Tworzenie gier na urządzenia mobilne

dr Przemysław Juszczuk

Katedra Inżynierii Wiedzy

Wykład 4

O czym dzisiaj?

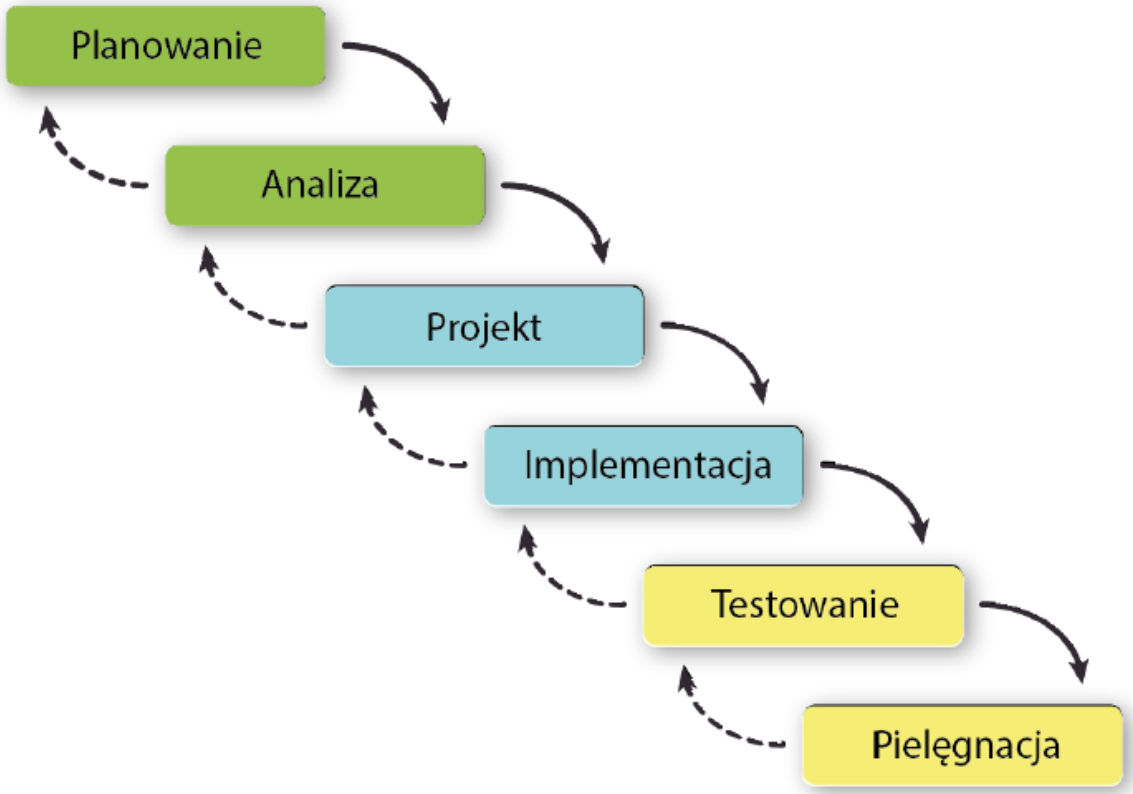
- Metodyki tworzenia oprogramowania;
- Praca w zespole;
- Zarządzanie projektem;
- Narzędzia wspomagające i dobre praktyki;
- Zabezpieczenie kodu.

Jaki model wybrać?

- zależny od dostępnych środków;
- jaki jest sposób organizacji zespołu;
- jakie narzędzia są dostępne;
- na ile można korzystać z gotowych rozwiązań?

Podejście podstawowe

1. Zaproponuj koncepcję;
2. Wykonaj projekt;
3. Uzgodnij z klientem/zespołem.
4. Zakończenie / powrót do punktu drugiego.



Rysunek: Model kaskadowy

*Winston Royce, Managing the Development of Large Software Systems, 1970 rok.

Cechy modelu

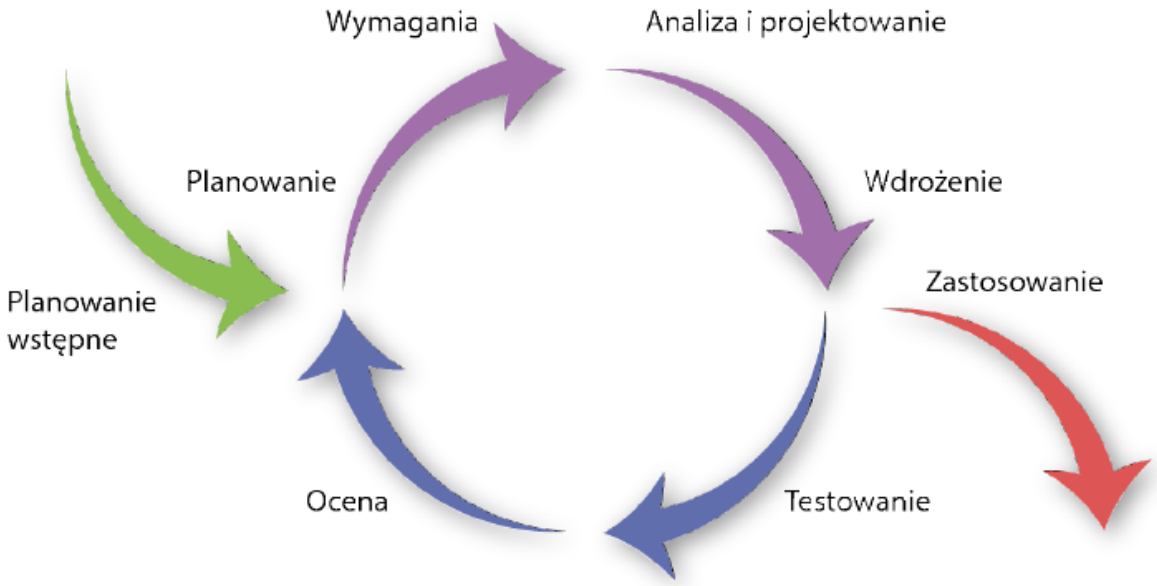
- iteracyjnie następujące po sobie fazy utrudniają elastyczne działanie;
- ewentualne powtórzenie iteracji jest kosztowne;
- problem przy komunikacji z klientem;
- żeby przejść dalej, to musimy zakończyć bieżącą fazę;
- może być skuteczny, jeżeli wymagania są poprawnie zdefiniowane;
- standaryzacja podejścia – dokładnie wiemy, jaki etap jest następny;
- ułatwia organizację;
- połowa zespołu może czekać druga połowa zakończy poprzednią fazę.

Model kaskadowy z nawrotami

- po każdej fazie następuje jej weryfikacja;
- powrót nie jest obligatoryjny – daje możliwość korekty błędów;
- konieczny dodatkowy nakład czasowy związany z weryfikacją;
- łatwiejsze (niż w klasycznym modelu kaskadowym) planowanie.

Model przyrostowy

- Przygotowanie ogólnych wymagań oprogramowania na niskim poziomie szczegółowości oraz wyodrębnienie funkcji systemu;
- Wyselekcjonowanie funkcji przeznaczonych do realizacji w najbliższym przyroście;
- Uszczegółowienie wybranych funkcji - szczegółowe zaprojektowanie;
- Implementacja wybranych funkcji;
- Testowanie wersji wzbogaconej o nowe funkcje dodane w przyroście;
- Dostarczenie wersji demo wzbogaconej o funkcje z aktualnego przyrostu;
- Powtarzanie przyrostów aż do ukończenia projektu.



Rysunek: Model przyrostowy

Model z prototypem

- ogólne określenie wymagań;
- budowa i weryfikacja prototypu;
- dobry moment na ewentualne porzucenie koncepcji / zmianę założeń i ponowne przygotowanie prototypu;
- szczegółowe określenie wymagań (już po pozytywnej weryfikacji prototypu);
- realizacja pełnej wersji;
- prototyp istotnie zwiększa nakład czasowy a także generuje dodatkowe koszty.

Model RAD - Rapid Application Development

- podział projektu na niezależne składowe;
- równoległa realizacja poszczególnych funkcjonalności bazując np. na modelu kaskadowym;
- zespoły pracują niezależnie nad różnymi funkcjami;
- konieczność integracji podzespołów na finiszu projektu.

Model reuse

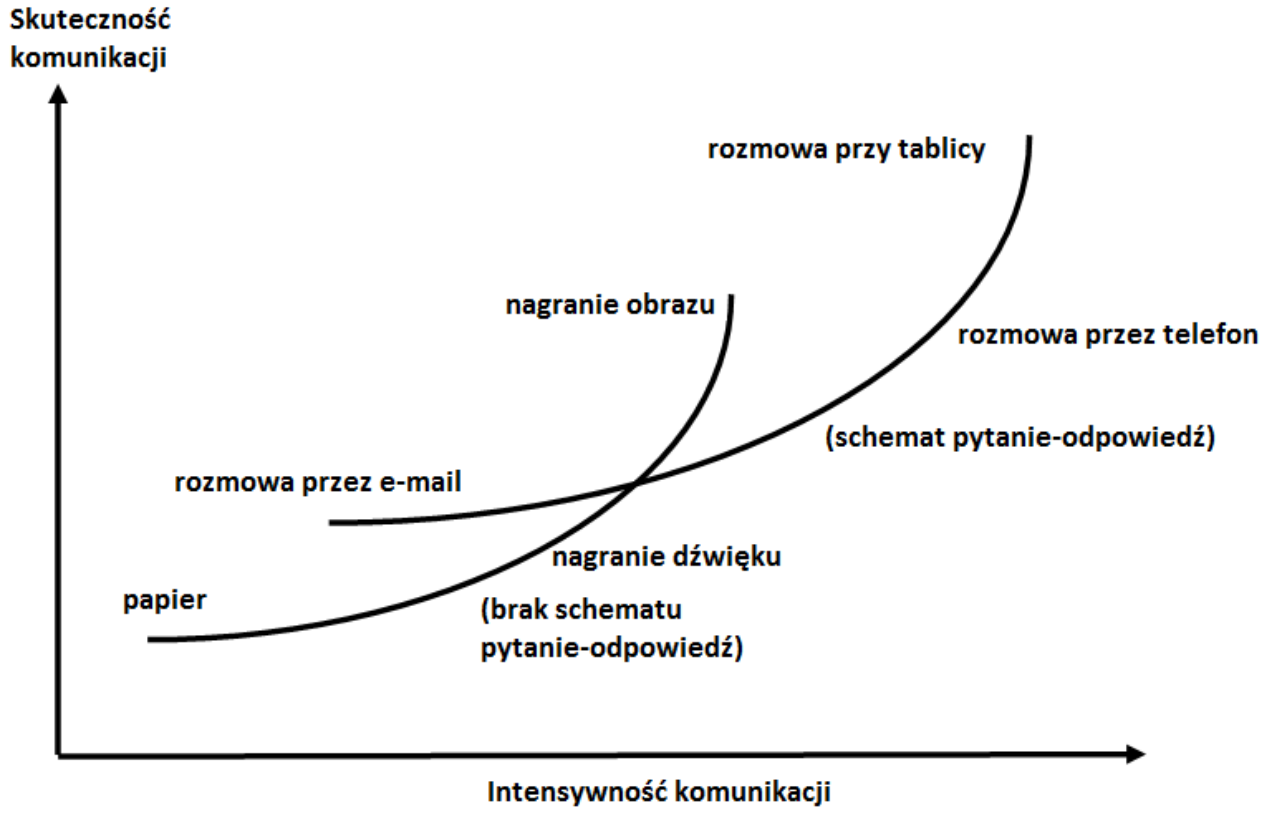
- możliwość wykorzystania gotowych rozwiązań/komponentów;
- znaczna redukcja kosztów i czasu realizacji projektu;
- narzucenie standardów.

Programowanie zwinne

- najważniejszym czynnikiem w procesie tworzenia oprogramowania jest zespół;
- zastosowanie złych procesów może wpłynąć negatywnie na pracę całego zespołu;
- zespół składający się z samych doskonałych programistów nie gwarantuje sukcesu, a wręcz przeciwnie - zwiększa szansę na nieudany projekt, jeśli nie uda się stworzyć prawdziwego zespołu;
- ważniejszy jest odpowiedni podział ról, zgranie zespołu i doświadczenie;
- dobra współpraca w zespole daje dużo lepsze rezultaty niż grupa indywidualistów;
- zastosowane narzędzia mają drugorzędne znaczenie, chociaż należy pamiętać o tym, że zespół powinien się stale rozwijać i poznawać nowe rozwiązania.

Programowanie zwinne - oprogramowanie ponad dokumentację

- w dokumentacji powinny być zawarte najważniejsze informacje związane z oprogramowaniem oraz strukturą programu;
- brak dokumentacji jest poważnym błędem, ponieważ zaciera wszystkie informacje o oprogramowaniu (niebezpieczne w przypadku zmian personalnych w zespole);
- niezależnie od dokumentacji wszystkie informacje o oprogramowaniu powinny być przekazywane w rozmowach pomiędzy członkami zespołu;
- najważniejsze jest włożenie odpowiedniego wysiłku w bezpośrednie przekazywanie wiedzy nowym członkom zespołu.



Rysunek: Efektywność komunikacji związanej z oprogramowaniem

Współpraca z klientem ponad formalne ustalenia

- w przypadku tworzenia oprogramowania nie ma możliwości założenia z góry dokładnych terminów realizacji i kosztów, jakie zostaną na to poniesione. Jeśli chcemy przygotować oprogramowanie dobrej jakości, to konieczne jest dostosowanie zmian do wymagań klienta;
- kontrakty powinny zawierać ogólne zapisy dotyczące terminów realizacji etapów;
- im częściej klient będzie mógł testować przygotowane oprogramowanie, tym częściej wprowadzane będą zmiany lub nawet modyfikowany będzie zakres prac.

Reagowanie na zmiany ponad podążanie za planem

- bardzo trudne jest przygotowanie pełnych i jasnych wymagań dotyczących projektu, ale nawet ich opracowanie nie gwarantuje poprawnego określenia ilości czasu potrzebnego do realizacji zadań;
- doświadczony zespół jest w stanie szacować czas potrzebny na stworzenie projektu, jednak dopiero w trakcie jego realizacji zyskuje pełną wiedzę na temat wymagań klienta;
- klient dopiero po pewnym czasie jest w stanie dokładnie określić swoje wymagania w sposób zbieżny do stosowanej metodyki;
- w przypadku ogólnego opisu wszystkie wymagania powinny być możliwie ogólne, a dopiero szczegółowe zadania w poszczególnych przyrostach powinny być maksymalnie uszczegółowione.

Programowanie ekstremalne

Programowanie ekstremalne (ang. Extreme Programming), w skrócie oznaczane również, jako XP, to metodyka programowania bazująca na zasadach zwinnego tworzenia oprogramowania. W przypadku tej metodyki jej głównym przeznaczeniem są małe i średnie projekty prowadzone przez niewielkie zespoły z zachowaniem modelu iteracyjnego.

* Kent Beck Extreme Programming Explained, 1999.

Programowanie ekstremalne miało na celu zbudować zmotywowany zespół tworzący dobrej jakości oprogramowanie, bez szczegółowych ograniczeń czasowych i tworzenia zbędnej dokumentacji.

4 czynności w XP

- tworzenie kodu źródłowego – podstawa systemu to kod źródłowy, więc jego wytwarzanie prowadzi do rozwijania systemu;
- testowanie – dopóki nie przetestujemy, to nie wiemy, czy wszystko działa (testy jednostkowe, testy akceptacyjne).
- komunikacja – trzeba rozumieć ideę na tyle, żeby klientowi przedstawić techniczne aspekty rozwiązania;
- projektowanie – szczególnie istotne w przypadku złożonych projektów.

4 wartości w XP

- wsparcie dla prostych projektów. Liczy się komunikacja oraz opinie zwrotne; wymiana informacji pomiędzy członkami zespołu;
- zaczynamy od prostoty – minimal working example (solution). Kolejne funkcjonalności dokładamy w miarę rozwijania systemu;
- im więcej feedback, tym lepiej – info zwrotne od systemu (testy jednostkowe), od zespołu – zespół szacuje czas potrzebny na implementację lub modyfikację przy zmianie funkcjonalności;
- realne spojrzenie na swój kod i jego obiektywna ocena – jeżeli coś nie działa, to nie brniemy dalej. Usuwamy i zaczynamy jeszcze raz.

12 praktyk XP

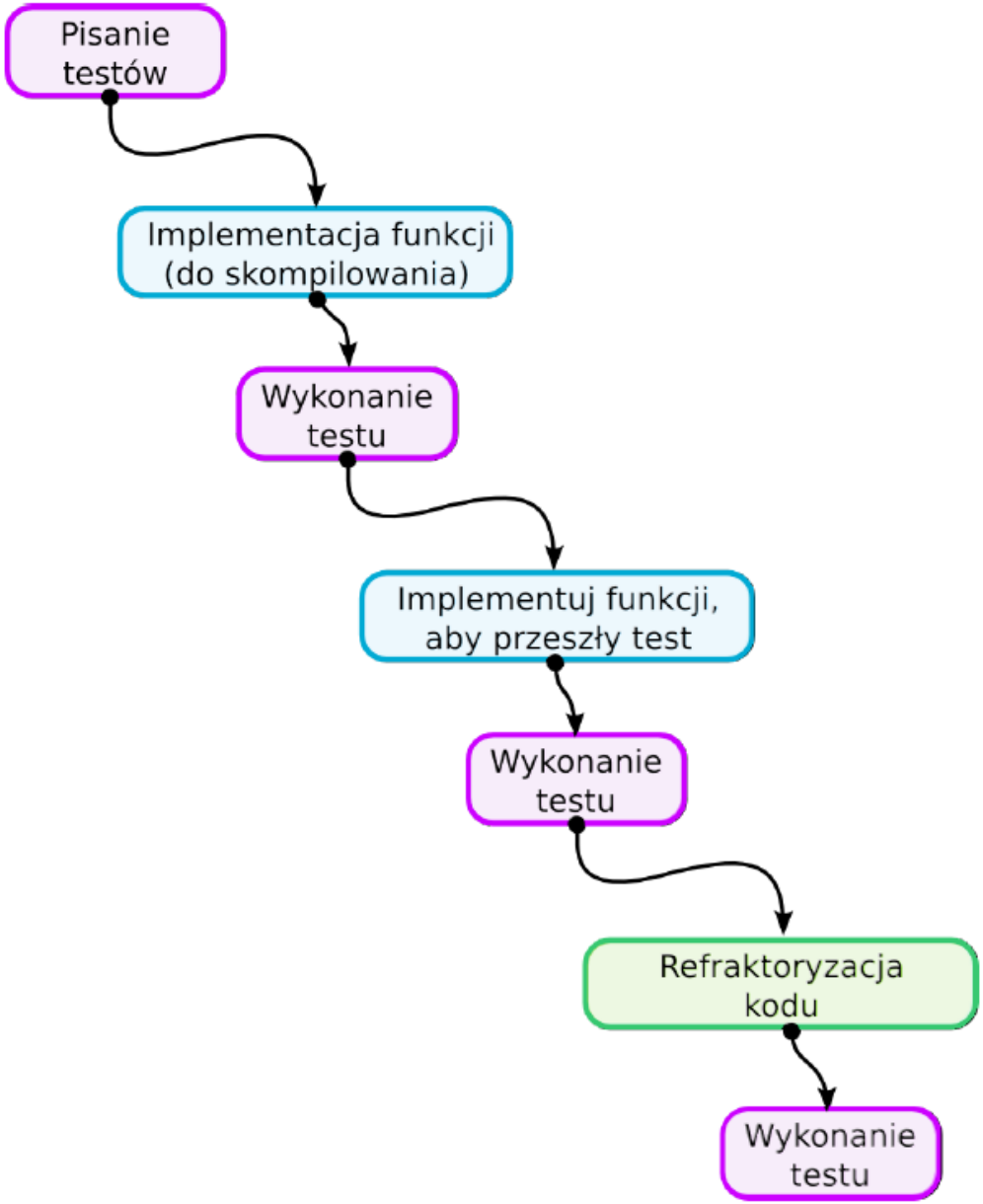
- programowanie w parach – na to nie będzie czasu;
- Test-driven development – o tym nie będzie. Tylko 3 miesiące na projekty – za mało;
- wspólna odpowiedzialność – raczej podział na zadania;
- planowanie – od tego zaczynamy projekt;
- refaktoryzacja – o tym będzie teoretycznie.
- ciągłe łączenie – to powinno być w projektach.
- małe wydania – to też.
- standardy programowania – o tym będzie teoretycznie;
- ciągłe testowanie – na to nie będzie czasu;
- prosty projekt – od tego powinien zacząć się projekt;
- metafora systemu – czyli słownik pojęć systemu. Nie przyda się;
- zrównoważone tempo – czyli stała wydajność i stałe obciążenie (patrz ocena bardzo dobra).

Zespół - programowanie parami

- programiści, deweloperzy;
- przedstawiciele klienta;
- testerzy, których zadaniem jest m.in. współpraca z klientem przy tworzeniu scenariuszy;
- śledzący (lub bezpośrednio tracker) odpowiadający za kontrolę projektu, a więc m.in. określanie stopnia jego realizacji i szacowanie czasu jego zakończenia;
- trener (lub bezpośrednio coach) jest w przypadku programowania ekstremalnego nie tylko mentorem i doradcą zespołu, ale pełni rolę nadzorcy i kierownika projektu.

Syndrom LOOP

- Late - późno;
- Over budget - przekroczony budżet;
- Overtime - nadgodziny;
- Poor quality - kiepska jakość.



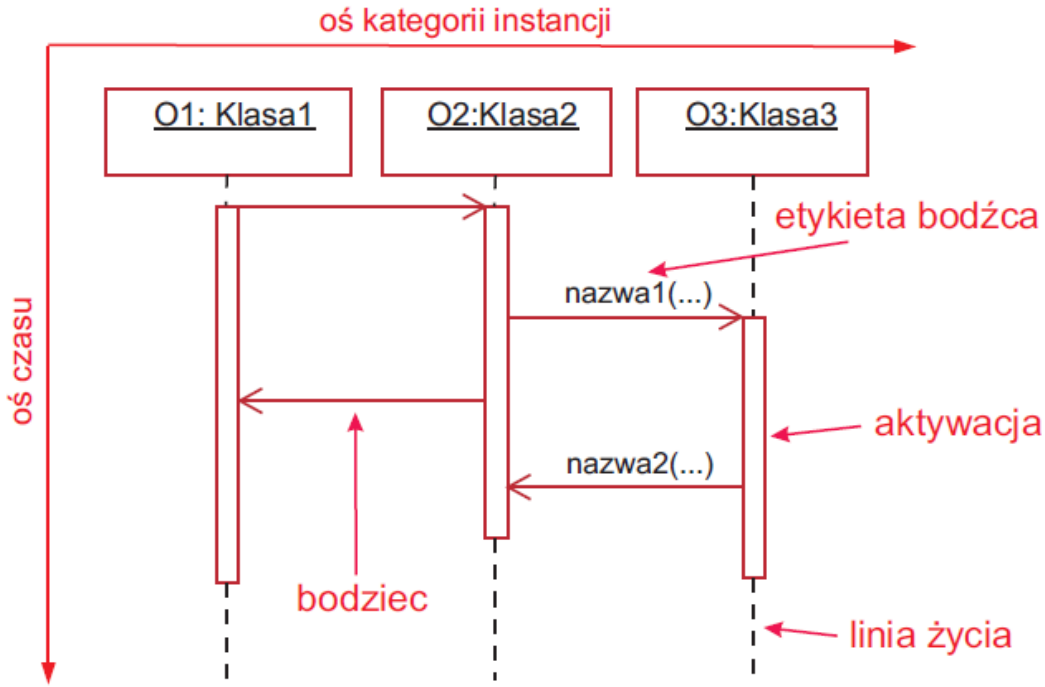
Rysunek: Programowanie ekstremalne - model przyrostowy

Programowanie ekstremalne jako metodyka wysokiego ryzyka

- bezpośrednia komunikacja;
- prostota – zawsze należy poszukiwać najprostszych rozwiązań, które spełniają obecne lub przyszłe (możliwe do przewidzenia) wymogi;
- sprzężenie zwrotne pomiędzy klientem a zespołem;
- odwaga, która jest potrzebna przy projektach dużego ryzyka realizowanych przy zastosowaniu programowania ekstremalnego.

UML a skrypty

- diagram use case;
- diagram klas;
- diagram stanów;
- diagram sekwencji



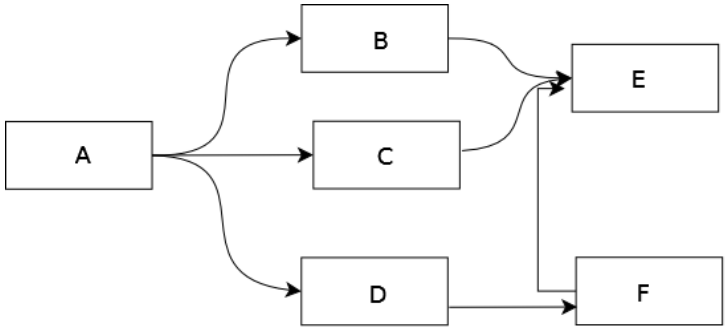
Rysunek: Przykład diagramu sekwencji

Zarządzanie projektem - harmonogramowanie zadań

Zagadnienie harmonogramowania zadań dotyczy ustalenia, jakie działania powinny zostać wykonane, określenia ich priorytetów i powiązań. Układ harmonogramu zależy od dostępności zasobów oraz istotności zadań.

Diagram sieciowy - relacje

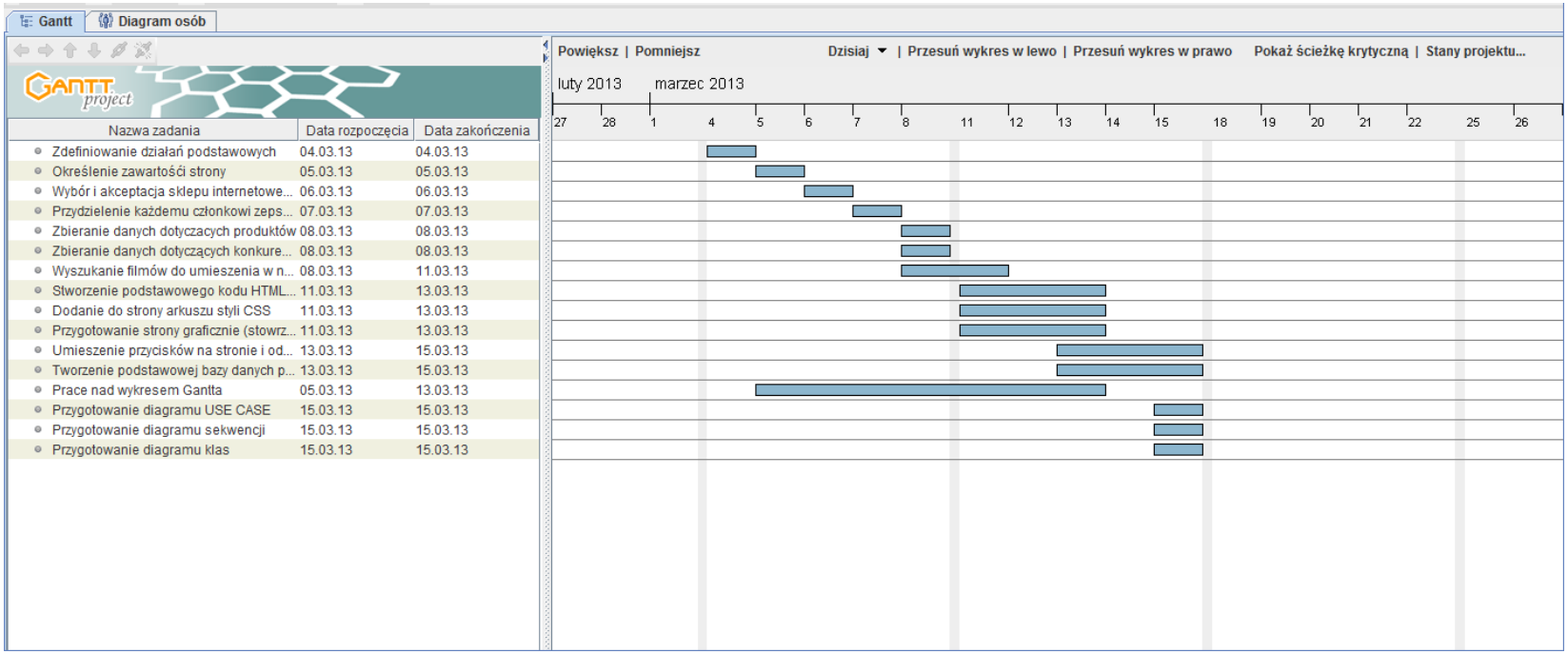
- koniec - początek - zadanie może być wykonane dopiero po zakończeniu poprzedniego;
- koniec - koniec - dane zadanie musi być zakończone, aby możliwe było zakończenie innego zadania;
- początek - początek - zadanie musi zostać rozpoczęte, aby rozpocząć inne zadanie.



Rysunek: Diagram sieciowy

Definicje

- działanie krytyczne - działanie kluczowe do zrealizowania całego projektu. W jego przypadku nie jest możliwe żadne przesunięcie w czasie, gdyż wpłynie to na cały projekt;
- scieżka krytyczna - najdłuższa scieżka w całym diagramie. Wszystkie elementy scieżki krytycznej są działaniami krytycznymi;
- nakład pracy - liczba godzin roboczych (lub dowolnej innej jednostki) konieczne do wykonania danego zadania.



Rysunek: Diagram Gantta

Diagram Gantta

- zadanie krytyczne - podobnie, jak w diagramie sieciowym, zadanie kluczowe dla całego systemu;
- zadanie niekrytyczne - zadanie mniej istotne, nie wpływa wyraźnie na projekt, jednak jego ukończenie może znacznie przyspieszyć wykonanie innych zadań;
- podsumowanie - oznaczenie informujące o pewnym etapie projektu. Najczęściej po podsumowaniu znajduje się kamień milowy;
- kamień milowy - szczególne zadanie warunkujące pomyślne zakończenie pewnej fazy. Jego wykonanie konieczne jest do przejścia do dalszego etapu projektu.

Scenariusze operacyjne

- opis tego, co użytkownik robi, aby osiągnąć założony cel;
- bazujemy na przypadkach użycia;
- wskazujemy aktora, który wykonuje dany scenariusz.

Scenariusze testowe

- wskazujemy zakres testów – wycinek funkcjonalności;
- podajemy założenia – wersja aplikacji, miejsce zgłoszenia ewentualnych błędów;
- przy akcji użytkownika oczekujemy konkretnej odpowiedzi systemu.

Co na następnym wykładzie?

- krótkie przypomnienie dotyczące Unity;
- scena i elementy GUI;
- podstawy modeli (trochę więcej teorii);
- transformacje obiektów w przestrzeni;
- źródła światła na scenie;
- mapy świetlne, wypalanie a nawigacja NPC;
- kafelki a moduły.

Dziękuję za uwagę