Model implementacyjny - pokazujemy jakie komponenty nasz system powinien zawierać, pokazujemy pewne zależności między komponentami

O - specyfikacja komponentów (odpowiada plikom z rozszerzeniem .h lub .cpp)

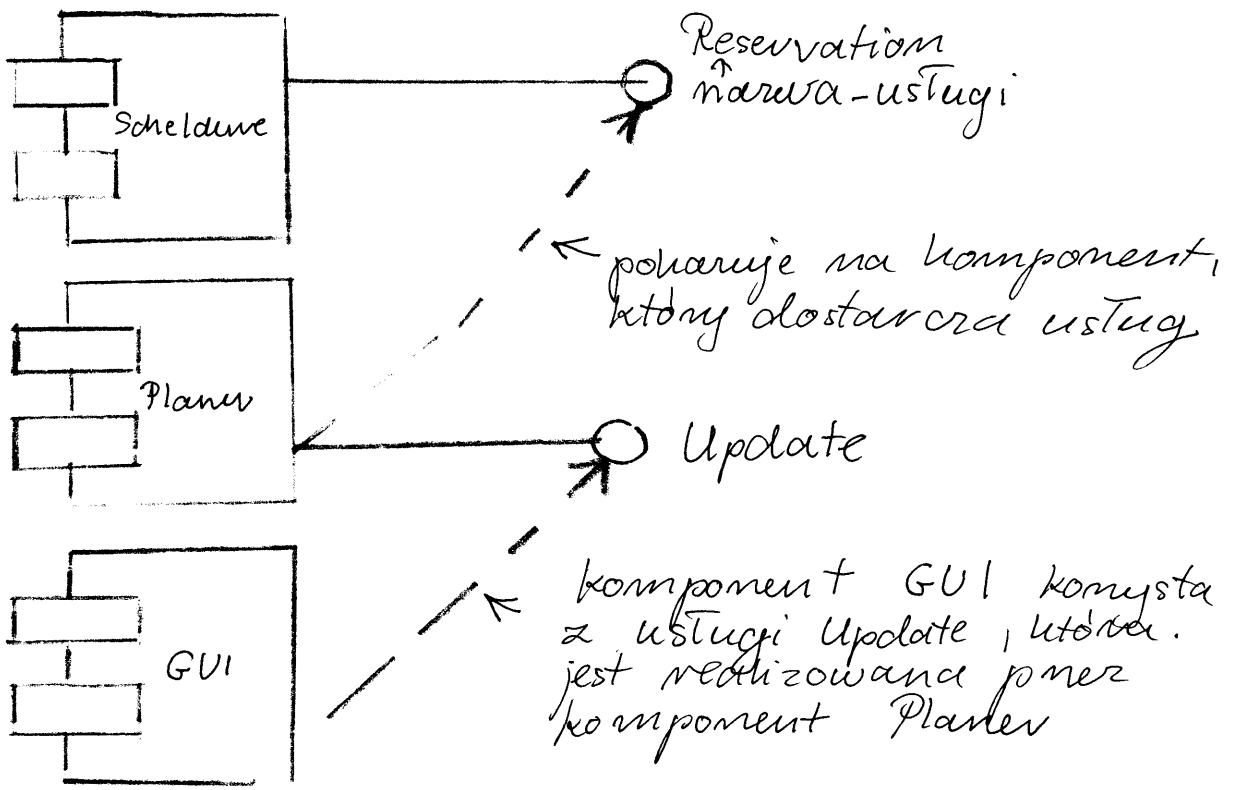
Komponent - coś co powstanie ze zbioru kilku lub kilkunastu klas z naszego projektu lub jakiś zewnętrzny element oprogramowania, który dołączamy do naszego systemu np. jakaś biblioteka lub jakiś program, który będziemy dostarczać do naszej aplikacji.

Zależności między komponentami pokazujemy, za pomocą przerywanej linii zależności

— — — — → — linia zależności

Linia ta jest skierowana ze względu dostawcy/ela np. komponent A korzysta z jakiś usług komponentu B, np takich, które komponent B udostępnił.

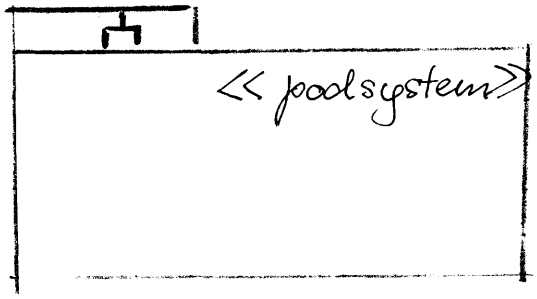
Diagram komponentów

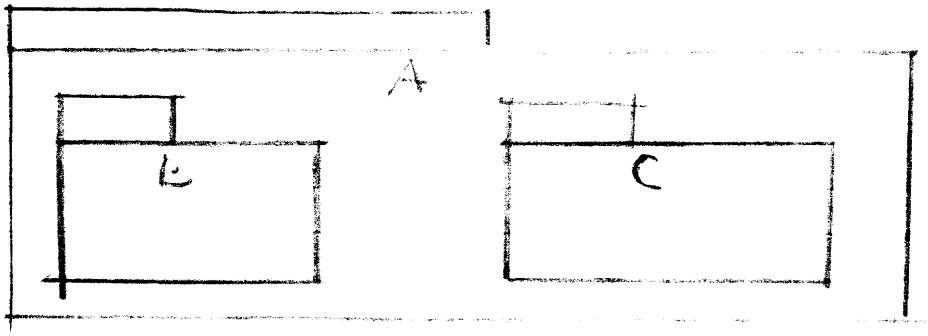


Jeżeli komponent jest elementem oprogramowania to musimy określić jakie klasy wchodzą w jego skład.

podsystemy

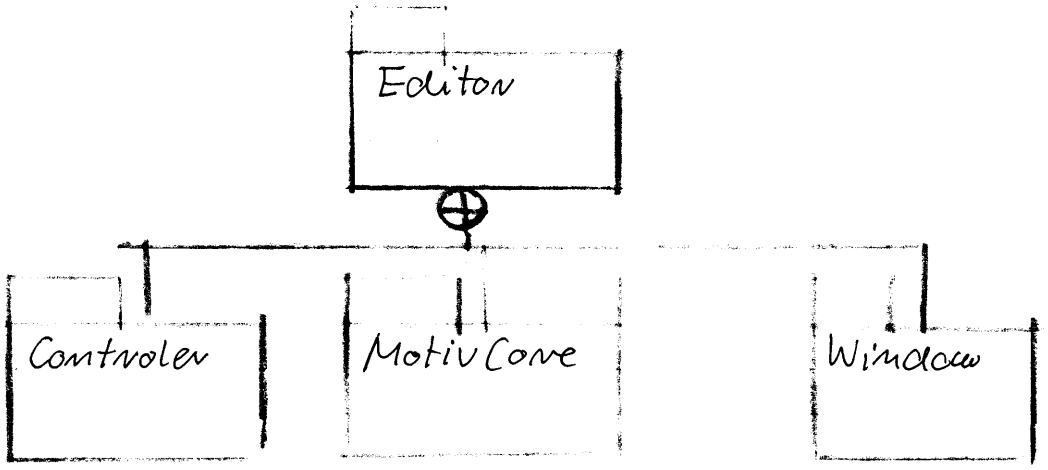
Model - fizyczny sposób pokrywania architektury naszego systemu, projekt architektoniczny. Pewna grupa klas może tworzyć podsystem. Klasy współpracując mogą dostarczać jakieś funkcje systemu - podsystem - poziom fizyczny. Za pomocą podsystemów możemy zaprojektować architekturę naszego systemu





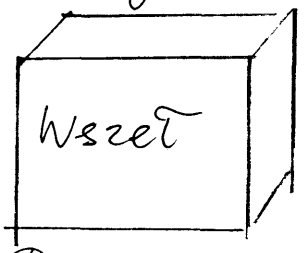
Podsystem A zawiera podsystemy B i C, pokrywa budowę fizyczną naszego systemu

Żeby pokazać zależności między podsystemami należy umieścić stereotypy, które opisują rodzaj zależności



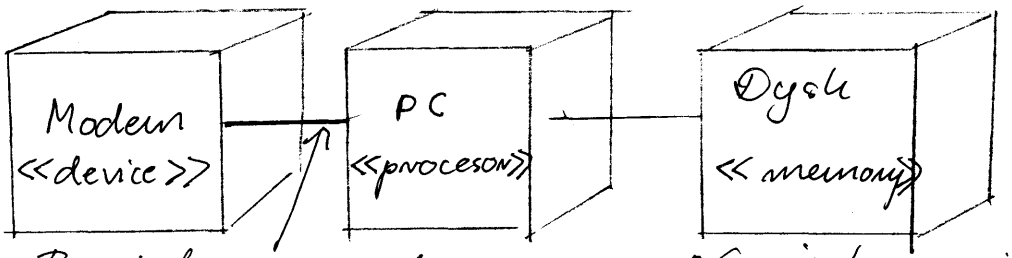
Edytor zawiera takie podsystemy jak Controler, MotivCore, Window.

Diagram (ang. deployment diagram) wdrożeniowy - pozwala jak software który zbudowaliśmy został zaimplementowany i gdzie na jakich jednostach.. Elementami diagramu wdrożeniowego są węzły. Symbolem graficznym węzła jest prostopadłościan.



Takim węzłem może być jakieś urządzenie np. modem, procesor. Podajemy zwykle typ tego węzła w postaci stereotypu.

Pomiędzy węzłami pokazujemy połączenia wiązące dekacje np. asocjacji



Pomiędzy modemem a PC jest możliwy przepływ informacji (asocjacja nieskierowana)

Diagram rozmieszczenia, instalacyjny, wdrożeniowy

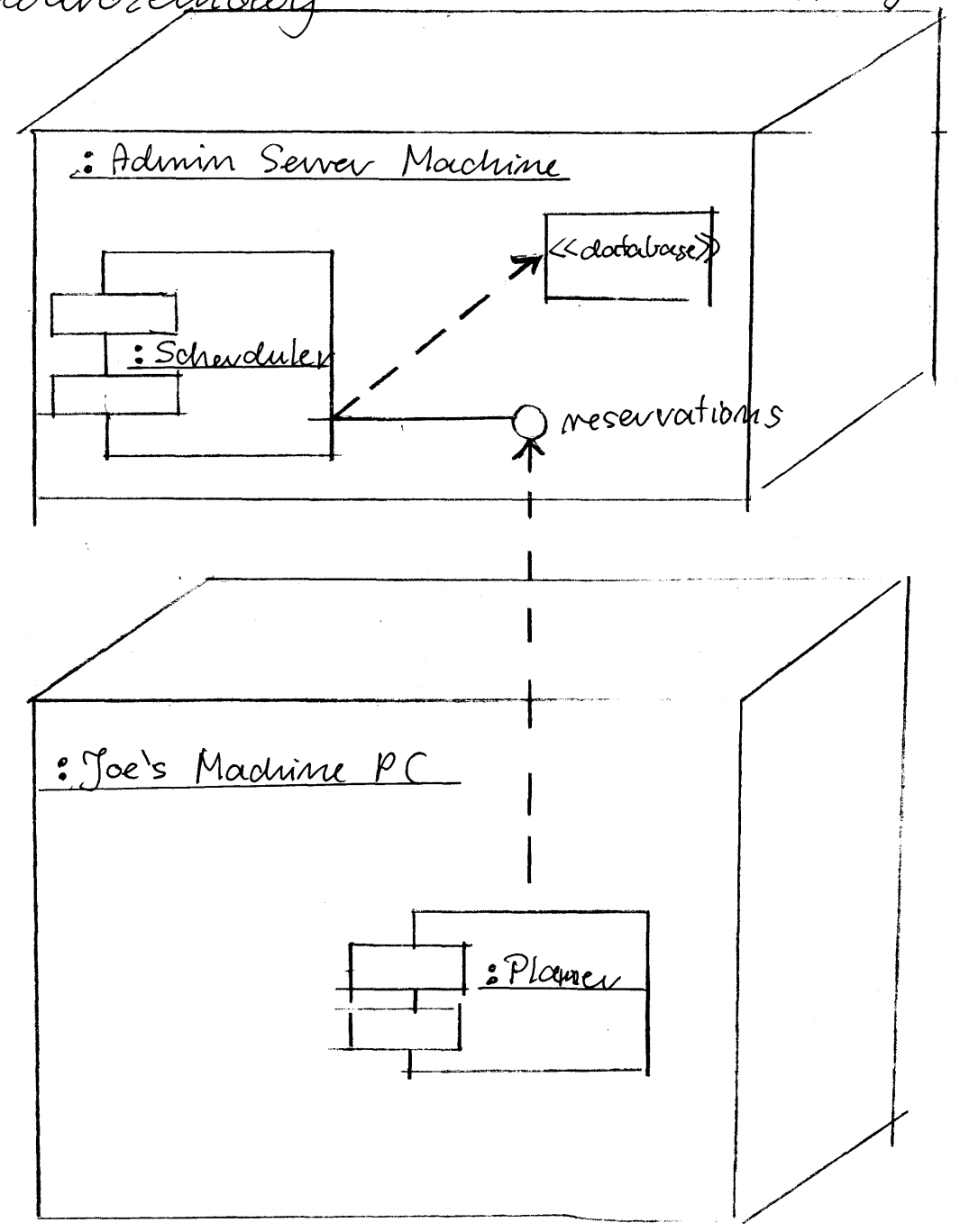


Diagram rozmieszczenia wyraża się razem z diagramem komponentów, połączonych z jakie są elementy sprzętowe oraz jak rozmieszczone jest oprogramowanie na tych sprzętach.

Komponent Planer został zainstalowany na Joe's Machine PC, a komponent Scheduler na serwerze Administratora

W diagramie rozmieszczenia pokazujemy konkretne instancje komponentów, które zostały zainstalowane na konkretnych komputerach (nazwa komponentu musi być podkreślona i pisana po :)

Ponieważ pod koniec lat sześćdziesiątych miał miejsce tzw. kryzys oprogramowania, wiele realizowanych projektów upadało, a koszty oprogramowania rosły w sposób lawinowy (teraz rosną trochę wolniej), szukano więc rozwiązania, które pomogłoby te koszty obniżyć. Tak zaczęła powstawać inżynieria oprogramowania (software engineering). Zaczęto szukać wskazówek w innych zawodach inżynierskich, które próbowano przenieść do produkcji softwaru.

Cele inżynierii oprogramowania:
- poszukiwanie i udoskonalenie metod oraz technik produkcji o wysokiej jakości
- produkcja w sposób najbardziej efektywny (żeby opłacało się budować oprogramowanie).

Oprogramowanie wysokiej jakości: (warunki)
- musi działać zgodnie z wymaganiami określonymi w specyfikacji projektowej
- musi być szybkie, wydajne i funkcjonalne jak określnie tego ceny/technologii
- dać się łatwo pielęgnować (konserwacja i modyfikacja)
- posiadać pełną dokumentację użytkową i projektową

Inżynieria oprogramowania dotyczy oprogramowania opracowywanego przez zespoły, które muszą postępować zgodnie z jej zasadami, ponieważ bez nich projekt upadnie.

Inżynieria oprogramowania oferuje:
- techniki i narzędzia ułatwiające pracę na złożonymi systemami
- systematyzuje proces produkcji oprogramowania, tak by ułatwić jego monitorowanie i planowanie.
- metody usprawniające analizę nieznanych problemów.

Czym zajmuje się Inżynieria Oprogramowania:
- sposobami prowadzenia przedsięwzięć informatycznych
- technikami szacowania kosztów, harmonogramowania.
- metody analizy i projektowania systemów (model obiektowy i strukturalny)
- parametami niezawodności systemów (dla systemów czasu rzeczywistego np. dla systemu kontroli lotów)
- sposobami przygotowania dokumentacji technicznej i użytkowej
- sposobami testowania systemów
- procedurami kontroli jakości
- technikami pracy zespołowej

Jakość oprogramowania

Model Mc Call'a

Kryteria oceny jakości:
a) związane ze sposobem działania oprogramowania
b) związane z możliwością wprowadzenia zmian i poprawek

c) związane z przenośnością oprogramowania

Modele cyklu życia oprogramowania pokazują, jak organizacja proces produkcji oprogramowania.

1. Model wodospadowy - przejście do kolejnej fazy następuje po zamknięciu poprzedniej (możliwe też powroty do faz wcześniejszych)
Fazy:
- specyfikacja wymagań
- projektowanie
- implementacja
- testowanie
- wdrożenie i pielęgnacja

Im później znajdziemy błąd w oprogramowaniu tym więcej godzin musieli się naprodować aby go naprawić (trzeba się cofać do wcześniejszych faz).

Wykrycie błędu w fazie wdrożenia i pielęgnacji kosztuje około 100 i więcej razy więcej niż w fazie produkcji)