JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 4 (2017) 92-98
Wysłane: 2016-12-14
Przyjęte: 2016-12-15

# Pojęcie, modele i metryki jakości oprogramowania – przegląd

Yuliia Horobets*, Marek Miłosz
Politechnika Lubelska, Instytut Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

**Streszczenie**. W artykule przedstawiono analiza istniejących modeli i metryk pomiaru i oceny jakości oprogramowania. Do porównania zostały wybrane następujące modele jakości oprogramowania: McCalla, Boehma, Boeinga, FURPS, Dormey'a, ISO/IEC 9126, ISO/IEC 25000:2005 oraz metryki oprogramowania: SLOC, McCabe'a, Halsteada.

**Słowa kluczowe:** cykl życia; metryki jakości oprogramowania; modele jakości oprogramowania

*Autor do korespondencji.

Adres e-mail: Yuliia.horobets@gmail.com

# The concept, models and metrics of software quality – an overview

Yuliia Horobets*, Marek Miłosz
Institute of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

**Abstract.** The article presents analysis of existing models and metrics measurement and software quality assessment. For comparison were chosen following models of software quality: McCall, Boehm, Boeing, FURPS, Dormey, ISO / IEC 9126, ISO / IEC 25000: 2005 and software metrics: SLOC, McCabe, Halstead.

**Keywords:** software lifecycle; software quality metrics; models of software quality

*Corresponding author.

E-mail address: Yuliia.horobets@gmail.com

## 1. Wstęp

Celem artykułu jest próba zdefiniowania pojęcia jakości oprogramowania oraz przedstawienie przeglądu modeli i metryk, związanych z jakością oprogramowania. Modele i metryki jakości oprogramowania znajdują zastosowanie na różnych etapach cyklu życia oprogramowania: od wymagań po jakość kodu.

Metodą badawczą zastosowaną w niniejszym artykule jest systematyczny przegląd literatury przedmiotu i jej krytyczna ocena połączona z wyciąganiem wniosków.

Przegląd literatury był wykonywany w następujących krokach:

- badania literaturowe z zakresu inżynierii oprogramowania w celu ustalenia aktualnego stanu i sposobów poprawy procesu oceny jakości oprogramowania;
- analiza istniejących modeli i metryk pomiaru jakości oprogramowania;
- analiza dotycząca różnych modeli i metryk oceny jakości oprogramowania.

W trakcie przeglądu literaturowego wytypowano do opisu i analizy następujące modele jakości oprogramowania: McCalla, Boehma, Boeinga, FURPS, ISO/IEC 9126, ISO/IEC 25000:2005, SQuaRE i Dromeya oraz statyczne metryki oprogramowania (kodu): SLOC, McCabe'a i Halsteada. Pierwsza grupa (modele) ocenia jakość oprogramowania z punktu widzenia użytkownika końcowego, a druga z punktu technicznego – czyli programistów tworzących kod programu.

Modele i metryki dotyczą bowiem różnych etapów w cyklu rozwoju oprogramowania.

## 2. Pojęcie jakości oprogramowania

Pojęcie jakości w odniesieniu do produktu informatycznego jest trudne do sformułowania i stanowi przedmiot wielu badań teoretycznych i empirycznych. Trudność ta wynika nie tylko ze specyfiki samego produktu, ale także z procesu jego tworzenia. Zagadnienie jakości oprogramowania jeszcze bardziej komplikuje tempo zmian, jakie obserwuje się u użytkowników oprogramowania. Zmiany te powodują, że dobry produkt szybko traci wymagany poziom jakości. Wychodząc od dwóch aksjomatów jakości: jakość nie jest wielkością bezpośrednio mierzalną i jakość jest właściwością zbiorczą, proponuje się, w analogii do inżynierii jakości wyrobów, zastosowanie odpowiednich metod wartościowania, pozwalających na automatyzację wielokrotnie wykonywanych obliczeń, ułatwiających prowadzenie badań i formułowanie ocen jakości [1].

Okazuje się, że zarówno definicje jakości, jak i oceny jakości produktu nie zależą od tego, czy produkt jest materialny czy niematerialny. W obu przypadkach jakość nie jest bezpośrednio mierzalna i jest pojęciem wymagającym bliższego określenia. Dla każdego produktu programowego istotne są aspekty jakości, ustalone wcześniej dla produktów materialnych jako znamiona jakości. Określają one jakość

produktów, takich jak: pojazdy, narzędzia pracy, urządzenia pomiarowe, czy sprzęt gospodarstwa domowego, i programu komputerowego. Cechy jakości natomiast, w zależności od rodzaju produktu, mogą mieć różny poziom ważności [2].

Podobnie wygląda zagadnienie opisywalności cech składających się na jakość produktu. Najczęściej są to właściwości zbiorcze, wymagające dookreślenia. Dookreślanie polega na przedstawianiu cząstkowych kryteriów oceny jakości. W przypadku potrzeby uzyskania oceny obiektywnej, konieczna jest jej postać ilościowa. O ile dla produktów materialnych istnieją fizyczne miary, które łatwo jest zastosować do mierzenia atrybutów jakości, o tyle w przypadku produktu informatycznego zagadnienie to jest trudniejsze. Proste miary fizyczne, jak liczba bajtów czy linii kodu źródłowego programu, przedstawiające aspekt materialny produktu informatycznego, mają niewielkie znaczenie dla oceny jego jakości. Istotne aspekty jakości wymagają poszukiwania miar, adekwatnych do celu badania. Stąd też zasadniczy element wyróżniający podejście do jakości produktów materialnych i produktu informatycznego uważa się aspekt mierzalności. Jednakże nie z powodu jego znaczenia, ponieważ w obu przypadkach jest tak samo ważny, ale trudności w doborze właściwych miar.

Jakość oprogramowania to ogół funkcjonalności i cech oprogramowania, które mają wpływ na jego zdolność do zaspokojenia stwierdzonych lub domniemanych potrzeb [3].

## 3. Modele jakości

Subiektywna interpretacja pojęcia jakości wymaga, aby dla otrzymania obiektywnego opisu przyjąć za podstawę uniwersalny model jakości. Celem analizy wybranych modeli, spośród wielu opisanych w literaturze przedmiotu, było określić najbardziej odpowiedni model dla zapewniania jakości oprogramowania na wszystkich etapach cyklu życia z kosztem najmniejszych zasobów.

W omawianych modelach (tab. 1) przyjęto zasadę określania składników jakości oprogramowania za pomocą złożonych charakterystyk oraz atrybutów zewnętrznych i wewnętrznych.

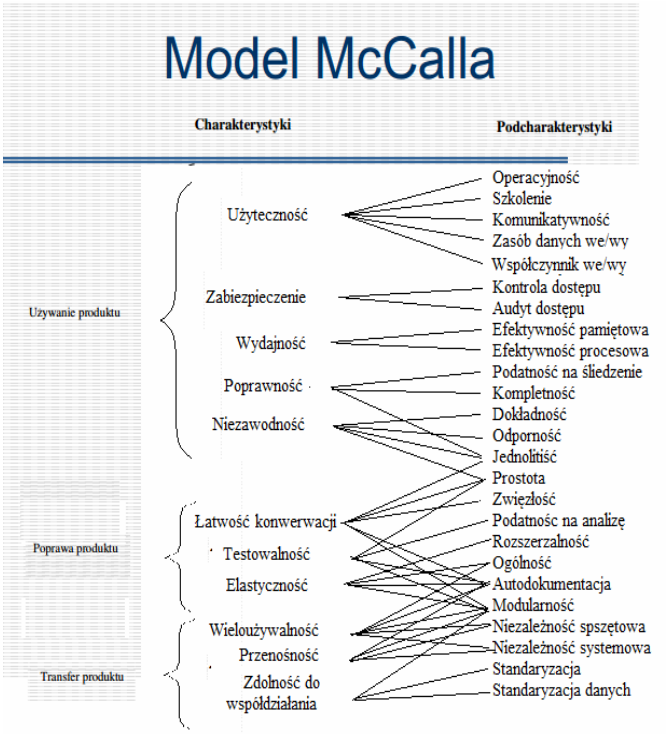
Tabela 1. Liczba charakterystyk modeli jakości oprogramowania

| **Model** | **Lichba charakterystyk/ podcharakterystyk** |
|---|---|
| McCalla(1977) | 11/25 |
| Boehma(1978) | 7/12 |
| Boeinga(1987) | 15/27 |
| FURPS(1987) | 5/26 |
| Dromeya | 4/7 |
| ISO/IEC 9126(2001) | 6/27 |
| ISO 2500 | 8/27 |

Pomiędzy nimi zachodzą relacje, które pozwalają określić wpływ atrybutów wewnętrznych na zewnętrzne, co w przypadku badania jakości oprogramowania może być wykorzystane np. do przewidywania jakości produktu końcowego na podstawie mierzonych atrybutów jakości wewnętrznej.

Model McCalla (1997) został opracowany na zlecenie Sił Zbrojonych USA, w celu poprawy jakości wytwarzanego oprogramowania [4].

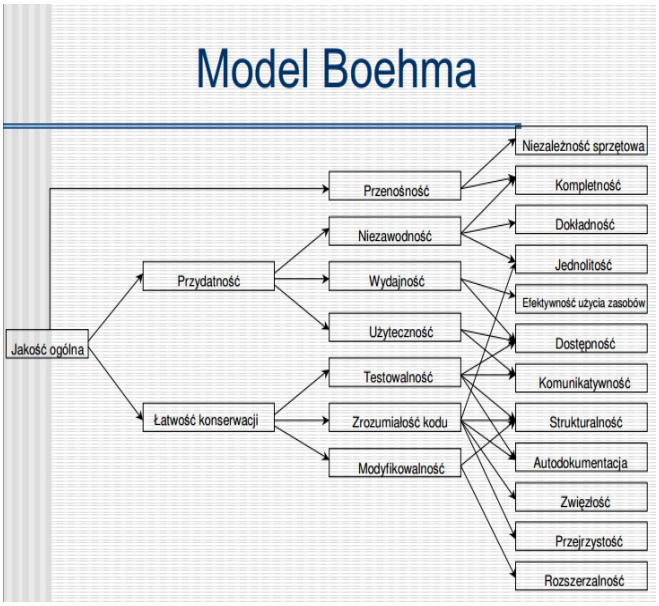
Oprogramowanie z perspektywy użytkownika jest badane ze względu na trzy wymiary: działanie, poprawianie oraz różne aspekty dotyczące zmian w środowisku. Są to tzw. zewnętrzne atrybuty jakości wysokiego poziomu, charakteryzujące się brakiem możliwości bezpośredniego mierzenia. Na tej podstawie wprowadzono 25 atrybutów niższego poziomu, który są nazwany kryteriami jakości. Jednak, że wiele z tych kryteriów jakości wymaga dalszej dekompozycji, w celu doprowadzenia do możliwości bezpośredniego ich mierzenia. Atrybuty mierzalne, nazywane metrykami jakości, znajdują się na czwartym poziomie tego drzewa (rys. 1).




Rys. 1. Model McCalla [4]

Ten model stosowany jest wyłącznie do oceny jakości produktu końcowego, ponieważ nie było zaproponowane podejście do stosowania tego modelu w innych etapach cyklu życia oprogramowania. Model trudno zastosowywać do nowoczesnych technologii i technik rozwoju oprogramowania. Metody ustalania właściwości wagi charakterystyk jakości są subiektywne, ponieważ są one oparte na wyborze konkretnych deweloperów. Wykorzystywane są liniowe metody, które nie biorą pod uwagę korelacji między charakterystykami. Główną wadą stosowania modelu McCall jest to, że każda metryka wpływa na ocenę innych czynników jakości.

W modelu Boehma [5] korzeń drzewa jest nazwany jakością ogólną. Składnikami tej jakości są użyteczność, pielęgnowalność oraz przenośność (rys. 2).




Rys. 2. Model Boehma [5]

Punkt widzenia użytkownika został określony przez użyteczność, pozostałe dwa składniki sformułowano natomiast z punktu widzenia wytwórcy zainteresowanego łatwością modyfikowania i rozumienia kodu. Na poziomie pierwszym pojawiają się atrybuty zewnętrzne, zwane składnikami pośrednimi. Poziom następny zawiera 12 składników, nazwanych podstawowymi, które odpowiadają atrybutom wewnętrznym. i tutaj, podobnie jak u McCalla, dopiero czwarty poziom wskazuje na atrybuty mierzalne.

Model Boeinga [6] składa się z dwóch poziomów: na pierwszym określono 15 atrybutów zewnętrznych, a na drugim 27 atrybutów wewnętrznych (rys 3).

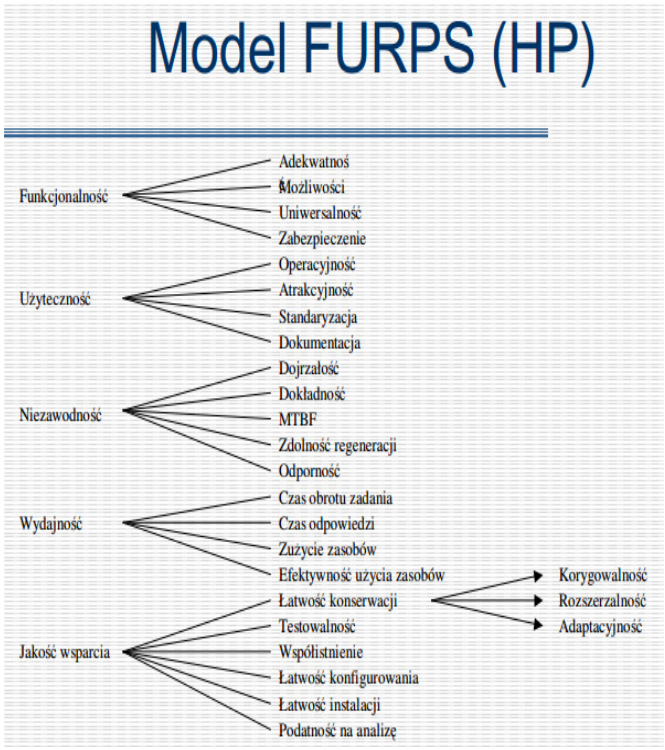
Nie wszystkie wewnętrzne atrybuty są bezpośrednio mierzalne, np. dokumentacja czy szkolenia, dlatego zastosowanie tego modelu wiąże się z ustaleniem mierzalnych atrybutów na kolejnym poziomie.

Model FURPS/FURPS+(1992) zaproponowany przez Roberta Grady'ego [7], składa się z pięciu charakterystyk (Funkcjonalność, Użyteczność, Niezawodność, Wydajność i Wsparcie).

FURPS zbudowany podobnie do modelu McCall i Boehm, ale w przeciwieństwie do nich składa się z dwóch warstw, pierwsza określa charakterystyki, a druga - związane z nimi atrybuty (rys. 4).

Ten model może być wykorzystany zarówno do oceny jakości, jak i specyfikacji wymagań funkcjinalnych i niefunkcjonalnych dla produktu.

Model FURPS + jest szeroko stosowany w tworzeniu oprogramowania oraz przy identyfikacji wymagań dla systemu w fazie rozwoju.

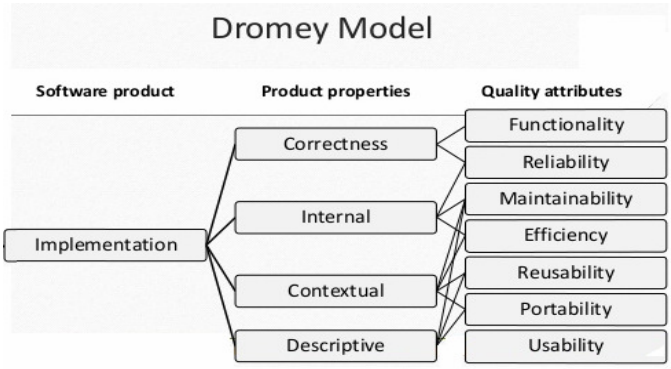



Rys. 3. Model Boeinga [6]


Rys. 4. Model FURPS [7]

Model zaproponowany przez Dromey'a w 1994 roku [8] i w 1998 koncentruje się na jakości produktu w oderwaniu od sposobu jego wytworzenia, co było pewnym przełomem w postrzeganiu jakości oprogramowania w latach 90'tych XX wieku. Model ma na celu zwiększenie wiedzy na temat relacji między atrybutami (charakterystyka) oraz atrybutów podrzędnych (sub-charakterystyka) jakości. Model Dromey'a definiuje dwie warstwy, atrybuty wysokiego szczebla

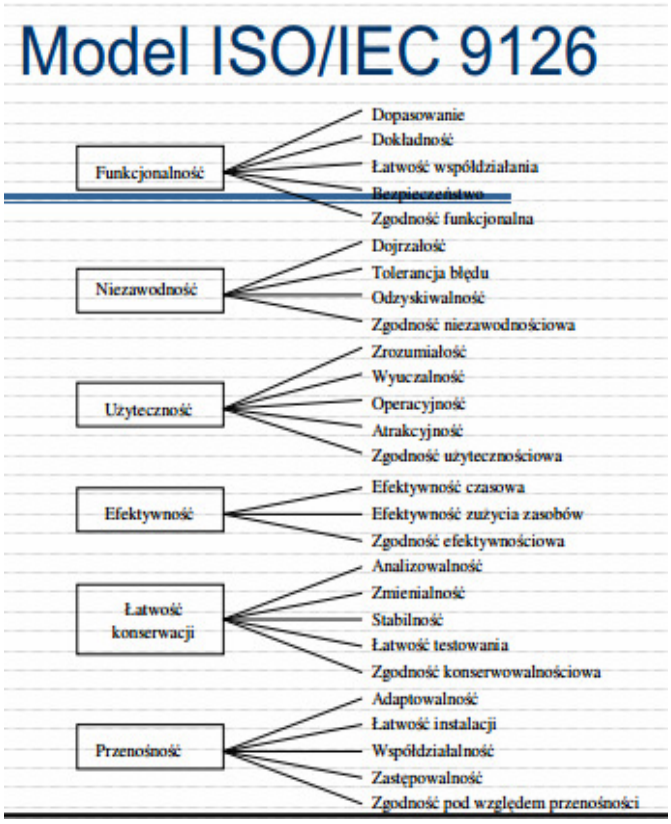
i atrybuty podległych (rys 5). Dlatego też model ten cierpi z powodu braku kryteriów mierzenia jakości oprogramowania.




Rys. 5. Model Dromey'a [8]

Jakość produktu, według autora, ma być jakością artefaktów związanych z tym produktem. Przez produkt w tym modelu nie był rozumiany wyłącznie produkt końcowy, ale także dokumentacja, projekt itd.

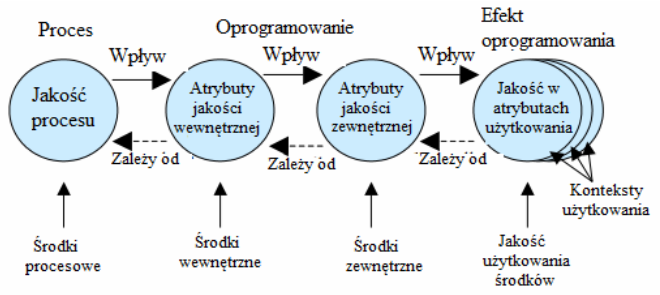
Model ISO/IEC 9126 [9], opracowany w 2001 roku, stanowi rozszerzenie modelu z roku 1991, opracowanego na bazie modelu McCalla oraz Boehma. Jego podstawą jest zestaw sześciu charakterystyk (rys.6). Wprowadzono trzy nowe pojęcia: jakość wewnętrzną, jakość zewnętrzną i jakość użytkową.




Rys. 6. Model ISO/IEC 9126 [9]

Dla każdej z nich określono odpowiednie zestawy spośród 27 podcharakterystyk. Zewnętrzne podcharakterystyki odzwierciedlają efekt użycia oprogramowania, stanowiącego istotną część systemu komputerowego i są wynikiem jakości atrybutów wewnętrznych tego oprogramowania. Dla jakości użytkowej określono cztery charakterystyki, podkreślając jej uzależnienie od jakości wewnętrznej i zewnętrznej.

Model zdefiniowany w normie pokazuje również różne aspekty jakości dyskutowane we wcześniejszych modelach, czyli jakość produktu, jakość techniczną produktu, a jakość procesu, tak jak to przedstawiono na rysunku 7.




Rys. 7. Jakość w cyklu życia systemu [9]

Każdy z tych obszarów ma odmienne znaczenie i odmiennego interesariusza. Wewnętrzna i zewnętrzna jakość definiowana jest jako suma następujących charakterystyk: funkcjonalność (*functionality*), niezawodność (*reliability*), użyteczność (*usability*), wydajność (*efficiency*), łatwość utrzymania (*maintainability*), przenośność (*portability*).

Jakość użyteczna jest definiowana jest jako suma następujących charakterystyk: przydatność (*effectiveness*), produktywność (*productivity*), bezpieczeństwo (*safety*) i zdolność do zaspokojenia potrzeb (*satisfaction*).

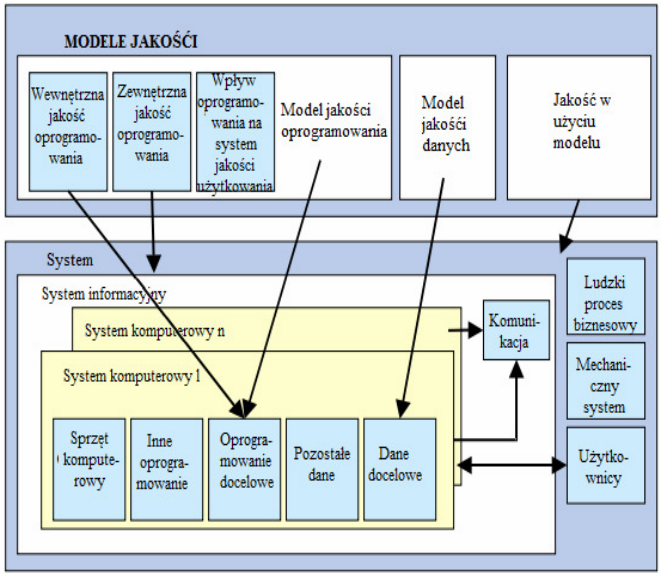
Te właściwości i atrybuty pozwalają systematycznie opisywać wymagania do oprogramowania, określając, które właściwości oprogramowaia dla tej charakterystyky chcą widzieć osoby na pewnym etapie cyklu życia.

Model ISO/IEC 25000:2005. Model nazywany Drugą Generacją Standardów Jakości - SQuaRE (*Software product Quality Requirements and Evaluation*) [10] publikowany w serii norm ISO/IEC 25000.

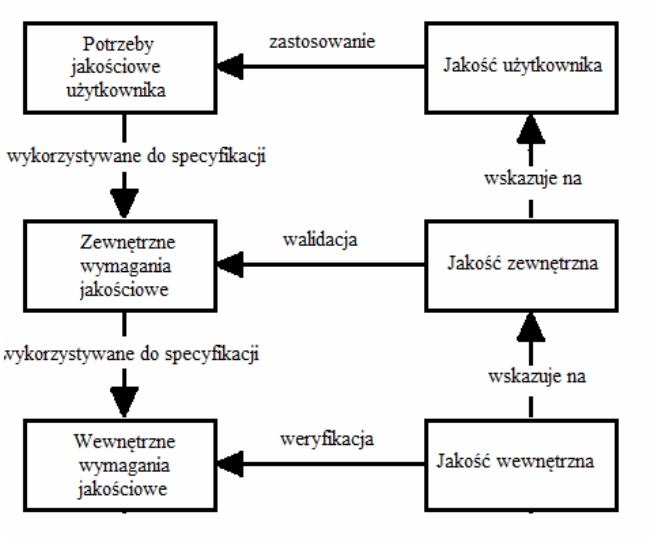
Na najwyższym poziomie modelu jakości oprogramowania pozostawiono podział na jakość wewnętrzną, zewnętrzną i użyteczną. Jakość wewnętrzna i zewnętrzna oprogramowania zdefiniowana jest analogicznie do definicji w ISO/IEC 9126:2001 [9], z tym że zmieniono charakterystyki wysokiego poziomu – zamiast sześciu w nowym modelu jest osiem: funkcjonalność (*functionality*), poufność (*security*), zgodność techniczna (*interoperability*), niezawodność (*reliability*), użyteczność (*usability*), wydajność (*efficiency*), łatwość utrzymania (*maintainability*) i przenośność (*portability*).

Jakość użyteczna oprogramowania, została zdefiniowana na nowo w modelu SQuaRE. Jej główne charakterystyki to: użyteczność (*usability in use*), zgodność z kontekstem (*context in use*), bezpieczeństwo (*safety in use*), poufność (*security in use*) i łatwość użycia (*adaptability in use*).

Modele jakości SQuaRE opisują zarówno oprogramowanie, jak i dane. Twórcy normy zauważają, że nie są to jedyne modele jakości dla bytu nazywanego „systemem", lecz norma odnosi się tylko do wspomnianych modeli. Na rysunkach 8 i 9 pokazano mapowanie modeli jakości SQuaRE na model systemu, oraz cykl życia jakości w ramach rozwoju produktu jakim jest oprogramowanie.




Rys. 8. Odniesienie modeli SQuaRE do jakości elementów systemu [10]




Rys. 9. Model jakości w cyklu życia oprogramowania [10]

Spośród wszystkich istniejących modeli ten model najlepiej nadaje się do oceny jakości oprogramowania na wszystkich etapach cyklu życia.

Analiza porównawcza właściwości różnych modeli jakości najlepiej złożyć w formie tabeli 2.

Tabela 2. Analiza porównawcza modeli jakości oprogramowania [11]

| Charakterystyka jakości \ Model | McCalla | Boehma | FURPS | ISO 9126 | ISO 25010 | Dromeya |
|---|---|---|---|---|---|---|
| Poprawność | | | | | + | |
| Niezawodność | + | + | + | + | + | + |
| Skuteczność | + | + | + | + | + | + |
| Elastyczność | + | | | + | + | |
| Funkcjonalność | | | + | + | + | + |
| Ergonomia | | + | | | | |
| Integralność | | | | | + | |
| Zgodność | + | | | + | + | |
| Wsparcie | + | + | + | + | + | + |
| Możliwość zmiany | | + | | + | + | |
| Produktywność | | | + | | | |
| Przenośność | + | + | | + | + | + |
| Stabilność | | | | + | + | + |
| Skalowanie | | | | | + | |
| Bezpieczeństwo | | | + | | + | |
| Dojrzałość | | | | | + | + |
| Testowalność | + | + | | + | + | |
| Zrozumiałość | | + | + | + | + | |
| Praktyczność | + | | + | + | + | + |

## 4. Metryki jakości oprogramowania

Metryki stanowią szybki i wygodny sposób oceny jakości oprogramowania. Znajomość wartości poszczególnych metryk i wielkości, które mierzą, pozwala oszacować złożoność systemu, koszt jego pielęgnacji czy elastyczność.

Metryka oprogramowania – miara pewnej własności oprogramowania lub jego specyfikacji [12].

Celem użycia metryk oprogramowania jest otrzymanie dokładnych wartości, które dotyczą wytwarzanych aplikacji. Metryki pozwalają na obiektywne spojrzenie na oprogramowanie i porównanie ze sobą poszczególnych jego elementów lub różnych produktów. Mierzenie jakości aplikacji oraz wydajności pracy bez danych liczbowych są bardzo trudne, a utrzymanie obiektywności jest niemal niemożliwe.

Obliczanie metryk jest istotnym ułatwieniem we wszystkich fazach procesu wytwarzania oprogramowania. Na tym etapie metryki są przydatne przede wszystkim dla klientów oraz projektantów. w fazie produkcji dużą rolę odgrywają metryki statyczne, pomagające w utrzymywaniu jakości kodu źródłowego. Korzystają z tego programiści, którzy dzięki metrykom mogą łatwo odnajdywać te miejsca w kodzie, które są potencjalnym źródłem nadmiernej złożoności, a co za tym idzie, powstających błędów. w fazie testów wykorzystywane są zarówno metryki statyczne, jak i dynamiczne, pozwalające na badanie przebiegu wykonania programu. Na tym etapie gromadzone są też dane liczbowe dotyczące wydajności czy niezawodności aplikacji. Wyniki testów są pożytecznym źródłem informacji dla wszystkich osób zaangażowanych w tym projekcie: kierowników, programistów i klientów [13].

W ocenie złożoności oprogramowania występują trzy główne grupy metryk:

- metryki rozmiaru;
- metryki logicznej struktury programu, czyli przepływu sterowania;
- metryki struktur danych.

Metryki rozmiaru są najbardziej proste, pozwalają na ocenę jakości kodu źródłowego i łączą się ściśle z analizą statyczną, dziedziną inżynierii zajmującą się badaniem struktury kodu źródłowego. Metryki te najbardziej przydatne są dla samych programistów i innych osób bezpośrednio zaangażowanych w proces powstawania oprogramowania. Pozwalają na bieżące śledzenie jakości kodu i zwrócenie uwagi na miejsca, które wymagają uproszczenia bądź szczególnie uważnego testowania.

Do tego typu metryk należy liczba linii kodu źródłowego - najprostsza metryka rozmiaru oprogramowania, oznaczana jako LOC (*lines of code*) lub SLOC (*source lines of code*). Wielkość ta daje ogólne pojęcie o skali programu, nie jest jednak wystarczająca do bardziej szczegółowych analiz. Wartość LOC jest bardziej zależna od użytego języka programowania – ten sam algorytm w języku wysokiego poziomu będzie miał wielokrotnie mniej linii kodu niż w asemblerze. Istotny jest też sposób zliczania linii kodu – ich liczba może się zdecydowanie zmieniać, zależnie od tego, czy wliczane są puste linie, komentarze czy linie zawierające jedynie nawiasy klamrowe bądź słowa kluczowe begin i end. Liczba linii z pominięciem wymienionych elementów jest określana jako eLOC (*effective lines of code*) [14].

Jedną z najbardziej podstawowych metryk złożoności jest CC (złożoność cyklomatyczna) zaproponowana przez Thomasa J. McCabe'a w 19760 [15]. Pierwotnie pomyślana była jako metryka dla programów strukturalnych, nadaje się jednak równie dobrze do programów obiektowych, dzięki czemu jest popularna także dzisiaj. Złożoność cyklomatyczna jest liczbą charakteryzującą funkcję lub metodę i odzwierciedlającą jej poziom skomplikowania.

Początkowo złożoność cyklomatyczna miała mierzyć liczbę niezależnych ścieżek przebiegu programu, co bezpośrednio przekłada się na łatwość jego przetestowania. Ponieważ skoki wstecz mogą spowodować nieskończony wzrost takich ścieżek, mierzy się liczbę prostych ścieżek bez uwzględniania cykli.

Złożoność cyklomatyczną dla danej funkcji można policzyć, mając do dyspozycji graf przebiegu programu. Wówczas wyraża się ona wzorem [15]:

$$v(G) = e - n + 2 \tag{1}$$

gdzie *e* - oznacza liczbę krawędzi, a *n* - liczbę wierzchołków w grafie. Inny wzór podaje prostą zależność CC od liczby węzłów, w których podejmujemy decyzje:

$$v(G) = d + 1 \tag{2}$$

gdzie *d* - to liczba węzłów decyzyjnych, w których podejmowana jest decyzja o charakterze binarnym (tak/nie). Taki wzór pozwala na szybkie wyznaczanie wartości CC – wystarcza tu zliczyć liczbę wystąpień słów kluczowych if, while, for czy case.

Niska wartość CC wskazuje na łatwość zrozumienia danej funkcji czy metody. Wartość powyżej 20 charakteryzuje złożony kod i wiąże się z dużym ryzykiem wystąpienia błędów.

Zaletami metryki CC są m.in. łatwość jej obliczenia i możliwość wskazania elementów aplikacji, które należy przeprojektować. Liczba rozgałęzień przepływu programu nie daje jednak pełnej informacji, ponieważ nie rozróżnia zagnieżdżonych i niezagnieżdżonych pętli oraz prostych instrukcji case, a także nie bierze pod uwagę skomplikowanych warunków w węzłach decyzyjnych. Istnieją modyfikacje definicji CC, które pozwalają zredukować lub wyeliminować te wady [15].

Złożoność Halsteada. Dana metryka dotycza rozmiaru programu pozwalają na zdefiniowanie bardziej skomplikowanych metryk złożoności. Wyróżnia się wśród nich [16]:

- trudność (*difficulty*), czyli podatność na błędy, proporcjonalna do liczby unikatowych operatorów ($n_1$), a także do stosunku pomiędzy wszystkimi ($N_2$) a unikatowymi operandami ($n_2$):

$$D = \left(\frac{n_1}{2}\right)\left(\frac{N_2}{n_2}\right) \tag{3}$$

- poziom programu (*program level*), im niższy, tym bardziej aplikacja jest podatna na błędy:

$$L = 1/D \tag{4}$$

- wysiłek (*effort*) potrzebny do zaimplementowania programu, proporcjonalny do trudności oraz objętości:

$$E = V \cdot L \tag{5}$$

- czas (*time*) potrzebny do zaimplementowania, wyrażony w sekundach – współczynnik 18 został wyznaczony eksperymentalnie i jest modyfikowalny indywidualnie dla danego programisty:

$$T = E/18 \tag{6}$$

- szacunkowa liczba błędów (*number of delivered bugs*) – wartość ta jest zależna od użytego języka programowania, jest na przykład wyższa dla aplikacji w Club C++:

$$B = E^{\frac{2}{3}} / 3000 \tag{7}$$

Niniejsze wartości mimo oczywistej niedokładności stanowią cenną informację podczas etapu testowania aplikacji – przykładowo wartość B w porównaniu z rzeczywistą liczbą znalezionych błędów może dać wskazówki na temat efektywności przeprowadzonych testów.

## 5. Wnioski

W podsumowaniu opisanych modeli jakości stwierdza się występowanie cech wspólnych: punkt widzenia, złożoność pojęcia jakości, złożoność charakterystyk, problem mierzalności atrybutów, występowanie zależności między charakterystykami i podcharakterystykami. Zauważone różnice dotyczą natomiast: rodzaju specyfikowanych charakterystyk, liczebności charakterystyk na różnych poziomach, kwalifikacji atrybutów na wewnętrzne i zewnętrzne. Modele te stanowią wzorce, które dostarczają istotną pomoc przy określaniu zestawu charakterystyk jakości.

Podsumowując można stwierdzić, że modele ISO 9126 i ISO 25010 są najlepiej przystosowane do kontroli jakości, które mają najwyższe kryteria jakości i pozwalają ich obliczyć na każdym etapie cyklu życia oprogramowania.

Analiza wartości metryk dla tworzonego kodu programu pozwala w szybki i zgrubny sposób ocenić jakość programu, a przede wszystkim jego projektu. Dzięki temu można łatwo stwierdzić istnienie błędów w strukturalizacji danych lub nieprawidłowej faktoryzacji klas.

## Literatura

[1] ISO 8402:1994 - Quality management and quality assurance.1994

[2] Поморова О.В. Сучасні проблеми оцінювання якості програмного забезпечення, Радіоелектронні і комп'ютерні системи. – 2013. - №5(64). – с. 319-327.

[3] ISO/IEC 25000:2014 - Systems and software engineering. 2014

[4] McCall J. A. i inni, Factors in Software Quality: Metric Data Collection and Validation. Final Technical Report. Vol. 1, National Technical Information Service, Springfield 1977.

[5] Boehm B.W, Characteristics of Software Quality, TRW Series of Software Technology, Amsterdam, North Holland 1978

[6] Deutsch M.S., Willis R.R.:Software Quality Engineering: A Total Technical and Management Approach, Prentice-Hall 1988

[7] Grady R.B.: Practical Software Metrics for Project Management and Process Improvement. Prentice Hall, Englewood Cliff. 1992

[8] Dromey R.G.: A model for software product quality. IEEE transaction on Software Egineering, 21(2), 1995, s 145-162

[9] ISO/IEC 9126- - Software engineering. - Product quality. - Part 1: Quality model / ISO/IEC, 2001.

[10] ISO/IEC 25010:2011. Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models / ISO/IEC. – 2011.

[11] Жарко Е. Ф. Сравнение моделей качества программного обеспечения: аналитический подход, Институт проблем управления им. В. А. Трапезникова, Москва 2014.

[12] Метрики //http://www.metrix.narod.ru/page1.htm.[27.09.2016]

[13] Харченко О. Розробка та керування вимогами до програмного забезпечення на основі моделі якості – Вісник ТДТУ – 2009. Том 14. №1. – с. 201-207

[14] Ледовских И. Метрики сложности кода. – 2012. – с. 22.

[15] T. McCabe, A Software Complexity Measure. IEEE Transactions on Software Enginnering, 1976.

[16] Харченко В.С. Использование метрик Холстеда при оценке безопасности критического программного обеспечения 2003. – с.7.