Robert ŁUKASZEWSKI, Piotr ZAWISTOWSKI
POLITECHNIKA WARSZAWSKA. WYDZIAŁ ELEKTRONIKI I TECHNIK INFORMACYJNYCH, INSTYTUT RADIOELEKTRONIKI,
ul. Nowowiejska 15/19, 00-665 Warszawa

Metodyka prowadzenia projektów oprogramowania systemów pomiarowo-sterujących

Dr inż. Robert ŁUKASZEWSKI

Od 1997 roku pracownik pracowni Komputerowej Techniki Pomiarowej, Instytutu Radioelektroniki Politechniki Warszawskiej. Prowadzone prace naukowo-badawcze oraz publikacje ściśle związane są z projektowaniem, modelowaniem i implementacją oprogramowania rozproszonych systemów pomiarowo-sterujących. Szczegółowy obszar zainteresowań obejmuje interfejsy pomiarowe, sieci komputerowe, projektowanie systemów, języki formalne, bezpieczeństwo systemów.

e-mail: rlukaszewski@ire.pw.edu.pl

Mgr inż. Piotr ZAWISTOWSKI

Doktorant na kierunku Elektronika na Wydziale Elektroniki i Technik Informacyjnych Politechniki Warszawskiej oraz student na kierunku Magisterskie Studia Menadżerskie na Wydziale Zarządzania Uniwersytetu Warszawskiego. Pracuje jako analityk w obszarze integracji systemów informatycznych w PTK Centertel. Obszar zainteresowań zawodowych to zarządzanie projektami i integracja systemów informatycznych.

e-mail: piotr.zawistowski@gmail.com

Streszczenie

Artykuł dotyczy problematyki metodyk prowadzenia projektów systemów pomiarowo-sterujących. W artykule przedstawiono koncepcję autorskiej metodyki opracowanej na podstawie metodyk już istniejących. Zaprezentowano założenia organizacji zespołu, cyklu życia i zarządzania czasem, procesy oraz eksperyment przeprowadzony z wykorzystaniem metodyki.

Słowa kluczowe: systemy pomiarowo-sterujące, zarządzanie projektami, inżynieria oprogramowania, PRINCE2, SCRUM, DSDM.

Project management methodology for control and measuring systems

Abstract

Project management as well as control and measuring systems are the issues that the paper is focused on. It is the first time the correlation between them is discussed. The conception of the project management methodology developed on the basis of several existing methodologies is presented in the paper (Section 2). Key issues of team organization, life cycle, time management and processes (Fig. 1) are shown and described in order to fully understand the assumptions of the methodology (Section 3). To analyse correctness of the methodology, the experiment was conducted (Section 4) during which participants wrote programs for two distributed control and measuring systems with use of the software engineering technic – Test-Driven Development methodology with dedicated LabVIEW tools (NI Unit Test Framework Toolkit and JKI VI Tester). The positive result of the experiment as well as reception of the developed approach proved methodology to be highly usable for the environment of control and measuring systems.

Keywords: control-measuring systems, project management, software engineering, PRINCE2, SCRUM, DSDM.

1. Wprowadzenie

Inżynieria oprogramowania, metodyka prowadzenia projektów informatycznych, metodyka tworzenia oprogramowania - te pojęcia są często ze sobą mylone, nawet przez osoby związane z prowadzeniem projektów. Inżynieria oprogramowania opisuje cykl życia oprogramowania oraz metody używane w trakcie każdej fazy cyklu życia. Metodyka prowadzenia projektu koncentruje się na aspektach związanych z organizacją projektu - podziałem na fazy, formowaniem zespołów, tworzeniem dokumentacji projektowej. Metodyka tworzenia oprogramowania koncentruje się na sposobach ułatwiających, przyśpieszających czy też poprawiających jakość tworzonego oprogramowania i obejmuje etapy projektowania, implementowania oraz testowania [1, 2].

Bardzo często granica między metodyką tworzenia oprogramowania a prowadzenia projektu jest cienka, ponieważ metodyki stają się coraz bardziej kompleksowe i obejmują więcej elementów niż wynikałoby z ich definicji.

W przypadku niektórych metodyk tworzenia oprogramowania ich specyfika wymusza pewną organizację projektu, konkretny podział na zespoły czy interwały czasowe.

Dostrzegalny jest problem braku metodyki prowadzenia projektów dla Systemów Pomiarowo-Sterujących (SPS). Oczywistym wydaje się, że SPS powinny być wspierane przez dedykowane rozwiązania, ale jest inaczej. W szczególności nie znaleziono żadnej metodyki prowadzenia projektów w LabVIEW. Celem tego artykułu jest zarówno przedstawienie własnej koncepcji metodyki, jak i zwrócenie uwagi na ważność zagadnienia.

2. Koncepcja i założenia metodyki

U źródeł opracowania nowej metodyki znajduje się konieczność opracowania nowoczesnego podejścia do prowadzenia projektów SPS. Metodyka powinna umożliwiać przeprowadzenie projektu w sposób zestandaryzowany i uniwersalny, wykorzystując: zdefiniowane uprzednio procesy, standardową strukturę zespołu, zarządzanie czasem i dokumentami. Zarówno sposób tworzenia oprogramowania, najlepsze praktyki programistyczne [3, 4] czy też rozmiar przedsięwzięcia nie powinny być narzucone lub w jakikolwiek sposób ograniczone przez metodykę. Efektem końcowym powinien być produkt wysokiej jakości, przy czym wysoka jakość jest pojęciem względnym, oddzielnie definiowalnym dla każdego projektu. Metodyka powinna uwzględniać możliwości wykorzystania rozwiązań inżynierii komputerowej, w tym metodyk zwinnego programowania (ang. Agile software development), m.in. Test-Driven Development (TDD), Feature-Driven Development (FDD), programowanie przyrostowe.

W celu opracowania nowej metodyki połączono zalety istniejących już metodyk. Koncepcję procesów i dokumentów tworzonych w czasie projektu zaczerpnięto z PRINCE2 (ang. PRojects In Controlled Environment) [5], modyfikując je i dostosowując ich liczbę do rzeczywistych potrzeb projektów SPS. Zrezygnowano m.in. z procesów Zarządzania strategicznego projektem na rzecz zwiększenia kompetencji Kierownika Projektu (KP), zmieniono procesy Planowania usuwając procesy dające ten sam efekt, a pojawiające się w różnych momentach projektu. Różnice w ilości procesów i w podziale są znaczne, natomiast założenia i koncepcja nie zmieniły się. Zarządzanie czasem i podział zespołu wzorowano na SCRUM (nazwa oznacza figurę w rugby) [6], zmieniono jednak nazwy na podobne do tych z PRINCE2. Przy opracowywaniu koncepcji zespołu położono nacisk na znalezienie równowagi między wymaganiami klienta a możliwościami wykonawcy oprogramowania, czego efektem jest kombinacja koncepcji zespołu według SCRUM i DSDM (ang. Dynamic Systems Development Method) [7]. Metodyki prowadzenia projektów na pierwszym miejscu stawiają klienta, co jest słusznym założeniem, jednak w przypadku SPS należy brać pod uwagę zdanie wykonawcy, ponieważ może się okazać, że zamówiono funkcjonalno-

ści, które nie są możliwe do zrealizowania albo koszt ich realizacji jest bardzo duży.

3. Opis metodyki

Metodyka opiera się na czterech filarach - organizacji zespołu, cyklu życia i zarządzaniu czasem, procesach i dokumentach. Z powodu ograniczeń w rozdziale pominięto opis dokumentów.

Dobór właściwych osób i przydział odpowiednich obowiązków jest kluczem do sukcesu. Osoby zaangażowane w projekt można podzielić na trzy hierarchiczne grupy - nadzorczą, zarządzającą i wykonawczą. Grupa wykonawcza odpowiada przed zarządzającą, zarządzająca - przed nadzorczą.

Zadaniami grupy nadzorczej jest podejmowanie kluczowych decyzji odnośnie projektu. Grupa składa się z Zespołu Nadzorczego (ZN) wybieranego przez zarząd firmy. Przewodniczący ZN podlega zarządowi, spełnia kryteria: dostępności w czasie trwania projektu, umocowania w organizacji i kompetencji (posiada doświadczenie we wdrażaniu projektów i wiedzę techniczną).

Przewodniczący wybiera KP, Konsultanta Technicznego (KT) i Głównego Architekta (GA). KP to przedstawiciel grupy zarządzającej, posiada kompetencje w prowadzeniu projektów informatycznych. GA i KT to osoby posiadające wiedze na temat SPS. GA jest przedstawicielem wykonawcy i odpowiada za wizję systemu, rozwiązuje wątpliwości natury technicznej. KT jest osobą z firmy zlecającej wykonanie systemu. Odpowiada za kwestie związane z funkcjonalnością systemu, spełnianiem wymagań projektowych. Współpraca KP, GA i KT jest kluczowa dla sukcesu projektu. Rozdział kompetencji wymusza kompromisy, sprawia, że realizacja projektu jest możliwa, a funkcjonalność zgodna z oczekiwaniami klienta.

Grupa wykonawcza to Kierownicy Zespołów (KZ) i programiści. KZ odpowiadają przed KP, a wątpliwości natury technicznej wyjaśniają z GA. KZ powinien łączyć w sobie umiejętność kierowania zespołem i wiedzę techniczną. Do jego kompetencji należy wybór zespołu, który przedstawia GA. Zespół złożony jest z programistów i wykonuje powierzone mu zadania. Żadna funkcja w zespole ani przydział do konkretnej grupy nie musi być stały. W przypadku małych projektów KZ może brać udział w tworzeniu czy testowaniu aplikacji, w czasie dużych projektów powinien koncentrować się na sprawnym zarządzaniu zespołem.

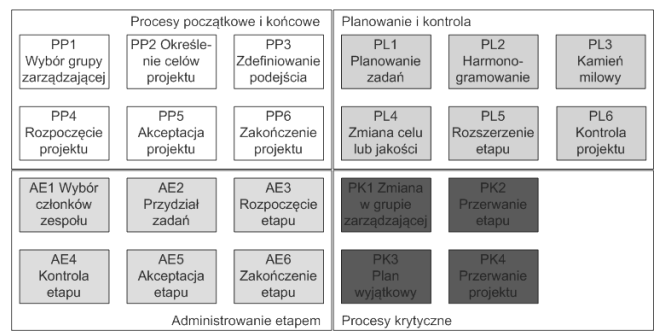
Zamierzeniem opracowanej metodyki nie jest określanie merytorycznych aspektów tworzenia oprogramowania, lecz koncentracja na administrowaniu projektem. Z tego powodu metodyka nie narzuca modelu cyklu życia oprogramowania, umożliwiając realizację dowolnej koncepcji. Wyznaczanie ścisłych ram czasowych jest trudne i praktycznie niemożliwe do zaplanowania. Projekt zmienia się dynamicznie, ma swoją specyfikę, swoje problemy. Zarządzanie czasem powinno pozostać w kompetencjach KP. Jego doświadczenie i wsparcie GA umożliwi lepsze zarządzanie czasem niż wykorzystanie narzuconych ram.

Projekt powinien mieć swoją rytmikę, dlatego sugestią jest wprowadzenie równych podziałów czasu - etapów. Długość etapu powinna być uzależniona od skali projektu, napotykanych problemów. Okresy tygodniowe, dwutygodniowe czy miesięczne pozwolą na nadanie rytmu, ułatwią kontrolę nad postępami i przydział zadań.

W ramach zadań projektowych należy określić tzw. kamienie milowe, czyli momenty szczególnie ważne dla projektu. Przykładem kamienia milowego może być przeprowadzenie pierwszych pomiarów, łączność pomiędzy węzłami systemu. Ich określenie należy do grupy zarządzającej, która przedstawia je ZN do akceptacji. Kamienie milowe nie muszą być osiągane w równych odstępach czasu. Liczba etapów potrzebnych na ich realizację może być różna i leży w gestii KP i GA.

Ze względu na odmienność każdego projektu i brak możliwości przewidzenia wszystkich sytuacji mogących wystąpić w projekcie, w trakcie jego trwania powinny być uruchamiane różne procesy. Uruchamianie poszczególnych procesów ma dwie główne zalety - jest przejrzyste oraz ułatwia zarządzanie i kontrolę. Idea procesów opiera się na ich użyciu wtedy, kiedy zachodzi taka konieczność. Dzięki temu projekt ma szansę stać się sformalizowany w minimalnym stopniu i maksymalnie efektywny. Zaleta takiego podejścia jest widoczna szczególnie w przypadku dużych projektów, które wymagają ścisłej kontroli. Zrezygnowanie z procesów może oznaczać chaos, co w konsekwencji może doprowadzić do porażki przedsięwzięcia.

Procesy podzielono na cztery grupy (rys. 1), które określają zawarte w nich podprocesy. Ich nazwy to: Procesy początkowe i końcowe, Planowanie i kontrola, Administrowanie etapem i Procesy krytyczne. Przykładowo pierwsza grupa obejmuje procesy związane z rozpoczynaniem i końcem projektu: Wybór grupy zarządzającej, Określenie celów i jakości projektu. Zdefiniowanie projektu i Rozpoczęcie projektu, jak również Akceptację projektu i Zakończenie projektu; Procesy krytyczne to Zmiana w grupie zarządzającej, Przerwanie etapu, Plan wyjątkowy, Przerwanie projektu. Powyższy podział procesów i podprocesów jest intuicyjny i przejrzysty w przeciwieństwie do popularnego PRINCE2. Uniknięto też redundancji funkcji procesów.



Rys. 1. Procesy i podprocesy metodyki zarządzania projektami
Fig. 1. Processes and subprocesses of the project management method

Wykorzystanie procesów przypomina układanie kostek domina. Przykładowo rozpoczynamy projekt podprocesem wybór grupy nadzorczej (PP1), po którym następuje określenie celów i jakości projektu (PP2), zdefiniowanie podejścia (PP3) i formalne rozpoczęcie (PP4). Kolejnymi procesami są planowanie zadań (PK1) i harmonogramowanie (PK2). W tym momencie rozpoczynamy etap – wybieramy członków zespołu (AE1), przydzielamy zadania (AE2), itd. Projekt kończy się sukcesem (PP6) lub porażką (PK4).

Proponowana metodyka uwzględnia specyfikę projektowania SPS. Z jednej strony na rynku brakuje osób posiadających wiedzę na temat SPS, dlatego trzeba wykorzystywać ich kreatywność i doświadczenie w optymalnym stopniu. Z drugiej strony nie zawsze wszystkie zespoły mają dostęp do sprzętu w dowolnym momencie. Właściwe administrowanie zasobami pozwoli na uniknięcie oczekiwania zespołów na dostęp do urządzeń. W praktyce skróci to czas trwania projektu i zmniejszy koszty. Zarządzanie zasobami ma miejsce w kilku podprocesach, m.in. w harmonogramowaniu (PK2), w czasie którego KP i GA przedstawia do akceptacji KT terminy realizacji zadań i kamieni milowych. W czasie podprocesu tworzony jest Harmonogram, który może ulegać zmianom w czasie trwania projektu, podobnie jak lista zadań. Inny podproces przydzielający zasoby to: Wybór członków zespołu (AE1), który może odbywać się przed rozpoczęciem każdego etapu. W przeciwieństwie do tradycyjnych metodyk opracowana koncepcja kładzie duży nacisk na sam etap tworzenia. Inżynierowie tworzą finalny produkt, dlatego nie należy marginalizować ich roli. Z drugiej zaś strony nie można prowadzić bardzo poważnych projektów w sposób wręcz spontaniczny, jak to ma miejsce w przypadku metodyk zwinnych. Metodyka jest więc próbą znalezienia złotego środka pomiędzy dwoma skrajnie różnymi podejściami.

4. Wykorzystanie metodyki w projektowaniu SPS

Opracowana metodyka zarządzania projektami SPS może być wykorzystana podczas tworzenia oprogramowania SPS w typowych środowiskach, takich jak LabVIEW, LabWindows czy VEE. Przydatność opracowanej metodyki zarządzania projektami SPS sprawdzono w praktyce. Jako środowisko programowe wykorzystano LabVIEW, najpopularniejsze i najbardziej zaawansowane środowisko tworzenia oprogramowania SPS. Projekt wykonano według metodyki tworzenia oprogramowania zgodnej z filozofią programowania zwinnego – TDD (ang. Test-Driven Development) [8]. Proces tworzenia aplikacji zgodnie z TDD jest następujący: najpierw pisany jest test, którego program nie przechodzi. Następnie dodawany jest fragment kodu i program poddawany jest wszystkim do tej pory napisanym testom. Zaletą takiego rozwiązania jest wysoka odporność programu na błędy i kontrola postępów. Program nie musi być dodatkowo testowany po zakończeniu tworzenia. Wadą jest konieczność przełamania bariery psychologicznej (program powstaje w sposób całkowicie inny od powszechnie stosowanych metodyk) oraz konieczność wykorzystania dedykowanych narzędzi programistycznych. Podczas projektu wykorzystano narzędzia NI Unit Test Framework oraz JKI VI Tester. Są to jedyne narzędzia wspierające TDD dedykowane dla LabVIEW.

W eksperymencie brało udział 15 osób – jeden Przewodniczący ZN, jeden KP, jeden GA i dwa zespoły złożone z jednego KZ i pięciu programistów. Zadania zespołów polegały na napisaniu aplikacji w środowisku LabVIEW. Jednym z założeń metodyki jest wspieranie nowoczesnego tworzenia oprogramowania, dlatego narzucono uczestnikom wykorzystanie metodyki TDD wraz z dedykowanymi narzędziami. Obydwa zespoły dostały to samo zadanie, przy czym każda z grup musiała skorzystać z wskazanego narzędzia. Projekt trwał osiem tygodni. Przez pierwszy tydzień uczestnicy zapoznali się z założeniami opracowanej metodyki i TDD, instalowali oprogramowanie i starali się opanować narzędzia. Programy powstawały w ciągu sześciu tygodni, dodatkowo przewidziany został tydzień na poprawki.

Obydwu zespołom udało się pozytywnie zakończyć projekt w wyznaczonym czasie. Pierwszy zespół przedstawił projekt gorszej jakości. Pojawiały się problemy przy komunikacji pomiędzy węzłami sieci, nie wszystkie wymagania projektowe zostały zrealizowane, np. wyświetlanie informacji o pomiarach na stronie internetowej. Obsługa bazy danych nie była do końca dopracowana, m.in. brakowało opcji filtrowania wyników. Projekt drugiego zespołu był pozbawiony tych wad i lepiej spełniał oczekiwania.

Zastosowanie narzędzi do TDD nie wpłynęło na jakość końcowego produktu. Uczestnicy ”tracili” czas na poznawanie narzędzi zamiast na pisanie programu. Musieli także podnieść umiejętności programowania w LabVIEW oraz przywyknąć do nowego podejścia programistycznego. Ostatecznie zespół z gorszym narzędziem szybciej dotarł do kresu jego możliwości, dlatego miał więcej czasu na tworzenie aplikacji.

Metodyka spełniła swoje zadanie i obydwa zespoły ukończyły projekt. Jednym z czynników sukcesu była sprawna komunikacja oraz jasny podział kompetencji. Uczestnicy potrafili zaakceptować swoje role oraz zadania, jakie im przydzielano. Pojawiające się problemy były rozwiązywane przez odpowiednie osoby. Przewodniczący ZN miał kontakt tylko z KP oraz GA. KZ kontaktował się z KP w celu wyjaśnienia problemów, przedstawienia cotygodniowych raportów i w sprawach organizacyjnych. KZ odpowiadał za komunikację w swoim zespole. Każdy członek miał kontakt z KZ i GA, z którym konsultował się w sprawach technicznych.

Opracowane procesy nie komplikują pracy. Są niewidoczne dla zespołów programistów, pozwalają natomiast KP, jak i Przewodniczący ZN na kontrolę postępów w projekcie. Rzeczywisty przebieg projektu jest zgodny z wykorzystanymi procesami, co dowodzi ich logiczności i słuszności.

Wprowadzenie równych interwałów czasowych okazało się bardzo trafnym pomysłem – uczestnicy przyzwyczaili się do regularnej pracy. Problemem wielu projektów jest intensyfikowanie pracy przed zbliżającym się terminem oddania. W każdym projekcie pojawiają się problemy i w większości przypadków można je rozwiązać, potrzeba na to jednak czasu. Ponadto zespoły osiągnęły dodatkowy cel (opanowanie narzędzi i wyciągnięcie wniosków z metodyki TDD). TDD jest nietypowa pod względem sposobu tworzenia oprogramowania. Użyteczność metodyki prowadzenia projektów w konteście wykorzystania TDD dowodzi jej uniwersalności.

Metodyka wymienia kilkanaście standardowych rodzajów dokumentów służących komunikacji. W czasie eksperymentu wykorzystano Harmonogram, Listę Kamieni Milowych, Raport Tygodniowy i Raport Końcowy. Liczba dokumentów i częstotliwość ich dostarczania była odpowiednia - umożliwiały kontrolę nad projektem nie obciążając zbytnio KZ. Problemem tradycyjnych metodyk jest przeładowanie formalnościami. Dokumenty tworzone są w zbyt dużej ilości i często nie są wykorzystywane. Proponowana metodyka pozbawiona jest tego problemu - tworzone są tylko potrzebne dokumenty.

5. Wnioski

W artykule zaprezentowano autorską metodykę prowadzenia projektów SPS. Przedstawiono jej główne założenia – kształtowanie zespołu, zarządzanie czasem, zarządzanie projektami. Z uwagi na ograniczony rozmiar artykułu pominięto problem dokumentacji. Wykonano projekty SPS z wykorzystaniem TDD oraz autorskiej metodyki prowadzenia projektów w środowisku LabVIEW. Połączenie obydwu metodyk jest rozwiązaniem gwarantującym właściwą organizację projektu SPS, co w konsekwencji może przełożyć się na wysoką niezawodność.

Oczywiście nie można twierdzić, że metodyka gwarantuje sukces. Na sukces składa się wiele czynników. Od członków zespołów wymagane są umiejętności adekwatne do ich obowiązków. Wyniki projektu świadczą jednoznacznie o tym, że trudno nadrobić braki w wiedzy i doświadczeniu przez tak krótki okres czasu. Bez wątpienia istotne są umiejętności interpersonalne. Obydwie grupy osiągnęły cel, ponieważ potrafiły się zorganizować, były odpowiednio motywowane oraz zaangażowały się w projekt. Ważna okazała się praca w grupie, która jest kluczem do sukcesu. Zarządzanie projektami nie jest celem samym w sobie, pozwala na uporządkowanie przepływu informacji i podział obowiązków, a bez tych i wielu innych czynników projekt skazany jest na porażkę.

6. Literatura

[1] Szyjewski Z.: Metodyki zarządzania projektami informatycznymi, Placet, Warszawa 2004.
[2] Jaszkiewicz A.: Inżynieria oprogramowania, Helion, Gliwice 1997.
[3] Conway J., Watts S.: A Software Engineering Approach to LabVIEW, Prentice Hall, New Jersey 2003.
[4] Blume P.: The LabVIEW Style Book, Prentice Hall, Indiana 2007.
[5] Office of Government Commerce: PRINCE2, Crown, London 2002.
[6] Asproni G.: Wstęp do SCRUM, Software Developer’s Journal 6/2006, Warszawa.
[7] Voigt B.: Dynamic System Development Method, Department of Information Technology, University of Zurich, Zurich 2004.
[8] Szemraj R.: Test-Driven Development, portal Flashzone, Warszawa 2007.

otrzymano / received: 31.07.2010
przyjęto do druku / accepted: 01.10.2010 *artykuł recenzowany*