23. Metodyki wytwarzania oprogramowania

**Metodyka** – zbiór zasad, sposobów wykonywania określonej pracy albo osiągnięcia określonego celu; szczegółowe normy postępowania właściwe danej nauce.

Opis metodyki zawiera:

* **Definicje ról (Rola)** (ang. role definition)

reprezentuje jednostkę lub zespół, która wykonuje określone aktywności i jest odpowiedzialna za pewne artefakty.

* **Definicje zadań (zadanie)** (ang. task definition)

opisuje czynności podejmowane przez rolę w celu wyprodukowania określonych (znaczących) rezultatów.

* **Definicje produktów pracy (Artefakt lub produkt pracy)** (ang. work product definition) jest wynikiem pracy pewnej roli w ramach określonego zadania. Może to być:

dokument

model

element modelu

kod źródłowy

Własności metodyk:

* **Zakres** (scope) metodyki – pokrycie cyklu życia, ról i aktywności
* **Rozmiar** metodyki (size) – liczba elementów kontrolnych w metodyce (wytwarzany artefakt, standard, aktywnośd, miara jakości, technika, itd.)
* **Dokładność** metodyki (ceremony) – wymagany stopień precyzji, nakładu pracy, tolerancja odstępstw od reguł
* **Waga** metodyki (weight) – iloczyn rozmiaru i dokładności dla poszczególnych elementów kontrolnych („iloczyn koncepcyjny”)
* **Widzialność** (visibility) – łatwość oceny przez zewnętrznego obserwatora, czy projekt jest wykonywany zgodnie z metodyką

Klasyfikacja metodyk ze względu na wagę:

* ciężkie (tradycyjne), np. USDP, RUP, Enterprise RUP
* lekkie, np. Scrum, XP, Crystal, Agile UP, Agile RUP

Klasyfikacja metodyk ze względu na formalizację opisu:

* nieformalne
* półformalne, zwykle oparte na notacjach graficznych, jak UML, np. USDP, RUP, XP
* formalne, np. VDM, oparte na językach formalnych, jak VDM-SL

Klasyfikacja metodyk ze względu na rodzaj procesów cyklu życia, które metodyka obejmuje:

* metodyki zarządcze, np. SCRUM, Prince 2, CMMI
* metodyki wytwórcze, np. RUP, USDP, XP

|  |  |
| --- | --- |
| **Metodyki wytwórcze** | **Metodyki zarządcze** |
| |  |  | | --- | --- | | Lekkie (Zwinne) | Ciężkie | | * XP * Agile RUP * Crystal | * USDP * RUP | | |  |  | | --- | --- | | Lekkie (Zwinne) | Ciężkie | | * SCRUM * FDD | * PRINCE2 * CMMI | |

Pożądane cechy metodyk

* Przewidywalność
* Wsparcie dla testowania i pielęgnacji
* Wsparcie dla zmian
* Wczesne wykrywanie i usuwanie błędów (Reguła 1-10-100 -> koszt naprawy błędu rośnie wykładniczo w zależności od fazy procesu, w której został wykryty).
* Możliwość zebrania informacji zwrotnej i poprawy procesu

# Zwinne metodyki

Wszystkie zwinne metodyki posiadają wspólne zasady określone przez manifest Agile.

**Manifest Agile:**

Należy przedkładać:

**Ludzi i ich wzajemne interakcje** (**współdziałanie**) ponad procedury i narzędzia.  
**Działające oprogramowanie** nad wyczerpującą dokumentację.  
**Współpracę z klientem** nad negocjację umów.  
**Reagowanie na zmiany** nad realizowanie planu.

**Programowanie zwinne (Agile software development)**

Grupa metodyk wytwarzania oprogramowania opartego o **programowanie iteracyjne** (model przyrostowy). Wymagania oraz rozwiązania ewoluują przy współpracy **samozarządzalnych zespołów**, których celem jest przeprowadzanie procesów wytwarzania oprogramowania. Pojęcie zwinnego programowania zostało zaproponowane w 2001w Agile Manifesto.

Generalnie metodyka oparta jest o zdyscyplinowane zarządzanie projektem, które zakłada **częste inspekcje wymagań i rozwiązań wraz z procesami adaptacji** (zarówno specyfikacji jak i oprogramowania). Metodyka ta najczęściej znajduje zastosowanie w **małych zespołach programistycznych**, **w których nie występuje problem komunikacji, przez co nie trzeba tworzyć rozbudowanej dokumentacji kodu**. Kolejne etapy wytwarzania oprogramowania zamknięte są w iteracjach, w których za każdym razem przeprowadza się testowanie wytworzonego kodu, zebranie wymagań, planowanie rozwiązań itd. Metoda nastawiona jest na szybkie wytwarzanie oprogramowania wysokiej jakości.

**Skład zespołów jest zazwyczaj wielofunkcyjny oraz samoorganizowalny**, bez zastosowania jakiejkolwiek hierarchii korporacyjnej. Członkowie zespołu biorą odpowiedzialność za zadania postawione w każdej iteracji. Sami decydują jak osiągnąć postawione cele.

Metoda nastawiona jest na **bezpośrednią komunikację pomiędzy członkami zespołu**, minimalizując potrzebę tworzenia dokumentacji. Jeśli członkowie zespołu są w różnych lokalizacjach, to planuje się codzienne kontakty za pośrednictwem dostępnych kanałów komunikacji (wideokonferencja, e-mail itp.).

### **Przykład zwinnej metodyki wytwórczej: Programowanie ekstremalne (XP)**

Metodyka programowania mająca na celu wydajne tworzenie małych i średnich "projektów wysokiego ryzyka". Przyświeca temu koncepcja prowadzenia projektu informatycznego, wywodząca się z obserwacji innych projektów, które odniosły sukces.

Podstawą ekstremalnego programowania jest synergia wynikająca ze stosowania rozmaitych praktyk, które same w sobie mają wiele zalet, lecz mogą być trudne w zastosowaniu. Łączne użycie tych praktyk ma zapewniać wyeliminowanie niedogodności każdej z nich.

**Zalecenia**

* **Iteracyjność**
  + Program tworzy się w iteracjach (krótkie, przyrostowe kroki programistyczne) - i co ważniejsze - planuje tylko następną iterację. Efektem każdej iteracji (kilka tygodni) powinna być wersja programu spełniającą założenia dla danej iteracji. Następnie planuje się co zrobić dalej. Odpowiada to zasadzie Open Source: "release early, release often" (wczesne i częste wydania).
* **Nie projektować z góry**
  + Nie można z góry przewidzieć, jaka architektura będzie najlepsza dla danego problemu. Dlatego należy ją tworzyć w miarę rozszerzania programu.
* **Testy jednostkowe**
  + Testy jednostkowe pisze się zanim w ogóle zacznie się pisać kod - najlepiej na początku iteracji. Potem pisze się kod, który potrafi je wszystkie przejść. Takie testy dają zapewnienie (o ile testy są dobrze napisane), że to, co ważne, zostanie zaprojektowane, na to zaś, co nie jest ważne, programiści nie będą tracić czasu.
* **Ciągłe modyfikacje architektury**
  + Architektura nie jest czymś, czego nie wolno ruszać. Jeśli modyfikacja architektury ułatwi przejście danej iteracji i nie zepsuje wyników testów uzyskanych na poprzednich, należy ją wykonać. Pod tę zasadę podlega także usuwanie wszystkich znanych błędów przed rozszerzeniem funkcjonalności.
* **Programowanie parami**
  + Programiści piszą w parach: jedna osoba pracuje przy klawiaturze i jest głównym koderem, druga obserwuje pierwszą, zgłasza poprawki, zadaje pytania wyjaśniające. Programiści programujący w parze zamieniają się rolami co kilkadziesiąt minut. Ta technika umożliwia wyłapanie wielu błędów oraz wzajemną naukę. Kod, którym zajmuje się tylko jedna osoba, ma tendencje do stawania się całkowicie niezrozumiałym dla kogokolwiek innego niż autor, więc dodatkowy programista zwiększa jakość kodu.

Praktyki XP:

* Gra w planowanie (planning game)
* Krótkie wydania
* Metafora
* Prosty projekt
* Testowanie
* Refaktoryzacja
* Programowanie parami
* Kolektywne prawo do zmian w kodzie
* Ciągła integracja
* 40-godzinny tydzień pracy
* Klient na miejscu
* Standardy kodowania

Nie jest jasne, czy sumarycznie łączna wydajność pracy przy takiej metodzie jest wyższa, taka sama, czy niższa niż w tradycyjnym programowaniu indywidualnym.

**Kwestie kontrowersyjne**

* Brak dokładnej specyfikacji.
* Konieczna stała dostępność przedstawiciela klienta.
* Wspólna "własność" kodu - każdy może zmieniać dowolny fragment systemu.

### **Przykład zwinnej metodyki zarządczej: Scrum**

To metodyka prowadzenia projektów. Zaliczana do tzw. metodyk zwinnych, zgodnych z Agile Manifesto. Najczęściej wykorzystywana jest w projektach informatycznych. Używana jest w skomplikowanych projektach, w których nie można przewidzieć wszystkiego, co może się przydarzyć lub w przypadku przedsięwzięć o wysokim stopniu innowacyjności.

Metodyka skupia się na:

* dostarczaniu kolejnych, coraz bardziej dopracowanych wyników projektu
* włączaniu się przyszłych użytkowników w proces wytwórczy
* samoorganizacji zespołu projektowego.

**Zespół i role**

* od 5 do 9 osób.
* charakter interdyscyplinarny, składa się z osób reprezentujących różne umiejętności.
* osoby uczestniczące w zespole nie mogą uczestniczyć w innych zespołach.

**Główne role w projekcie grają:**

* Product Owner (Właściciel produktu - WP)
  + Definiuje wymagania dotyczące produktu (product backlog)
  + Podaje priorytety wymagań
  + Wskazuje wymagania do realizacji podczas sprintu (sprint backlog)
  + Rozstrzyga wątpliwości dotyczące funkcjonalności
  + Dba o maksymalizację pracy zespołu
* Scrum Team (zespół wytwórczy - Z)
  + Konstruuje oprogramowanie zgodne ze standardami
  + Testuje oprogramowanie
  + Buduje wersje dystrybucyjne oprogramowania
* Scrum Master (mistrz Scrum - MS)
  + Odpowiedzialny za proces wytwórczy i zarządczy zgodny ze Scrum
  + Ochrania zespół przez środowiskiem zewnętrznym

**Założenia SCRUM:**

* Minimalizacja nakładów pracy związanych ze zmianą wymagań
* Ciągłe planowanie
* Częsta prezentacja cząstkowego produktu

**Charakterystyka:**

* Iteracyjna i przyrostowa (stało-czasowe iteracje, nazywane **sprintami**)
* Adaptacyjna

**Opis metodyki**

Zespół projektowy pracuje w określonym przedziale czasowym zwanym przebiegiem (ang. sprint). Efektem przebiegu za każdym razem powinno być dostarczenie użytkownikom kolejnego działającego produktu. Zasadą jest to, że zmiany wprowadzane w jednym przebiegu muszą być namacalne dla użytkowników. Muszą wnosić wartość funkcjonalną (addytywną). Przebieg może trwać od 2 do 6 tygodni. Zaleca się stosowanie przebiegów o stałych długościach.

W pierwszym etapie tworzona jest lista wymagań użytkownika, są one gromadzone w postaci "historyjek". Każda historyjka opisuje jedną cechę systemu. Właściciel projektu jest też zobowiązany do przedstawienia priorytetów wymagań oraz głównego celu przebiegu. Po tym formułowany jest rejestr wymagań. Cel przebiegu jest zapisywany w widocznym miejscu w pokoju członków zespołu.

Następnie wybierane są zadania o najwyższym priorytecie, a jednocześnie przyczyniające się do realizacji celu projektu. Szacuje się czas realizacji każdego zadania. Lista zadań wraz z oszacowaną czasochłonnością nosi nazwę rejestru zadań przebiegu (sprint backlog).

Po etapie przygotowawczym zespół przechodzi do realizacji przebiegu. W jego trakcie Właściciel Produktu nie może ingerować w prace zespołu. Nie powinno się także zmieniać zakresu sprintu.

Jako że zespół z założenia jest samoorganizującym się ciałem, nie ma mowy o odgórnym przypisywaniu zadań do poszczególnych członków zespołu, lecz samodzielnie dokonują oni wyboru realizowanych zadań, według wspólnych ustaleń, umiejętności czy innych preferencji.

Komunikacja - spotkania:

* **Sprint planning meeting** (planowanie sprintu, co będzie zrobione., szacowanie czasu) Zespół + product owner, na początku każdego sprintu
* **Daily scrum** (stand-up meeting) – codziennie, ok 15 min, na stojąco, odpowiedź na pytania: co zostało zrobione, co planuje się zrobić dziś, jakie były problem I jak je rozwiązano.
* **Sprint review** – na koniec każdego sprintu, prezentowanie wyników pracy zespołu. Powinni w nim uczestniczyć wszyscy zainteresowani projektem. Na spotkaniu każdy członek zespołu może zabrać głos i wyrazić opinię o produkcie. Nie prezentuje się nieskończonych produktów.
* **Sprint retrospective –** po każdym sprint review. Spotkanie zespołu + scrum mastera. Odpowiedź na pytania: co poszło źle w ostatnim sprincie? Co może być poprawione w następnym?

# Metodyki ciężkie

**Metodyki ciężkie (twarde)** charakteryzują się

* znaczną liczbą niezbędnej dokumentacji projektowej oraz
* sformalizowanymi procesami:

1. podejmowania decyzji,
2. raportowania,
3. sprawozdawczości,
4. odbiorów produktów itp.

Metodyki ciężkie kojarzone są najczęściej z dużymi projektami i dużymi zespołami projektowymi. Nie oznacza to jednak, że nie mogą być stosowane w mniejszych projektach.

### **Przykład ciężkiej metodyki wytwórczej: **Rational Unified Process (RUP)****

Proces iteracyjnego wytwarzania oprogramowania opracowany przez firmę Rational Software Corporation (firma została przejęta przez IBM).

Proces RUP nie jest pojedynczym, ściśle określonym procesem, ale raczej szablonem procesu. Został on zaprojektowany w celu przystosowania do charakteru konkretnej organizacji (przedsiębiorstwa), konkretnego zespołu projektowego lub nawet charakteru konkretnego projektu. Z szablonu RUP można wybrać elementy w zależności od konkretnych potrzeb.

Rational Unified Process (RUP) to także nazwa oprogramowania, opracowanego przez Rational Software (obecnie dostępnego w IBM). Produkt ten zawiera hipertekstową bazę wiedzy z przykładowymi artefaktami oraz szczegółowymi opisami wielu typów czynności. Process RUP definiowany jest także w produkcie Rational Method Composer (RMC), który pozwala na tworzenie spersonalizowanych wersji RUP.

**Cykl życia projektu w RUP:**

* Faza rozpoczęcia (Inception chase)
* Faza opracowywania (Elaboration phase)
* Faza konstrukcji (Construction phase)
* Faza przekazania systemu (Transition phase)

**Dyscypliny w RUP:** ( dyscyplina - pakiet grupujący zadania wykonywane w ramach danego etapu wytwórczego)

* Modelowanie biznesowe
* Wymagania
* Analiza i projekt
* Implementacja
* Testowanie
* Wdrożenie
* + Zarządzanie zmianami i konfiguracją
* +Zarządzanie projektem
* +Środowisko

**Praktyki RUP:**

1. Iteracyjnym wytwarzaniu oprogramowania (Iterative Development)
2. Zarządzaniu wymaganiami (Requirement Management)
3. Używaniu architektury bazującej na komponentach (Component-based architecture)
4. Graficznym projektowaniu oprogramowania
5. Kontroli jakości oprogramowania (Quality Assurance)
6. Procesu kontroli zmian w oprogramowaniu (Change Management)

**RUP używa podejścia iteracyjnego i przyrostowego z następujących powodów:**

* Integracja oprogramowania robiona krok po kroku podczas wytwarzania oprogramowania, ograniczając go do mniejszej liczby elementów
* Integracja jest prostsza i mniej kosztowna
* Składowe oprogramowania są projektowane oddzielnie i łatwiej poddają się reużywalności
* Łatwiej wykrywać zmiany wymagań i łatwiej nimi zarządzać
* Ryzyka identyfikowane i atakowane są wcześnie ponieważ każda iteracja pozwala wykryć kolejne ryzyka
* W iteracjach ulepszana jest architektura oprogramowania

### **Przykład ciężkiej metodyki zarządczej: **Prince2****

Omówienie w innym pytaniu.

## Porównanie założeń w metodykach ciężkich i lekkich

|  |  |
| --- | --- |
| Metodyki ciężkie | Metodyki lekkie |
| Klient wie czego chce | Klient dowiaduje się z czasem czego chce |
| Deweloperzy wiedzą jak zbudować system | Deweloperzy dowiadują się z czasem jak zbudować system |
| Nic się nie zmieni po drodze | Zmiany będą się pojawiać |
| Można z góry określić każdy krok wytwarzania oprogramowania |  |