


Miary niezawodności oprogramowania



Dr hab. inż. Ilona Bluemke

[Miary niezawodności oprogramowania]

- Ewoluowały z miar sprzętowych.
- Błędy **sprzętowe** są często **stałe** – aż do naprawy,
- błędy **oprogramowania** są **przemijające** – pokazują się dla pewnych wejść, system często może pracować nadal po ujawnieniu się błędu.

POFOD (probability of failure on demand)

Prawdopodobieństwo błędu żądanej usługi

- systemy sterowania, safety-critical

np.:

0.002

2 na 1000 żądanych usług może dać błąd

ROCOF (rate of failure occurrence)

Współczynnik pojawienia się błędu, częstotliwość nieoczekiwanych zachowań systemu

- systemy operacyjne, transakcyjne (duże koszty startu systemu)

np.

0.01 - 1 błąd prawdopodobnie pojawi się w 100 jednostkach czasu działania systemu

[MTTF (mean time to failure)]

Średni czas między obserwowalnymi błędami

np.

1 błąd na każde 500 jednostek czasu

MTTF=500 (odwrotność ROCOF)

MTTF > czas trwania transakcji

- systemy o długim czasie transakcji ,
CAD

[**AVAIL** (availability)]

Dostępność

- systemy ciągle pracujące, telekomunikacyjne

Miara prawdopodobieństwa dostępności dla użycia systemu

np.

0.997 oznacza, że na 1000 jednostek czasu system jest dostępny w ciągu 997

[Jednostki czasu]

- zegar
- czas procesora (cykl)
- liczba transakcji np. systemy o zmiennym obciążeniu - systemy rezerwacji (noc/dzień)

Wymagania niezawodnościowe

- Wymagania niezawodnościowe są określone nieformalnie.
- Do pomiarów używa się **testowania statystycznego**.

TESTOWANIE STATYSTYCZNE

Służy do zmierzenia miar niezawodności (do wykrycia błędów oprogramowania służy defect testing).

Kroki:

1. Określenie **profilu działania systemu** (wzór użycia), można osiągnąć analizując historyczne dane wejściowe, określając ich prawdopodobieństwo występowania. Profil określa jak system jest używany w praktyce.
2. Wybór lub generacja danych wg określonego profilu.
3. Wykonanie testu, zbieranie czasu pracy między obserwowalnymi błędami w odpowiednich jednostkach czasu.
4. Po obserwacji wielu błędów obliczenia miar niezawodności

[Testowanie statystyczne]

Jest trudne do wykonania:

- niepewny profil operacyjny
- wysokie koszty generacji profilu
- statystyczna niepewność (przy wysokiej niezawodności)

Modele wzrostu niezawodności

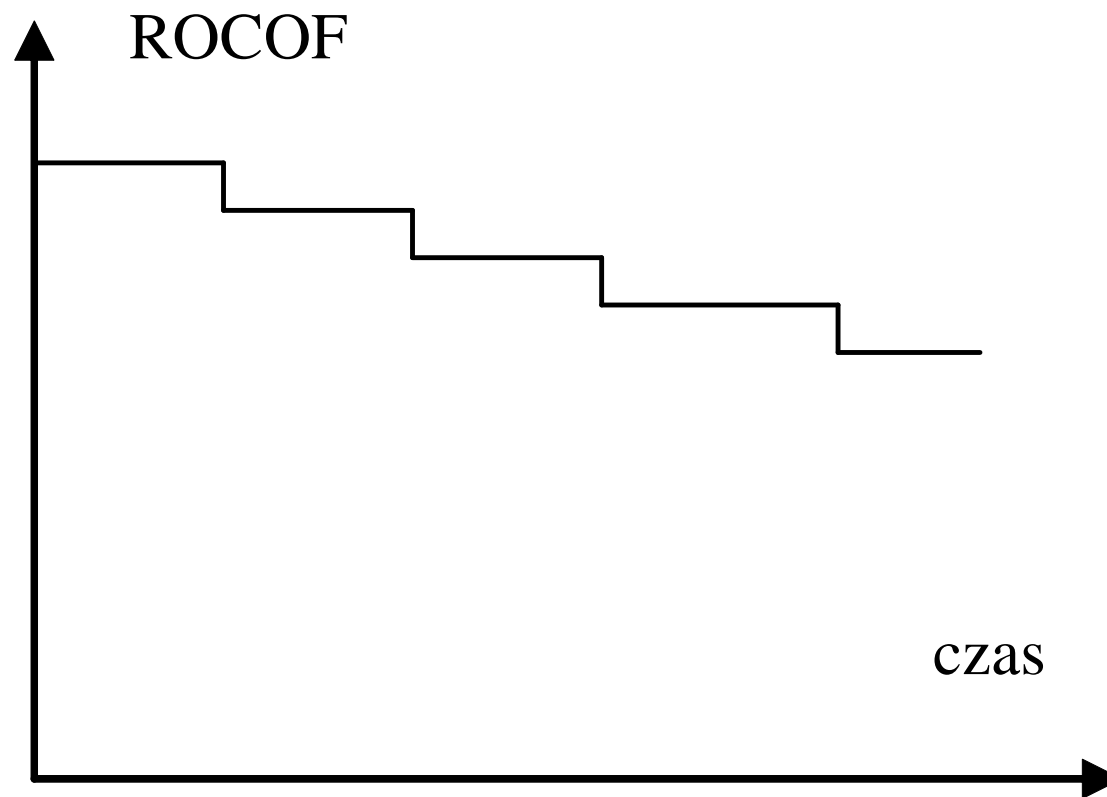
Testowanie powinno być prowadzone aż do osiągnięcia wymaganego poziomu niezawodności. Ponieważ testowanie jest kosztowne powinno być zaprzestane jak tylko stanie się to możliwe

- **Funkcja jednakowego kroku**
- **Funkcja losowego kroku**
- **Modele ciągłe**

[Funkcja jednakowego kroku] (Jelinski & Moranda 1972)

- Niezawodność wzrasta o stałą wartość po wykryciu i poprawieniu każdego błędu.
- Zakłada, że **naprawa błędu jest zawsze poprawna** i nigdy nie powoduje wzrostu liczby błędów w systemie.
- Poprawienie błędu nie zawsze powoduje wzrost niezawodności, mogą być wprowadzone nowe błędy.
- Model zakłada także, że każdy błąd powoduje jednakowy wzrost niezawodności.

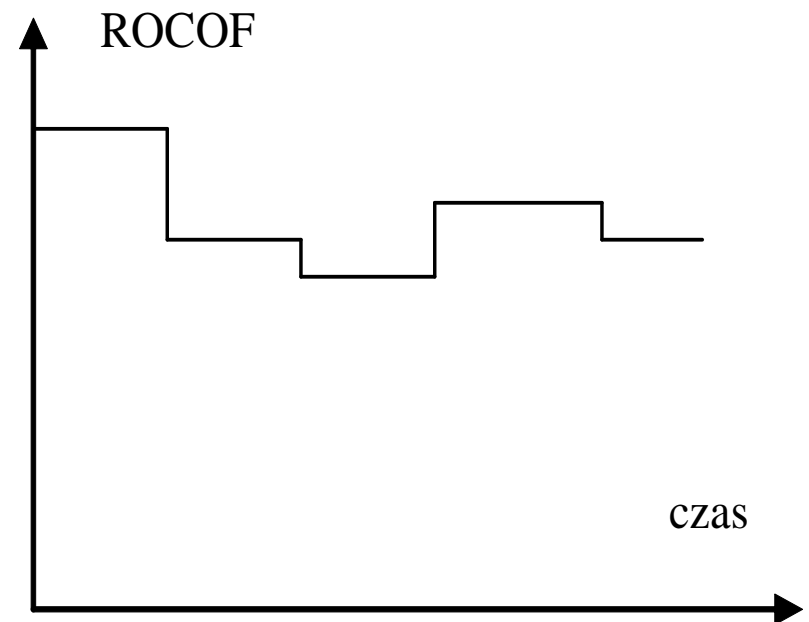
[Funkcja jednakowego kroku]



Funkcja losowego kroku

(Littlewood & Verrall 1973)

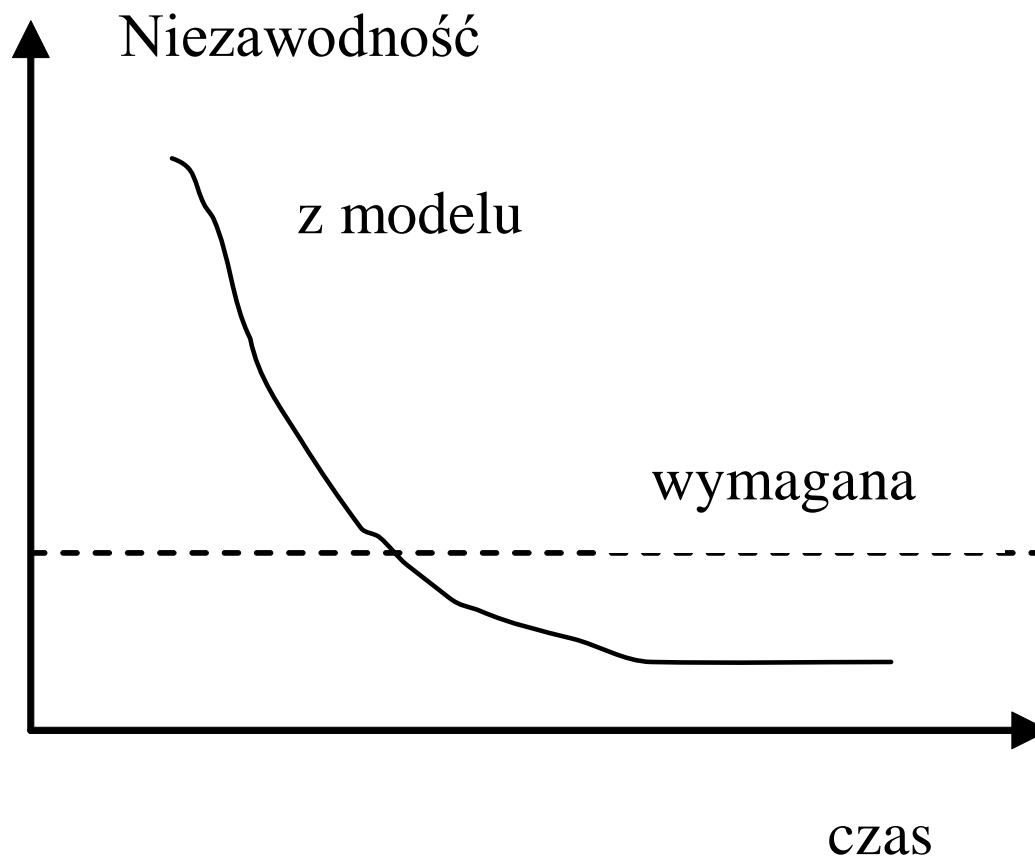
Modeluje fakt, że wraz z naprawianiem błędów średnie polepszenie niezawodności na naprawę zmniejsza się (może być także negatywny wzrost niezawodności).



[Modele ciągłe]

- Musa et al. 1987, Abdel – Ghaly 1986, ...
- Modele używane do określenia jak szybko oprogramowanie „poprawia się” w czasie.
- Oprogramowanie jest testowane statystycznie, mierzona jest niezawodność, błędy poprawiane, testowanie itp.

[Model ciągły]



Techniki programowania dla systemów o dużej niezawodności

W systemach o wysokich parametrach niezawodnościowych stosuje się następujące strategie:

- **unikanie błędów** (ang. fault avoidance) technika dostosowana do wszystkich typów systemów, polega na organizacji procesu projektowania i implementacji ukierunkowanej na system „bez błędów”
- **tolerowanie błędów** (ang. fault tolerance) pewne błędy pozostają w systemie, są rozwiązywane tak, by system działał nadal mimo błędu.
- **detekcja błędów** (ang. fault detection) – detekcja błędów przed dostarczeniem systemu. Proces walidacji systemu korzysta z metod statycznych (przeglądy) i dynamicznych (testowanie) wykrywania błędów.

[unikanie błędów i detekcja]

- Zazwyczaj unikanie błędów i detekcja błędów są wystarczające do uzyskania żadanego poziomu niezawodności.
- Proces produkcji oprogramowania powinien być ukierunkowany na **unikanie błędów**, a nie na ich detekcję.
- Oprogramowanie bez błędów „fault-free” oznacza oprogramowanie odpowiadające specyfikacji. Mogą być błędy w specyfikacji, powodujące, że oprogramowanie nie będzie zachowywało się tak, jak by chciał użytkownik.

Czynniki sprzyjające bezusterkowemu oprogramowaniu

- Precyzyjna specyfikacja (ew. formalna), która jest niesprzecznym opisem tego co ma być wykonane.
- Podejście do projektowania i implementacji bazujące na ukrywaniu informacji, enkapsulacji.
- Właściwa organizacja procesu produkcji – programiści piszą oprogramowanie bez błędów.
- Korzystanie z języków programowania ze sprawdzaniem typów.
- Ograniczenia konstrukcji programowych będących źródłem wielu błędów np. liczby zmiennoprzecinkowe często nieprecyzyjne, wskaźniki, dynamiczny przydział pamięci, współbieżność, rekursja, przerwania są często źródłem błędów.

[System tolerujący uszkodzenia]

Kontynuuje działanie mimo pojawienia się błędu. Tolerowanie uszkodzeń jest konieczne w pewnych typach systemów np. kontrola lotów, w systemach, gdzie niesprawność systemu może powodować duże straty ekonomiczne lub ludzkie.

Aspekty tolerowania uszkodzeń

- **Detekcja uszkodzenia** – system musi wykrywać, że pewna kombinacja dała, lub może dać błąd.
- **Rozmiar zniszczeń** (ang. damage assesment) – wykrycie części systemu, na które błąd miał wpływ.
- **Powrót z błędu** (ang. fault recovery) – przejście systemu do stanu „bezpiecznego”. Możliwe jest:
 - **poprawienie stanu błędnego** (ang. forward error recovery) – jest bardzo trudne, wymaga przewidywania stanu systemu
 - **odtworzenie stanu systemu** (ang. backward error recovery).

Aspekty tolerowania uszkodzeń -2

- **Naprawienie błędu** (ang. fault repair) – modyfikacja systemu.
- Błędy oprogramowania są często przemijające i w wielu sytuacjach naprawa nie jest konieczna. Jeżeli błąd nie jest przemijający to powinna być zainstalowana nowa wersja systemu. Dla systemów ciągle pracujących powinno to być wykonywane dynamicznie.

[podejście redundancyjne]

N-wersji oprogramowania.

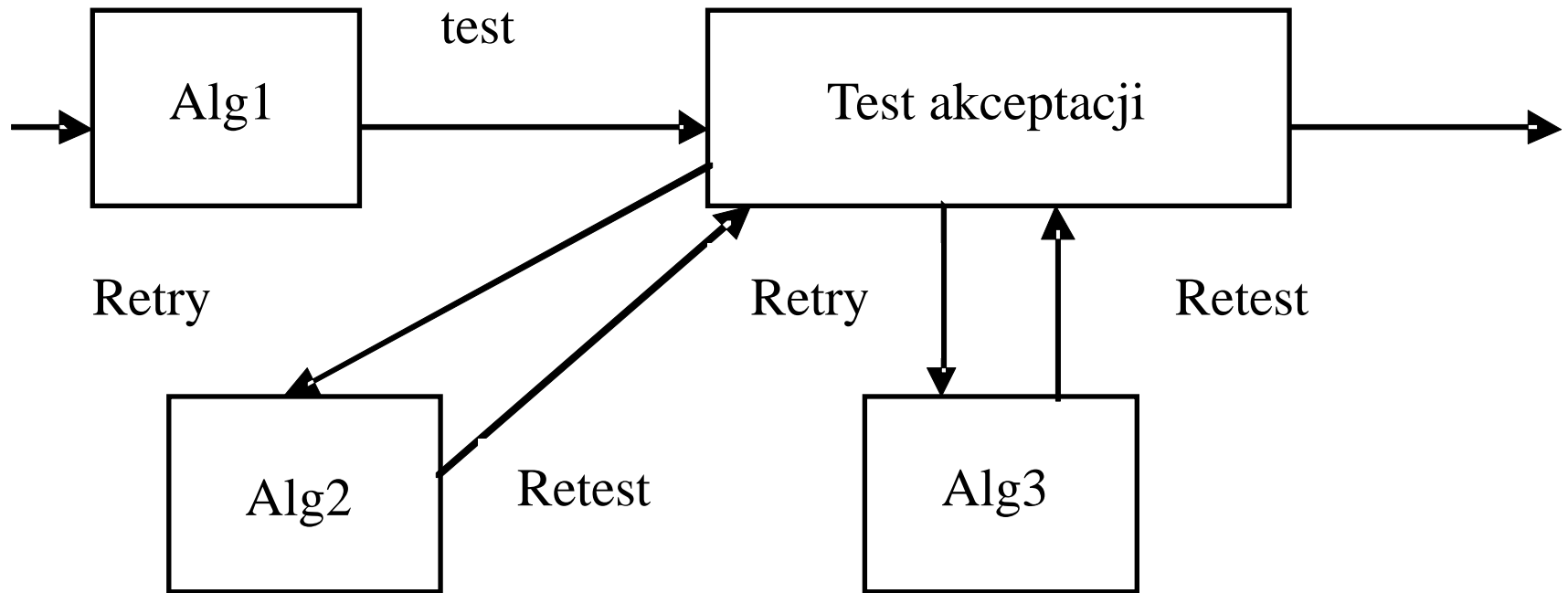
- Dla tej samej specyfikacji różne zespoły produkują oprogramowanie.
- Wersje są wykonywane równolegle.
- Wyjścia są porównywane w systemach głosujących (Avizienis 1985,1995).
- Wyjście niespójne jest odrzucone.

W podejściu tym zakłada się, że ludzie nie popełnią tego samego błędu projektowego lub implementacyjnego, a to okazało się nieprawdą np. niejasności w specyfikacji mogą być zinterpretowane w ten sam sposób przez różne grupy ludzi.

[bloki rezerwowe (ang. recovery blocks).]

- Każdy komponent programu zawiera test sprawdzający, czy komponent pracował poprawnie.
- Zawiera także kod pozwalający systemowi na odtworzenie i powtórzenie innego bloku kodu jeśli test wykrył błąd.
- Wykonanie bloków rezerwowych jest sekwencyjne.
- Bloki rezerwowe często są napisane w innych językach programowania, korzystają z różnych algorytmów. Autorami tego rozwiązania byli Randell(1975) Randell & Xu (1995).

[Recovery blocks]



[sytuacje wyjątkowe]

- W niektórych językach programowania np. C++ mamy możliwość **obsługi sytuacji wyjątkowych** (ang. exception handling), czyli określenia kodu obsługi sytuacji wyjątkowej.
- Sytuacje wyjątkowe pozwalają na wykrywanie pewnych błędów wykonania (badanie czy wartości nie przekraczają dozwolonych zakresów, czy zachowane są relacje między zmiennymi).

[programowanie defensywne]

[ang. defence programming]

- Programista zakłada, że w programie mogą wystąpić błędy i niespójności.
- Włącza kod redundancyjny do sprawdzania stanu systemu i powrotu do stanu właściwego.

Np. Programista opracowuje procedury współpracy ze stosem, *push* – włożenie elementu na stos i *pop* – zdjęcie elementu ze stosu. Programista w procedurze *pop* **włącza kod sprawdzający, czy stos nie jest pusty.**

[prewencja błędów (failure prevention).]

- Pewne typy błędów są wykrywane przez kompilatory, statycznie.
- W kod programu mogą być włączane asercje, dynamicznie sprawdzające stan zmiennych systemowych.
- Asercje zwalniają wykonanie programu, zajmują dodatkową pamięć ale pozwalają uchronić system przed poważnymi błędami.

[ocena zniszczeń]

Stosowane techniki umożliwiające ocenę zniszczeń to:

- użycie sum kontrolnych,
- użycie linków redundancyjnych,
- w systemach współbieżnych użycie zegarów kontrolnych (ang. watch dog), resetowanych po zakończeniu wykonania przez proces, jeśli proces nie zakończy się, zegar nie zostanie zresetowany i kontroler zauważy tę sytuację.

[Powrót z błędu - fault recovery]

Przeprowadzenie system do stanu „bezpiecznego”, w którym rezultaty błędu będą zminimalizowane, a system może kontynuować pracę, być może w formie zdegradowanej.

- **Poprawienie stanu błędnego** (forward error recovery) jest trudne, wymaga przewidywania stanu systemu, stosowania odpowiednich mechanizmów programowania np. kody korekcyjne, zwielokrotnione linki.
- **Odtworzenie stanu systemu** (backward error recovery) jest prostsze, system powraca do zachowanego wcześniej stanu (np. zapisanego na nośniku).