Inżynieria oprogramowania

Wykład 7:

Testowanie i konserwacja oprogramowania

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

Wersja 1.0

Podstawowe pojęcia

- **Testowanie** - polega na przeprowadzeniu eksperymentów z pewnymi częściami systemu (lub jego całością), wykorzystujące specjalnie dobrane dane testowe i porównaniu wyników uzyskanych z oczekiwanymi; tak postępując można w sposób kontrolowany i systematyczny, zademonstrować obecność określonej funkcjonalności systemu oraz badać obecność niepożądanych efektów
- **Weryfikacja** (ang. *verification*) - sprawdzenie zgodności systemu z określo-nymi na starcie wymaganiami
  „Are we building the system right"
- **Atestowanie** (ang. *validation*) - ocena systemu podczas lub na końcu procesu jego rozwoju na zgodność z rzeczywistymi wymaganiami użytkownika
  „Are we building the right system"
  (ten o który chodzi użytkownikowi)

Podstawowe pojęcia (2)

- **Błąd** (ang. *fault, error*) - niepoprawna konstrukcja znajdująca się w kodzie, która może doprowadzić do niewłaściwego działania; pojęcie statyczne, najczęściej skutek działania programisty
- **Błędne wykonanie, awaria** (ang. *failure*) - niepoprawne działanie systemu w trakcie jego pracy; program nie jest w stanie prawidłowo wykonać co najmniej jednej swojej funkcji w określonych warunkach operacyjnych; pojęcie dynamiczne; objaw błędu
- Jeden błąd może prowadzić do różnych błędnych wykonań, ale może również nie objawiać się przy typowych sytuacjach
- To samo błędne wykonanie może być spowodowane różnymi błędami
- Proces weryfikacji oprogramowania można określić jako poszukiwanie i usuwanie błędów na podstawie obserwacji błędnych wykonań oraz innych testów

Rodzaje testów

Główne cele testowania:

- wykrycie i usunięcie błędów w systemie
- ocena niezawodności systemu

Na tej podstawie wyróżniamy:

- Wykrywanie błędów - testy, których głównym celem jest wykrycie jak największej liczby błędów w programie
- Testy statystyczne - celem jest wykrycie przyczyn najczęstszych błędnych wykonań oraz ocena niezawodności systemu

Z punktu widzenia techniki wykonywania testów można je podzielić na:

- dynamiczne (polegają na wykonywaniu programu i porównywaniu uzyskanych wyników z wynikami poprawnymi):
  - funkcjonalne - zakładają jedynie znajomość wymagań wobec testowanych elementów;na zasadzie „czarnej skrzynki"
  - strukturalne - znany sposób implementacji testowanej funkcji
- statyczne (oparte na analizie kodu):
  - dowody poprawności
  - metody nieformalne

Typowe fazy testowania systemu

- Testy częściowe (np. modułów) - wykonywane najczęściej podczas implementacji, bezpośrednio po zakończeniu realizacji poszczególnych modułów
- Testy systemu - wykonywane po zintegrowaniu części składowych; testowane są poszczególne podsystemy oraz system jako całość
- Testy akceptacji (ang. *acceptance testing*) - w przypadku oprogramowania realizowanego na zamówienie system przekazywany jest klientowi do przetestowania przez przyszłych użytkowników (testy takie nazywa się testami alfa);w przypadku oprogramowania sprzedawanego rynkowo testy takie polegają na nieodpłatnym przekazaniu pewnej liczby kopii systemu grupie użytkowników (testy beta)

Testowane elementy (1)

- Kompletność i jakość założonych funkcji systemu
- Wydajność systemu i poszczególnych jego funkcji
- Zabezpieczenie systemu - odporność systemu na naruszenia prywatności, tajności, integralności, spójności i dostępności
- Własności operacyjne systemu, np. wymagania logistyczne, organizacyjne, użyteczność/ stopień skomplikowania instrukcji kierowanych do systemu, czytelność ekranów, operacje wymagające zbyt wielu kroków, jakość komunikatów systemu, jakość informacji o błędach, jakość pomocy
- Przenaszalność oprogramowania - poprawność działania w zróżnicowanym środowisku, różnych rozmiarach zasobów i rodzajach sprzętu
- Odtwarzalność oprogramowania (ang. *maintainability*) - mierzoną zwykle średnim czasem doprowadzenia do sprawnego działania po wystąpieniu awarii (od zgłoszenia awarii do ponownego działania)
- Bezpieczeństwo oprogramowania - stopień minimalizacji katastrofalnych skutków wynikających z niesprawnego działania (standardowy przykład - awaria zasilania)

Testowane elementy (2)

- Obciążalność systemu - zdolność do poprawnej pracy przy dużych obciążeniach; np. maksymalnej liczbie użytkowników, bardzo dużych rozmiarach danych; w tych testach czas nie odgrywa najistotniejszej roli, chodzi wyłącznie o to, czy system poradzi sobie w ekstremalnych warunkach)
- Skalowalność systemu - spełnienie warunków (m.in. czasowych) przy znacznym wzroście obciążenia
- Niezawodność oprogramowania - zwykle mierzoną średnim czasem pomiędzy błędami (MTBF)
- Modyfikowalność oprogramowania - zdolność do zmiany przy zmieniających się założeniach lub wymaganiach
- Jakość dokumentacji, pomocy, materiałów szkoleniowych
- Testy wykorzystania zasobów - np. czas jednostki centralnej, pamięć operacyjna, przestrzeń dyskowa, ...
- Spełnianie ograniczeń - np. na zajmowaną pamięć, obciążenia procesora, ...
- Interfejsy systemu na zgodność z wymaganiami
- Akceptowalność systemu, tj. stopień usatysfakcjonowania użytkowników.

Testy statystyczne

Schemat testów:

- losowa konstrukcja danych wejściowych zgodnie z rozkładem prawdopodobieństwa tych danych
- określenie wyników poprawnego działania systemu na tych danych
- uruchomienie systemu oraz porównanie wyników jego działania z poprawnymi wynikami

Powyższe czynności powtarzane są cyklicznie

- Podstawową zaletą jest możliwość ich automatyzacji, a co za tym idzie, możliwości wykonania dużej ich liczby
- Stosowanie testów statystycznych wymaga określenia rozkładu prawdopodobieństwa danych wejściowych możliwie bliskiemu rozkładowi, który pojawi się w rzeczywistości (dokładne przewidzenie takiego rozkładu jest trudne, w związku z czym wnioski wyciągnięte na podstawie takich testów mogą nie być wystarczająco wiarygodne)
- Założeniem jest przetestowanie systemu w typowych sytuacjach (pojawiają się znacznie częściej); aby przetestować system w sytuacjach skrajnych, nietypowych, ale dostatecznie ważnych należy zmodyfikować wykorzystywany rozkład prawd.
- Jest to przykład techniki tzw. „brutalnej siły", która jest jednak stosunkowo mało efektywna

Testy funkcjonalne
(ang. black-box testing)

- Sprawdzanie funkcji oprogramowania bez zaglądania do środka programu; testujący traktuje sprawdzany moduł jak „*czarną skrzynkę*", której wnętrze jest niewidoczne
- Powinno obejmować cały zakres danych wejściowych, co zwykle jest praktycznie niemożliwe ze względu na olbrzymią liczbę dopuszczalnych danych wejściowych (efekt „eksplozji danych testowych")
- Można podzielić dane wejściowe w „klasy równoważności", co do których istnieje duże przypuszczenie, że będą produkować te same błędy; klasy mogą być również zależne np. od wyników zwracanych przez testowane funkcje
- Konieczne jest także przetestowanie wartości granicznych
- Wiele wejść dla danych (wiele parametrów funkcji) może wymagać zastosowania pewnych systematycznych metod określania ich kombinacji, np. tablic decyzyjnych lub grafów przyczyna-skutek.

Testy strukturalne

- W celu odróżnienia od testów funkcjonalnych nazywane testowaniem na zasadzie białej skrzynki (ang. *white-box testing*)
- Celem jest sprawdzanie wewnętrznej logiki oprogramowania poprzez odpowiedni dobór danych wejściowych, dzięki czemu można prześledzić wszystkie istotne ścieżki przebiegu sterowania programu
- Tradycyjnie programiści wstawiają kod diagnostyczny do programu aby śledzić wewnętrzne przetwarzanie; debuggery pozwalają programistom obserwować wykonanie programu krok po kroku
- Zwykle niezbędne jest wcześniejsze odpowiednie przygotowanie danych testowych lub wykorzystanie specjalnych programów usprawniających testowanie (np. programu wywołującego testowaną procedurę z różnymi parametrami)
- Ograniczeniem testów strukturalnych jest niemożliwość wykrycia brakujących funkcji w programie (wadę tę usuwa testowanie funkcjonalne)

Ocena liczby błędów

- Z punktu widzenia użytkownika liczba błędów w oprogramowaniu niekoniecznie musi być bezpośrednio powiązana z jego zawodnością
- Oszacowanie liczby błędów ma natomiast duże znaczenie dla producenta oprogramowania, gdyż ma wpływ na koszty konserwacji
- Przewidywane koszty konserwacji oprogramowania związane z usuwaniem błędów mogą być prognozowane na podstawie:
  - szacunkowej liczby błędów w programie (np. przy wykorzystaniu techniki „posiewania błędów")
  - średnim procencie błędów zgłaszanych przez użytkownika systemu (oszacowanym na podstawie danych z poprzednich przedsięwzięć)
  - średnim koszcie usunięcia błędu na podstawie danych z poprzednich przedsięwzięć.
- Szczególnie istotne dla firm sprzedających oprogramowanie pojedynczym lub nielicznym użytkownikom (relatywnie duży koszt usunięcia błędu)

Technika "posiewania błędów"

- Do oprogramowania celowo wprowadza się pewną liczbę błędów (powinny być podobne do tych, które zwykle występują)
- Wykryciem tych błędów zajmuje się inna grupa programistów niż ta, która dokonała "posiania" błędów
- Szacunkowa liczba błędów przed wykonaniem testów: $(M - X) * N/X$
- Szacunkowa liczba błędów po usunięciu wykrytych: $(M - X) * (N/X - 1)$
- Oznaczenia:
  - N oznacza liczbę posianych błędów
  - M oznacza liczbę wszystkich wykrytych błędów
  - X oznacza liczbę posianych błędów, które zostały wykryte
- Szacunki te mogą być mocno chybione, jeżeli sztucznie wprowadzone błędy nie będą podobne do rzeczywistych błędów występujących w programie
- Technika ta pozwala również na zweryfikowanie skuteczności metod testowania oprogramowania i zbyt mały stosunek X/N oznacza konieczność poprawy tych metod

Testy statyczne

Polegają na analizie kodu bez uruchomienia programu, możliwe techniki:

- dowody poprawności (nie są praktycznie osiągalne dla rzeczywistych programów);
- sformalizowane przeglądy;
- metody nieformalne (polegają na analizie kodu przez autora i jeśli uzna on swój kod za poprawny przekazuje go bardziej doświadczonemu koledze; szczególnie istotne części kodu są analizowane przez grupę osób), dwie możliwości:
  - śledzenie przebiegu programu (wykonywanie programu "w myśli" przez analizujące osoby)
  - wyszukiwanie typowych błędów (niezainicjowane zmienne, porównania liczb zmiennoprzecinkowych, indeksy wykraczające poza tablice, błędne operacje na wskaźnikach i w warunkach instrukcji warunkowych, niekończące się pętle, ...)

Testy nieformalne są niedocenione, chociaż bardzo efektywne w praktyce

Przeglądy

Przegląd jest procesem lub spotkaniem, podczas którego produkt roboczy lub pewien zbiór produktów roboczych jest prezentowany dla personelu projektu, kierownictwa, użytkowników, klientów lub innych zainteresowanych stron celem uzyskania komentarzy, opinii i akceptacji

Wyróżniamy przeglądy:

- nieformalne
- formalne:
  - przegląd techniczny - służy do oceny zgodność postępu prac z przyjętym planem (ANSI/IEEE Std 1028-1988 „*IEEE Standard for Reviews and Audits*")
  - przejście (ang. *walkthrough*) - wczesna ocena dokumentów, modeli, projektów i kodu, której celem jest zidentyfikowanie defektów i rozważenie możliwych rozwiązań; wtórnym celem jest szkolenie i rozwiązanie problemów stylistycznych (np. z formą kodu, dokumentacji, interfejsów użytkownika)
  - audyt - potwierdzają zgodność oprogramowania z wymaganiami, specyfikacjami, zaleceniami, standardami, procedurami, instrukcjami, kontraktami i licencjami

Audyt projektu informatycznego

Celem audytu projektu informatycznego jest dostarczenie odbiorcy i dostawcy obiektywnych, aktualnych i syntetycznych informacji o stanie całego projektu

Zbierane są dowody, że zespół projektu:

- posiada możliwości (zasoby, kompetencje, metody, standardy) by osiągnąć sukces,
- optymalnie wykorzystuje te możliwości,
- rzeczywiście osiąga założone cele (cząstkowe)

Zebrane informacje służą jako podstawa do podejmowania strategicznych decyzji w projekcie

Przedmioty audytu:

- procesy projektu informatycznego - celem jest sprawdzenie prawidłowości wykonywanych prac jak i sposobu ich wykonywania
- produkty (cząstkowe) projektu informatycznego - celem jest sprawdzenie czy rezultaty poszczególnych prac odpowiadają zakładanym wymaganiom

Perspektywy audytu:

- technologia - celem jest sprawdzenie czy użyte techniki oraz opracowane rozwiązania są prawidłowe i prawidłowo stosowane
- zarządzanie - celem jest sprawdzenie czy sposób zarządzania projektem umożliwia jego powodzenie

Konserwacja oprogramowania

- Inne używane terminy to pielęgnacja lub utrzymanie (ang. *maintenance*)
- Konserwacja polega na zapewnieniu poprawnego i efektywnego funkcjonowania systemu poprzez wprowadzenie niezbędnych modyfikacji

Istnieją trzy główne klasy wprowadzanych w oprogramowaniu modyfikacji:

- **poprawiające** - polegają na usuwaniu z oprogramowania wykrytych podczas normalnej pracy błędów (nie zostały one wykryte podczas testowania) a popełnionych w czasie analizy wymagań, projektowania lub najczęściej implementacji
- **ulepszające** - polegają na poprawie jakości oprogramowania,
- **dostosowujące** - polegają na dostosowaniu oprogramowania do zmian zachodzących w wymaganiach użytkownika lub w środowisku pracy oprogramowania

Modyfikacje oprogramowania

- Wymaga się, aby wprowadzanie modyfikacji polegało na zmianie nie tylko kodu, ale również dokumentacji (np. projektu)
- Dzięki temu dysponujemy cały czas spójną dokumentacją i minimalizujemy ryzyko związane z niekontrolowanymi zmianami kodu

Przykłady modyfikacji ulepszających:

- Poprawa wydajności istniejących funkcji (np. nowy algorytm przetwarzania)
- Poprawa ergonomii interfejsu użytkownika (np. automatyzacja powtarzalnych operacji)
- Poprawa przejrzystości raportów lub inne sposób prezentacji

Modyfikacje dostosowujące wynikają z:

- Zmiany (rozszerzenia) wymagań użytkowników (np. nowy profil działalności)
- Zmian przepisów prawnych dotyczących dziedziny problemu lub jego otoczenia
- Zmian organizacyjnych po stronie klienta (np. inna forma własności lub zmiany terytorialne)
- Zmian sprzętu i oprogramowania systemowego
- Zmian innych systemów z którymi współpracuje oprogramowanie (potrzeba opracowania nowych interfejsów)

Analiza celowości wprowadzania modyfikacji

Użytkownicy zwykle zgłaszają wiele potencjalnych usprawnień i modyfikacji, które ich zdaniem są wskazane lub niezbędne w pracy systemu; należy podchodzić do takich propozycji z należytą uwagą, ale i odpowiednim krytycyzmem

Analiza celowości wprowadzenia zmian powinna uwzględniać:

- Znaczenie wprowadzenia zmiany dla użytkowników
- Koszt wprowadzenia zmiany i uzyskane korzyści
- Ocenę ryzyka destabilizacji normalnej pracy w wyniku błędu nowej części
- Wpływ zmiany na poszczególne składowe systemu i składowe dokumentacji technicznej

Dopiero po dokonaniu oceny zmiany podejmowana jest decyzja o jej ewentualnej realizacji; w przypadku bardzo dużych przedsięwzięć może zostać powołana w tym celu specjalna komisja

Zaleca się grupowanie zmian, których wykonanie prowadzi do nowej wersji systemu

Koszty konserwacji oprogramowania

- Występuje tendencja, aby zbyt nisko oceniać koszt konserwacji; okazuje się jednak, że koszty te mogą być bardzo duże, zwłaszcza w przypadku organizacji funkcjonujących w zmiennym środowisku
- Niedocenianie nakładów pracy na konserwacji jest jedną z głównych przyczyn opóźnień przedsięwzięć i tylko częściowego ich wykorzystania

Obiektywne czynniki wpływające na koszty konserwacji:

- Stabilność środowiska w którym pracuje system - zmiany zachodzące w przepisach prawnych, zmiany struktury organizacyjnej i sposobów działania po stronie klienta prowadzą do zmian wymagań wobec systemu
- Stabilność platformy sprzętowej i oprogramowania systemowego - wymiana sprzętu może skutkować koniecznością dostosowania systemu
- Czas użytkowania systemu - całkowite koszty konserwacji oczywiście rosną, gdy system jest eksploatowany przez dłuższy czas

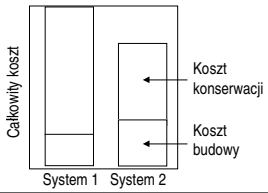
Czynniki redukcji kosztów konserwacji (1)

- **Znajomość dziedziny problemu** - jeżeli analitycy pracujący nad systemem dobrze znają daną dziedzinę problemu, mają mniej trudności z właściwym zebraniem wymagań oraz budową oddającego rzeczywistość modelu; możliwe jest również przewidzenie i uwzględnienie potencjalnych kierunków rozwoju systemu
- Wysoka **jakość modelu i projektu**, w szczególności jego spójność, stopień powiązania składowych oraz przejrzystość
- Wysoka **jakość dokumentacji technicznej** (powinna w pełni odpowiadać systemowi, być wystarczająco szczegółowa i zgodna z przyjętymi w firmie standardami)
- **Stabilność personelu** - niezależnie od jakości dokumentacji, pewne aspekty systemu są znane lepiej osobom bezpośrednio uczestniczącym w realizacji; nie muszą one dokonywać modyfikacji, ale dobrze jest gdy mogą wspomagać innych poprzez konsultacje w momencie pojawienia się wątpliwości

Czynniki redukcji kosztów konserwacji (2)

- **Środowisko implementacji** - odpowiednio zaawansowane i niezawodne narzędzia (np. RAD) sprzyjają skróceniu czasu niezbędnego na wprowadzenie modyfikacji
- **Niezawodność oprogramowania** - wysoka niezawodność systemu przekazanego klientowi zmniejsza liczbę modyfikacji zwłaszcza o charakterze poprawiającym
- **Inżynieria odwrotna** (ang. *reverse engineering*) - odtwarzanie dokumentacji technicznej na podstawie istniejącego oprogramowania; daje się w pewnym zakresie automatyzować, ale może wymagać dodatkowej korekty
- **Zarządzanie wersjami**

Wiele działań zmierzających do redukcji kosztów konserwacji musi być podjęte już w fazie budowy systemu.



Wiarygodność systemów (ang. *dependability*)

Wiarygodność jest atrybutem (wysokiego poziomu), na bazie którego mamy prawo zaufać usługom oferowanym przez system; wyróżniamy bardziej szczegółowe atrybuty m.in.:

- **Niezawodność** (ang. *reliability*) - określa zdolność systemu do nieprzerwalnego dostarczania usług, w określonych warunkach funkcjonowania
- **Dyspozycyjność** (ang. *availability*) - charakteryzuje procent czasu, w ramach którego system jest zdolny do świadczenia oczekiwanych usług, w odniesieniu do wyspecyfikowanych warunków funkcjonowania
- **Bezpieczeństwo** (ang. *safety*) - gwarancja, że awaria systemu nie spowoduje katastrofy w środowisku funcjonowania
- **Zabezpieczenie** (ang. *security*) - wiąże się z dostępem do informacji przetwarzanej, przechowywanej lub przesyłanej
  - **poufność** (ang. *confidentiality*) - określa stopień zabezpieczenia przed nieupoważnionym dostępem do informacji
  - **integralność** (ang. *integrity*) - określa stopień zabezpieczenia przed nieuprawnioną modyfikacją
  - **dyspozycyjność** - określa stopień gwarancji, że informacja będzie możliwie najszybciej udostępniona na żądanie uprawnionych podmiotów

Niezawodność

- Zapobieganie defektom (ang. *fault prevention*) - dwa podejścia: unikanie defektów i usuwanie defektów;
- Samo zapobieganie defektom nie gwarantuje jednak uzyskania wysokiej niezawodności; wynika to przede wszystkim z tego, że:
  - starzenie się i zużycie elementów sprzętowych powoduje pojawianie się defektów fizycznych w trakcie działania systemu (nie ma możliwości całkowitej eliminacji problemu)
  - praktycznie wykorzystywane oprogramowanie jest na tyle skomplikowane, że nie ma możliwości przetestowania go dla wszystkich sytuacji; podczas konserwacji często wprowadzane są do oprogramowania kolejne defekty
- Oznacza to, że techniki zapobiegania defektom muszą być uzupełnione metodami mającymi na celu **tolerowanie** skutków **defektów**, które wciąż występują w systemie i ujawniają się podczas jego pracy
- Tolerowanie defektów obejmuje zarówno sprzęt jak i oprogramowanie składające się na funkcjonujący system

Zasady tolerowania defektów

- Celem jest zapobieganie spowodowanych defektami awarii systemu; działania muszą być podejmowane zanim błędne funkcjonowanie poszczególnych komponentów przerodzi się w awarię całego systemu

Wyróżniono 4 fazy obejmujące całość zagadnień związanych z tolerowaniem defektów:

- wykrycie błędnego wykonania (ang. *error detection*) - pomiędzy wystąpieniem defektu a jego wykryciem może upłynąć pewien czas, gdyż stan systemu nie jest obserwowany w sposób ciągły po kątem wykrywania defektów
- ograniczenie i ocena zniszczeń (ang. *demage confinement and assessement*)
- naprawa stanu systemu po wystąpieniu błędnego wykonania (ang. *error recovery*) - przywrócenie stanu uprawnionego, tak aby możliwa była kontynuacja działania bez negatywnych skutków
- usunięcie defektu i kontynuacja pracy systemu (ang. *fault treatement and continued system service*) - identyfikacja komponentów zawierających defekty i np. ich wymiana

Nadmiarowość

Odnosi się do tych części systemu, których zadaniem jest osiąganie efektu tolerowania defektów:

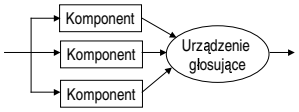
- **sprzętowa** - komponenty sprzętowe specjalnie dodane do systemu i wykorzystywane tylko do tolerowania defektów
- **programowa** - (analogicznie jak sprzętowa, ale dotyczy oprogramowania)
- **temporalna (czasowa)** - operacje są powtarzane; stosowana zwykle w celu wykrycia i naprawy skutków defektów przejściowych, które nie ujawniają się w trakcie wszystkich powtórzeń danej operacji (np. w protokołach komunikacyjnych komunikat może być wysyłany wielokrotnie w celu tolerowania awarii sieci)

Inna klasyfikacja rozróżnia nadmiarowość:

- **dynamiczną** (w celu wykrycia błędu; np. sprawdzanie bitu parzystości)
- **statyczną** (w celu zamaskowania awarii komponentu i uniemożliwienia im spowodowania awarii systemu; np. potrójna nadmiarowość modularna)

Potrójna nadmiarowość modularna
(ang. *Triple Modular Redundancy*)

- Polega na utrzymywaniu trzech identycznych kopii danego modułu (komponentu)
- Wszystkie trzy kopie odbierają identyczne sygnały wejściowe, a ich sygnały wyjściowe są przesyłane do **urządzenia głosującego** (ang. *voter*)
- Urządzenie głosujące porównuje otrzymane sygnały i podejmuje decyzję na zasadzie "głosowania większościowego" (na wyjściu podawana jest wartość pochodząca z co najmniej 2 kopii modułu, a pozostała ignorowana)



- Służy do zamaskowania awarii jednej kopii modułu
- W nieznacznym stopniu wpływa na obniżenie efektywności czasowej systemu (kopie działają równolegle, jedynie urządzenie głosujące wprowadza opóźnienie)

Potrójna nadmiarowość modularna (2)

- Do poprawnego funkcjonowania niezbędne jest doprowadzenie do sytuacji w której ewentualne awarie różnych kopii modułu są **niezależne** (awaria w jednej kopii nie jest w żaden sposób związana z awarią w innej)
- Może to wymagać niezależności fizycznej (oddziaływania elektryczne, magnetyczne, termiczne) oraz by kopie nie współdzieliły defektów o wspólnych przyczynach (np. uszkodzenie zasilania)
- Kluczowe jest również poprawne funkcjonowanie urządzenia głosującego oraz linii przesyłowych sygnałów wymienianych w ramach struktury urz. głosującego (w przypadku jego awarii nadmiarowość komponentów traci oczywiście sens)
- Struktura jest efektywna podczas awarii wynikających z defektów fizycznych (z natury niezależnych w każdej kopii), nie jest przydatna w wypadku defektów projektowania (jeżeli komponent zawiera defekt projektowania to bedzie on obecny we wszystkich kopiach i będzie się objawiał jednocześnie)

Testy związane z powtarzaniem obliczeń
(ang. *replication check*)

- Polegają na powtarzaniu obliczeń dokonywanych w systemie i można je traktować jako alternatywną implementację specyfikacji systemu
- W celu wykrycia rozbieżności wyniki pierwotne i wtórne są porównywane
- Zakres obliczeń zależy od typu defektu, który zamierzamy tolerować
- Mają bardzo wysoką zdolność wykrywania, ale ze względu na zakres nadmiarowości są kosztowne w realizacji
- Jeżeli rozpatrujemy wyłącznie defekty fizyczne można wykorzystywać dokładne repliki systemu; technika ta jest stosowana w systemach, które muszą wykazywać się szczególnie wysoką dyspozycyjnością (np. centrale telefoniczne)
- Przykładowo w systemi Stratus wszystkie komponenty sprzętowe są duplikowane w ramach wspólnej płyty montażowej,wyposażonej dodatkowo w urządzenia samodiagnozujące i wykrywające defekty; dodatkowo każda płyta ma kopię działającą z nią w konfiguracji dupleksowej; w momencie wykrycia sytuacji niepożądanej, system kontynuuje prace przy użyciy drugiej płyty;

Testy związane z kontrolą czasu
(ang. *timing checks*)

- Stosowane są w sytuacji gdy specyfikacja rozpatrywanego komponentu narzuca ograniczenia czasu obliczeń
- Wiąże się to z wprowadzeniem urządzeń kontroli czasu, które są każdorazowo ustawiane na moment w którym musi nastąpić zakończenie obliczeń; w przypadku braku wyników przed upływem przyznanego czasu, urządzenie sygnalizuje przekroczenia czasu obliczeń (ang. *timeout*); sytuacja taka jest interpretowana jako awaria komponentu
- Jeżeli komponent wyprodukuje wyniki w limicie czasu, to nie oznacza to jednak, że są one poprawne (mogą okazać się błędne, ale nie zostanie to wykryte)
- W celu wykrywania defektów oprogramowania (np. w postaci zapętlenia się programu) stosuje się testy tzw. nadzorców (ang. *watch dog timer*); komunikacja pomiędzy programem a jego nadzorcą polega na tym, że okresowo program odświeża ustawienie czasu obserwacji u swego nadzorcy; jeżeli nadzorca stwierdza wyczerpanie limitu wówczas sygnalizuje sytuację wyjątkową
- Podobnie realizowane jest wykrywanie awarii procesorów w systemach wieloprocesorowych i rozproszonych; każdy z procesorów co jakiś czas rozsyła do innych sygnał potwierdzający poprawne funkcjonowanie

Testy związane z kodowaniem informacji

- Polegają na dodaniu nadmiarowych bitów kontrolnych, których wartości pozos-tają w zadanej relacji z bitami danych podlegającymi kodowaniu; ukierunkowane są na określone typy uszkodzeń i efektywne tylko w tych przypadkach
- Jeżeli na skutek defektu nastąpi zmiana niektórych bitów w taki sposób, że relacja nie będzie spełniona, wówczas uzyskiwana jest informacja o problemie
- Przykładowe techniki:
  - testy parzystości (umożliwiają wykrycie przekłamania pojedynczego bitu, nie dają jednak możliwości naprawy uszkodzenego bitu ani nie potrafią wykryć sytuacji, gdy przekłaniu ulegnie parzysta liczba bitów)
  - kod Hemminga (jest zdolny do naprawy przekłamania pojedynczego bitu oraz wykrywania przekłamań wielu bitów; wprowadza jednak większą nadmiarowość)
- W celu efektywnego wykrywania błędów powstających podczas transmisji bloków danych stosowane mogą być:
  - cykliczne kody nadmiarowe (ang. *Cyclic Redudancy Codes* - CRC)
  - sumy kontrolne

Odtwarzanie stanu poprawnego

Celem jest powrót z obecnego stanu błędnego do poprawnego stanu systemy, od którego można wznowić obliczenia:

- **odtwarzanie zstępujące** (ang. *backward error recovery*) - przywraca pewien stan poprzedni, co do którego jest domniemanie, że jest poprawny; wymaga zastosowania mechanizmu, który w reguralnych odstępach zachowuje bieżący stan systemu w celu jego ew. późniejszego użycia; może być stosowane do defektów nieoczekiwanych w postaci niezależnej od specyfiki systemu
- **odtwarzanie wstępujące** (ang. *forward error recovery*) - wykorzystuje aktualny (błędny) stan, aby przekształcić go w stan poprawny; skuteczne jest jedynie wtedy, gdy możliwe jest dokonanie dokładnej oceny zakresu defektów (tylko w odniesieniu do defektów oczekiwanych)

Efekt domina - występuje wtedy, gdy dezaktualizacja efektów pewnych akcji pociąga za sobą konieczność dezaktualizacji akcji wcześniejszych; obserwowany w systemach składających się z powiązanych komponentów; w szczególnych sytuacjach może spowodować niemożność pełnego przywrócenia stanu wcześniejszego

Tolerowanie defektów oprogramowania

- W odróżnieniu od sprzętu oprogramowanie nie ulega starzeniu i fizycznym uszkodzeniom; program, który jest poprawny w chwili obecnej będzie zawsze poprawny o ile nie ulegną zmianie warunki w jakich jest wykonywany
- Zmiana w środowisku wykonania (zmiana warunków wejściowych, dostępnych zasobów, ...) lub szczególny układ wartości danych wejściowych oraz zależności czasowe mogą prowadzić do błędnego wykonania pomimo intensywnych zabiegów weryfikacyjnych i walidacyjnych
- W oprogramowaniu, które powinno charakteryzować się szczególnie wysoką niezawodnością stosuje się:
  - bloki odtwarzania (ang. *recovery blocks*)
  - programowanie w N wersjach (ang. *N-version programming*)
- Oba podejścia bazują na rozmaitości projektowania (ang. *design diversity*), a różnią się jedynie sposobem wykorzystania modułów
- W żaden sposób nie mogą zastąpić weryfikacji i walidacji; określane są jako "ostatnia linia obrony"

Bloki odtwarzania

Blok odtwarzania składa się z trzech składowych:

- modułu pierwotnego, którego celem jest realizacja określonego zadania,
- testu akceptacyjnego, który dokonuje oceny wyników dostarczonych przez moduł pierwotny (czuły punkt, gdyż błąd w tym miejscu niweczy całą konstrukcję)
- modułu alternatywnego, który jest aktywowany w sytuacji, gdy test wyników jego poprzednika będzie negatywny (dopuszcza się większą liczbę modułów alternatywnych)

- Każdy moduł jest projektowany niezależnie (z zachowaniem zasady rozmaitości) na podstawie tej samej dokumentacji zewnętrznej
- Dążymy do tego, aby moduł pierwotny stosował najbardziej efektywny algorytm (jest aktywowany zawsze), ale być może w związku z tym bardziej skompliko-wany; kolejne moduły mogą opierać się o wolniejsze, ale i prostsze algorytmy
- Zakłada się, że dla danego zestawu danych wejściowych, przynajmniej jeden moduł będzie funkcjonował poprawnie
- Jeżeli wersja pierwotna nie zawiera defektów, narzut czasowy jest nieznaczny

Programowanie w N wersjach

- Dany program (lub jego część) jest realizowana w N wersjach z zachowaniem rozmaitości projektowania (tzn. wersje tworzone są niezależnie, na bazie wspólnej specyfikacji zewnętrznej)
- Wszystkie wersje działają równolegle, a ich wyniki poddawane są głosowaniu
- Wynik mający większość przyjmowany jest jako poprawny (podobnie jak w PNM) i udostępniany jest na zewnątrz (nie są wymagane specjalne testy akceptacyjne)

Realizacja takiego schematu wymaga specjalnego programu sterującego, który:

- synchronizuje wywołania wszystkich wersji (środowisko wykonania musi dopuszczać równoległe wykonanie)
- zbiera wyniki wyprodukowane przez poszczególne wersje (wyniki muszą być w tym samym formacie, aby można je było porównać; kwestia dopuszczenia marginesu odchyleń przy obliczeniach numerycznych)
- wykorzystuje te wyniki w głosowaniu