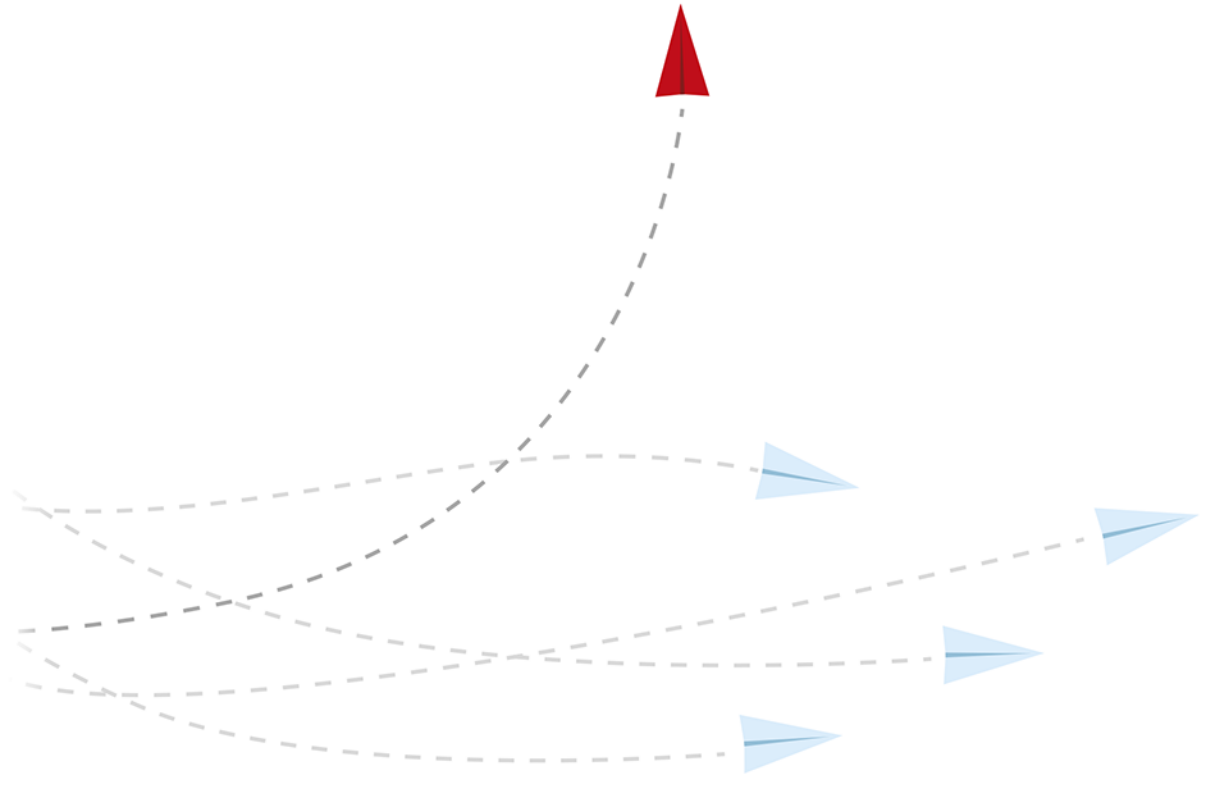
PROGRAMUJ TAK, ABY NIE NAPRAWIAĆ

[Planowanie projektów i systemów]



Helion

JUVAL LÖWY

Tytuł oryginału: Righting Software

Tłumaczenie: Piotr Rajca

ISBN: 978-83-283-6793-7

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/protak
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

SPIS TREŚCI

1 Metoda

Zen of Architects[1] stwierdza jedynie tyle, że początkujący architekt wszystko może robić na wiele sposobów. Natomiast dla mistrzów architektury istnieje jedynie kilka opcji, a przeważnie tylko jedna.

Początkujący architekci często są przytłoczeni mnogością wzorców, podejść, metod oraz możliwości projektowania tworzonego systemu oprogramowania. Branża komputerowa pęka w szwach od liczby koncepcji, jak również osób pragnących uczyć się i samodoskonalić, a Ty, Czytelniku tej książki, zaliczasz się do tego grona. Niemniej jednak, ponieważ istnieje bardzo niewiele prawidłowych sposobów na wykonanie danych zadań projektowych, równie dobrze można się skoncentrować tylko na nich i pominąć pozostały szum. Mistrzowie projektowania oprogramowania wiedzą, że tak należy robić; dzięki jakby nadnaturalnej inspiracji są w stanie błyskawicznie wskazać i zastosować prawidłowe rozwiązanie projektowe.

Zen of Architects znajduje zastosowanie nie tylko podczas projektowania systemów, lecz także w projektach, w ramach których te systemy powstają. Owszem, istnieją niezliczone sposoby określania struktury projektów i przypisywania zadań poszczególnym członkom zespołu, jednak czy wszystkie one są równie bezpieczne, szybkie, tanie, użyteczne i efektywne? Mistrzowie architektury oprogramowania planują także projekty, w ramach których będą tworzone systemy, a nawet pomagają kadrze kierowniczej w podejmowaniu decyzji, czy organizację będzie stać na wykonanie projektu.

Osiągnięcie prawdziwego mistrzostwa w każdej dziedzinie jest podróżą. Oprócz bardzo nielicznych wyjątków nikt nie rodzi się ekspertem. Doskonałym tego przykładem jest moja

[1] *https://en.wikipedia.org/wiki/Zen_Mind,_Beginner's_Mind*

własna kariera. Rozpocząłem pracę jako młodszy architekt niemal 30 lat temu, kiedy określenie „architekt oprogramowania" nie było powszechnie używane w branży komputerowej. Następnie pełniłem funkcję architekta projektu, kierownika działu projektowania, a w końcu, pod koniec lat 90. ubiegłego wieku, byłem głównym projektantem w jednej z firm z listy Fortune 100 w Dolinie Krzemowej. W roku 2000 założyłem IDesign — firmę zajmującą się jedynie projektowaniem oprogramowania. Od tamtej pory zaprojektowaliśmy w niej setki systemów i projektów. Choć każde zlecenia miało swoją specyficzną architekturę i własny plan tworzenia projektu, zauważyłem, że niezależnie od klienta, projektu, systemu, technologii czy programistów moje zalecenia projektowe były — w pewnym abstrakcyjnym sensie — takie same.

Dlatego zadałem sobie proste pytanie: czy naprawdę trzeba być mistrzem architektury oprogramowania dysponującym kilkudziesięcioma latami doświadczenia i dziesiątkami wykonanych projektów, by wiedzieć, co należy robić? I drugie pytanie: czy można w jakiś sposób usystematyzować tę wiedzę, by każdy dysponujący dobrym zrozumieniem stosowanych metod programista mógł tworzyć solidne systemy i dobre projekty?

Odpowiedź na to drugie pytanie jest bezsprzecznie twierdząca. Nazywam to ***metodą*** i to właśnie ona jest tematem niniejszej książki. Po zastosowaniu *metody* w bardzo wielu projektach i nauczeniu jej kilku tysięcy architektów na całym świecie mogę zagwarantować, że prawidłowo zastosowana, da ona dobre efekty. Nie dyskredytuję tu znaczenia właściwego nastawienia, umiejętności technicznych i możliwości analitycznych. Są one elementami koniecznymi do osiągnięcia sukcesu niezależnie od stosowanej metody. Niestety jednak nie są one jednocześnie składnikami wystarczającymi — często spotykam się z projektami, które zakończyły się porażką pomimo tego, że były tworzone przez osoby posiadające doskonałe kwalifikacje i przymioty. Niemniej kiedy zostaną one połączone z *metodą*, składniki te zapewnią siłę niezbędną do odniesienia sukcesu. Opierając swój projekt na solidnych podstawach inżynierskich, nauczysz się unikać wszechobecnych złych praktyk i błędnej intuicji.

Czym jest metoda?

Metoda jest prostą i efektywną techniką analizy i projektowania. Można by ją wyrazić, używając takiego oto wzoru:

metoda = projekt systemu + plan projektu

W ramach projektu systemu *metoda* określa sposób podziału dużych systemów na mniejsze, modularne komponenty. Udostępnia ona wytyczne na temat struktury, roli oraz semantyki komponentów oraz określa, jak komponenty te powinny ze sobą współdziałać. Efektem jest architektura systemu.

Jeśli chodzi o plan projektu, *metoda* pomaga zapewnić zarządzanie tworzeniem systemu, udostępniając przy tym kilka opcji. Każda z tych opcji stanowi połączenie harmonogramu,

kosztów oraz ryzyka. Oprócz tego każda z tych opcji stanowi instrukcję konstruowania systemu i przygotowuje projekt pod kątem jego realizacji i śledzenia.

Planowanie projektu jest tematem drugiej części tej książki i ma znacznie większe znaczenie dla odniesienia końcowego sukcesu niż projekt systemu. Nawet dysponując przeciętnym projektem systemu, można odnieść sukces, jeśli będziemy dysponować odpowiednim czasem i zasobami oraz jeśli ryzyko jest możliwe do zaakceptowania. Niemniej jednak nawet jeżeli będziemy dysponować najlepszym projektem systemu, nasz projekt zakończy się porażką, jeśli nie będziemy dysponować odpowiednim czasem czy zasobami albo jeśli projekt będzie zbyt ryzykowny. Oprócz tego planowanie projektu jest bardziej skomplikowane niż projekt systemu i, co za tym idzie, wymaga dodatkowych narzędzi, koncepcji oraz technik.

Ponieważ *metoda* składa się zarówno z projektu systemu, jak i planu projektu, stanowi ona proces projektowy. Na przestrzeni lat branża informatyczna poświęciła bardzo wiele uwagi, by stworzyć proces wytwarzania oprogramowania, a jednocześnie nie przykładała jej do procesu projektowania. Niniejsza książka ma wypełnić te luki.

Weryfikacja projektu

Weryfikacja projektu ma kluczowe znaczenie, gdyż organizacja nie powinna ryzykować, by zespół rozpoczął prace nad projektem o nieodpowiedniej architekturze lub takim, na którego stworzenie organizacji w ogóle nie stać. *Metoda* wspiera i umożliwia wykonanie tego krytycznego zadania, pozwalając architektowi na ustalenie, z rozsądnym poziomem pewności, czy proponowany projekt jest odpowiedni, czyli czy spełnia dwa kluczowe cele: po pierwsze, projekt musi zaspokajać wymagania klienta, a po drugie, musi uwzględniać możliwości i ograniczenia organizacji bądź zespołu.

Kiedy zostaną już rozpoczęte prace nad kodem, modyfikowanie architektury zazwyczaj będzie już niedopuszczalne ze względu na wiążące się z tym koszty i opóźnienia. W praktyce oznacza to, że bez weryfikacji projektu systemu istnieje ryzyko utrwalenia architektury, która w najlepszym razie będzie niedoskonała, a w najgorszym — monstrualna. Organizacja będzie musiała starać się działać, używając takiego systemu przez kilka następnych lat, aż po kilku jego wersjach zdecyduje się na znaczącą modyfikację lub przepisanie. Kiepsko zaprojektowane oprogramowanie może w dużym stopniu zaszkodzić prowadzonej działalności, ograniczając możliwości reagowania na okazje biznesowe, a nawet doprowadzić organizację do bankructwa wskutek rosnących kosztów jego utrzymania.

Wczesne weryfikowanie projektu ma ogromne znaczenie. Na przykład, choć stwierdzenie po trzech latach od rozpoczęcia prac, że dana koncepcja lub cała architektura jest nieprawidłowa, jest interesujące pod względem intelektualnym, to jednak pod względem praktycznym nie ma ono żadnej wartości. W idealnym przypadku po tygodniu od rozpoczęcia prac nad projektem należy wiedzieć, czy dana architektura będzie właściwa. Każde przekroczenie

tego czasu zwiększa ryzyko prac nad architekturą, której poprawność będzie dyskusyjna. W kolejnych rozdziałach dokładnie opiszę, w jaki sposób należy weryfikować projekt systemu.

Należy zwrócić uwagę, że odnoszę się tutaj do ogólnego projektu systemu — architektury — a nie szczegółowego projektu systemu. Projekt szczegółowy dla każdego z komponentów architektury tworzy kluczowe artefakty implementacji, takie jak hierarchie interfejsów i klas oraz kontrakty danych. Tworzenie szczegółowych projektów trwa dłużej, można je wykonywać podczas prac nad projektem, a oprócz tego mogą one ulegać zmianom w trakcie ewolucji konstruowanego systemu.

Analogicznie należy także weryfikować plan projektu. Przekroczenie limitu czasu lub dostępnego budżetu (bądź obu) podczas prac nad projektem jest całkowicie niedopuszczalne. Nieumiejętność wywiązywania się z zobowiązań może zaszkodzić naszej karierze. Dlatego należy aktywnie weryfikować plan projektu, by upewnić się, że zespół jest w stanie zrealizować projekt.

Oprócz przygotowania planów architektury i projektu celem *metody* jest wyeliminowanie projektu jako ryzyka dla powodzenia realizacji całego przedsięwzięcia. Żaden projekt nie powinien zakończyć się niepowodzeniem dlatego, że architektura była zbyt złożona, by programiści byli w stanie ją przygotować i utrzymać. *Metoda* pozwala określić architekturę sprawnie i efektywnie i zrobić to dość szybko. Dokładnie to samo dotyczy planu projektu. Żaden projekt nie powinien zakończyć się niepowodzeniem dlatego, że od samego początku na jego wykonanie brakowało czasu lub zasobów. Tak książka pokazuje, w jaki sposób można dokładnie wyliczyć czas prac nad projektem i jego koszty oraz jak podejmować przemyślane decyzje.

Presja czasu

Używając *metody*, można stworzyć cały projekt systemu w ciągu kilku dni, zazwyczaj od trzech do pięciu. Przygotowanie planu projektu zajmuje podobny okres czasu. Zważywszy na szczytne cele podejmowanych wysiłków, a konkretnie przygotowanie architektury systemu oraz planów projektu nowego systemu, ten czas może się wydawać zbyt krótki. Typowe systemy biznesowe dają możliwość tworzenia nowych projektów jedynie co kilka lat. Dlaczego nie można by poświęcić 10 dni na przygotowanie architektury? Wziąwszy pod uwagę późniejszy czas stosowania takiego systemu, liczony w latach, te pięć dodatkowych dni nie stanowi nawet błędu zaokrąglenia. Niemniej jednak wydłużenie czasu projektowania często nie poprawia jego efektów, a niekiedy nawet sprawia, że będą one gorsze.

Większość środowisk roboczych cechuje się horrendalnie niewydajnym zarządzaniem czasem, co głównie wynika z natury ludzkiej. Presja czasu zmusza nas (jak również wszystkie inne zaangażowane osoby) do skoncentrowania się, określenia priorytetów oraz wykonania projektu. *Metodę* należy zrealizować szybko i zdecydowanie.

Ogólnie rzecz biorąc, projektowanie (w przeciwieństwie do implementacji) nie jest zadaniem czasochłonnym. Architekci budowlani na projekt budynku poświęcają przeważnie nie więcej niż dwa tygodnie, zazwyczaj mieszcząc się w tygodniu, z kolei wzniesienie budynku na podstawie takiego projektu może trwać nawet dwa lub trzy lata, o ile zatrudnimy z podwykonawców.

Presja czasu pomaga także unikać cyzelowania projektu. Prawo Parkinsona[2] głosi, że prace zawsze przeciągają się, by zająć cały przeznaczony na nie czas. Mając 10 dni na wykonanie projektu, który można zakończyć w 5 dni, architekt najprawdopodobniej będzie nad nim pracował przez 10 dni. Dodatkowy czas architekt poświęci na stworzenie błahych aspektów projektu, które nie dodają do niego niczego oprócz złożoności, co w nieproporcjonalny sposób zwiększy w następnych latach koszty implementacji i utrzymania. Ograniczanie czasu zmusza nas na przygotowania projektu, który będzie jedynie wystarczająco dobry.

Eliminowanie paraliżu analizy

Paraliż analizy to kłopot, który przydarza się, kiedy osoba (albo grupa), która jest zdolna, inteligentna, a nawet pracowita (jak to zazwyczaj jest z architektami oprogramowania), utknie w nieskończonym cyklu analizy, projektowania, nowych odkryć i ponownej analizy. Taka osoba (lub grupa) ulega paraliżowi i nie jest w stanie stworzyć żadnego produktywnego wyniku.

Drzewo decyzyjne projektu

Głównym powodem występowania paraliżu jest brak wiedzy o istnieniu drzewa decyzyjnego zarówno dla systemu, jak i dla projektu. Drzewo decyzyjne projektu jest ogólnym pojęciem odnoszącym się do wszelkich zadań projektowych — nie jest powiązane wyłącznie z inżynierią oprogramowania. Każdy projekt o pewnym poziomie złożoności jest zbiorem wielu mniejszych decyzji projektowych, uporządkowanych hierarchicznie w formie struktury drzewiastej. Każde rozgałęzienie w tym drzewie reprezentuje możliwą opcję decyzyjną, prowadzącą do kolejnych, bardziej szczegółowych decyzji projektowych. Liście tego drzewa reprezentują gotowe rozwiązania projektowe dla istniejących wymagań. Każdy z tych liści jest spójnym, unikalnym i prawidłowym rozwiązaniem, które w pewien sposób różni się od wszystkich pozostałych.

Kiedy osoba lub grupa odpowiedzialna za stworzenie projektu nie zna prawidłowego drzewa decyzyjnego, może zacząć prace od innego miejsca niż jego korzeń. W takim przypadku jakaś decyzja projektowa w nieuchronny sposób doprowadzi do konfliktu z którąś z poprzednich decyzji, a co za tym idzie, konieczne będzie odrzucenie wszystkich decyzji pośrednich. Projektowanie w taki sposób można by porównać z sortowaniem bąbelkowym drzewa decyzyjnego projektu. Ponieważ liczba operacji w przypadku

[2] Cyryl N. Parkinson, *Parkinson's Law*, „The Economist", 19 listopada 1955.

sortowania bąbelkowego odpowiada kwadratowi liczby sortowanych elementów, konsekwencje takiego postępowania będą poważne. Prostszy system, wymagający podjęcie 20 decyzji związanych z zaplanowaniem systemu i projektu, może potencjalnie wymagać 400 iteracji projektu, o ile nie będziemy pracować w oparciu o drzewo decyzyjne. Przeprowadzenie takiej liczby spotkań (nawet jeśli będą one rozłożone w czasie) doprowadzi do paraliżu. Jest mało prawdopodobne, byśmy mieli czas na wykonanie nawet 40 iteracji projektu. Kiedy zakończy się czas przeznaczony na zaplanowanie systemu i projektu, pisanie kodu rozpocznie się, kiedy ani system, ani projekt nie będą jeszcze dostatecznie dojrzałe. To z kolei opóźnia odkrycia prowadzące do unieważniania decyzji projektowych, odsuwając je w jeszcze odleglejszą przyszłość, kiedy czas, wysiłki oraz artefakty zostaną już skojarzone z niewłaściwymi wyborami. W zasadzie oznacza to maksymalizację kosztów niewłaściwych decyzji projektowych.

Drzewo decyzyjne projektu systemu software'owego

Jak się okazuje, większość software'owych systemów biznesowych ma ze sobą wiele wspólnego i przynajmniej zarys drzewa decyzyjnego będzie dla nich nie tylko wspólny, lecz także jednakowy. Oczywiście liście tych drzew będą odmienne.

Metoda określa postać drzew decyzyjnych dla typowych systemów biznesowych, i to zarówno dla projektu systemu, jak i dla planu projektu. Dopiero po zaprojektowaniu systemu jest sens zabierać się za planowanie projektu, który pozwoli zbudować ten system. Każdy z tych wysiłków projektowych, zarówno związany z systemem, jak i z projektem, stanowi odrębne poddrzewo decyzji projektowych. *Metoda* prowadzi nas przez te decyzje, zaczynając od korzenia drzewa, dzięki czemu pozwala uniknąć przerabiania i ponownego oceniania wcześniejszych decyzji.

Jedną z najcenniejszych technik oczyszczania drzewa decyzyjnego jest stosowanie ograniczeń. Jak zauważył doktor Frederick Brooks[3], w przeciwieństwie do powszechnej opinii i założeń, najgorszym problemem projektowym jest pusta karta. Bez żadnych ograniczeń projekt powinien być bardzo prosty, nieprawdaż? Otóż nie. Pusta karta powinna przerażać każdego architekta. Istnieje nieskończenie wiele sposobów, by zboczyć z właściwej drogi albo przekroczyć ograniczenia, które nie zostały jawnie określone. Im więcej jest ograniczeń, tym projektowanie jest prostsze. Im mniej mamy swobody, tym projekt będzie bardziej oczywisty i przejrzysty. W systemie, w którym występują same ograniczenia, nie ma nic do projektowania — system jest jednoznaczny. Jednak zawsze istnieją jakieś ograniczenia (jawne bądź niejawne), dlatego w przypadku postępowania zgodnie z drzewem decyzyjnym *metoda* narzuca rosnące ograniczenia zarówno na system, jak i na projekt, dzięki czemu plan wyłania się i kształtuje całkiem szybko.

[3] Frederick P. Brooks Jr., *The Design of Design: Esseys from a Computer Science*, Addison-Wesley, Upper Saddle River, 2010.

Komunikacja

Ważną zaletą *metody* jest komunikowanie koncepcji projektowych. Kiedy osoby biorące udział w pracach poznają już strukturę architektury i semantykę projektu, *metoda* pozwoli im dzielić się koncepcjami projektowymi i precyzyjnie przekazywać wymagania projektu. Proces myślowy leżący u podstaw projektu można przekazywać zespołowi. Należy dzielić się informacjami o kompromisach i pomysłach, które prowadziły nas do powstania architektury, dokumentując w jednoznaczny i zrozumiały sposób przyjmowane założenia do planu i wynikające z nich decyzje projektowe.

Ten poziom przejrzystości w intencjach projektowych ma krytyczne znaczenie dla przetrwania architektury. Dobry projekt to taki, który został poprawnie wymyślony, przetrwał proces implementacji i zakończył się, działając na komputerach klientów. Musimy być w stanie przekazać ten projekt programistom i zagwarantować, że oni zrozumieją i docenią pomysły leżące u jego podstaw. Ten projekt należy wyegzekwować, stosując kontrole, inspekcje oraz mentoring. *Metoda* celuje w komunikacji tego typu dzięki wykorzystaniu połączenia dobrze zdefiniowanej semantyki usług i struktury.

Można mieć pewność, że jeśli programiści pracujący nad stworzeniem systemu nie zrozumieją i nie docenią projektu, spartaczą go. I żaden plan ani żadne inspekcje nie będą w stanie temu zapobiec. Celem tych kontroli powinno być możliwie wczesne wychwytywanie niezamierzonych odstępstw od zaplanowanej architektury.

Dokładnie tak samo dzieje się w przypadku komunikowania planu projektu menadżerom projektu, kierownictwu oraz interesariuszom. Przejrzyste, jednoznaczne i zapewniające możliwość porównywania opcje mają kluczowe znaczenie dla podejmowania świadomych i przemyślanych decyzji. Kiedy ludzie podejmują błędne decyzje, często wynika to z niezrozumienia projektu i wykształcenia nieprawidłowego mentalnego modelu sposobu jego działania. Tworząc prawidłowe modele projektu dla czasu, kosztów i ryzyka, architekt może umożliwić podejmowanie właściwych decyzji. *Metoda* zapewnia doskonały słownik i wskaźniki do prowadzenia prostej i świadomej komunikacji z osobami odpowiedzialnymi za podejmowanie decyzji. Kiedy menadżerowie będą już znali możliwości planu projektu, staną się jego największymi orędownikami i będą nalegać na pracę w proponowany sposób. Żadna ilość żarliwych argumentów nie jest w stanie dokonać tego, co prosty zbiór wykresów i liczb. Co więcej, plan projektu jest ważny nie tylko na początku prac. Kiedy prace się już rozpoczną, narzędzi planu projektu można używać do przekazywania kierownictwu efektów i zauważalnych zmian. W dodatku A przedstawiam zagadnienia związane ze śledzeniem projektu i zarządzaniem zmianami.

Oprócz komunikowania projektu programistom i menadżerom *metoda* pozwala architektom na precyzyjne i proste prezentowanie projektu innym architektom. Pomysły, jakie można zyskać dzięki takim opiniom i konstruktywnej krytyce, są wprost nieocenione.

Czym metoda nie jest?

W 1987 roku Brooks napisał, że „nie istnieje żadna srebrna kula”[4]. Bez wątpienia nie jest nią także *metoda*. Stosowanie *metody* nie gwarantuje sukcesu i może pogorszyć sytuację, jeśli będzie używana w izolacji lub jedynie dla samego faktu jej użycia.

Metoda nie odbiera architektom kreatywności ani nie zwalnia ich z konieczności włożenia wysiłku w stworzenie dobrej architektury. To wciąż architekt będzie ponosił odpowiedzialność za błędy w projektowanej architekturze oraz za nieumiejętność przekazania informacji o niej programistom i przeprowadzenia prac programistycznych w sposób umożliwiający oddanie gotowego produktu bez odstępstw od zaplanowanej architektury, a wszystko to pod coraz większą presją. Co więcej, zgodnie z wyjaśnieniami podanymi w drugiej części książki architekt jest odpowiedzialny za stworzenie wykonalnego planu projektu wynikającego z przyjętej architektury. Architekt musi dostosować projekt do dostępnych zasobów, do tego, co można stworzyć, dysponując określonymi zasobami, do ryzyka związanego z projektem oraz do wyznaczonego terminu. Przechodzenie przez wszystkie prace projektowe tylko dla nich samych jest pozbawione sensu. Architekt musi wyeliminować wszystkie uprzedzenia i stworzyć prawidłowy zestaw założeń do planu i wynikające z nich obliczenia.

Metoda stanowi dobry punkt początkowy do prac nad projektem systemu i planem projektu i zawiera także listę rzeczy, których należy unikać. Niemniej jednak będzie ona działać wyłącznie wówczas, gdy podejdziemy do niej rzetelnie oraz poświęcimy czas i wysiłek intelektualny, by zebrać wszystkie niezbędne informacje. Trzeba uważnie dbać o proces projektowy i jego efekty.

[4] Frederick P. Brooks, *No Silver Bullet: Essence and Accidents of Sortware Engineering*, „Computer”, 20, nr 4, kwiecień 1987.

MISTRZ W PROJEKTOWANIU OPROGRAMOWANIA **ROZWAŻA BARDZO NIEWIELE OPCJI!**

Tworzenie oprogramowania, zwłaszcza dużych i złożonych aplikacji, bywa problemem. Często objawy złego planowania projektu nie mają wiele wspólnego z uwarunkowaniami technicznymi: wysoki poziom stresu, duża rotacja pracowników, wypalenie zawodowe, brak zaufania, niska samoocena, a nawet różnego rodzaju dolegliwości fizyczne. Przyczyny tego stanu rzeczy są podobne: określane w nierealny sposób koszty, terminy i wymagania. Później okazuje się, że wewnętrznej złożoności systemu nikt nie rozumie, nad koniecznymi ciągłymi zmianami nikt nie panuje, a wytworzone oprogramowanie jest niemal niemożliwe do utrzymania. Jak widać, problem jest wielowymiarowy. Wielowymiarowe musi więc też być narzędzie, które umożliwi jego rozwiązanie.

W tym praktycznym przewodniku uniwersalne zasady projektowania zostały dostosowane do specyfiki wytwarzania oprogramowania. Znalazł się tu zbiór najważniejszych zasad inżynierii oprogramowania, jak również wyczerpujący zestaw narzędzi i technik do stosowania w projektach programistycznych. Ich zastosowanie sprawi, że gotowy system będzie łatwy do utrzymania, rozszerzalny, nie będzie zbyt kosztowny, a jego wykonanie będzie realne pod względem czasu i ryzyka. Koncepcje opisane w książce stanowią doskonały punkt wyjścia, gdyż pozwalają sporządzić dobry projekt systemu oprogramowania oraz dobry plan budowy tego systemu. Dopiero to umożliwi ukończenie pracy w wyznaczonym terminie i bez przekroczenia założonego budżetu, a wytworzony produkt będzie łatwy w utrzymaniu, możliwy do rozszerzania oraz wielokrotnego użycia.

W książce między innymi:

- koncepcja projektowania systemu i planowania projektu
- dekompozycja systemu, jego struktura i łączenie komponentów
- narzędzia i techniki potrzebne w planowaniu i projektowaniu
- mierzenie i wyliczanie ryzyka projektu i jego wariantów
- zaawansowane techniki projektowania systemów o wysokiej złożoności

JUVAL LÖWY, jest światowej klasy architektem oprogramowania. Specjalizuje się w projektowaniu systemów i planowaniu projektów. Jest uznawany za jednego z najlepszych w swojej dziedzinie, uczestniczył w tworzeniu takich produktów jak C#, WCF i związanych z nimi technologii. Wydał kilka bestsellerowych książek oraz opublikował niezliczone artykuły poświęcone niemal wszystkim aspektom nowoczesnego tworzenia oprogramowania.

ISBN 978-83-283-6793-7
9 788328 367937
Cena: 89,00 zł

Pearson
Addison-Wesley

INFORMATYKA W NAJLEPSZYM WYDANIU