Porzeziński Michał
Politechnika Gdańska, Gdańsk, Polska

Software quality and reliability management in safety-related systems Zarządzanie jakością i niezawodnością oprogramowania w systemach związanych z bezpieczeństwem

Keywords / Słowa kluczowe

software quality and reliability, functional safety, safety-related systems
jakość i niezawodność oprogramowania, bezpieczeństwo funkcjonalne, systemy związane z bezpieczeństwem

Abstract

This article is concerned with the methods of ensuring the required quality and reliability of software in safety-related systems. The basic types of software reliability models and their specific properties are presented. The principles of managing the process of software development, based on the "Model V" life cycle, with particular emphasis on the role of inspection and testing processes are discussed. Also the methodology of quality management and reliability of the software recommended by the PN-EN 61508-3 are outlined. At the end the concept of application to support the process of assessing the quality and integrity of the safety-related software is described.

1. Wprowadzenie

Nieodłącznym składnikiem większości nowoczesnych systemów sterowania, w tym również systemów bezpieczeństwa funkcjonalnego, są elementy zbudowane na bazie systemów mikroprocesorowych. Wykorzystanie elementów programowalnych pociąga za sobą konieczność opracowywania odpowiedniego oprogramowania, które jest ich niezbędnym składnikiem.

Cechą charakterystyczną oprogramowania jest to, że w odróżnieniu od elementów sprzętowych, nie podlega ono awariom wynikającym z procesów starzenia czy zużycia. Może natomiast zawierać w sobie błędy systematyczne wprowadzone w trakcie jego projektowania i wytwarzania. Błędy te mogą pozostawać ukryte przez znaczną cześć cyklu życia oprogramowania i ujawniać się dopiero przy zajściu pewnych specyficznych warunków będących odpowiednią kombinacją sygnałów wejściowych lub sekwencją zdarzeń. Może wówczas dojść do awarii systemu uniemożliwiającej jego poprawne działanie. Czasami zdarza się to nawet po wielu latach prawidłowej pracy systemu [9].

Problemu istnienia błędów systematycznych nie da się rozwiązać za pomocą prostej redundancji opartej na powieleniu sterowników zawierających identyczne oprogramowanie użytkowe. Przy takim podejściu te same błędy systematyczne będą występowały w oprogramowaniu każdego ze sterowników. Aby redundancja była skuteczna zaleca się stosowanie programów pisanych przez różnych programistów i najlepiej w różnych językach programowania. Wówczas ryzyko wystąpienia tego samego błędu systematycznego oprogramowania we wszystkich sterownikach jest dużo mniejsze.

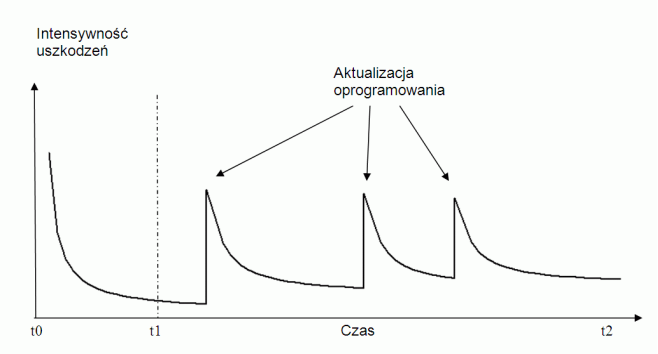
Inną metodą walki ze skutkami błędów systematycznych jest wyposażanie oprogramowania w wewnętrzne mechanizmy kontrolne pozwalające wykryć błędne działanie programu i ograniczyć jego skutki. Należy się jednak liczyć z tym, że implementacja takich mechanizmów komplikuje oprogramowanie i może powodować wprowadzenie kolejnych błędów systematycznych.

Niezależnie od wymienionych metod, podstawową metodą zapewnienia wymaganej jakości i niezawodności oprogramowania pozostaje eliminacja błędów systematycznych jeszcze „w zarodku", czyli w trakcie powstawania programu. Można to osiągnąć poprzez wdrożenie odpowiednich zasad projektowania, właściwe testowanie i szczegółową kontrolę całego procesu jego

wytwarzania. Zagadnienia te są tematem niniejszego artykułu.

2. Modele probabilistyczne

Od chwili rozpowszechnienia się systemów programowalnych podejmowane są próby matematycznego opisu właściwości niezawodnościowych oprogramowania. Co prawda oprogramowanie zawiera zwykle błędy systematyczne niezmienne pod względem ilościowym i jakościowym w czasie, jednak ich ujawnianie się podlega procesom stochastycznym, związanym z losowym charakterem sygnałów wejściowych.
Poglądowy wykres funkcji intensywności uszkodzeń oprogramowania został pokazany na *Rysunku 1* [5]. Różni się on od typowych charakterystyk elementów sprzętowych brakiem końcowej fazy wzrostu intensywności uszkodzeń związanej ze zjawiskiem „zużycia" elementów. Występują natomiast zjawiska okresowego wzrostu intensywności uszkodzeń związane z aktualizacją oprogramowania.



Rysunek 1. Wykres intensywności uszkodzeń oprogramowania [5]

Przyjmuje się, że t0 jest chwilą rozpoczęcia testów oprogramowania. Czas t1 jest momentem oddania oprogramowania do użytkowania. Czas t2 chwilą wycofania oprogramowania z użycia. Pomiędzy chwilami t0 i t1 powinna zostać wykryta i usunięta większość błędów, niewielka część z nich jednak nadal pozostaje i może ujawnić się w okresie eksploatacji. Ponadto, każda aktualizacja zakresu funkcjonalnego oprogramowania wiąże się z możliwością wprowadzenia do oprogramowania nowych błędów.
Obserwacje pokazują, że po każdej takiej aktualizacji intensywność uszkodzeń gwałtownie wzrasta, a następnie stopniowo maleje w czasie, przy czym nie osiąga już poziomu tak niskiego jak sprzed aktualizacji. Tłumaczy się to dwoma czynnikami. Po pierwsze, po wprowadzeniu aktualizacji testowanie skupia się zwykle na nowej funkcjonalności, podczas gdy zmiany mogą wprowadzać błędy również w funkcjonalności wbudowanej wcześniej. Po drugie, zmiany wprowadzane w oprogramowaniu często zaburzają jego pierwotną architekturę skutkując zwiększeniem jego skomplikowania i zmniejszeniem przejrzystości (zjawisko nazywane „lava flow"), a tym samym zwiększają ryzyko istnienia niewykrytych błędów.

2.1. Modele estymacyjne

Modele estymacyjne służą do opisu właściwości niezwodnościowych istniejącego oprogramowania. W większości modeli tego typu wykorzystuje się rozkład wykładniczy. Wówczas niezawodność oprogramowania opisana jest zależnością:

$$R(t) = e^{(-\lambda \cdot t)} \tag{1}$$

gdzie: λ jest intensywnością uszkodzeń, a t czasem działania oprogramowania.
Intensywność uszkodzeń jest najczęściej wiązana z liczbą błędów systematycznych obecnych i już wykrytych w oprogramowaniu. Przykładowe modele estymacyjne tego typu to: Lloyd-Lipow, Musa's Basic, Shooman's oraz Goel-Okumoto, które zostały przedstawione w opracowaniu [5]. Ponadto, w dodatku D normy PN-EN 61503-7 [8] podano zależności umożliwiające oszacowanie prawdopodobieństwa wystąpienia błędu w oprogramowaniu na podstawie doświadczeń z dotychczasowej eksploatacji/testowania danego programu traktowanego jak „czarna skrzynka".
Metody te są szczególnie przydatne w przypadku oprogramowania narzędziowego i systemowego, bibliotek programistycznych oraz innego oprogramowania, które jest dostarczane w postaci gotowego produktu, bez informacji o szczegółach związanych z jego cyklem wytwarzania.
Przedstawione metody opierają się na informacji o czasie bezawaryjnej pracy danego programu w przypadku pracy w trybie ciągłym oraz w trybie częstego przywołania lub na liczbie dotychczasowych testów poprawnego zadziałania programu w przypadku pracy w trybie rzadkiego przywołania. Minimalna liczba prób lub minimalny czas bezawaryjnej pracy programu, które pozwalają zakwalifikować go do danego poziomu nienaruszalności bezpieczeństwa są określone zależnościami [8]:

$$n = -\frac{\ln \alpha}{p} \tag{2}$$

$$t = -\frac{\ln \alpha}{\lambda} \tag{3}$$

gdzie: *n* jest minimalną liczbą testów (dotyczy trybu rzadkiego przywołania do działania), *p* jest prawdopodobieństwem niezadziałania na przywołania (dotyczy trybu rzadkiego przywołania), *t* oznacza minimalny czas pracy programu w godzinach (dotyczy trybu pracy ciągłej), λ jest prawdopodobieństwem wystąpienia błędu na jednostkę czasu godzinę (dotyczy trybu pracy ciągłej), natomiast *1-α* jest zakładanym poziomem ufności.
Spełnionych musi być przy tym szereg założeń. Między innymi: statystyczny rozkład danych testowych musi odpowiadać rozkładowi danych wejściowych podczas pracy programu w rzeczywistym środowisku, przebiegi testów w trybie rzadkiego przywołania muszą być statystycznie niezależne od siebie, liczba testów musi być odpowiednio duża ($n > 100$) i musi istnieć odpowiedni mechanizm umożliwiający wykrycie każdego błędu.
Jak widać, do przedstawionych zależności należy podchodzić ostrożnie, gdyż bazują na modelach probabilistycznych, będących dużym uproszczeniem rzeczywistości w przypadku analizy niezawodności oprogramowania.

2.2. Modele predykcyjne

Inną kategorią modeli opisujących właściwości oprogramowania są modele predykcyjne. Przykładami mogą być opisane w literaturze [5] modele: Musa, Putama, TR-92-52 czy TR-92-15. Ich cechą wspólną jest powiązanie parametrów niezawodnościowych oprogramowania z takimi czynnikami jak: liczba linii programu, rodzaj użytego języka programowania, doświadczenie programisty itp. Mając na względzie, że modele te uwzględniają tylko niektóre czynniki wpływające na jakość oprogramowania, a ich wyskalowanie wymaga posiadania dużej liczby danych historycznych, co nie zawsze jest możliwe, ich znaczenie praktyczne wydaje się być niewielkie. Są jednak źródłem informacji wskazujących to, w jaki sposób czynniki związane z procesem projektowania wpływają na niezawodność finalnego oprogramowania.
Modele te pokazują, że jedną z dróg uzyskania niezawodnego oprogramowania jest dążenie do maksymalnego uproszczenia programu przy zachowaniu jego wymaganej funkcjonalności. Program powinien być przejrzysty i realizować tylko te funkcje, które są niezbędne do realizacji danego zadania. W opracowaniu [9] podawane są dane, według których „ … typowo, dla dużych systemów na każdy milion linii kodu przypadało 20 tys. błędów. Normalnie 90% z nich było wykrywanych podczas testowania. Dalsze 200 uzewnętrzniało się w ciągu roku eksploatacji …”.
Uproszczenie i przejrzystość oprogramowania aplikacyjnego można uzyskać poprzez dobór odpowiednich narzędzi, bibliotek i języków oprogramowania. Z tego względu w systemach związanych z bezpieczeństwem preferowane są języki o ograniczonej zmienności (LVL), w których istnieje mniejsze ryzyko popełnienia błędu niż w językach o pełniej zmienności (FVL).

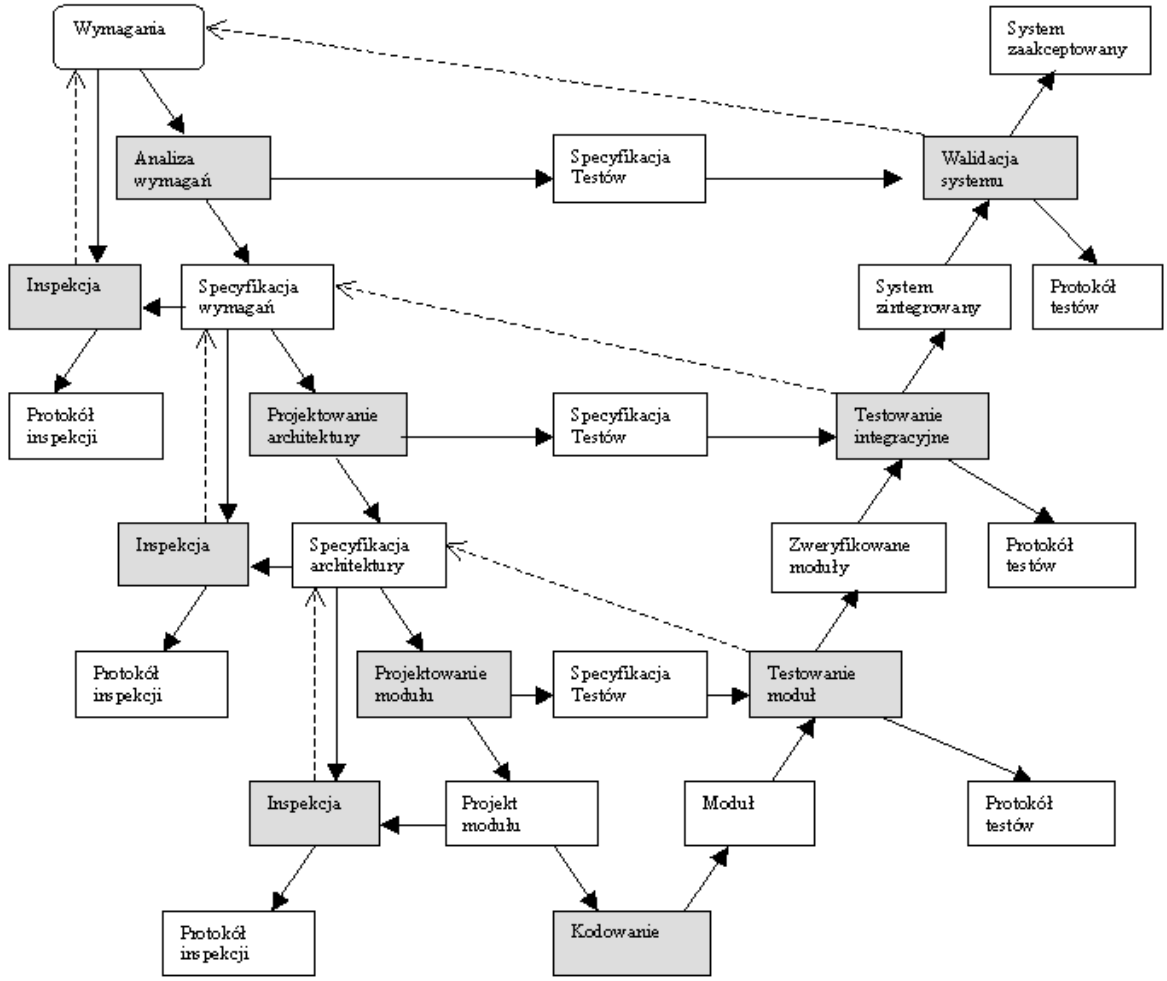
3. Zarządzanie procesem wytwarzania oprogramowania

Niezależnie od rodzaju i złożoności oprogramowania najważniejszą metodą zwiększenia jego niezawodność jest odpowiednie zarządzanie cyklem życia całości projektu. Ponieważ wytwarzanie oprogramowania jest procesem złożonym, na potrzeby jego opisu opracowano różne modele opisujące poszczególne fazy i zależności pomiędzy nimi. Charakterystykę najczęściej stosowanych modeli można znaleźć w pracy [2] oraz [3].

3.1. Modele cyklu życia

Najprostszym modelem jest model kaskadowy. Posiada on liniową strukturę następujących po sobie działań począwszy od specyfikacji, a skończywszy na wdrożeniu systemu. Model ten może być stosowany tylko do bardzo prostych projektów. Jego wadą jest brak uwzględnienia iteracyjnego charakteru rzeczywistego procesu projektowania i wytwarzania oprogramowania, który zakłada możliwość cofania się do wcześniejszych etapów projektowania i ich modyfikacji.
Inną formą opisu procesu projektowania jest model spiralny, w którym każdy etap projektu przechodzi przez cztery fazy: fazę wyznaczania celów, szukania alternatyw i szacowania ryzyka, wytworzenia i weryfikacji oprogramowania oraz planowania następnego etapu.
W praktyce, do zarządzania dużym projektem najczęściej wykorzystywany jest jednak tzw. "Model V", którego rozbudowaną wersję przedstawiono na *Rysunku 2*. Zasadniczą cechą tego modelu jest podział na dwa główne etapy wytwarzania oprogramowania: etap projektowania i etap testowania. Są one reprezentowane przez odpowiednie ramiona litery V. Kolorem szarym oznaczono aktywne fazy wytwarzania oprogramowania. Kolorem białym oznaczono dokumenty i specyfikacje powstające podczas tego procesu. Ciągłymi strzałkami oznaczono kierunek

przepływu informacji. Strzałki przerywane pokazują drogę do ewentualnej korekty odpowiedniej specyfikacji, jeżeli okaże się to konieczne w wyniku przeprowadzonych testów i inspekcji.



Rysunek 2. Rozbudowany model V cyklu wytwarzania oprogramowania

3.2. Inspekcje

Każdy z etapów projektowania powinien wiązać się z opracowaniem właściwej specyfikacji projektowej, opracowaniem odpowiedniej specyfikacji testów oraz przeprowadzeniem inspekcji dokumentacji wraz ze sporządzeniem odpowiedniego protokołu. Inspekcja dokumentów związanych z projektem jest bardzo efektywną, pod względem kosztów, metodą zwiększenia niezawodności oprogramowania. Pozwala bowiem zmniejszyć liczbę defektów już we wczesnych fazach cyklu życia projektu. Badania pokazują, że koszt znalezienia i korekcji defektu rośnie od 5 do10 razy dla każdej następnej fazy, względem fazy w której ten defekt został wprowadzony [3]. W tym samym opracowaniu podano, że dobrze przeprowadzona inspekcja pozwala wykryć ponad 80% bledów. Przeprowadzanie inspekcji pozwala ponadto zredukować koszt projektowania i czas potrzebny na wykonanie projektu nawet o 50%.

W inspekcji powinny wziąć udział strony pełniące role: autora odpowiadającego za przygotowanie ocenianego elementu projektu, inspektora oceniającego projekt i moderatora/lidera – osoby czuwającej nad przebiegiem inspekcji. Główny ciężar procesu inspekcji spoczywa na stronie pełniącej funkcję inspektora. Aby inspekcja była efektywna powinna być dokonywana przez niezależną osobę/zespół względem osoby/zespołu przygotowującego daną specyfikację lub projekt. Ważne są tutaj również aspekty psychologiczne. Autor nie może czuć się atakowany ani krytykowany jako osoba. Nie powinno się też podczas inspekcji naprawiać wykazanych błędów pozostawiając to zadanie autorowi oprogramowania. Efektem

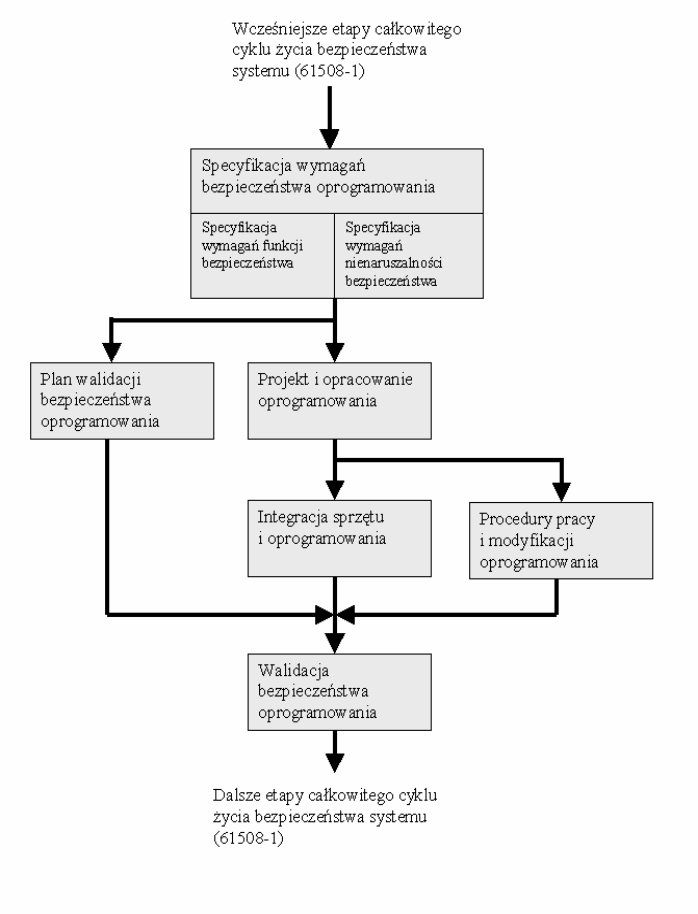
końcowym formalnej inspekcji powinno być przede wszystkim opracowanie protokołu wykazującego stwierdzone błędy.
Szczegółowy opis procesu przeprowadzania inspekcji wraz z propozycją wykorzystywanych w niej formularzy i dokumentów można znaleźć w pracach [2] i [3].

3.3. Testowanie

W chwili powstania pierwszych modułów programu pojawia się możliwość weryfikacji poprawności oprogramowania poprzez jego testowanie. Przeprowadzanie odpowiednio dobranych eksperymentów pozwala wykryć znaczną część defektów, które nie zostały wykryte w fazie projektowania (lewe ramie modelu „V"). Problematyka testowania i weryfikacji oprogramowania została przedstawiona w pracy [2].
Przyjmuje się, że odpowiednie testy powinny zostać przeprowadzone dla każdego z poziomów modelu V w oparciu o specyfikację testów przygotowaną w trakcie projektowania. Dokumentacja powinna zawierać opis środowiska testowego, harmonogramu prac (plan testów), specyfikacje przypadków testowych oraz kryteria zaliczenia testu.
Wynikiem przeprowadzonego testu powinien być log testowy (opis wykonanych czynności testowych i uzyskanych rezultatów), raport zdarzeń, które nie były uwzględnione w specyfikacji testów, a wystąpiły w trakcie testowania oraz podsumowanie testu. Bardzo ważnym czynnikiem mającym wpływ na skuteczność testowania jest dobór technik testowania i zestawu danych testowych. Jest on uzależniony od poziomu testowania wynikającego z modelu cyklu życia (testowanie modułu, testowanie integracyjne, walidacja) jak i rodzaju oprogramowania, jego struktury, itp. Zestaw możliwych do zastosowania technik i ich szczegółowy opis można znaleźć w literaturze [1], [2], [8].

4. Wymagania normy EN-PN 61508-3

Dobranie odpowiednich technik i miar stosowanych podczas tworzenia i weryfikacji poszczególnych elementów projektu jest zadaniem trudnym. W dużej mierze zależy od specyfiki danego oprogramowania i powinno być wykonywane przez specjalistów znających zarówno środowisko eksploatacji programu jak i zagadnienia związane z jego wytwarzaniem.
W przypadku oprogramowania wykorzystywanego w systemach bezpieczeństwa funkcjonalnego dobór odpowiednich metod stosowanych podczas projektowania, inspekcji oraz testowania oprogramowania precyzuje norma PN-EN 61508-3. Wymagania tej normy oraz problematyka integracji oprogramowania ze sprzętem są również tematem opracowania [4].
W wymienionych opracowaniach oparto się na modelu cyklu życia bezpieczeństwa oprogramowania przedstawionym na *Rysunku 3*.



Rysunek 3. Cykl życia bezpieczeństwa oprogramowania wg PN-EN 61508-3 [7]

Jego struktura stanowi wycinek cyklu życia bezpieczeństwa całego projektowanego systemu. Norma dopuszcza również stosowanie innych modeli, jeżeli będą lepiej pasować do potrzeb danego projektu lub organizacji. W szczególności, wybór modelu będzie podyktowany złożonością oprogramowania i całego systemu. Podstawowy model obejmuje:
- przygotowanie specyfikacji wymagań bezpieczeństwa,
- projekt i opracowanie oprogramowania,
- integrację sprzętu i oprogramowania,
- walidację bezpieczeństwa oprogramowania,
- opracowanie planu walidacji oprogramowania oraz procedur pracy i modyfikacji.

Etap projektowania i opracowania oprogramowania obejmuje ponadto, nie uwidocznione na rysunku, fazy szczegółowe:
- projekt architektury oprogramowania,

- wybór narzędzi wspomagających i języków oprogramowania,
- opracowanie szczegółowego projektu systemu,
- opracowanie szczegółowego projektu modułów,
- szczegółową implementację kodu,
- testowanie modułów oprogramowania,
- testowanie integracji oprogramowania.

Dla każdej z faz cyklu życia norma PN-EN 61508-3 określa wymagania ogólne oraz wykaz zaleceń szczegółowych, który jest uzależniony od żądanego poziomu nienaruszalności bezpieczeństwa systemu (SIL). Wartość SIL jest wynikiem odrębnych analiz i musi być wcześniej określona. Wymagania szczegółowe zostały przedstawione w postaci tabel zawierających zestawienie rekomendowanych technik i miar stosowanych na poszczególnych etapach cyklu życia projektu. Techniki te są zebrane i opisane w normie PN-EN 61503-7 [8].

W tabelach zawierających zestawienie wymagań szczegółowych zastosowano cztery poziomy rekomendacji: techniki wysoce rekomendowane (HR), rekomendowane (R), obojętne (---) oraz niezalecane (NR).
Techniki alternatywne i równoważne oznaczane są kolejnymi literami alfabetu umieszczonymi po cyfrach np. 4a, 4b. Do spełnienia wymagań normy wystarczy zastosowanie jednej z technik alternatywnych. Nie zastosowanie w danym projekcie rekomendowanej dla danego wymaganego poziomu nienaruszalności bezpieczeństwa techniki wymaga podania szczegółowego formalnego uzasadnienia.
Przykład zestawienia zalecanych technik związanych z etapem wyboru języków programowania i narzędzi wspomagających został pokazany w *Tabeli 1*.

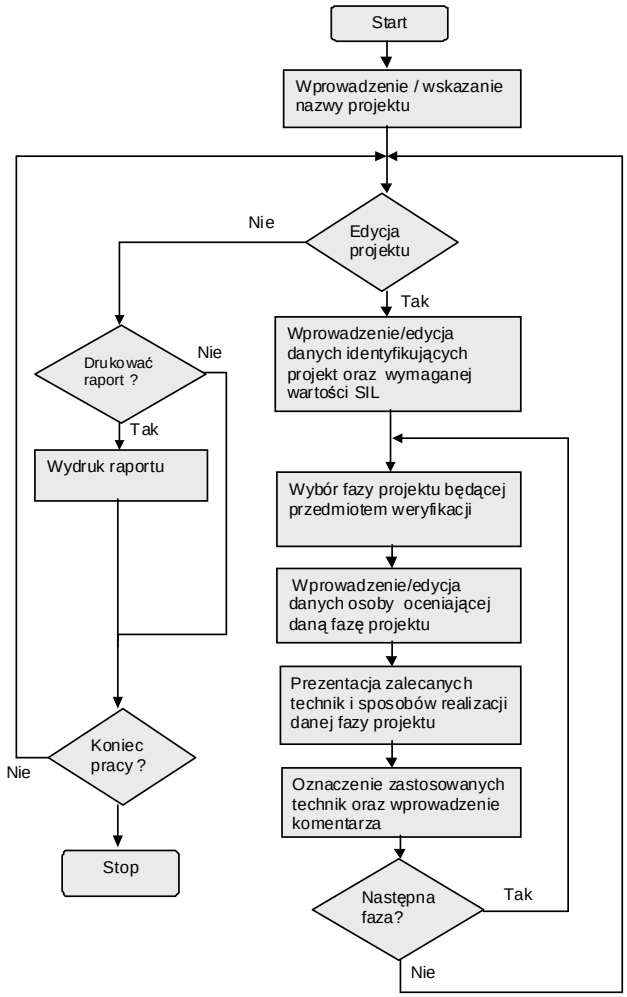
Tabela 1. Przykładowa lista wymagań szczegółowych dotyczących wyboru języków programowania i narzędzi wspomagających (na podstawie [7])

Nr	Technika/sposób	Powołanie	SIL1	SIL2	SIL3	SIL4
1	Odpowiedni język programowania	C.4.6	HR	HR	HR	HR
2	Język programowania silnie stypizowany	C.4.1	HR	HR	HR	HR
3	Podzbiór języka	C.4.2	---	---	HR	HR
4a	Narzędzia certyfikowane	C.4.3	R	HR	HR	HR
4b	Narzędzia o zwiększonym zaufaniu wskutek stosowania	C.4.4	HR	HR	HR	HR

5. Moduł oceny jakości i integralności oprogramowania

Mając na uwadze dużą liczbę szczegółowych wymagań przedstawionych w normie PN-EN 61508-3 opracowano koncepcję programu wspomagającego proces inspekcji oprogramowania stosowanego w systemach bezpieczeństwa funkcjonalnego. Program jest obecnie w fazie wdrażania. Ma on postać dodatkowego modułu rozszerzającego możliwości oprogramowania „Pro-SIL" opracowanego w ramach projektu „Opracowanie metod analizy i narzędzi do komputerowo wspomaganego zarządzania bezpieczeństwem funkcjonalnym w ramach systemu warstw zabezpieczeniowo-ochronnych obiektów przemysłowych podwyższonego ryzyka".
Wsparcie to polega na prezentowaniu użytkownikowi wymaganych miar i technik wytwarzania oprogramowania podlegającego ocenie, wynikających z przyjętego poziomu nienaruszalności SIL. Moduł ułatwia proces inspekcji dzięki możliwości wprowadzania przez użytkownika komentarzy oraz archiwizacji efektów oceny w postaci elektronicznej i w postaci drukowanych raportów. Moduł może być wykorzystywany również podczas projektowania oprogramowania poprzez prezentację wymagań stawianych dla jego poszczególnych faz.
Założono, że przed użyciem modułu do oceny jakości i integralności oprogramowania musi on zostać skonfigurowany. Celem konfiguracji modułu jest dostosowanie go do potrzeb danego przedsiębiorstwa lub projektu, w którym będzie wykorzystywany. Konfiguracja polega na wprowadzeniu do bazy danych zestawienia faz cyklu życia projektu oprogramowania podlegającego ocenie oraz listy miar i technik zalecanych do przeprowadzenia oceny dla różnych wartości wymaganego poziomu nienaruszalności bezpieczeństwa SIL. Źródłem tych danych powinny być odpowiednie dokumenty normatywne, w szczególności norma EN-PN 61503 oraz normy pochodne, specyficzne dla danej branży przemysłu, w której stosowane jest oceniane oprogramowanie. Konfiguracja modułu powinna być dokonana przez osobę posiadającą uprawnienia administratora systemu Pro-SIL.
Proces oceny oprogramowania został przedstawiony w postaci schematu blokowego na *Rysunku 4*. Proces

rozpoczyna się od wprowadzenia danych identyfikujących oprogramowanie: nazwy, rodzaju oprogramowania, realizowanej funkcji oraz wymaganego poziomu SIL. Informacje te musza być znane przed rozpoczęciem oceny jakości i integralności danego oprogramowania. Wymagany poziom SIL ocenianego oprogramowania jest wartością wynikającą z analizy całości systemu bezpieczeństwa funkcjonalnego, którego elementem jest oprogramowanie, z uwzględnieniem istniejących zagrożeń oraz możliwych skutków awarii.



Rysunek 4. Algorytm działania modułu oceny jakości i integralności oprogramowania.

Każda z faz cyklu życia projektu podlega osobnej ocenie. Podczas wyboru ocenianej fazy projektu wprowadzane są również dane personalne osoby oceniającej tą fazę. Dane te są później umieszczane w sporządzanych raportach.
Dla każdej fazy projektu prezentowane są wybrane techniki i miary zalecane dla danego etapu projektowania oraz wymaganej wartości SIL określonej dla projektu. Prezentowane techniki/miary ładowane są z bazy danych przygotowanej na etapie konfigurowania modułu. Każda z technik/miar jest prezentowana wraz rekomendację zgodną z konwencją EN-PN 61508-3. Dodatkowym ułatwieniem jest podawanie dla każdej techniki/miary odniesienia do dokumentu, w którym zalecenie zostało szczegółowo zdefiniowane.
Każda pozycja wymaga od oceniającego zaznaczenia - czy została zastosowana, a także wprowadzenia odpowiedniego komentarza uzasadniającego jej zastosowanie lub pominięcie. W komentarzu powinny znaleźć się również odniesienia do odpowiednich dokumentów będących podstawą dokonania danej oceny. Informacje te są później prezentowane w generowanych raportach.
Raport obejmuje wydruk zgromadzonych danych w postaci struktury składającej się ze strony tytułowej projektu i rozdziałów związanych z poszczególnymi fazami projektu. Fazy projektu, które nie zostały poddane ocenie, a także wymagane dla danej fazy techniki, których zastosowanie nie zostało zweryfikowane przez oceniającego są podczas drukowania wyróżniane czerwonym kolorem.

6. Podsumowanie

Opracowanie uniwersalnych metod oceny niezawodności oprogramowania jest zadaniem bardzo trudnym. Trudność ta wynika zarówno z bardzo dużej złożoności typowego oprogramowania jak i jego różnorodności. W szczególności bardzo trudna i nie zawsze miarodajna jest analiza ilościowa parametrów związanych z niezawodnością oprogramowania, gdyż modele wykorzystywane w analizie ilościowej są bardzo dużym uproszczeniem rzeczywistych obiektów. Z tego powodu w literaturze przedstawiane są najczęściej jedynie ogólne zasady i zalecenia, które powinny być stosowane w trakcie cyklu życia wytwarzania oprogramowania.
Bardziej systematyczne podejście zastosowano w normie PN-EN 61508-3, w której określono rekomendowany zestaw technik i sposobów zapewnienia jakości stosowania podczas poszczególnych faz cyklu życia oprogramowania. Do szczegółowej oceny niezawodności danego rozwiązania niezbędna jest jednak zawsze dodatkowa wiedza ekspercka dotycząca zarówno zagadnień projektowania oprogramowania jak i środowiska jego eksploatacji.
Zaprojektowany moduł oceny jakości i integralności oprogramowania ma za zadanie ułatwiać proces inspekcji i testowania oprogramowania poprzez prezentowanie zalecanych technik i miar, gromadzenie danych z przebiegu inspekcji oraz drukowanie odpowiednich raportów. Oprogramowanie ProSIL jest obecnie w trakcie testowania.

Podziękowanie

Autor niniejszego artykułu dziękuje Ministerstwu Nauki i Szkolnictwa Wyższego za wsparcie badań oraz Centralnemu Laboratorium Ochrony Pracy – Państwowemu Instytutowi Badawczemu za współpracę w przygotowaniu projektu badawczego VI.B.10 do realizacji w latach 2011-13 dotyczącego zarządzania bezpieczeństwem funkcjonalnym w obiektach podwyższonego ryzyka z włączeniem zagadnień zabezpieczeń / ochrony i niezawodności człowieka.

Literatura

[1] Beizer, B. (1990). *Software Testing Techniques*. Second Edition, Van Nostrand Reinhold, New York.
[2] Górski, J. (2000). *Inżynieria oprogramowania w projekcie informatycznym.* Mikom, Warszawa.
[3] Herard, J., Hedberg, J., Kivipuro, M, Malm, T., Edler, H., Sjostrom, H. & Strawinski, T. (2003). *Validation of communication in safety-critical controls system*. Nordtest Tekniikantie.
[4] Kosmowski, K. & Śliwiński, M. (2004). Jakość i bezpieczeństwo oprogramowania E/E/PE. Zarządzanie bezpieczeństwem funkcjonalnym, Konferencja naukowo-techniczna, Jurata 2004, 159-168.
[5] MIL-HDBK-338B (1998). Military Handbook. Electronic Reliability Design Handbook.
[6] PN-EN 61508-2 (2004). Cześć 2: Wymagania dotyczące E/E/P systemów związanych z bezpieczeństwem.
[7] PN-EN 61508-3 (2004). Część 3: Wymagania dotyczące oprogramowania.
[8] PN-EN 61508-7(2003). Część 7: Przegląd technik i miar.
[9] Żurakowski, Z. (2004). Cele certyfikacji systemów komputerowych. Zarządzanie bezpieczeństwem funkcjonalnym, Konferencja naukowo-techniczna, Jurata 2004, 169-180.