ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 2003
Seria: TRANSPORT z. 48 Nr kol.1604

Katarzyna TRZASKA

BADANIE NIEZAWODNOŚCI I POPRAWNOŚCI OPROGRAMOWANIA W KOLEJOWYCH SYSTEMACH KOMPUTEROWYCH

Streszczenie. W referacie przedstawiono problemy budowy bezpiecznego oprogramowania aplikacji kolejowych przy wykorzystaniu teorii poprawności oprogramowania. Poprawne oprogramowanie ma bardzo istotny wpływ na działanie systemów uwarunkowanych bezpieczeństwem, zwłaszcza 4 poziomu bezpieczeństwa (SIL4). Jako przykład przedstawiony został uproszczony system samoczynnej sygnalizacji przejazdowej.

ANALYSIS OF SOFTWARE RELIABILITY AND CORECTNESS IN RAILWAYS' COMPUTER SYSTEMS

Summary. This paper presents a problem of build the safety software of railway control system using theory of software correctness. The correctness of software has an important effect to performance the fail-safe systems, especially SIL4 (*safety integrity levels 4*). For example was presented the railway control system applied to cross level protection.

1. WSTĘP

Niezawodność i poprawność oprogramowania w kolejowych systemach komputerowych ma bardzo istotne znaczenie dla prawidłowego działania całego systemu kolejowego. Głównym zadaniem oprogramowania jest realizowanie określonych przez programistę funkcji, np. sterowanie sygnalizacją na przejazdach kolejowych czy zabezpieczenie ruchu pociągów na szlaku. Aby zapewnić bezpieczeństwo użytkownikom komunikacji szynowej, oprogramowanie stosowane w kolejowych systemach komputerowych musi być niezawodne.

Istotnym elementem wpływającym na niezawodność oprogramowania uzyskuje się przez zastosowanie odpowiedniego języka programowania. Każdemu z wyżej wymienionych poziomów bezpieczeństwa systemu odpowiada odpowiedni język programowania, którego programista może użyć do stworzenia prawidłowo działającego systemu. Istotnym elementem w procesie tworzenia oprogramowania jest także sam programista. Jeżeli oprogramowanie zostało właściwie napisane i realizuje zadania założone przez programistę, to system na nim oparty działa poprawnie i jest niezawodny. Jeżeli zawiera błędy, które wynikły z błędu samego programisty, system oparty na takim oprogramowaniu nie będzie spełniał wymaganych zadań, czyli będzie zawodny.

Godne uwagi są także ekonomiczne skutki zawodności oprogramowania. Jedna trzecia do połowy nakładów związanych z rozwojem i konserwacją oprogramowania jest poświęcona testowaniu i usuwaniu błędów. Ponieważ przy realizacji dużych systemów komputerowych znacznie więcej nakładów jest związanych z oprogramowaniem niż ze sprzętem, bezpośrednie koszty zawodności oprogramowania stanowią dużą część łącznych kosztów. Gdy uwzględni się również pośrednie koszty błędów (np. mniejsze korzyści), ekonomiczne znaczenie niezawodności stanie się jeszcze bardziej widoczne [1].

2. NIEZAWODNOŚĆ OPROGRAMOWANIA

Niezawodność oprogramowania jest prawdopodobieństwem, że oprogramowanie działa bez awarii w określonym przedziale czasu, opatrzonym wagą odpowiadającą kosztom poniesionym przez użytkownika z powodu każdej zaistniałej awarii [2]. Niezawodność oprogramowania jest więc funkcją wpływu błędów na użytkowników systemu, niekoniecznie odzwierciedlającą rzeczywistą wielkość błędu wewnątrz samego systemu. I tak na przykład poważny błąd projektowy może mieć tylko nieznaczny wpływ na użytkownika, i przeciwnie: pozornie trywialny błąd może mieć ogromny wpływ na użytkownika systemu. Niezawodność nie jest wrodzoną cechą programu: zależy ona przede wszystkim od sposobu użycia programu. Słowo prawdopodobieństwo w podanej powyżej definicji oznacza prawdopodobieństwo, że użytkownik nie wprowadzi określonego zestawu danych prowadzącego do awarii systemu [2]. Jeżeli założymy, że proces generacji błędów jest opisany modelem Poissona, to niezawodność programu (prawdopodobieństwo braku błędu przez czas t) określa następująca funkcja [3]:

$$R(t) = e^{-(\lambda_L+\lambda_I+\lambda_T)t} = e^{-\lambda t}$$

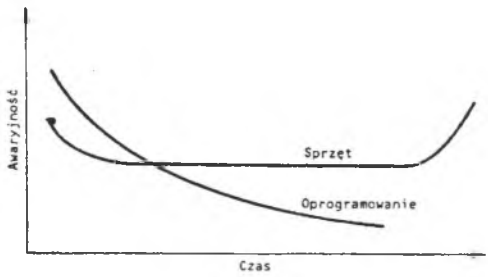
gdzie:

- λ_L - stopa błędów programowych,
- λ_I - stopa błędów implementacji,
- λ_T - stopa błędów przejściowych.

W praktyce możliwe jest wyeliminowanie błędów programowych (charakteryzowanych przez λ_L) oraz znaczna redukcja błędów implementacji (charakteryzowanych przez λ_I). Eliminację błędów programowych umożliwiają metody programowania strukturalnego, systematycznego i specyfikowanego gwarantujące uzyskanie modularnie zbudowanych programów w językach wysokiego poziomu (np. ADA, CHILL, PASCAL, MODULA) z

dokumentacją poprawności w postaci asercji umożliwiającej przeprowadzenie odpowiedniej weryfikacji programu.

Aby głębiej zrozumieć niezawodność oprogramowania, warto ją porównać z niezawodnością sprzętu. Urządzenie sprzętowe może być zawodne z trzech powodów: błędów projektowych, wad produkcyjnych i awarii. Niezawodność oprogramowania różni się w sposób istotny od niezawodności sprzętu. Oprogramowanie nie psuje się, ani się nie zużywa. Zawodność oprogramowania jest więc w całości powodowana błędami projektowymi, to znaczy błędami obecnymi już na początku procesu produkcyjnego. Zawodność sprzętu polega na jego losowych awariach, natomiast zawodność oprogramowania ma charakter deterministyczny. Awarie sprzętu zależą od czasu. Awarie oprogramowania - które mogą się wydawać funkcjami czasu - są w rzeczywistości funkcjami danych systemu i jego bieżącego stanu. Poniższy wykres przedstawia różnice pomiędzy niezawodnością sprzętu i oprogramowania [2].



Rys. 1. Różnice pomiędzy niezawodnością sprzętu i oprogramowania [2]
Fig. 1. Difference between software and hardware reliability [2]

3. BEZPIECZEŃSTWO OPROGRAMOWANIA

Bezpieczeństwo S na poziomie systemu jest określane jako zdarzenie przeciwne do zdarzenia przebywania w stanach uznanych za niebezpieczne. Przy założeniu że suma zdarzeń wyczerpuje cały zbiór zdarzeń, wartość bezpieczeństwa określa poniższy wzór [3]:

$$S = 1 - P_{NB} = 1 - \lim_{t \to \infty} \left(\sum_{i=1}^{n} P_i(t) \right)$$

gdzie:

- P_{NB} - prawdopodobieństwo przebywania w zbiorze stanów niebezpiecznych,
- P(t) - prawdopodobieństwo przebywania systemu w „i-tym" stanie zagrażającym bezpieczeństwu.

Wskaźnik opisany powyższą zależnością jest ilościową miarą bezpieczeństwa komputerowych systemów srk. Dla podsystemu komputerowego „k" prawdopodobieństwo wystąpienia błędu katastroficznego P_{FK} można określić w funkcji intensywności uszkodzeń λ_k oraz czasu reakcji systemu na błąd t_{rk}:

$$S_k - 1 - P_{FK} = -\frac{\lambda_k}{t_{rk}^{-1}}$$

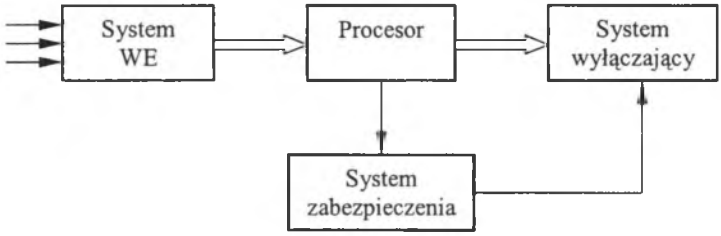
Obliczenie bezpieczeństwa systemu w złożonej strukturze opiera się na określeniu wypadkowego prawdopodobieństwa poprawnego działania takiej struktury [3].

4. BUDOWA NIEZAWODNYCH SYSTEMÓW STEROWANIA WG ZASADY ZAPEWNIENIA BEZPIECZEŃSTWA PRZY USZKODZENIACH

W systemach sterowania, które powinny odpowiadać wysokim wymaganiom bezpieczeństwa, często się wykorzystuje zasadę zapewnienia bezpieczeństwa przy uszkodzeniach (fail-safe). W technice kolejowej zasada fail-safe jest stosowana dość dawno w celu zapewnienia niezawodności i bezpieczeństwa w działaniu urządzeń srk. Zasada zapewnienia bezpieczeństwa przy uszkodzeniach może być wyrażona w następujący sposób: „uszkodzenia sprzętowe lub błędy oprogramowania w systemie przetwarzania mogą doprowadzić do różnych stanów niesprawności prowadzących do stanu uszkodzenia systemu, które jednak zawsze powinny być bezpieczne, tzn. takie, przy których nie występuje zagrożenie sterowanego procesu lub otoczenia" [4]. Ta podstawowa zasada zapewnienia bezpieczeństwa przy uszkodzeniach znajduje zastosowanie w dwóch formach, różnych z punktu widzenia realizacji zapewnienia bezpieczeństwa, tzn. w postaci bezpośredniego lub pośredniego zapewnienia bezpieczeństwa.

Metoda bezpośredniego zapewnienia bezpieczeństwa zakłada, że wszystkie uszkodzenia sprzętowe wprowądzające zakłócenia funkcjonalne bezpośrednio doprowadzają system do stanu bezpiecznego (rys. 2).

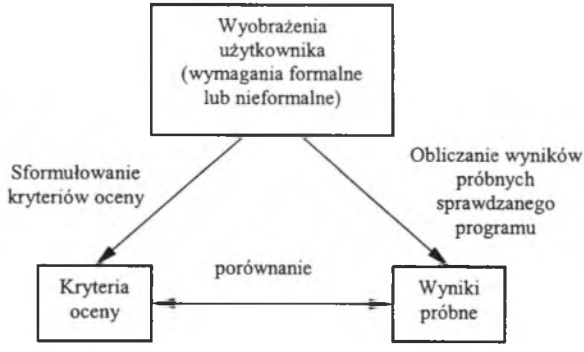
Metoda pośredniego zapewnienia bezpieczeństwa przy uszkodzeniach jest odmianą przedstawionej wyżej, umożliwiającą zapewnienie niezawodności poprzez wprowadzenie do systemu dodatkowych funkcji: kontrolnych i sterujących. Metoda ta jest realizowana za pomocą logiki sterowania wielokanałowego, którego kanały nie są całkowicie niezawodne i dlatego są kontrolowane za pomocą specjalnego układu komparatora, detektora pierwszego uszkodzenia i układu wyłączającego uszkodzony kanał (system), kontrolujących zgodność stanów wyjściowych [4].



Rys. 2. Bezpieczny system sterowania [4]
Fig. 2. Safety control system [4]

5. WERYFIKACJA OPROGRAMOWANIA

Dużą niezawodność i poprawność oprogramowania albo istnienie błędów można stwierdzić dopiero wtedy, gdy wyniki próbne uzyskane z weryfikowanego programu lub sam program porówna się z kryteriami oceny, które wynikają z pośrednio z wyobrażeń użytkownika na temat zadań systemu. Ten związek między wymaganiami, kryteriami oceny oraz wynikami próbnymi można przedstawić w postaci „trójkąta testowania" (rys. 3).



Rys. 3. Trójkąt testowania [1]
Fig. 3. Testing triangle [1]

Jeśli wyniki próbne nie odpowiadają przyjętym kryteriom oceny, to w trójkącie testowania występuje co najmniej jeden błąd. Tylko przy założeniu że sformułowanie kryteriów oceny jest bezbłędne i porównanie wyników próbnych z kryteriami oceny jest poprawne, można stwierdzić istnienie błędu w programie. Na podstawie przedstawionego trójkąta testowania nie można wyciągnąć żadnych wniosków dotyczących sytuacji poza tym trójkątem [1].

6. POZIOMY BEZPIECZEŃSTWA

Europejskie systemy komputerowe stosowane w kolejnictwie zostały sklasyfikowane względem pięciu poziomów bezpieczeństwa (ang. Safety Integrity Levels), uporządkowanych z uwagi na potencjalne skutki dla pasażerów i personelu obsługi, tak jak to przedstawia tablica 1. W tablicy 2 przedstawiono przypisanie poziomów bezpieczeństwa typowym systemom i podsystemom NSRK we Francji (SNCF), Niemczech (DB) i Wielkiej Brytanii (BR). Zaproponowano też klasyfikację systemów NSRK w kolejnictwie polskim uwzględniając aktualny stan regulacji prawnych i obowiązujące standardy. Jak widać, w europejskich zarządach kolejowych obowiązują zbliżone wymagania bezpieczeństwa, pewne różnice wynikają raczej z innych założeń dla niektórych podsystemów. Podstawową cechą bezpiecznych realizacji komputerowych systemów sterowania przyjętą w kolejnictwie jest zasada „fail-safe" (odporny na uszkodzenia), która mówi, że pojedyncze uszkodzenie (sprzętu, oprogramowania) lub zakłócenie nie może spowodować sytuacji niebezpiecznej, przy założeniu że prawdopodobieństwo wystąpienia uszkodzenia podwójnego (wielokrotnego) jest pomijalnie małe. Dodatkowo zakłada się detekcję błędów pojedynczych w stosunkowo krótkim czasie i odpowiednią reakcję systemu na fakt wykrycia uszkodzenia.

7. KRYTERIA WYBORU METOD OPROGRAMOWANIA [5]

Kryteria wyboru metod i miar dla oprogramowania systemu srk zawiera ocenę odnośnie do konieczności stosowania przy projektowaniu oprogramowania i przy analizie jego bezpieczeństwa danej metody lub miary. Symbol „O" oznacza, że użycie danej metody lub miary jest obowiązkowe dla danego poziomu bezpieczeństwa. Symbol „SZ" oznacza, że dana metoda lub miara jest bardzo zalecana dla danego poziomu bezpieczeństwa. Jeśli ta metoda lub miara nie jest stosowana, należy to uzasadnić w planie jakości. Symbol „U" oznacza, że dana metoda lub miara jest zalecana dla danego poziomu bezpieczeństwa. Jest to niższy poziom zalecenia do stosowania niż poziom „SZ". Symbol „ – „ oznacza, że dana metoda lub miara nie jest zalecana lub istnieje wskazanie przeciwko jej użyciu dla danego poziomu bezpieczeństwa. Symbol „NZ" oznacza, że dana metoda lub miara jest bezwzględnie niezalecana dla danego poziomu bezpieczeństwa. Jeżeli jednak jest ona stosowana, należy to uzasadnić w planie. Symbol „Z" oznacza, że użycie danej metody lub miary jest zabronione dla danego poziomu bezpieczeństwa. Jest to niższy poziom zalecenia do stosowania niż poziom „SZ". Symbol „ – „ oznacza, że dana metoda lub miara nie jest zalecana lub istnieje wskazanie przeciwko jej użyciu dla danego poziomu bezpieczeństwa.

Tablica 1

Poziomy bezpieczeństwa systemów komputerowych w kolejnictwie (TC-IRSE)

Poziom bezpieczeństwa systemu	Wymagany stan bezpieczeństwa Konsekwencje wystąpienia błędu systemu		Charakterystyka systemu w kolejnictwie	Nazwa systemu stosowana w kolejnictwie
4	Bardzo wysoki	Utrata życia ludzi	Zabezpieczenie przed wykolejeniem i kolizją pociągów	System bezpieczny
3	Wysoki	Obrażenia i utrata zdrowia ludzi	Zapewnienie poprawnego prowadzenia pociągu	System o wysokim poziomie bezpieczeństwa
2	Średni	Skażenie środowiska	Zapewnienie kierowania ruchem pociągów	System o znaczącym poziomie bezpieczeństwa
1	Niski	Utrata lub zniszczenie własności funkcjonalnych	Zapewnienie obsługi pasażerów	System o niskim poziomie bezpieczeństwa
0	Nie dotyczy	Utrata informacji nie wpływających na bezpieczeństwo	Zapewnienie prawidłowego utrzymania kolei	Nie związany z bezpieczeństwem

Tablica 2

Przykłady bezpiecznych systemów kolejowych

Lp.	Bezpieczne systemy i podsystemy kolejowe	Francja (SNCF)	Niemcy (DB)	Wielka Bryt. (BR)	Polska (PKP)
1	Systemy zależnościowe (różne rodzaje)	4	4	4	4
2	Sterowniki zwrotnic i sygnalizatorów	4	4	4	4
3	Układy detekcji pociągu (czujniki, liczniki osi)	4	4	4	4
4	Układy hamowania pociągu	4	2	2	4
5	Automatyczne, półautomatyczne blokady liniowe	4	4	4	4
6	Sygnalizacja przejazdowa (klasa A, B, C)	4	4	4	4
7	Urządzenia nadzoru sygnalizacji przejazdowej	-	4	4	3
8	Układy kontroli dyspozytorskiej	4	2	2	3
9	Pulpity sterownicze	4	4	2	2
10	Monitory ekranowe (dla sterowania ruchem)	4	4	-	4
11	Monitory ekranowe (do nadzorowania ruchem)	2	2	2	3
12	Zdalne sterowanie (dla sterowania ruchem)	4	4	4	3/2
13	Zdalne sterowanie (do nadzorowania ruchem)	2	2	2	2
14	Wykres ruchu i wykrywanie komunikatów	2	2	2	2
15	Automatyczne nastawianie przebiegów	-	2	2	3/2
16	Automatyczne ograniczanie prędkości (ATP)	-	4	4	4
17	Automatyczne sterowanie pociągiem (ATC)	2	-	-	3

Symbol „NZ" oznacza, że dana metoda lub miara jest bezwzględnie niezalecana dla danego poziomu bezpieczeństwa. Jeżeli jednak jest ona stosowana, należy to uzasadnić planie. Symbol „Z" oznacza, że użycie danej metody lub miary jest zabronione dla danego poziomu bezpieczeństwa. Wybrane metody i miary lub ich kombinacje należy podać w planie jakości, jeżeli kryteria nie zawierają innych ustaleń. Ustalenia te mogą zawierać odniesienia do zatwierdzonych metod i technik lub ich kombinacji. W przypadku zastosowania zatwierdzonych metod i technik dokonuje się jedynie sprawdzenia, czy zostały one

prawidłowo zastosowane. Jeżeli będzie zastosowany inny zestaw metod i technik, należy to uzasadnić w celu ich zatwierdzenia.

Metody projektowania można podzielić pod względem ich sformalizowania, czyli zwięzłości i ścisłości opisu oraz sposobu przedstawienia różnych aspektów systemu. Stosując pierwsze kryterium można wyodrębnić metody: nieformalne, półformalne i formalne.

Metody nieformalne bazują na słownym opisie rzeczywistości w języku naturalnym, który jest elastyczny, ale bardzo niejednoznaczny. Zaletą opisu słownego jest to, że można go łatwo zaprezentować klientowi. Różnego rodzaju notatki mogą mieć zastosowanie w przypadku tworzenia oprogramowania przez jedną osobę. Oprogramowanie stworzone w ten sposób jest jednak całkowicie nieodporne na modyfikacje. Stosując takie podejście można wytworzyć sterownik napisany w asemblerze, ale nie system informatyczny.

Tablica 3

Architektura oprogramowania

Metody i miary	Poziomy bezpieczeństwa	
	3	4
Programowanie defensywne	O	O
Diagnostyka i wykrywanie błędów	SZ	SZ
Kod korekcji błędów	NZ	NZ
Kody wykrywania błędów	SZ	SZ
Programowanie z kontrolą poprawności wykonywania programu	SZ	SZ
Metody monitorowania bezpieczeństwa	U	U
Zróżnicowanie oprogramowania	SZ	SZ
Metody bloków alternatywnych	U	U
Analiza zastępująca	NZ	NZ
Analiza wstępująca	NZ	NZ
Mechanizmy maskowania niesprawności przez ponowne wykonanie kodu	U	U
Rejestracja wykonywanych przypadków	SZ	SZ
Sztuczna inteligencja – korekcja defektów	NZ	NZ
Rekonfiguracja dynamiczna	NZ	NZ
Analiza skutku błędów	SZ	SZ
Analiza drzewa niezdatności	SZ	SZ

Tablica 4

Specyfikacja wymagań dla oprogramowania

Metody i miary	Poziomy bezpieczeństwa	
	3	4
Metody formalne łącznie np. z metodami CCS, SP, HOL, LOTOS, OBJ, logiką temporalną, VDM, Z	SZ	SZ
Metody półformalne	SZ	SZ
Metodologia strukturalna łącznie np. z metodami JSD, MASKOT, SADT,SDL, SSADM, metoda Yourdona	SZ	SZ

Metody półformalne opierają się na pewnym graficznym lub symbolicznym zapisie. Do tej grupy należy większość powszechnie stosowanych metod, np. UML.

Metody formalne są stosowane rzadko i przeważnie w systemach, w których zachodzi konieczność przeprowadzenia matematycznego dowodu poprawności działania systemu. Projektowanie i wytwarzanie systemu odbywa się drogą formalnych przekształceń. Inżynieria oparta na rodzinach produktów nie jest ściśle metodą formalną, uznaną przez teoretyków, choć ideowo odpowiada metodom formalnym. Istnieje (symboliczny) opis systemu podlegający (automatycznym) przekształceniom w system. Wspomniany VDM znajduje się na styku metod formalnych i nieformalnych. W tym języku zaprojektowano np. system nadzoru ruchu kolejowego w Wielkiej Brytanii.

Metody strukturalne zakładają dekompozycję informatyzowanej rzeczywistości (czyli przyszłego systemu) na dane i procesy na tych danych operujące.

System informatyczny składa się z modułów, między którymi mogą zachodzić różne związki. W obrębie jednego modułu mogą się znajdować procedury działające na wspólnych danych lub realizujące podobną funkcjonalność, np. interakcję z użytkownikiem. Dekompozycja systemu na moduły jest kluczowa dla powodzenia dalszego projektu. Systemy zaprojektowane z użyciem metod strukturalnych są trudno modyfikowalne i trudne w integracji, zwłaszcza przy złym podziale na moduły, co dla dużych systemów nie jest banalne. Metody strukturalne są wystarczające dla prostych systemów dostępu do danych.

Tablica 5

Języki oprogramowania

Metody i miary	Poziomy bezpieczeństwa	
	3	4
ADA	U	U
MODULA – 2	U	U
PASCAL	U	U
FORTAN 77	U	U
C lub C++ (bez ograniczeń)	NZ	NZ
Podzbiór języka C lub C++ ze standardami kodowania	U	U
PL\M	NZ	NZ
BASIC	NZ	NZ
Asembler	-	-
Diagramy drabinkowe	U	U
Bloki funkcjonalne	U	U
Lista stanów	U	U
Podzbiór C ze standardami kodowania	U	U

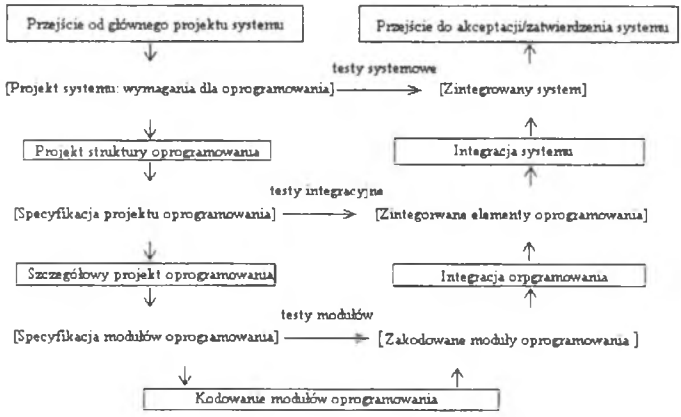
8. POPRAWNOŚĆ OPROGRAMOWANIA SYSTEMÓW STEROWANIA RUCHEM KOLEJOWYM

Zalecenia Komitetu CENELEC opracowane dla projektowania i analizy oprogramowania sterującego w zastosowaniach kolejowych w systemach sterowania ruchem kolejowym (SRK)

wprowadzają metody, procedury i kryteria oparte na teorii poprawności programów wywodzące się z matematycznej teorii programowania i logiki matematycznej [9]. Matematycznym modelem programu jest algorytm, intuicyjny graf zbudowany ze stanów i przejść między nimi określonych na dziedzinie zmiennych występujących w programie [4]. Zależności między zmiennymi występujące w danych stanach (szczególnie w stanie zakończenia działania algorytmu) pozwalają wnioskować o takich własnościach, jak niesprzeczność, poprawność częściowa, zakończenie, itp. Wprowadzone formalne lub półformalne (*ang.* semi-formal) metody analizy pozwalają wyrazić warunki poprawnego sterowania oraz wykazać, że zachowanie programu w zaznaczonych stanach będzie dokładnie takie, jak przedstawiają to dodatkowe warunki narzucone na zmienne programowe (kontrolowane podczas wykonywania programu) [8], [9]. Problem poprawności programu sterującego może być rozważany na kilku płaszczyznach. Mamy więc do czynienia z poprawnością algorytmu (problem niesprzeczności, poprawności częściowej czy zakończenia), poprawnością implementacji (problem certyfikacji narzędzi programowania, takich jak: edytory, kompilatory czy systemy operacyjne), poprawnością programów zróżnicowanych (problem równoważności).

9. ZALECENIA DLA OPROGRAMOWANIA BEZPIECZNYCH SYSTEMÓW STEROWANIA RUCHEM I METODY TEORII POPRAWNOŚCI PROGRAMÓW

Oprogramowanie sterujące tworzy wraz z komputerowym sprzętem zintegrowany system sterowania spełniający wysokie wymagania bezpieczeństwa. Chociaż na etapie projektowania sprzęt i oprogramowanie są traktowane niezależnie, to jednak integracja obu tych części wymaga specjalnych procedur nie tylko na etapie uruchamiania oprogramowania, ale praktycznie na każdym etapie powstawania oprogramowania, tak jak to przedstawia schemat na rys. 3 obowiązujący przy projektowaniu komputerowych systemów SRK [10].



Rys. 4. Schemat tworzenia oprogramowania dla systemu bezpiecznego
Fig. 4. Software creation schema for safety system

10. MATEMATYCZNA TEORIA POPRAWNOŚCI PROGRAMU STERUJĄCEGO

Metody wymienione w tabl. 3 i 4 odpowiadają pewnym pojęciom stosowanym w matematycznej teorii programowania i mogą być określone w sposób bardziej sformalizowany. Do opisu programu można zaproponować intuicyjny model relacyjny [8], [11], w którym relacja binarna przypisana do instrukcji (zarówno prostej, jak i złożonej) transformuje stan zmiennych, których wartości ulegają zmianie podczas wykonania instrukcji. Program P jest funkcjonalnym złożeniem m modułów $\{P_1, P_2, \ldots, P_i, \ldots, P_m\}$:

$$P = F(P_1, P_2, \ldots, P_i, \ldots, P_m) \tag{1}$$

Moduły mogą mieć postać bloków, podprogramów, procedur, funkcji. Na przykład złożenie modułów jest złożeniem funkcji które realizują:

$$F(P_i; P_j) = F(P_i)\, F(P_j) \tag{2}$$

Rachunek relacji binarnych pozwala wyznaczyć funkcję F interpretującą wykonanie modułu z uwzględnieniem spełnienia warunku ~c, oraz takie operacje funkcjonalne, jak złożenie i suma. Operacje takie można wprowadzić dla innych konstrukcji stosowanych w algorytmicznych językach programowania. Każdy moduł wyższego poziomu może być zdekomponowany na mniejsze moduły przez zastosowanie przedstawionych powyżej operacji strukturalnych. Moduły ostatniego poziomu są już dekomponowane na sekwencje instrukcji,

w podobny sposób mogą być zdekomponowane instrukcje złożone do sekwencji instrukcji prostych.

$$P_i = \{IN_{kl}\}i_{\ k,l\ =1,2,\ldots,n},$$

$$IN_{kl} = \{\ IN_{k\ k+1}, IN_{k+1\ k+2}, \ \ldots, \ IN_{l-1\ l}\}$$

$$IN_{s\ t} = \{\ IN_{s\ s+1}, IN_{s+1\ s+2}, \ \ldots, \ IN_{t-1\ t}\} \qquad (3)$$

Reguły kompozycji/dekompozycji programu (1), (2), (3) są przykładem tzw. reguł semantycznych opracowanych dla danego języka programowania. Taki sposób programowania zakładający dekompozycję programu na moduły oraz na elementy niższego poziomu (procedury, instrukcje złożone, instrukcje proste) jest nazywany programowaniem strukturalnym [10]. Popularne języki programowania są bardziej skomplikowane i uwzględniają bardziej złożone konstrukcje, dodatkowo należy też uwzględnić obiekty (PASCAL, C), mechanizmy semaforowe przy programowaniu procesów współbieżnych (ADA, MODULA) czy zdarzenia i upływ czasu w programowaniu czasu rzeczywistego (języki RT) [9]. Formalne metody analizy programu wprowadzają własności zdefiniowane w matematycznej teorii programowania. Każdy moduł (lub element niższego poziomu) może być scharakteryzowany przez specyfikację typu wejście-wyjście

$$< \varphi, \boldsymbol{P}, \psi > \qquad (4)$$

gdzie φ jest warunkiem początkowym (pre-condition) opisującym zmienne wejściowe w stanie początkowym, przed wykonaniem programu P, zaś ψ jest warunkiem końcowym (post-condition) opisującym oczekiwane wyniki – wartości zmiennych wstanie końcowym po zakończeniu programu $\boldsymbol{P}$.

Poprawność jest cechą trójki $< \varphi, \boldsymbol{P}, \psi >$

$$C\ (< \varphi, \boldsymbol{P}, \psi >) \qquad (5)$$

i może być wykazana przez zastosowanie odpowiedniego aparatu matematycznego (rachunek predykatów, rachunek relacji binarnych lub inny system wnioskowania). Na przykład poprawność częściowa C może być wyrażona w rachunku relacji binarnych w postaci:

$$C\ (< \varphi, \boldsymbol{P}, \psi >) \equiv \varphi\ [\boldsymbol{P}\ ] \subseteq \psi \qquad (6)$$

gdzie $[\boldsymbol{P}]$ jest relacją binarną programu $\boldsymbol{P}$ przedstawiającą transformację zmiennych podczas wykonania programu $\boldsymbol{P}$ (zbiór wejść przetransformowanych przez program

φ [P] powinien zawierać się w zbiorze pożądanych wyników ψ). Każda taka własność programu (jak w przypadku poprawności częściowej) powinna być dowiedziona przez zastosowanie formalnego lub półformalnego systemu wnioskowania. Ponieważ obliczenie relacji wynikowej (relacji wejście-wyjście) dla programu czy modułu jest stosunkowo trudne, globalny warunek poprawności może być zdekomponowany do zbioru warunków lokalnych:

$$C\ (\varphi, \boldsymbol{P}, \psi) \equiv \{ C\ (\varphi_i, \boldsymbol{P}_i, \psi_i) \}_{i=1,2,\dots,m}$$

$$C\ (\varphi_i, \boldsymbol{P}_i, \psi_i) \equiv C\ (\varphi_i, (\{IN_{kl}\}i_{\ k,l=1,2,\dots,n}, \{Q_j\}i_{\ j=1,2,\dots,n}), \psi_i) \qquad (7)$$

Taka dekompozycja uwzględnia instrukcje złożone IN_{kl} i odpowiadające im warunki początkowe i końcowe Q_k i Q_l zapewniające poprawność częściową. Postępowanie takie może być stosowane aż do uwzględnienia elementarnych instrukcji elementarnych $\{IN_{ij}\}$ i odpowiadających im parom warunków początkowych $\{Q_i, Q_j\}$. Dowodzenie poprawności odwołuje się do wprowadzonych reguł semantycznych, na tej podstawie tworzone są reguły wnioskowania oparte na zastosowanej teorii matematycznej (np. algebrze relacji czy rachunku predykatów). Takie wprowadzenie do programu specyfikacji warunków poprawności (φ, $\{Q_j\}$, ψ) prowadzi do ***programowania specyfikowanego*** [7], [10]. W niektórych językach programowania są przewidziane specjalne konstrukcje syntaktyczne zapewniające specyfikację warunków poprawności (np. wyrażenia logiczne specyfikowane w konstrukcji *assert* w języku ADA). Warunek początkowy φ określa na wejściu zbiór wejść dla potencjalnych testów, warunki $\{Q_i\}$ określają wyniki w stanach pośrednich 'i', zaś warunek końcowy ψ określa oczekiwany zbiór wyników w stanie końcowym. Zbiór danych wejściowych dla takich testów może być przedstawiony w postaci sumy podzbiorów:

$$\varphi = \varphi 1 \cup \varphi 2 \cup .. \cup \varphi j \cup ... \cup \varphi m \qquad (8)$$

(przy czym w ostatecznym podziale zbiór φ może być sumą elementarnych pojedynczych wejść - wartości zmiennych w stanie początkowym). Proces testowania $t(\varphi j, \boldsymbol{P})$ odpowiada wykonaniu programu od stanu początkowego '1' do stanu pośredniego 'i', w którym powinien być spełniony warunek Q_i (w stanie końcowym 'n' spełniony jest warunek $Q_n \subseteq \psi$).

$$< \varphi j, \boldsymbol{P}, Q_i > \equiv t(\varphi j, \boldsymbol{P}) \Rightarrow Q_i \qquad (9)$$

W rachunku relacji binarnych testowanie jest złożeniem:

$$t(\varphi j, \boldsymbol{P}) \Rightarrow Q_i \equiv \varphi j\ [\ IN_{ji}\ ] \subseteq Q_i \qquad (10)$$

gdzie $[IN_{ji}]$ jest relacją wynikową opisująca przejście od stanu początkowego '1' do stanu pośredniego 'i' (lub końcowego 'n'), gdzie realizowany jest test. W ten sposób można wprowadzić ***symboliczne testowanie*** (połączone z symbolicznym wykonaniem) programu [8], [9], [11]. Relacja wynikowa opisuje sumę wszystkich możliwych przejść pomiędzy zaznaczonymi punktami:

$$[IN_{kl}] = [IN^1_{k\,i+1}][\,IN^1_{k+1\,k+2}\,] \ldots [IN^1_{l-1\,l}] \cup [IN^2_{k\,k+1}][\,IN^2_{k+1\,i+2}\,] \ldots [IN^2_{l-1\,l}] \cup \ldots$$
$$\cup [IN^n_{k\,k+1}][\,IN^n_{k+1\,k+2}\,] \ldots [IN^n_{l-1\,l}] = \bigcup_{i=1}^{n} [IN^i_{k\,k+1}][\,IN^i_{k+1\,k+2}\,] \ldots [IN^i_{l-1\,l}] \tag{11}$$

Dla wprowadzonych wcześniej operacji strukturalnych (złożenia, rozejścia i iteracji) można podać reguły obliczania relacji wynikowych. Dla operacji złożenia:

$$[\,IN_{ik}\,,\,IN_{kj}\,] = [IN_{ik}][IN_{kj}] \tag{12}$$

Jeżeli program umożliwia przeprowadzenie specjalnej reakcji w przypadku, gdy nie będą spełnione warunki poprawnej realizacji programu (opisane przez Q_i)

$$\{\varphi\,[\,[IN_{1j}\,]\}^f \not\subseteq Q_i \;\Rightarrow\; \{\varphi\;[IN_{1j}]^f\}[IN^s{}_{jn}] \subseteq \psi \tag{13}$$

gdzie $\{\varphi\,[\,[IN_{1j}\,]\}^f$ jest błędną realizacją danego wejścia programu, to mamy do czynienia z ***programowaniem defensywnym*** [7], [9], w którym ma miejsce reakcja bezpieczeństwa inicjująca sterowanie awaryjne (opisane relacją $[IN^s{}_{jn}]$).

W pewnych aplikacjach oprogramowania sterującego mamy do czynienia z programami (modułami) równoważnymi. W celu detekcji błędów porównuje się wyniki uzyskane przez m programów pracujących równolegle (współbieżnie). Dla każdej pary takich programów P' i P" można wprowadzić następującą relację podobieństwa określoną na porównywalnych, poprawnych stanach w obu programach:

$$S \subseteq \{Q'_i\} \times \{Q''_i\} \tag{14}$$

tak aby poprawność wynikała z następującego diagramu:

$$[\boldsymbol{P'}]\,S \subseteq S\,[\boldsymbol{P''}] \tag{15}$$

Takie podejście do projektowania programów równoważnych nosi nazwę ***programowania zróżnicowanego*** [9], [11].

11. PRZYKŁADOWY MODEL OPROGRAMOWANIA

Rysunek 5 przedstawia poprawne stany pracy sygnalizacji przejazdowej wraz z odpowiadającymi im stanami urządzeń ostrzegania. Kolejne elementy wektora odpowiadają następującym zmiennym:

TabWejsc:(Tor1Ruch, Tor1Kier, Cz1, Cz2, Cz3, Tor2Ruch, Tor2Kier, Cz4, Cz5, Cz6), TabWyjsc:(To1b, To1p, To2b, To2p, To3b, To3p, To4b, To4p, Sygnały)

Zmienne ToriRuch informują, czy po danym torze odbywa się ruch, zmienne ToriKier określają, czy kierunek ruchu jest właściwy czy nie, pozostałe zmienne Czi odpowiadają za informacje o zajętości strefy i-tego czujnika detekcji pojazdu. Zmienne Toib oraz Toip wskazują odpowiednią tarczę ostrzegawczą przejazdową oraz odpowiednie komory świateł (b - białe, p - pomarańczowe). Zmienna Sygnały informuje o załączeniu ostrzegania na przejeździe kolejowym.

```
const
max = 64;
T = True;
F = False;
   {tablica możliwych stanów wejściowych}
   TabWejsc: array [1..max, 0..9] of Boolean=
               ((F,F,F,F,F,F,F,F,F,F),
                (F,F,F,F,F,F,T,F,F,F),
                (F,F,F,F,F,T,F,F,F,F),
                (F,F,F,F,F,T,F,F,F,T),
                (F,F,F,F,F,T,T,F,F,F),
                                  :

   {przypisanie odpowiednich wyjść do stanów
    wejściowych}
   TabWyjsc: array[1..max, 0..8] of Boolean =
               ((F,F,F,F,F,F,F,F,F),
                (F,F,F,F,F,F,F,F,F),
                (F,F,F,F,F,F,F,F,F),
                (F,F,T,F,F,F,F,F,T),
                (F,F,F,F,F,F,F,F,F),
                                  :
```

Rys. 5. Sposób zapisu poprawnych stanów pracy sygnalizacji wraz z odpowiadającymi im stanami urządzeń ostrzegania

Fig. 5. Corectness of signalling states with appropriate states of warning devices

Można zauważyć, że tablica (rys. 5) traktuje jako prawidłowe stany, w których dwa sąsiednie czujniki (np. Cz1 i Cz2) przyjmują wartość true. Odpowiada to dwóm sytuacjom (dla przykładu Cz1 i Cz2): prawidłowa sekwencja: pociąg zjechał z czujnika 1, a następnie wjechał na czujnik 2, nieprawidłowa sekwencja: obydwa czujniki wskazują zajętość. W celu odróżnienia sekwencji prawidłowej od nieprawidłowej i zagwarantowania odpowiedniej reakcji systemu, a także rozpoznania kierunku poruszającego się pojazdu każdy z czujników podzielony został na 2 strefy: wjazdową i wyjazdową. Oprócz tego wprowadzono dodatkową zmienną typu tablicowego *Licznik [i]*, która zmienia swój stan w miarę przejeżdżania przez pojazd kolejnych stref czujnika. Nieprawidłowa sekwencja przejazdu przez czujniki Cz1 i

Cz2 jest rozpoznawana przy wykorzystaniu zmiennej Licznik1[1] i Licznik[2]. Poszczególne elementy wektora *Licznik[i]* są uzależnione od siebie, tak jak to przedstawia tabela 6. Symbolem „M" oznaczone zostały stany możliwe do wystąpienia w systemie w przypadku pracy prawidłowej, symbolem „U" stany usterkowe. Symbol „$M/_{Czi}$" dotyczy sytuacji, gdy stan jest możliwy do wystąpienia, pod warunkiem że zmienna opisująca stan czujnika *Czi=True,* czyli wcześniej czujnik ten zadziałał.

Tablica 6

Zależności pomiędzy zmienną Licznik [i]

Wartość zmiennej Licznik[1]	0	1	2	3	4
Cz2=True	**U/M**	**U**	**U**	**U**	**M**
Wartość zmiennej Licznik[3]	0	1	2	3	4
Cz2=True	**M**	**U**	**U**	**U**	**U/M**

Zależności pomiędzy zmiennymi zaimplementowane zostały w kilku miejscach (modułach) programu. Dodatkowo system uwzględnia zależności, takie jak np. powiązanie między kolejnymi stanami i-tego detektora pojazdu. W przypadku gdy sekwencja nie zostanie zachowana, następuje obsługa sytuacji uznanej za awaryjną. Na rysunku 6 został przedstawiony fragment reprezentacji kodowej programu odpowiadający blokowi decyzyjnemu odpowiedzialnemu za rozpoznawanie sytuacji awaryjnej.

```
                    :
                    :
Jest_Usterka := True;
  for i:=1 to max_do
    if (TabWejsc[i,0] = Tor1Ruch) and
       (TabWejsc[i,1] = Tor1Kier) and
       (TabWejsc[i,2] = Cz1)      and
       (TabWejsc[i,3] = Cz2)      and
       (TabWejsc[i,4] = Cz3)      and
       (TabWejsc[i,5] = Tor2Ruch) and
       (TabWejsc[i,6] = Tor2Kier) and
       (TabWejsc[i,7] = Cz4)      and
       (TabWejsc[i,8] = Cz5)      and
       (TabWejsc[i,9] = Cz6)      then
       begin
         Form1.Wyjscia(TabWyjsc[i,0], TabWyjsc[i,1], TabWyjsc[i,2],
                       TabWyjsc[i,3], TabWyjsc[i,4], TabWyjsc[i,5],
                       TabWyjsc[i,6], TabWyjsc[i,7], TabWyjsc[i,8]);
         if not Sekwencja_Czujnikow then Jest_Usterka := False;
         Break;
       end;
  if Jest_Usterka then Usterka_Czujniki
  else if ((Cz1 and Cz2) or (Cz2 and Cz3) or (Cz4 and Cz5)
  or (Cz6 and Cz5)) then Usterka_Czujniki;

  if Sekwencja_Liczniki then Usterka_Sekwencji;
                    :
                    :
```

Rys. 6. Fragment reprezentacji kodowej programu związany z obsługą usterek
Fig. 6. Fragment of the program code representation connected with malfunctions service

System uznaje za awaryjne następujące stany czujników:

- jeżeli na torze nie ma ruchu, tzn. zmienna *ToriRuch=False*, oraz którykolwiek czujnik *Czi=True*,

- jeżeli na torze 1 lub 2 jednocześnie pojawia się sekwencja *Cz1, Cz2, Cz3=True* lub *Cz4, Cz5, Cz6=True*,
- jeżeli jednocześnie pojawiają się sekwencje *Cz1, Cz3=True* oraz *Cz4, Cz6=True*,
- jeżeli w systemie czujnik 2 lub 5 sygnalizuje zajętość (*Cz2 lub Cz5=True*) bez uprzedniego zajęcia odpowiednio czujnika 1 lub 3 oraz 4 i 6,
- gdy pojawia się jedna z następujących sekwencji bez uwzględnienia zależności pomiędzy zmiennymi Licznik[i]:
  - Cz1, Cz2=True,
  - Cz3, Cz2=True,
  - Cz4, Cz5=True,
  - Cz6, Cz4=True.

Dodatkowo w systemie uwzględniono usterki związane z niezachowaniem sekwencji przejazdu przez i-ty czujnik. Uwzględnione zostały tu tylko stany możliwe do zrealizowania – wszystkie pozostałe powodują uruchomienie obsługi usterki. Reakcja na każdą z tych usterek polega na włączeniu ostrzegania na przejeździe kolejowym oraz pomarańczowych świateł na tarczach ostrzegawczych przejazdowych. Jeżeli awarii ulega czujnik 2 lub 5, wtedy tarcze ostrzegawcze sygnalizują usterkę po stronie kierunku właściwego i niewłaściwego. Ostrzeganie zostaje wyłączone dopiero po usunięciu usterki (restartowanie układu).

W przypadku pojawienia się awarii jest ona wykrywana za pomocą wywoływanej cyklicznie co 500 μs procedury *TimerPoprawnosc*. Po rozpoznaniu stanu usterkowego (jeszcze nie sklasyfikowanego) procedura przekazuje sterowanie do odpowiedniego podprogramu sterującego, odpowiedzialnego za klasyfikację i odpowiednią reakcję (np. zaświecenie odpowiednich tarcz ostrzegawczych). Pozostałe stany awaryjne (np. awarie urządzeń ostrzegania) nie zostały uwzględnione w systemie. Zostaną one uwzględnione jako kontynuacja pracy.

12. PODSUMOWANIE

Wprowadzając nowe systemy komputerowe w miejsce dotychczasowych układów (np. w przekaźnikowych systemach srk), należy zapewnić co najmniej taki sam poziom funkcjonalności czy bezpieczeństwa względem parametrów niezawodnościowych czy też eksploatacyjnych. Specyfiką tych nowych standardów jest podział na część sprzętową oraz na oprogramowanie. Analiza niezawodności na poziomie sprzętu opiera się na znanych z teorii niezawodności metodach uwzględniających głównie strukturę powiązań pomiędzy podsystemami, urządzeniami czy elementami systemu. Analizując oprogramowanie należy

posługiwać się innymi metodami szacowania prawdopodobieństwa poprawnego wykonania programu związanymi np. z teorią procesów Markowa [9], [12].

Projektowanie programów poprawnych wiąże się ze stosowaniem specjalnych procedur i metod programowania, jak programowanie systematyczne, strukturalne, specyfikowane czy defensywne [10]. Istotną sprawą jest zapewnienie zarówno poprawności pod względem syntaktycznym, jak i semantycznym. Jakkolwiek poprawność syntaktyczna nie jest wielkim problemem, ze względu na ścisłe reguły składni języków programowania, to poprawność semantyczna nie jest już tak jednoznacznie identyfikowalna. Stąd też liczne metody dowodzenia poprawności semantycznej, które najczęściej bazują na sprawdzeniu, czy zachowanie programu będzie takie, jak wskazują to dodatkowo narzucone na zmienne programowe warunki. Jest to bardzo istotne zagadnienie w przypadku systemów sterowania związanych z bezpieczeństwem, jak to ma miejsce w systemach NSRK (Nadzoru i Sterowania Ruchem Kolejowym). Według UIC i CENELEC na każdym etapie powstawania oprogramowania systemu bezpiecznego, od specyfikacji wymagań do zatwierdzenia oprogramowania, stosować należy odpowiednie analizy i metody tworzenia oprogramowania [8]. Są to elementy istotne zwłaszcza w świetle przyszłego członkostwa Polski w strukturach Unii Europejskiej, gdzie obecnie obowiązują raporty oraz standardy CENELEC.

Literatura

1. Kopetz H.: Niezawodność oprogramowania, Wydawnictwa Naukowo-Techniczne, Warszawa 1980.
2. Myers G. J.: Projektowanie niezawodnego oprogramowania, Wydawnictwa Naukowo-Techniczne, Warszawa 1980 r.
3. Lewiński A., Konopiński L., Siergiejczyk M.: Prognozowanie niezawodności i bezpieczeństwa komputerowych systemów sterowania ruchem kolejowym, Przegląd Kolejowy, Nr 3/2000, s. 5÷11.
4. Wiater I.: Systemy komputerowe o zwiększonej niezawodności dla urządzeń automatycznego sterowania pociągami, Telekomunikacja i Sterowanie Ruchem, Technika Transportu Szynowego, Nr 6-7/1995, s. 49÷53.
5. CENELEC EN 50128, Railway Applications, Software for railway control and protection systems, 1997.
6. Alagic S., Arbib M.A.: Projektowanie programów poprawnych i dobrze zbudowanych, WNT, Warszawa 1982.

7. Blikle A.: A specified programming, Prace Instytutu Podstaw Informatyki PAN, Warszawa 1979.
8. Dembiński P., Małuszyński J.: Matematyczne metody definiowania języków programowania, WNT, Warszawa 1981.
9. Jones C.B.: Projektowanie oprogramowania metodą systematyczną, WNT, Warszawa 1984.
10. Lauber R., Konakovsky R., Reinshagen K.P.: Structured documentation method for safety related controlled systems, Proceedings of IFAC 78 Congress, Helsinki 1878.
11. Lewiński A.: Problemy oprogramowania bezpiecznych systemów komputerowych w zastosowaniach transportu kolejowego, Politechnika Radomska im. K. Pułaskiego, MONOGRAFIA nr 49, Radom 2001.
12. Opracowanie CNTK, Zakład Sterowania Ruchem Kolejowym: Wymagania bezpieczeństwa dla urządzeń sterowania ruchem kolejowym, Warszawa, luty 1998.
13. Lewiński A., Trzaska K.: The software safety of railway control system applied to cross level protection, materiały III Międzynarodowej Konferencji Transport Systems Telematic, Katowice 2003.
14. Lewiński A., Trzaska K.: Projektowanie bezpiecznego oprogramowania dla systemu samoczynnej sygnalizacji przejazdowej, materiały VII edycji konferencji TRANSCOMP 2003, Zakopane, grudzień 2003.

Recenzent: Dr hab. inż. Andrzej Lewiński, Prof. Politechniki Radomskiej

Abstract

The paper deals with problems of the safe software railway applications designing when the software correctness theory is applied. The hardware reliability analysis is based on well known methods in the reliability theory that mainly include the connection structure between subsystems, and devices or elements of the system. Analyzing the software there is a need to apply other methods of the probability estimation of the program correct execution. Creating acceptable programs depends of applying special procedures and programming methods (i.e. systematic, structural, defensive, or specified programming). Very important is to provide the syntactic and semantic correctness. As far as the syntactic correctness is not a problem due to precise roles of the programming languages, the semantic correctness is not clearly specified.

And that is why there are many methods of proving the semantic correctness. Those methods are based on checking if the program runs according to the applied variable program conditions. That is very important for the control systems working for the traffic safety such as NSRK (Management and Railway Traffic Control) systems. According to UIC and CENELEC at each stage of the safe system software designing (beginning from specifications up to approving the conception) there is a necessity to apply sufficient analyses and methods of the software creation. Those elements are very important especially due to the EU expansion. The CENELEC standards and statements have been there already in use.

As a sample the authors have introduced the simplified system of automatic light crossing signals.