Wydział Informatyki PB

Inżynieria oprogramowania

Wykład 11:

Zapewnienie jakości oprogramowania i metryki oprogramowania

Marek Krętowski
pokój 206
e-mail: m.kretowski@pb.edu.pl
http://aragorn.pb.bialystok.pl/~mkret

Wersja 1.26

Jakość oprogramowania

- Zapewnienie jakości jest rozumiane jako zespół działań zmierzających do wytworzenia u wszystkich zainteresowanych przekonania, że dostarczony produkt właściwie realizuje swoje funkcje i odpowiada aktualnym wymaganiom i standardom
- Problem jakości, oprócz mierzalnych czynników technicznych, włącza dużą liczbę niemierzalnych obiektywnie czynników psychologicznych
- Podstawą obiektywnych wniosków co do jakości oprogramowania są pomiary pewnych parametrów użytkowych (niezawodności, szybkości, itd.) w realnym środowisku, np. przy użyciu metod statystycznych
- Obiektywne pomiary cech produktów programistycznych są utrudnione lub niemożliwe
- Jakość gotowych produktów programistycznych jest bardzo trudna do zmierzenia ze względu na ich złożoność, wieloaspektowość oraz niską przewidywalności wszystkich aspektów ich zastosowań w długim czasie

Zakres zapewnienia jakości

- Modele i miary służące ocenie kosztu i nakładu pracy
- Modele i miary wydajności ludzi
- Gromadzenie danych
- Modele i miary jakości
- Modele niezawodności
- Ocena i modelowanie wydajności oprogramowania
- Miary struktury i złożoności
- Ocena dojrzałości technologicznej
- Zarządzanie z wykorzystaniem metryk
- Ocena metod i narzędzi

Celem ostatecznym jest poprawa jakości oprogramowania

Trudności z oceną jakości oprogramowania

- Oceny jakości najczęściej muszą być znane zanim powstanie gotowy, działający produkt, co wyklucza zastosowanie obiektywnych metod pomiarowych
- Wiele czynników składających się na jakość produktu jest niemierzalna
- Produkty programistyczne mogą działać w różnych zastosowaniach, o różnej skali, przez co pomiary jakości mogą okazać się nieadekwatne przy zmianie skali (np. zwiększonej liczbie danych lub użytkowników), w innym środowisku, ...
- Pomiary mogą okazać się bardzo kosztowne, czasochłonne lub niewykonalne (z powodu niemożliwości stworzenia środowiska pomiarowego przed wdrożeniem);
- Nie ma zgody co do tego, w jaki sposób zmierzone cechy danego produktu składają się na syntetyczny wskaźnik jego jakości
- Oceny jakości produktów programistycznych są skazane na metody spekulacyjne, oparte na uproszczeniach oraz dodatkowych założeniach, algorytmach, wzorach i heurystykach

Zarządzanie przez jakość

- Koncepcja zarządzania przez jakość (TQM) wymyślona przez Japończyka Eiji Toyodę dla potrzeb naprawy japońskiego przemysłu motoryzacyjnego (~1950)

„Jakość jest najważniejszym kryterium oceny przydatności produktów dla klienta, a to właśnie klient umożliwia funkcjonowanie wytwórcy tych produktów”

- W związku z tym, że to klient stanowi o rentowności przedsiębiorstwa, to należy tak sterować wszystkimi fazami procesu produkcyjnego wyrobu, aby klient był zadowolony z jakości tego wyrobu
- Wniosek: producent wytwarzający produkty kiepskie powinien wypaść z rynku
- TQM została sformalizowana przez Amerykanów (W.E.Deming, P.Crosby, J.M.Juran, A.V.Feigenbaum), Japończyków (E.Toyoda, M.Imai, K.Ishikawa) i Brytyjczyka J.Oaklanda

Pomiary oprogramowania

- Pomiar (ang. *measurement*) jest to proces, w którym atrybutom świata rzeczywistego przydzielane są liczby lub symbole w taki sposób, aby charakteryzować te atrybuty według jasno określonych zasad
- Jednostki przydzielane atrybutom nazywane są miarą danego atrybutu
- Metryka (ang. *metric*) jest to proponowana (postulowana) miara; nie zawsze charakteryzuje ona w sposób obiektywny dany atrybut (np. ilość linii kodu jest metryką charakteryzującą atrybut “długość programu źródłowego”, ale nie jest miarą ani złożoności ani rozmiaru programu (choć występuje w tej roli)

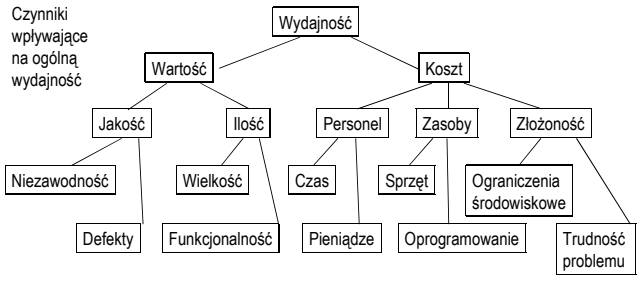
Pomiarom podlega:

- Proces: każde określone działanie w ramach projektu, wytwarzania lub eksploatacji oprogramowania.
- Produkt: każdy przedmiot powstały w wyniku procesu (kod źródłowy, specyfikację projektową, udokumentowaną modyfikację, plan testów, dokumentację, ...)
- Zasób: każdy element niezbędny do realizacji procesu (osoby, narzędzia, metody wytwarzania, ...)

Elementy pomiaru oprogramowania

Obiekty	Atrybuty bezpośrednio mierzalne	Wskaźniki syntetyczne
Produkty		
Specyfikacje	rozmiar, ponowne użycie, modularność, nadmiarowość, funkcjonalność, ...	zrozumiałość, pielęgnacyjność, ...
Projekty	rozmiar, ponowne użycie, modularność, spójność, funkcjonalność,...	jakość, złożoność, pielęgnacyjność, ...
Kod	rozmiar, ponowne użycie, modularność, spójność, złożoność, strukturalność, ...	niezawodność, używalność, pielęgnacyjność, ...
Dane testowe	rozmiar, poziom pokrycia,...	jakość,...
Procesy		
Specyfikacja arch.	czas, nakład pracy, liczba zmian wymagań, ...	jakość, koszt, stabilność, ...
Projekt szczegółowe	czas, nakład pracy, liczba usterek w specyfik.	koszt, opłacalność, ...
Testowanie	czas, nakład pracy, liczba błędów kodu, ...	koszt, opłacalność, stabilność, ...
Zasoby		
Personel	wiek, cena, ...	wydajność, doświadczenie, inteligencja, ...
Zespoły	wielkość, poziom komunikacji, struktura,...	wydajność, jakość, ...
Oprogramowanie	cena, wielkość, ...	używalność, niezawodność, ...
Sprzęt	cena, szybkość, wielkość pamięci	niezawodność, ...
Biura	wielkość, temperatura, oświetlenie,...	wygoda, jakość,...
....	....	....

Modele i miary wydajności



Mylące, wręcz niebezpieczne jest zastępowanie wielu miar jedną miarą, np. długością wyprodukowanego kodu

Techniki szacowania nakładów pracy

- Modele algorytmiczne - opis przedsięwzięcia za pomocą charakteryzujących go atrybutów (liczbowych) i na tej bazie wyliczenie (np. prosta formuła matematyczna) nakładów
- Ocena przez ekspertów - bazują na własnym doświadczeniu zdobytym przy realizacji innych projektów
- Ocena przez analogię - przy założeniu gromadzenia informacji o uprzednio wykonanych projektach; wyszukuje się podobne systemy poprzednio zrealizowane i wykorzystuje nakłady poniesione na ich stworzenie
- Wycena dla wygranej (ang. *pricing to win*) - na podstawie oceny możliwości klienta oraz przewidywanych działań konkurencji; opiera się na prawie Parkinsona - przedsięwzięcia są wykonywane przy założonych kosztach (jakie by one nie były)
- Szacowanie wstępujące - dzieli się przedsięwzięcie na mniejsze zadania, które łatwiej oszacować; koszt całości jest ich sumą (ew. z narzutem na integrację)

Modele algorytmiczne szacowania nakładów

- Większość modeli przyjmuje, że jednym z istotnych atrybutów jest rozmiar systemu, mierzony np. liczbą instrukcji (linii) kodu (ang. *Lines Of Code* - LOC), KLOC (ang. *Kilo*-LOC), KDSI (ang. *Kilo (thousand) of Delivered Source code Instructions*)
- Specjalizacja metryk w kierunku konkretnej klasy oprogramowania powinna dawać lepsze i bardziej adekwatne oceny niż metryki uniwersalne
- Najlepiej jest stosować zestawy metryk, co pozwala zmniejszyć błędy pomiaru
- Metryki powinny być wykorzystywane jako metody wspomagania ekspertów (stosowane formalistycznie mogą być groźne)
- Pomimo pochodzenia empirycznego, metryki skutecznie pomagają w szybkiej i mniej subiektywnej ocenie oprogramowania
- Żadna metoda przewidywania kosztów nie jest doskonała i jest oparta na szeregu arbitralnych założeń; niemniej dla celów planowania tego rodzaju metody stają się koniecznością

Nauka o programach Halsteada
(Halstead's software science)

- Halstead zdefiniował szereg metryk złożoności w oparciu o elementy składniowe programu - kodu (operandy i operatory)
- W programie P mamy:
  n1 (n2) różnych operatorów (operandów);
  słownik P zawiera n=n1+n2 elementów;
  N1 (N2) - oznacza całkowitą liczbę wystąpień operatorów (operandów);
  wielkość P wynosi N=N1+N2;
- Wysiłek E potrzebny do wytworzenia P wynosi (elementarnych jednostek):

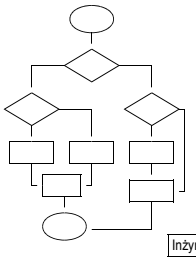
$$E = (n1*N2*N*\log n) / (2*n2)$$

- Czas potrzebny do wytworzenia P wynosi:

$$T = E / 18s$$

Liczba cyklomatyczna
(McCabe's cyclomatic number)

- Zaproponowana przez McCabe'a; odnosi się do schematu blokowego programu i jest równa liczbie niezależnych dróg w takim schemacie (liczba decyzji w programie +1)
- Krytykowana m.in. za nieuwzględnienie złożoności przepływów danych lub złożoności programów niestrukturalnych



Program P jest reprezentowany przez graf G, gdzie:
e - liczba krawędzi
n - liczba węzłów
d - liczba węzłów decyzyjnych

$$v(P)=e-n+2$$
$$v(P)=d+1$$

v(P) - liczba niezależnych ścieżek w G

Metoda COnstructive COst MOdel (COCOMO 81)

Powstała w oparciu o dane z rzeczywistych projektów, realizowanych tradycyjnie (np. bez narzędzi CASE) w oparciu o model kaskadowy, autor Boehm, 1981 r.

Wymaga oszacowania liczby instrukcji, z których będzie składał się system (co może być tak samo trudne jak oszacowanie nakładów)

Wyróżnia się trzy klasy przedsięwzięć:

- **Łatwe** (ang. *organic*) - wykonywane przez stosunkowo małe zespoły, złożone z osób o podobnych wysokich kwalifikacjach; dziedzina oraz wykorzystywane metody i narzędzia są dobrze znane
- **Pośrednie** (ang. *semi-detached*) - członkowie zespołu różnią się stopniem zaawansowania; pewne aspekty dziedziny problemu lub wykorzystywane narzędzia nie są dobrze znane
- **Trudne** (osadzone, ang. *embedded)* - przedsięwzięcia realizujące systemy o bardzo złożonych wymaganiach; dziedzina problemu, stosowane narzędzia i metody mogą być w dużej mierze nieznane; członkowie zespołu nie mają doświadczenia w realizacji podobnych zadań

Podstawowy model COCOMO

- Podstawowy wzór dla oszacowania nakładów w osobomiesiącach (zależność wykładnicza):

$$\text{Nakład} = A * K^{b}$$

- K oznacza rozmiar kodu źródłowego mierzony w tysiącach linii (KDSI); nie obejmuje kodu, który nie został wykorzystany w systemie
- Wartości stałych A i b zależą od klasy, do której zaliczono przedsięwzięcie
- Dla niewielkich przedsięwzięć są to zależności bliskie liniowym; wzrost jest szczególnie szybki dla przedsięwzięć trudnych (duży rozmiar kodu)

Przedsięwzięcie łatwe:

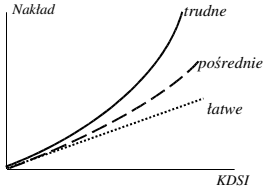
$$\text{Nakład} = \mathbf{2.4} * K^{\mathbf{1.05}}$$

Przedsięwzięcie pośrednie:

$$\text{Nakład} = \mathbf{3.0} * K^{\mathbf{1.12}}$$

Przedsięwzięcie trudne:

$$\text{Nakład} = \mathbf{3.6} * K^{\mathbf{1.20}}$$



Model COCOMO (2)

- Znając nakład można oszacować czas (w miesiącach) realizacji, z czego wynika przybliżona wielkość zespołu
- Z obserwacji wiadomo, że dla każdego przedsięwzięcia istnieje optymalna liczba członków zespołu wykonawców
- Zwiększenie tej liczby może nawet wydłużyć czas realizacji

Przedsięwzięcie łatwe:

$$\text{Czas} = 2.5 * \text{Nakład}^{0.32}$$

Przedsięwzięcie pośrednie:

$$\text{Czas} = 2.5 * \text{Nakład}^{0.35}$$

Przedsięwzięcie trudne:

$$\text{Czas} = 2.5 * \text{Nakład}^{0.38}$$

Otrzymane w ten sposób oszacowania powinny być skorygowane przy pomocy tzw. **czynników modyfikujących**

Atrybuty przedsięwzięcia brane pod uwagę:

- wymagania wobec niezawodności systemu
- rozmiar bazy danych w stosunku do rozmiaru kodu
- złożoność systemu (złożoność struktur danych i algorytmów, komunikacja z innymi systemami, stosowanie obl. równoległych)
- wymagania co do wydajności systemu
- ograniczenia pamięci
- zmienność sprzętu i oprogramowania systemowego tworzącego środowisko pracy systemu

Wady metody COCOMO

- Liczba linii kodu jest znana dokładnie dopiero wtedy, gdy system jest napisany a szacunki mogą być (i zwykle są) obarczone bardzo poważnym błędem (niekiedy ponad 100%)
- Określenie "linii kodu źródłowego" inaczej wygląda dla każdego języka programowania (np. 1 linia w Smalltalk'u jest równoważna 10-ciu linii w C; dla języków 4GL ten stosunek może być nawet 1000:1)
- Koncepcja oparta na liniach kodu źródłowego natomiast całkowicie nie przystaje do współczesnych narzędzi programistycznych, np. opartych o programowanie wizyjne
- Opiera się tylko na długości kodu i nie bierze w ogóle pod uwagę funkcjonalności ani złożoności produktu
- Kwalifikacja przedsięwzięcia do predefiniowanych klas oraz dobór czynników modyfikujących jest trudny, a ewentualne błędy mogą prowadzić do znacznych rozbieżności pomiędzy oczekiwanym i rzeczywistym kosztem

COCOMO II

- Uwspółcześniona wersja COCOMO opublikowana w 2000 r.
- Zakłada, że pracochłonność przedsięwzięcia zależy od wielu czynników, które ujawniają się w czasie realizacji
  - opracowanie architektury jest kluczowym momentem
- Dwa różne schematy obliczeń:
  - uproszczony model (wczesnego projektu) (ang. *early design model*)
  - dokładny model (gotowej architektury) (ang. *post-architecture model*)
- Postać formuły wyliczania pracochłonności w funkcji rozmiaru jest analogiczna, różne są tylko uwzględniane czynniki

$$\textbf{\textit{Pracochłonność} = A * (rozmiar)}^{B} * \textbf{\textit{Korekta}}$$

- współczynnik A jest stały (A=2,94)
- parametr B decyduje o skalowalności $B = 0{,}91 + 0{,}01 * \sum Si$ (i=1..5)
- *Korekta* = iloczyn czynników: 7 w uproszczonym, 17 w pełnym modelu

COCOMO II (2)

- Współczynniki *Si* są ocenami następujących cech projektu :
  - innowacyjność projektu
  - elastyczność wymagań
  - jakość planowania i zarządzania ryzykiem
  - zgodność celów udziałowców projektu
  - dojrzałość procesu wytwórczego
- W efekcie *B* może zmieniać się w zakresie od 0,91 do 1,23
- Pełna wersja modelu COCOMO II jest bardzo złożona
  - zawiera ok. 160 parametrów umożliwiających obliczanie rozmiaru programu (uwzględniając np. kod generowany automatycznie, dostosowywany z innych projektów) oraz reguły wyliczania współczynników korygujących

Analiza Punktów Funkcyjnych

- Metoda analizy punktów funkcyjnych (ang. *function points* - FP), została opracowana przez Albrechta (początek lat 80-tych); łączy własności metody badającej rozmiar projektu programu z możliwościami metody badającej produkt programowy (jego funkcjonalność)
- Najpierw ocenia się zobiektywizowaną złożoność problemu, wynikającą ze złożoności danych i algorytmów przetwarzania realizowanych przez aplikację
  - koncentruje się na perspektywie użytkowników; nie bierze się pod uwagę technologii implementacji
  - liczbę pierwotnych (nieskorygowanych) punktów funkcyjnych (ang. *Unadjusted FP*) wylicza się korzystając z 5 kategorii elementów składowych modelu
  - każdy zidentyfikowany element modelu przetwarzania jest oceniany w 3 stopniowej skali opisowej
- W drugim etapie uwzględnia się dodatkowe czynniki wpływające na złożoność przedsięwzięcia

$$FP = UFP * VAF$$

Kategorie elementów modelu przetwarzania

- Zbiory wewnętrzne – modelują dane przechowywane i utrzymywane przez oceniane oprogramowanie
  - zbiór odpowiada grupie logicznie powiązanych danych opisujących jakiś istotny aspekt przetwarzania
- Zbiory zewnętrzne – modelują współpracę z innymi systemami
  - zbiór reprezentuje logicznie powiązane dane otrzymywane z innego systemu
- Wejścia użytkownika (zewnętrzne) - modelują elementarne operacje przetwarzające dane dostarczane z zewnątrz (przez użytk. lub inne systemy)
  - operacje opisują działania istotne z punktu widzenia użytkownika => efekt zmiana zawartości zbiorów wewnętrznych lub zmiana zachowania systemu
- Wyjścia użytkownika (zewnętrzne) - modelują elementarne operacje przetwarzające dane i przekazujące wyniki do otoczenia
  - działanie musi obejmować coś więcej niż tylko proste odczytanie danych
- Zapytania – modelują operacje prostego wyszukania i przedstawienia danych zgromadzonych w zbiorach aplikacji (brak obliczeń i zmian wartości w zbiorach)

Ocena złożoności elementów modelu

- Reguły oceny złożoności elementów modelu są ściśle określone; brane są pod uwagę następujące charakterystyki:
  - liczba typów rekordu zbioru (liczba różnych encji reprezentowanych przez zbiór)
  - liczba pól zbioru (liczba różnych atrybutów występujących w encjach zbioru; uwzględnia się też atrybuty łączące zbiory – klucze obce)
  - liczba wykorzystywanych zbiorów (zarówno wewnętrznych jak i zewnętrznych w operacji)

Czynnik złożoności	Prosty	Średni	Złożony
Wejścia użytkownika	3	4	6
Wyjścia użytkownika	4	5	7
Zbiory danych wewnętrzne	7	10	15
Zbiory danych zewnętrzne	5	7	10
Zapytania zewnętrzne	3	4	6

Wagi przypisywane poszczególnym elementom modelu zależą od ich złożoności

Pierwotne i skorygowane punkty funkcyjne

- Złożoność oprogramowania (pierwotna liczba punktów funkcyjnych) – ważona suma ilości elementów modelu przetwarzania
  - wagi wynikają ze złożoności poszczególnych elementów modelu
- Aby otrzymać finalną (skorygowaną) liczbę punktów funkcyjnych należy uwzględnić dodatkowe czynniki złożoności przedsięwzięcia:
  - pod uwagę branych jest 14 czynników korygujących i przypisuje się im wagi (skala 6-stopniowa: od 0 - „brak wpływu" do 5 -„bardzo duży wpływ")
  - suma ich ocen oraz odpowiednie wyskalowanie pozwalają wyliczyć współczynnik korekcyjny (ang. *Value Adjustment Factor*)
  - pozwala to skorygować liczbę punktów funkcyjnych o 35% w obie strony
- Można pokusić się o przeliczenie otrzymanej złożoności projektu na pracochłonność albo koszt wykonania
  - dostępne są uśrednione tabele i wykresy (np. pracochłonności) dla różnych języków
  - należy jednak być ostrożnym; zaleca się wykorzystanie własnych danych historycznych

Czynniki korygujące

- występowanie urządzeń komunikacyjnych
- rozproszenie przetwarzania
- długość czasu oczekiwania na odpowiedź systemu
- stopień obciążenia sprzętu istniejącego
- częstotliwość wykonywania dużych transakcji
- wprowadzanie danych w trybie bezpośrednim
- wydajność użytkownika końcowego
- aktualizacja danych w trybie bezpośrednim
- złożoność przetwarzania danych
- możliwość ponownego użycia programów w innych zastosowaniach
- łatwość instalacji
- łatwość obsługi systemu
- rozproszenie terytorialne
- łatwość wprowadzania zmian - pielęgnowania systemu

Wykorzystanie punktów funkcyjnych

- Ocena złożoności realizacji systemów
- Audyt projektów
- Wybór systemów informatycznych funkcjonujących w przedsiębiorstwie do reinżynierii (wg. koszt utrzymania/FPs)
- Szacowanie liczby testów
- Ocena jakości pracy i wydajności zespołów ludzkich
- Ocena stopnia zmian, wprowadzanych przez użytkownika na poszczególnych etapach budowy systemu informatycznego
- Prognozowanie kosztów pielęgnacji i rozwoju systemów
- Porównanie i ocena różnych ofert dostawców oprogramowania pod kątem merytorycznym i kosztowym

1 FP ≈ 125 instrukcji w C

10 FPs - typowy mały program tworzony samodzielnie przez klienta (1 m-c)

100 FPs - większość popularnych aplikacji; wartość typowa dla aplikacji tworzonych przez klienta samodzielnie (6 m-cy)

1,000 FPs - komercyjne aplikacje w MS Windows, małe aplikacje klient-serwer (10 osób, ponad 12 m-cy)

10,000 FPs - systemy (100 osób, ponad 18 m-cy)

100,000 FPs - MS Windows'95, MVS, systemy militarne

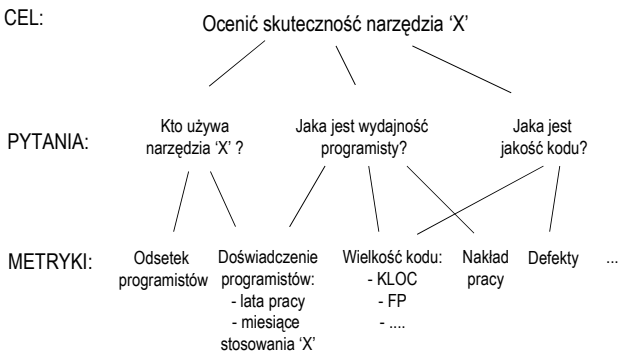
Punkty aplikacyjne (obiektowe)

- Alternatywna metoda w stosunku do punktów funkcyjnych zaproponowana na początku lat 90-tych przez Bankera i współpracowników
- Wykorzystywana do języków programowania baz danych, 4GL
  - obiektowe w nazwie nie ma tutaj związku z programowaniem obiektowym
- Liczba punktów obiektowych jest ważoną sumą:
  - liczby różnych formatek ekranowych (prosty ekran - 1 punkt, średnio złożony - 2 punkty, bardzo złożony - 3 punkty)
  - liczba generowanych raportów (prosty raport - 2 punkty, średnio skomplikowany - 5 punktów, a potencjalnie najtrudniejsze do utworzenia - 8 p.)
  - liczba modułów w językach 3GL (takich jak C++; Java) , które należy utworzyć w celu uzupełnienia kodu 4GL - każdy moduł 10 punktów
- Zaletą punktów obiektowych w porównaniu z punktami funkcyjnymi jest to, że łatwiej je obliczyć na podstawie specyfikacji oprogramowania wysokiego poziomu

Cel, Pytanie, Metryka
(Goal, Question, Metric - GQM)

- Wiele zamierzeń związanych z metrykami oprogramowania zawiodło, gdyż cele były zwykle słabo zdefiniowane; aby temu przeciwdziałać na Uniwersytecie Maryland (Vic Basili *et al.*) opracowano metodę GQM
- Trzy podstawowe kroki:
  - ustalenie celów w kategoriach *zamiaru*, *perspektywy* i *środowiska*
  - ponowne wypowiedzenie celów w terminach odpowiedzi na łatwe do "skwantyfikowania" pytania
  - określenie metryk i danych, które należy zgromadzić, aby udzielić odpowiedzi na postawione pytania
- Dzięki takiemu ułożeniu koncentrujemy się tylko na metrykach, które odnoszą się do interesującego nas celu
- Kilka pomiarów, aby odpowiedzieć na jedno pytanie; pojedynczy pomiar może mieć zastosowanie do kilku pytań

Przykład wykorzystania GQM



Dojrzałość procesów wytwórczych

Niedojrzałość

- ☹ Improwizacja podczas procesu wytwórczego
- ☹ Proces jest wyspecyfikowany, ale specyfikacja nie jest stosowana
- ☹ Doraźne reagowanie w sytuacji kryzysów
- ☹ Harmonogram i budżet są przekraczane
- ☹ Funkcjonalność jest stopniowo okrajana
- ☹ Jakość produktu jest niska
- ☹ Brak obiektywnych kryteriów oceny

Dojrzałość

- ☺ Zdolność do budowy oprogramowania jest cechą organizacji a nie personelu
- ☺ Proces jest zdefiniowany, znany i wykorzystywany
- ☺ Proces jest obserwowany i ulepszany
- ☺ Prace są planowane i monitorowane
- ☺ Role i odpowiedzialności są zdefiniowane
- ☺ Obiektywna, ilościowa ocena

Model dojrzałości procesu wytwórczego

- Model dojrzałości procesu wytwórczego (ang. *Capability Maturity Model* - CMM) został opracowany w latach 80-tych przez amerykański Instytut Inżynierii Oprogramowania na zlecenie rządu USA
- Wykorzystywany był w procedurach oceny (klasyfikacji) potencjalnych wykonawców oprogramowania dla Departamentu Obrony
- Opiera się w znacznym stopniu na koncepcjach TQM (ang. *Total Quality Management*)
- Ocenia wiele różnych atrybutów wytwarzania oprogramowania, obejmujących użycie narzędzi i standardowych praktyk
- Model ten szybko zyskał szeroką akceptację jako wzorzec do usprawniania procesu programowania
- Wywarł znaczny wpływ na uświadomienie znaczenia miar i ich stosowanie, gdyż w CMM miary są uważane za ważne dla osiągnięcia wyższych poziomów usprawnienia procesu

Poziomy dojrzałości wytwórców

Wyróżniono 5 poziomów dojrzałości wytwórców (poczynając od p. najniższego):

- początkowy (1) - proces chaotyczny, nie istnieją żadne standardy, decyzje podejmowane *ad hoc*; może dotyczyć nawet firm o dobrym zaawansowaniu technicznym
- powtarzalny (2) - proces zindywidualizowany;przedsięwzięcia wykonywane w podobny sposób (standardy *de facto*); standardy nie są udokumentowane i nie istnieją ścisłe procedury kontroli
- zdefiniowany (3) - proces zinstytucjonalizowany; standardy postępowania są zdefiniowane, sformalizowane i ich stosowanie jest kontrolowane
- zarządzany (4) - proces nie tylko podlega kontroli ale jest również mierzony w sposób ilościowy; informacje zwrotne wykorzystywane są do sterowania procesem
- optymalizujący (5) - standardy są ciągle uaktualniane; informacje zwrotne wpływają na ulepszenie procesu; standardy zawierają elementy pozwalające na dostrojenie procesu do aktualnych potrzeb
- Początkowo niewiele firm uzyskiwało poziom 3-ci, umożliwiający dostarczanie oprogr. dla Dep. Obrony; tylko IBM w zakresie oprogr. promu kosmicznego dla NASA uzyskał poziom 5-ty

Przygotowano na podstawie:

- *Inżynieria oprogramowania w projekcie informatycznym, red. J. Górski, Mikom 2000.*
- *Wprowadzenie do inżynierii oprogramowania*, K. Subieta, Wydawnictwo PJWSTK, 2002.
- *Inżynieria oprogramowania*, K. Sacha, PWN, 2010.