[24] Sposoby radzenia sobie ze złożonością. Wybrane aspekty oceny jakości oprogramowania.

Sposoby radzenia sobie ze złożonością.

Podstawowym sposobem mierzenia złożoności w projektach informatycznych stanowić mogą metody tzw. punktów funkcyjnych lub punktów przypadków użycia (Use Case Points). W dalszej części będzie tylko o tej drugiej, jako że więcej mieliśmy z nią do czynienia na IO.

Metoda ta polega na przypisaniu każdemu przypadkowi użycia wartości punktowej w zależności od stopnia jego złożoności czyli np. ilości klas czy wywołań które są przez ten przypadek użycia wywoływane. Po zdefiniowaniu wszystkich przypadków użycia i określeniu ich skomplikowania punkty są sumowane a otrzymany wynik reprezentuje pierwotną ogólną złożoność systemu.

Dodatkowo są wprowadzane są jeszcze funkcje korygujące które w zależności od dodatkowych warunków wpływają korzystnie (zmniejszają liczbę punktów) lub nie. Wartość funkcji korygujących przemnaża się przez pierwotną liczbę punktów w ten sposób otrzymując ogólną złożoność systemu. Do warunków niekorzystnych zaliczyć można np. stosowanie systemu rozproszonego, zwiększoną kontrolę dostępu czy zatrudnianie ludzi z zewnątrz. Natomiast kilka pozytywnych to min. doświadczenie osób zarządzających projektem, kwalifikacje zespołu, czy klarowna specyfikacja wymagań. Od siebie mogę dodać, że znaczący jest też w miarę myślący i zdecydowany klient.

Po korekcji wyniku otrzymujemy wartość, która odpowiada stopniowi skomplikowania oprogramowania. Dodatkowo, szacuje się że każdy punkt oznacza około 20 osobogodzin pracy programisty.

Myślę że po takim wstępie sposoby radzenia sobie ze złożonością stają się powoli jasne.

Najlepszym sposobem na ograniczenie złożoności może być ograniczenie funkcjonalności lub takie zaprojektowanie przypadków użycia aby można było ich użyć w wielu miejscach systemu. Pozwala to zmniejszyć projekt o pojedyncze zadania.

Drugą dobrą metodą może być ograniczenie komplikacji samych przypadków użycia. Może się więc okazać, że połączenie dwóch klas, między którymi istnieje wiele wzajemnych odwołań oraz z których korzysta wiele przypadków użycia, przyniesie spadek złożoności więcej niż jednego przypadku użycia, co przełoży się na mniejszą liczbę punktów.

Z trzeciej strony warto się zastanowić czy nie da się ominąć niektórych niekorzystnych warunków wpływających na projekt np. wymaganie od klienta przenośności kodu. Nie ma po co pisać w C++ aplikacji która pójdzie na Windach, Linuksie i Macach skoro w firmie są tylko windy, a instalacji pozostałych nie planuje się w najbliższym czasie.

Jednym z chyba bardziej niedocenianych na początku czynników wpływających na komplikacje w kodzie jest klient, który w momencie gdy dana część systemu jest już prawie gotowa postanawia dołożyć funkcję która wymaga przebudowy połowy wnętrza. Takie sytuacje w przypadku silnej presji czasu powodują powstanie "na szybko" łatek (często bez przyzwoitej dokumentacji), które potrafią bardzo skomplikować późniejszą pielęgnację kodu.

Ostatnim z wybranych przeze mnie sposobem na ograniczenie komplikacji kodu jest coś co nie łączy się ściśle z powyższą metodologią. Chodzi o mądre zaprojektowanie systemu nie w postaci monolitu (jedna czarna skrzynka) a zestawu połączonych elementów (kilka klocków które się komunikują). Na takim podejściu zyskujemy głównie dzięki zmianie perspektywy. Zamiast jednego "kloca" złożonego z powiedzmy kilkudziesięciu klas, mamy kilka modułów a w każdym kilkanaście klas. To pozwala uprościć działanie na poziomie pojedynczych modułów. Dodatkowo traktowanie każdego modułu jako czarnej skrzynki pozwala na niezależne rozwijanie poszczególnych części systemu.

Wybrane aspekty oceny jakości oprogramowania.

Mówiąc o jakości oprogramowania mamy większości wypadków do czynienia z tzw. metrykami oprogramowania. Są to wzory pozwalające w sposób automatyczny określić pewne cechy oprogramowania i zwrócić liczbę opisującą daną cechę np. liczba linii kodu.

Oprogramowanie

Jakość oprogramowania zależy od wielu czynników. Do wymienionych w normie zaliczają się funkcjonalność, niezawodność, użyteczność (tu rozumiana jako przyjazny dla użytkownika), efektywność (wydajność), konserwowalność, czy przenośność.

Każdy z tych aspektów posiada własne zbiory metryk. Najciekawszym moim zdaniem aspektem jest konserwowalność, ponieważ łączy się ona z możliwością poddania kodu wszelkiego rodzaju analizom (metryki kodu) i testom.

Równie ciekawa wydaje się atrybut o nazwie funkcjonalność, ponieważ w przypadku dania klientowi możliwości zmiany funkcjonalności oprogramowania w trakcie prac mamy możliwość śledzenia wpływu tych zmian na projekt. Ma to znaczenie np. w przypadku gdy odkrywamy, że w danym okresie liczba nowych funkcjonalności sprzecznych z już istniejącym kodem rośnie. Szybkie wychwycenie takich tendencji pozwala kierownictwu projektu zareagować odpowiednio wcześnie i złagodzić skutki ewentualnej refaktoryzacji.

Użytkownik

Istnieje również cały zbiór metryk zajmujący się oceną jakości oprogramowania od strony użytkownika. Jego zasadnicza część znajduje się w normie ISO/IEC 9126. Norma ta nie definiuje ściśle wskaźników, jest jedynie zbiorem wskazówek na temat aspektów działania oprogramowania, które mogą być mierzone.

Jednym z takich aspektów jest jakość użytkowa. Do najważniejszych atrybutów tej cechy należy min. efektywność, produktywność, bezpieczeństwo (tu w kontekście odporności na użytkownika) czy satysfakcja klienta z używanego oprogramowania.

Ze względu na subiektywność dane te nie mogą być zbierane automatycznie, tak jak metryki kodu. Ich akwizycja może się odbywać w formie okresowych ankiet wypełnianych przez użytkowników.

Procedury

Ostatnim z aspektów oceny jakości oprogramowania może być ocena procesu jego tworzenia. Tu jedną z głównych wytycznych jest norma ISO 9001. Ta norma określa w jaki sposób powinno się budować proces produkcji np. oprogramowania. Jest to o tyle ważna, że program pisany wedle ustalonego standardu ma większą szansę utrzymać stałe tempo rozwoju i zmieścić się w wymaganiach (czasowych, kosztowych, itp.).

Dodatkowo norma ta wymusza ścisłą współpracę z klientem i okresowe zbieranie od niego informacji zwrotnych (patrz punkt wyżej). Co ma duże znaczenie w przypadku wytwarzania oprogramowania pod konkretnego klienta, ponieważ pozwala dużo szybciej wychwytywać ewentualne problemy i rozwiązywać je gdy nie stanowią jeszcze zagrożenia dla projektu.