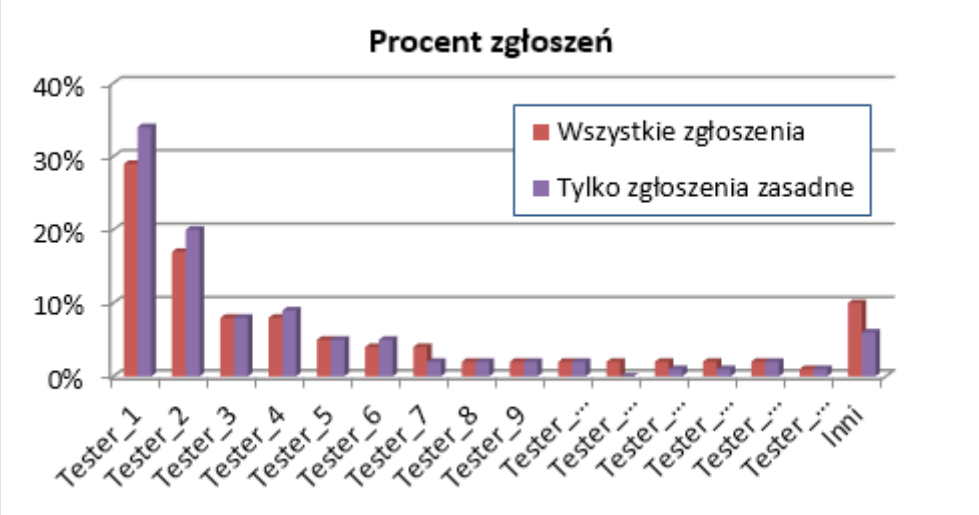
Niestety, takie podejście miałoby też sporo wad.

- Samo przygotowanie scenariuszy testowych pozwoliło na zidentyfikowanie błędów i luk w dokumentach analizy systemowej. Zwykły przegląd takich materiałów kończy się często przega-

Tabela 1. Liczba zgłoszeń poszczególnych testerów

Tester	Wszystkie zgłoszenia		Tylko zgłoszenia zasadne		Procent zgłoszeń zasadnych
	Liczba	Procent	Liczba	Procent	
Tester_1	581	29%	569	34%	98%
Tester_2	345	17%	337	20%	98%
Tester_3	166	8%	127	8%	77%
Tester_4	154	8%	141	9%	92%
Tester_5	102	5%	75	5%	74%
Tester_6	76	4%	75	5%	99%
Tester_7	72	4%	35	2%	49%
Tester_8	49	2%	27	2%	55%
Tester_9	44	2%	28	2%	64%
Tester_10	43	2%	41	2%	95%
Tester_11	43	2%	5	0%	12%
Tester_12	36	2%	23	1%	64%
Tester_13	36	2%	19	1%	53%
Tester_14	35	2%	26	2%	74%
Tester_15	30	1%	10	1%	33%
Inni	203	10%	119	6%	59%
Razem	**2 015**		**1 657**		

Rysunek 1. Procent zgłoszeń poszczególnych testerów

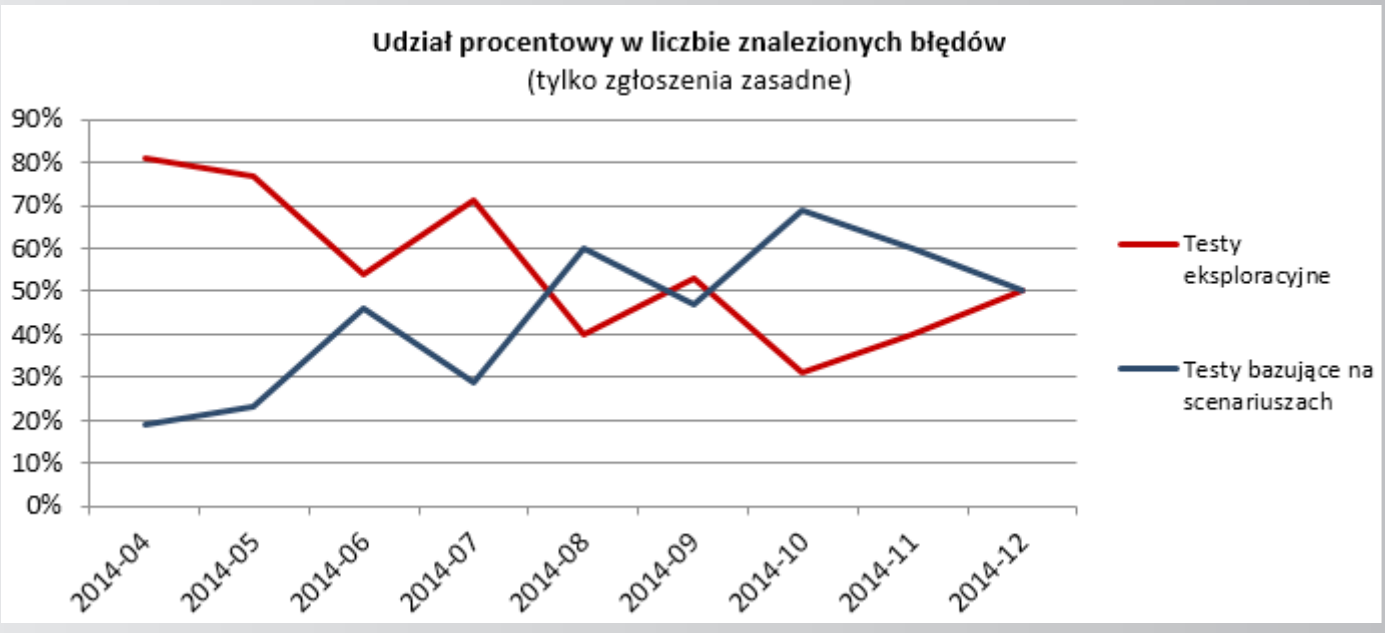


pieniem usterek i sprzeczności w algorytmach. Dopiero przygotowanie scenariuszy testowych wymusza dokładne wczytanie się w dokumentację i pozwala zidentyfikować błędy bez faktycznego wykonania testów algorytmów.

- Na każdym cotygodniowym statusie projektu słyszałem 3 pytania: ile mamy błędów, jaka jest dynamika liczby błędów oraz **jaki jest procent wykonania scenariuszy testowych**. Klasyczne podejście na podstawie scenariuszy testowych pozwala w łatwy sposób raportować procent wykonanych testów i stopień pokrycia aplikacji testami. Niestety, odpowiedź „przetestowaliśmy eksploracyjnie cały system i znaleźliśmy sto nowych błędów" nie przekonywała zarządu.
- Rozkład usterek znajdowanych obydwoma metodami nie był stały w czasie. Jak widać na rysunku 2, w kolejnych miesiącach coraz więcej błędów identyfikowano za pomocą scenariuszy testowych. Przyczyny były trzy.

1. Testy eksploracyjne są najciekawsze, gdy znajduje się intrygujące błędy. Kiedy aplikacja jest już „przeczesana", zaczynają nużyć. Wówczas lepiej sprawdza się metodyczność scenariuszy testowych.
2. Wraz z upływem czasu rosło doświadczenie pozostałej części zespołu, co pozwoliło na przeprowadzanie skuteczniejszych testów.
3. Im bardziej zbliżało się wdrożenie, tym mocniej testerzy „eksploracyjni" byli obciążeni innymi zadaniami (takimi jak weryfikacja instrukcji użytkownika, przygotowanie parametryzacji produkcyjnej itp.). W efekcie mogli poświęcać coraz mniej czasu na testowanie.

Rysunek 2. Udział procentowy testów eksploracyjnych w liczbie zarejestrowanych incydentów



- Ryzyko. Gdy testy są realizowane przez 5 osób z dużym doświadczeniem, nieobecność jednej z nich powoduje duże problemy dla projektu, zwłaszcza jeżeli brakuje scenariuszy testowych. Podejście bazujące na dokumentacji testowej pozwala znacząco zmniejszyć to ryzyko i ułatwia zastąpienie danego członka zespołu.

Pozostaje więc łączyć testy eksploracyjne i testy bazujące na scenariuszach testowych, tak aby maksymalnie wykorzystać zalety obu metod. Ciekawe podejście do tego zagadnienia przedstawił James A. Whittaker w książce *Exploratory Software Testing*. Publikacja ta powstała, zanim zaczął on wieszczyć rychły koniec manualnego testowania. W całości poświęcona jest zagadnieniu realizacji testów ręcznych.

Podejście turystyczne do testów eksploracyjnych (tours tests)

Whittaker obrazowo porównuje testy eksploracyjne do zwiedzania nieznanego miasta. Metaforę tę rozwija w całej książce, zestawiając możliwe podejścia do poznawania nowych miejsc z różnymi strategiami realizacji testów eksploracyjnych. W efekcie przedstawia kilkanaście grup „wycieczek" pozwalających identyfikować odmienne kategorie błędów. Poniżej przedstawiam kilka z nich.

Wycieczka z przewodnikiem

Podejście jest proste – wielu turystów uzbrojonych w książki i przewodniki podpowiadające, co i jak najlepiej zwiedzić, rusza w nieznane sobie miasto. W przypadku oprogramowania takim przewodnikiem może być instrukcja użytkownika lub pomoc dostępna w samej aplikacji. Zadaniem testera jest podążanie przez program dokładnie według nakazów instrukcji. Cel: znalezienie błędów – zarówno w aplikacji, jak i w dokumentacji użytkownika [6].

Wycieczka na podstawie punktów orientacyjnych

Zamiast żmudnie realizować scenariusz opisany w przewodniku, wielu z nas woli przygotować listę atrakcji wartych zobaczenia, a następnie zwiedzić je w kolejności dogodnej dla siebie. Analogicznie możemy podejść do testów. Na bazie dokumentacji lub materiałów reklamowych wypisujemy najważniejsze funkcjonalności. Następnie wykorzystujemy je w różnej kolejności. W efekcie powinniśmy „zwiedzić" całe oprogramowanie, a jednocześnie użyć jego modułów z różnymi danymi wejściowymi,tym samym zwiększając szansę na znalezienie błędów.

Podejście „taksówkarskie"

W każdym mieście z punktu A do punktu B możemy dostać się wieloma drogami – jedna będzie najkrótsza, inna umożliwi ominięcie korków, a kolejna przy okazji pozwoli napawać się pięknymi widokami. Wszystkie potencjalne trasy powinien znać kierowca taksówki – stąd nazwa podejścia.

W przypadku oprogramowania jest podobnie – do danej funkcjonalności dostajemy się wieloma drogami. Możemy użyć menu, skrótu klawiszowego, przejść z poziomu innej funkcji itp. Powinniśmy więc metodycznie przetestować każde takie wejście, aby potwierdzić, że działa.

Jeszcze ciekawszy będzie jednak test odwrotny – gdy użycie jakiejś funkcji jest w danym momencie zabronione, warto sprawdzić, czy programista pamiętał o za-

bezpieczeniu każdej możliwej ścieżki jej wywołania. Gwarantuję, że o którejś zapomniał.

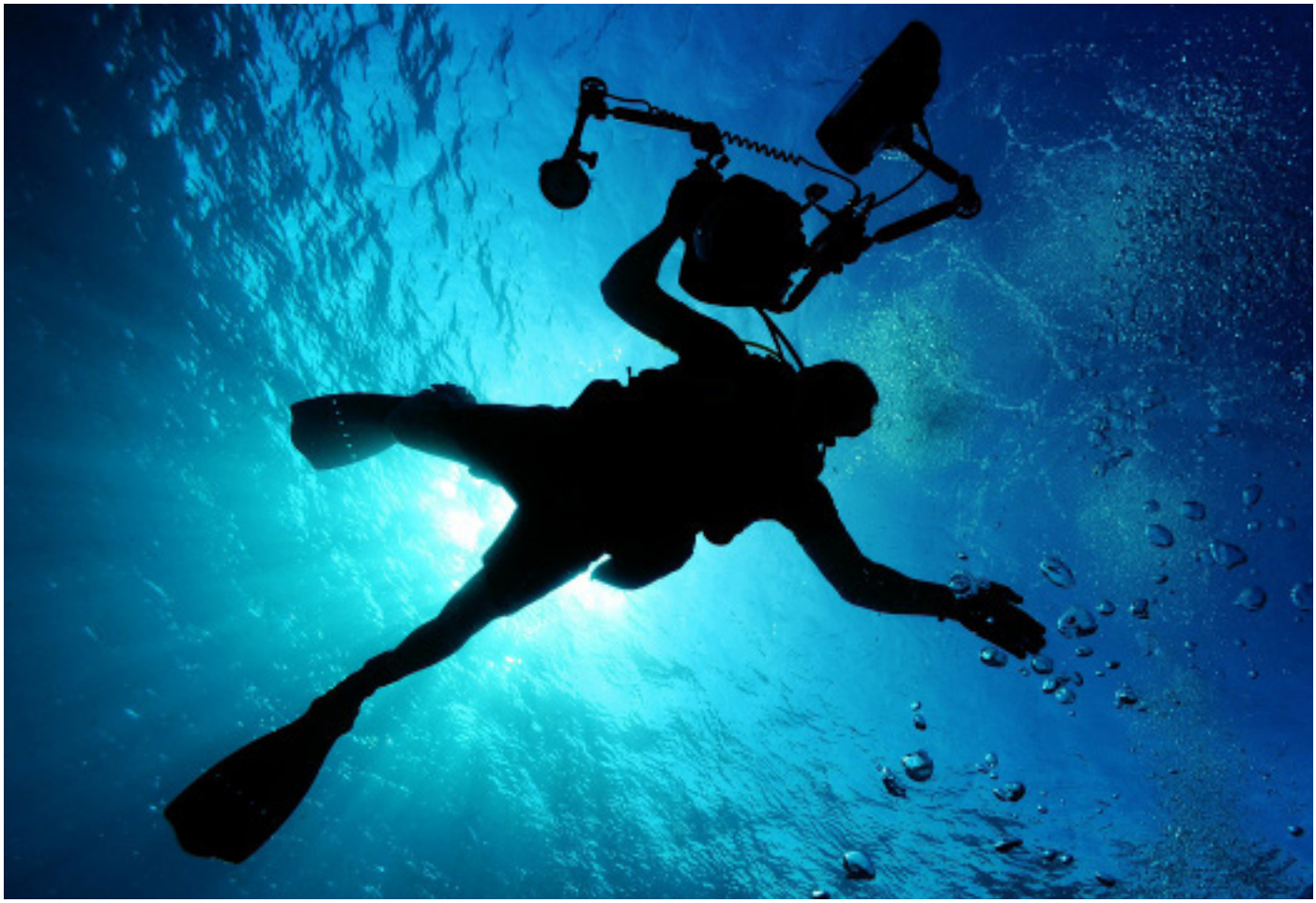
Podejście à la FedEx

Whittaker używa firmy FedEx jako ikony branży kurierskiej – przedsiębiorstwa, które przekazuje przesyłkę poprzez sieć magazynów rozsianych na całym świecie w taki sposób, by ostatecznie dotarła w przewidywalnym czasie do finalnego odbiorcy.

W przypadku oprogramowania zamiast paczki powinniśmy śledzić dane. System umożliwia wprowadzenie ich w jednym miejscu, by po licznych transformacjach użyć w zupełnie innym. Ciekawym ćwiczeniem będzie wiec prześledzenie drogi wybranej danej – od miejsca, gdzie została wprowadzona, przez wszystkie funkcjonalności, w których została zaprezentowana lub użyta. Jeżeli będzie to liczba z kilkoma miejscami po przecinku, gwarantuję, że uda się znaleźć fragment kodu stosujący nieodpowiednie zaokrąglenie lub formularz wyświetlający ją niezgodnie z oryginalną precyzją.

Zrezygnowany turysta

Czasem pogoda zmusza turystę do porzucenia zwiedzania. Podobnie jest w trakcie testów – oprogramowanie często pozwala na przerwanie czynności, którą właśnie zainicjowaliśmy. To genialna kopalnia błędów. Przerwana funkcja z dużym prawdopodobieństwem nie posprząta po sobie lub wręcz oprogramowanie nie pozwoli na zakończenie procesu w połowie drogi. Każdy przycisk *Anuluj* powinien zatem mówić testerowi „wciśnij mnie!”.



Podziwianie widoków

W oryginale wycieczka ta nosi nazwę *The Supermodel Tour*. Jej istota polega na tym, by skoncentrować się wyłącznie na wyglądzie oprogramowania, a nie na jego funkcjonalności. Czy wszystkie tabele są posortowane tak samo? Czy formatowania na wszystkich ekranach są spójne? Czy dane w kolumnach zostały właściwie wyrównane? Czy układ pól i kolejność ich przechodzenia przy użycia klawisza TAB są intuicyjne na wszystkich formularzach? To przykłady wycieczek z tej kategorii.

W książce Whittakera można znaleźć dużo więcej gotowych „wycieczek"; są one również dokładniej opisane. Mam nadzieję, że tych kilka zaprezentowanych powyżej pozwoliło na zrozumienie samej koncepcji.

Jak połączyć testy bazujące na scenariuszach i testy eksploracyjne

Po tej krótkiej dygresji wróćmy do opisywanego projektu i rozważań, co bym w nim zmienił, aby lepiej połączyć testy eksploracyjne z testami bazującymi na scenariuszach testowych.

Koncepcja Whittakera pozwala nieco ujarzmić chaos i losowość testów eksploracyjnych. Umożliwia też dokumentowanie i raportowanie poczynań testerów. Jak sobie wyobrażam zastosowanie jej w następnym projekcie?

- W pierwszym kroku na bazie wymagań biznesowych i innych tego typu dokumentów chciałbym przygotować listę funkcjonalności (potencjalne „punkty widokowe"). Jednocześnie chciałbym zidentyfikować te z nich, dla których przygotowanie scenariuszy testowych będzie konieczne.

  Scenariusze testowe powinny objąć krytyczne elementy systemu, których działanie będzie się opierać na sformalizowanych algorytmach. Przygotowanie w tym obszarze scenariuszy testowych pozwoli zweryfikować poprawność algorytmów, a jednocześnie zapewni dużą dokładność i powtarzalność testów. Jednocześnie takich funkcjonalności nie powinno być zbyt dużo, tak aby umożliwić aktualizację scenariuszy przy każdej zmianie wymagań.

- Na podstawie powyższego zestawienia przygotowano by konkretne scenariusze testowe oraz listę „wycieczek" realizowanych w ramach testów eksploracyjnych. Przykładowo dla zwiedzania opierającego się na punktach orientacyjnych powstałby zbiór wycieczek, w którym każdej z nich przypisana byłaby konkretna lista punktów orientacyjnych do zwiedzenia wraz z kolejnością zwiedzania. Analogicznie utworzono by listę danych, których życie w aplikacji należy prześledzić, spis cech interfejsu do zweryfikowania w ramach wycieczek skupionych na „widokach" itp.

- W trakcie testów oraz na potrzeby raportowania wycieczki byłyby traktowanie analogicznie do scenariuszy testowych. Przypisywano by je do poszczególnych przebiegów testów i testerów. A co najważniejsze – byłoby możliwe raportowanie postępu testów jako procent wykonanych scenariuszy i wycieczek.

- Proste wycieczki, np. skupione na śle-

REFERENCJE

1. *Certyfikowany tester. Plan poziomu podstawowego (sylabus ISTQB)*, SJSI 2011.
2. James A. Whittaker, Jason Arbon, Jeff Carollo, *Testuj oprogramowanie jak Google. Metody automatyzacji*, Gliwice 2014.
3. Wiktor Żołnowski, *(Manual) Test is dead!*, „Quale" 2/2014.
4. Tomasz Olszewski, *Automation testing is not testing*, „Quale" 3/2014.
5. Opinia zamieszczona na blogu Jamesa Bacha i Michaela Boltona Exploratory Testing 3.0.
6. Skrajnym przykładem takich testów będą testy opierające się na scenariuszach testowych.

dzeniu konkretnej danej lub weryfikacji cechy interfejsu graficznego, mogłyby być z łatwością realizowane także przez mniej doświadczonych testerów (w tym osoby nieznające systemu). Jednocześnie pozwalałyby im zapoznać się z całą aplikacją i w efekcie realizować coraz trudniejsze scenariusze.

- Dla najtrudniejszych wycieczek, realizowanych przez doświadczonych testerów, powinna powstawać dokumentacja ich wykonania w postaci użytych danych i uzyskanych odpowiedzi. Dzięki temu w kolejnych przebiegach testów lub w testach regresyjnych byłoby możliwe odtworzenie ich przez mniej doświadczonych członków zespołu.

Przedstawione powyżej podejście jest tylko teorią (i to dosyć ogólną). I pozostanie nią, dopóki nie zostanie zastosowane w praktyce. Whittaker przytacza w swojej książce przykłady użycia „wycieczek" w testach realizowanych przez Microsoft. Może i ja będę miał szansę zastosować je w praktyce – oczywiście w dużo mniejszej skali.

Tak czy inaczej książkę Whittakera polecam wszystkim testerom. Teoretycznie każda z prezentowanych w niej wycieczek dotyczy możliwego podejścia do testów eksploracyjnych. Jednak tak naprawdę dostarcza wskazówek na temat tego, w jaki sposób tester powinien myśleć, aby znajdować błędy. Nie ma znaczenia, czy przygotowuje scenariusz testowy, kod automatyzujący testy, czy też realizuje testy eksploracyjne.

AUTOR

Grzegorz Dawid

Absolwent studiów informatycznych na Politechnice Wrocławskiej (1998-2003). Od ponad 10 lat uczestniczy w tworzeniu oprogramowania, głównie dla branży finansowej. Obecnie kieruje zespołem testowym w projektach IT w jednym z banków.

Radosław Smilgin

Mr Buggy – (nie)typowy projekt informatyczny

Realizowałem w życiu wiele projektów informatycznych. Z całym szacunkiem dla wszystkich tego typu przedsięwzięć – Mr Buggy jest najbardziej oryginalny. Abstrahując od jego przeznaczenia i miejsca używania, to produkt jednocześnie typowy i nietypowy.

Kilka tygodni po zakończeniu TestingCup wreszcie można odetchnąć i wrócić pamięcią do ostatnich miesięcy przygotowywania przeciwnika wszystkich testerów – Mr Buggiego – i ostatecznego starcia na Stadionie Wrocław. Gdy patrzy się na to z zewnątrz, być może trudno sobie wyobrazić skalę pracy. Mr Buggy to aplikacja, która angażuje cały mój zespół już 5 miesięcy przed Mistrzostwami w Testowaniu Oprogramowania.

Zanim do pracy ruszą twórcy i weryfikatorzy, jest etap koncepcyjnego wymyślania aplikacji i jej projektowania. Tworzymy specyfikację, testujemy ją w ramach zespołu, który później stworzy Komisję Sędziowską. Potem grafik projektuje ekrany i logo kolejnej wersji, programiści próbują przełożyć specyfikację wymagań na produkt, a testerzy w pocie czoła starają się znaleźć „wszystkie" defekty. Jak wiadomo, nie udaje się nam to i co roku uczestnicy TestingCup udowadniają nam, ile razy się pomyliliśmy. Tak było w 2013 i 2014 r. kiedy to Mr Buggy 1 i 2 zawodnicy Mistrzostw szukali defektów i oceniali jakość Mr Buggy 1 i 2. Wówczas defekty nie były „zaprojektowane"; zostały znalezione (lub nie) i pozostawione bez naprawy. To wzmacniało naturalność samej aplikacji.

W 2015 r. koncept został zmieniony. Uznaliśmy, że nie jesteśmy w stanie przejąć pełnej kontroli nad aplikacją, więc na początku stworzyliśmy stabilną platformę, a potem wrzuciliśmy do Mr Buggy 24 zadania z 24 defektami. Posiane usterki nie są wyssane z palca. Zostały wyczytane z systemów raportowania incydentów z projektów informatycznych, w jakich mieliśmy okazję brać udział. Odwołując się do

analogii ze świata malarstwa, są to doskonałe reprodukcje prawdziwych dzieł sztuki (programistycznej).

To nowe podejście ma nam pomóc oceniać testerów poprzez częściową automatyzację procesu sprawdzania prac. Podczas poprzednich edycji bardzo nam tego brakowało. Po przeczytaniu ankiet i podsumowaniu rozmów z zawodnikami TestingCup mamy potwierdzenie, że to rozwiązanie zostało przez uczestników zaakceptowane, choć widzimy, że nie jest doskonałe. Nikt, kto zna się na zarządzaniu czy na motywacji i psychologii, nawet nie próbuje oceniać pracy testerów ilościowo. Zawsze powinno być to złożenie miar ilościowych i jakościowych oraz wypadkowej ocen subiektywnych i obiektywnych. Z Mr Buggim dokonujemy więc rzeczy teoretycznie niemożliwej i próbujemy wyłonić najlepszych testerów. Ocena pracy to wypadkowa zdolności reprodukowania defektów, umiejętnego dobierania kategorii problemu oraz dobrego opisu zgłoszenia.

Wiemy, że nasza metoda nie była doskonała. Między raportami dostarczanymi przez różnych zawodników i drużyny jest przepaść. Łatwo więc wskazać grupę najlepszych. Jak jednak obiektywnie wybrać najlepszych z najlepszych i ustawić ich w odpowiedniej kolejności na podium? Może podobnie jak w przypadku skoków narciarskich, przy skokach na tę samą odległość, wykonanych w podobnych warunkach, liczy się średnia ocen sędziów? Czy da się jednak zapewnić sędziowie, że nie popełnią błędu przy ocenie 90 soków? Nie można wykluczyć pomyłki. Dlatego też system oceniania Mr Buggy 3 w wersji przygotowanej na eliminacje był zautomatyzowany, choć pozostawał pod pełną kontrolą Komisji Sędziowskiej. Była ona wspierana przez Managera Mr Buggy, niewidocznego dla uczestników imprezy. Mówimy o skomplikowanym mechanizmie serwerowym, w którym wyniki są zbierane online, tam przetwarzane, a potem za pomocą dodatkowego serwera wyświetlane na ekranach jako klasyfikacja. Wszystko to z wbudowanym mechanizmem podglądu prac testerskich i opcją ich weryfikacji. Co ważne, Komisja mogła nadpisać każdą automatyczną ocenę.

Warto zwrócić uwagę na fakt, że realizacja projektu, jakim jest TestingCup czy też sam Mr Buggy, to ewenement na skalę światową. W takim zakresie jak w Polsce nie robi tego nikt. Była próba organizacji Mistrzostw Świata – Software Testing World Championship – ale koncept został porzucony po jednej edycji (chyba że formułą jest przeprowadzanie zawodów co cztery lata). Do takiego przedsięwzięcia przymierzali się Szwedzi, Bułgarzy, Belgowie… Nawet słynny EuroStar chciał spróbować, ale wszyscy się wycofali. Dlaczego? Prawdopodobnie dlatego, że to absolutne szaleństwo! Nakład pracy, koszty i ryzyko braku zwrotu z inwestycji wydały się wszystkim


Fotografia: ©TestingCup

zbyt duże. Złożoność organizacyjna, liczba możliwych pułapek czy po prostu ryzyka, że coś pójdzie nie tak, są olbrzymie i przerastają ludzi, którzy o Mistrzostwach w Testowaniu myślą jedynie w kategoriach komercyjnych. Potrzeba więc czegoś więcej – odrobiny szaleństwa, dużo pasji i zespołu, który pomyśli o wszystkim.

Czy wiecie, że plany awaryjne dla Mr Buggiego 2 liczone były do literki „G"? Plan A – wersja serwerowa lokalna, plan B – wersja lokalna (desktopowa), plan C – wersja serwerowa w Internecie… aż po plan G – zupełnie inna aplikacja desktopowa. Jeden z liderów projektu Mr Buggy jest istnym perfekcjonistą i obmyślił nieskończoną liczbę pomysłów na to, co mogłoby pójść nie tak. Aplikacja na zawody to mocny przeciwnik zarówno dla zespołu wytwórczego, jak i dla uczestników TestingCup. Przy czym zespół wytwórczy myśli o uczestnikach jako o potencjalnym źródle kłopotów.

- Jeśli nasza aplikacja będzie online, a na sali pojawi się człowiek, który będzie zakłócał fale radiowe, jak zagwarantujemy wszystkim równy dostęp do oprogramowania? Stwórzmy aplikację desktopową.
- A jeśli w przypadku aplikacji online ktoś będzie chciał przeciążyć serwer? Przeprowadźmy testy na maksymalnym obciążeniu pomnożonym razy 3.
- A jeśli ktoś będzie chciał zdekompilować aplikację? Przenieśmy logikę na serwer.
- A jak ktoś podejrzy nasz adres serwera i będzie próbował się do niego dostać? Zablokujmy wszystkie IP oprócz naszych i dodatkowo ukryjmy adres aplikacji w pseudoadresie z przekierowaniem w plik vhost.
- A jeśli serwer padnie? Weźmy ze sobą jeszcze jeden komputer, a najlepiej dwa…

I tak bez końca. Jest ryzyko, jest plan awaryjny. Nie można wymyślić planu awaryjnego – trzeba przebudować całą aplikację. Wszystko po to, by tego jednego dnia przez 2-3 godziny testerzy oprogramowania z całego kraju mieli szansę… powrócić do swoich codziennych zawodowych obowiązków. Z tą drobną różnicą, że mogą się tu zmierzyć z innymi ludźmi (co trudno wyobrazić sobie w pracy) i wygrać parę tysięcy złotych.

Mr Buggy będzie w pewnych obszarach standardowym projektem. Powstaje dla wymagającego odbiorcy, próbuje rozwiązywać wiele problemów w fazie projektowania – z założeniem, że pomyłki są niedopuszczalne. Dostarcza dużo frajdy, ale i mnóstwo stresu. Pod względem godzin poświęconych na testowanie aplikacji może równać się z wieloma komercyjnymi systemami.

Na swój sposób jest to też projekt specyficzny. Testujemy go i wyszukujemy w nim defekty, żeby ostatecznie ich nie usuwać. Naszym największym zmartwieniem jest to, żeby aplikacja działała 3 godziny; poświęcamy na to kilka tysięcy roboczogodzin. Nie zapominamy o tym, że aplikację przygotowujemy nie dla amatorów, lecz dla zawodowców ze świata jakości.

Mr Buggy to projekt, którego nie wyobrażam sobie zakończyć. Wraz z zespołem przygotowuję się zatem do 4. edycji zawodów.

AUTOR

Radosław Smilgin

Niezależny konsultant i trener z zakresu testowania i zapewnienia jakości aplikacji.

Tester oprogramowania z zamiłowania. Właściciel serwisu testerzy.pl.

Autor publikacji z obszaru testowania i zapewniania jakości. Prelegent polskich i zagranicznych konferencji testerskich: TestWarez, PTI, Free Test, CzechTest i wielu innych. Ambasador największych konferencji: EuroSTAR, Agile Testing Days, BTD.

Fan użyteczności i testowania w Agile.

Pomysłodawca i organizator TestingCup – Mistrzostw Polski w Testowaniu Oprogramowania.

Akredytowany przez Stowarzyszenie Jakości Systemów Informatycznych trener ISTQB oraz tłumacz sylabusa i słownika ISTQB.

Fotografia: ©TestingCup

Adrian Brodny
Zuzanna Gościcka-Miotk
Piotr Krajewski
Patrycja Nowicka
Dominika Wojtko
Katarzyna Zatorska-Radlińska

Testy eksploracyjne – analiza podejścia i wnioski z realizacji

Wprowadzenie

Testowanie oprogramowania to złożona dziedzina, oferująca wiele technik i narzędzi, które jednak należy rozważnie dopasowywać do konkretnych przypadków. W zależności od stopnia skomplikowania systemu, rodzaju i przeznaczenia aplikacji oraz kontekstu biznesowego, budżetu i innych ograniczeń decydujemy się na odmienne podejścia. Niekiedy charakter oprogramowania ogranicza nas do stosowania testów automatycznych. Obecnie świat IT zdaje się koncentrować właśnie na nich. Wielu ekspertów zwraca uwagę na fakt, że testy automatyczne pozwalają wykrywać więcej usterek, ponadto uzyskiwane efekty są mierzalne i weryfikowalne, a raz zaprojektowany test można z powodzeniem wielokrotnie wykorzystywać i modyfikować.

Czy popularność takiego podejścia oznacza, że wkrótce w ogóle zrezygnujemy z eksploracji? Raczej nie. Prawdopodobnie nigdy nie powstanie oprogramowanie, którego jedyną i najlepszą metodą testowania będzie podejście automatyczne. Aplikacje tworzone są dla klientów. A przecież każdy człowiek ma odmienne preferencje, oczekuje od systemu określonego zachowania, wyglądu, inaczej rozumie intuicyjność i wygodę obsługi; krótko mówiąc, w tej kwestii nie jest obiektywny. Wreszcie najważniejsze – przeciętny użytkownik nie dysponuje taką samą wiedzą na temat budowy i przeznaczenia oprogramowania jak jego twórca. Wszystko to sprawia, że ograniczanie się do testów automatycznych mija się z celem – gustów, oczekiwań i aktywności człowieka nie da się na sztywno zamknąć w algorytmach. Opra-

cowując testy automatycznie, zakładamy najbardziej prawdopodobne sposoby wykorzystywania oprogramowania. Tymczasem motto testera powinno brzmieć raczej: „Spodziewaj się niespodziewanego". Użytkownik to nie maszyna – nie da się zaprogramować go na stosowanie aplikacji zgodnie z jej przeznaczeniem. Ludzie są z natury ciekawi świata, lubią eksperymentować. Zadanie całego zespołu osób uczestniczących w tworzeniu oprogramowania (m.in. testerów) polega na zapewnieniu, że niestandardowe użycie systemu nie przyniesie nikomu szkody, a gotowy produkt spełni oczekiwania klienta.

Niezbędne jest zatem wyjście poza schemat i wcielenie się w rolę potencjalnego odbiorcy aplikacji. Zadanie to okazuje się jednocześnie ekscytujące i pełne pułapek. Jak pisze Elisabeth Hendrickson w książce *Explore It!*:

„Eksplorowanie oprogramowania ma wiele wspólnego z eksplorowaniem terenu:

- *Czeka cię mnóstwo niespodzianek i wyzwań (włączając w to bugi).*
- *Podczas podróży możesz wspomagać się narzędziami, jednak najważniejsze, które masz przy sobie, znajduje się między twoimi uszami.*
- *Czasami eksplorowanie przypomina wesołe hasanie, innym razem to harówka, zdarza się też, że stąpasz po niepewnym gruncie.*
- *Jeśli mapa i wygląd terenu różnią się, zaufaj terenowi"* [2].

Siłę testowania eksploracyjnego podkreśla również James A. Whittaker w książce *Exploratory Software Testing*: *„Zwolennicy [podejścia eksploracyjnego – przyp. ZGM] przekonują, że pozwala ono wykorzystać pełną moc ludzkiego umysłu do wynajdowana defektów i weryfikowania funkcjonalności bez wcześniej założonych ograniczeń [i rezultatów – przyp. ZGM]"* [9].

Eksploracja zapewnia zatem większą swobodę i umożliwia identyfikowanie problemów, których nie wykryją testy automatyczne. Nie wymaga sztywnych scenariuszy, pozwala również angażować w cały proces osoby nieposiadające technicznych umiejętności i stawiające pierwsze kroki w zawodzie. Tacy testerzy nierzadko są bardzo spostrzegawczymi obserwatorami i potrafią wyszukiwać usterki, których nie zauważają pozostali członkowie zespołu przygotowującego oprogramowanie.

Wielu ekspertów z dziedziny IT – mimo obecnej popularności testów automatycznych – nadal wysoko ceni podejście eksploracyjne. Wystarczy wymienić takie osoby, jak przywołani już Elisabeth Hendrickson (dawniej m.in. testerka i programistka; obecnie konsultantka w Quality Tree Software; prowadzi również blog Test Obsessed) czy James A. Whittaker (jedna z ważniejszych postaci w Microsofcie; w przeszłości współpracował m.in. z Google i IBM).

Na popularność testowania eksploracyjnego z pewnością wpłynęły liczne publikacje Cema Kanera, prof. inżynierii oprogramowania we Florida Institute of Technology. Książka *Testing Computer Software*, napisana przez Kanera wraz z Jackiem Falkiem i Hung Quoc Nguyenem, zyskała status bestsellera wśród prac dotyczących tej dziedziny.

Nie sposób pominąć również Jamesa Bacha, który postuluje wręcz, że każde testo-

wanie jest testowaniem eksploracyjnym. Ten kontrowersyjny guru sporej grupy testerów, od 12 lat piszący blog o tematyce IT, niedawno wydał wraz z Michaelem Boltonem książkę *Exploratory Testing 3.0*. Autorzy przedstawili w niej ewolucję testowania eksploracyjnego: od początków w 1968 r. aż po współczesność.

Co odróżnia eksplorację od innych podejść? Kiedy warto ją stosować, a w jakich sytuacjach lepiej korzystać z innych rozwiązań? Kim jest tester eksploracyjny i z jakimi problemami musi się mierzyć? Jak wygląda sesja testowa?

Przypadek 1. Podejście eksploracyjne – kiedy, dlaczego oraz przez kogo może być wykorzystane?

Klemens (kierownik projektu) i Alojzy (kierownik zespołu testerów) spotkali się, by omówić możliwe metody testowania przygotowywanej aplikacji dla pilotów i wybrać optymalne rozwiązanie. Firma jakiś czas temu podpisała kontrakt z jedną z linii lotniczych. Wszyscy byli podekscytowani – do tej pory organizacja współpracowała z operatorami z zupełnie innych branż.

Przy okazji rozmowy okazało się, że obaj panowie od dziecka chcieli być pilotami. Wkrótce merytoryka dyskusji gdzieś się ulotniła, a zastąpiły ją sentymentalne wynurzenia. Klemens i Alojzy stracili poczucie czasu. Od kilku godzin z wypiekami na twarzy opowiadali o swoich pierwszych lotach, ponętnych stewardesach i podróżach w kokpicie. Gdy okazało się, że upłynął czas przewidziany na zapoznanie się z wymaganiami i wybranie najlepszej metody testowej, panowie spojrzeli na siebie nieco zaskoczeni.

– To co, eksplorujemy? – zaproponował Klemens.

Alojzy, w którego uszach najwyraźniej rozbrzmiewał jeszcze ryk silników samolotu, nie protestował.

Panowie wrócili do swoich obowiązków. Metoda planowania i realizacji testów została wybrana.

Gdy Alojzy przedstawił plany zespołowi, kilka osób nie kryło zdziwienia. Wspominały coś o dokumentacji, procedurach, konieczności przygotowania danych, opracowania scenariuszy, o dokładnym testowaniu i dowodach pokrycia… Kierownik zespołu nie bardzo jednak ich słuchał.

Wkrótce testerzy rozpoczęli eksplorację. Testowanie było przyjemne, krótkie i powierzchowne.

Otrzeźwienie przyszło nagle – w pewnym momencie okazało się, że „przetestowany" system nie tylko nie spełnia oczekiwań klienta, lecz również jest niezgodny z przepisami obowiązującymi w lotnictwie. Oczywiście nie obyło się bez kar umownych, terminy trzeba było wydłużyć, a testy wykonać od nowa – tym razem już wykorzystując zupełnie inne podejście i metody. Stało się jasne, że tworzenie systemów krytycznych jest bardzo specyficzne, wymaga ogromnej dokładności, bazowania na licznych dokumentach, procedurach, przepisach i standardach. Tu nie ma miejsca na pomyłki, bo każde niedociągnięcie może być tragiczne w skutkach. A zatem w przypadku tego typu systemów ograniczenie się jedynie do eksploracji to kiepskie rozwiązanie.

Jakimi zatem kryteriami powinniśmy się kierować, wybierając podejście do testów?

Na wszystkie te pytania odpowiadamy w artykule. Każdy z rozdziałów teoretycznych poprzedziliśmy krótką historyjką, przedstawiającą omawiany problem w praktyce. Ponadto przeprowadziliśmy testy eksploracyjne niewielkiej aplikacji i opracowaliśmy wnioski. Na koniec oceniliśmy przydatność podejścia eksploracyjnego w tym konkretnym przypadku.

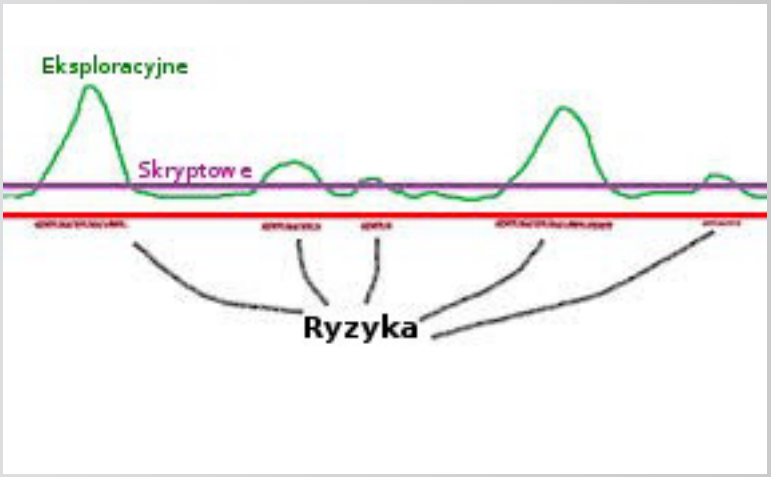
Podejście eksploracyjne – kiedy, dlaczego oraz przez kogo może być wykorzystane?

Zastosowanie testów eksploracyjnych

Tylko od nas zależy, jakie metody i techniki zastosujemy, by uzyskać informację o jakości, zwiększyć zaufanie do produktu, a po naprawieniu usterek – także podnieść jego jakość. W niniejszym rozdziale zaprezentujemy sposoby oraz recepty na wykorzystanie testów eksploracyjnych.

Kiedy warto testować eksploracyjnie? Wtedy, gdy mamy do czynienia ze szczątkową dokumentacją albo wręcz z jej brakiem, jak również wówczas, gdy dysponujemy bardzo małą ilością czasu na przeprowadzenie testów. W takich wypadkach przychodzi nam z pomocą podejście eksploracyjne, które pozwala poznać oraz zrozumieć wytwarzany produkt w relatywnie krótkim czasie. Na warunek konieczny do zastosowania testów eksploracyjnych składają się wiedza biznesowa oraz doświadczenie osoby przeprowadzającej testy. Za pomocą tego podejścia tester jest w stanie szybciej i często efektywniej przetestować aplikację, ponieważ równolegle projektuje oraz wykonuje testy, a zebrane na tej podstawie informacje wykorzystuje do projektowania nowych, lepszych przypadków testowych. Podejście eksploracyjne może pomóc w planowaniu zakresu i czasu trwania innych rodzajów testów. Przeprowadzając testy eksploracyjne, poznajemy działanie produktu. Dzięki temu jesteśmy w stanie dokładniej oszacować czas potrzebny do przetestowania całej aplikacji oraz poznać newralgiczne punkty, wymagające np. dopisania nowych scenariuszy testowych. Podczas testowania eksploracyjnego może bowiem okazać się, że w niektórych obszarach systemu wystę-

Rysunek 1. Ryzyko występowania błędów a intensywność testowania [3]



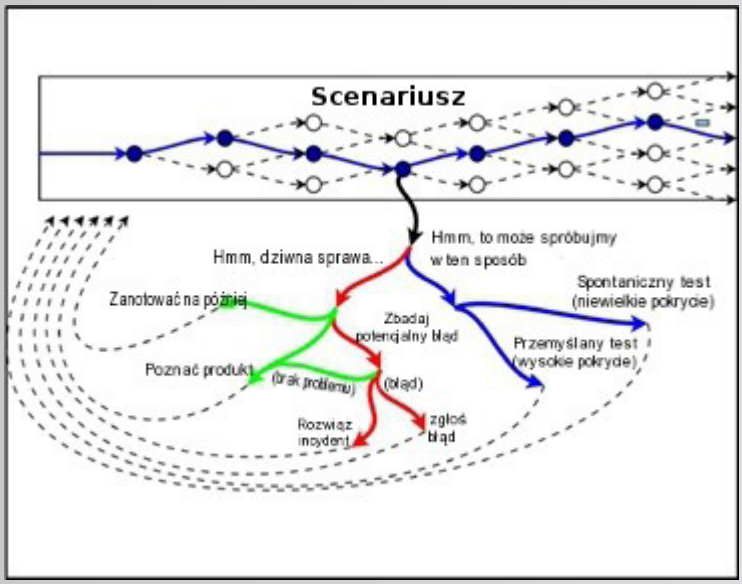
puje większe ryzyko wystąpienia błędów, którego wcześniej się nie spodziewaliśmy (rysunek 1).

Podejście eksploracyjne pozwala również znaleźć wcześniej niewidoczne zależności między modułami, co może wpłynąć znacząco na metody i techniki wykorzystywane podczas testowania, jak i na czas trwania samych testów. Niestety, dokumentacja (spis wymagań, specyfikacja czy instrukcje/podręczniki użytkownika) często nie dostarcza pełnej informacji o działaniu systemu oraz o tym, jak zachowa się on w poszczególnych sytuacjach. W testach eksploracyjnych wcielamy się niejako w rolę użytkownika i sprawdzamy, w jaki sposób nasze działania wpływają na funkcjonowanie aplikacji. Według Jamesa Bacha eksplorację należy stosować wówczas, gdy nie jest oczywiste, co będzie sprawdzane w kolejnym kroku albo kiedy zamierzamy wykroczyć poza oczywisty przypadek testowy (rysunek 2) (*„In general, ET is called for in any situation where it's not obvious what the next test should be, or when you want to go beyond the obvious tests"* [1]).

Poniżej sytuacje, w których warto zdecydować się na tego typu testy:

- musimy szybko dostarczyć informacje o działaniu i jakości nowego produktu albo funkcjonalności;
- musimy szybko i w odpowiednim stopniu zapoznać się z produktem;
- gdy zakończyły się testy systematyczne – eksploracja pozwoli urozmaicić testy (zwiększyć ich efektywność i zredukować ryzyko efektu pestycydów);
- chcemy szybko znaleźć newralgiczne miejsca oraz krytyczne defekty;
- chcemy zweryfikować efekty pracy innego testera;
- chcemy głębiej zbadać konkretną usterkę;
- chcemy ocenić, w jakim stopniu testowanie skryptowe ma zastosowanie w danym przypadku.

Rysunek 2. Graficzny przykład wykroczenia poza oczywisty scenariusz/przypadek testowy [5]



Niekiedy nawet nie jesteśmy świadomi, że stosujemy podejście eksploracyjne w naszej codziennej pracy. Oto przykłady takich sytuacji [1]:

- improwizujemy w skryptach testowych;
- mamy do czynienia z niejasnymi scenariuszami testowymi, które musimy zinterpretować;
- analizujemy wymagania, specyfikację i planujemy testy;
- wykonujemy testy regresywne, bazując na doświadczeniach z poprzednich projektów.

Dlatego też pytanie nie powinno brzmieć „Kiedy testować eksploracyjnie?", a raczej: „Jak bardzo eksploracyjne są twoje testy i jak dobrze je wykonujesz?" (*„That means the question is not when do you do exploratory testing, but rather how exploratory is the testing that you do, and how well do you do it"* [6]).

Z eksploracji rezygnujemy zazwyczaj wtedy, gdy informacje, które otrzymujemy w wyniku stosowania tego podejścia, nie wnoszą dużej wartości dodanej albo proces uzyskiwania danych trwa zbyt długo. Powracamy wówczas do tradycyjnego podejścia do testowania. Pomijamy również to podejście podczas pracy z oprogramowaniem krytycznym oraz wtedy, gdy klient lub kierownictwo potrzebuje udokumentowanego monitorowania postępu testów albo musimy wykonać testy akceptacyjne, podczas których nie byłoby wskazane znalezienie jakichkolwiek błędów, a niestety testy eksploracyjne zwiększają ryzyko ich znalezienia.

Cechy dobrego testera eksploracyjnego

Bardzo ważnymi czynnikami wpływającymi na testy eksploracyjne są umiejętności oraz podejście testera do eksploracji. Niezbędnymi wymogami przy zastosowaniu tego podejścia są doświadczenie oraz wiedza biznesowa, bez których nie możemy mówić o testowaniu eksploracyjnym. Osoba zajmująca się eksploracją musi przede wszystkim być ciekawa, chcieć „zajrzeć do każdego kąta" testowanej aplikacji. Powinna również być zaradna, czyli potrafić szukać informacji na nurtujące ją pytania. Musi też umieć znajdować potrzebne narzędzia i uczyć się ich obsługi. Tester nie może bać się pytać oraz prosić o pomoc, a jego pytania powinny być konkretne. Musi wiedzieć, o co i dlaczego pyta. Tester powinien potrafić precyzyjnie zidentyfikować problem i umiejętnie formułować pytania dotyczące tego zagadnienia. Cytując słowa Jona Bacha: „Testowanie to zadawanie pytań, a test to pytanie w przebraniu" (*„All testing is questioning. In fact the test is just a question in disguise"* [8]).

Podczas przeprowadzania testów tester musi być ukierunkowany na cele, które przyświecają temu procesowi, oraz skupiać się na problemie. Krótko mówiąc, jego zadaniem jest podążanie ścieżką wytyczoną przez założone efekty. Po szybkim testowaniu danej funkcjonalności (np. na prośbę kolegi z zespołu) powinien powrócić do niej i bazując na informacjach, które już uzyskał, przetestować ją lepiej i dokładniej (zastosować nowe, wydajniejsze testy). Testując eksploracyjnie, możemy powiedzieć, że przeprowadzamy własne małe, świadome „eksperymenty" na aplikacji. Wykonujemy krok i obserwujemy, jak na to reaguje system. Dzięki temu

zdobywamy wiedzę o wytwarzanym oprogramowaniu. Kształtuje to świadomość tego, gdzie i dlaczego klikamy.

Tester eksploracyjny musi być dobrym obserwatorem. Powinien potrafić wychwycić podejrzane zachowanie aplikacji, czyli takie, które nie pokrywa się z oczekiwaniami czy przypuszczeniami docelowego odbiorcy. Ważne jest też, by umiał dostrzegać rzeczy łatwe do przeoczenia dla „zwykłych śmiertelników". Czasami drobne nieprawidłowości w zachowaniu/wyglądzie aplikacji są tylko niedoróbkami estetycznymi, innym razem mogą świadczyć o bardziej złożonym problemie.

Równie istotną cechą jest zdolność do interakcji oraz konfiguracji, obycia z aplikacją – klikając, nie boimy się, że coś zepsujemy. Klikamy i stwarzamy niestandardowe sytuacje, sprawdzając, jak zachowa się oprogramowanie.

Do kluczowych umiejętności testera, nie tylko eksploracyjnego, należy praca zespołowa. Testy eksploracyjne można bowiem przeprowadzać nie tylko w pojedynkę. Praca w grupie i umiejętność komunikowania się z członkami zespołu, kierownictwem oraz klientami jest niezmiernie istotna. Wszyscy jednak wiemy, że nie zawsze okazuje się to proste; nierzadko dochodzi do nieporozumień i konfliktów. W takich momentach warto pamiętać, że wszyscy dążymy do wspólnego celu, którym jest najwyższa jakość wytwarzanego produktu.

Tester musi wiedzieć, na podstawie zdobytych danych dotyczących systemu, któremu obszarowi aplikacji należy poświęcić więcej uwagi, a który nie będzie wymagał aż tak dużego zaangażowania. Powinien również zdawać sobie sprawę, że niekiedy ślepe dążenie do wytyczonego na początku celu nie ma sensu, zwłaszcza gdy wiele przesłanek kieruje nas w zupełnie innym kierunku. Przydatną cechą osoby zajmującej się eksploracją jest też identyfikowanie różnych ścieżek prowadzących do tego samego efektu. Drogi te mogą być rozłączne, ale uzyskany dzięki nim cel będzie podobny.

Do dobrych nawyków testera należy sporządzanie notatek. Usprawniają one proces i dostarczają informacji, do których zawsze będzie można powrócić. Istotna jest zatem umiejętność dokumentowania postępów w procesie testowania. Do niezbędnych cech testera, nie tylko eksploracyjnego, należy też zdolność odpowiedniego raportowania wyników swojej pracy w mowie i piśmie klientom lub przełożonym. Wymaga to zarówno umiejętności komunikacji, jak i rzetelnego zbadania i przedstawienia istoty problemu. Niekompletny opis usterki i lakoniczne komentarze to nie jest to, co tygryski (przełożeni i klient) lubią najbardziej.

Środowisko w testach eksploracyjnych

Nie ma dwóch identycznych środowisk. Nie da się gruntownie przetestować aplikacji/produktu, a nawet gdyby była taka możliwość, nigdy nie mamy pewności, czy w bardzo podobnych warunkach system będzie działał poprawnie. Dzieje się tak, ponieważ dane środowisko niekiedy decyduje o tym, jak zachowa się program. Ogólnie rzecz biorąc, system operacyjny i jego konfiguracja, wszystkie koegzystujące w nim aplikacje, sterowniki, pliki, ustawienia w sposób pośredni lub bezpośredni wpływają na program i jego zacho-

wanie. Dobrym przykładem są wtyczki do przeglądarki, mogące zakłócić poprawne wyświetlanie lub funkcjonowanie strony. Również parametry sieci, z którą aplikacja jest połączona, czasami mają istotny wpływ na poprawne działanie programu. Ważnymi elementami są też sprzęt oraz infrastruktura. Zbyt mało miejsca na dysku lub niewielka ilość pamięci RAM na maszynie lokalnej bądź serwerze aplikacyjnym mogą prowadzić do przekłamania wyników testów.

Wszystkie powyższe elementy składają się na środowisko, dlatego też należy je rozważyć podczas testowania. Testerzy muszą zadbać o to, by aplikacja przed wydaniem została wypróbowana w warunkach zgodnych z wymaganiami odbiorcy. Zasymulowanie, stworzenie wszystkich rodzajów dostępnych środowisk jest niemożliwe. Dlatego też bierzemy pod uwagę głównie te, z których będą korzystać użytkownicy. Przykładem powyższej sytuacji może być dobór przeglądarek, na których będziemy testować daną aplikację. Dokonując selekcji, opieramy się zwykle na umowie zawartej z klientem lub na ocenie popularności danych rozwiązań wśród docelowych użytkowników. Cytując Jamesa A. Whittakera: „Środowisko jest rzeczą kluczową, a jednocześnie piekielnie trudną do przetestowania" (*„The environment is crucial, and it is devilishly difficult to test"* [9]).

Wymagania środowiska w testowaniu eksploracyjnym nie różnią się od tych wykorzystywanych w innych podejściach, ale eksploracja ma szersze zastosowanie w dynamicznie zmieniającym się środowisku. Wynika to z faktu, że w takich warunkach należałoby często aktualizować dokumentację dotyczącą scenariuszy oraz przypadków testowych, których nie wytwarzamy podczas testowania eksploracyjnego.

Dokumentacja w testach eksploracyjnych

Dokumentację testową można podzielić na dwa rodzaje: tworzoną przed testami oraz opracowywaną podczas i po zakończeniu testów. W przypadku podejścia eksploracyjnego zazwyczaj albo dostępna jest bardzo mała ilość dokumentacji napisanej przed rozpoczęciem testów, albo nie ma jej w ogóle. Natomiast jeśli chodzi o dokumentację wytwarzaną podczas testów, jak już wcześniej wspomnieliśmy, bardzo użyteczne na własne potrzeby są notatki sporządzane w trakcie sesji testowej. W szybki sposób możemy zrobić je ręcznie lub za pomocą odpowiednich programów, takich jak np. Rapid Reporter. Po zakończeniu całego procesu warto przygotować raport zawierający statystyki incydentów uwzględniające ich krytyczność i ryzyko, który zostanie przedstawiony przełożonym i klientom.

Zalety i wady podejścia eksploracyjnego

Testy eksploracyjne mają dużo zalet, wśród których w pierwszej kolejności możemy wymienić efektywność oraz szybkość uzyskiwania wyników. Dzięki temu podejściu jesteśmy w stanie w krótkim czasie znaleźć błędy o wysokiej krytyczności. Testy eksploracyjne często można przeprowadzić dużo szybciej od systematycznych. Stosowana jest np. eksploracja w sesjach, które zależnie od ilości czasu, jaki możemy poświęcić, trwają od 60 do 90 min, chociaż bywają wydłużane nawet do 120 min. Ponadto sesje pozwalają zmniejszyć koszty (np. dotyczące usuwania błędów) poprzez wczesne wykrywanie i naprawia-

Przypadek 2. Zalety i wady podejścia eksploracyjnego

Firma, w której pracują Helga i Alojzy – kierownicy zespołów testerów – podpisała kontrakt na oprogramowanie dla biura podróży. Nasi bohaterowie spotkali się, aby wspólnie przejrzeć dokumentację i wybrać najlepsze podejście do realizacji testów.

Helga nalegała, by testować eksploracyjnie. Na testy przewidziano relatywnie niewiele czasu, a zespół ostatnio opuściło kilka osób mocnych w automatyzacji. Młoda ekipa mogłaby nie poradzić sobie z takim zadaniem. Poza tym Helga ceniła podejście polegające na wcielaniu się w rolę klienta i użytkownika.

Z kolei Alojzemu bardzo zależało na testach automatycznych. W jego zespole było kilka technicznych osób, m.in. programiści, którzy przekwalifikowali się na testerów. Ponadto zawsze uważał automatyzację za skuteczniejsze podejście, pozwalające wyszukać więcej usterek.

Ostatecznie Helga i Alojzy wypracowali kompromis. Postanowili, że jeden z zespołów będzie eksplorować, a drugi – równolegle – automatyzować.

– Alojzy, załóżmy się. Jeśli po dwóch tygodniach moje podejście okaże się skuteczniejsze, będziesz przez miesiąc codziennie kupował mi czekoladę. Dużą. Z orzechami – zaproponowała Helga.
– OK, ale jeżeli to automatyzacja okaże się najlepszym rozwiązaniem, stawiasz mi skrzynkę piwa. Drogiego.
– Umowa stoi.

Helga i Alojzy wrócili do swoich zespołów. Przekazali wytyczne podwładnym i wkrótce rozpoczęły się testy.

W pierwszym tygodniu zespół Helgi wyszedł na prowadzenie. Testerzy zidentyfikowali sporo awarii, których nie dałoby się wywołać podczas automatyzacji. Niestety, nikt nie dokumentował okoliczności wystąpienia usterek, więc po pewnym czasie mało kto o nich pamiętał.

Ekipa Alojzego początkowo miała problemy z nowym narzędziem do automatyzacji, ale dość szybko opanowała jego tajniki. Testy, które wykonała, były dokładne, rzetelne i świetnie udokumentowane.

Gdy minęły dwa tygodnie, Helga i Alojzy ponownie spotkali się, aby porównać skuteczność wybranych podejść. Okazało się, że zespół eksploracyjny przetestował sporo niezbyt oczywistych scenariuszy, które specjalistom ds. automatyzacji nie przyszłyby do głowy. Z kolei druga ekipa wykonała swoje zadanie zgodnie z założeniem i udokumentowała przebieg testów.

Z Helgi i Alojzego zeszło powietrze. Każde z nich jeszcze niedawno było przekonane, że wygrało. Tymczasem okazało się, że w zasadzie zakład zakończył się remisem, a podejścia, na które się zdecydowali, najlepiej byłoby połączyć. Przyszła więc pora na wykorzystanie zalet obu rodzajów testów.

W kolejnych tygodniach zespoły dzieliły się ze sobą doświadczeniem z dotychczasowych testów. Eksploratorzy przekazywali ekipie zajmującej się automatyzacją uwagi dotyczące niestandardowego zachowania systemu w określonych warunkach i scenariuszy niezbyt dokładnie opisanych w wymaganiach/specyfikacji, a od technicznych nauczyli się, że warto sporządzać choćby notatki (niestety, niektórych z awarii, na jakie natrafiono w pierwszych tygodniach, nie dało się już odtworzyć).

Ostatecznie projekt zakończył się sukcesem, a Helga i Alojzy wiedzieli już, że ich ulubione metody testowe mają wiele zalet, ale nie są pozbawione wad. Często najlepszym rozwiązaniem okazuje się połączenie obu podejść.

nie krytycznych defektów. Sprawdzają się idealnie w środowisku internetowym. Zaletą tego rodzaju technik jest również to, że eksploracja obszarów, które nie są pokryte scenariuszami bądź przypadkami testowymi, pozwala na zwiększenie pokrycia testowego.

Do wad podejścia eksploracyjnego możemy zaliczyć ryzyko błędnej interpretacji testów (np. przeprowadzenie tylko pozytywnych scenariuszy). Jeśli podczas eksplorowania nie sporządzamy nawet notatek, ciężko będzie udowodnić klientowi wysiłek włożony w przeprowadzenie testów oraz udowodnić, że w ogóle zostały wykonane. Poważną wadą omawianego podejścia jest też problem z reprodukowaniem znalezionych defektów, ponieważ nie dysponujemy konkretnym scenariuszem, według którego badamy dane obszary aplikacji, więc gdy znajdziemy usterkę, powtórzenie tych samych operacji w odpowiedniej kolejności może okazać się kłopotliwe.

Testujemy eksploracyjnie – i co teraz?

W poprzednich rozdziałach wyjaśniliśmy, na czym polegają testy eksploracyjne oraz pokazaliśmy ich zastosowanie. Przyjrzyjmy się teraz bardziej szczegółowo możliwościom, które daje ta technika. Rozważmy również ograniczenia związane z testowaniem eksploracyjnym i zastanówmy się, w jakich sytuacjach taka metoda się nie sprawdzi.

Jakie możliwości daje testowanie eksploracyjne?

Przypomnijmy, czym są testy eksploracyjne. Jak zauważył James Bach, testowanie eksploracyjne to potężna technika, choć często nierozumiana. W niektórych sytuacjach może się okazać o wiele produktywniejsza od testowania za pomocą technik skryptowych [1]. Z kolei według Jamesa A. Whittakera eksploracja to testowanie bez zdefiniowanego wcześniej planu testów [7].

Jakie zatem możliwości daje nam ten rodzaj testowania? Przede wszystkim oszczędzamy czas i pieniądze, jakie musielibyśmy przeznaczyć na przygotowanie całej dokumentacji testowej oraz skryptów w technikach skryptowych i testach automatycznych. W naszym przypadku tester od razu przystępuje do pracy, bez tworzenia formalnego planu i pisania przypadków testowych, co pozwala na szybszą ocenę jakości.

Ten rodzaj testowania świetnie sprawdza się w przypadku testowania nowych, jeszcze niepoznanych aplikacji, często w środowisku internetowym, gdy tester eksploruje program i odkrywa jego działanie oraz funkcje. Zdarza się bowiem, że czytanie wymagań lub specyfikacji – o ile istnieją – nie pozwala uzyskać całościowego obrazu zachowania systemu. Również dokumentacja użytkownika zwykle nie zawiera opisu funkcjonowania aplikacji w określonych sytuacjach.

Ponadto metoda eksploracyjna często pozwala na zlokalizowanie luk w strategii testowania. Znalezione błędy mogą sygnalizować istnienie nieopracowanych przypadków testowych. Najcenniejszą możli-

wością, jaką dają nam testy eksploracyjne, jest zatem poszerzenie przypadków testowych o elementy, które nie zostały przewidziane nawet w udokumentowanych testach. Zmienia to znacząco podejście do ich przeprowadzania.

Jak widać, testowanie eksploracyjne okazuje się efektywne, gdy trzeba szybko uzyskać informację na temat jakości aplikacji, a czasu na przygotowanie testów w pełni udokumentowanych jest niewiele. Ponadto eksploracja okazuje się przydatna w sytuacji, gdy tester chce sprawdzić wyniki pracy innego testera w szybki i niezależny sposób.

A co z ograniczeniami? Kiedy unikać testów eksploracyjnych?

Jak każda metoda testowania, również eksploracja niesie ze sobą ograniczenia. Jednym z nich są wysokie wymagania wobec testera – taka osoba powinna posiadać odpowiednie doświadczenie i umiejętności; inne pożądane cechy to kreatywność i intuicja. Niedoświadczony tester może pominąć niektóre obszary do testowania, a zamiast tego skupiać się zbyt długo na mniej istotnych aspektach. Wiąże się to z kolejnym ryzykiem – nieznalezienia najważniejszych błędów w aplikacji.

Istotnym ograniczeniem testów eksploracyjnych jest brak możliwości generowania pokrycia. Trudno udowodnić, że aplikacja spełnia wymagania klienta, jeśli nie mamy dokumentacji, do której można by się odnieść, a tester tak naprawdę tylko się domyśla, na czym powinno polegać poprawne działanie systemu. Brak zdefiniowanych przypadków testowych sprawia, że powtórzenie testów oraz określenie, czy zostały właściwie wykonane, sprawia więcej trudności niż przy zastosowaniu innych podejść do testowania.

Kolejnym problemem jest brak metryk. Zwykle kierownika testów lub project managera interesują informacje na temat postępu testów w projekcie (takie jak procentowe pokrycie testami funkcjonalności czy liczba wykonanych przypadków testowych), wyrażone choćby cyfrowo. Testowanie eksploracyjne często nie pozwala na generowanie tego typu wyników, choć niektóre jego warianty rozwiązują tę kwestię. Przykładem jest testowanie w sesjach

GŁÓWNE ZASADY TESTOWANIA EKSPLORACYJNEGO

- Testy eksploracyjne pozwalają na poznawanie aplikacji i odkrywanie jej funkcjonalności
- Jakość wykonanej pracy zależy głównie od umiejętności i doświadczenia testera
- Znikoma dokumentacja testowa lub wręcz jej brak
- Eksploracja doskonale sprawdza się w metodykach agile'owych
- Brak wyroczni testowej i w efekcie brak pokrycia przypadków testowych
- Odpowiedzią na brak metryk jest testowanie w sesjach (więcej na ten temat w dalszej części rozdziału)
- Metoda sprawdza się podczas testowania nowych, jeszcze niepoznanych aplikacji
- Eksploracja to idealne rozwiązanie, gdy w krótkim czasie potrzebujemy informacji o jakości
- Z uwagi na opisane ograniczenia podejście warto wesprzeć innymi technikami testowania

oparte na Session-Based Management.

Brak metryk może się wiązać z istotnymi nadużyciami ze strony zarówno wykonawcy, jak i zamawiającego. Dostawca np. przedstawi zbyt wysokie wyceny i dostarczy system niespełniający potrzeb klienta. Ten ostatni z kolei może przyjąć postawę roszczeniową i często zmieniać wymagania.

W jakich zatem sytuacjach powinniśmy unikać eksploracji?

- Gdy przeprowadzamy testy akceptacyjne, które powinny być w pełni udokumentowane.
- Gdy wymagane jest udokumentowanie wszystkich scenariuszy testowych użytych w testach i pokrycia testowego.

Kilka słów o sesji testowej

Sesja testowa jest rozwiązaniem, które pozwala radzić sobie z wyżej wymienionymi ograniczeniami testowania eksploracyjnego. Pojęcie sesji odnosi się do metody zarządzania testami eksploracyjnymi i mierzenia ich wyników – Session-Based Test Management, której autorami są James i Jonathan Bachowie. Podejście SBTM jest odpowiedzią na oczekiwania dotyczące sposobu wykonywania pracy i raportowania jej rezultatów. Główne elementy składowe tej metody zarządzania testami eksploracyjnymi przedstawiono w ramce.

Powszechnie wykorzystywanym narzędziem służącym do rejestrowania przebiegu sesji testowej i zapisywania notatek jest łatwy w użyciu, darmowy Rapid Reporter. Podczas dokumentowania wyników tester może przechodzić między elementami:

- Środowisko (Setup);
- Status (Charter);
- Test;
- Notatki;
- Defekt;
- Kontrola (Check);
- Pytania.

Rysunek 3. Raport sesji testowej zapisany w formacie HTML przy użyciu narzędzia Rapid Reporter

Session Report | Powered by Rapid Reporter

Time	Reporter	Type	Content	Screenshot	RTF Note
22/04/2015 13:56:43	Dominika Wojtko	**(Rapid Reporter version)**	1.12.12.28		
22/04/2015 13:56:43	**Dominika Wojtko**	**Session Reporter**	**Dominika Wojtko**		
22/04/2015 13:56:43	**Dominika Wojtko**	**Session Charter**	**iPad Sanity Tests - Patient records**		
22/04/2015 14:00:15	Dominika Wojtko	**Setup**	iPad 3 iOS7		
22/04/2015 14:08:25	Dominika Wojtko	**Test**	Marking patient as favourite		
22/04/2015 14:09:39	Dominika Wojtko	**Test**	Add patient to briefcase		
22/04/2015 14:11:26	*Dominika Wojtko*	***autogenerated***	*screenshot saved*		
22/04/2015 14:13:36	Dominika Wojtko	**Check**	Pass		
22/04/2015 14:15:42	*Dominika Wojtko*	***autogenerated***	*extended note saved*		*1_20150422_141541.rtf*
22/04/2015 14:15:47	Dominika Wojtko	**Bug**	The patient cannot be removed from filter lists		1_20150422_141541.rtf
22/04/2015 14:16:27	**Dominika Wojtko**	**Session End. Duration**	**00:19:44**		

SESSION-BASED TEST MANAGEMENT

Misja
Misja definiuje nasze cele, czyli mówi o tym, co testujemy i z jakimi problemami mamy do czynienia podczas testów eksploracyjnych.

Statut (test charter)
Statuty – cele sesji testowej i jej agenda – są formułowane przez zespół testerów na początku testowania.

Sesja (test session)
Sesja opiera się za zdefiniowanym statucie i trwa od 60 do 120 minut. Podczas jej trwania testerzy często odkrywają nowe możliwości testowania oraz tworzą przypadki testowe.

Podsumowanie sesji (session report)
Każda sesja powinna zostać prawidłowo zaraportowana. Na tym etapie warto wykorzystywać notatki i odpowiednie narzędzia, takie jak np. Rapid Reporter.

Na podsumowanie sesji składają się:

- imię i nazwisko testera;
- statut;
- lista znalezionych defektów;
- lista otwartych pytań, problemów;
- czas rozpoczęcia i zakończenia testów.

Spotkanie z kierownikiem testów (debrief)
Sesja kończy się krótkim spotkaniem lidera z testerem, które ma na celu omówienie raportu z testów oraz uzyskanie odpowiedzi na pytania:

- Co wydarzyło się podczas sesji?
- Co osiągnęliśmy?
- Jakie napotkaliśmy przeszkody?
- Co należałoby poprawić?
- Co tester sądzi na temat przeprowadzonej sesji?

Dodatkowo podczas testowania możemy zapisać zrzut ekranu i dołączyć go do jednego z wymienionych wcześnie elementów. Rapid Reporter rejestruje także czas sesji, łącznie z jej rozpoczęciem i zakończeniem. Cały proces kończy się wygenerowaniem raportu w pliku CSV, który możemy zapisać w bardziej czytelnym formacie HTML (rysunek 3).

Realizacja testów eksploracyjnych

Opis zagadnienia

Przedmiot testów

Przetestowano darmowy program GiosPSM, służący do dzielenia i łączenia plików PDF. Plik o rozszerzeniu .exe (Version_2009.07.01.0_GiosPSM.exe) nie wymagał specjalnego przygotowania środowiska.

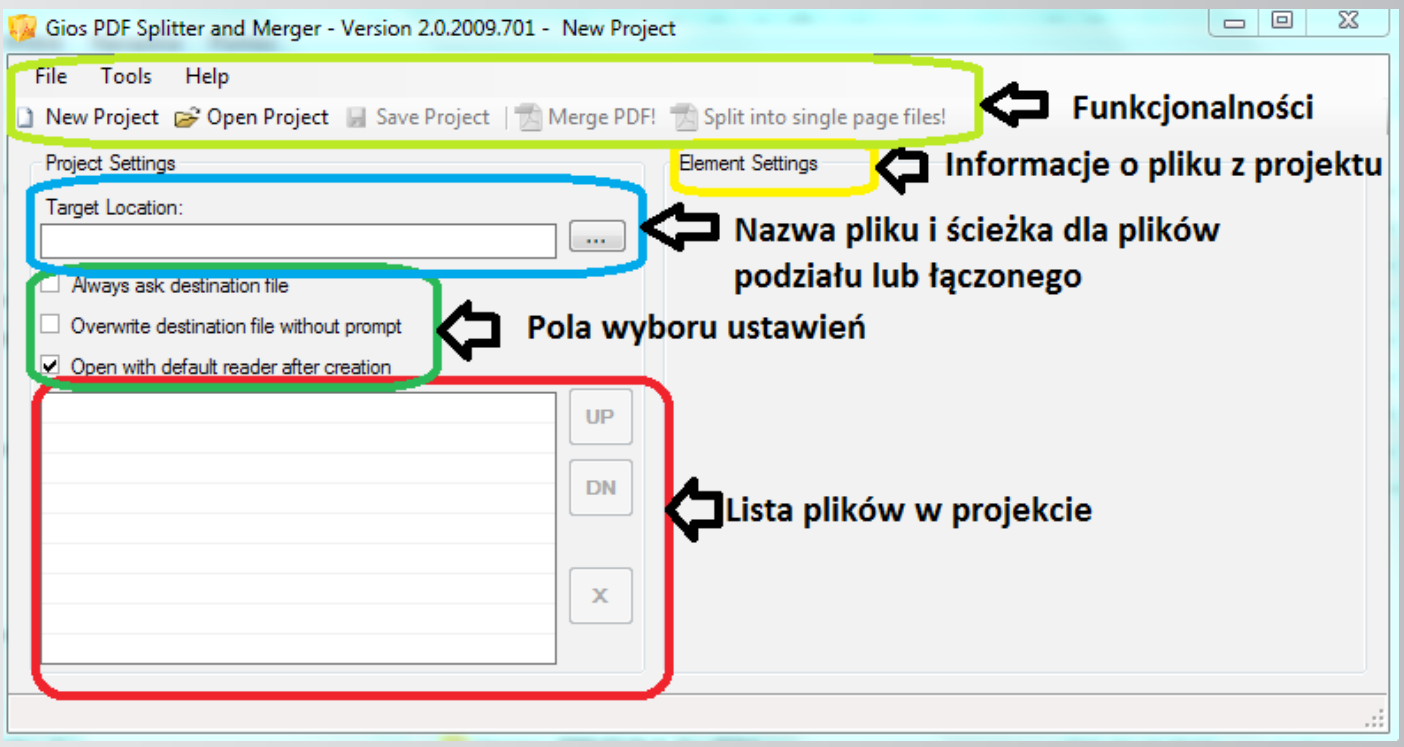
Uczestnicy testów
Oprogramowanie zostało przetestowane przez dwóch niezależnych testerów.

Czas trwania testów
Testy odbyły się w 6 sesjach testowych (po 3 na testera), każda z nich trwała średnio 3 godziny. Przygotowanie do testów, zapoznanie z formatem PDF, wybranie i przygotowanie plików wejściowych, zainstalowanie i poznanie funkcjonalności nowych narzędzi zajęło ok. 8 godzin.

Środowisko testowe
Testy zostały przeprowadzone na następujących środowiskach:

- Windows 7 Professional (64-bitowy system operacyjny), procesor AMD Phenom II X4 955 (3.20 GHz), pamięć RAM: 4 GB (komputer stacjonarny);
- Windows XP Home Edition (32-bitowy system operacyjny), procesor Intel Premium M (1.60 GHz), pamięć RAM: 512 MB (laptop MSI M510C);
- Windows 7 Home Premium (32-bitowy

Rysunek 4. Okno aplikacji GiosPSM



system operacyjny), procesor i3 (M350 2.27 GHz), pamięć RAM: 3 GB (laptop HP ProBook 4520s);
- Windows 8.1 Home Edition (64-bitowy system operacyjny), procesor Intel i3, pamięć RAM: 4 GB (laptop Lenovo B5400).

Użyte narzędzia

W celu wykonania testów i zaraportowania ich wyników testerzy posłużyli się poniższymi narzędziami:

- Microsoft Office Excel 2007 – tworzenie logów testów i opracowywanie raportu z testów;
- Greenshoot – wykonywanie zrzutów ekranu;
- Funkcja systemu Windows umożliwiająca wykonywanie zrzutów ekranu za pomocą klawisza PrtScn (lub odpowiednika; nazwa klawisza różni się w zależności od rodzaju klawiatury) – wykonywanie zrzutów ekranu;
- Paint – zapisywanie zrzutów ekranu;
- Notepad++ – notatnik do zapisywania notatek i kopiowania danych;
- DiffPDF – porównywanie plików PDF;
- SpeedFan – monitorowanie temperatury i pracy procesora, komputera (płyty głównej) i dysku twardego.

Przebieg procesu testowego

Program GiosPSM, Version_2009.07.01.0_GiosPSM.exe, został poddany 60 testom. Sprawdzono, czy aplikacja się uruchamia i czy można utworzyć nowy projekt. Sprawdzono funkcjonalności takie jak: dodawanie pliku PDF (za pomocą funkcji Add PDF File i poprzez przeciąganie) i dodawanie pliku JPG (za pomocą funkcji Add Image File oraz poprzez przeciąganie). Przetestowano także zapisywanie projektów przy użyciu opcji Save Project oraz Save Project As, otwieranie nowych projektów (Open Project), a także działanie funkcji dzielenia i łączenia plików PDF – Split into single page! i Merge PDF.

Pierwszy z testerów – Piotr Krajewski – rozpoczął testy od ogólnego i systematycznego zapoznawania się z aplikacją. Sprawdzono ogólne działanie nowego projektu, jego zapis i odczyt. Przetestowano różnorodne ustawienia programu. Finalnie zweryfikowano podstawową funkcjonalność programu, czyli dzielenie i łączenie różnych wersji plików PDF i JPG.

Testy drugiego testera – Adriana Brodnego – polegały na sprawdzeniu użyteczności aplikacji oraz zweryfikowaniu, w jaki sposób zachowa się oprogramowanie, gdy dojdzie do awarii. Przetestowano m.in. działanie funkcji dodawania i usuwania załączanych plików PDF i JPG oraz dzielenie i łączenie plików PDF w różnych konfiguracjach (np. obsługa plików PDF zawierających w tytule tylko znaki specjalne, plików uszkodzonych i zabezpieczonych hasłem, pustych plików PDF, plików o rozmiarze powyżej 400 MB).

Po pierwszej sesji testowej zaobserwowano brak skutecznej metody weryfikującej poprawność działania programu pod kątem dzielenia i scalania plików PDF. Wzrokowe porównanie dokumentów nie było obiektywną metodą, więc zdecydowano się na użycie programu DiffPDF. Aplikacja automatycznie analizowała dane wejścia i wyjścia dla plików PDF pod kątem objętości, poprawności, formatu etc. Porównywanie dokumentów w programie DiffPDF trwało co najmniej kilkadziesiąt minut i bardzo obciążało procesor oraz dysk komputera, jednak narzędzie okazało się bar-

Tabela 1. Przykład logów testowanej aplikacji

Konfiguracja	Notatka	Test	Sprawdzenie	Incydent	Pytanie
Uruchomiony program: *Version_ 2009.07.01 .0_ GiosPSM.exe.*	Sprawdzenie funkcjonalności zapisu projektu/plików do zadań.	Dodanie plików do listy zadań: *Zapisanie projektu.*	Czy coś się zmieniło w wyglądzie po zapisaniu pliku? Na górnej belce brak informacji o nazwie projektu.	Na górnej belce brak informacji o nazwie projektu. Brak możliwości nadpisywania po zmianach w projekcie – przycisk *Save project* jest nieaktywny. Screenshot: *Zapis_projektu_pod_nazwa_ Merged_files_*	Co się stanie, gdy otworzymy projekt/program i projekt? Odpowiedź: nazwa pojawi się na górnej belce.
Uruchomiony program: *Version_ 2009.07.01 .0_ GiosPSM.exe.*	*Sprawdzanie funkcji Split i Open with default reader after creation.*	Podział pliku: *„pliki_wejsciowe\jewishtimes44.pdf”.*	Podział utworzony w folderze: Output\ JewishTime_ podziel_wszystko. Czy uruchomiła się domyślna przeglądarka i otworzyła utworzone pliki?	Przy funkcji podziału *Split* nie działa check--box *Open with default reader after creation.*	Czy funkcja/ checkbox: *Open with default reader after creation* ma być dostępna dla fukcji *Split*? Jeśli nie, program powinien wyświetlać odpowiednią informację, np. *Only for merged.*

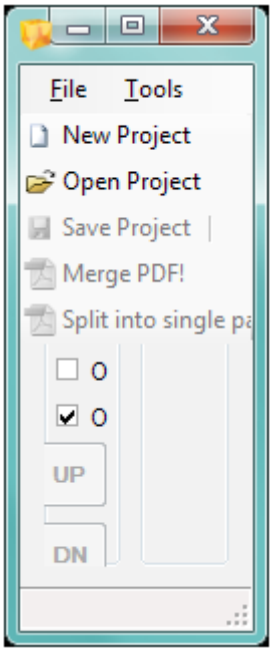
dzo pomocne podczas oceny testowanej aplikacji.

Przebiegi testów zostały odnotowane w logach, które umieszczono w raportach z testów (tabela 1).

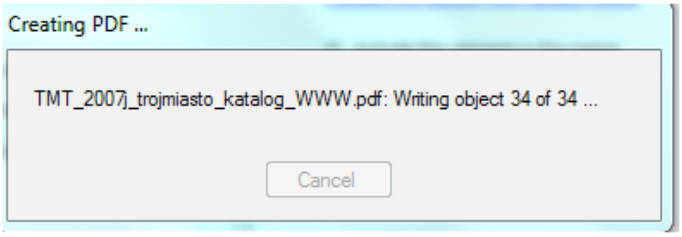
Podsumowanie testów

Zgłoszono łącznie 20 incydentów: 5 o wysokim priorytecie, 7 o średnim i 8 o niskim. Oto opisy przykładowych incydentów wraz ze zrzutami ekranów.

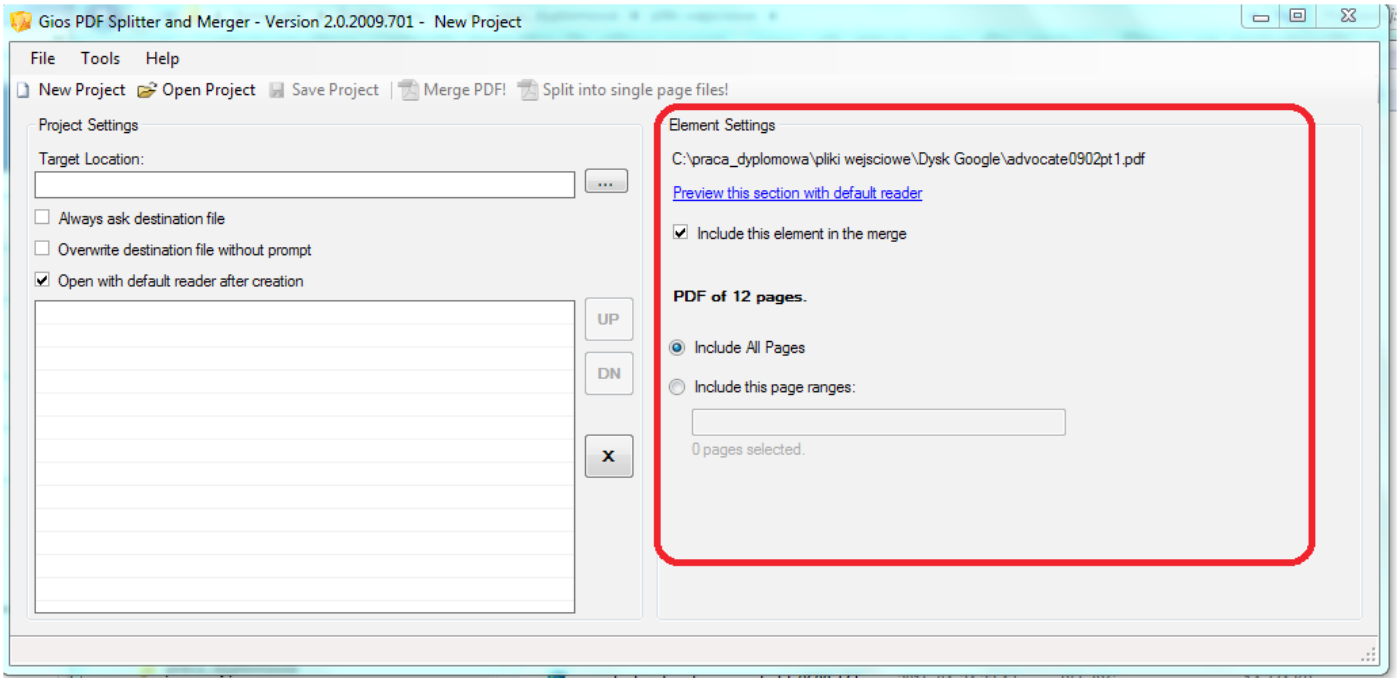
Przypadek 1. – brak skalowania programu – aplikacja nie udostępnia funkcji blokowania rozmiaru podczas zmniejszania okna. Przyciski, menu, ścieżka do pliku wyjściowego są ukrywane.



Przypadek 2. – brak możliwości przerwania działania programu w trakcie dzielenia pliku PDF, mimo że pojawia się okno informacyjne z przyciskiem *Cancel*. Po próbie anulowania przycisk staje się nieaktywny, a aplikacja nadal dzieli plik.



Przypadek 3. – po otwarciu nowego projektu program nie przywraca ustawień domyślnych w sekcji *Element Settings*. Aplikacja „zapamiętuje" konfigurację z wcześniejszego projektu.



Problemy napotkane w czasie testów

W czasie testów pierwszą przeszkodą, jaką napotkali testerzy, był brak informacji na temat prawidłowego działania programu. Przycisk *Help* kierował na stronę główną twórcy aplikacji: http://www.paologios.com/?help=gios.psm, na której nie znaleziono instrukcji, a ostatnią aktualizację zamieszczono w 2009 r.

Program został uznany za mało intuicyjny. Aplikacja udostępnia dwie funkcjonalności: *Merged PDF* oraz *Split into single page*, jednak ustawienia: *Always ask destination files, Overwrite destination file without prompt* i *Open with default reader after creation* działały tylko podczas łączenia plików PDF. W trakcie dzielenia, nawet jeśli wybrano którąś z powyższych opcji, program ignorował to ustawienie. Trudno było jednoznacznie stwierdzić, czy takie działanie aplikacji było nieprawidłowe. Podczas testów zaraportowano je jako incydent o niskim priorytecie.

Kolejnym ograniczeniem był fakt, że testowanie eksploracyjne w dużej mierze zale-

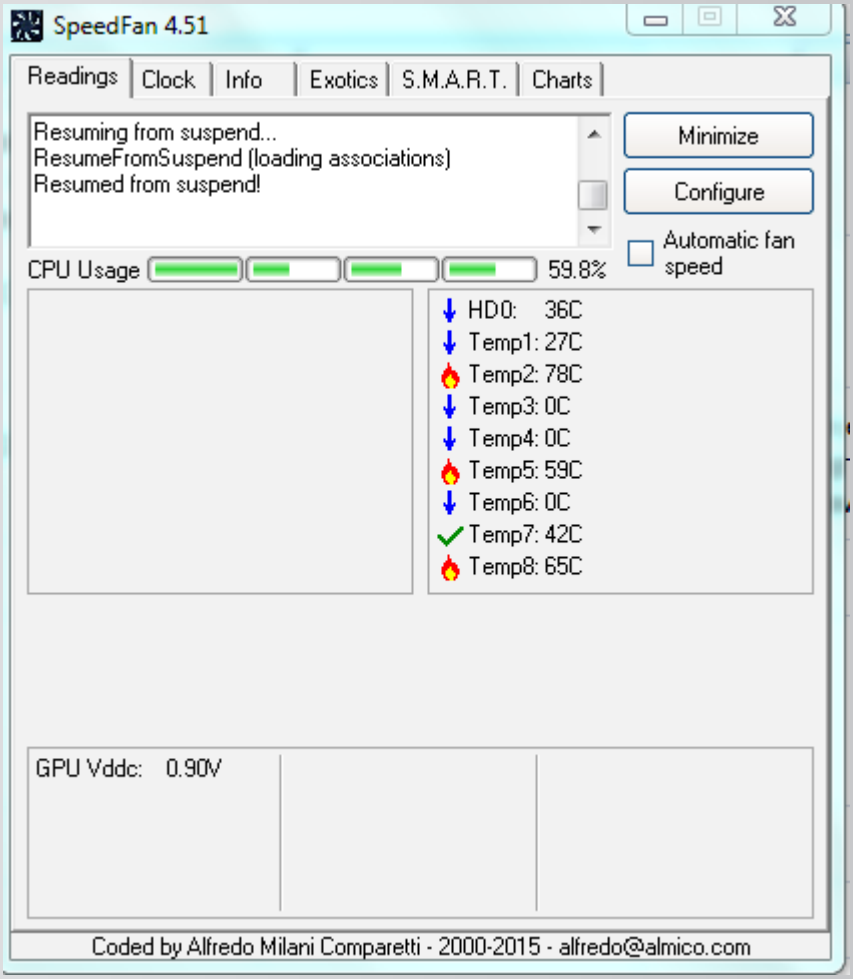
ży od wiedzy i umiejętności testerów. W trakcie pracy nad aplikacją GiosPSM testujący uzupełniali braki w wiedzy na bieżąco, pomiędzy kolejnymi sesjami. Zapoznawali się m.in. ze standardami plików PDF, JPG, metodami zabezpieczeń plików i ich typami.

Samo testowanie nie okazało się szczególnie trudnym zadaniem. Testerzy dość szybko tworzyli nowe testy, polegając na swojej intuicji, kreatywności i stosując tzw. zgadywanie błędów. Testowanie zostało jednak uznane za czasochłonne, zwłaszcza w przypadku plików o dużej objętości, np. kilkudziesięciu MB, tj. ponad 1000 stron (największy plik, liczący 11,5 MB, składał się 2139 stron). W takich sytuacjach na wynik trzeba było czekać od kilkudziesięciu minut do kilku godzin (np. w przypadku pliku 405 MB – 4 godziny).

Teoria a praktyka

Testy eksploracyjne sprawdzają się w sytuacjach, gdy informacje na temat programu są niekompletne lub nie ma ich wcale. W trakcie takich testów przydatne są także wiedza i doświadczenie testera, ponieważ często rekompensują one braki w

Rysunek 5. Zrzut ekranu programu SpeedFan, monitorującego temperaturę procesora i dysku podczas porównywania plików PDF za pomocą programu DiffPDF



dokumentacji. Eksploracja jest skuteczna, gdy nie wiadomo, w jaki sposób zaprojektować kolejny test i jaki on powinien być, a także gdy tester zamierza wykroczyć poza dotychczasowy schemat.

Jak wspomniano w poprzednim rozdziale, testowaniu eksploracyjnemu towarzyszą pewne ograniczenia, z którymi można sobie poradzić za pomocą sesji testowej. Ta metoda zarządzania testami eksploracyjnymi składa się z kilku elementów: misji, statutu, sesji, podsumowania sesji i spotkania z kierownikiem testów. Z perspektywy organizacji uporządkowany sposób przebiegu testów eksploracyjnych pozwala na skuteczne zarządzanie i przedstawienie klientowi informacji na temat jakości testowanego produktu.

Testy eksploracyjne opisywane w niniejszym rozdziale także przebiegały w określonym porządku. Przede wszystkim wybrano przedmiot testów. Następnie zdefiniowano, ile czasu zostanie przeznaczone na testy; było to trudne ze względu na niewielkie doświadczenie testerów w zakresie testowania aplikacji desktopowych. Zgodnie z założeniami eksploracji w trakcie trwania sesji odkrywano nowe możliwości testowania i tworzono przypadki testowe. Niezwykle istotną częścią testów jest odpowiednie raportowanie ich przebiegu. Testerzy, którzy badali GiosPMS, do dokumentowania postępów prac wykorzystywali stosunkowo proste i powszechnie dostępne narzędzie – arkusz kalkulacyjny. Podczas trwania każdej sesji na bieżąco logowali swoje aktywności i obserwacje. Jako podsumowanie czynności testowych stworzyli raporty z listą incydentów, logami (w tym pytaniami i problemami) oraz ogólnym podsumowaniem testów, w którym wymienili informacje dotyczące tego, kto testował, obiektu testowego, środowiska testowego, narzędzi, plików referencyjnych, podstawowych założeń.

Ujednolicenie nazewnictwa i formatowania danych podczas tworzenia logów testów i pisania raportów okazało się istotne nawet dla tak małego zespołu, jakim jest grupa pracująca nad pracą dyplomową. Precyzyjne, zrozumiałe raportowanie przebiegu testów umożliwiło późniejsze opracowywanie i odpowiednie przedstawienie wyników, podobnie jak w przypadku większych organizacji, w których raporty z testów eksploracyjnych muszą być czytelne i dostarczać szybkiej informacji o jakości testowanego produktu.

Wnioski z testów

Testy eksploracyjne produktu GiosPSM pozwoliły w krótkim czasie zrozumieć logikę oprogramowania, co skutkowało lepszym poznaniem aplikacji i odkryciem jej funkcjonalności. Testerzy nie byli ograniczeni obszerną dokumentacją testową, dzięki czemu skupili się na eksperymentowaniu, poszukiwaniu mniej oczywistych usterek i wywoływaniu awarii. Jakość wykonywanej pracy poprawiała się podczas każdej kolejnej sesji wraz ze wzrostem wiedzy i doświadczenia testerów. W przypadku testowanej aplikacji eksploracja została wsparta użyciem dodatkowych narzędzi, które zobiektywizowały i zautomatyzowały proces.

Testy eksploracyjne pozwoliły odpowiedzieć na pytanie o jakość i użyteczność testowanego programu. Aplikacja spełniała swoje podstawowe funkcje (podział i łączenie plików PDF oraz JPG). Znaleziono w

niej jednak wiele nieobsługiwanych wyjątków i braków walidacji ustawień oraz braki w kompozycji. Stwierdzenie powiązań pomiędzy opcjami wyboru a przypisanymi do nich funkcjonalnościami byłoby łatwiejsze, gdyby twórca programu zastosował np. tooltipy.

Uwagi i znalezione incydenty pomogłyby autorowi GiosPMS udoskonalić aplikację. Mogłyby również zostać wykorzystane podczas tworzenia nowego, podobnego oprogramowania.

Podsumowanie

Z powodu rosnącej popularności automatyzacji można by się zastanawiać, czy eksploracja wkrótce nie wyjdzie z użycia. W tym artykule przekonaliśmy – taką mamy nadzieję – Czytelników, że nie wszystko da się przetestować automatycznie. Niekiedy podejście opierające się w dużej mierze na intuicji, w zasadzie pozbawione sztywnych procedur i metryk, okazuje się konieczne. Według internetowego Słownika Języka Polskiego PWN eksploracja to *„badanie, poszukiwanie, odkrywanie czegoś”* [4]. W związku z tym testy eksploracyjne polegają na stosowaniu nieoczywistych rozwiązań i zapoznawaniu się z nieznanym „terenem”, czyli oprogramowaniem. Podejście to okazuje się szczególnie użyteczne w początkowych fazach projektu, gdy dokumentacja jest uboga (lub w ogóle jej nie ma). Eksploracja stanowi także dobry wstęp do automatyzacji – testerzy muszą poznać funkcje tworzonej aplikacji, zanim będą mogli przystąpić do opracowywania półprogramistycznych testów.

W przypadku programu GiosPMS omawiane podejście zdało egzamin. Aplikacja jest niewielka i niezbyt skomplikowana. Projektowanie testów automatycznych dla takiego oprogramowania można by wręcz uznać za przesadę. Paradoksalnie podejście automatyczne, w założeniu pozwalające – poprzez możliwość wielokrotnego używania – zaoszczędzić wiele godzin, okazałoby się w tym przypadku marnowaniem cennego czasu.

Podsumowując, nie wydaje się, by w najbliższym czasie pojawił się godny następca eksploracji. Pomysłowość człowieka nie zna granic, jego skłonność do popełniania błędów – również. Na razie jedynie podejście eksploracyjne umożliwia odkrywanie nieoczywistych usterek i wczuwanie się w rolę przyszłego użytkownika, przyjmowanie jego subiektywnej perspektywy. Żaden program tego nie potrafi.

REFERENCJE

1. Bach James, *Exploratory Testing Explained*, https://www.satisfice.com/articles/et-article.pdf
2. Hendrickson Elisabeth, *Explore It! Reduce Risk and Increase Confidence with Exploratory Testing*, The Pragmatic Programmers, e-book: wersja P2.0 (wrzesień 2013)
3. http://www.a-sisyphean-task.com/2013_01_01_archive.html
4. http://sjp.pwn.pl/szukaj/eksploracja.html
5. http://wrotqa.org/wp-content/uploads/2014/08/Exploratory-testing-WrotQA.pdf
6. http://www.satisfice.com/articles/where_fits.shtml
7. https://www.satisfice.com/articles/et-article.pdf
8. https://www.youtube.com/watch?v=8Y4WCdJRfV4
9. Whittaker James A., *Exploratory Software Testing: Tips, Tricks, Tools and Techniques to Guide Test Design*, Addison-Wesley, 2009

PRZYDATNE MATERIAŁY

1. http://www.computerworld.pl/news/311111_2/Testy.w.pospiechu.html
2. http://www.quardev.com/content/whitepapers/ET_dynamics_Google_3_06.pdf
3. https://quorum.akademiq.pl/discussion/1912/czy-testy-eksploracyjne-sa-w-jakis-sposob-lepsze--od-innych/p1
4. http://www.satisfice.com/sbtm/
5. http://www.satisfice.com/articles/what_is_et.shtml
6. http://socjoit.blogspot.com/2013/09/testowanie-nieformalne-testy.html
7. http://testerzy.pl/baza-wiedzy/testy-ekploracyjne-materialy
8. http://www.testowanie.net/testowanie/testy-ad-hoc-eksploracyjne/
9. Kaner, Cem, *Testing Computer Software,* Tab Books, 1988

LINKI DO NIEKTÓRYCH PRZYDATNYCH NARZĘDZI

1. Rapid Reporter: http://testing.gershon.info/reporter/
2. ScreenToGif: https://screentogif.codeplex.com/
3. Selenium IDE: http://www.seleniumhq.org/download/

Zuzanna Gościcka-Miotk

Redaktorka, korektorka. Słuchaczka studiów podyplomowych (Tester oprogramowania). Interesuje ją zarówno inżynieria wymagań – bo ceni dobrą, przemyślaną organizację i wsłuchiwanie się w potrzeby klientów, eksploracja – bo uwielbia wszystko, co nowe i nieodkryte, jak i automatyzacja – bo kocha wyzwania i zdobywanie zupełnie nowych umiejętności.

Od kwietnia certyfikowany tester (poziom podstawowy). W wolnym czasie lubi jeździć i latać tam, gdzie jeszcze jej nie było, grać w planszówki, biegać po korcie tenisowym i obmyślać kolejne challenge.

Adrian Brodny

Certyfikowany tester oprogramowania. W zawodzie testera pracuje od ponad dwóch lat. Jest absolwentem Politechniki Gdańskiej i Polsko-Japońskiej Akademii Technik Komputerowych. Specjalizuje się w testowaniu aplikacji edukacyjnych oraz stron internetowych. Dodatkowo posiada doświadczenie w projektowaniu aplikacji mobilnych.

Interesują go numizmatyka, szachy i podróżowanie.

Patrycja Nowicka

Inżynier mechanik (Eksploatacja Instalacji Przemysłowych), słuchaczka studiów podyplomowych (Tester oprogramowania), od miesiąca Junior Tester w polsko-holenderskiej firmie Goyello, specjalizującej się w dostarczaniu rozwiązań informatycznych tworzonych zgodnie z zasadami metodologii Agile.

Od kwietnia certyfikowany tester (poziom podstawowy).

Fanka ciężkiego brzmienia oraz średniowiecznego fechtunku.

AUTORZY

Piotr Krajewski

Test Development Manager w międzynarodowej firmie kontraktowej produkcji elektroniki. Kieruje działem projektowania urządzeń testujących (hardware & software) do produkcji urządzeń elektronicznych.

Zainteresowania: elektronika i automatyka, zasady działania rynków inwestycyjnych, sport (tenis, kolarstwo górskie), muzyka rockowa.

Dominika Wojtko

Karierę testera rozpoczęła prawie rok temu, ponadto ma ponad 5 lat doświadczenia w branży human resuorces. Podczas pracy w firmie IT (w dziale HR) i organizowania szkoleń dla testerów odkryła, że testowanie to ścieżka leżąca w obszarze jej zainteresowań.

Zamierza rozwijać karierę testerską, a przede wszystkim poszerzyć wiedzę z zakresu automatyzacji testów.

Katarzyna Zatorska-Radlińska

Logopeda, specjalista ds. komunikacji i kreowania wizerunku. Obecnie menedżer, poszukująca nowej ścieżki rozwoju.

Od września 2014 r. słuchaczka studiów podyplomowych na kierunku Tester oprogramowania. Bo na zmiany (szczególnie te dobre) nigdy nie jest za późno!

Prywatnie żona, mama i motocyklistka. Tak, to da się łączyć.

How will „The Future of Testing“ look like?

imbus presents trend study by Dr. Bernd Flessner

The IT industry is developing rapidly – and so is the software testing sector.
What should one brace oneself for in the years ahead? imbus has asked the futurologist and science writer Dr. Bernd Flessner to cast light on the future of software testing.

The trend study “The Future of Testing” is the result.

Dr. Bernd Flessner invites the readers to take a look into the future. On circa 40 pages, scenarios for the time periods “from 2020”, “from 2035” and “from 2050” describe possible developments of the testing sector – each examing both a positive and a negative viewpoint. The study features forecasts on much-discussed technological developments, including industry 4.0, Outernet and smart home.
It explains, how they could directly affect the testing sector – and it derives, what this means for the software tester profession.

“The Future of Testing” features besides the scenarios also the results of an exclusive Delphi survey.
Each five well-known international experts of the IT and the testing industry were asked to assess the effects of specific trends on the software testing industry in the near future.
Dr. Bernd Flessner draws in “The Future of Testing” on the theses from his keynote “Wir Prothesengötter” at the Software-QS-Tag 2013 and the panel discussion at the Software-QS-Tag 2014.

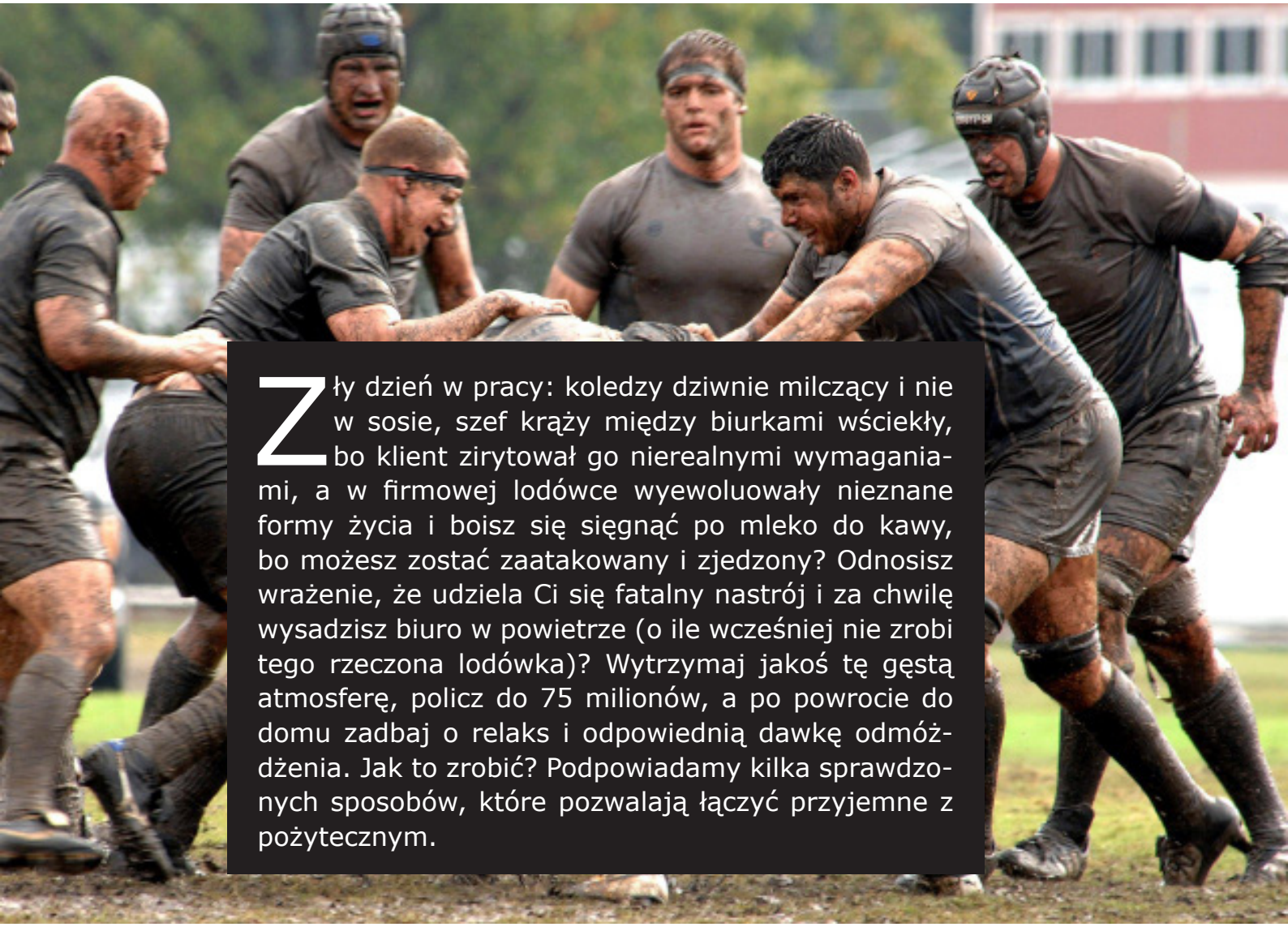
The study was published in English and German. Both versions can be downloaded in PDF format free of cost at www.imbus.de/en/downloads.
Reader feedback on the scenarios illustrated in the study is always welcome – please send it to presse@imbus.de. imbus will gladly forward the feedback to the author and thus establish the contact.

> free PDF...

imbus AG, Kleinseebacher Str. 9, 91096 Moehrendorf, Germany
Phone +49 9131 7518-0, presse@imbus.de, www.imbus.de

Zuzanna Gościcka-Miotk

W stres z półobrotu



Zły dzień w pracy: koledzy dziwnie milczący i nie w sosie, szef krąży między biurkami wściekły, bo klient zirytował go nierealnymi wymaganiami, a w firmowej lodówce wyewoluowały nieznane formy życia i boisz się sięgnąć po mleko do kawy, bo możesz zostać zaatakowany i zjedzony? Odnosisz wrażenie, że udziela Ci się fatalny nastrój i za chwilę wysadzisz biuro w powietrze (o ile wcześniej nie zrobi tego rzeczona lodówka)? Wytrzymaj jakoś tę gęstą atmosferę, policz do 75 milionów, a po powrocie do domu zadbaj o relaks i odpowiednią dawkę odmóżdżenia. Jak to zrobić? Podpowiadamy kilka sprawdzonych sposobów, które pozwalają łączyć przyjemne z pożytecznym.

Wycieraczki w rytmie disco

Przyjrzyj się oknom w swoim mieszkaniu. Widzisz jeszcze przez nie blok naprzeciwko? A może nie bardzo da się wywnioskować, czy nadal jest dzień, czy jednak zapadła już noc? Ponieważ mycie okien nie jest najrozkoszniejszą z czynności, zakładam, że szyby i ościeżnice w Twoim mieszkaniu dawno nie zaznały pieszczot dostarczanych przez płyn i szmatę.

A zatem to dzieła! Przebierz się w najbrzydszy dres, jaki masz (i najwygodniejszy), włącz ulubioną muzykę (osobiście najchętniej sprzątam przy piosenkach zespołów z lat 80., przy czym im gorsze i bardziej tandetne są to nagrania, tym lepiej: Modern Talking, italo disco, Boney M, Pet Shop Boys, Kombi, Fancy itd.) i do roboty! Przy okazji mycia okien zauważ, że każde pociągnięcie szmatą pozwalające z zaskoczeniem odkryć (bo wreszcie coś widać!), że drzewo pod blokiem już się zazieleniło („O, przyszła wiosna"), to czysta, klasyczna eksploracja, a jednostajne ruchy rękami, wystudiowane, bardzo powtarzalne i z użyciem profesjonalnych, starannie dobranych narzędzi, dziwnie kojarzą się z automatyzacją. Nie możesz zatem narzekać, że tracisz czas – doskonalisz przecież umiejętności testerskie!

Wypełnij pustkę w żołądku

Okna umyte? Pora uzupełnić miliony kalorii utraconych podczas tej niezwykle wyczerpującej czynności.

Nie gotuj tego, co zwykle. Dzisiaj zaszalej. Nie, nie będę Cię namawiać do wizyty w nadętych delikatesach i kupowania obrzydliwie drogich specjałów. Zamiast tracić czas na slalom między półkami z żarciem za pół pensji, wykorzystaj to, co masz w lodówce (zakładam, że nie wyhodowałeś w niej bliźniaka potwora z firmowej kuchni; jeśli się mylę, poszukaj w domu czegoś, co Cię nie zaatakuje i po skonsumowaniu czego nie opuścisz przedwcześnie tego łez padołu). Kombinacja jest dowolna: jajka z kalafiorem i keczupem albo np. szynka duszona w mleku i podlana musztardą miodową. Śmiało przyprawiaj całość pieprzem, solą, goździkami czy kurkumą. Gotowe danie możesz przyozdobić gałką lodów malinowych i ząbkiem czosnku.

Ponieważ jesteś profesjonalnym, odpowiedzialnym testerem, zasymuluj środowisko – nie eksperymentuj na sobie! Zaproś na obiad sąsiada i uważnie go obserwuj. Jeśli po kwadransie od rozpoczęcia posiłku nie padnie trupem, możesz do niego dołączyć. W przeciwnym razie uznaj eksperyment kulinarny za nieudany i dyskretnie pozbądź się zwłok. Zadbaj również o alibi. I najważniejsze: w drodze wyjątku poskrom testerską naturę i nie uwzględniaj tego incydentu w raporcie z testów.

Pożryj wroga

Lubisz planszówki? Nie? To polub.

Zacznij od podstaw. Namów kogoś (im więcej osób, tym lepiej) na rozgrywkę w Carcassonne, najlepiej z rozszerzeniami. Masz ochotę ubić programistę? Pozbawienie życia prawdziwego człowieka mogłoby nieść ze sobą dość nieprzyjemne skutki (od kłopotów z pozbyciem się trupa po ewentualny nieplanowany długoletni urlop w hotelu o niskim standardzie, spędzony

w nieciekawym towarzystwie). Zamiast tego podczas rozgrywki w Carcassonne podkradnij się do czyjegoś miasta (przez większość graczy i tak nazywanego zam--kiem), przejmij je, ukradnij mu rzekę albo zagarnij łąkę. Jeśli kolega z pracy wyjątkowo Ci podpadł, wywal go z zamku za pomocą księżniczki. Na koniec uaktywnij smoka i zjedz kilku współgraczy.

Jeśli znudzą Ci się mało wyrafinowane sposoby manifestowania swojej perfidii, wybierz Agricolę. Pod pozorem sielskiego budowania gospodarstwa, hodowania zwierząt, uprawiania warzyw i obsiewania pól pszenicą masz szansę skutecznie utrudniać współtowarzyszom życie. Za pomocą odpowiedniej konfiguracji małych i dużych usprawnień oraz pomocników podkradniesz komuś plony (a przynajmniej ich część) lub będziesz blokować innym rozwój poprzez umiejętne podbieranie zasobów. Testerski bonus: jeśli nie bardzo pojmujesz działanie pętli w programowaniu (podobno są tacy), przyjrzyj się mechanice Agricoli, a szybko nadrobisz zaległości.

To tylko dwie z ogromnej liczby gier, które skutecznie Cię odstresują i pozwolą rozładować agresję. Jeśli drzemie w Tobie dusza finansisty i biznesmena, wybierz Wysokie Napięcie — stwórz sieć elektrowni i nie pozwól przeciwnikowi za bardzo się rozwinąć. Miłośnikom perfidnych zagrań proponujemy Osadników (grę Ignacego Trzewiczka; nie mylić z nudnawymi Osadnikami z Catanu). Będziecie mogli się wyżyć: plądrować przeciwników, napadać samych siebie (!), przejmować cudze budowle i korzystać z udogodnień innych graczy. Wyczerpanym i niezbyt już zdolnym do wysiłku umysłowego testerom polecamy Wsiąść do Pociągu. Gra, ciesząca się wyjątkowym powodzeniem w grupie wiekowej 50+, jest prosta, spokojna i nie wymaga wielkiego zaangażowania. Autorka podczas rozgrywki z nudów układa wagoniki w szachownicę, czekając na ruchy pozostałych. Jeśli z kolei codzienne funkcjonowanie w zespole jeszcze nie wywołuje u Ciebie odruchu wymiotnego, wybierz Pandemię – grę kooperacyjną, w której co najmniej dwie osoby ratują świat przed unicestwieniem przez wyjątkowo jadowite choroby.

Można by tak wymieniać w nieskończoność: CO2 (redukujemy emisję dwutlenku węgla poprzez zastępowanie konwencjonalnych elektrowni bardziej ekologicznymi odpowiednikami; przy

założeniu, że wszyscy gracze będą chcieli wyłącznie się wzbogacić, a dobro planety mają w głębokim poważaniu, najpewniej Ziemia w końcu wybuchnie, ale co tam… Nie takie rzeczy się ze szwagrem robiło, nie?), Kolejka (dla osób z łezką w oku wspominających realia PRL-u), Ora et labora (bardziej wypasiona i nieco bardziej skomplikowana wariacja na temat Agricoli), Alternatywy 4 (gra oparta na serialu – wciel się np. w docenta Furmana, aby donosić na innych, zostań Balcerkiem – jednostką aspołeczną, ewentualnie gospodarzem bloku – Stanisławem Aniołem, który trzyma wszystkich w kieszeni; i tak nie unikniesz próby chóru w klubie lokatorskim; gra przyjemna, choć instrukcja bywa niejednoznaczna), Splendor (łatwy, prosty i przyjemny), Famiglia (dwuosobowa karcianka, w której trzeba zbudować najsilniejszą mafię, i to złożoną z przedstawicieli półświatka gangsterskiego w odmianie włoskiej, kolumbijskiej, rosyjskiej oraz amerykańskiej).

Wymęcz się przeokrutnie

Gdy wyżyjesz się na współgraczach planszówkowych, pora na konkretny wysiłek.

Jeśli lubisz biegać (są tacy?), zrób porządny trening. W trakcie tej katorgi możesz sobie wyobrażać, że z każdym przebiegniętym kilometrem zostawiasz za plecami mnóstwo złych emocji i nabierasz sił.

Jeżeli wolisz sporty zespołowe, daj z siebie wszystko na boisku, walcz do upadłego. Potraktuj grę jak wojnę. OK, broń palną i ostre narzędzia zostaw w domu.

Autorka felietonu co tydzień jeździ na tenisa. Szczerze poleca tę formę aktywności wszystkim, którzy chcą bez niepotrzebnego rozlewu krwi zredukować stres. Czyż nie można sobie wyobrazić, że niepozorna piłeczka tenisowa jest głową wyjątkowo wrednej osoby? Nic prostszego. Potem wystarczy już tylko wkładać maksymalnie dużo siły w uderzenia rakietą. Złość mija błyskawicznie, a Ty zaliczasz najlepszy trening w życiu. Jedynym niezbyt przyjemnym skutkiem ubocznym są potworne zakwasy na ramionach następnego dnia. Ale i tak warto.

Podczas uprawiania sportu dotleniasz mózg i wytwarzasz endorfiny. Poza tym spalasz kolejną tabliczkę czekolady, która podstępnie, złośliwie wskoczyła Ci do gardła. Same zalety.

Którą metodę odstresowywania wybierzesz? A może chciałbyś wypróbować wszystkie?

Dobrej zabawy!

AUTOR

Zuzanna Gościcka-Miotk

Absolwentka polonistyki (!) (specjalność edytorska) na Uniwersytecie Gdańskim. Współpracowała jako redaktor/korektor z wydawnictwami książkowymi, prasowymi oraz biurem tłumaczeń. Obecnie zajmuje się korektą magazynu „Branża Dziecięca" oraz, okazjonalnie, publikacji grupy wydawniczej Helion. W 2015 r. roku dołączyła jako korektorka do zespołu pisma „Quale".

W 2014 r. autorka zapragnęła zmian, i to dość radykalnych. Postanowiła zmienić branżę z wydawniczej na informatyczną. W tym celu rozpoczęła studia podyplomowe (kierunek: Tester oprogramowania) w gdańskiej filii Wyższej Szkoły Bankowej. W przyszłości chciałaby zawodowo znęcać się nad oprogramowaniem. Na razie testuje głównie cierpliwość męża :).

W wolnych chwilach autorka namiętnie grywa w planszówki. Raz w tygodniu pogłębia żenujący brak talentu tenisowego. Uwielbia podróżować, najchętniej w odległe (czytaj: zdecydowanie cieplejsze) regiony Europy.

TESTWAREZ

7-9 października 2015

Windsor Palace Hotel & Conference Center, Jachranka

www.testwarez.pl

Weź udział w największej w Polsce konferencji dla testerów oprogramowania.

Zapraszamy na jubileuszową, dziesiątą, edycję!

Zarejestruj się już dzisiaj!

TESTWAREZ

Zostań sponsorem Testwarez 2015

Z przyjemnością przedstawiamy Państwu ofertę wsparcia dziesiątej, jubileuszowej edycji konferencji Testwarez 2015.

Tegoroczna edycja naszej konferencji będzie przebiegała pod hasłem „Inspiracje". Jest to bardzo dobre miejsce promocji dla firmy działającej na rynku IT, a zwłaszcza na „rynku testerskim i QA". Sponsoring Testwarez pozwoli Państwu dotrzeć do szerokiego grona profesjonalistów, bardzo często odgrywających również kluczowe role w procesie decyzyjnym ich organizacji. Kooperacja z SJSI wpływa pozytywnie zarówno na budowanie marki, wizerunku i pozycji firmy, jak i na promocję usług i produktów, utrwalanie dobrych relacji z klientami oraz budowanie przewagi rynkowej. Dzięki wszystkim tym cechom sponsoring naszej konferencji korzystnie przełoży się na wyniki handlowe Państwa organizacji.

Jubileuszowa edycja Testwarez odbędzie się w hotelu Windsor w **Jachrance** niedaleko Warszawy w dniach **7-9 października 2015**. W tym roku konferencja będzie trzydniowa, przy czym dzień pierwszy w całości poświęcimy na warsztaty. Już dziś wiemy, że w Testwarez zgodzili się uczestniczyć (a także poprowadzić część warsztatów) znani propagatorzy jakości w systemach informatycznych (Dorothy Graham, David Evans, Matthias Rasking, Richard Taylor), a także nasi polscy koledzy – propagatorzy idei testowania (m.in. Tomasz Osojca, Adam Roman, Wiktor Żołnowski). Lista prelegentów nie jest jeszcze zamknięta i liczymy na liczny udział osób zajmujących się testowaniem oprogramowania z wielu organizacji z całej Polski.

Zapraszamy Państwa organizację do współpracy, a zwłaszcza do autoprezentacji podczas konferencji własnych rozwiązań związanych z testowaniem oprogramowania i zarządzania jakością. Jesteśmy otwarci na sugestie dotyczące merytorycznych aspektów programu.

Lucjan Stapp
Przewodniczący Komitetu
Organizacyjnego Konferencji
l.stapp@sjsi.org

SPONSOR GŁÓWNY

Tytuł Sponsora Głównego

Promocja przed konferencją, podczas trwania i po zakończeniu:
- 60-minutowa prezentacja podczas konferencji bądź dwie prezentacje 30 minutowe;
- logo w materiałach mailingowych (dwukrotnie większe od logo Sponsorów Wspierających);
- logo na głównej stronie Stowarzyszenia Jakości Systemów Informatycznych (www.sjsi.org);
- logo, krótka informacja o Firmie i Prelegencie w materiałach konferencyjnych;
- zamieszczenie prezentacji w materiałach konferencyjnych;
- logo na tablicach umieszczonych w salach konferencyjnych (dwukrotnie większe od logo Sponsorów Wspierających);
- możliwość dołączenia materiałów, prospektów do pakietów dla uczestników konferencji;
- możliwość dystrybucji własnych materiałów i gadżetów promocyjnych wśród uczestników;
- możliwość zbudowania punktu informacyjnego Firmy w czasie konferencji;
- bezpłatny udział w konferencji dla 2 osób.

Koszt: 12.500 PLN netto

UDZIAŁ KORPORACYJNY W KONFERENCJI

Promocja przed konferencją, podczas trwania i po zakończeniu:

- możliwość dystrybucji własnych materiałów i gadżetów promocyjnych wśród uczestników;
- możliwość zbudowania punktu informacyjnego Firmy w czasie konferencji (za dodatkową opłatą);
- możliwość umieszczenia roll-upu w foyer;
- bezpłatny udział w konferencji dla 1 osoby.

Koszt: 3.500 PLN netto

SPONSOR AFTER PARTY

- 15-minutowa prezentacja na dowolny temat/przywitanie uczestników imprezy;
- możliwość umieszczenia logo Firmy w sali, w której odbędzie się wieczorna impreza;
- informacja na stronie internetowej i w agendzie o sponsorze after party;
- bezpłatny udział w konferencji dla 1 osoby.

Taka forma wsparcia umożliwia zaprezentowanie się sponsora w zupełnie innej, przyjemniejszej i bardziej luźnej atmosferze.

Koszt: 7.500 PLN netto

Magazyn

Wydawca
VWT Polska Michał Kruszewski
Przy Lasku 8 lok. 52, 01-424 Warszawa
Numer NIP 5272137158
Numer REGON 142455963

Redaktor naczelny
Karolina Zmitrowicz
karolina.zmitrowicz@quale.pl

Zastępca redaktora naczelnego
Krzysztof Chytła
krzysztof.chytla@quale.pl

Redaktorzy
Bartłomiej Prędki
bartlomiej.predki@quale.pl
Michał Figarski
michal.figarski@quale.pl

Współpraca
Tomasz Olszewski
tomasz.olszewski@quale.pl
Piotr Poznański
piotr.poznanski@quale.pl
Zuzanna Gościcka-Miotk
zuzanna.goscicka-miotk@quale.pl

WWW
www.qualemagazine.com
www.quale.pl

Facebook
http://www.facebook.com/qualemagazine

Reklama
info@quale.pl

Współpraca
Osoby zainteresowane współpracą w zakresie publikacji prosimy o kontakt:
info@quale.pl

Grafiki użyte w magazynie
http://pixabay.com/
Grafiki na licencji CC
Free for commercial use / No attribution required

Okładka
Northern Lights, Sweden, Lapland, Aurora Borealis
http://pixabay.com/
Grafiki na licencji CC
Free for commercial use / No attribution required