Metody wytwarzania i zastosowania systemów czasu rzeczywistego

Praca zbiorowa pod redakcją
Leszka Trybusa i Sławomira Samoleja

Wydawnictwa Komunikacji i Łączności
Warszawa 2010

ROZDZIAŁ 12

NOWE SPOJRZENIE NA POMIARY JAKOŚCI W SYSTEMACH KOMPUTEROWYCH CZASU RZECZYWISTEGO I SYSTEMACH KRYTYCZNYCH DLA BEZPIECZEŃSTWA[1]

Andrew J. KORNECKI[2], Janusz ZALEWSKI[3]

Rozdział przedstawia nowe podejście do pomiarów jakości oprogramowania w systemach komputerowych czasu rzeczywistego i w ściśle związanych z nimi – systemach komputerowych krytycznych dla bezpieczeństwa. Podejście jest oparte na rozważaniach filozoficznych Hermanna von Helmholtza i polega na wyróżnieniu trzech zasadniczych elementów w metodologii pomiarów: mierzonej właściwości (atrybutu fizycznego), metryki (jednostki pomiarowej) i narzędzia pomiarowego. Operując się na tej koncepcji, autorzy przedstawiają wyniki własnych prac dotyczących pomiarów wrażliwości systemów komputerowych czasu rzeczywistego i jakości narzędzi programistycznych do wytwarzania oprogramowania i sprzętu dla systemów komputerowych krytycznych dla bezpieczeństwa.

1. WPROWADZENIE

Problematyka badania i zapewniania jakości oprogramowania systemów komputerowych czasu rzeczywistego i systemów krytycznych dla bezpieczeństwa jest bardzo rozległa i niejednorodna. Klasyczne podejście nie odbiega od tradycyjnych metod oceny jakości oprogramowania, np. podsumowanych przez Zuse [16], a w systemach krytycznych dla bezpieczeństwa – skatalogowanych ponad dwadzieścia lat temu przez European Workshop on Industrial Computer System [11]. Nowsze wyniki są regular-

[1] Praca częściowo finansowana ze środków U.S. Federal Aviation Administration (FAA), w ramach kontraktów DTFA0301–C–00048 i DTFACT–07–C–00010. Wyniki prezentowane w tym artykule niekoniecznie reprezentują poglądy FAA.

[2] Dept. of Electrical, Computer, Software and System Engineering, Embry–Riddle Aeronautical University; Daytona Beach, FL 32114, USA; kornecka@erau.edu

[3] Dept. of Computer Science, Florida Gulf Coast University; Fort Myers, FL 33965, USA; zalewski@fgcu.edu

nie publikowane w cyklu sympozjów SAFECOMP [1] (por. też przegląd dokonany przez autorów [15]).

W niniejszym rozdziale autorzy opierają się na koncepcji pomiarów wielkości fizycznych, sformułowanej po raz pierwszy przez niemieckiego lekarza, fizyka i filozofa nauki, Hermanna von Helmholtza, w drugiej połowie XIX wieku [4]. Zasadniczym elementem tego podejścia jest wyróżnienie trzech elementów niezbędnych do przeprowadzenia procesu pomiarowego: (1) mierzonej właściwości fizycznej (nazywanej też atrybutem), (2) jednostki miary (tj. metryki, ang. *metric*) i (3) samego instrumentu pomiarowego, nazywanego często miarą (ang. *measure*).

Przy pomiarze wielkości fizycznych konieczne jest ustalenie związku między mierzoną wielkością a wskazaniami przyrządu pomiarowego. Związek ten istnieje niezależnie od fizycznego obiektu, który podlega pomiarowi, jak też niezależnie od użytego przyrządu pomiarowego. Jest to więc związek obiektywny. Charakterystyką tego związku jest metryka (jednostka miary), a sam proces mierzenia ma zawsze charakter obliczeniowy.

Metryka, czyli jednostka miary, determinuje skalę pomiarową, na ogół, choć nie zawsze, liczbową i czasami – nieliniową. Metryka jest często mylona z miarą, tj. procedurą znajdowania punktu na tej skali dla mierzonej właściwości pewnego obiektu. Choć terminy te bywają mylone, jedno jest pewne, że zawsze istnieją dwa oddzielne pojęcia:

- jedno dotyczące urządzenia lub procedury do wykonywania pomiaru i otrzymywania wyniku (w niniejszym rozdziale nazywane *miarą*),
- drugie dotyczące wzorca lub jednostki używanej jako obiektywny standard w wykonywaniu pomiaru przy użyciu tego urządzenia lub procedury (nazywane tutaj *metryką*).

Krótko mówiąc, związek ten, a zatem proces pomiarowy, można wyrazić bardzo prozaicznie, jak poniżej (symbol <==> jest symbolem odpowiedniości, a nie równoważności logicznej):

właściwość <==> metryka <==> miara

Tak więc, stosuje się miarę (przyrząd pomiarowy), która używając metryki prowadzi do znalezienia (obliczenia) liczbowej wartości mierzonej właściwości danego obiektu.

Łatwo podać trywialne przykłady tych związków będące w codziennym użyciu, na przykład:

- długość (odległość), dla której metryką jest metr, a miarami – różne urządzenia, np. taśma miernicza, suwmiarka, mikrometr, drogomierz itd.,
- czas (upływ czasu), dla którego metryką jest sekunda, a miarami różnego rodzaju chronometry, zegary, liczniki itp.,
- prędkość, dla której metryką może być, na przykład, kilometr na godzinę, a miarą – prędkościomierz, urządzenie radarowe lub inne instrumenty.

Aczkolwiek związki te wydają się trywialne na codzienny użytek, to z pewnością takimi nie są w sensie naukowym i inżynierskim.

Przede wszystkim, należy zwrócić uwagę, że znalezienie i zdefiniowanie właściwej metryki jest, w ogólnym przypadku, procesem skomplikowanym, gdyż powinna być zachowana maksymalna obiektywność umożliwiająca niezmienność wyniku pomiaru niezależnie od warunków pomiarowych. Dla przykładu, definicja metra brzmi następująco [10]:

> *metr* – długość drogi przebytej przez promień świetlny w próżni w przeciągu 1/299 792 458 części sekundy

a bardziej skomplikowana definicja sekundy [10]:

> *sekunda* – czas trwania 9 182 631 770 okresów promieniowania odpowiadającego przejściu między dwoma poziomami struktury nadsubtelnej stanu podstawowego atomu cezu 133.

Nawet jeśli uda się poprawnie zdefiniować metrykę dla pomiaru danej właściwości, konieczne jest też określenia miary, a więc procedury pomiarowej, lub wręcz – zbudowanie odpowiedniego przyrządu pomiarowego.

Przenosząc te rozważania na systemy oprogramowania, można powiedzieć że mamy tu do czynienia z trzema podstawowymi problemami. Gdy mówi się o jakości oprogramowania: (1) niezbędne jest określenie mierzonej *właściwości*, tzn. trzeba odpowiedzieć na pytanie, co w danej sytuacji rozumie się przez jakość oprogramowania, jaka jego cecha charakterystyczna będzie podlegała mierzeniu; (2) konieczne jest określenie *metryki*, a więc podstawowej jednostki miary, która umożliwi ustalenie skali porządkowej i obiektywne wykonywanie pomiarów; (3) dla rozróżnienia różnych systemów oprogramowania na skali porządkowej, pod względem mierzonej właściwości, należy wprowadzić *miarę*, tj. metodę obliczeniową (a najlepiej – przyrząd), która pozwoli na liczbowe przedstawienie mierzonej właściwości przy użyciu zdefiniowanej metryki.

Sytuacja w programowaniu jest jednak znacznie trudniejsza niż w fizyce, gdyż nie tylko nie daje się tak precyzyjnie zdefiniować metryk, jak dla metra lub sekundy, ale – co więcej – obiekty których właściwości podlegają pomiarowi, tj. oprogramowanie, nie wydają się istnieć fizycznie. Warto, jednak, przypomnieć, że procesy prowadzące do opracowania precyzyjnej definicji metryki, na przykład – dla długości (odległości), były niezwykle długotrwałe, od mierzenia odległości – wiele lat temu – w stopach, do dzisiejszego pojęcia metra i jego definicji za pomocą promienia świetlnego. Autorzy wyrażają przekonanie, że podobnie rzecz się ma z pomiarami właściwości oprogramowania.

W literaturze przedmiotu pojawiają się podobne podejścia, np. do określania niezawodności oprogramowania na podstawie normy IEEE Std 982.1 i IEEE Std 982.2 [9, 13], ale są one niepełne i niewystarczająco zdefiniowane. Nie dotyczą też

systemów czasu rzeczywistego lub systemów do zastosowań krytycznych dla bezpieczeństwa.

W pozostałych częściach rozdziału, przedstawiono wyniki własnych prac autorów nad pomiarami jakości oprogramowania, w sensie określonym powyżej. Część druga traktuje o badaniu wrażliwości i dynamiki oprogramowania systemów czasu rzeczywistego, a część trzecia – o próbach oceny jakości narzędzi programistycznych do wytwarzania oprogramowania stosowanego w systemach krytycznych dla bezpieczeństwa.

2. POMIAR JAKOŚCI SYSTEMÓW CZASU RZECZYWISTEGO

W niniejszej części przedstawiono ogólne podejście do pomiarów jakości systemów czasu rzeczywistego i wprowadzono pojęcia wrażliwości i dynamiki oprogramowania.

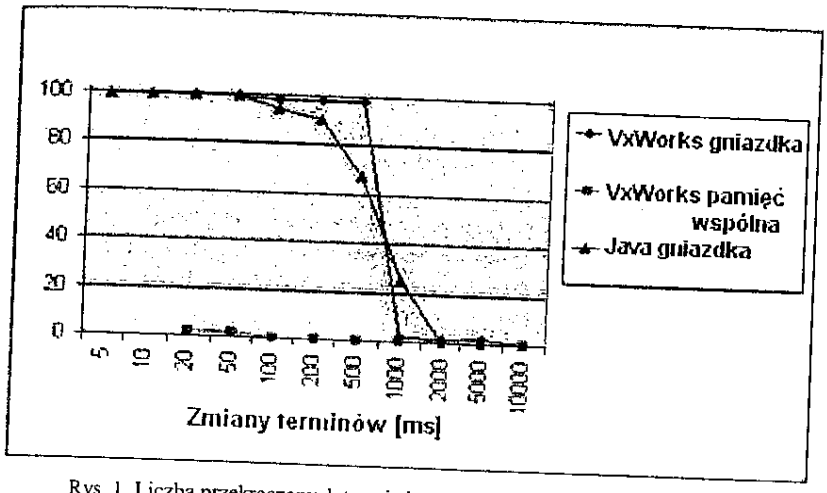
2.1. PODEJŚCIE OGÓLNE

W pomiarach jakości oprogramowania systemów czasu rzeczywistego zasadniczą rolę odgrywa zachowanie oprogramowania w sensie odpowiedzi na bodźce zewnętrzne. Podstawowym parametrem jest tu stopień spełnienia zadania (lub zadań) przed upływem *terminu* (ang. *deadline*). Jednakże, *terminowość* wykonania zadania jest parametrem binarnym, tzn. stwierdzającym że zadanie zostało wykonane lub nie, i nie daje odpowiedzi na dokładniejsze pytania co do zachowania systemu, np. jaka część systemu przyczynia się bardziej niż inne do zwiększenia *czasu odpowiedzi* na bodźce zewnętrzne, jak szybko system adaptuje się do zmian w otoczeniu lub – jaki jest wpływ czynników zewnętrznych na *szybkość reakcji*. Odpowiedzi na te i podobne pytania dotyczące zachowania systemu wymagają głębszej analizy.

Metoda analizy zaproponowana przez jednego z autorów [14] polega na badaniu zachowania systemu przy zmianach wymagań na terminowość wykonania zadań. Dla ustalonej, standardowej architektury oprogramowania systemu czasu rzeczywistego, złożonej z typowych modułów lub komponentów [5, 12], ocenia się reakcję modułu na skracanie i przedłużanie terminu zakończenia zadania w odpowiedzi na komunikaty otrzymywane od innych modułów. Wielkościami mierzonymi są: (1) całkowity czas o który przekroczone są terminy podczas stu eksperymentów i (2) liczba terminów przekroczonych o X procent (np. 2%) w stosunku do wymagań podczas stu eksperymentów.

Przykładowe wyniki eksperymentalne dla przekroczenia terminów o 2% przedstawiono na rys. 1. W eksperymencie porównano przekazywanie komunikatów przy użyciu gniazdek (ang. *socket*) w systemie operacyjnym VxWorks i systemie języka

Java oraz z użyciem pamięci wspólnej w systemie VxWorks. Dokładniejsze informacje i pełna analiza wyników umieszczone są w pracy [14].



Rys. 1. Liczba przekroczonych terminów o 2% przy stu eksperymenatch [14]

Zastosowanie tego rodzaju eksperymentów do analizy oprogramowania systemów czasu rzeczywistego pod względem wrażliwości i dynamiki omowiono poniżej.

2.2. POJĘCIE I ANALIZA WRAŻLIWOŚCI OPROGRAMOWANIA

Obserwacja krzywych przedstawionych na rysunku 1 prowadzi do pytania: Jak szybko pogarsza się jakość systemu pod względem terminowego wykonywania zadań, gdy terminy ulegają skróceniu? Nietrudno zauważyć, że krytyczne dla tego systemu są zmiany terminów między 200 ms a 2000 ms. W tych przypadkach, jakość systemu rozumiana jako podatność na przekraczanie terminów wykonywania zadań, pogarsza się szybciej w implementacji gniazdek w VxWorks (napisanej w języku C) niż przy użyciu systemy języka Java, co wynika z bardziej stromego nachylenia odpowiedniej krzywej dla systemu VxWorks.

Powyższa obserwacja ma daleko idące konsekwencje, bo wykazuje, że stosując zaprezentowaną metodykę pomiarów można ocenić podatność systemu czasu rzeczywistego na zmienność warunków zewnętrznych, czego nie widać bezpośrednio gdy analizuje się tylko spełnianie narzuconych na zadania deterministycznych terminów. Wynika stąd możliwość stosowania specyficznej właściwości oprogramowania nazwanej *wrażliwością*, jako parametru oceny jakości. Definicja wrażliwości oprogramowania, w powyższym sensie, może brzmieć następująco:

wrażliwość oprogramowania – wielkość zmian w zachowaniu oprogramowania w odpowiedzi na zmiany wartości terminów wykonania zadania.

Interpretacja tej definicji, jak wynika z rysunku 1, polega na tym, że im większe są nachylenia odpowiednich krzywych, a więc im szybsze są zmiany mierzonej wielkości, tym bardziej wrażliwy jest system. Oprogramowanie jest bardziej wrażliwe, gdy małe zmiany w długości terminów powodują względnie duże zmiany w wielkości ocenianego parametru. W praktyce oznacza to, czy ewentualna degradacja wydajności systemu będzie następować szybko czy wolno.

Z inżynierskiego punktu widzenia, zrozumienie i interpretacja pojęcia wrażliwości oprogramowania jako specyficznej właściwości systemu czasu rzeczywistego są oczywiste. Trochę mniej oczywiste jest zinterpretowanie metryki (jednostki pomiaru) i miary wrażliwości oprogramowania.

Z rysunku 1 wynikałoby, że wrażliwość mierzoną jako nachylenie krzywej (im bardziej stroma krzywa, tym większa wrażliwość) należy interpretować w liczbie przekroczonych terminów na sekundę, czyli w Hz. Interpretacja taka jednak nie umożliwia porównania różnych systemów. W związku z tym, w [3] zaproponowano względne obliczanie wrażliwości S, przy lokalnej liniowości, według wyrażenia

$$S = \frac{(y_1 - y_0)/[(y_1 + y_0)/2]}{(x_1 - x_0)/[(x_1 + x_0)/2]}$$

gdzie y_1 i y_0 oznaczają, odpowiednio, większą i mniejszą wartość punktów na osi rzędnych, a x_1 i x_0 – wartości odpowiednich punktów na osi odciętych. Przy takiej interpretacji, powyższe wyrażenie jest miarą wrażliwości, a jej jednostką, czyli metryką, jest liczba rzeczywista bezwymiarowa z przedziału $(0, +\infty)$.

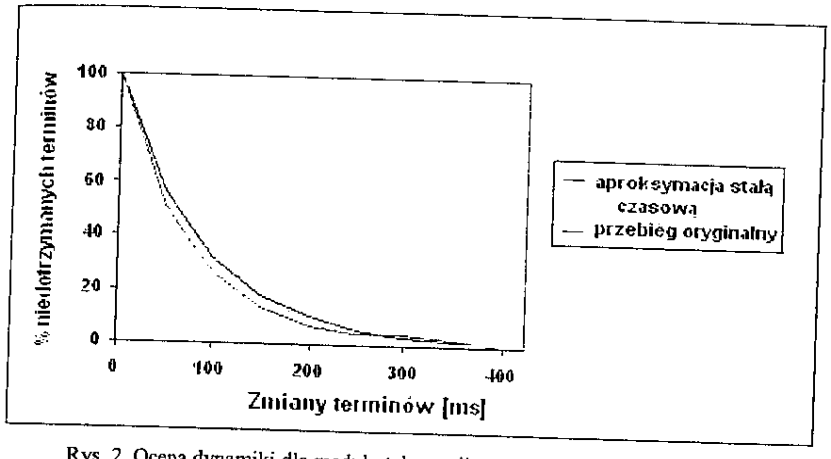
2.3. DYNAMIKA OPROGRAMOWANIA

Idąc dalej drogą interpretowania wykresów z rysunku 1, można zauważyć, że dynamika zmian niektórych z nich przypomina odpowiedź liniowego systemu dynamicznego na pobudzenie skokowe (wykres dla systemu w języku Java).

Taka interpretacja prowadzi do określenia nowego parametru oprogramowania systemów czasu rzeczywistego, mianowicie – dynamiki oprogramowania. Zakładając, że zmiana jest eksponencjalna, można uznać że badany system jest liniowym systemem dynamicznym pierwszego rzędu i scharakteryzować go następującą funkcją transferową, G:

$$G(s) = y(s)/x(s) = K/(\tau * s + 1)$$

gdzie symbol s oznacza zmienną Laplace'a, K jest wzmocnieniem systemu, a τ – stałą czasową. Właśnie stała czasowa, τ, jest odpowiednikiem miary do określania właściwości dynamicznych oprogramowania. Metryką, w sensie wyjaśnionym w podrozdziale 1, jest sekunda (jako jednostka pomiaru), ale stała czasowa jest miarą dynamiki, ponieważ jest zdefiniowana w automatyce jako miara szybkości odpowiedzi systemu na sygnał wejściowy, czyli miara dojścia systemu do stanu ustalonego. Typowo określa się jej wartość eksperymentalnie jako 0,25 czasu ustalania, przy czym za czas ustalania można uważać czas, po którym krzywa odpowiedzi osiąga 2%–4% wartości w stanie ustalonym.



Rys. 2. Ocena dynamiki dla modułu telemetrii w modelu stacji satelitarnej [14]

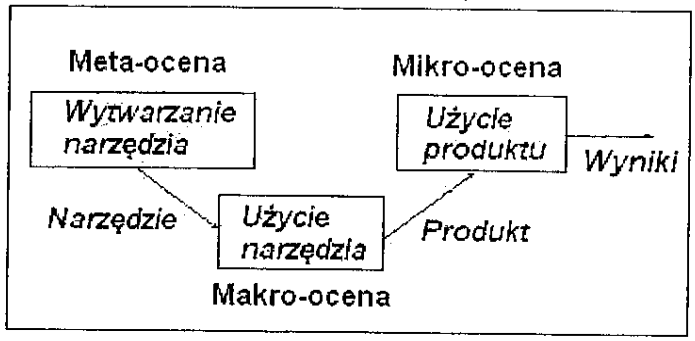
Badania eksperymentalne dynamiki oprogramowania dla modelu naziemnej stacji satelitarnej opublikowane w [14] potwierdziły sensowność takiego podejścia. Badane trzy moduły wykazały różną dynamikę: od 15 ms dla GPS, do 87 ms dla telemetrii i 165 ms dla bazy danych. Wyniki dla modułu telemetrii przedstawiono na rysunku 2 [14]. Dokładniejsze omówienie warunków otrzymania tych wyników oraz ich analiza znajduje się w oryginalnym artykule.

3. POMIARY JAKOŚCI NARZĘDZI DO WYTWARZANIA SYSTEMÓW KRYTYCZNYCH DLA BEZPIECZEŃSTWA

W niniejszej części omówiono metodykę i wyniki eksperymentów oceny narzędzi programistycznych do wytwarzania oprogramowania i sprzętu dla systemów krytycznych dla bezpieczeństwa.

3.1. METODYKA POMIARÓW

W systemach czasu rzeczywistego do zastosowań krytycznych dla bezpieczeństwa, istotną rolę odgrywają narzędzia programistyczne, jako że systemy te są obecnie tak złożone, że ich wytwarzanie odbywa się w dużej części automatycznie [7]. W ocenie narzędzi istnieje, jednak, pewien podstawowy problem, zilustrowany na rysunku 3. Do pełnej oceny należałoby podejść trójstopniowo: ocenić sam proces wytwarzania narzędzia, jak również użycie narzędzia i działanie produktu wytworzonego przez to narzędzie. Ze względu na fakt, że procesy wytwarzania narzędzi programistycznych są objęte ścisłą tajemnicą handlową i nie są ujawniane przez producentów, a działanie produktów wytwarzanych przez te narzędzia jest przedmiotem odzielnych prac, autorzy skupili się na samej ocenie użycia takich narzędzi, zwanej tutaj makro–oceną.



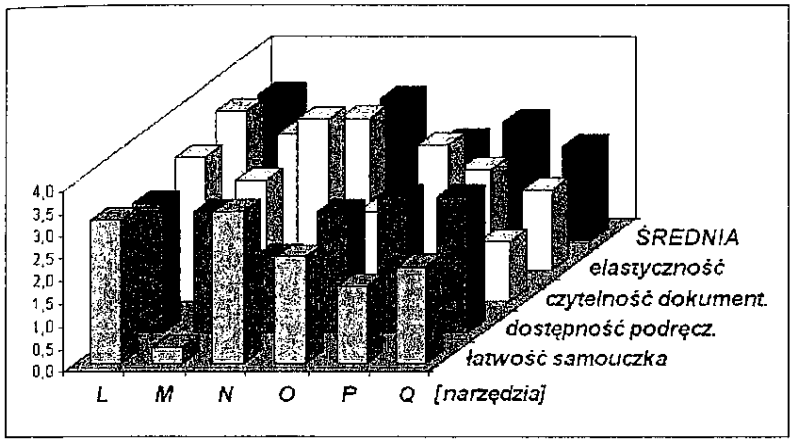
Rys. 3. Model procesu oceny narzędzi programistycznych [6]

Ponieważ ocena jakości narzędzi programistycznych wiąże się z rozważeniem wielu czynników mających wpływ na jakość, wyróżniono grupę kryteriów, które wspólnie reprezentują ocenę jakości. Do kryteriów tych zaliczono: funkcjonalność, wydajność, użyteczność i śladowalność (ang. *traceability*). Aczkolwiek istnieje wiele innych kryteriów, które mogłyby podlegać ocenie i są zalecane w różnych normach, jak np. przenośność, niezawodność, odporność, zgodność wewnętrzna (ang. *consistency*) itd., wybór dokonany przez autorów był podyktowany nie tylko istotnością kryterium, ale także praktycznością pomiaru.

3.2. EKSPERYMENT OCENY NARZĘDZI PROGRAMISTYCZNYCH

Według metodyki opisanej w podrozdziale *Wprowadzenie*, dla każdego z czterech wybranych kryteriów jakości wyliczonych powyżej, dobrano odpowiednią metrykę i miarę. Na przykład, *użyteczność* mierzono jako wysiłek wymagany do wykonania eksperymentu, wyrażony w czasie poświęconym na każdą fazę projektu. *Wydajność*

mierzono jako – wyrażony w liczbie linii programu – rozmiar kodu wygenerowanego przez narzędzie dla modelu zrealizowanego w typowym projekcie. *Funkcjonalność* mierzono subiektywną oceną użytkownika wyrażoną na skali liczbowej od 0 do 5, dla czterech wyróżnionych cech narzędzia: dostępność podręczników, łatwość samouczka (ang. *tutorial*), czytelność dokumentacji i elastyczność. Określenie *śladowalności* okazało się najtrudniejsze do oceny numerycznej i udało się jedynie jakościowo przez ręczne śladowanie kodu wstecz do wymagań.



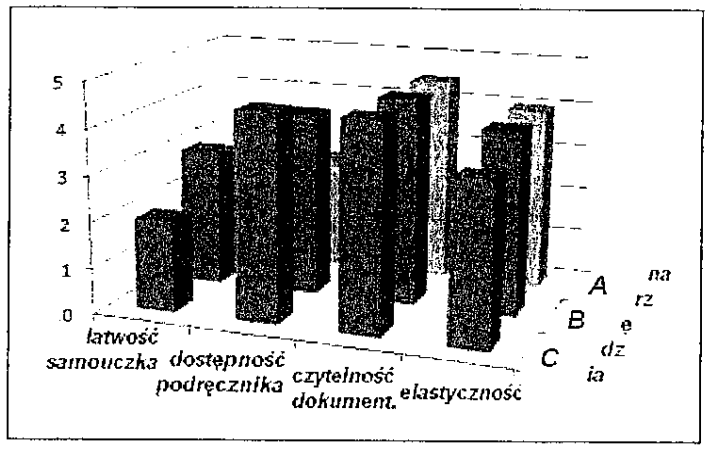
Rys. 4. Funkcjonalność narzędzi mierzona ocenami na skali od 0 do 5 [6]

Na rysunku 4 przedstawiono przykładowe oceny funkcjonalności sześciu narzędzi programistycznych dla eksperymentu polegającego na użyciu tych narzędzi w przykładowym projekcie wytwarzania oprogramowania awioniki działającego pod systemem operacyjnym VxWorks we współpracy z symulatorem systemu sterowania samolotem dostarczającym sygnały z czujników. Zespół szesnastu projektantów miał do dyspozycji sześć narzędzi podzielonych na dwie kategorie: narzędzia do projektowania modeli dyskretnych, jak np. Rhapsody i RoseRT, oraz narzędzia do projektowania modeli ciągłych, jak Matlab lub SCADE. Dokładniejszy opis eksperymentu i jego analizę przedstawiono w [6].

3.3. NARZĘDZIA PROGRAMISTYCZNE DO WYTWARZANIA SPRZĘTU

Oddzielny projekt wykonany przez autorów dotyczył oceny narzędzi programistycznych do wytwarzania sprzętu dla systemów krytycznych dla bezpieczeństwa [8]. Zasada oceny była taka sama, jak przy ocenie narzędzi do wytwarzania oprogramowania, głównie dlatego, że we współczesnych zastosowaniach sprzęt jest oparty na systemach FPGA i ASIC, których projektowananie prawie nie różni się od projektowania opro-

gramowania, gdyż polega na tworzeniu programów w postaci graficznej lub w języku VHDL i generowaniu – za pomocą narzędzia – kodu binarnego, który jest następnie ładowany do układu elektronicznego.



Rys. 5. Funkcjonalność narzędzi do wytwarzania sprzętu mierzona ocenami na skali od 0 do 5 [2]

W eksperymencie oceny użyto trzech narzędzi najbardziej popularnych producentów na rynku: Quartus II firmy Altera, ISE firmy Xilinx i LabVIEW z wkładką FPGA firmy National Instruments. Na prostym przykładzie projektowania licznika binarnego użyto metody sprawdzonej w poprzednim przykładzie, stosując trzy rodzaje kryteriów oceny: użyteczność, funkcjonalność i wydajność (mierzoną w bajtach wygenerowanego kodu). Przykładowy wykres porównawczy funkcjonalności narzędzi przedstawiono na rys. 5. Jak wynika z tego wykresu, narzędzia nie różnią się od siebie w istotny sposób pod względem użytego kryterium. Wydaje się to być skutkiem podobnego stanu zaawansowania technologii narzędzi, powszechnie znanej producentom. Dokładniejsza analiza znajduje się w przygotowywanym raporcie [2].

4. ZAKOŃCZENIE

Ocena i pomiary jakości oprogramowania systemów czasu rzeczywistego i systemów krytycznych dla bezpieczeństwa są niezwykle skomplikowane w realizacji, choćby ze względu na fakt, że trudno uznać oprogramowanie za obiekt fizyczny, więc można oceniać jego zachowanie tylko na podstawie pomiarów pośrednich. Podejście zaprezentowane przez autorów polega na wyrażeniu właściwości oprogramowania podlegającej pomiarowi, i odpowiednim określeniu metryki dla tej właściwości, jak też jej miary. Koncepcja ta pozwala na ilościową ocenę wrażliwości i dynamiki oprogramo-

wania, jak również na względnie obiektywny pomiar jakości przy użyciu subiektywnych kryteriów porównawczych.

Konieczne są dalsze prace nad weryfikacją i przydatnością zaprezentowanych modeli pomiarowych. Naturalnie, wiele innych, istotnych aspektów procesu pomiarowego, jak błędy, dokładność, niepewność itp., wymaga dokładniejszych studiów, ale nie zostało poruszonych w tym rozdziale ze względu na brak miejsca.

LITERATURA DO ROZDZIAŁU

[1] Buth B., G. Rabe, T. Seyfarth (Eds.): Proc. SAFECOMP2009, 28th International Conference on Computer Safety, Reliability, and Security. Hamburg, Germany, September 14–18, 2009, LNCS 5775, Springer–Verlag, Berlin/Heidelberg 2009.

[2] Butka B., Kornecki A., Zalewski J.: Qualification of Tools for Airborne Electronic Hardware. Report DOT/FAA/AR–10/xx. Federal Aviation Administration, Washington DC, 2010 (w przygotowaniu).

[3] Guo D., van Katwijk J., Zalewski J.: A New Benchmark for Distributed Real–Time Systems: Some Experimental Results. Proc. WRTP'2003, 27th IFAC/IFIP/IEEE Workshop on Real–Time Programming, Łagów, Poland, May 14–17, 2003, pp. 141–146.

[4] von Helmholtz H.: Zählen und Messen, erkenntnisstheoretisch betrachtet. In: Philosophische Aufsätze. Eduard Zeller zu seinem fünfzigjährigen Doktorjubiläum gewidmet, Fues' Verlag, s. 17–53, Leipzig 1887.

[5] Kornecki A., Zalewski J.: Real–Time Safety Critical Systems: Fundamental Concepts, Design Principles and Software Development. Tutorial at the IFAC Summer School on Control, Computing and Communication. Prague, Czech Republic, June 27 – July 1, 2005. URL: http://dce.felk.cvut.cz/hanzalek/ifacss05/slides.php

[6] Kornecki A., Zalewski J.: Experimental Evaluation of Software Development Tools for Safety–Critical Real–Time Systems. Innovations in Systems and Software Engineering: A NASA Journal, 1(2), pp. 176–188, September 2005.

[7] Kornecki A., Zalewski J.: Certification of Software for Real–Time Safety–Critical Systems: State of the Art. Innovations in Systems and Software Engineering: A NASA Journal, 5(2), pp. 149–161, June 2009.

[8] Kornecki A., Zalewski J.: Hardware Certification for Real–Time Safety–Critical Systems: State of the Art. Annual Reviews in Control, 34(1), pp. 163–174, April 2010.

[9] Li M., Smidts C.S.: A Ranking of Software Engineering Measures Based on Expert Opinion. IEEE Trans. on Software Engineering, 29(9), pp. 811–824, September 2003.

[10] Międzynarodowy Urząd Miar i Wag, Sèvres, Francja, URL: http://www.bipm.fr/

[11] Redmill F. (Ed.): Dependability of Critical Computer Systems. Vols. 1 and 2. Elsevier, London 1989.

[12] Sanz R., Zalewski J.: Control Systems Engineering Using Design Patterns. IEEE Control Systems, 23(3), pp. 43–60, June 2003.

[13] Schneidewind N.: A Quantitative Approach to Software Development Using IEEE 982.1. IEEE

Software, 24(1), pp. 65–72, January/February 2007.

[14] Zalewski J.: From Software Sensitivity to Software Dynamics: Performance Metrics for Real-Time Software Architectures. SIGBED Review, 2(3), pp. 20–24, July 2005.

[15] Zalewski J., W. Ehrenberger, F. Saglietti, J. Górski, A. Kornecki: Safety of Computer Control Systems: Challenges and Results in Software Development. Annual Reviews in Control, 27(1), pp. 23–37, 2003.

[16] Zuse H.: A Framework of Software Measurement. Walter de Gruyter, Berlin 1998.