Modele wzrostu niezawodności oprogramowania

R. PEŁKA
e-mail: radoslaw.pelka@wat.edu.pl

Instytut Systemów Informatycznych
Wydział Cybernetyki WAT
ul. S. Kaliskiego 2, 00-908 Warszawa

Historia badań nad niezawodnością oprogramowania sięga lat 70. ubiegłego wieku. Od momentu pojawienia się pierwszych publikacji poświęconych tej tematyce nastąpił znaczący rozwój i postęp prac mających na celu między innymi budowę matematycznego modelu umożliwiającego badanie wzrostu niezawodności oprogramowania w procesie jego testowania. Analizując dostępną literaturę, można dojść do wniosku, że nie istnieje rozwiązanie uniwersalne, które dałoby się zastosować w każdym przypadku. Możliwa jest natomiast klasyfikacja dostępnych modeli ze względu na cechy charakterystyczne poszczególnych rozwiązań, takie jak dziedzina danych, sposób opisu błędów pojawiających się w procesie testowania, sposób opisu niezawodności czy też pozostałych założeń, w tym narzędzi matematycznych wykorzystywanych w procesie ewaluacji. W artykule przedstawiono przegląd istniejących rozwiązań modelowania niezawodności oprogramowania, kładąc nacisk na różnorodność aspektów oraz metod wykorzystywanych w tym procesie.

Słowa kluczowe: modelowanie, oprogramowanie, niezawodność oprogramowania

1. Wprowadzenie

Historia badań niezawodności w odniesieniu do oprogramowania sięga lat 70. ubiegłego wieku. Prekursorami nowej idei byli Jeliński i Moranda, Shooman oraz Coutinho, którzy w owym czasie opublikowali swoje prace poświęcone tej tematyce. W pracach tych podjęto się zadania budowy modelu matematycznego umożliwiającego, na podstawie zgromadzonych danych testowych, prognozowanie niezawodności badanego programu, który będzie eksploatowany w przyszłości. Niezawodność definiowana przez standard IEEE/ANSI 982.2, jako prawdopodobieństwo bezbłędnego działania programu w określonym czasie i środowisku, należy do kluczowych cech, charakteryzujących oprogramowanie. W dzisiejszych czasach wyrasta wręcz na kluczową wartość obok pozostałych istotnych elementów wpływających na zadowolenie klienta, takich jak np. funkcjonalność lub wydajność.

Od momentu pojawienia się pierwszych publikacji przedstawiających modele umożliwiające badanie niezawodności oprogramowania, powstały dziesiątki nowych prac prezentujących zupełnie nowe podejścia do tej tematyki lub też modyfikujących już istniejące rozwiązania. Do tej pory nie istnieje jednak uniwersalny model, który dałoby się zastosować w każdym przypadku. Wynika to z założeń, jakie są czynione na gruncie budowania modelu, a które nie mają charakteru uniwersalności. Z tego względu należy, na podstawie specyfiki badanego oprogramowania oraz warunków jego eksploatacji, określić rozwiązanie najodpowiedniejsze. Nie jest to zadanie łatwe i wymaga często długotrwałych prób, które mają doprowadzić do wyboru odpowiedniego modelu, a następnie określenia wartości jego parametrów.

2. Informacje podstawowe

Modele niezawodności oprogramowania można podzielić generalnie na dwa typy. Pierwsza grupa wykorzystuje do szacowania poziomu niezawodności charakterystyki kodu (metryki), takie jak liczba linii kodu, poziom zagnieżdżenia pętli, zewnętrzne referencje, liczba wejść i wyjść itp. W drugiej grupie poszukuje się statystycznych korelacji danych o wykrywanych błędach ze znanymi funkcjami, np. funkcją wykładniczą. W momencie uzyskania takiej korelacji, właściwości funkcji są wykorzystywane do przewidywania zachowania oprogramowania w przyszłości. Dlatego też tego rodzaju modele nazywane są modelami wzrostu niezawodności oprogramowania. To podejście jest dużo szerzej rozpatrywane przez badaczy.

Modele wzrostu niezawodności oprogramowania można podzielić na dwie kategorie w zależności od dziedziny, w jakiej operują. Najszerszą i jednocześnie najbardziej popularną grupę stanowią modele z dziedziną czasu. W tych modelach niezawodność,

związana m.in. z intensywnością błędów, definiowana jest jako funkcja czasu. Drugą kategorię stanowią modele z dziedziną danych. W tych modelach z kolei niezawodność jest funkcją wykonania programu dla pewnych danych wejściowych i określa liczbę udanych uruchomień w stosunku do całkowitej ich liczby. Znacznie większą uwagę naukowcy przywiązują do modeli badających niezawodność na gruncie upływającego czasu. Warto w tym miejscu wspomnieć, że z punktu widzenia niezawodności oprogramowania można wyróżnić trzy modele upływu czasu:

- czas wykonania – czas, jaki jednostka centralna komputera poświęca na wykonanie programu
- czas kalendarzowy – czas w powszechnie rozumianym i na co dzień używanym znaczeniu (godziny, dni, tygodnie, miesiące, lata)
- czas zegarowy – czas, jaki upływa od momentu uruchomienia oprogramowania, odliczając momenty, gdy sprzęt, na którym działa program, jest np. wyłączony.

Czas kalendarzowy był początkowo jedynym sposobem definiowana czasu w proponowanych modelach, czego celowość poddał w wątpliwość Musa. Okazało się, że przejście na czas wykonania spowodowało uproszczenie modeli i pozwalało uzyskać lepsze rezultaty. Wyższość jednego rozwiązania nad drugim udowadniali w swoich pracach Trachtenberg (1985), Musa i Okumoto (1984), Hecht (1981). Niemniej jednak są modele, które używają czasu kalendarzowego lub nie określają wprost tego, jaki rodzaj czasu powinien zostać zastosowany. W 1987 r. Musa podał sposób na zamianę rezultatów modelowania pomiędzy jednym a drugim sposobem definiowania czasu.

Modele wzrostu niezawodności określają w sposób ogólny zależność pojawiania się błędów w oprogramowaniu od głównych czynników, które wpływają na ten proces, czyli wprowadzanie błędów, usuwanie błędów oraz środowisko działania. Główną ideą modelowania niezawodności oprogramowania jest to, aby w miarę wykrywania i usuwania błędów, liczba błędów pojawiających się w jednostce czasu malała lub też z drugiej strony czas pomiędzy kolejnymi wystąpieniami błędów wydłużał się. Jest to sytuacja, którą należy w modelach uwzględnić. Kiedy to się stanie, można wykorzystując mechanizmy statystyczne określić zachowanie się procesu w przyszłości. Taką wiedzę można wykorzystać na dwa sposoby. Po pierwsze do określenia dodatkowego czasu, jaki jest potrzebny do uzyskania pożądanego poziomu niezawodności, po drugie do określenia poziomu niezawodności na koniec okresu testowania, przy zachowaniu obecnego poziomu wykrywalności błędów. Na podstawie takich informacji możliwe jest zweryfikowanie planów testowych w celu osiągnięcia zamierzonego celu. Istnieje więc możliwość oszacowania bieżącej sytuacji i odpowiedniej na nią reakcji.

Pomiary czynione w celu określania niezawodności oprogramowania związane są z dwoma pojęciami: estymacją (szacowaniem) miar niezawodności oraz przewidywaniem (prognozowaniem) charakterystyk niezawodności. Pierwsze pojęcie określa niezawodność w aktualnym momencie poprzez zastosowanie metod statystycznych dla danych o błędach uzyskanych podczas fazy testowania lub dotychczasowej eksploatacji. Głównym powodem tych obliczeń jest uzyskanie obecnego stanu niezawodności i określenie, czy model wzrostu niezawodności dobrze oddaje zachowanie oprogramowania w przeszłości – czy został dobrze dobrany (skalibrowany). Funkcje charakteryzujące błędy pojawiające się w czasie eksploatacji oprogramowania można podzielić ze względu na sposób, w jaki reprezentują one te błędy:

- funkcja wartości średniej – określa spodziewaną, całkowitą liczbę błędów w każdym momencie czasu
- funkcja intensywności występowania błędów – określa tempo zmian funkcji wartości średniej
- funkcja ryzyka występowania błędów – określa prawdopodobieństwo, że błąd w odniesieniu do pewnej jednostki czasu wystąpi w przedziale $[t, t+D_t]$ pod warunkiem, że błąd nie wystąpił do chwili t
- funkcja wartości średniej czasu do wystąpienia błędu (MTTF – ang. mean time to failure) – określa spodziewaną wartość czasu, w którym błąd wystąpi; MTTF jest także znana jako MTBF; czyli funkcja wartości średniej czasu pomiędzy kolejnymi wystąpieniami błędów (ang. mean time between failures).

Modele z dziedziną czasu, ze względu na typ danych w nich stosowanych, można podzielić na dwie główne klasy:

- modele używające pojęcia liczby błędów w pewnym okresie czasu
- modele używające pojęcia czasu pomiędzy kolejnymi wystąpieniami błędów.

Podejścia stosowane w obu modelach, chociaż inne, nie stanowią trwałego podziału, ponieważ istnieje możliwość zastosowania

modeli przeznaczonych dla jednego rodzaju danych, podczas gdy dysponuje się tylko drugim rodzajem danych. Oczywiście, w tym przypadku konieczne jest wykonanie dodatkowych przekształceń.

Funkcja odpowiadająca liczbie wykrytych błędów w czasie może być funkcją wklęsłą lub S-kształtną. W drugim przypadku zakłada się, że w początkowym okresie czasu ma się do czynienia z procesem uczenia się, gdy tempo wykrywania błędów jest dużo mniejsze niż w późniejszej fazie testowania. Pomijając ten przedział czasu, funkcje obu rodzajów wykazują jednakowe zachowanie, tzn. w miarę zwiększania się liczby wykrytych błędów spada tempo ich detekcji (oprogramowanie staje się mniej zawodne). Asymptotą wykresu takiej funkcji może być prosta odpowiadająca całkowitej liczbie błędów w programie.

Modelem niezawodności jest pewna funkcja, której parametry należy oszacować na podstawie danych z testów. Estymacja może odbywać się np. poprzez wstawienie danych do równań, w których umieszczone są parametry. Najczęstszą taką bezpośrednią metodą jest metoda maksymalnego prawdopodobieństwa. Metoda pośrednia zakłada dopasowanie danych do krzywej funkcji i estymację parametrów poprzez najlepsze dopasowania do krzywej. Metoda najmniejszych kwadratów jest tu dominującą metodą. Metoda maksymalnego prawdopodobieństwa może być bardzo złożona, ponieważ prowadzi do układu wielu równań, którego rozwiązanie jest skomplikowane.

W roku 1983 Musa i Okumoto zaproponowali sposób podziału modeli, bazując na ich różnych atrybutach:

- sposób reprezentowania czasu – czas kalendarzowy lub czas wykonania
- całkowita liczba błędów, jakie mogą pojawić się w nieskończonym przedziale czasu – skończona lub nieskończona
- rozkład wystąpień błędów w określonym czasie – dwa najważniejsze to rozkład Poissona i rozkład dwumianowy
- forma funkcji intensywności występowania błędów.

Rozkład Poissona i rozkład dwumianowy odgrywają kluczową rolę w klasyfikacji modeli wzrostu niezawodności oprogramowania, a wykorzystanie niejednorodnego procesu Poissona wydaje się najbardziej praktycznym i użytecznym wyborem dla modelowania niezawodności oprogramowania. Fakt ten wynika z wielu doświadczeń. Modele oparte na rozkładzie dwumianowym są modelami ze skończoną liczbą błędów, tzn. zakładają, że skończona liczba błędów ujawni się w nieskończonym czasie. Natomiast modele oparte na rozkładzie Poissona mogą być modelami zarówno ze skończoną, jak i teoretycznie nieskończoną liczbą błędów, w zależności od tego, jak zostaną określone.

Zdecydowana większość modeli bazuje na teorii procesów Markowa. W zakresie tych modeli można podać przykłady pogrupowane ze względu na rozkład wystąpień błędów:

a) z rozkładem Poissona:
   - Crow (1974)
   - Musa (1975)
   - Moranda (1975, 1979)
   - Schneidewind (1975)
   - Goel i Okumoto (1979)
   - Brooks i Motley (1980)
   - Angus i współpracownicy (1980)
   - Yamada i współpracownicy (1983, 1984)
   - Yamada i Osaki (1984)
   - Ohba (1984),
b) z rozkładem dwumianowym:
   - Jelinski i Moranda (1972)
   - Shooman (1972)
   - Schick i Wolverton (1973, 1978)
   - Wagoner (1973)
   - Goel (1988)
   - Shanthikumar (1981)
   - Littlewood (1981),
c) z rozkładem innego typu:
   - Shooman i Trivedi (1975)
   - Kim i współpracownicy (1982)
   - Kremer (1983)
   - Laprie (1984)
   - Shanthikumar i Sumita (1986).

Najbardziej popularnymi w literaturze oraz odgrywającymi kluczową rolę w obszarze badania niezawodności oprogramowania są modele wykładnicze. Używając klasyfikacji Musa – Okumoto, ta grupa zawiera wszystkie modele o skończonej liczbie błędów, gdzie funkcja intensywności ich pojawiania się ma postać wykładniczą. Modele z rozkładem dwumianowym zaliczające się do tej kategorii charakteryzują się tym, że funkcja ryzyka ($z_I(t) = \mathrm{F}$) jest stała w odniesieniu do indywidualnego błędu i zależy od liczby błędów pozostałych w programie (N-(i-1)). Natomiast funkcja intensywności występowania błędów ma postać wykładniczą ($\mathrm{l}(t) = N\,\mathrm{fexp}(-\mathrm{f}t)$). Modele z rozkładem Poissona zaliczające się do tej kategorii charakteryzują się również tym, że funkcja ryzyka ($z_I(t) = \mathrm{f}$) jest stała w odniesieniu do indywidualnego błędu oraz tym, że czas do wystąpienia awarii spowodowanej takim błędem ma postać

wykładniczą ($f_X(x) = N\,\mathrm{f}\exp(-\mathrm{f}x)$). Zarówno dla jednorodnych, jak i niejednorodnych procesów Poissona, liczba błędów, jakie wystąpią w dowolnym, określonym przedziale czasu, jest zmienną losową o rozkładzie Poissona. Dla modeli określających odstępy czasu pomiędzy kolejnymi wystąpieniami błędów wykorzystywany jest rozkład wykładniczy.

W pierwszych modelach niezawodności oprogramowania opracowanych przez Jelinskiego i Morandę oraz Shoomana czas pomiędzy kolejnymi awariami jest opisywany rozkładem wykładniczym z parametrem proporcjonalnym do liczby błędów pozostałych w oprogramowaniu − np. czas średni pomiędzy błędami w momencie t jest równy $1/\mathrm{f}(N-(i-1))$, gdzie t oznacza dowolny moment czasu pomiędzy wystąpieniem i-pierwszego oraz i-tego błędu, f jest współczynnikiem proporcjonalności, natomiast N jest całkowitą liczbą błędów w oprogramowaniu w momencie rozpoczęcia analizy. Gdy ma miejsce wystąpienie błędu, wówczas funkcja ryzyka zmniejsza się proporcjonalnie o stałą wartość współczynnika f. To pokazuje, że usunięcie każdego błędu ma taki sam wpływ na całkowitą niezawodność oprogramowania. W klasyfikacji Musa – Okumoto jest to model typu dwumianowego. W modelach tych przyjęto następujące założenia:

- wszystkie błędy występujące w programie mają taki sam wpływ na całkowitą niezawodność. W dowolnym momencie czasu intensywność wystąpienia błędu zależy więc wprost proporcjonalnie od liczby błędów pozostałych w programie
- częstotliwość ujawniania się błędów pozostaje stała w przedziałach pomiędzy kolejnymi ich wystąpieniami
- wykrycie błędu jest równoznaczne z jego usunięciem i nie ma możliwości wprowadzenia przy tej okazji nowego błędu
- program będzie użytkowany w podobnych warunkach jak te, w których ma miejsce określanie jego niezawodności
- każdy błąd ma to samo prawdopodobieństwo ujawnienia się i ma taki sam negatywny wpływ na program, jak pozostałe błędy
- ujawnianie się błędów odbywa się niezależnie.

Ostatnie trzy założenia są wspólne również dla innych podstawowych modeli. Przykładami modeli, które − pomijając kwestię notacji − są bądź identyczne, bądź też są bardzo bliskimi przybliżeniami modelu wykładniczego, to modele, których autorami są Musa (1975), Schneidewind (1975) oraz Goel i Okumoto (1979).

Innego rodzaju modelami, zasadniczo różniącymi się pod względem pewnych deterministycznych ustaleń mających miejsce w modelach z markowskim i wykładniczym podejściem, są modele wykorzystujące teorię Bayesa. Na przykład, w podejściu wykładniczym w większości przypadków zakłada się, że każdy pojedynczy błąd ma jednakową wagę w odniesieniu do funkcji intensywności błędów w programie. Natomiast w podejściu bayesowskim kwestionuje się to założenie, zakładając, że waga pojedynczego błędu może być uwzględniona w modelu (Littlewood, 1981). Poprzednio omawiane modele zakładają także zmianę w niezawodności jedynie w momencie ujawnienia się błędu. W tym podejściu natomiast dopuszcza się subiektywny punkt widzenia, np. w przypadku, gdy przez pewien okres czasu, kiedy program jest poddawany badaniu, żaden błąd nie zostaje ujawniony. Pozwala to na przyjęcie założenia o wzroście niezawodności, odzwierciedlając w ten sposób rosnące przekonanie użytkownika o poprawności działania programu. Można zatem uznać, że niezawodność jest odzwierciedleniem zarówno liczby wykrytych błędów, jak i przedziałów czasu, gdy żaden błąd się nie ujawnia.

Zgodnie z przekonaniem, że różne błędy mają inny wpływ na niezawodność programu, ogólna liczba błędów nie jest tak istotna, jak ich znaczenie. Takie podejście wydaje się bardziej adekwatne z punktu widzenia praktycznego. Jeżeli na przykład istnieje program, który ma kilka błędów w rzadko używanej części kodu, oraz program, który ma tylko jeden błąd, ale w często wykorzystywanej części kodu, to w modelu bayesowskim nie można powiedzieć, że program pierwszy jest mniej niezawodny od programu drugiego. Ważniejsze jest spojrzenie na działanie całego kodu, aniżeli szacowanie ogólnej liczby błędów w nim zawartych. Z tego powodu średnia wartość czasu do wystąpienia błędu (MTTF) naturalnie stała się bardzo istotną statystyką w tym podejściu. Charakterystyczną cechą tego podejścia jest także to, aby wykorzystywać dane historyczne, np. z wcześniejszych i podobnych projektów, do szacowania niezawodności obecnie badanego programu. Jest to jednocześnie duże wyzwanie, aby umiejętnie skorzystać z doświadczeń przeszłości. Model Littlewood–Verrall (1973, 1974) jest prawdopodobnie najlepszym przykładem modelu tej klasy. W modelu tym przyjęto założenie, że program może stać się mniej niezawodny w procesie ewaluacji,

niż był poprzednio. Z powodu dopuszczenia niepewności co do ulepszenia programu w momencie wykrycia i usunięcia błędu, nowa wersja może być zarówno lepsza, jak i gorsza od poprzedniej. Odzwierciedlone jest to w ten sposób, że parametry, które definiują rozkład występowania błędów w czasie, są ustalane losowo. Rozkład występowania błędów jest, tak jak we wcześniejszych modelach, wykładniczy z pewną ustaloną intensywnością występowania błędów, ale ta intensywność jest tutaj losowa, a nie stała, jak poprzednio. Rozkład tej intensywności na podstawie np. danych historycznych może być rozkładem gamma. Odmianami tego modelu są np. modele, których autorami są Mazzuchi i Soyer (1988), Musa (1984) oraz Keiller i współpracownicy (1983).

Każdy z klasycznych modeli może stać się modelem wykorzystującym założenia Bayesa poprzez określenie odpowiedniego rozkładu dla przynajmniej jednego z parametrów modelu. Większość modeli wykorzystujących teorię Bayesa przyjmuje, jako punkt wyjścia, model wykładniczy np. Littlewood i Verrall (1974), Goel (1977), Littlewood (1980), Jewell (1985), Langberg i Singpurwalla (1985), Littlewood i Sofer (1987), Becker i Camarinopoulos (1990) oraz Csenki (1990). Występują jednakże modele całkiem nowe: Littlewood i Verrall (1973), Thompson i Chelson (1980), Kyparisi i Singpurwalla (1984) oraz Liu (1987). Podstawowym problemem w tego typu modelach jest ich złożoność oraz problem z doborem odpowiedniego rozkładu dla parametrów. Ponadto, większość programistów nie posiada wystarczającej wiedzy z zakresu statystyki, aby móc w pełni zrozumieć i korzystać z tego typu modeli. Zdecydowanie częściej spotykamy w praktyce zastosowania modeli klasycznych, których jest również znacznie więcej. Tym, co łączy wiele modeli klasycznych z modelami bayesowskimi jest idea, że wczesne korekcje błędów mają większy wpływ na intensywność błędów programu niż te, które są robione w późniejszym etapie.

Jak już wspomniano wcześniej, dobór odpowiedniego modelu dla posiadanego zbioru danych testowych nie jest oczywisty i łatwy. Różne modele mogą zwracać różne przewidywania dla tego samego zestawu danych o wystąpieniach błędów. Nie jest to zjawisko typowe tylko dla modeli wzrostu niezawodności oprogramowania, ale występuje także w innych modelach, które są wykorzystywane do ewaluacji pewnych wartości zmiennych w czasie. Co więcej, dany model może dać sensowne wyniki dla jednego zestawu danych o błędach i kontrowersyjne dla innego. Poszukiwania najlepszego modelu niezawodności oprogramowania rozpoczęły się na dobre pod koniec lat 70. i na początku lat 80. ubiegłego wieku. Początkowe wysiłki na gruncie porównawczym, Schick i Wolverton (1978) oraz Sukert (1979), nie dawały oczekiwanych rezultatów z powodu braku dobrej bazy w postaci informacji o błędach oraz braku zgody dotyczącej kryteriów, które powinny być wykorzystywane do takiej analizy porównawczej. Pierwszy niedostatek był uzupełniany sukcesywnie, a duży wkład w to miał Lyu, który w 1996 r. opublikował pięćdziesiąt, dobrej jakości, zbiorów danych o błędach. Zbiory te były tworzone pod specjalnym nadzorem i reprezentowały dane związane z różnego rodzaju aplikacjami, takimi jak systemy dowodzenia i kontroli czasu rzeczywistego, systemy komercyjne, wojskowe i kosmiczne. Zadanie wypracowania pewnego konsensusu w doborze kryteriów analizy porównawczej modeli zrealizował Iannino wraz ze współpracownikami. W 1984 r. zaproponowali oni następujące kryteria:

- przydatność modelu do przewidywania przyszłych awarii na podstawie znanych lub przyjętych charakterystyk oprogramowania np. oszacowanie linii kodu, języka użytego do programowania oraz obecnych i przeszłych błędnych zachowań (awarii)
- przydatność modelu do szacowania, z zadowalającą dokładnością, wielkości wykorzystywanych do planowania i zarządzania rozwojem oprogramowania w trakcie projektu lub podczas użytkowania systemu. Te wielkości to m.in. obecna intensywność błędów, data osiągnięcia pożądanej intensywności błędów, wielkość zasobów oraz koszty niezbędne do zrealizowania celu w postaci określonej intensywności błędów
- jakość założeń modelu np. dostępność danych, klarowność i precyzyjność
- stopień użyteczności modelu w zakresie oprogramowania różniącego się pod względem rozmiaru, struktury, funkcji, środowisk działania oraz przynależności do różnych faz w cyklu życia programu
- stopień prostoty modelu np. prostota użytych pojęć lub prosty i niedrogi sposób zbierania danych, czy implementacji w postaci oprogramowania użytkowego.

Zastosowanie podziału na podstawie zaproponowanych kryteriów może w pewien sposób zawęzić zakres poszukiwań odpowiedniego modelu. Jednakże istnieje wiele

czynników, które mogą wpływać na właściwości procesu pojawiania się błędów, a które nie są uwzględniane przez żaden z modeli. Najlepszą zalecaną metodą doboru odpowiedniego modelu jest sprawdzenie różnych wariantów na tym samym, posiadanym zbiorze danych. Ze względu na to, że proces taki jest żmudny, ma on rację bytu głównie dzięki narzędziom wspomagającym modelowanie niezawodności oprogramowania, takim jak CASRE, SMERFS, SRMP czy SREPT.

3. Ewolucja koncepcji modelowania niezawodności oprogramowania

Teoria modelowania niezawodności oprogramowania podlegała ciągłemu rozwojowi. Pojawiały się różne propozycje, w jaki sposób przekształcając istniejące modele lub stosując zupełnie nowe rozwiązania, dochodzić do miarodajnych rezultatów.

Podejście rekalibracji (Abdel − Ghaly i inni, 1986), czyli przewidywania adaptacyjnego miało doprowadzić do poprawy jakości przewidywań w modelach. Jest to statystyczna procedura, która umożliwia zmianę parametrów modelu na podstawie wcześniejszych błędów i produkowanie ulepszonych przewidywań, redukując poziom „zafałszowania" w modelu. Innym pomysłem jest kombinacja liniowa modeli (Lyu i Nikora, 1991, 1992), która nawet w najprostszej postaci daje bardziej adekwatne przewidywania, niż sam pojedynczy model. Model wczesnej predykcji oparty na fazach (Gaffney i Davis (1988)) proponuje wykorzystanie statystyk błędów sporządzonych w czasie technicznego przeglądu specyfikacji wymagań projektu, projektowania oraz implementacji do przewidywania niezawodności w czasie testów oraz późniejszej eksploatacji. Jedną z pierwszych i najbardziej znanych prób „przewidywania niezawodności" oprogramowania we wczesnej fazie produkcji była metoda opracowana przez naukowców pracujących dla Rome Air Development Center (1987). W swoim modelu opracowali oni metodę przewidywania gęstości błędów, która mogła być później przekształcona w innego rodzaju pomiary niezawodności, na przykład analizując intensywność błędów. Z kolei model, który zaproponowali Kapur i Garg (1992) uwzględnia w procesie usuwania błędów istnienie błędów zależnych.

Teoria wzrostu niezawodności wykorzystuje w procesie modelowania jedynie dane o błędach z badanego systemu, podczas gdy jego struktura jest ignorowana. Bliższe spojrzenie na architekturę systemu stało się interesujące w momencie rozpowszechnienia produkcji oprogramowania opartego na komponentach. Sprzyjający rozwój technologii sieciowych umożliwiających budowanie systemów rozproszonych oraz wykorzystywanie do programowania języków obiektowych ułatwiających logiczny podział funkcjonalności na odrębne części spowodował, że filozofia modularności w inżynierii oprogramowania stała się dominującym podejściem. Istotne znaczenie ma fakt badania systemu, w którym tylko część komponentów uległa modyfikacji bądź została napisana od nowa, a pozostała część nie zmieniła się. Pierwszy model uwzględniający modularność został stworzony przez Littlewooda (1979). Oczywiście, z czasem temat ten był dalej podejmowany. Smidts i Sova (1999) rozważali modelowanie uwzględniające architekturę systemu na potrzeby szacowania niezawodności oprogramowania, bazując na dekompozycji wymagań na funkcje i atrybuty programu. Kuball i współpracownicy (1999) wprowadzili model hierarchiczny do szacowania prawdopodobieństwa awarii na potrzeby systemu opartego na komponentach, wykorzystując podejście Bayesa. Lyu i współpracownicy (2002) sformułowali wymagania dotyczące zasobów dla etapu testowania oprogramowania opartego na komponentach, jako kombinatoryczną optymalizację problemu ze znanymi kosztami, niezawodnością, poniesionym wysiłkiem oraz innymi atrybutami komponentów. Inne prace na ten temat przedstawili m.in. Kubat (1989), Gokhale (1998), Ledoux (1999), Yamada (2000) oraz Okamura (2004). Ogólnie mówiąc, wprowadzenie parametrów strukturalnych do procesu inżynierii niezawodności dało kolejne możliwości rozwoju tego zagadnienia.

Parametryzacja modeli niezawodności oprogramowania może także bazować na alternatywnych źródłach informacji, takich jak metryki opracowane w modelach wczesnej predykcji. Innego rodzaju metryki mogą dotyczyć pokrycia testów oraz obciążenia systemu. Piwowarski i współpracownicy (1993) zaproponowali prosty model wzrostu niezawodności oprogramowania oparty na pokryciu testami. Malaiya i współpracownicy (1994) zaprezentowali model logarytmiczny, który uwzględnia wysiłek zespołu testowego w odniesieniu do pokrycia specyfikacji testów, co może bezpośrednio wpływać na pokrycie defektów występujących w programie, a zatem na wzrost jego niezawodności. Chen i współpracownicy (2001) włączyli do modelowania

pokrycie testów w postaci odniesienia się do czasu ich wykonania. Fujiwara i Yamada (2001) zaproponowali model uwzględniający charakterystykę przygotowanego zestawu testów systemu. W szczególności chodzi tutaj o umiejętności i wiedzę testerów projektujących scenariusze testowe. Większe doświadczenie wpływa na zwiększenie zakresu błędów, które potencjalnie mogą zostać wykryte przy wykorzystaniu wyselekcjonowanego zbioru testów. Warto w tym miejscu wspomnieć o tym, że zgodnie z założeniami inżynierii niezawodności oprogramowania testy powinny być pisane tak, aby odpowiadały profilowi operacyjnemu badanego systemu. Profil operacyjny jest zdefiniowany jako zbiór operacji, jakie oprogramowanie jest w stanie wykonać razem z prawdopodobieństwem, z jakim one wystąpią. Operacja jest grupą wywołań, która zazwyczaj pociąga za sobą podobne przetwarzanie.

Mówiąc o pokryciu testami, można wspomnieć także o modelowaniu uwzględniającym pokrycie kodu przez testy. Pokrycie rozumiane jest jako wykonanie przynajmniej jednokrotnie wyodrębnionych instrukcji programu lub przejście określonych ścieżek funkcjonalnych. Tego typu dane łączy się z danymi o wykrytych błędach. Modele prezentujące to podejście zaprezentowali Fujiwara i Yamada (2002), Malaiya (2002), Pham i Zhang (2003) oraz Inoue i Yamada (2004).

Innym podejściem jest modelowanie metryk jakości (zmiennych zależnych) na bazie ich powiązań z innymi zmiennymi niezależnymi, takimi jak rozmiar kodu, rozmiar danych, złożoność, operatory, operandy itp., do formułowania przewidywań o niezawodności i jakości oprogramowania. Agresti i Evanco (1992) podjęli próbę stworzenia modelu do przewidywania gęstości błędów, bazując na charakterystykach procesu produkcji dla języka Ada. Gokhale i Lyu (1997) zastosowali technikę analizy drzewa regresji do ustanowienia zależności modelowania pomiędzy zmiennymi zależnymi i niezależnymi. Schneidewind (2000) opracował metodę badania oprogramowania pozwalającą, dla celów kontroli jakości programu i późniejszego jego utrzymania, na rozróżnienie pomiędzy modułami „skłonnymi" i „nieskłonnymi" do błędów. Innym przykładem jest wykorzystanie danych w postaci złożoności cyklomatycznej, czy liczby linii kodu, w połączeniu z informacjami o błędach w kodzie do badania niezawodności oprogramowania sterującego satelitami (NASA JM1) przez NASA. Model użyty w tych badaniach opisał Schneidewind (2008).

Atrakcyjnym polem dla badań naukowych nad szacowaniem niezawodności oprogramowania są również techniki symulacji. Von Mayrhauser i współpracownicy (1993) wykonali eksperyment w celu sprawdzenia natury powiązań pomiędzy błędami programu a jego strukturą i zaproponowali technikę symulacji dla celów przewidywania niezawodności. Tausworthe i Lyu (1996) wykazali, że do prognozowania niezawodności oprogramowania można wykorzystywać metodę Monte Carlo. Ustanowili oni technikę symulacji pozwalającą odwzorować kompletny cykl rozwoju programu, włączając w to cykl życia błędów oraz awarii. Gokhale i współpracownicy (1998) rozwinęli dalej tę technikę symulacji dla potrzeb analizy niezawodności systemów opartych na komponentach w różnych architekturach i konfiguracjach.

Najprostszym założeniem badań nad procesem wykrywania i usuwania błędów jest to, że do wykrycia poszczególnych błędów wymagane są podobnej skali wysiłek oraz strategia testowania. W praktyce jednak może to nie być prawda. Aby odzwierciedlić to zróżnicowanie, błędy mogą zostać sklasyfikowane w różne kategorie, określające trudność ich wykrycia. W ten sposób tempo wykrywania i usuwania błędów różnej kategorii jest różne. W zmodyfikowanym modelu wykładniczym Yamada (1983) założył istnienie dwóch rodzajów błędów. Pham (1993) zaproponował model z wieloma typami błędów. Kapur (1995) także poczynił tego rodzaju założenia w swoim modelu, gdzie czas pomiędzy wystąpieniem błędów oraz czas potrzebny na ich usunięcie jest zależny od wagi błędu. Ten sam autor w innej pracy (2000) zaproponował kwalifikację błędów do różnych kategorii na podstawie czasu wykrycia poszczególnego błędu. Kolejną kwestią jest to, czy tempo wykrywania błędów powinno pozostać stałe przez cały proces testowania. Okazuje się, że wiele czynników, takich jak strategia testowania, zmiana środowiska testowego, skład zespołów testujących oraz motywacja testerów, przekładają się bezpośrednio na proces detekcji i usuwania błędów. Zmiana ta może być analizowana przy użyciu koncepcji podziału czasu testowania na przedziały, zakładając, że w poszczególnym przedziale strategia oraz środowisko testowania są mniej więcej takie same, przy tym różne od pozostałych przedziałów. Tempo detekcji błędów (alternatywnie tempo usuwania błędów) jest

w przedziale stałe lub jest funkcją czasu testowania i różni się od analogicznych wielkości w innych przedziałach. Koncepcja ta zapoczątkowana została przez Zhao (1993), a później rozwijana była przez różnych autorów: Chang (2001), Chen (2001), Shyur (2003), Zou (2003), Kapur (2006) i Gupta (2008). Inne modele, uwzględniające zmieniający się stopień wysiłku wkładanego w testowanie, to np. Yamada (1991, 1993), Bokhari (2006), Kapur (2004), Kuo (2001) oraz Huang (2007).

Odejście od ogólnej liczby błędów, jako wyznacznika niezawodności oprogramowania, zaproponowali Sawada i Sandoh (1999). W ich propozycji oprogramowanie poddawane jest testowaniu w rozumieniu testów demonstracyjnych i na tym etapie zbierane są informacje dotyczące liczby ujawnionych błędów oraz ich negatywnych skutków dla działania programu na danym etapie produkcji oprogramowania. Jest to swego rodzaju testowanie prototypów z sukcesywnie zwiększanym zakresem funkcjonalności. Przed każdą demonstracją zakłada się określony limit liczby błędów oraz ich skutków, który decyduje o akceptacji lub odrzuceniu produktu na tym etapie. Obliczanymi wielkościami pomocniczymi, przy użyciu narzędzi statystycznych, są także stopień ryzyka producenta oraz klienta. Koncepcja zaproponowana przez japońskich badaczy wydaje się użyteczna z punktu widzenia nowoczesnych technik produkcji oprogramowania np. SCRUM, gdzie poszczególne etapy, zwane sprintami, mają na celu uzyskanie kompletnej, pod pewnymi względami, wersji programu, którą można przedstawić klientowi.

Tradycyjne modele wzrostu niezawodności oprogramowania polegają w znacznej mierze na danych historycznych z wcześniejszych etapów testowania. Te informacje mogą nie być reprezentatywne ze względu na zmiany w środowisku testowym, zróżnicowanie strategii testowania itp. Xie, Hong i Wohlin (1997) podali metodę wykorzystania techniki „wygładzania wykładniczego" do prognozowania niezawodności. W tym podejściu dużo większą wagę mają informacje bieżące, co pozwala m.in. wyeliminować negatywny wpływ początkowej fazy testowania, gdy system jeszcze nie jest stabilny. Dodatkowe zalety zaproponowanego modelu to intuicyjność jego parametrów oraz niewielkie wymagania odnośnie do pracochłonności obliczeń.

Tradycyjne modele wzrostu niezawodności oprogramowania wykorzystywane są do modelowania jedynie procesu pojawiania się (wykrywania) błędów z założeniem, że usuwanie błędu odbywa się natychmiastowo i niezawodnie tzn. nie dopuszcza się błędnej naprawy oraz wprowadzenia przy tej okazji kolejnych błędów. Założenie takie odbiega dalece od rzeczywistości w projektach programistycznych. Z upływem czasu badacze zaczęli bliżej przyglądać się i tej aktywności, wprowadzając do modelowania element zawodności korekcji błędów. Pierwszymi, którzy wprowadzili niedoskonałe debugowanie, byli Goel (1985) oraz Kapur i Garg (1990). Pierwszymi, którzy wprowadzili generację błędów (poprawne usunięcie wykrytego błędu ale wprowadzenie przy tej okazji innych), byli Ohba i Chou (1989). Później rozwijali to podejście m.in. Yamada (1992), Kapur (1996), Pham (2000, 2006), Shneidewind (2001), Shyur (2003) oraz Chatterjee (2004). W praktyce istnieją dwa powody niedoskonałego debugowania. Pierwszy to niemożność skutecznego usunięcia istniejącego błędu, ponieważ istnieje pewne prawdopodobieństwo, że właśnie tak będzie (Goel − Okumoto – 1979, Yamada – 1993). Drugi to wprowadzanie nowych błędów, ponieważ istnieje pewne prawdopodobieństwo, że właśnie tak będzie. Można także przyjąć założenie, że całkowita liczba błędów w programie jest rosnącą funkcją czasu (Ohba – Chou – 1989, Yamada – 1992, Pham – Zhang – 1997). Powstały również prace łączące w swoich założeniach oba wspomniane powyżej powody (Zeephongsekul – 1994, Pham – 1995). Analiza czasu i efektów związanych z tzw. debugowaniem tworzy zupełnie osobny proces. W ten sposób model powinien uwzględniać dane dotyczące zarówno procesu detekcji, jak i korekcji błędów. Proces korekcji błędów można traktować jako opóźniony w czasie proces detekcji błędów, ponieważ błąd może zostać naprawiony jedynie po jego wykryciu, czyli może to być np. niejednorodny proces Poissona. Opóźnienie odpowiada czasowi usuwania błędu i może być wartością stałą lub losową.

Modele zakładające probabilistyczny charakter procesu detekcji błędów zostały skrytykowane przez Cai (1991). Uznał on, że unikatowość oprogramowania powinna determinować fakt stosowania logiki rozmytej do modelowania niezawodności, jako że proces ten jest rozmyty z natury. Za teorią tą ma przemawiać przede wszystkim argument, że proces debugowania nie jest w żadnym wypadku powtarzalny w rozumieniu teorii prawdopodobieństwa i żaden ogromny zbiór próbek nie gwarantuje posiadania wystarczających

informacji do zaproponowania dobrej prognozy. Model realizujący to nowe podejście, model Cai – Wen – Zhang'a, został zaproponowany w 1993 r. Elementy teorii zbiorów rozmytych były wykorzystane także w późniejszym czasie, np. Utkin i Gurov (2002), gdzie dodatkowo uwzględniono niedoskonałe debugowanie oraz usuwanie za jednym razem więcej niż jednego błędu (błędy zależne). W innej swojej pracy (2006) Cai zakwestionował stosowanie niejednorodnego procesu Poissona (NHPP – ang. non-homogeneous Poisson process), jako sposobu opisu procesu detekcji błędów podczas testowania oprogramowania. W przypadku przyjęcia takiego założenia, wartość oczekiwana i wariancja liczby wystąpień błędów do czasu t są sobie równe, co wynika z własności rozkładu Poissona. Cai zaprezentował rezultaty eksperymentu przeprowadzonego na oprogramowaniu ze znaną liczbą błędów. Okazało się, że oszacowania dla wartości oczekiwanej oraz wariancji zupełnie się nie pokrywały, co świadczyłoby o błędnym założeniu odnośnie do NHPP. W tej samej pracy również podejście markowskie, równie szeroko stosowane w modelowaniu niezawodności, zostało skrytykowane. Zgodnie z tym podejściem przejście pomiędzy kolejnymi stanami w czasie działania programu, gdzie w przypadku badania niezawodności przejście następuje w momencie ujawnienia się błędu, jest realizacją procesu Markowa. Na podstawie eksperymentu stwierdzono luki w tej hipotezie. Wykryte nieścisłości nie powodują jednak potrzeby odwrotu od stosowanych szeroko technik, a jedynie pokazują, że nie mają one charakteru uniwersalnego.

W modelach z NHPP funkcja intensywności jest ciągła w czasie. Główny argument przemawiający za używaniem NHPP to prostota. Podstawową informacją jest wartość oczekiwana liczby błędów. Założenie o ciągłości jest nierealistyczne, gdyż proces debugowania powinien wywoływać przerwy. Dodatkowo, oprogramowanie nie starzeje się, więc jeśli nie jest modyfikowane, to jego podatność na awarie nie zmienia się. Dlatego intensywność awarii pomiędzy debugowaniami powinna być stała. Niedoskonałość modeli skończonej liczby błędów z niejednorodnym procesem Poissona dla niektórych zbiorów danych, spowodowała poszukiwania modeli wykorzystujących innego rodzaju teorie. W ten sposób powstały m.in. modele wykorzystujące rozkład logistyczny (Gokhale i Trivedi), czy też rozkład hipergeometryczny (Hou – 1995, Tohma – 1989).

Innym, pojawiającym się elementem w budowie modeli wzrostu niezawodności oprogramowania są sieci neuronowe. Jako pierwszy koncepcję taką zaprezentował Karunanithi (1991), a później swoje badania na tym polu prezentowali także m.in. Khoshgoftar (1992, 1993, 1996), Guo i Lyu (2000), Cai (2001), Tiang (2004), Su (2005, 2007) i Kapur (2008). Modele sieci neuronowych zostały wykorzystane do określania atrybutów jakości oprogramowania, takich jak niezawodność, lub też np. do detekcji kodu podatnego na błędy. Niektórzy autorzy wprowadzili do swoich rozważań rozróżnienie złożoności błędów i niedoskonałe debugowanie oraz proponowali różne postacie architektury samej sieci. Sieć neuronowa ma potencjał do uzyskiwania miarodajnych danych na podstawie skomplikowanych i nieprecyzyjnych informacji wejściowych. Stanowi doskonałe narzędzie do analizy procesu pojawiania się błędów, którego charakter, zwłaszcza przy redukcji upraszczających założeń, nie jest w ogóle intuicyjny i prosty do odczytania przez człowieka. Doświadczenia pokazały, że jest to dobra alternatywa dla tradycyjnych modeli parametrycznych.

Modele markowskie powstały na bazie założenia, że pojawienie się w procesie debugowania kolejnego błędu jest uwarunkowane stanem oprogramowania w danym momencie. Stany poprzednie, wynikające z faktu wykrywania poprzednich błędów, można w tym momencie pominąć. Jest to zatem proces bez pamięci. Różne założenia, do parametrów procesu, zaowocowały różnymi modelami: Jeliński − Moranda (1972), Moranda (1979), Littlewood − Verral (1973) oraz Gaudion (1994, 1999). Dane dotyczące niezawodności dotyczą zazwyczaj kolejnych czasów pomiędzy wystąpieniami błędów lub skumulowanej liczby błędów w danej chwili. Informacje o tym, czy naprawa miała miejsce po fakcie wystąpienia błędu, czy też zostało to odłożone na późniejszy czas, oraz czy naprawa jest „kosmetyczna", czy poważna, nie są zazwyczaj zbierane, a mogą stanowić wartościowe dane w procesie modelowania niezawodności. Tym zagadnieniem zajmują się modele wykorzystujące teorię ukrytych łańcuchów Markowa, np. model zaproponowany przez Duranda i Gaudoina (2003, 2005).

Relatywnie nowym podejściem jest też wykorzystanie do modelowania niezawodności oprogramowania mieszanego rozkładu Poissona. Ogólnie mówiąc, dobrze znane modele wykorzystujące NHPP (Goel − Okumoto, Yamada, Ohba − Osaki) bazują na stochastycz-

nych procesach zliczających, opisujących liczbę defektów wykrytych w czasie testowania. Jednakże proces debugowania nie jest w rzeczywistości tak prosty, aby opisywać go w ten sposób. Spowodowane jest to zależnościami i związkami pomiędzy błędami. W tym kontekście procesy markowskie mogą stanowić alternatywę, ale z punktu widzenia obliczeń statystycznych jest to znaczące utrudnienie. Zmodyfikowanie NHPP może dać lepsze rezultaty.

4. Podsumowanie

Modele wzrostu niezawodności oprogramowania służą generalnie do ustalania obecnej oraz przewidywania przyszłej niezawodności oprogramowania. Informacje uzyskiwane dzięki zastosowaniu modelowania niezawodności oprogramowania mogą mieć istotne znaczenie w wielu przypadkach podejmowania decyzji w procesie produkcji oprogramowania, takich jak analiza kosztów, alokacja zasobów do testów, czy data zakończenia testowania. Spośród technik mających największy wpływ na modelowanie niezawodności oprogramowania wyróżnić należy wykorzystanie niejednorodnego procesu Poissona. Dużą rolę w tym odgrywa jeden z pierwszych modeli – Goel i Okumoto (1979). Niektóre badania udowodniły, że modele proste w implementacji mogą być jednakowo skuteczne jak modele złożone, uwzględniające dużo więcej ważnych aspektów.

W obecnym czasie pojawiają się głosy, że modele wzrostu niezawodności opracowywane, gdy w procesie produkcji oprogramowania dominował model kaskadowy, są mało użyteczne dla nowoczesnych, lekkich (ang. agile) technik prowadzenia projektów. Spowodowane jest to pojawiającymi się trudnościami w procesie szacowania parametrów modeli, z powodu braku odpowiednich danych lub też odpowiedniej ich liczby.

Naturalnym wnioskiem płynącym z obserwacji procesu wzrostu niezawodności podczas testowania jest to, że im dłużej oprogramowanie poddawane jest testowaniu, tym lepszą jego jakość udaje się uzyskać. Nadgorliwe testowanie zwiększa jednak niepotrzebnie koszty projektu i opóźnia wprowadzenie produktu na rynek. Krótki czas udostępnienia oprogramowania klientowi (ang. time-to-market) jest dzisiaj czynnikiem bardzo pożądanym. Przy dynamicznie zmieniających się warunkach i potrzebach rynku produkcja oprogramowania, choćby najbardziej niezawodnego, ale niespełniającego wymagań czasowych, jest nieopłacalna, bo może skutkować brakiem klientów. Jest to duże wyzwanie dla firm realizujących projekty, które niejednokrotnie muszą zmieniać nieadekwatną aktualnie metodykę prowadzenia projektów. Z drugiej strony udostępnienie oprogramowania zawierającego istotne błędy, które ujawnią się dopiero w środowisku klienta, jest porażką dla firmy. Takie błędy są dużo bardziej kosztowne, bo dodatkowo obniżają zaufanie do dostawcy i jego reputację. Jest to kolejne poważne wyzwanie, aby umiejętnie określić moment udostępnienia oprogramowania do powszechnego użytku. Badania nad optymalnym momentem zakończenia procesu testowania prowadzili m.in. Yamad (1985), Brown (1989), Ohter i Yamad (1990), Ehrlich (1993), Hou (1997), Pham (1999, 2004), Rinsak (2004) oraz Huang (2006).

Inżynieria niezawodności oprogramowania daje również wskazówki dotyczące istotnych działań, jakie należy podejmować w każdej z faz cyklu życia oprogramowania, w celu zapewnienia wysokiej jakości produktu (Donelly, Everett, Musa, Wilson, 1996). Liczące się na świecie przedsiębiorstwa, takie jak np. AT&T, wdrożyły dla swoich potrzeb produkcyjnych sposób postępowania zgodny z określonymi wskazówkami, które można opisać następująco:

a) faza definiowania wymagań i ich analizy:
   - określenie profilu funkcyjnego systemu
   - zdefiniowanie i sklasyfikowanie potencjalnych awarii
   - określenie potrzeb niezawodnościowych z punktu widzenia klienta
   - przeprowadzenie badań rynku
   - ustalenie założeń odnośnie niezawodności.
b) faza projektowania i implementacji:
   - klasyfikacja komponentów pod względem wymagań niezawodności
   - implementacja pod kątem spełnienia założeń niezawodności
   - alokacja zasobów zgodnie z profilem funkcyjnym
   - zarządzanie procesem obsługi pojawiających się błędów
   - śledzenie wartości niezawodności uzyskanego oprogramowania.
c) faza testowania:
   - określenie profilu operacyjnego systemu,
   - przeprowadzenie testu wzrostu niezawodności
   - śledzenie postępów testowania
   - weryfikacja spełnienia założeń niezawodności.

d) faza obsługi po dostarczeniu do klienta:
   - monitorowanie niezawodności względem założeń
   - śledzenie zadowolenia klienta z niezawodności oprogramowania
   - planowanie ulepszeń produktu oraz samego procesu.

Na zakończenie warto przytoczyć słowa pewnego statystyka, który poczynił znaczący wkład m.in. w obszarze badania niezawodności. George E.P. Box powiedział: „Zasadniczo, wszystkie modele są złe, ale niektóre są użyteczne".

5. Bibliografia

[1] M.R. Lyu, *Handbook of software reliability*, IEEE Computer Society Press, 1996.

[2] Ch.H. Lee, Y.T. Kim, D.H. Park, „S-shaped software reliability growth models derived from stochastic differential equations", *IIE Transactions*, Vol. 36, 1193-1199 (2004).

[3] Kai-Yuan Cai, „Software Reliability Experimentation and Control", *Journal of Computer Science and Technology,* Vol. 21, No. 5 (2006).

[4] W. Everett, S. Keene, A. Nikora, „Applying Software Reliability Engineering in the 1990s", *IEEE Transactions on Reliability*, 1998.

[5] P. K. Kapur, A. Kumar, K. Yadav, S. K. Khatri, „Software reliability growth modelling for errors of different severity using change point", *International Journal of Reliability, Quality and Safety Engineering*, Vol. 14, No. 4 (2007).

[6] K. Sawada, H. Sandoh, „Software Reliability Demonstration Testing with Consideration of Damage Size of Software Failures", *Electronics and Communications in Japan*, (1999).

[7] M. Xie, G. Y. Hong, C. Wohlin, „A study of the exponential smoothing technique in software reliability growth prediction", *Quality and Reliability Engineering International*, Vol. 13, No. 6, 347-353 (1997).

[8] M. Xie, Q. P. Hu, Y. P. Wu, S. H. Ng, „A Study of the Modeling and Analysis of Software Fault-detection and Fault--correction Processes", *Quality and Reliability Engineering International*, Vol. 23, No. 4, 459-470 (2007).

[9] L. V. Utkin, S. V. Gurov, „A fuzzy software reliability model with multiple-error introduction and removal", *International Journal of Reliability, Quality and Safety Engineering*, Vol. 9, No. 3, 215-227 (2002).

[10] A. Yadav, R. A. Khan, „Critical Review on Software Reliability Models", *International Journal of Recent Trends in Engineering*, Vol. 2, No. 2, 114-116 (2009).

[11] S. Yamada, K. Sera, „Imperfect Debugging Models with Two Kinds of Software Hazard Rate and Their Bayesian Formulation", *Electronics and Communications in Japan*, (2001).

[12] S. Yamada, „Software Reliability Growth Models Incorporating Imperfect Debugging with Introduced Faults", *Electronics and Communications in Japan*, (1998).

[13] J-Y. Park, Y-S.Hwank, T. Fuiwara „Integration of imperfect debugging in general testing-domain dependent NHPP SRGM", *International Journal of Reliability, Quality and Safety Engineering*, Vol. 12, No. 6, 493-505 (2005).

[14] P. Zeephongsekul, W. Bodhisuwan, „On a generalized dual process software reliability growth model", *International Journal of Reliability, Quality and Safety Engineering*, Vol. 6, No. 1, 19-30 (1999).

[15] K. Esaki, M. Takahashi, „A model for program error prediction based on testing characteristics and its evaluation", *International Journal of Reliability, Quality and Safety Engineering*, Vol. 6, No. 1, 7-18 (1999).

[16] P. K. Kapur, O. Singh, R. Mittal, „Software reliability growth and innovation diffusion models: an interface", *International Journal of Reliability, Quality and Safety Engineering*, Vol. 11, No. 4, 339-364 (2004).

[17] F. Padberg, „Maximum likelihood estimates for the hypergeometric software reliability model", *International Journal of Reliability, Quality and Safety Engineering*, Vol. 10, No. 1, 41-53 (2003).

[18] P. K. Kapur, S. K. Khatri, M. Basirzadech, „Software reliability assessment using artificial neural network based flexible model incorporating faults of different complexity", *International Journal of Reliability, Quality and Safety Engineering*, Vol. 15, No. 2, 113-127 (2008).

[19] L. Tian, A. Noore, „Software reliability prediction using recurrent neural network with Bayesian resularization", *International Journal of Neural Systems*, (2004).

[20] J. Zheng, „Predicting software reliability with neural network ensembles", *Expert Systems with Applications*, (2009).
[21] S. S. Gokhale, „Software failure intensity, reliability and optimal stopping time incorporating repair policies", *International Journal of Reliability, Quality and Safety Engineering*, Vol. 13, No. 5, 455-477 (2006).
[22] P. J. Boland, H. Singh, „Determining the optimal release time for software in the geometric Poisson reliability model", *International Journal of Reliability, Quality and Safety Engineering*, Vol. 9, No. 3, 201-213 (2002).
[23] X. Zhang, H. Pham, „Comparison of nonhomogeneous Poisson process software reliability models and its application', *International Journal of System Science*, (2000).
[24] K. Worwa, *Modelowanie i ocena wzrostu niezawodności oprogramowania w procesie testowania*, WAT, 2005.
[25] S.H. Khan, *Metryki i modele w inżynierii jakości oprogramowania*, Wydawnictwo Naukowe PWN SA, 2006.
[26] M.R. Lyu, „Software Reliability Engineering: A Roadmap", *IEEE Computer Society*, (2007).
[27] S. Chatterjee, S.S. Alam, R.B. Misra, „Sequential Bayesian technique: An alternative approach for software reliability estimation", *Sadhana*, Vol. 34, Part 2 (2009).
[28] T.M. Khoshgoftaar, T.G. Woodcook, „Software reliability model selection", *Quality and Reliability Engineering International*, (1992).
[29] M.R. Lyu, *Software Reliability Theory*, John Wiley & Sons, Inc., 2002.
[30] R.I. Zequeira, „A model for Bayesian software reliability analysis", *Quality and Reliability Engineering International*, (2000).
[31] M. Kimura, S. Yamada, S. Osaki, *Statistical Software Reliability Prediction and Its Applicability Based on Mean Time between Failures*, Elsevier Science Ltd., 1995.
[32] K. Sawada, H. Sandoh, „A summary of software reliability demonstration testing models", *International Journal of Reliability, Quality and Safety Engineering*, (1999).
[33] S. Ramani, S.S. Gokhale, K. S. Trivedi, „SREPT: software reliability estimation and prediction tool", *Performance evaluation* 39, 2000.
[34] T. Fujiwara, S. Yamada, „Software Reliability Growth Modeling Based on Testing-Skill Characteristics: Model and Application", *Electronics and Communication in Japan*, (2001).
[35] S. Yamada, Y. Tamura, M. Kimura, „A Software Reliability Growth Model for a Distributed Development Environment", *Electronics and Communications in Japan*, (2000).
[36] H. Okamura, S. Kuroki, T. Dohi, S. Osaki, „A Reliability Growth Model for Modular Software", *Electronics and Communications in Japan*, (2004).
[37] H. Tanaka, S. Yamada, S. Osaki, „Software Reliability Growth Model with Continuous Error Domain – Application of a Linear Stochastic Differential Equation", *Electronics and Communications in Japan*, (1992).
[38] A. Wood, *Software Reliability Growth Models*, Tandem Computers, 1996.
[39] H. Okamura, T. Dohi, „Software reliability modeling based on mixed Poisson distribution", *International Journal of Reliability, Quality and Safety Engineering*, vol. 15, No. 1, 19-32 (2008).
[40] J-B. Durand, O. Gaudoin, „Software reliability modelling and prediction with hidden Markov chains", *Statistical Modelling*, No. 5(1), 75-93 (2005).
[41] S. Inoue, S. Yamada, „Testing-coverage dependent software reliability growth modeling", *International Journal of Reliability, Quality and Safety Engineering*, Vol. 11, No. 4, 303-312 (2004).
[42] S. Yamada, T. Fujiwara, „Testing-domain dependent software reliability growth models and their comparison of goodness--of-fit", *International Journal of Reliability, Quality and Safety Engineering*, Vol. 8, No. 3, 205-218 (2001).
[43] T. Fujiwara, S. Yamada, „A Testing--domain dependent software reliability growth model for imperfect debugging environment and its evaluation of goodness--of-fit", *Electronics and Communications in Japan*, (2003).
[44] A. Gupta, R. Kapur, P. C. Jha, „Considering testing efficiency and testing resource consumption variations in estimating software reliability", *International Journal of Reliability*, *Quality and Safety Engineering*, Vol. 15, No. 2, 77-91 (2008).
[45] O. Gaudion, „Software reliability models with two debugging rates", *International*

Journal of Reliability, *Quality and Safety Engineering*, Vol. 6, No. 1, 31-42 (1999).
[46] N. Schneidewind, „Complexity-driven reliability model", *International Journal of Reliability, Quality and Safety Engineering*, Vol. 15, No. 5, 479-494 (2008).

Software reliability growth models

R. PEŁKA

History of research on reliability of software began on early seventies of the last century. A significant progress of the work aimed at construction of a mathematical model of software reliability growth has been performed since the first publication devoted to this subject was presented. Analysis of existing literature may lead to the conclusion that there is no universal solution which could be applied in every single case. However, it is possible to classify existing models, based on their characteristics such as data domain, way to describe faults discovered during testing process, way to express reliability, or other remaining assumptions, including mathematical concepts used in evaluation process. This article presents an overview of existing solutions related to software reliability modeling, focusing on variety of aspects and methods used within this process.

Keywords: modeling, software, software reliability