Pomiary Automatyka Robotyka, ISSN 1427-9126, R. 24, Nr 4/2020, 85–91, DOI: 10.14313/PAR_238/85

Nowoczesne technologie projektowania systemów automatyki

Mariusz Pauluk
AGH Akademia Górniczo-Hutnicza, Katedra Automatyki i Robotyki, al. Mickiewicza 30, 30-059 Kraków

Streszczenie: W pracy omówiono stosowane obecnie techniki wytwarzania systemów automatycznej regulacji. Rozpoczęto od przybliżenia stosowanych w inżynierii oprogramowania pojęć: kryzys oprogramowania, katastrofa oprogramowania oraz cykl życia oprogramowania. To ostatnie zostało w kolejnych rozdziałach poszerzone o najbardziej popularne modele wytwarzania oprogramowania. Następnie przybliżono na bazie modelu V rolę weryfikacji i walidacji w cyklu życia oprogramowania i sterownika oraz przedstawiono techniki testowe stosowane w walidacji sterownika. Są to testy typu: oprogramowanie w pętli, procesor w pętli oraz sterownik w pętli. Ostatni rozdział opisuje technikę projektowania systemów automatyki w oparciu o zaawansowane modele matematyczne Model Based Design.

Słowa kluczowe: systemy wbudowane, jakość oprogramowania, kryzys oprogramowania, cykl życia oprogramowania, Model V, weryfikacja i walidacja, techniki testowania MIL, SIL, PIL, HIL

1. Wstęp

Ostatnia dekada, to kolejny okres intensywnego rozwoju mikroprocesorowych układów elektronicznych oraz narzędzi informatycznych. Doprowadziło to do zastosowania w praktyce znacznie bardziej zaawansowanych matematycznie algorytmów oraz do wytworzenia oprogramowania składającego się ogromnej liczby rozkazów (nawet rzędu dziesiątków milionów) i posiadającego niespotykaną wcześniej funkcjonalność. Ten rozwój wymusił także opracowanie nowych metodyk, koncentrujących się na jakości wytwarzanego oprogramowania. Praca ma charakter interdyscyplinarny. Łączy w sobie elementy z obszarów informatyki, automatyki i robotyki, a także elektroniki. Celem tego opracowania jest przybliżyć czytelnikom stosowane obecnie koncepcje realizacji projektów z punktu widzenia osób zajmujących się projektowaniem i budową autonomicznych sterowników. W informatyce, tego rodzaju rozwiązania klasyfikowane są jako systemy wbudowane (ang. *Embedded Systems*). Ogólnie pod tym pojęciem rozumie się urządzenie mikroprocesorowe wraz ze zintegrowanym oprogramowaniem, zaprojektowane do realizacji konkretnej funkcjonalności, na ogół wymagającej intensywnej interakcji z otoczeniem. Bardziej ogólną definicją posługuje się I. Sommerville [1, str. 21]:

Autor korespondujący:
Mariusz Pauluk, mp@agh.edu.pl

Artykuł recenzowany
nadesłany 06.08.2020 r., przyjęty do druku 11.12.2020 r.

System wbudowany. Jest to system, w którym oprogramowanie steruje pewnym urządzeniem i jest umieszczone w tym urządzeniu.

W pracy zostały przybliżone technologie i pojęcia opracowane i stosowane przez informatyków na potrzeby wytwarzania oprogramowania. Zaprezentowano także rozwiązania bazujące na powyższych pomysłach, a zaadaptowane przez inżynierów do konstrukcji systemów wbudowanych.

W pierwszym rozdziale przybliżono pojęcie *kryzysu oprogramowania* [1, 2] i podano parę przykładów, w których miały miejsce katastrofy oprogramowania, obrazujących skalę problemów, z jaką obecnie mierzą się twórcy oprogramowania.

Jednym ze sposobów przeciwdziałania kryzysowi oprogramowania jest doprecyzowanie i utworzenie zbioru dobrych praktyk, a następnie zalecenie ich stosowania przez wprowadzenie uregulowań lub opublikowaniu ich w formie unormowań. Rozdział drugi prezentuje najbardziej popularne unormowania pomocne przy projektowaniu i budowie systemów wbudowanych.

Innym sposobem przeciwdziałania temu zjawisku jest projektowanie i wytwarzanie oprogramowania w sposób zaplanowany i systematyczny. Popularne koncepcje modeli wytwarzania oprogramowania przybliżono pokrótce w rozdziale trzecim.

Rozdział czwarty omawia *Model V* i powiązane z nim procesy weryfikacji i walidacji. Jest to jeden z bardziej znanych modeli wytwarzania oprogramowania.Wyszukiwarka Google zwraca ok. 6,5 mld wyników dla tego pojęcia.

W rozdziale piątym zaprezentowano wybrane techniki testowania w tzw. pętli. Są to podejścia typu: model w pętli, procesor w pętli oraz sterownik w pętli. W literaturze anglojęzycznej odpowiednio spotyka się określenia: MIL – Model In the Loop, PIL – Procesor In the Loop oraz HIL – Hardware in the Loop.

W rozdziale szóstym przybliżono proces wytwarzania systemów wbudowanych z wykorzystaniem modeli matematycznych do symulacji obiektu sterowanego i regulatora. Układ regulacji rozpoczyna się projektować w środowisku projektowo-symu-

lacyjnym wyższego rzędu, w sposób graficzny. W literaturze anglojęzycznej dla takiego podejścia używa się określenia: MBD – Model Based Design. Całość zakończono podsumowaniem.

2. Kryzys oprogramowania

Pojęcie to sformułowano po raz pierwszy w 1968 r. na konferencji NATO na temat inżynierii oprogramowania [3]. Już wówczas F. David i A. Fraser zwracali uwagę,

„...na pojawiający się rozdźwięk: między obietnicami składanymi użytkownikowi, a jakością finalnie dostarczanego oprogramowania między tym, co wydaje się być możliwe do osiągnięcia a tym, co jest w praktyce osiągalne, między przewidywanymi kosztami wytworzenia oprogramowania, a rzeczywistymi wydatkami..."

Ponadto zauważali, że

„...ten rozdźwięk wraz z upływem czasu powiększa się, a konsekwencje awarii oprogramowania we wszystkich jego aspektach są coraz poważniejsze. Szczególnie alarmująca jest wyraźnie widoczna i nie do uniknięcia zawodność dużej skali oprogramowania, ponieważ awaria zaawansowanego systemu sprzętowo-programowego może być sprawą życia i śmierci..."

Ciekawe spostrzeżenia kilka lat później zamieścił w swojej pracy E.W. Dijkstra [4] pisząc o narastającym rozczarowaniu, czy wręcz frustracji związanej z gwałtownie rozwijanymi możliwościami sprzętu i nie nadążają za nimi funkcjonalnością oprogramowania. On również użył pojęcia kryzys oprogramowania.Współcześnie pojęcie to poszerza się ponadto o ogromne koszty utrzymania oprogramowania, kosztowny i długotrwały cykl wytworzenia oprogramowania, a także o brak odpowiednich narzędzi i języków do pełnego wykorzystania możliwości oferowanych przez sprzęt. Jako kryzys oprogramowania traktuje się także powszechne zjawisko wielokrotnego przesuwania terminów oddania oprogramowania w związku z wykrywaniem nowych błędów. Warto zwrócić uwagę, że pojęcie to sformułowano już pięćdziesiąt lat temu.

Obecnie w użytkowaniu na masową skalę są m.in.: systemy wspomagające kierowanie pojazdami, autopiloty, systemy bezpieczeństwa, systemy podtrzymujące funkcje życiowe i nadzorujące stan zdrowia, urządzenia diagnostyczne, a także wszelkiego rodzaju aplikacje uruchamiane w urządzeniach przenośnych.

Przewidywane ponad pół wieku temu „nieuniknione awarie dużego oprogramowania" niestety miały miejsce. Prawdopodobnie jednym z bardziej znanych i spektakularnych przykładów katastrofalnych konsekwencji błędu oprogramowania jest nieudany start rakiety ARIANE 5, zakończony samozniszczeniem w 40 sekundzie lotu na wysokości ok. 3700 m [5] . Ten wypadek zainicjował zintensyfikowanie wysiłków nad opracowaniem metod wytwarzania bezpiecznego w użytkowaniu oprogramowania. Mniej poważne w skutkach tego typu zdarzenia obserwujemy na co dzień, jak zawieszający się smartfon lub laptop. Jednakże łatwo sobie wyobrazić bardziej niebezpieczne sytuacje, jak np. zawieszenie się komputera odpowiedzialnego za wspomaganie układu kierowniczego w samochodzie, czy też systemów pokładowych w samolocie. O realnej możliwości wystąpienia takiego zdarzenia świadczy, nie tak odległy w czasie, wypadek samochodu Tesla ze skutkiem śmiertelnym, mający miejsce w maju 2016 r. [6]. Przyczyną wypadku był błąd funkcjonalny w oprogramowaniu autopilota samochodu. Skręcająca przed pojazdem ciężarówka z naczepą miała duży prześwit i nie została prawidłowo rozpoznana przez algorytm analizujący obraz z kamer. Samochód w pełnym pędzie wjechał w ciężarówkę.

3. Unormowania

Istnieje wiele unormowań dotyczących zapewnienia bezpieczeństwa przy projektowaniu systemów sprzętowo-programowych wobszarze automatyki i robotyki. Przestrzeganie tych norm często jest konieczne, aby otrzymać wymagane w niektórych branżach i zastosowaniach certyfikaty dopuszczające produkt do użytku. Przytaczane poniżej normy zakreślają wymagany poziom bezpieczeństwa dla oprogramowania oraz współpracującego z nim sprzętu i stanowią punkt odniesienia dla jakości oferowanych przez przemysł produktów.

Norma IEC 61508, do której bardzo często spotyka się w literaturze odwołania, np. [5, 7–15], precyzuje m.in. reguły postępowania przy konstrukcji i upowszechnianiu urządzeń elektrycznych, elektronicznych oraz programowalnych elektrycznie, chociaż została zredagowana jako ogólna norma, bez precyzowania branż, w których zaleca się ją stosować.

Odpowiednikiem tej normy dla układów projektowanych na potrzeby przemysłu samochodowego jest norma: ISO 26262 [16].

Norma ISO 13482 [17] określa standardy bezpieczeństwa dla układów robotycznych, wytwarzanych w celu sprawowania bezpośredniej opieki nad człowiekiem. Z kolei norma ISO 10218 [7] została poprawiona w kontekście robotów przemysłowych i ich współpracy z człowiekiem, np. na liniach montażowych i podczas prac spawalniczych. Dalsze prace nad tą normą mają na celu doprecyzowanie innych czujników bezpieczeństwa, aby móc poszerzyć w sposób bezpieczny obszar współpracy człowieka i maszyny. W szczególności norma IEC 61496 [18] określa standardy bezpieczeństwa dla stosowania czujników obrazu typu 3D w celu wykrycia ludzkiego ciała, ewentualnie przeszkody.

Jednocześnie powyższe normy zostawiają bardzo dużo swobody w doborze narzędzi, którymi można się posługiwać przy wytwarzaniu bezpiecznych systemów. Takie podejście czyni te unormowania elastycznymi i zapewnia ich niezmienność w dłuższej perspektywie czasu.Pozostawia to również twórcom narzędzi dużą niezależność w dobie szybko zmieniających się technologii i rozwiązań informatycznych.

W dalszej części pojęcie system (często pojawiające się w normach) będzie stosowane zamiennie z pojęciami: układ regulacji, sterownik, system wbudowany lub układ: regulator-obiekt.

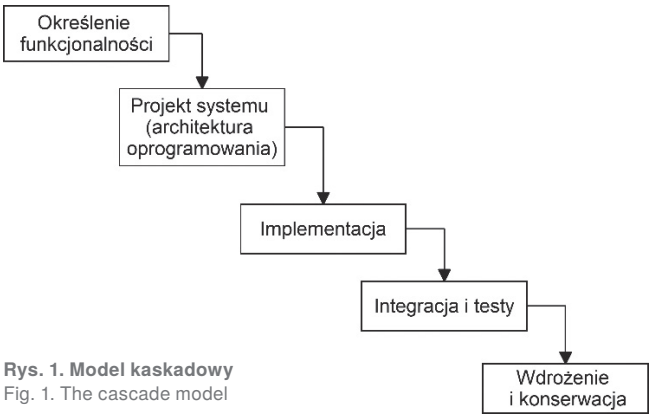
4. Cykl życia oprogramowania

Z pojęciem tym wiążą się wszystkie etapy, jakie podejmuje się w celu wytworzenia oprogramowania oraz utrzymania go funkcjonalnym przez okres użytkowania. W literaturze cykl życia oprogramowania na ogół pojawia się zamiennie z modelem procesu wytwarzania oprogramowania.

Istnieje wiele modeli procesu wytwarzania oprogramowania. Uznaje się, że praktycznie w każdym takim przedsięwzięciu powinny znaleźć się następujące elementy:
- określenie funkcjonalności oprogramowania,
- wytworzenie oprogramowania,
- testy,
- wdrożenie (konserwacja) oprogramowania.

Jednym z najczęściej przytaczanych w literaturze modelem cyklu życia oprogramowania jest model kaskadowy (ang. *waterfall model*), którego początki sięgają lat siedemdziesiątych ubiegłego wieku (rys. 1). W modelu tym proces tworzenia oprogramowania podzielony jest na etapy, które twórcy kolejno przechodzą.

Model kaskadowy nie jest jedynym, jaki rozważa się przy opisywaniu oprogramowaniu lub stosuje się w trakcie jego wytwarzania. Obecnie jest on już dość anachronicznym podejściem,



Rys. 1. Model kaskadowy
Fig. 1. The cascade model

głównie ze względu na hermetyczność poszczególnych etapów i brak elastyczności w przechodzeniu między wyszczególnionymi na rysunku etapami. Tym niemniej, bardzo trafnie ujmuje specyfikę procesu powstawania oprogramowania, a także jego użytkowania. Wiele zależy od charakteru opracowywanego systemu. Nie istnieje uniwersalne podejście. Prezentowane w następnych częściach artykułu rozwiązania są rozwinięciem tego podejścia. Poniżej pokrótce wspomniano o innych wybranych i spotykanych w literaturze modelach. Są to:

- zmodyfikowany model kaskadowy – pozwalający na większą swobodę w przechodzeniu między kolejnymi etapami, przewidujący także możliwość powrotu do wcześniejszych etapów,
- model spiralny – iteracyjnie przechodzi się wielokrotnie wszystkie etapy przyjęte w modelu (niekoniecznie takie same jak w modelu kaskadowym), dobrane adekwatnie do rozwiązywanego problemu,
- model ewolucyjny – dopuszcza rozwój specyfikacji w trakcie projektowania i tworzenia oprogramowania
- podejście oparte na modelowaniu formalnym – inaczej, operujące na formalnym opisie matematycznym od samego początku, tj. od formułowania specyfikacji. Przejścia do kolejnych etapów w tym podejściu odbywają się na podstawie przekształcania równań. Właściwości modelu dowodzi się również matematycznie. W tym modelu nie ma potrzeby testowania, zgodność ze specyfikacją można wykazać w formie dowodu matematycznego, podobnie jak brak błędów w oprogramowaniu,
- projektowanie z wykorzystaniem modelu MBD (ang. *Model Based Design*) – omówiono nieco szerzej w poniższym akapicie i ostatnim rozdziale, zaliczane jest do wariantu metod formalnych.

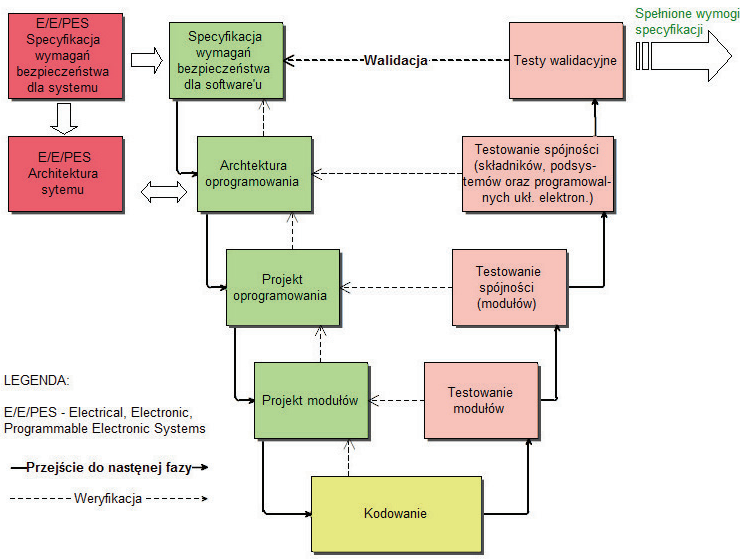
Metody formalne wydają się być najtrudniejsze do zastosowania w praktyce, z racji m.in. trudności w precyzyjnym opisie zjawisk fizycznych za pomocą równań, a także z konieczności odwzorowania ich następnie w języku programowania. W ostatnich latach nastąpił znaczny rozwój narzędzi pozwalających modelować zjawiska fizyczne [19], szczególnie w środowisku graficznym. Nastąpiła istotna poprawa jakości narzędzi rozwiązujących równania matematyczne metodami numerycznymi [20–22], jak również symbolicznymi [23, 24]. Duży postęp odbywa się także w automatycznej generacji kodu [25, 26] na podstawie modeli utworzonych w środowiskach graficznych wyższego rzędu.

5. Model V

Pojęcie Model V (ang. *V-Model*) nie ma ścisłej definicji. Jest jednym spośród wielu znanych w literaturze określeń cyklu życia oprogramowania. Model V najczęściej kojarzony jest z usystematyzowanym zbiorem procedur, ścieżką postępowania, która ma zapewnić otrzymanie wysoce niezawodnego produktu finalnego. W postępowaniu według Modelu V szczególny nacisk kładziony jest na prowadzenie testów w usystematyzowany sposób.

Istnieją formalne dokumenty określające metodykę zgodną z Modelem V, jednakże przyjęte w nich definicje i ich rozumienie są w dość szerokim zakresie płynne. Koncepcja ta narodziła się pośród twórców oprogramowania w Wielkiej Brytanii. Swoje własne odpowiedniki w tym zakresie rozwinęły Stany Zjednoczone oraz Niemcy.

Pojęcie Model V często spotyka się tam, gdzie istotna jest duża niezawodność produktu. W szczególności, formalnym dokumentem regulującym to pojęcie jest wspomniana już norma IEC 61508 (rys. 2) traktująca o bezpieczeństwie funkcjonalnym projektowanych urządzeń elektrycznych, elektronicznych oraz ogólnie pojętych jako programowalnych E/E/PE (ang. *Electrical/Electronic/Programmable Electronic Systems*). Nie wszyscy jednakże pod pojęciem Model V rozumieją ten właśnie sposób postępowania.

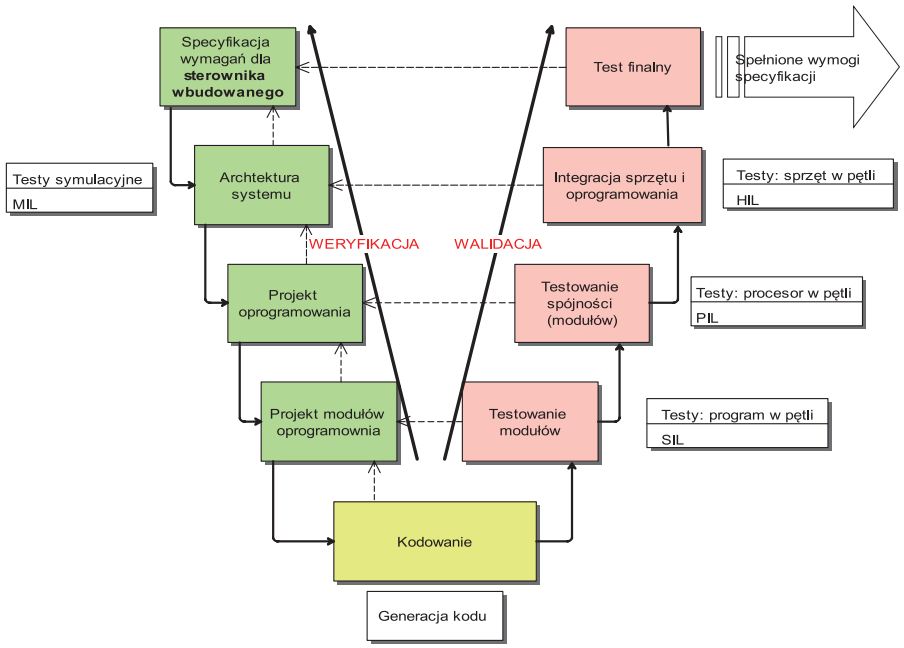


Rys. 2. Model-V według normy IEC 61508
Fig. 2. V-model according to IEC 61508 standard

Intensywny rozwój techniki mikroprocesorowej i narzędzi programistycznych doprowadził do pojawienia się w skali masowej sterowników wbudowanych. Koncepcja tworzenia oprogramowania w oparciu o Model V stała się atrakcyjna także dla dostawców kompleksowych rozwiązań mechatronicznych. Na rysunku 3 zaprezentowano koncepcję Modelu V zaadaptowaną na potrzeby rozwoju systemów wbudowanych.

Podejście to jest popularne również wśród producentów rozwiązań dla przemysłu lotniczego, samochodowego oraz obronnego. Przyjęcie tej metodyki w złożonych projektach znacząco skraca czas pomiędzy chwilą pojawienia się pomysłu (rozpoczęcia określania specyfikacji), a momentem przedłożenia gotowego (tzn. „wolnego od błędów" i zgodnego ze specyfikacją) do masowej produkcji wytworu. Koncepcja Model V ma także szerokie grono krytyków, podnoszących uciążliwość konieczności prowadzenia w sposób rygorystyczny dokumentacji. W przypadku mniej złożonych projektów jest to istotnym utrudnieniem i czynnikiem znacząco spowalniającym tempo prac.

Z pojęciem Model V wymiennie stosuje się inne sformułowanie: weryfikacja i walidacja V&V (ang. *Verification and Validation*) [12]. Weryfikacja to badanie zgodności opracowywanego



Rys. 3. Ogólna koncepcja realizacji systemów wbudowanych bazująca na metodyce Model V
Fig. 3. General concept of the implementation of embedded systems based on the Model V methodology

projektu ze specyfikacją. Podczas weryfikacji bada się także spójność specyfikacji i ewentualnie występujące w niej błędy.

Walidacja to działania, procedury i metodyki mające na celu wykazać poprawność zrealizowania projektu i brak błędów związanych bezpośrednio z programowaniem i projektem układów mechatronicznych.

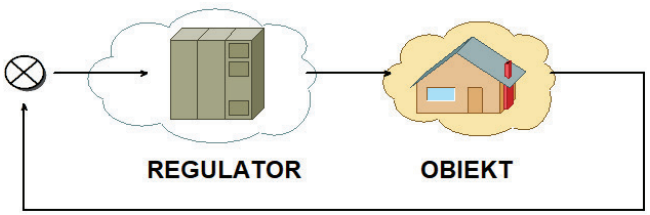
Proces walidacji i weryfikacji realizuje się w sposób kołowy, np. po wykryciu błędu podczas walidacji z reguły powraca się na ten sam poziom do weryfikacji i ponownie przechodzi się przez kolejne etapy w modelu. Każdy wykryty błąd wymusza cofnięcie się o kilka kroków. Im na dalszym etapie błąd zostanie odkryty, tym większą ilość kroków należy się cofnąć w testowaniu.

6. Wybrane techniki procesu weryfikacji i walidacji

Stosuje się kilka sposobów weryfikowania i testowania opracowanych rozwiązań. Są one tak podzielone, aby stopniowo wykrywały niezgodności ze specyfikacją oraz w kolejnych etapach błędy w implementacji. Etapy te przechodzi się w sposób iteracyjny. Poruszanie się w iteracyjny sposób po Modelu V minimalizuje prawdopodobieństwo odkrycia znaczącego błędu w końcowych etapach. W literaturze, opisane powyżej postępowanie można spotkać pod określeniem testowanie w pętli MIL/SIL/PIL/HIL (rys. 3). Dalej przedstawiono wybrane techniki testowania.

6.1. Model w pętli

Ten rodzaj testowania stosuje się na etapie opracowywania architektury systemu. Matematyczne modele regulatora i obiektu sterowanego opracowuje się w środowisku wyższego rzędu, np. SCADE Suit, MATLAB/Simulink, LabVIEW [20–22].Weryfikacja odbywa się symulacyjnie, a rezultaty są na tyle wiarygodne, na ile dokładny jest zastosowany model (rys. 4).

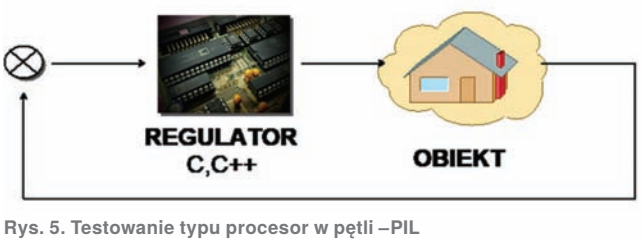


Rys. 4. Model systemu w pętli testującej – MIL
Fig. 4. Model-in-the-loop testing – MIL

Technikę model w pętli stosuje się także w celu uzyskania tzw. dowodu poprawności koncepcji. Na tym poziomie projektowania nie uwzględnia się jeszcze szczegółowych rozwiązań, jak: typ procesora, wielkość pamięci czy rodzaj portu szeregowego.

6.2. Procesor w pętli

Testowanie typu procesor w pętli wymaga zaimplementowania algorytmu regulatora w tym samym rodzaju procesora, który ma być zastosowany w finalnym produkcie. Procesor nie musi być umiejscowiony w otoczeniu planowanym jako docelowe. Może to być np. płytka ewaluacyjna. Obiekt sterowany w dalszym ciągu jest symulowany w środowisku wyższego poziomu (rys. 5).



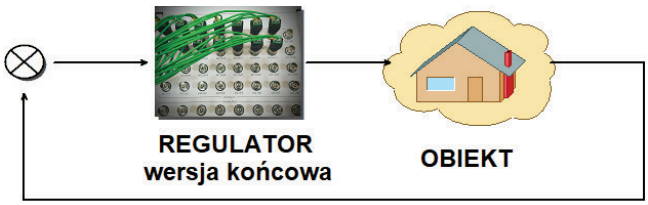
Rys. 5. Testowanie typu procesor w pętli –PIL
Fig. 5. Processor-in-the-loop testing – PIL

Ponieważ w układzie testowania typu procesor w pętli kod uruchamia się w tym samym procesorze, który planowany jest w rozwiązaniu końcowym, procedury testowe można więc poszerzyć o:
- testowanie zależności czasowych,
- testowanie przerwań,
- testowanie prawidłowości wykorzystania zasobów procesora,
- testowanie wydajności mocy obliczeniowej procesora,
- testowanie skali procesora, czyli szerokości magistral danych, układu ALU, sprzętowego układu mnożącego itp.

Testowanie procesor w pętli przeprowadza się na etapie testowania spójności podsystem*ów* (rys. 3). Nie testuje się prawidłowości zachowania jednostki sterującej podczas współpracy z układami peryferyjnymi, np. łączami szeregowymi typu: RS-232/485 czy Ethernet.

6.3. Sterownik w pętli

Jest to ostatni etap walidacji projektu przed finalnym testem.W układzie regulacji umieszczony jest kompletny sterownik w wersji finalnej wraz z zaprojektowanymi peryferiami, tj. portami transmisji szeregowej, czujnikami oraz układami mocy. Testowaniu poddawany jest sterownik w takiej formie, w jakiej ma sterować rzeczywistym obiektem. Obiekt sterowania w dalszym ciągu jest symulowany (rys. 6).



Rys. 6. Sterownik w pętli –HIL
Fig. 6. Hardware-in-the-loop – HIL

Testy typu sterownik w pętli umożliwiają walidację regulatora w pełnym zakresie jego funkcjonalności z uwzględnieniem także zastosowanych peryferii. Testy ułatwiają weryfikację zgodności zachowania sterownika z warunkami podanymi w specyfikacji z uwzględnieniem także wszelkich interakcji czasowych.

6.4. Model środowiska pracy regulatora

Przedstawione techniki (MIL, PIL i HIL) wymagają dobrze przygotowanego modelu matematycznego całego środowiska, w którym planuje się umieścić sterownik. Jest to podstawowy warunek, aby przeprowadzić proces weryfikacji i walidacji w sposób wiarygodny. Pod pojęciem dobrze przygotowany rozumie się nie tylko własność dokładnego symulowania dynamiki obiektu, ale także możliwość podpięcia regulatora do sterowanego obiektu za pomocą zaprojektowanych linii wejściowych czujników, magistral komunikacyjnych, elementów wykonawczych i wzmacniaczy mocy. Spełnienie tego warunku nie zawsze jest łatwe. W zastosowaniach przemysłowych klasyczny komputer PC wyposażony w standardowe karty wejścia/wyjścia na ogół nie wystarcza. Dalej zamieszczono przykładowe zdjęcie przemysłowych symulatorów dedykowanych do testów typu HIL (rys. 7).

Rys. 7. Przykładowe symulatory do zastosowań typu sterownik w pętli [15]
Fig. 7. Exemplary simulators for hardware-in-the-loop applications [15]

Przemysłowe symulatory wyposażone są w specjalistyczne linie I/O, które potrafią symulować sygnały z enkoderów, magistrale: CAN, FlexRay lub RS-485, sygnały z czujników typu: kamera, radar lub lidar, a także odpowiednio reagować na podłączone do nich sygnały sterujące.

7. Projektowanie na bazie modelu matematycznego

Obserwowany gwałtowny rozwój technologiczny spowodowany jest w dużej mierze pojawieniem się zaawansowanych narzędzi do modelowania zjawisk fizycznych i ogólnie pojętego otoczenia, z którym współpracują systemy wbudowane [19–22, 27]. Powstały biblioteki z kompletnymi modelami:

- procesów fizycznych i chemicznych – przepływ ciepła lub cieczy, rozkład naprężeń w materiałach, fermentacja, destylacja itp.,
- urządzeń mechanicznych i elektroenergetycznych,
- otoczenia – ruch drogowy z uwzględnieniem sygnalizacji świetlnej, przemieszczających się pojazdów i pieszych, ruch morski, portowy, ruch powietrzny,
- systemów – układy wspomagania autonomiczne przemieszczanie się pojazdów, czujniki radarowe, laserowe, kamery, autopiloty,
- sieci neuronowych.

Obecnie nie jest konieczna rozległa wiedza matematyczna, aby przygotować zaawansowany matematycznie model środowiska pracy sterownika. Najczęściej w tym celu wystarczy umiejętność obsługi jednego z dostępnych środowisk projektowo-symulacyjnych oraz znajomość zawartości dostarczonych wraz z tym środowiskiem bibliotek. Łatwość, z jaką można wykorzystać zgromadzoną w udostępnianych modelach wiedzę spowodował powstanie nowej metodyki projektowania systemów wbudowanych. Jest to projektowanie określane jako: na bazie modelu matematycznego (ang. *Model Based Design*). Jest ono także intensywnie promowanym podejściem projektowym przez twórców od narzędzi do modelowania [21, 24, 28].

Podstawowa różnica między podejściem MBD, a klasycznym, polega na prowadzaniu większości testów, ocen i poprawek w środowisku wirtualnym, tj. symulującym warunki panujące w rzeczywistości. Największą zaletą tego podejścia jest możliwość szybkiego przygotowania i przeprowadzenia testów. W praktyce, nierzadko przygotowanie pojedynczego testu zajmuje kilka dni lub dłużej. W środowisku wirtualnym przygotowanie analogicznego testu może trwać kilkanaście minut.

W metodyce opartej na MBD można wskazać kilka charakterystycznych faz:

- modelowanie docelowego środowiska pracy regulatora lub sterownika wbudowanego,
- opracowanie wirtualnego sterownika,
- przeprowadzenie symulacji pracy wirtualnego układu sterowania[1],
- wdrożenie regulatora w wirtualnym środowisku,
- finalne testy w środowisku rzeczywistym.

Koszty testów wirtualnych są również nieporównywalnie mniejsze, zwłaszcza jeśli podczas eksperymentów może dojść do uszkodzenia mechanicznego. Przeprowadzenie symulacji prowadzącej do zniszczeń w środowisku lub sterowniku nie zatrzymuje dalszych testów. Symulacje z tymi samymi wirtualnymi obiektami można dalej kontynuować. W tych cechach podejścia typu MBD przejawia się ogromna przewaga. Trudno sobie wyobrazić, aby można było prowadzić dziesiątki testów, które prowadzą do uszkodzenia obiektów sterowanych. Często są to bardzo drogie urządzenia, np. samoloty lub instalacje rafineryjne. MBD umożliwia sprawdzenie projektu również w warunkach prowadzących do katastrofy.

[1] Środowiskiem wirtualnym dla np. samochodowego sterownika ABS jest model konstrukcji ramy samochodu wraz z modelem zawieszenia, modelem silnika, modelami innych sterowników, zamontowanych w samochodzie, czujników itp.

Podobny sposób postępowania przy tworzeniu oprogramowania narzucają normy przemysłowe ISO 26262 oraz DO-178C.

8. Podsumowanie

Analizując korzyści oraz niedoskonałości przedstawionych technik weryfikacji i walidacji, w pierwszej chwili można odnieść wrażenie, że jest to przysłowiowe „dzielenie włosa na czworo" i można by z powodzeniem pominąć testy: model w pętli i oprogramowanie w pętli, a poprzestać na procesor w pętli i sterownik w pętli i równie dobrze na poziomie tych ostatnich wykrywać błędy. W przypadku mniejszych projektów takie podejście jest racjonalne. Należy jednak podkreślić, że błędy niewykryte na wcześniejszym etapie mogą w kolejnych fazach projektu generować dodatkowe „swoje" błędy. Pomijając techniki model i oprogramowanie w pętli twórca systemu wbudowanego może stanąć przed koniecznością „rozprawienia się" ze znacznie większą ilością błędów, niż wynikałoby to z prostej sumy niewykrytych błędów z poszczególnych etapów. Pominięcie wspomnianych etapów testowania zwiększa również prawdopodobieństwo przedostania się większej ilości błędów do końcowego etapu testów, czyli podczas testowania systemu w naturalnym środowisku pracy, lub w jeszcze gorszym scenariuszu, część z nich zostanie wykryta dopiero na etapie użytkowania przez nabywców. Do tego ostatniego szczególnie nie należy dopuszczać, gdyż jest to poziom, na którym usuwanie i wykrywanie błędów jest najdroższe i najbardziej również destrukcyjne dla wizerunku producenta.

Warto zwrócić uwagę, że wdrożenie nowoczesnych narzędzi do modelowania zjawisk fizycznych odwróciło nieco trend niewykorzystywania przez oprogramowanie możliwości sprzętu. Obecnie ma się wrażenie, że sytuacja jest odwrotna. Istnieje wiele problemów, dla których znane są algorytmy prowadzące do ich rozwiązania, ale moc obliczeniowa sprzętu jest zbyt mała, aby otrzymać rozwiązanie w satysfakcjonującym czasie.

Bibliografia

1. Sommerville I., *Inżynieria Oprogramowania*, X. Warszawa: Wydawnictwo Naukowe PWN, 2020.
2. Sacha K., *Inżynieria oprogramowania*. Wydawnictwo Naukowe PWN, Warszawa 2014.
3. Nauri P., Randell B. (red.), *Software Engineering: Report on a conference sponsored by the NATO SCIENCE COMMITTEE*, Garmisch, Germany, październik 1969 r. [Online]. http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF.
4. Dijkstra E.W., *The humble programmer*, Commun. ACM, t. 15, Nr 10, 1972, 859–866, DOI: 10.1145/355604.361591.
5. Lions J.-L., *Ariane 5, Flight 501 Failure Report by the Inquiry Board*, The Inquiry Board set up by Director General of ESA and the Chairman of CNES, Paris 1996. Udostępniono: 4.07.2020 r. [Online]. https://esamultimedia.esa.int/docs/esa-x-1819eng.pdf.
6. Zhang B., *Pierwszy śmiertelny wypadek Tesli jadącej z włączonym autopilotem*, Business Insider, 1.07 2016 r. //businessinsider.com.pl/technologie/nowe-technologie/tesla-z-autopilotem-pierwszy-smiertelny-wypadek/jhcjz8m (udostępniono 2020.07.04.)
7. Technical Committee : ISO/TC 299 Robotics, ISO 10218-1:2011 Robots and robotic devices – Safety requirements for industrial robots – Part 1: Robots. 2011.
8. Technical Committee : ISO/TC 210, IEC 62304:2006 Medical device software – Software life cycle processes, 2006.
9. Kirner R., *Testen von Embedded, Hardware in the Loop (HIL) Testing*. wrzesień 2008 r., Udostępniono: 1.10.2006 r. [Online]. http://ti.tuwien.ac.at/cps/teaching/courses/testing_emb_sys-ws07/Documents/slides/tes6_hil_testing.pdf.
10. Hanaii R. i in., *Proposal of architecture and implementation process for IEC61508 compliant, dependable robot systems*, 2012 IEEE International Conference on Robotics and Biomimetics (ROBIO), Guangzhou, China, grudzień 2012 r., 1218–1223, DOI: 10.1109/ROBIO.2012.6491136.
11. International Electrotechnical Commission, IEC 61508-3:2010 Functional safety of electrical/electronic/programmable electronic safety-related systems – Part 3: Software requirements. 2010.
12. Erkkineni T., Conrad M., *Verification, Validation, and Test with Model-Based Design*, październik 2008 r., DOI: 10.4271/2008-01-2709.
13. Babbage Charles named blog by N.V., „Tech. View: Cars and software bugs", The Economist, 16.05.2010 r.
14. *dSPACE*. [www.dspace.com/en/pub/start.cfm] (udostępniono 6.07.2020 r.).
15. *dSPACE press images for downloading*. [www.dspace.com/en/ltd/home/news/dspace_pressroom/visuals.cfm] (udostępniono 1.06.2016 r.).
16. „ISO 26262-1:2011(en), Road vehicles — Functional safety — Part 1: Vocabulary". https://www.iso.org/obp/ui/#iso:std:iso:26262:-1:ed-1:v1:en (udostępniono 6.07.2020 r.).
17. Technical Committee : ISO/TC 299 Robotics, ISO 13482:2014 Robots and robotic devices — Safety requirements for personal care robots. 2014.
18. IEC 61496-1:2012 Safety of machinery – Electro-sensitive protective equipment – Part 1: General requirements and tests.
19. *COMSOL: Multiphysics Software for Optimizing Designs*, COMSOL. [www.comsol.com] (udostępniono 21.10.2020 r.).
20. *MathWorks– Makers of MATLAB and Simulink*. [www.mathworks.com].
21. *SCADE Suite: Integrated Model-Based Design & Development Environment | Ansys*. [www.ansys.com/products/embedded-software/ansys-scade-suite] (udostępniono 8.07.2020 r.).
22. *What is LabVIEW? – National Instruments*. [www.ni.com/pl-pl/shop/labview.html].
23. Baranowski J., Garbarz-Glos B., Noga H., Pauluk D., Pauluk M., *Platforma Maple T.A. – zastosowanie w edukacji matematycznej*, „Edukacja Ustawiczna Dorosłych", T. 1, Nr 92, 2016, 132–140.
24. *Model-Based Design of Large-Scale Mining Equipment using MapleSim and Maple – Maplesoft*. [www.maplesoft.com/company/casestudies/Stories/Model-BasedDesign.aspx].
25. Embedded Coder. [www.mathworks.com/products/embedded-coder.html].
26. *TargetLink – dSPACE*. [www.dspace.com/en/pub/home/products/sw/pcgs/targetlink.cfm].
27. *Dymola – Dassault Systèmes*. [www.3ds.com/products-services/catia/products/dymola].
28. *Simulink – Simulation and Model-Based Design*. [www.mathworks.com/products/simulink.html].

Modern Technologies of Designing Automation Systems

Abstract: The paper presents the currently used techniques for the development of the automation control systems. It begins with the introduction of the concepts used in software engineering: software crisis, software disaster, and software life cycle. The subsequent chapters extend the latter by including the most popular software development models. Then, based on the V model, the role of verification and validation in the software and controller life cycle is presented, and the test techniques used in the controller validation are given. These test types include: software in the loop, processor in the loop, and controller in the loop. The last chapter describes the technique of designing automation systems based on the advanced mathematical models Model-Based Design.

Keywords: embedded systems, quality of software, software crisis, software life cycle, V-Model, verification and validation, MIL, SIL, PIL, HIL model-in-the-loop testing

dr inż. Mariusz Pauluk
mp@agh.edu.pl
ORCID: 0000-0002-3829-3561

Absolwent Akademii Górniczo-Hutniczej w Krakowie w dyscyplinie elektronika. Stopień naukowy doktora nauk technicznych w dyscyplinie automatyka i robotyka uzyskał w 2001 r. w macierzystej uczelni na Wydziale Elektrotechniki, Automatyki, Informatyki i Elektroniki. Pracuje na stanowisku adiunkta w Katedrze Automatyki i Robotyki AGH. W swoich pracach badawczych zajmuje się praktycznym wdrażaniem zaawansowanych algorytmów sterujących w konstrukcjach mechatronicznych, ze szczególnym uwzględnieniem metod optymalizacji, przetwarzania cyfrowego sygnałów oraz systemów czasu rzeczywistego.