Michał **Śmiałek**, Kamil **Rybiński**

INŻYNIERIA OPROGRAMOWANIA W PRAKTYCE

Od wymagań do kodu z językiem UML

Helion

Redaktor prowadzący: Małgorzata Kulik

Projekt okładki: Studio Gravite/Olsztyn
Obarek, Pokoński, Pazdrijowski, Zaprucki

Materiały graficzne na okładce zostały wykorzystane za zgodą Shutterstock.

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *https://helion.pl* (księgarnia internetowa, katalog książek)

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

ISBN: 978-83-289-0051-6

Printed in Poland.

Spis treści

Rozdział 2.

Cykle wytwarzania oprogramowania

Celem procesu wytwarzania oprogramowania jest dostarczenie zamawiającemu spełniającego wymagania i sprawnego systemu oprogramowania. Z góry narzucone ograniczenia (np. czasowe, finansowe) oraz stopień złożoności problemu i technik jego produkcji wymuszają przedsięwzięcie odpowiednich czynności oraz właściwą ich organizację w planie realizacji projektu. Cechą każdego projektu mającego na celu stworzenie produktu jest ukierunkowanie wszystkich zadań objętych procesem wytwórczym na realizację wspólnego celu. W przypadku projektu informatycznego wspólnym celem jest wytworzenie systemu oprogramowania.

Jak pokazaliśmy w poprzednim rozdziale, zakończenie projektu informatycznego sukcesem nie jest łatwym zadaniem. Różne stopnie skomplikowania problemów, często zmieniające się środowisko i wymagania na system oraz różne zasoby wymuszają indywidualne podejście do każdego projektu. Wytwarzanie oprogramowania można jednak przedstawić jako powtarzalny cykl rozwiązywania poszczególnych problemów, prac technicznych i łączenia rozwiązań. Na podstawie doświadczeń zebranych na przestrzeni lat opracowano wiele modeli procesów wytwórczych oprogramowania, które przedstawiają różne cykle wytwarzania oprogramowania.

W niniejszym rozdziale zostanie przedstawiony zestaw typowych i powtarzalnych zadań, podzielonych na dyscypliny procesu wytwarzania oprogramowania. Ujęcie ich w trzy główne cykle wytwórcze pozwoli zrozumieć potrzebę uporządkowania czynności związanych z wytwarzaniem oprogramowania.

2.1. Dyscypliny cyklu wytwarzania oprogramowania

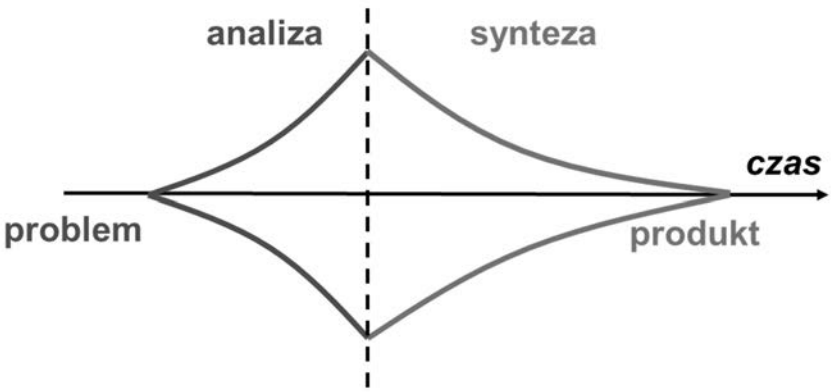
W procesie wytwarzania oprogramowania powstaje przede wszystkim system oprogramowania. Oprócz tego w trakcie projektu konieczne jest wytworzenie innych produktów, takich jak instrukcja użytkownika, specyfikacja wymagań, dokumentacja projektowa itd. Takie produkty uzupełniające (zwane także pośrednimi) powstają w wyniku różnych działań, które możemy podzielić na grupy nazywane dyscyplinami. **Dyscyplina inżynierii wymagań** stanowi spójny zestaw czynności, które są powiązane z zasadniczym obszarem działania w ramach projektu konstrukcji oprogramowania. Podział na dyscypliny jest realizacją zasady dekompozycji problemu na mniejsze składniki, co ułatwia organizację pracy w projekcie.

Jak dla każdej działalności inżynierskiej, czynności w inżynierii oprogramowania można podzielić na dwie grupy: czynności analizy i czynności syntezy. Taki podział jest naturalną dla człowieka metodą radzenia sobie ze skomplikowanymi zadaniami

twórczymi: analiza polega na dokładnym zidentyfikowaniu i zrozumieniu problemu, którego rozwiązanie osiąga się poprzez syntezę, czyli realizację i scalanie mniejszych części.

Podążając za naturalną dla człowieka ścieżką analizy i syntezy (patrz rysunek 2.1), od problemu do jego rozwiązania, proces wytwarzania oprogramowania definiuje się jako ciąg czynności podzielonych na dyscypliny, które prowadzą od postawienia problemu do wytworzenia produktu głównego (systemu oprogramowania) i produktów pośrednich. Do czynności analitycznych należą: opisanie środowiska, specyfikowanie wymagań, natomiast do czynności syntetycznych należą projektowanie, implementacja i wdrożenie. Rysunek 2.2 obrazuje wzajemną zależność czynności analitycznych i syntetycznych. Produkty dyscyplin analitycznych, które prowadzą od ogółu do szczegółu, są podstawą do rozpoczęcia prac syntetycznych, które poprzez realizację i scalanie mniejszych części prowadzą do wytworzenia spójnego produktu.

RYSUNEK 2.1. Analiza i synteza — naturalny sposób rozwiązywania złożonych problemów

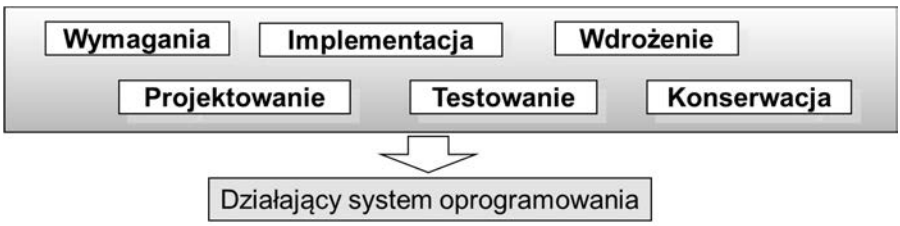


RYSUNEK 2.2. Analiza i synteza w procesie wytwarzania oprogramowania

Ogólne określenie procesu wytwarzania oprogramowania jako sekwencji analizy i syntezy wyznacza jedynie ścieżkę od problemu do produktu. Praktyczne zastosowanie takiego podejścia inżynierskiego prowadzi do różnych rozwiązań w zakresie zdefiniowania i uporządkowania konkretnych zadań. Duża różnorodność projektów informatycznych wymaga bowiem dostosowywania kształtu procesu do indywidualnych potrzeb. W następnej sekcji niniejszego rozdziału zostaną przedstawione główne typy cykli wytwórczych w inżynierii oprogramowania. Najpierw jednak przedstawimy uporządkowanie zadań w ramach różnych cykli życia. Metodą na takie uporządkowanie jest podział na dyscypliny inżynierii oprogramowania. Należy

przy tym zwrócić uwagę na to, że podział na dyscypliny nie oznacza podziału na etapy, czyli nie decyduje o uporządkowaniu czasowym wykonywania zadań w poszczególnych dyscyplinach. Takie uporządkowanie jest głównym elementem definicji cyklu życia.

Niezależnie od wybranego typu cyklu życia każdy projekt wytwarzania oprogramowania obejmuje kilka dyscyplin. Każda z dyscyplin dostarcza przynajmniej jeden produkt (pośredni). Produkty pośrednie są punktami wyjścia dla czynności wykonywanych z tej lub innych dyscyplin. Na rysunku 2.3 przedstawiono główne dyscypliny procesu wytwarzania oprogramowania w kolejności zgodnej ze ścieżką „analiza – synteza".



RYSUNEK 2.3. Dyscypliny cyklu wytwarzania oprogramowania

Dyscyplina wymagań dostarcza informacji o kształcie budowanego systemu z punktu widzenia klienta (np. użytkowników). Jest to określenie dość ogólne, albowiem na kształt systemu ma wpływ wiele czynników. Główne z nich to:

- środowisko, w którym system będzie funkcjonował,
- zadania, które system będzie realizował,
- sposób, w jaki system będzie funkcjonował.

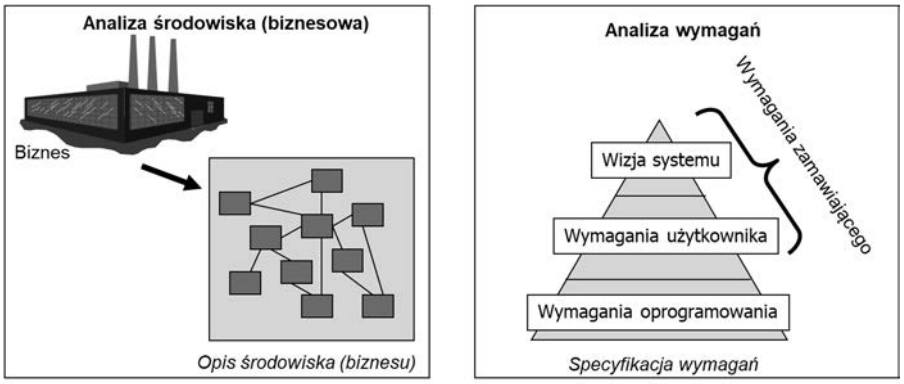
Główne produkty dyscypliny wymagań to opis środowiska (np. środowiska biznesowego) i specyfikacja wymagań. Na podstawie specyfikacji wymagań dyscyplina projektowania wytwarza zbiór modeli projektowych (architektura, projekt bazy danych, projekty szczegółowe komponentów, projekty dynamiki działania komponentów). Dyscyplina implementacji realizuje założenia określone w modelach projektowych. Powstaje działający system, który jest gotowy do kontroli jakości w ramach dyscypliny testowania. Pomyślne przejście testów umożliwia wdrożenie systemu w środowisku produkcyjnym (dyscyplina wdrożenia), co obejmuje również m.in. przeszkolenie użytkowników i przygotowanie dokumentacji technicznej. Ostatecznym produktem jest działający system informatyczny, który powinien podlegać ciągłemu utrzymaniu w ramach dyscypliny konserwacji.

Poniżej omawiamy najważniejsze cechy poszczególnych dyscyplin. Szczegóły dotyczące każdej z nich znajdują się w pozostałych rozdziałach tego podręcznika. Zauważmy, że poniższe opisy są rozwinięciem opisów dyscyplin z rozdziału 1. W tym rozdziale koncentrujemy się na podstawowych produktach pracy tworzonych w poszczególnych dyscyplinach. Zauważmy również, że rozszerzyliśmy zestaw dyscyplin o dwie — wdrożenie oraz konserwację. Dyscyplina wdrożenia jest rozszerzeniem dyscypliny implementacji o czynności związane z bezpośrednim przekazaniem systemu

do użytkowania. Dyscyplina konserwacji stanowi *de facto* kontynuację całego procesu wytwarzania oprogramowania już po zakończeniu właściwego projektu konstrukcji systemu. Z drugiej strony poniższy opis koncentruje się na czynnościach technicznych. Pominęliśmy zatem tutaj dyscypliny nadzoru (zarządzania) oraz środowiska. Dyscypliny te pojawią się natomiast podczas omawiania metodyk w rozdziale 3.

Dyscyplina wymagań

W ramach dyscypliny wymagań powinniśmy w pierwszej kolejności zidentyfikować problemy, które może rozwiązać system oprogramowania, a także podjąć decyzję o budowie takiego systemu. Podstawą określenia potrzeb klienta wynikających z zadanego problemu jest specyfikacja wymagań. Głównym celem specyfikacji wymagań jest określenie zakresu i kształtu przyszłego systemu, który będzie realizował potrzeby klienta. Rysunek 2.4 przedstawia ogólny podział i strukturę dyscypliny wymagań. W ramach proponowanego przez nas podziału dyscyplina ta dzieli się na dwie zasadnicze dyscypliny składowe: analizę środowiska (np. biznesu) oraz analizę wymagań. Warto tutaj jednak podkreślić, że w niektórych metodykach te dwa składniki są traktowane jako osobne dyscypliny główne.



RYSUNEK 2.4. Główne elementy dyscypliny wymagań

W ramach analizy środowiska opisujemy aktualny oraz docelowy stan środowiska (biznesu). Wykonywane jest to przy współpracy przedstawicieli klienta (osób zamawiających system lub ich reprezentantów). Na tej podstawie definiuje się główne potrzeby użytkowników systemu, które po uwzględnieniu ograniczeń środowiskowych tworzą wymagania zamawiającego. Poprawnie zdefiniowane wymagania zamawiającego zwykle są podstawą do zawarcia kontraktu między zamawiającym i wykonawcą na dostarczenie produktu spełniającego mieszczące się w tym dokumencie wymagania. Kolejną częścią specyfikacji wymagań jest opis wymagań oprogramowania, przedstawiający szczegóły działania systemu (m.in. szczegółowy opis interakcji systemu z użytkownikiem, wygląd okien).

Opis środowiska (biznesu) stanowi pierwszy zasadniczy produkt dyscypliny wymagań. Jest on opisem fragmentu świata, który jest obszarem działań danej organizacji wraz ze środowiskiem, w jakim ta organizacja (biznes) działa. Formułowanie takiego

opisu polega na stworzeniu precyzyjnego i zrozumiałego modelu, który zawiera wszystkie elementy środowiska (biznesowego) oraz zasady oddziaływania ich na siebie. Taki model składa się zazwyczaj z setek lub nawet tysięcy różnych elementów oddziałujących na siebie w określony, złożony sposób. Podstawowe elementy organizacji biznesowej to współpracownicy, pracownicy, jednostki organizacyjne, produkty, surowce, podzespoły, dokumenty, systemy informatyczne. Jednym z tych elementów jest budowany system informatyczny. Modelowanie może przebiegać na różnym poziomie abstrakcji: możemy modelować całą firmę lub tylko jakiś jej fragment (np. dział czy placówkę). Model może opisywać tylko stan docelowy (po zbudowaniu systemu informatycznego) lub być uzupełniony o opis stanu aktualnego.

Opis środowiska zawiera zwykle słownik pojęć (biznesowych) reprezentujący strukturę środowiska (biznesowego) oraz składniki procesów (biznesowych). Procesy opisywane są z wykorzystaniem pojęć ze słownika pojęć. Procesy biznesowe są serią powiązanych ze sobą działań, które prowadzą do osiągnięcia celu biznesowego. Dokładna ich analiza dostarcza informacji na temat rzeczywistych potrzeb użytkowników, które są podstawą do zdefiniowania tzw. wymagań funkcjonalnych. Analiza środowiska, w którym realizowane są procesy biznesowe, dostarcza informacji na temat uwarunkowań środowiskowych, które są podstawą do sformułowania tzw. wymagań jakościowych (inaczej: pozafunkcjonalnych) na budowany system.

Kompletna **specyfikacja wymagań** wynika bezpośrednio z opisu biznesu i można ją porównać do piramidy (patrz ponownie rysunek 2.4). Na samym szczycie znajduje się wizja systemu, która dostarcza informacji na temat ogólnych cech systemu w ścisłym powiązaniu z potrzebami biznesowymi klienta. Zakres systemu wyznaczają wymagania użytkownika w postaci uporządkowanego zbioru cech i funkcji systemu, które powinny być podstawą do zawarcia kontraktu. Wizja systemu i wymagania użytkownika tworzą wymagania zamawiającego, które są odzwierciedleniem rzeczywistych potrzeb zdefiniowanych przez biznes. Na najniższym poziomie piramidy są wymagania oprogramowania, które opisują szczegóły funkcjonalne komunikacji między użytkownikami i systemem, wszystkie wymieniane między nimi dane oraz planowany wygląd systemu.

Specyfikacja wymagań, obok ogólnych cech systemu, zawiera opis jego dynamiki oraz struktury. Słownik pojęć (biznesowych) jest podstawą do stworzenia słownika dziedziny. Słownik dziedziny jest rozszerzany w trakcie opisywania cech systemu, wymagań funkcjonalnych i jakościowych. Słownik przyjmuje ostateczny kształt na koniec tworzenia specyfikacji wymagań. Zapewnia spójność całej specyfikacji wymagań i jest istotnym jej elementem. Na podstawie takiej spójnej i kompletnej specyfikacji wymagań można zbudować system informatyczny, będący naturalnym fragmentem rzeczywistości przedstawionej w opisie biznesu i realizujący rzeczywiste potrzeby zamawiającego.

Dyscyplina projektowania

W ramach dyscypliny projektowania dokonujemy syntezy problemu opisanego w ramach dyscypliny wymagań poprzez opracowanie „planów projektowych" dla systemu. Jako produkt bazowy przyjmujemy zatem specyfikację wymagań i opis środowiska

biznesu. Celem działań tej dyscypliny jest stworzenie modeli projektowych na różnych poziomach abstrakcji, na podstawie których zostanie zaimplementowany system. Projektowanie często zaczynamy od najbardziej ogólnych (abstrakcyjnych) modeli, a potem — w zależności od złożoności systemu — przechodzimy do bardziej szczegółowych poziomów (tzw. podejście *top-down*). Możliwe jest również podejście odwrotne, w którym zaczynamy od szczegółów, a dopiero potem syntetyzujemy widok ogólny systemu (tzw. podejście *bottom-up*).

Mając do dyspozycji opis środowiska biznesu i specyfikację wymagań, można zdefiniować, jakich elementów fizycznych (np. serwer bazy danych, serwer aplikacyjny, maszyny klienckie, urządzenia mobilne) będzie potrzebował budowany system i w jaki sposób te elementy będą się ze sobą komunikować. Jest to najbardziej ogólny, **fizyczny model architektoniczny**. Nie definiuje on logiki działania systemu, która definiowana jest na poziomie **architektury logicznej**. Na tym poziomie definiuje się moduły wykonawcze (komponenty), z jakich będzie się składał budowany system, oraz powiązania komunikacyjne (np. interfejsy) między tymi modułami. W często stosowanych technologiach opartych na usługach lub mikrousługach (ang. *service, microservice*) komponenty w ramach logicznego modelu architektonicznego reprezentują poszczególne usługi. Interfejsy, które są specyfikacją funkcji udostępnianych przez komponenty architektoniczne, pozwalają traktować moduły jako czarne skrzynki. Na tym poziomie abstrakcji nie skupiamy się na tym, „co jest w środku" komponentów. Skupiamy się na tym, jakie funkcje komponenty spełniają.

Istotnym elementem modelu architektonicznego i zależnych od niego modeli szczegółowych są struktury danych, które służą do komunikacji poprzez interfejsy. Ich podstawą może być słownik dziedziny, zdefiniowany w specyfikacji wymagań. Na podstawie słownika możemy zdefiniować tzw. **obiekty transferu danych** (ang. *data transfer object*). Określają one „paczki danych", które mogą być przesyłane między poszczególnymi komponentami systemu.

Kolejnym poziomem projektowania jest stworzenie **szczegółowych projektów modułów**. Na tym poziomie definiujemy konkretne szczegóły „czarnych skrzynek". Dokładnie przedstawiamy strukturę oraz opracowujemy sekwencje wykonywanych operacji, których celem jest realizacja funkcji określonych w poszczególnych interfejsach komponentów. W zależności od złożoności systemu może być także wymagane zaprojektowanie algorytmów, które realizują złożone funkcjonalności wewnątrz systemu (np. algorytmy szyfrujące, obliczające składki ubezpieczenia czy wykonujące obliczenia na macierzach).

Produktem projektowania jest zbiór spójnych modeli dostarczających szczegółowych informacji o składnikach systemu, który ma spełniać wymagania opisane w specyfikacji wymagań i funkcjonować w opisanym środowisku. Modele projektowe bezpośrednio odzwierciedlają konstrukcje programistyczne i są bezpośrednio wykorzystywane podczas kodowania systemu, czyli jego implementacji.

Dyscyplina implementacji

W wyniku działań objętych dyscypliną implementacji zostaje wytworzony **kod (program)** realizujący wymagania zdefiniowane podczas analizy i spełniający założenia projektowe, określone w modelach projektowych. Do wytworzenia działającego systemu wykorzystuje się zbiór technologii, które są zgodne z wymaganiami technicznymi i wpisują się w środowisko działania budowanego systemu. Technologie te zostały również określone podczas projektowania komponentów systemu.

Proces programowania wymaga odpowiedniego środowiska i organizacji pracy. Aby stworzyć system, który będzie prawidłowo funkcjonował w **środowisku produkcyjnym** klienta, należy takie środowisko zbudować również w wersji testowej (implementacyjnej). Czynności z tym związane mogą obejmować na przykład instalację serwera i potrzebnych bibliotek w wersjach wykorzystywanych przez organizację, uruchomienie dedykowanych rozwiązań lub tzw. zaślepek potrzebnych do funkcjonowania systemu. Duży wpływ na kształt **środowiska implementacyjnego** mają wymagania jakościowe i ograniczenia środowiska biznesowego.

Implementacja systemu oznacza implementację poszczególnych modułów (komponentów) zaprojektowanych w ramach dyscypliny projektowania. Liczba modułów do zaimplementowania jest oczywiście zależna od stopnia skomplikowania systemu. Każdy moduł może być realizowany niezależnie od pozostałych (przez różnych programistów), ale powinien poprawnie z nimi współdziałać. W szczególności w architekturach komponentowych zadaniem programistów jest zaprogramowanie realizacji interfejsów dostarczanych przez komponenty, zgodnie z założeniami projektowymi.

Każdy zespół programistów realizujących pewną część systemu (zbiór modułów) jest odpowiedzialny za jej prawidłowe funkcjonowanie. Dlatego też nawet najmniejsze „kawałki" kodu powinny być na bieżąco testowane podczas całego procesu implementacji. Testowaniu podlegają poszczególne elementy kodu (klasy, funkcje), a także moduły (zestawy klas) oraz ich interfejsy (zestawy funkcji). Programiści tworzą tzw. **testy jednostkowe**. Są to zestawy procedur, których celem jest sprawdzenie działania tworzonego kodu dla różnych sytuacji (dla różnych danych wejściowych). Zauważmy tutaj, że testy jednostkowe są traktowane jako element implementacji systemu, a nie jego walidacji (patrz niżej).

Prace w ramach dyscypliny implementacji wymagają zastosowania odpowiednich narzędzi, tworzących tzw. zintegrowane środowisko deweloperskie (ang. *Integrated Development Environment* — IDE). Podstawą jest oczywiście dobre narzędzie służące do kompilacji kodu, jego wykonywania w środowisku implementacyjnym (testowym) oraz do przeprowadzania testów jednostkowych. Inne narzędzia wspomagają pracę grupową, kontrolę wersji kodu oraz integrację całego systemu. Niezależnie jednak od stosowania dobrych narzędzi gwarancją wyprodukowania poprawnego i optymalnego kodu jest wykorzystanie dobrej znajomości technologii i dobrych praktyk programistycznych. Pożądaną cechą kodów źródłowych, oprócz oczywistej zgodności z modelami projektowymi i wymaganiami, powinny być: wspólna konwencja programistyczna, dobra dokumentacja i gwarancja poprawności działania potwierdzona dokumentacją testów jednostkowych.

Po implementacji sprawnie działających (przetestowanych) modułów i ich połączeniu całość powinna przejść **testy integracyjne**. Poprawność działania zostaje sprawdzona globalnie, dla całego systemu. Tak sprawdzony kod systemu jest głównym produktem dyscypliny implementacji. Pozostałe produkty to dokumentacja testów jednostkowych i integracyjnych oraz na przykład pliki wykonywalne i instalacyjne.

Dyscyplina walidacji (testowania)

Celem testowania w ramach osobnej dyscypliny walidacji jest sprawdzenie, czy zbudowany system oprogramowania spełnia wymagania określone w specyfikacji wymagań i działa poprawnie w środowisku biznesowym. Jak już zauważyliśmy wyżej, czynności testowe wykonywane są również w ramach dyscypliny implementacji. Tutaj natomiast następuje ostateczna kontrola jakości.

Testy, niezależnie od ich poziomu, najczęściej wykonuje się w specjalnie przygotowanym środowisku testowym. W takim środowisku system uruchamiany jest w sposób bardzo zbliżony do tego, jak będzie uruchamiany docelowo w tzw. środowisku produkcyjnym. W odróżnieniu od testów przeprowadzonych w czasie implementacji, głównymi odbiorcami testów w ramach walidacji są przedstawiciele zamawiającego. Zwykle są to przyszli użytkownicy systemu, wytypowani i specjalnie przygotowani do nadzorowania i wykonywania czynności związanych z testowaniem. Poza sprawdzeniem poprawności integracji, które dokonywane jest przez osoby odpowiedzialne za infrastrukturę informatyczną organizacji, sprawdzana jest również zgodność z wymaganiami zamawiającego oraz szczegółowymi wymaganiami oprogramowania. Ogół tego typu testów nazywamy **testami akceptacyjnymi** (ang. *acceptance test*), gdyż są one warunkiem akceptacji systemu przez klienta.

Przeprowadzenie testów akceptacyjnych systemu wymaga **planu testów**. Zazwyczaj plan testów jest dokumentem powstałym na podstawie specyfikacji wymagań. Poszczególne testy ułożone są w odpowiedniej kolejności, która pozwala na właściwe sprawdzenie prawidłowości działania poszczególnych funkcjonalności systemu. Testy takie przypominają instrukcje postępowania przygotowane dla użytkowników systemu. Rezultatem jest powstanie kompletnych **scenariuszy testów**. Efektem wykonania konkretnego scenariusza testów jest potwierdzenie poprawności działania lub opis błędu. W przypadku niepomyślnego wyniku testów może nastąpić powrót do działań w ramach innych dyscyplin (projektowanie, implementacja), a potem ponowienie testów po zaimplementowaniu poprawek.

Wynikiem dyscypliny testowania jest dokumentacja testów akceptacyjnych, która jest podstawą do zatwierdzenia systemu i przejścia do czynności objętych dyscypliną wdrożenia.

Dyscyplina wdrożenia

Wdrożenie systemu polega zasadniczo na przeniesieniu systemu do warunków środowiska produkcyjnego i przekazaniu go do właściwego użytkowania. Następuje to po pomyślnym przejściu testów akceptacyjnych i zatwierdzeniu uruchomienia aktualnej

wersji budowanego systemu. Dyscyplina wdrożenia obejmuje czynności związane z instalacją systemu oraz umożliwieniem korzystania z systemu przez jego użytkowników. Instalacja systemu obejmuje zarówno przygotowanie środowiska produkcyjnego do wdrożenia nowego systemu, jak i samo fizyczne umieszczenie plików wykonywalnych systemu w środowisku docelowym. Głównym produktem dyscypliny wdrożenia jest **zainstalowany system**, działający w docelowym środowisku wykonawczym. Innymi produktami są na przykład **dokumentacja dla użytkowników** oraz **dokumentacja techniczna**. Strategie działań w ramach dyscypliny wdrożenia zależą od rodzaju budowanego systemu. W dalszym opisie koncentrujemy się na najbardziej typowej sytuacji, tzn. budowie systemu na zamówienie dla klienta (np. dla organizacji biznesowej).

Można wyróżnić kilka najbardziej popularnych strategii wdrożenia nowego systemu do środowiska produkcyjnego:

- Wdrożenie bezpośrednie — nowy system zostaje wdrożony od razu w całości. Jest to strategia niosąca często spore ryzyko. Konieczne jest zapewnienie wysokiej jakości nowego systemu oprogramowania, gdyż usterki wykrywane w trakcie działania mogą spowodować duże problemy w organizacji klienta. W niektórych przypadkach, jak wdrożenie pierwszego systemu oprogramowania w danym obszarze czy też wymiana oprogramowania wbudowanego, taka strategia jest jedyną możliwą do zastosowania.
- Wdrożenie równoległe — obok nowego systemu przez pewien czas działa również stary. Zaletą tej strategii jest weryfikacja poprawności działania nowego systemu i możliwość wykorzystania starego w przypadku wykrycia błędów w nowym systemie lub jego awarii. Wadą jest natomiast duży koszt utrzymania i obsługi obu systemów.
- Wdrożenie pilotażowe — w pierwszej fazie fragment nowego systemu lub cały system zostaje uruchomiony w jednej z jednostek organizacyjnych. Dopiero po sprawdzeniu poprawności tego wdrożenia system zostaje wdrożony w pozostałych jednostkach. Takie podejście pozwala zminimalizować ryzyko związane z wdrożeniem bezpośrednim, a jednocześnie zredukować koszty utrzymania dwóch systemów.
- Wdrożenie stopniowe — strategia polegająca na wprowadzaniu w działalność organizacji kolejnych fragmentów nowego systemu, uniezależniając pozostałe obszary działalności od nowego systemu. Pozwala to zminimalizować ryzyka związane z zatrzymaniem działalności całej organizacji przez uruchomienie nowego systemu w całości. Wadą jest natomiast rozległe w czasie wdrożenie.

Warto zauważyć, że często wdraża się system według strategii hybrydowej, łączącej wybrane cechy powyższych strategii. Dzięki temu możliwe jest dostosowanie czynności wdrożeniowych do specyfiki danej organizacji i wdrażanego systemu oprogramowania.

Istotnym elementem dyscypliny wdrożenia jest przygotowanie materiałów dla użytkowników oraz przeprowadzenie **szkoleń** dla przyszłych użytkowników. Forma materiałów oraz szkoleń jest uzależniona od wybranej strategii wdrożenia oraz rodzaju zbudowanego systemu. Podstawowym celem szkolenia jest nauka zasad działania i operowania systemem. Może to być realizowane poprzez dostarczenie

użytkownikom podręczników albo poprzez wbudowane w system materiały szkoleniowe (np. pomocniki kontekstowe). Szkolenia mogą być prowadzone na zasadzie bezpośredniego mentoringu w środowisku produkcyjnym lub być wykonywane na bazie przygotowanych prezentacji szkoleniowych oraz pracy w osobnym środowisku szkoleniowym.

Efektem pomyślnego wdrożenia systemu jest działający system, sprawnie używany przez jego użytkowników. Wdrożenie powinno również dostarczyć ocenę zbudowanego i wdrożonego systemu (ewentualnie jego wersji pośredniej). Jest to podstawą do dalszych działań w ramach obecnego projektu (kolejne wersje przyrostowe), w następnych projektach (rozbudowa systemu w przyszłości) oraz w ramach dyscypliny konserwacji.

Dyscyplina konserwacji

Czynności konserwacji, zwane także czynnościami utrzymania, stanowią zawsze ostatni etap w cyklu życia oprogramowania. Obejmują one zapewnianie poprawnego funkcjonowania systemu w środowisku biznesowym. W trakcie pracy z systemem oprogramowania mogą zostać wykryte wcześniej nieznane defekty. Wymagana funkcjonalność lub cechy jakościowe systemu oprogramowania, które zostały zdefiniowane w ramach analizy wymagań, mogą ulec zmianie już po oddaniu systemu do użytku. Mogą się także zmienić samo środowisko i procesy biznesowe (np. zmiany organizacyjne w firmie, wdrożenie zmian w innym systemie, zmiana otoczenia prawnego). Wszystkie te sytuacje wymuszają podjęcie pewnych działań, które można podzielić na cztery grupy:

- czynności naprawcze obejmujące usuwanie defektów oprogramowania;
- czynności adaptacyjne dostosowujące oprogramowanie do zmieniającego się środowiska;
- czynności ulepszające wprowadzające nowe funkcjonalności i zmianę już istniejących;
- czynności prewencyjne przygotowujące oprogramowanie do przyszłych modyfikacji.

W niektórych przypadkach wprowadzenie zmian w funkcjonującym systemie może wymagać przejścia przez wszystkie dyscypliny, od wymagań do implementacji i wdrożenia. Wszelkie działania konserwacyjne powinny być starannie dokumentowane. Pozwala to unikać błędów we wprowadzaniu zmian i czyni system oprogramowania przejrzystym.

Podjęcie wyżej wymienionych czynności powinna poprzedzić analiza kosztów i ryzyka wprowadzania zmian w funkcjonującym systemie. W wyniku takiej analizy może się okazać, że bardziej opłacalne od utrzymania bieżącego systemu będzie wytworzenie nowego systemu. W takiej sytuacji kończy się cykl życia oprogramowania, co prowadzi do rozpoczęcia kolejnego cyklu budowy i życia oprogramowania.

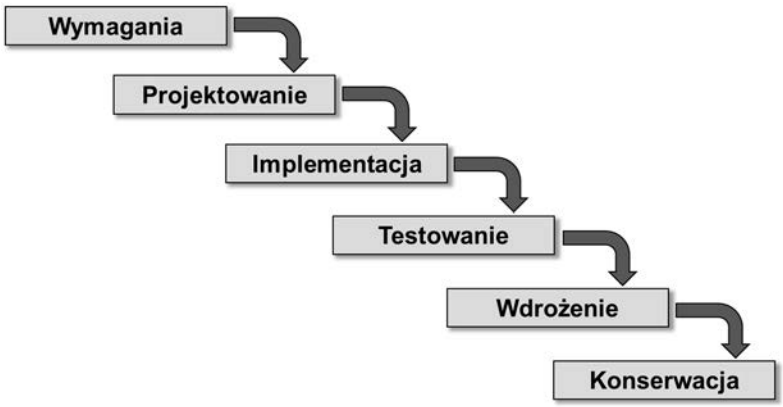
2.2. Przegląd cykli wytwarzania oprogramowania

Przedstawione wyżej dyscypliny określają zakres czynności i produktów wykonywanych w typowym projekcie wytwarzania oprogramowania. Czynności te tworzą pewien ciąg, który powinien być wykonywany w określonej kolejności. Taki ciąg czynności nazywamy procesem wytwarzania oprogramowania. W dobrze zorganizowanym projekcie proces ten przebiega zgodnie z jasno określonym cyklem wytwórczym, który jest elementem całego cyklu życia oprogramowania. Cykl wytwórczy definiuje rozłożenie czynności z poszczególnych dyscyplin w czasie.

W większości współczesnych projektów stosuje się jeden z dwóch cykli wytwórczych: cykl wodospadowy (również nazywany kaskadowym) lub cykl iteracyjny. Dodatkowo rozwinięciem cyklu iteracyjnego jest cykl spiralny. Cykle wytwórcze mają warianty, charakterystyczne dla różnych metodyk wytwórczych, które zostały opisane w kolejnym rozdziale. Z uwagi na różne uwarunkowania i przebieg projektów zazwyczaj nie udaje się zastosować ściśle do wszystkich reguł przyjętych w wybranym cyklu wytwórczym. Często eksponuje się pewne elementy kosztem innych, które są pomijane. Cykl wytwórczy jest rodzajem przewodnika wskazującego kierunek, który należy obrać, aby osiągnąć zamierzony cel, ale nie pokazuje drogi, którą należy pójść. Szczegóły tej drogi w szczegółach określa natomiast metodyka.

Cykl wodospadowy

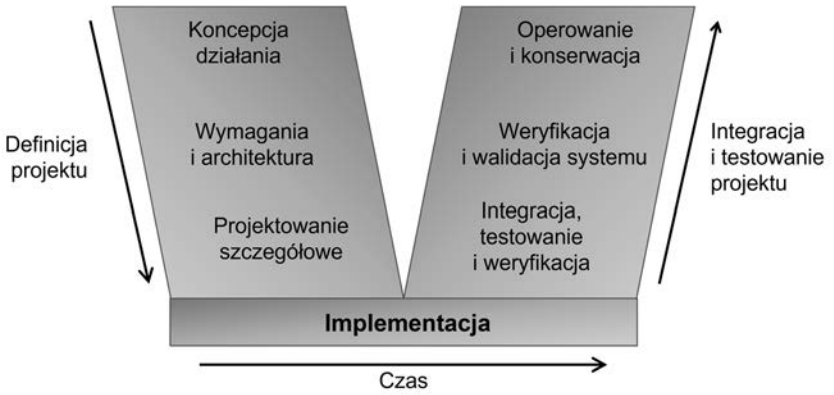
Koncepcja wodospadowego (ang. *waterfall*) cyklu życia oprogramowania, zwanego także kaskadowym lub klasycznym, została sformułowana już w 1970 roku. Główną ideą tego cyklu jest ujęcie dyscyplin wytwarzania oprogramowania w sekwencję dobrze wyodrębnionych kroków (faz lub etapów), które prowadzą do zbudowania systemu oprogramowania. Rysunek 2.5 przedstawia ogólny schemat cyklu wodospadowego. Kolejne etapy wykonywane są raz w trakcie trwania projektu i następują po sobie. Nazwa tego cyklu wynika z podobieństwa tego schematu do wodospadu lub kaskady.



RYSUNEK 2.5. Wodospadowy cykl wytwarzania oprogramowania

Jak wskazuje rysunek, poszczególne etapy „wodospadu" odpowiadają dyscyplinom cyklu życia oprogramowania przedstawionym w poprzedniej sekcji. Każdy etap dostarcza co najmniej jeden produkt, który podlega weryfikacji i akceptacji. Następny etap może się rozpocząć tylko po zakończeniu poprzedniego i wykorzystuje jego produkty jako punkt wyjścia. W praktyce jednak sąsiadujące ze sobą etapy zazębiają się i przekazują nawzajem informacje. Oznacza to, że w ramach jednego etapu możemy wrócić do poprzedniego etapu (tzw. wodospad z nawrotami). Można powiedzieć, że powtarzające się nawroty do poprzedniego etapu tworzą „miniiteracje" (patrz opis cyklu iteracyjnego).

Wariantem cyklu wodospadowego jest tzw. cykl „V". Jest on zilustrowany na rysunku 2.6. Cykl ten swoją nazwę zawdzięcza graficznemu ułożeniu faz projektu w kształt litery „V". Pierwsze trzy fazy obejmują czynności definiowania systemu, przy czym nieco inny jest podział dyscyplin. Na przykład dyscyplina wymagań rozbita jest między fazy „koncepcja działania" (odpowiednik analizy środowiska) oraz „wymagania i architektura". Po tych fazach następuje faza implementacji oraz trzy kolejne fazy obejmujące czynności integrowania i testowania systemu. Zauważmy, że trzy ostanie fazy odpowiadają trzem pierwszym fazom. Oznacza to, że fazy w drugiej części cyklu służą do realizacji oraz walidacji i weryfikacji rezultatów pierwszej części cyklu (w tym implementacji).



RYSUNEK 2.6. Cykl „V" dla wytwarzania oprogramowania

Cykl wodospadowy w naturalny sposób odzwierciedla proces rozwiązywania złożonych problemów, opisany wyżej (najpierw analiza, a później synteza). Wszystkie podejmowane czynności prowadzą od potrzeb użytkownika do gotowego systemu w jednym przebiegu przez poszczególne grupy czynności (dyscypliny). Umożliwia to łatwe zarządzanie zasobami (zespoły specjalistów, sprzęt). Przejście do kolejnych faz wymaga jednak zamrożenia wyników prac w fazach poprzednich. Wynika to z konieczności zachowania spójności produktów w ramach całego cyklu. Konieczność powrotu do poprzednich faz oznacza odejście od czystego cyklu wodospadowego.

Cykl wodospadowy stanowi istotny krok w kierunku uporządkowania działań dotyczących wytwarzania oprogramowania. Ma on niewątpliwe **zalety**, które powodują, że jest stosowany w wielu projektach. Podstawową zaletą jest prosta i naturalna struktura,

odpowiadająca naturalnemu procesowi rozwiązywania problemów. Porządkuje on wszystkie czynności i umożliwia łatwe śledzenie postępów prac. Postęp prac mierzony jest kolejnymi produktami zdefiniowanymi w poszczególnych etapach, odpowiadających dyscyplinom inżynierii oprogramowania. Zarządzanie projektem realizowanym w cyklu wodospadowym jest stosunkowo proste. Wynika to z tego, że w poszczególnych fazach projektu wykorzystujemy zasoby ludzkie przypisane do określonej dyscypliny (analityków wymagań, projektantów, programistów, testerów). Efekty prac są jasno określone i stanowią podstawę do rozpoczęcia działań w kolejnych fazach projektu.

Niestety, cykl wodospadowy ma **wady**, które rodzą zasadnicze zagrożenia. Podstawowym problemem jest późna weryfikacja wyników prac. Problemy z realizacją wymagań klienta odkrywane są zazwyczaj dopiero w fazie testowania, a nawet dopiero w fazie wdrożenia. Prowadzi to często do realizacji syndromu 90%/90%. Pod koniec projektu jego kierownictwo raportuje wykonanie 90% zadań. W trakcie testowania i wdrożenia okazuje się jednak, że do wykonania pozostaje nadal 90% prac, gdyż wiele funkcjonalności systemu nie jest zgodnych z oczekiwaniami klienta. Skutkuje to znacznym przesunięciem terminu oddania projektu. Efektem są podane w poprzednim rozdziale statystyki niepowodzeń projektów i średniego czasu przekroczenia ich terminów oraz budżetów. Zauważmy też, że istotnym czynnikiem jest tutaj subiektywizm oceny produktów. Na przykład jakość specyfikacji wymagań możemy w pełni ocenić dopiero w fazie implementacji. Często dopiero wtedy okazuje się, że programiści nie mogą wykonać pewnych elementów kodu, gdyż brakuje dokładnej specyfikacji dziedziny problemu. Oznacza to konieczność bardzo dalekiego nawrotu i wykonania dodatkowych czynności w ramach dyscypliny wymagań.

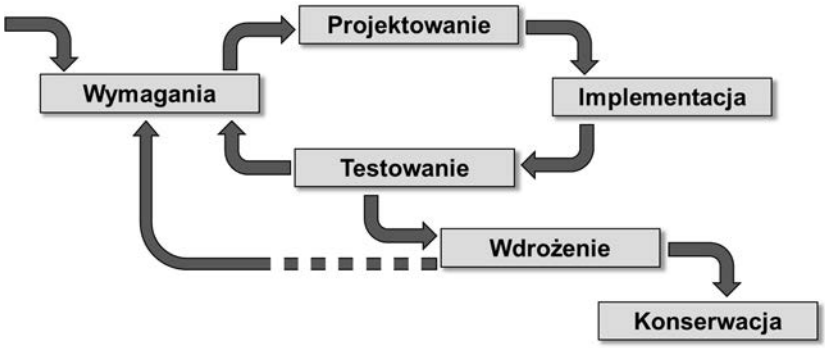
Inną wadą cyklu wodospadowego jest niewielka elastyczność w radzeniu sobie ze zmianami (w tym zmianami zakresu). Bardzo trudno jest stworzyć poprawną i kompletną specyfikację już na początku projektu. Rzeczywiste potrzeby klienta są często odkrywane lub zmieniane dopiero w trakcie trwania prac nad systemem oprogramowania. Szczególnie dotyczy to faz testowania i wdrożenia. To często wtedy klient zdaje sobie sprawę z tego, jakie skutki ma przyjęcie określonych założeń dotyczących funkcjonalności systemu lub jego cech jakościowych. Prowadzi to w rezultacie do dosyć chaotycznych działań w celu „wymuszenia" zgodności z oczekiwaniami, na przykład sporów o zakres projektu czy próby istotnych zmian warunków kontraktu. Każda jednak tego typu zmiana oznacza konieczność powrotu do początkowych faz projektu i ponowne przejście przez „wodospad".

Kolejnym problemem cyklu wodospadowego jest niewielka możliwość optymalizacji prac, w tym poprawy współdziałania między zespołami działającymi na różnych etapach (tu: dyscyplinach). Przekazanie produktów między etapami — w czystym procesie wodospadowym — następuje tylko raz. Oznacza to, że brakuje możliwości naprawy tych produktów w razie stwierdzenia usterek czy braków w kolejnych etapach. Oczywiście, taka naprawa jest zazwyczaj podejmowana, ale prowadzi to do działań wykraczających poza ustalony plan projektu. Oznacza to zatem często powstanie bałaganu w zarządzaniu projektem.

Wodospadowy cykl wytwarzania oprogramowania, mimo iż jest dosyć leciwy i ma wiele wad, wciąż jest popularny. Jak wynika z analizy jego wad, powodzenie projektu realizowanego cyklem wodospadowym mocno zależy od stabilności i poprawności specyfikacji wymagań. Dlatego jest polecany do krótkich projektów, w których wymagania są dobrze określone i zrozumiałe, a ich specyfikacja jest w pełni przygotowana. W przypadku dłuższych projektów powinniśmy być pewni, że problem jest bardzo dobrze określony i nie będzie podlegał modyfikacjom.

Cykl iteracyjny

Podstawą dla stworzenia cyklu iteracyjnego była chęć eliminacji zagrożeń związanych z „zamrażaniem" produktów poszczególnych faz cyklu wodospadowego. Zagrożenia takie możemy wyeliminować przez wprowadzenie wielokrotnego przechodzenia przez powiązane ze sobą dyscypliny. To podejście jest główną cechą iteracyjnego cyklu wytwarzania oprogramowania. Ilustruje to rysunek 2.7. Jak widzimy, cykl iteracyjny zachowuje podział na dyscypliny, ale czynności w ramach wszystkich dyscyplin są wykonywane wielokrotnie.



RYSUNEK 2.7. Iteracyjny cykl wytwarzania oprogramowania

W projekcie planowanych jest wiele iteracji, gdzie każda **iteracja** trwa zazwyczaj kilka tygodni (najczęściej od 2 do 6). Poszczególne iteracje rozpoczynają się od wyboru i wyspecyfikowania określonego zakresu wymagań, odpowiadającego przyrostowi funkcjonalności systemu. Potem następują czynności syntetyzowania problemu: projektowanie, implementacja oraz testowane. Czynności te dotyczą zakresu wymagań wybranych do danej iteracji oraz ewentualnie wymagań, które nie zostały prawidłowo zrealizowane w poprzednich iteracjach. Testowanie przeprowadza się w sposób regresyjny. Dokładnie testuje się funkcjonalność wykonaną w danej iteracji, a także przeprowadza się mniej dokładne testy funkcjonalności zrealizowanych wcześniej. Po zakończeniu testów i akceptacji ich wyników przechodzimy do kolejnej iteracji. W razie stwierdzenia błędów lub uwag od klienta odpowiednie zadania są przydzielane do kolejnych iteracji.

Po zakończeniu pewnej liczby iteracji możliwe jest wdrożenie systemu. W niektórych projektach wdrożenie wykonywane jest tylko raz — po ostatniej iteracji. Najczęściej jednak planuje się wdrożenia pośrednie. Takie podejście umożliwia

dostarczenie klientowi działającego (choć jeszcze nie w pełni funkcjonalnego) systemu w czasie trwania projektu. Po ostatnim wdrożeniu następuje faza konserwacji, podobnie jak w cyklu wodospadowym.

Zastosowanie cyklu iteracyjnego nie daje oczywiście gwarancji na końcowy sukces procesu wytwórczego. Zwiększa jednak szanse na jego powodzenie przy założeniu prawidłowego stosowania jego zasad. Przyczynia się do tego lepsze włączenie zamawiających i przyszłych użytkowników systemu w proces budowy oprogramowania. Dyscyplina wymagań angażuje użytkowników systemu w trakcie całego projektu. W dyscyplinach projektowania i implementacji tworzony jest system, który następnie jest udostępniony użytkownikom do testowania. Testowanie (często połączone z wdrożeniem) pozwala ocenić poprawność działania systemu, ale przede wszystkim zgodność z założeniami i rzeczywistymi potrzebami klienta. Wszelkie problemy odkrywane są po każdej iteracji. Jeżeli aktualny stan oprogramowania nie odpowiada rzeczywistym potrzebom lub funkcjonalność nie została w pełni zrealizowana, możemy w naturalny sposób — zgodny z założeniami cyklu — przejść do fazy wymagań, a następnie do faz projektowania, implementacji i testowania.

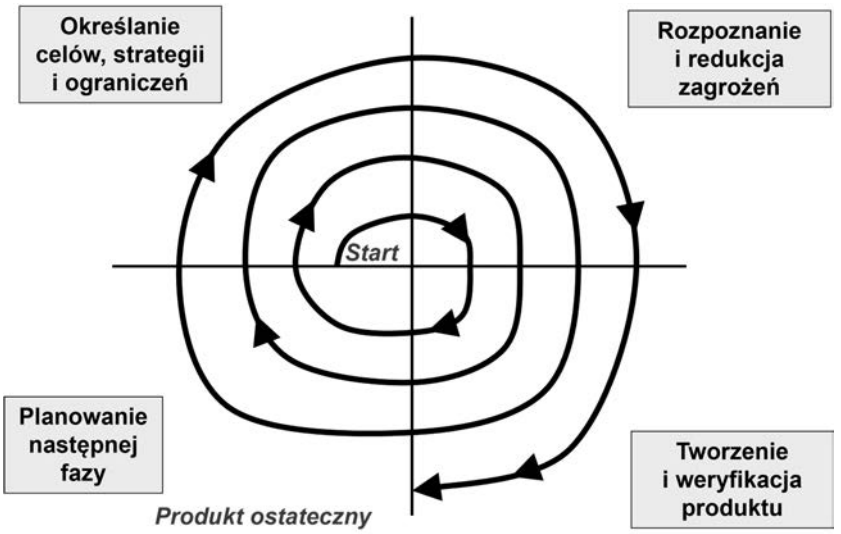
Cykl iteracyjny pozwala na przyrostowe tworzenie systemu oprogramowania. Daje to możliwość ustalenia priorytetów dla określonych funkcjonalności systemu oraz kolejności ich wykonania i wdrożenia. Uproszczeniu ulega także zarządzanie wprowadzaniem zmian w tworzonym oprogramowaniu, które odzwierciedlają zmieniające się ograniczenia środowiska i reguły biznesowe. Produkty poszczególnych faz nie muszą być zamrażane, dzięki czemu na koniec cyklu wytwarzania oprogramowania wszystkie specyfikacje i dokumentacje odpowiadają rzeczywistym warunkom.

Kosztem zmniejszenia ryzyka niepowodzenia procesu wytwarzania oprogramowania są bardziej skomplikowane czynności związane z organizacją i zarządzaniem zadaniami w projekcie. Zwykle niesie to ze sobą zwiększenie kosztów wykonania projektu i wydłuża czas od rozpoczęcia prac do ostatecznego ich zakończenia. Ten dodatkowy koszt można jednak porównać do kosztu ubezpieczenia projektu od różnych ryzyk, które występują w cyklu wodospadowym. Jest on zazwyczaj wielokrotnie niższy od kosztów związanych na przykład z koniecznością dostosowania prawie już gotowego systemu do wymagań klienta.

Iteracyjny cykl wytwarzania oprogramowania najlepiej sprawdza się w projektach budowy średnich i dużych systemów oprogramowania, gdzie nie wszystkie wymagania są dobrze zdefiniowane, a nawet rozpoznane.

Cykl spiralny

Istotnym rozszerzeniem cyklu iteracyjnego jest tzw. cykl spiralny. Główną przyczyną jego powstania była próba powiązania najlepszych cech wodospadowego i iteracyjnego cyklu wytwarzania oprogramowania oraz wzbogacenie go o rozbudowaną analizę ryzyka. Ogólny schemat cyklu spiralnego przedstawia rysunek 2.8**2.**. Pokazuje on ścieżkę dla procesu wytwarzania oprogramowania prowadzącą od postawienia problemu (punkt na wykresie na lewo od początku układu współrzędnych) do dostarczenia końcowego produktu. Odbywa się to poprzez realizację czterech zasadniczych etapów ułożonych w formę spirali.



RYSUNEK 2.8. Spiralny model wytwarzania oprogramowania

Każda pętla spirali jest podzielona na cztery części odpowiadające kolejnym ćwiartkom układu współrzędnych i może reprezentować dowolną fazę procesu wytwórczego. W tym cyklu nie ma stałych faz, takich jak analizowanie albo projektowanie. Cykl ten obejmuje inne procesy, przedstawione na modelu graficznym jako ćwiartki układu współrzędnych:

- Ustalanie celów — definiuje się konkretne cele dla danej fazy procesu wytwarzania oprogramowania wraz z ograniczeniami nałożonymi na produkty danej fazy.
- Rozpoznanie i redukcja zagrożeń — znalezione zagrożenia zostają poddane szczegółowej analizie, po czym podejmuje się kroki zmierzające do redukcji tych zagrożeń.
- Tworzenie i zatwierdzanie — na podstawie oceny zagrożeń zostaje wybrana metoda, która w największym stopniu minimalizuje ryzyko.
- Planowanie — produkt zostaje poddany ocenie i na tej podstawie planowane są następne czynności objęte kolejną pętlą spirali.

W jednej pętli, poświęconej wybranej fazie klasycznego procesu wytwórczego, można wykorzystać do wytworzenia spodziewanego produktu podejście kaskadowe, a w innej pętli — podejście iteracyjne. Na początku każdej pętli spirali wyznacza się cele, strategie i ograniczenia (pierwsza ćwiartka), po czym znajduje się różne sposoby ich osiągnięcia i realizacji (druga ćwiartka). Należy jednocześnie ocenić każdą opcję względem każdego celu. Dzięki tej ocenie możliwe jest oszacowanie źródeł zagrożeń przedsięwzięcia i wybór najlepszej opcji. Następne czynności polegają na wytworzeniu i weryfikacji następnego przybliżenia produktu (trzecia ćwiartka) konkretnej fazy. Następnie produkt podlega recenzji (czwarta ćwiartka) i na jej podstawie planowana jest następna pętla spirali. Jeżeli wynik recenzji jest pozytywny, następna pętla będzie dotyczyła kolejnej fazy wytworzenia oprogramowania. W przypadku negatywnego wyniku następna faza będzie dotyczyła tej samej fazy i opracowania następnego przybliżenia produktu.

Poprzez jawne potraktowanie zagrożeń cykl ten znacznie zmniejsza ryzyko niepowodzenia projektu. Należy jednak pamiętać, że cykl spiralny oparty o kontrolę ryzyka wymaga ciągłego dostosowywania kierowania projektem do warunków napotykanych po drodze od problemu do produktu końcowego. Ścisłe trzymanie się planu może uczynić wybranie spiralnego cyklu największym zagrożeniem dla projektu.

Zadania

Zadanie 1.

Zakładamy hipotetyczny duży projekt konstrukcji systemu oprogramowania dla konkretnej instytucji działającej w jednej z dziedzin (patrz sekcja 1.5). Zaproponuj cykl wytwarzania oprogramowania oraz uzasadnij swój wybór.

Zadanie 2.

Podaj przykłady projektów, w których korzystniejsze będzie zastosowanie cyklu wodospadowego lub iteracyjnego. Zwróć uwagę na dziedzinę zastosowania oraz rozmiar projektu. Uzasadnij swoją odpowiedź.

Słownik pojęć

Architektura fizyczna

Składnik opisu budowanego systemu oprogramowania przedstawiający elementy fizyczne (np. serwer bazy danych, serwer aplikacyjny, maszyny klienckie, urządzenia mobilne), na których będzie działał budowany system, oraz sposób komunikacji między tymi elementami.

Architektura logiczna

Składnik opisu budowanego systemu oprogramowania przedstawiający moduły wykonawcze (komponenty), z jakich będzie się składał budowany system, oraz powiązania komunikacyjne (np. interfejsy) między tymi modułami.

Dyscyplina inżynierii oprogramowania

Spójny zestaw czynności, które są powiązane z zasadniczym obszarem działania w ramach projektu konstrukcji oprogramowania.

Iteracja

Zestaw czynności, realizowanych w stosunkowo krótkim czasie (np. w ciągu dwóch tygodni), podczas którego wykonywany jest fragment systemu realizujący zadaną funkcjonalność. Iteracja kończy się wykonaniem działającego, zwalidowanego (przetestowanego) wydania systemu (wewnętrznego lub zewnętrznego).

Obiekt transferu danych

Struktura danych, która służy do wymiany informacji między komponentami w ramach architektury logicznej systemu.

Opis środowiska (biznesu)

Opis fragmentu świata (rzeczywistości), który jest obszarem działań danej organizacji wraz ze środowiskiem, w jakim ta organizacja działa. Formułowanie takiego opisu polega na stworzeniu precyzyjnego i zrozumiałego modelu, który zawiera wszystkie elementy danego fragmentu rzeczywistości (np. biznesowej) oraz zasady oddziaływania ich na siebie.

Specyfikacja wymagań

Specyfikacja (dokument, model) przedstawiająca potrzeby zamawiającego w stosunku do tworzonego systemu oprogramowania. W ramach tej specyfikacji formułowana jest ogólna wizja systemu oraz opis jego cech i funkcji na różnym poziomie szczegółowości.

Środowisko implementacyjne (testowe)

Wszystkie elementy fizyczne (sprzęt, sieci) oraz oprogramowanie (np. system operacyjny, bazy danych) zainstalowane w miejscu, gdzie system jest poddawany implementacji oraz testom przed jego przeniesieniem do środowiska produkcyjnego.

Środowisko produkcyjne

Wszystkie elementy fizyczne (sprzęt, sieci) oraz oprogramowanie (np. system operacyjny, bazy danych) zainstalowane w miejscu, gdzie system jest eksploatowany na co dzień przez jego użytkowników.

Test jednostkowy

Procedura testowa, której celem jest sprawdzenie działania tworzonego fragmentu kodu (np. klasy) dla różnych sytuacji (dla różnych danych wejściowych).

Test integracyjny

Procedura testowa, której celem jest sprawdzenie działania zestawu połączonych ze sobą (zintegrowanych) modułów tworzących system.

Test akceptacyjny

Procedura testowa, której celem jest sprawdzenie działania systemu z punktu widzenia jego akceptacji przez użytkowników.

Co trzeba zapamiętać

Produkty cyklu wytwarzania oprogramowania

Niezależnie od wybranego cyklu życia każdy projekt wytwarzania oprogramowania obejmuje kilka dyscyplin. Każda z dyscyplin dostarcza różne produkty, które są efektem czynności wykonywanych w jej ramach. Głównym produktem dyscypliny wymagań jest specyfikacja wymagań. W ramach dyscypliny projektowania powstają m.in. architektura fizyczna i architektura logiczna. Efektem dyscypliny implementacji jest przetestowany (jednostkowo i integracyjnie) kod systemu, a w ramach dyscypliny testowania przeprowadzane są testy akceptacyjne. Ostatecznym produktem wdrożenia systemu jest zainstalowany system w środowisku produkcyjnym.

Cykle wytwarzania oprogramowania

Aby zapanować nad złożonością procesu wytwórczego, czynności w ramach projektu porządkowane są w ramach cyklu wytwórczego. W przypadku niewielkich systemów, gdzie wszystkie wymagania są znane i dobrze udokumentowane, może sprawdzić się cykl wodospadowy. W przypadku bardziej rozbudowanych systemów, gdzie nie wszystkie wymagania są rozpoznane, powinien być wykorzystany cykl iteracyjny. Cykl spiralny łączy cechy cyklu wodospadowego i iteracyjnego.

Skorowidz

Ś

T

U

W

Z

Proces wytwarzania oprogramowania bezustannie ewoluuje. Coraz częściej stosowane są metodyki zwinne (agile), a dominującymi zasadami stają się między innymi iteracyjny cykl wytwórczy i ciągła integracja. Mimo to doświadczenia przemysłu i obiektywne badania wskazują na utrzymującą się od lat „chroniczną chorobę" przekroczonych budżetów, niedotrzymanych terminów i niezadowolonych klientów. Jako istotne przyczyny tego stanu można wskazać zaniechanie stosowania podstawowych zasad inżynierii i utożsamianie inżynierii oprogramowania z samym programowaniem. Programowanie koncentruje się przede wszystkim na pisaniu kodu programów na podstawie zadanych założeń. Inżynieria oprogramowania podchodzi do jego wytwarzania w sposób całościowy i kładzie szczególny nacisk na uzyskanie jak najwyższej jakości produktu software'owego.

Oto przystępny podręcznik wprowadzający w tajniki inżynierii oprogramowania. Kompleksowe ujęcie zagadnienia, od formułowania wymagań, poprzez projektowanie architektury i implementację, po testowanie i wdrożenie, a także skrupulatne omówienie różnorakich metodyk produkcji oprogramowania sprawiają, że to pozycja obowiązkowa dla każdego programisty, który ma ambicję być inżynierem oprogramowania. Jednocześnie książka jest wartościową lekturą dla wszystkich, którzy są zaangażowani w procesy związane z dostarczaniem na rynek programów komputerowych. Dotyczy to również procesów współpracy z klientem, takich jak analiza problemów biznesowych, planowanie i wdrożenie, wreszcie — taki dobór narzędzi i metodyk, aby możliwie jak najlepiej spełnić jego oczekiwania.

Dzięki książce:

- **poznasz najlepsze praktyki w zakresie produkcji wysokiej jakości oprogramowania**
- **zgłębisz inżynieryjne podejście do produkcji programów**
- **opanujesz podstawy specyfikowania wymagań i projektowania**
- **zaznajomisz się z różnymi metodykami wytwarzania oprogramowania**

Helion

helion.pl

HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl

KOD KORZYŚCI
Sięgnij po więcej!

ISBN 978-83-289-0051-6

9 788328 900516

Cena: 79,00 zł