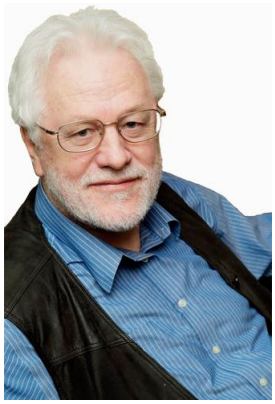
Manifest jakości: jakość oprogramowania to zadanie inżynierii systemowej

Autor: Tom Gilb

O autorze: Tom Gilb jest międzynarodowym konsultantem, nauczycielem i autorem. Jego dziewiąta książka „Competitive Engineering: A Handbook For Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage" to definicja języka planowania – "Planguage". Tom pracował z głównymi międzynarodowymi organizacjami, takimi jak Credit Suisse, Schlumberger, Bosch, Qualcomm, HP, IBM, Nokia, Ericsson, Motorola, US DOD, UK MOD, Symbian, Philips, Intel, Citigroup, United Health, Boeing, Microsoft. Tom gościnnie wykłada na uniwersytetach w Wielkiej Brytanii, Chinach, Indiach, USA, Korei i występuje jako prelegent na wielu znaczących międzynarodowych konferencjach.



Kontakt: www.gilb.com, twitter @imtomgilb

Abstrakt

Głównym celem niniejszego artykułu jest obudzenie inżynierów oprogramowania – oraz może niektórych inżynierów systemów – w kwestii jakości. Inżynierowie oprogramowania (przepraszam, raczej „rzemieślnicy") wydają się uważać, iż istnieje tylko jeden typ jakości (brak błędów) i tylko jedno miejsce, w którym błędy są znajdowane (czyli – programy). Sednem jest tutaj to, że pytanie o jakość obejmuje o wiele szerszy zakres. Jedynym sposobem na uzyskanie całkowitej koniecznej jakości w oprogramowaniu jest potraktowanie problemu z punktu widzenia dojrzałego inżyniera systemowego. Oznacza to rozpoznanie wszystkich krytycznych typów jakości w danym systemie. To znaczy – zastosować podejście architektoniczne i inżynierskie do dostarczenia wymaganej jakości. To znaczy – przestać być tak bardzo skoncentrowanym na programie komputerowym a uświadomić sobie, że nawet w świecie oprogramowania jest o wiele więcej domen projektowych, niż same programy. Świat oprogramowania jest blisko związany ze światem ludzi i sprzętu – i nie można po prostu próbować rozwiązywać istniejących w nich problemów jakości w zupełnym odosobnieniu. Proponuję tu kilka zasad, aby powyższe zasady uwydatnić i pomóc w ich zrozumieniu.

Manifest jakości

Grupa moich przyjaciół spędziła lato 2007r. na emailowej dyskusji o Manifeście Jakości Oprogramowania. Byłem tak niezadowolony z rezultatu tej dyskusji, że zdecydowałem napisać własny manifest. Przynajmniej nie byłem ograniczany przez komitet.

Tytuł: Jakość Oprogramowania to zadanie Inżynierii systemowej

Slogan:

Propozycja:

„Doskonałe cechy jakości systemu są ciągłym wyzwaniem dla zarządzania i inżynierii, jednak bez perfekcyjnych rozwiązań".

Wniosek:

"Kiedy zarządzanie i inżynieria zawodzą w wykonywaniu swoich obowiązków dotyczących jakości, poziomy jakości są przypadkowe i prawdopodobnie niesatysfakcjonujące dla większości udziałowców".

Manifest Jakości/Deklaracja

Jakość systemu może być postrzegana jako zbiór ilościowych atrybutów wykonywania, które opisują, jak dobrze system działa dla udziałowców, pod określonymi warunkami i w danym czasie.

Udziałowcy systemu oceniają przeszłe, obecne i przyszłe poziomy jakości – w odniesieniu do ich potrzeb i wartości.

Inżynierowie systemowi mogą analizować potrzebne, i pożądane, poziomy jakości; planować i zarządzać dostarczeniem zestawu takich poziomów jakości, przy określonych ograniczeniach i dostępnych zasobach.

Zarządzanie jakością jest odpowiedzialne za ustalenie priorytetów użycia zasobów, za dopasowanie satysfakcji dla poziomów jakości o określonych priorytetach; i za próbę zarządzania dostarczeniem zestawu wartości – co maksymalizuje stosunek wartości do kosztu – do określonych udziałowców.

Zasady jakości. Heurystyki działania: przegląd

1. Projekt jakości: ambitne poziomy jakości są wprojektowane, nie testowane. Dotyczy to zarówno procesów, jak i produktów.
2. Środowisko oprogramowania: jakość „oprogramowania" jest całkowicie zależna od wbudowanej jakości systemu i nie istnieje oddzielnie: „cechy jakościowe oprogramowania" są zależne od zdefiniowanych cech jakości systemu – włączając w to wyobrażenia i wartości udziałowców.
3. Entropia jakości: istniejące lub planowane poziomy jakości będą się pogarszać w miarę upływu czasu pod naciskiem innych wymagań o określonym priorytecie oraz w wyniku braku ustawicznej kontroli.
4. Zarządzanie jakością: poziomy jakości mogą być systematycznie zarządzane w celu wspierania określonej polityki jakości. Przykład: „W pierwszej kolejności wartość za cenę" lub „Najbardziej konkurencyjne, Światowej Klasy, Poziomy Jakości".

5. Inżynieria jakości: zbiór poziomów jakości może być przedmiotem inżynierii technicznej, celem spełnienia ambicji udziałowców, przy określonych ograniczeniach i priorytetach.
6. Postrzeganie jakości: jakość leży w oczach obserwatora: obiektywne poziomy jakości systemu mogą być oceniane jako świetne dla niektórych udziałowców, a beznadziejne dla innych.
7. Wpływ projektowania na jakość: każdy komponent projektu systemu, do czegokolwiek by nie był przeznaczony, będzie prawdopodobnie miał nieprzewidywalne – główne i poboczne – wpływy na wiele innych poziomów jakości, na wiele ograniczeń oraz zasobów.
8. Rzeczywiste wpływy projektowania: nie można być pewnym kompleksowości wpływów projektu jakości na system, za wyjątkiem mierzenia ich w praktyce – a nawet wtedy, nie można być pewnym, czy pomiar jest ogólny, czy się nie będzie pogarszał.
9. Niezależność designu: poziomy jakości mogą być mierzone i specyfikowane niezależnie od środków (czy projektów) potrzebnych do ich osiągnięcia.
10. Złożone cechy jakościowe: wiele cech jakościowych jest najlepiej definiowana jako przedmiotowy, lecz użyteczny zbiór elementarnych obszarów jakości; zależy to od stopnia kontroli, jaki chcemy uzyskać nad poszczególnymi obszarami jakości[1].

Zasady jakości. Heurystyki działania: szczegółowe uwagi

1. **Projekt jakości**: ambitne poziomy jakości są zaprojektowane, nie testowane. Dotyczy to zarówno procesów jak i produktów.

Istnieje za duży nacisk na testowanie i przeglądy, jako na środki radzenia sobie z defektami i błędami. Jest to dobrze znany paradygmat – „nie testujesz jakości w systemie, projektujesz ją dla systemu". Możemy spojrzeć na ten problem zarówno z ekonomicznego, jak i efektywnościowego punktu widzenia. Od 44% do 64% wszystkich defektów w kodzie jest wynikiem błędów w specyfikacji (wymaganiach, projekcie) dostarczonej programistom [Inspection for Managers [ATT, TRW], jako referencja dla tego i innych faktów dotyczących testowania i przeglądów]. Koszt usunięcia defektów w późnych fazach może wzrosnąć nawet 100-krotnie. Wcześniejsza naprawa oszczędza sporą część funduszy.

Z punktu widzenia efektywności, zarówno testowanie, jak i przeglądy nie są efektywne. Poziom efektywności wynosi od 25% do 75% (prawdopodobieństwo wykrycia defektów, które istnieją w systemie [Insp. For Mgt., Capers Jones]. Jones twierdzi, iż jeśli mieliśmy efektywne serie, około 11 przeglądów i testów, mogliśmy usunąć jedynie co najwyżej 95% wstrzykniętych defektów. Zmierzam do tego, że „czyszczenie wstrzykniętych

[1] CE rozdział 5, do pobrania z http://www.gilb.com/community/tikidownload_file.php?fileId=26 dostarczy obszernej ilustracji dla tego punktu.

defektów” jest sprawą beznadziejną. Istnieją lepsze możliwości. Ciekawą opcją jest to, że “uncja prewencji jest warta funta leku”. W pierwszej kolejności musimy nauczyć się, jak unikać pojawiania się defektów. Oczywiste jest, że możemy zredukować ilość defektów co najmniej ze 100 do 1. Obecnie większość dokumentów specyfikacji (szacunki mojego klienta) zawiera około 100 poważnych błędów na stronę (300 słów). Standardem, jaki dawno temu ustanowili doświadczeni programiści (IBM [Humphrey], NASA) jest tolerancja (poziom wyjścia procesu) na poziomie mniejszym, niż 1.0 poważnych błędów na stronę (IBM: 0.25, NASA: 0.1). To jest głównym celem Poziomu 5 CMMI (Proces Zapobiegania Błędom [Mays, Robert, IBM]). Mojemu klientowi zabiera około 6 miesięcy zredukowanie pojawiania się błędów o czynnik równy 10; następne 2-3 lata – zredukowanie o kolejny czynnik 10. To oczywiście bardziej efektywne pod względem kosztu, niż czekanie, aż będziemy mogli testować defekty, lub nim klienci zaczną się skarżyć.

2. **Środowisko oprogramowania**: jakość „oprogramowania” jest całkowicie zależna od wbudowanej jakości systemu i nie istnieje oddzielnie: „cechy jakościowe oprogramowania” są zależne od zdefiniowanych cech jakości systemu – włączając w to wyobrażenia i wartości udziałowców.

Mamy tendencję do traktowania jakości oprogramowania jako czegoś nieodłącznie rezydującego w samym oprogramowaniu. Jednak wszystkie cechy jakościowe (np. bezpieczeństwo, użyteczność, zdolność utrzymywania, niezawodność) są wysoce zależne od ludzi, ich kwalifikacji i sposobu, w jaki używają systemów. Konsekwencją jest to, że musimy planować, specyfikować i projektować z większą uwagą na identyfikację i kontrolowanie czynników, które rzeczywiście decydują o jakości systemu. Musimy tworzyć systemy jako całość, nie tylko „kod”. Musimy być inżynierami systemu, nie inżynierami programów. Ma to ogromne implikacje na sposób szkolenia ludzi, organizacji naszej pracy i tego, w jaki sposób motywujemy ludzi. Musimy również przenieść nacisk z samej technologii (środki) na rezultaty, których potrzebujemy (poziomy wymagań jakościowych).

3. **Entropia jakości**: istniejące lub planowane poziomy jakości będą się pogarszać w miarę upływu czasu, pod naciskiem innych wymagań o określonym priorytecie oraz w wyniku braku ustawicznej kontroli.

Nawet koncepcja numerycznych poziomów jakości, dla różnych cech jakościowych – użyteczności, bezpieczeństwa, zdolności adaptacji – jest obca dla większości inżynierów oprogramowania i zdecydowanie dla zbyt wielu inżynierów systemów. Mimo tego ubogiego środowiska startowego, mimo zbyt wielu ludzi zadowolonych ze słów („łatwy w użyciu”), zamiast liczb („30 minut na nauczenie się wykonywania zadania X przez pracownika typu Y”), musimy archiwizować planowane poziomy jakości od wstępnego ich dostarczenia do akceptacji systemu. Musimy zawrzeć w systemach pomiary takich cech jakościowych oraz środki ostrzegawcze, które powiedzą nam, kiedy te cechy pogarszają się lub drastycznie obniżają. Musimy oczekiwać podejmowania akcji zmierzających do ulepszenia poziomów jakości w odniesieniu do planowanych poziomów – oraz być może udoskonalić je jeszcze bardziej w przyszłości.

4. **Zarządzanie jakością**: poziomy jakości mogą być systematycznie zarządzane w celu wspierania określonej polityki jakości. Przykład: „W pierwszej kolejności wartość za cenę" lub „Najbardziej konkurencyjne, Światowej Klasy, Poziomy Jakości".

Użyteczne zarządzanie ma miejsce wtedy, kiedy istnieje polityka poziomów jakości, do których aspirujemy, zarówno na poziomie korporacji, jak i poziomu projektowego. Nie możemy pozwalać odizolowanym jednostkom, by ich wymarzone poziomy jakości były brane jako wymagania, bez równowagi z priorytetami innych konkurencyjnych poziomów. Dodatkowo, musimy „mieć oko" na dostępne zasoby oraz limity i możliwości technologiczne. Musimy zdecydować, czy jesteśmy tu by „reprezentować aktualny stan wiedzy" (jak kiedyś wyjaśnił mi Rockwell) w „uzyskiwaniu jak największego stosunku wartości do kosztu" – a inni niech się martwią. Taka polityka może być generalnie pożyteczna: „Poziomy jakości będą tworzone do poziomu, który daje nam wysoki zwrot inwestycji potrzebny, by uzyskać te poziomy – i w taki sposób, by owe poziomy nie skonsumowały zasobów dla innych równoległych możliwości inwestycji w jakość, lub gdziekolwiek indziej."

5. **Inżynieria jakości**: zbiór poziomów jakości może być przedmiotem inżynierii technicznej, celem spełnienia ambicji udziałowców, przy określonych ograniczeniach i priorytetach.

Decydowanie, które liczbowe poziomy jakości są odpowiednie, jest trudną sprawą. Wstępnie – nie możemy określić właściwych poziomów w izolacji. Musimy poznać większe środowisko, zarówno środowisko dla pojedynczego atrybutu jakości, jak i dla zbioru atrybutów – w ich środowisku. Musimy nauczyć się określać to środowisko wraz z samymi wymaganiami. Będzie łatwiej decydować o poziomach jakości i ich priorytetach, jeśli będziemy mieć decyzyjny zbiór danych o każdym atrybucie. Przykładowo – użyteczne jest posiadanie wiedzy o:

- Wartości dla danego poziomu
- Udziałowcach jakości oraz różnych poziomach
- Czasie potrzebnym dla poziomów jakości
- Planowanych strategiach i ich oczekiwanych kosztach wymaganych do uzyskania danego poziomu

Oraz jeszcze kilka innych rzeczy – które pomogą nam umotywować wybór właściwych poziomów jakości.

Elementarny szablon skalarnego wymagania <ze wskazówkami>

Tag: <Nazwa taga elementarnego skalarnego wymagania>,
Typ:
<{Wymaganie wydajnościowe: {Wymaganie Jakościowe,
Wymaganie Oszczędności Zasobów,
Wymaganie Obciążeniowe},
Wymaganie Zasobów: {Wymaganie Finansowe,
Wymaganie Czasowe,

Wymaganie Załogowe,

Inne}}>.

Podstawowe informacje

Wersja: <Data lub inny numer wersji>.

Status: <{Szkic, Zamknięte przez SQC, Zaakceptowane, Odrzucone}>.

Poziom Jakości: <Maksimum pozostałych defektów na stronę, przykładowy rozmiar, data>.

Właściciel: <Rola/e-mail/imię osoby odpowiedzialnej za specyfikację>.

Udziałowcy: <Nazwa każdego udziałowca mającego interes w specyfikacji>.

Sedno: <Krótki opis uwzględniający istotne znaczenie wymagania>.

Opis: <Opcjonalny, pełen opis wymagania>.

Ambicja: <Podsumowanie poziomu ambicji wyłącznie celów poniżej. Podaj ogólny rzeczywisty poziom ambicji w 5-20 słowach>.

Skala pomiaru

Skala: <Skala pomiaru dla wymagania (stwierdza jednostki pomiary dla wszystkich celów, ograniczeń i benchmarków) oraz kwalifikatory skali>.

Pomiar

Licznik: <Metoda, która będzie używana to uzyskania pomiarów w zdefiniowanej Skali>.

Benchmarki „Minione Wartości Liczbowe”

Minione [<kiedy, gdzie, czy>]: <Przeszły lub obecny poziom. Określ, jeśli jest to estyma ta> <-<Źródło>.

Rekord [<kiedy, gdzie, czy>]: <Stwierdzenie poziomu umiejętności> <-<Źródło>.

Trend [<kiedy, gdzie, czy>]: <Warunek początkowy wskaźnika zmian lub przyszłego stwierdzenia poziomu umiejętności> <-<Źródło>.

Cele „Przyszłe Wartości Liczbowe”

Cel/Budżet [<kiedy, gdzie, czy>]: <Planowany poziom celu> <-<Źródło>.

Rozciągnięcie [<kiedy, gdzie, czy>]: <Motywacyjny poziom ambicji> <-<Źródło>.

Życzenie [<kiedy, gdzie, czy>]: <Poziom „marzenia” (niebudżetowany)> <-<Źródło>.

Ograniczenia „Specyficzne Restrykcje”

Porażka [<kiedy, gdzie, czy>]: <Poziom porażki> <-<Źródło>.

Przetrwanie [<kiedy, gdzie, czy>]: <Poziom przetrwania> <-<Źródło>.

Powiązania

Jest częścią: <Odniesienie do tagów wszelkich ponad-wymagań (wymagań złożonych), których częścią jest dane wymaganie. Preferowana jest hierarchia tagów (Np. A, B, C)>.

Ma na niego wpływ: <Odniesienie do tagów wszelkich koncepcji projektowych, które mają wpływ na to wymaganie><-<Źródło>.

Wpływy: <Nazwij wszystkie wymagania, design lub plany, na które wymaganie ma znaczący wpływ>.

Priorytet i Zarządzanie Ryzykiem

Uzasadnienie: <Oceń, dlaczego wymaganie istnieje>.

Wartość: <Nazwa [udziałowiec, czas, miejsce, zdarzenie]: Określ ilościowo lub wyraź w słowach wartość deklarowaną jako wynik dostarczenia wymagania>.
Założenia: <Określ wszelkie założenia przyjęte w związku z wymaganiem> <-<Źródło>.
Zależności: < Określ wszystko to, od czego zależne jest osiągnięcie danego wymagania> <-<Źródło>.
Ryzyka: <Wyszczególnij lub podaj odniesienia do tagów określających wszystko, co może spowodować opóźnienia lub negatywny wpływ> <-<Źródło>.
Priorytet: < Wyszczególnij lub podaj odniesienia do tagów każdego elementu systemu, który musi być zaimplementowany przed lub po danym wymaganiu>.
Problemy: <Określ wszelkie znane problemy>.

Rysunek 1 Przykład szablonu do zbierania informacji, które według mnie powinniśmy posiadać, aby decydować, jak ustalić priorytety poszczególnych poziomów jakości dla pojedynczego atrybutu [CE, strona 135]

6. **Postrzeganie jakości**: jakość leży w oczach obserwatora: obiektywne poziomy jakości systemu mogą być oceniane jako świetne dla niektórych udziałowców, a straszne dla innych.

Sednem jest to, że jakikolwiek złożony system będzie miał wielu różnych udziałowców. Nawet jedna kategoria udziałowców [Novice User, Call Center Manager] może mieć wielu przedstawicieli o bardzo indywidualnych potrzebach i priorytetach. Wynikiem nieuchronnie będzie kompromis. Jednak możemy uczynić ten kompromis tak inteligentnym, jak tylko możliwe. Nie jest naszym celem projektowanie systemów tylko dla jednego poziomu wszystkich udziałowców. Możemy świadomie decydować o utrzymywaniu różnych poziomów jakości dla tej samej jakości – dla różnych udziałowców, w różnych momentach i sytuacjach.

Dla przykładu:

Zdolność uczenia się:

Skala: czas potrzebny dla określonego [Udziałowca], by zarządzać określonym [Procesem].

Cel [Udziałowiec = Manager Wysokiego Szczebla, Proces = Uzyskaj raport] 5 minut.

Cel [Udziałowiec = Zdalnie pracujący Urzędnik, Proces = Stwórz nowe konto] 1 godzina.

7. Wpływ projektowania na jakość: każdy komponent projektu systemu, do czegokolwiek by nie był przeznaczony, będzie prawdopodobnie miał nieprzewidywalne – główne i poboczne – wpływy na wiele innych poziomów jakości, na wiele ograniczeń oraz zasobów.

Dostrzegam zbyt wiele ograniczonego i jednostronnego rozumowania w stylu: „zamierzamy osiągnąć wspaniałą jakość X przy wykorzystaniu technologii X, Y i Z". Takie rozumowanie nie dotyczy liczb, a tylko miłych słów. Widziałem to nawet w projektach o budżecie 100 milionów $ - i to często!

Musimy nauczyć się specyfikować, analizować i myśleć w kategoriach „złożonych liczbowych wpływów różnych projektów (rozwiązań) na wiele naszych krytycznych wymagań dotyczących jakości i kosztu". Umożliwia to metoda Zastosowania Funkcji Jakości (ang. *Quality Function Deployment* (QFD)), jednak nie jestem zadowolony ze sposobu, w jaki używa się w niej liczb – zbyt subiektywnie, w niewystarczająco zdefiniowany sposób [QFD].

Musimy systematycznie, najlepiej jak potrafimy, szacować wielorakie wpływy każdego znaczącego rozwiązania (projektu).

	Wsparcie Online	Pomoc Online	Podręcznik	Pomoc Online + dostęp do indeksu
Uczenie się **Było: 60 min. Planowane: 10 min.**				
Wpływ na skalę	5 min.	10 min.	30 min.	8 min.
Niepewność skali	± 3 min.	± 5 min.	± 10 min.	± 5 min.
Wpływ procentowy	110%	100%	67% (2/3)	104%
Niepewność procentowa	± 6% (3 z 50 minut)	± 10%	± 20% ?	± 10%
Dowód	Projekt Ajax, 1997 r., 7 min.	Inne systemy	Zgadywanie	Inne systemy + zgadywanie
Źródło	Raport Ajax, str. 6	Raport światowy, str. 17	John B.	Raport światowy, str. 17 + John B.
Wiarygodność	0.7	0.8	0.2	0.6
Koszt developmentu	120K	25K	10K	26K
Stosunek korzyści do jakości	110/120=0.92	100/25=4.0	67/100=6.7	104/26=4.0
Stosunek korzyści do jakości po uwzględnieniu wiarygodności	0.92*0.7-0.6	4.0*0.8=3.2	6.7*0.2=1.3	4.0*0.6=2.4
Uwagi: Okres wynosi 2 lata	Dłuższy czas developementu			

Tabela 1 Systematyczna analiza czterech projektów jednego poziomu jakości. Tabela szacowania wpływu [CE, strona 267]

8. Rzeczywiste wpływy projektowania: nie można być pewnym kompleksowości wpływów projektu jakości na system, za wyjątkiem mierzenia ich w praktyce – a nawet wtedy, nie można być pewnym, czy pomiar jest ogólny, czy się nie będzie pogarszał.

Widziałem książki, artykuły i specyfikacje projektowe, które z dużą dozą pewności przewidywały dobre rezultaty (nie zawsze liczbowe) na podstawie poszczególnych projektów, rozwiązań, architektury lub strategii. Może łatwiej jest być pewnym siebie, jeśli nie zapewnia się żadnego wpływu liczbowego. W normalnej inżynierii – nieważne, co mówią podręczniki, nieważne, w co chcielibyśmy wierzyć – rozważny inżynier zadaje sobie trud zmierzenia *prawdziwych* efektów. Musimy starannie wykonywać wczesne pomiary, następnie powtarzać je w miarę skalowania systemu, w momentach akceptacji – i później, w długoterminowym działaniu. Nie możemy nigdy brać krytycznych cech jakościowych za pewnik, lub zakładać, iż są stabilne. Możemy w rozsądnym stopniu to zaplanować .

Dla przykładu:

Zdolność uczenia się: Skala: czas potrzebny do nauczenia się Zadania przez Użytkownika. Licznik [Development tygodniowy, 2 Użytkowników, 10 normalnych zadań] Licznik [Testy akceptacyjne, Czas trwania 60 dni, 200 Użytkowników, 10 normalnych zadań, 20 ekstremalnych zadań] Licznik [Normalne działanie, Częstotliwość działania 2%, Zadania = wszystkie zdefiniowane]

Każda specyfikacja „Licznika” definiuje lub nakreśla test o różnym zamierzeniu dla pomiaru poziomu jakości.

9. Niezależność designu: poziomy jakości mogą być mierzone i specyfikowane niezależnie od środków (czy projektów) potrzebnych do ich osiągnięcia.

Istnieje zdecydowanie za dużo idei projektowych z nazwanymi typami jakości. „Poprawimy zwinność produktu przy wykorzystaniu ustrukturyzowanych narzędzi” – oto typ specyfikacji. W naszej specyfikacji musimy położyć nacisk na wymagane poziomy jakości i unikać wymieniania ulubionych idei projektowych w tym samym zdaniu. Określanie „designu”, kiedy powinieneś się skupić na poziomie jakości, powinno być traktowane jako poważny błąd w specyfikacji. Tuzin lub więcej takich „fałszywych wymagań” na każdej stronie „wymagań” to nic niezwykłego w naszej kulturze „software’owej”.

10. Złożone cechy jakościowe: wiele cech jakościowych jest najlepiej definiowana jako przedmiotowy, lecz użyteczny zbiór elementarnych wymiarów obszarów jakości; zależy to od stopnia kontroli, jaki chcemy uzyskać nad poszczególnymi obszarami jakości.

Wydaje mi się, że brakuje świadomości na temat faktu, iż słowa określające jakość są często nazwą zbioru cech jakościowych. Jedynym sposobem określenia takich złożonych cech jest stworzenie listy wszystkich komponentów zbioru. Tylko takim sposobem zrozumiemy, czym są prawdziwe wymagania. Musimy nauczyć się ogólnych wzorów najczęściej występujących cech jakościowych – jak w przykładzie poniżej.

Musimy też unikać nadmiernego upraszczania cech jakościowych, kiedy uszczegółowiony zbiór podatrybutów daje nam łatwą szansę uzyskania kontroli nad krytycznymi cechami, którymi chcemy zarządzać.

Zdolność utrzymania:

Typ: Złożone Wymaganie Jakościowe.

Obejmuje: {Rozpoznanie Problemu, Opóźnienie Administracyjne, Gromadzenie Narzędzi, Analizę Problemu, Specyfikację Zmian, Kontrolę Jakości, Implementację Modyfikacji, Testowanie Modyfikacji {Testowanie Jednostkowe, Testowanie Integracyjne, Testowanie Beta, Testowanie Systemowe}, Odzyskiwanie}.

Rozpoznanie Problemu:

Skala: Godziny zegarowe od zdefiniowania [Wystąpienie Błędu: Domyślnie: Wystąpił błąd w jakimkolwiek użyciu lub testowaniu systemu] do oficjalnego rozpoznania błędu [Czynności Rozpoznawania: Domyślnie: Błąd jest logowany elektronicznie].

Opóźnienie Administracyjne:

Skala: Godziny zegarowe od zdefiniowania [Czynności Rozpoznawania] do zdefiniowania [Czynności Korekcyjna] zainicjowania i alokowania do określonej [Instancji Utrzymania].

Zbieranie Narzędzi:

Skala: Godziny zegarowe od zdefiniowania [Instancji Utrzymania: Domyślnie: Ktokolwiek jest alokowany] do uzyskania wszystkich wymaganych [Narzędzi: Domyślnie: wszystkie systemy i informacje konieczne do przeanalizowania, skorygowania i kontroli jakości poprawki].

Analiza Problemu:

Skala: Godziny zegarowe od zdefiniowania [Instancji Utrzymania] do przeanalizowania symptomów błędu i nabycia możliwości sformułowania hipotezy korekty.

Specyfikacja Zmian:

Skala: Godziny zegarowe od zdefiniowania [Instancji Utrzymania] do pełnego i prawidłowego opisania niezbędnych czynności korygujących, zgodnie z aktualnymi standardami odnoszącymi się do tego.

Uwaga: Obejmuje to każdy dodatkowy czas na korekty po kontroli jakości i testach.

Kontrola Jakości:

Skala: Godziny zegarowe dla kontroli jakości hipotez korekty (na zgodność z odpowiednimi standardami).

Implementacja Modyfikacji:

Skala: Godziny zegarowe do przeprowadzenia czynności korekcyjnych jak planowano. „Obejmuje wszelkie niezbędne korekty pojawiające się w wyniku kontroli jakości i testowania".

Testowanie Modyfikacji:

Testowanie Jednostkowe:

Skala: Godziny zegarowe do przeprowadzenia określonych [Testów Jednostkowych] dla poprawki błędu.

Testowanie Integracyjne:

Skala: Godziny zegarowe do przeprowadzenia określonych [Testów Integracyjnych] dla poprawki błędu.

Testowanie Beta:

Skala: Godziny zegarowe do przeprowadzenia określonych [Testów Beta] dla poprawki błędu przed pozwoleniem na oficjalne dostarczenie poprawki.

Testowanie Systemowe:

Skala: Godziny zegarowe do przeprowadzenia określonych [Testów Systemowych] dla poprawki błędu.

Odzyskiwanie:

Skala: Godziny zegarowe dla określonych [Typów Użytkowników]w celu przywrócenia system do stanu, w jakim był przed pojawieniem się błędu i do stwierdzenia gotowości do kontynuowania działania.

Źródło: Powyższe jest rozszerzeniem pewnych podstawowych koncepcji z Ireson, Editor, Reliability Handbook, McGraw Hill, 1966 (Ireson 1966).

Rysunek 2 Przykład Zdolności utrzymania jako zbioru innych mierników jakości [CE, strona 156]

Podsumowanie

Cel (Manifestu Jakości)

Promować zdrowy pogląd na jakość oprogramowania.

Analiza luk: pomóc ludziom zrozumieć, czego potrzebują by spełnić oczekiwania swoich udziałowców w zgodzie z dostępnymi zasobami.

Uzasadnienie (opisanych pozycji)

1. Musimy przyjąć podgląd na jakość koncentrujący się na **systemach**, nie na programowaniu. Powód: oprogramowanie posiada cechy jakościowe tylko w odniesieniu do ludzi, sprzętu, danych, sieci, wartości. Nie może być odizolowane od świata, który decyduje:

- jakie wymiary jakości leżą w obszarze zainteresowań (krytycznych),
- jakie poziomy jakości są wartościowe dla danej grupy udziałowców.

2. Musimy przyjąć punkt widzenia **udziałowca** – nie klienta, czy użytkownika, czy też jakiejkolwiek ograniczonej grupy udziałowców. Powód: cechy jakościowe, jakie muszą być wytworzone i ostatecznie obecne w oprogramowaniu zależą od całej grupy krytycznych udziałowców; nie od kilku z nich.
3. Musimy wyraźnie **odróżnić** różne typy **defektów** – tak jak zrobiły to już dobre standardy IEEE. Powód: nie możemy sobie pozwolić na mieszanie defektów w specyfikacji z ich potencjalnymi skutkami w postaci błędów produktu, oraz błędów produktu od potencjalnych dysfunkcji produktu.

Referencje

1. Gilb, Tom, Competitive Engineering [CE], „A Handbook For Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage", ISBN 0750665076, 2005, Wydawca: Elsevier Butterworth-Heinemann. Przykładowe rozdziały są dostępne pod www.Gilb.com
2. Chapter 5: Scales of Measure: http://www.gilb.com/community/tiki-download_file.php?fileId=26
3. Chapter 10: Evolutionary Project Management: http://www.gilb.com/community/tiki-download_file.php?fileId=77
4. Gilb: „Inspekcje dla managerów, zestaw slajdów z faktami i przypadkami" http://www.gilb.com/community/tiki-download_file.php?fileId=88
5. Gilb: "What's Wrong with QFD?" http://www.gilb.com/community/tiki-download_file.php?fileId=119
6. INCOSE Systems Engineering Handbook v. 3
7. INCOSE-TP-2003-002-03, June 2006 , www.INCOSE.org
8. Software World Conference Website:

- Bethesda Md., USA, September 15-18th 2008
- http://www.asq509.org/ht/display/EventDetails/i/18370