Концепции на проектирането

- **Концепции на проектирането**

Slide Set to accompany

Software Engineering: A Practitioner's Approach, 8/e

by Roger S. Pressman

Лектор: Доц. д-р Ася Стоянова-Дойчева

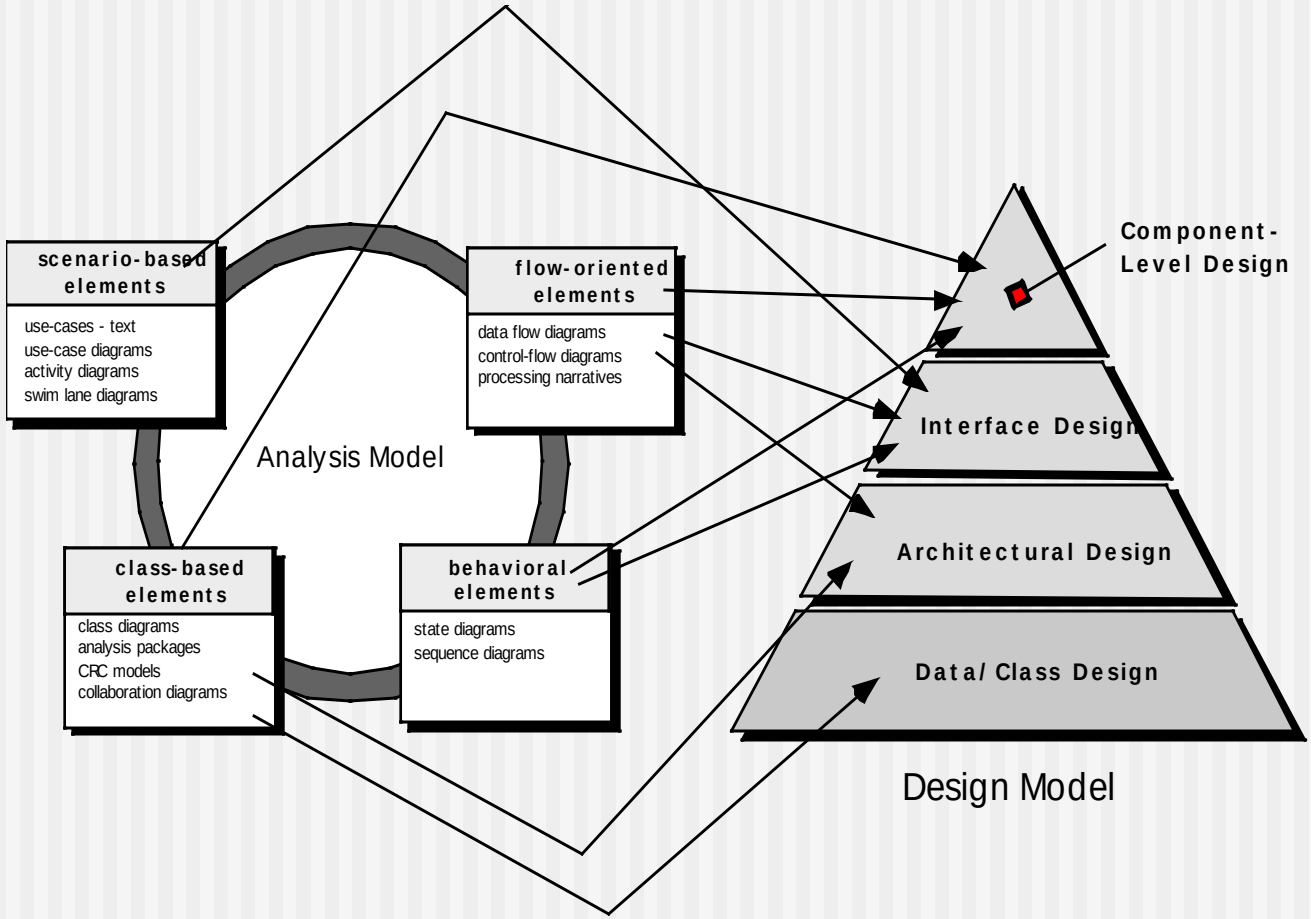
Проектиране

- Mitch Kapor, създателят на Lotus 1-2-3, представя “манифеста на софтуерното проектиране” в *Dr. Dobbs Journal.* Той казва:

Добро проектиране на софтуер означава:

- Програмата не трябва да има бъгове, които засягат нейната функционалност;
- Програмата трябва да е създадена за целите, за които е предназначена;
- Опитът при използване на програмата трябва да бъде удовлетворяващ.

Аналитичен модел-> Проектен модел



Проектиране и качество

- Проекта трябва да реализира всички изисквания включени в аналитичния модел и трябва да включва всички изисквания искани от клиента.
- Проекта трябва да бъде четаем и разбираем за тези, които ще пишат кода, които тестват и за тези, които ще поддържат софтуера.
- Проекта трябва да дава завършена картина на софтуера, определени данни, функции и поведение от гледна точка на реализацията.

Качество

- Проекта трябва да представя архитектура, която (1) е била създадена като са използвани разпознаваеми архитектурни стилове или шаблони, (2) се състои от компоненти, които имат добри проектни характеристики и (3) могат да бъдат имплементирани еволюционно
  - за малки системи проекта може да се реализира линейно
- Проекта трябва да бъде разделен на модули; софтуерът трябва да бъде логически разделен на елементи или подсистеми
- Проекта трябва да съдържа отделни представяния на данни, архитектура, интерфейси и компоненти
- Проекта трябва да води към структури от данни, които са подходящи за реализация на класовете и са създадени от разпознаваеми шаблони
- Проекта трябва да води към компоненти, които изпълняват независими функционални характеристики.
- Проекта трябва да води към интерфейси, които намалят сложността на връзките между компонентите и външната среда.
- Проекта трябва да бъде извлечен като се използва метод направляван от информация събрана по време на анализа на изискванията.
- Проекта трябва да бъде представен като се използва нотация, която ефективно показва значението му.

Принципи на проектирането

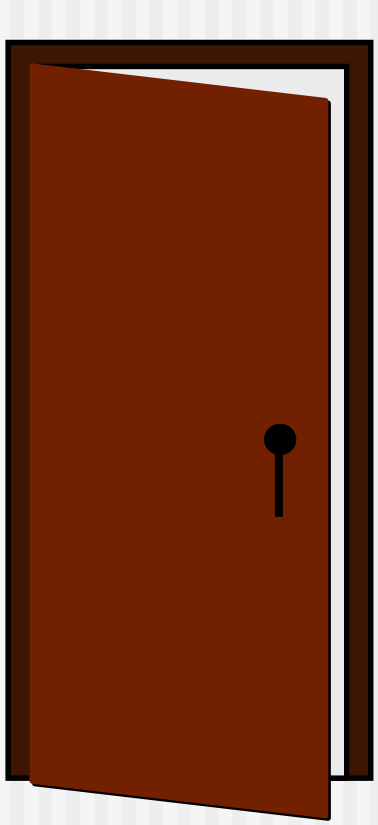
- Процесът на проектиране не трябва да страда от „тесногръдие“.
- Проектът трябва да бъде проследим в аналитичния модел.
- Проектът не трябва да преоткрива колелото.
- Проектът трябва да „минимизира интелектуалната дистанция“ [DAV95] между софтуера и проблема, който съществува в реалния свят.
- Проектът трябва да е структуриран така, че да позволява промяна.
- Проектирането не е кодиране и кодирането не е проектиране.
- Проектът трябва да бъде оценен за качество, когато се създава, не след това.
- Проектът трябва да бъде преглеждан, за да се минимизират концептуалните (семантични) грешки.

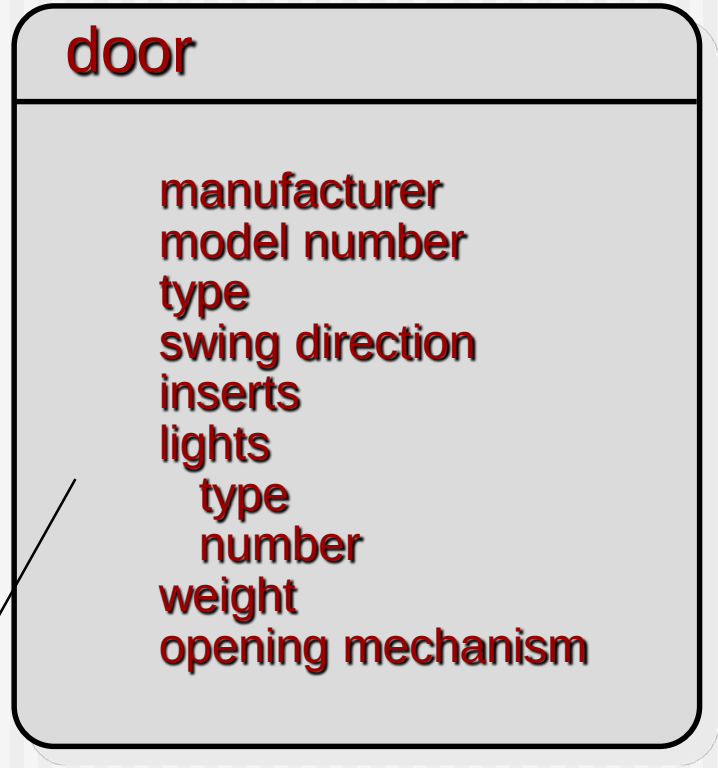
From Davis [DAV95]

Основни понятия

- Абстракции— данни, процедура, контрол
- Архитектура - цялостна структура на софтуера
- Шаблони—предава същността на доказани проектни решения
- Разделяне на части—всеки сложен проблем може да бъде по-лесно разбран, ако е разделен на малки части.
- Модулиране—раздробяване на данни и функция
- Скриване на инф.—контролирани интерфейси
- Функционална независимост—самостоятелни функции и слабо свързване
- Усъвършенстване—разработване на детайлите за всички абстракции
- Аспекти—механизъм за разбиране как изискванията засягат проектирането
- Refactoring—техника за реорганизация , която опростява проекта
- ООП концепции—класове, обекти, състояния, връзки и събития
- Проектни класове—представя детайли на проектирането, които ще позволят аналитичните класове да бъдат имплементирани

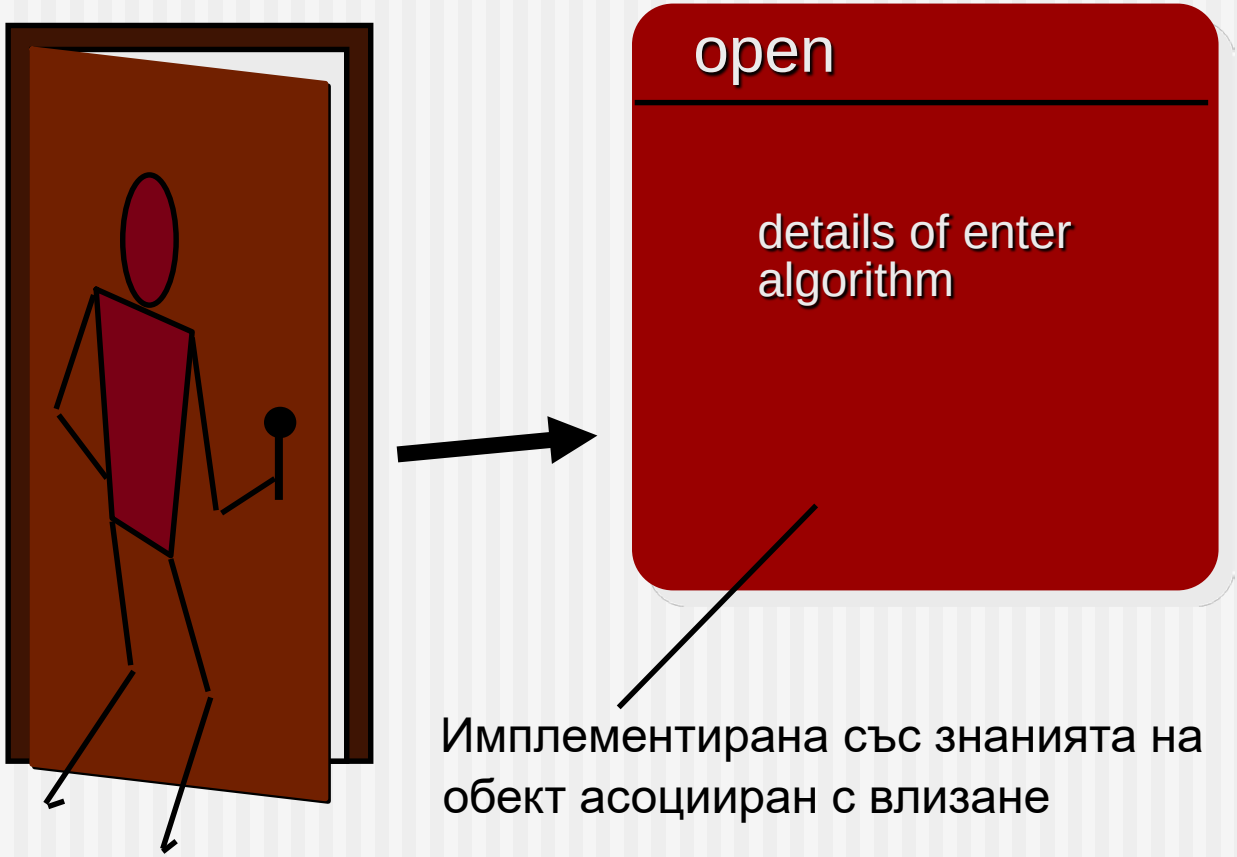
Абстракция на данните





Имплементирани като структура от данни

Процедурни абстракции



Архитектура

“Цялостната структура на софтуера и начините, по които тази структура осигурява концептуалната цялост на системата. "[SHA95a]

Структурни характеристики. Този аспект на представяне на архитектурния дизайн дефинира компонентите на системата (напр. модули, обекти, филтри) и начина, по който компонентите са пакетирани и взаимодействат една с друга. Например обектите са пакетирани да капсулират данни и процеси, които манипулират тези данни и си комуникират чрез извикване на методи.

Допълнителни функционални характеристики. Проекта на архитектурата трябва да включва описание на това как постига изисквания за производителност, капацитет, надеждност, сигурност, адаптивност, и други характеристики на системата.

Множество от свързани системи. Архитектурния проект трябва да включва повтаряеми шаблони, които са използвани при разработката на подобни системи. По същество проекта трябва да има възможност за преизползване на готови компоненти.

Шаблони

Design Pattern Template

Име на шаблон **— описва същността на шаблона накратко, но изчерпателно**

Предназначение **—описва шаблона и какво прави**

Също известен като **—списък на синоними за шаблона**

Мотивация **—представя пример на проблема**

Приложение **—описва специфични проектни ситуации, в които шаблона е приложим**

Структура **—описва класовете са необходими за имплементация на шаблона**

Участници **—описва отговорностите на класовете, които участват в имплементацията на шаблона**

Сътрудничество **—описва как участниците си сътрудничат, за да изпълнят отговорностите си**

Последствия **—описва „силата на проекта“, която носи използването на шаблона и възможните компромиси, които трябва да се вземат предвид при имплементация на шаблона**

Свързани шаблони **—референции свързани с проектния шаблон**

Разделяне на части

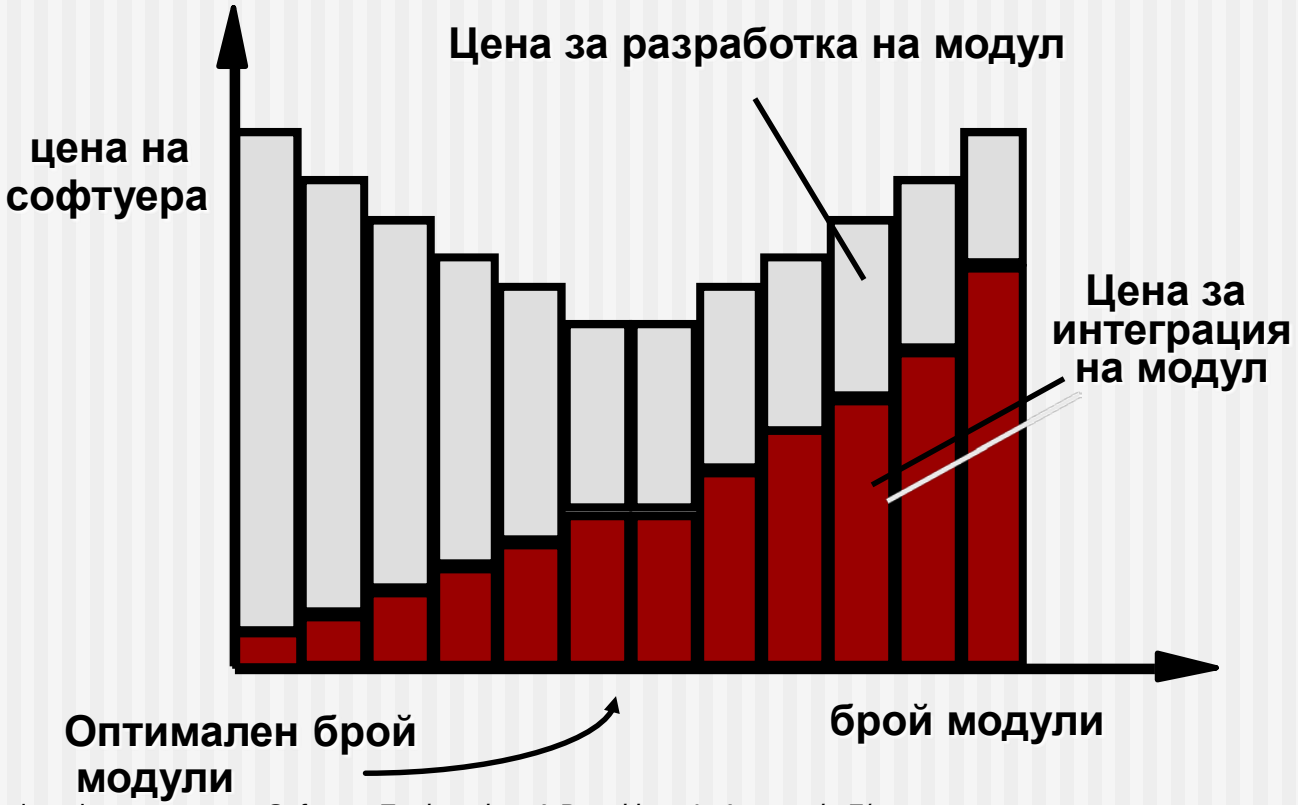
- Всеки сложен проблем може да бъде лесно обхванат, ако се раздели на части, които могат да бъдат решени и/или оптимизирани независимо
- Частта (*concern)* *е част от поведението, която е специфицирана като част от модела на изискванията на софтуера*
- Чрез разделяне на частите на по-малки задачи и следователно на по-лесно управляеми чати, проблема отнема по-малко усилия и време за решение.

Модулиране

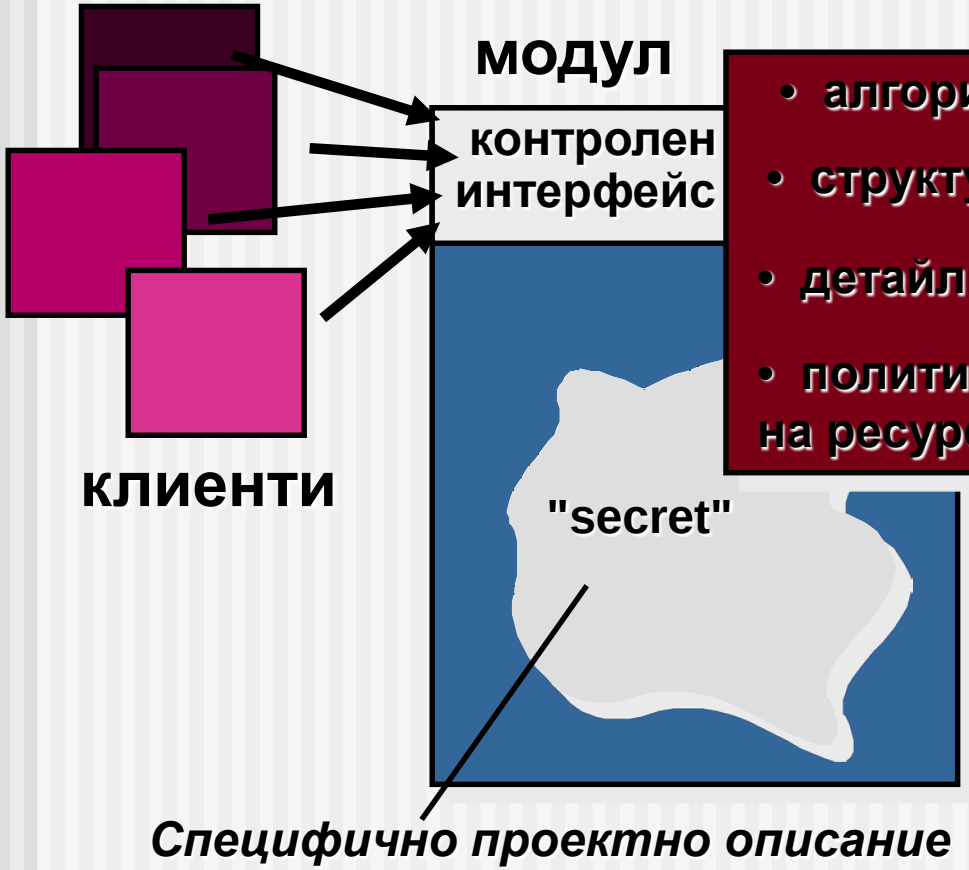
- „модулирането е атрибут на софтуера, който позволява програмата да бъде управлявана" [Mye78].
- Монолитния софтуер (голям софтуер съставен от един модул) не може да бъде лесно разбран от софтуерните инженери.
- В почти всички интерфейси е необходимо да разбием проекта на много модули, като се надяваме това да направи разбирането по-добро и като следствие намаляване на цената за софтуера.

Модулиране: компромиси

Какъв е„правилния" брой модули
за специфичен софтуерен проект?



Скриване на информация

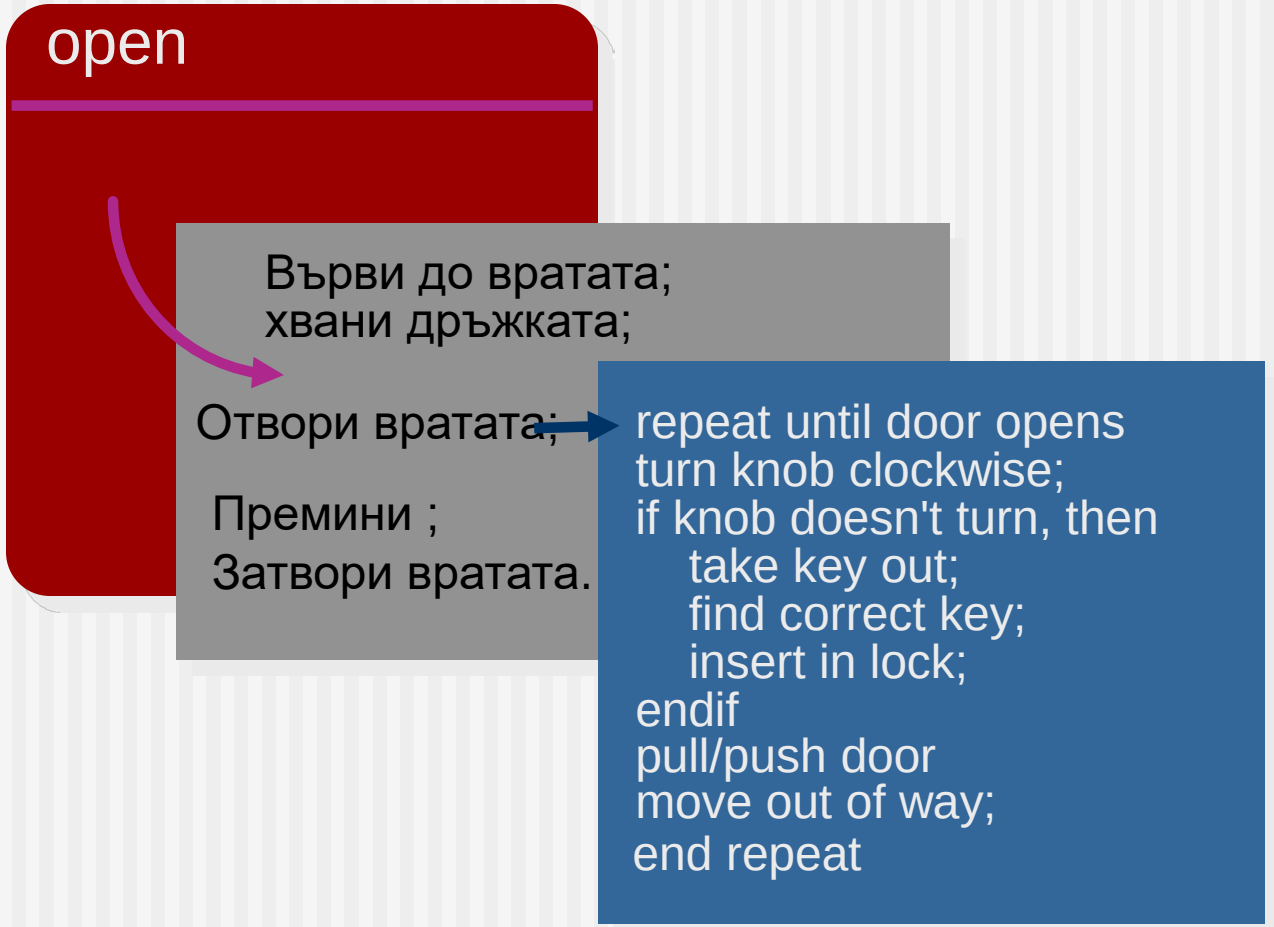


- алгоритми
- структури от данни
- детайли за външния интерфейс
- политика за разпределение на ресурси

Защо скриваме информация?

- намаля вероятността от странични ефекти
- ограничава общото въздействие на локални проектни решения
- набляга на комуникация през контролирани интерфейси
- не толерира използването на глобални данни
- води до капсулация – което е атрибут на високо качество на проектирането
- резултата е по-високо качество на софтуера

Постъпково усъвършенстване



Функционална независимост

- Функционална независимост се постига, чрез разработване на модули с конкретна функционалност и „нежелание“ за прекалено взаимодействие с други модули.
- *Съгласуваността (Cohesion*)е индикация за силна компонента.
  - Силно съгласуваният модул има една задача , изискваща малко взаимодействие с други компоненти на системата.
- *Свързване (Coupling)* е индикация за относителна взаимозависимост между модулите.
  - Свързаността зависи от сложността на интерфейсите между модулите.

Refactoring

- Fowler [FOW99] дефинира рефакторинг по следния начин:
  - „Refactoring е процес на промяна на софтуерната система, така че не променя нейното поведение, а подобрява вътрешната му структура.”
- Когато правим refactoring, съществуващата структура на кода се изследва за:
  - неизползвани елементи от кода
  - неефективни или ненужни алгоритми
  - лошо създадени или неподходящи структури от данни
  - или слаби места в структурата на кода , които могат да бъдат подобрени.

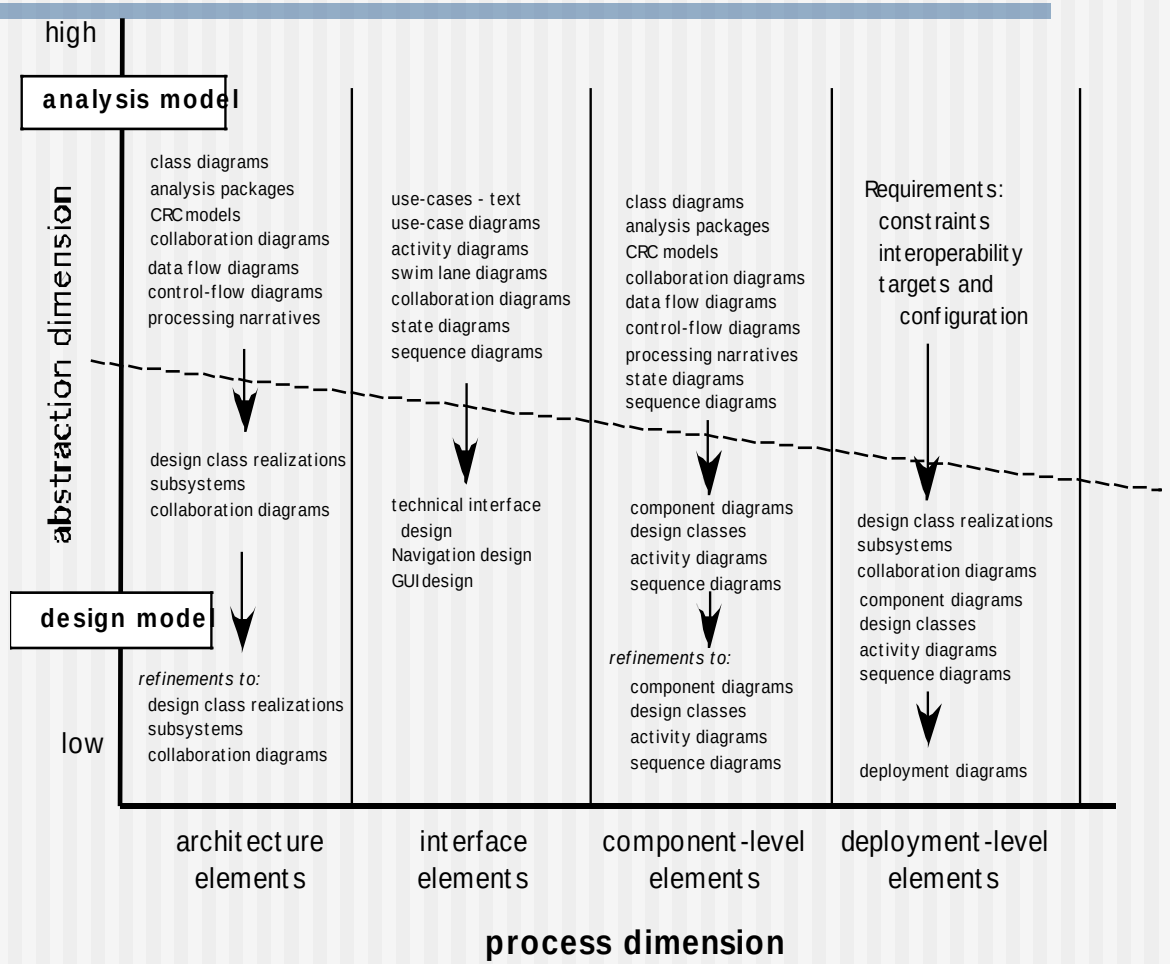
Концепции на обектно-ориентираното проектиране

- Проектни класове
  - Entity класове
  - Boundary класове
  - Controller класове
- Наследяване
- Съобщение
- Полиморфизъм

Проектни класове

- Аналитичните класове се усъвършенстват по време на проектирането, за да се превърнат в entity classes
- Boundary classes се разработват по време на проектирането, за да се създадат интерфейсите, които потребителя вижда и взаимодейства със софтуера.
  - Boundary classes се проектират с отговорността да управляват начина, по който entity обектите да се показват на потребителите.
- Controller classes се проектират да управляват:
  - Създаване или обновяване на entity обекти;
  - инстанцирането на boundary обекти, тъй като те получават информация от entity обектите;
  - сложна комуникация между множество от обекти;
  - валидация на данни при комуникация между обекти или между потребител и приложението.

Проектен модел



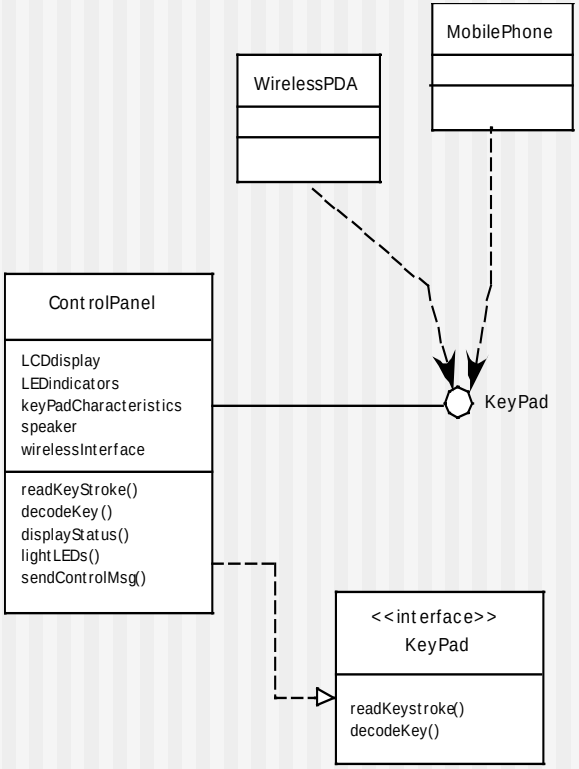
Елементи на проектния модел

- Данни
  - Модел на данните --> структури от данни
  - Модел на данните--> архитектура на базата данни
- Архитектурни елементи
  - Приложен домейн
  - Аналитични класове, техните връзки, сътрудничество и поведение се трансформират в проектна реализация
  - Шаблони
- Интерфейсни елементи
  - Потребителски интерфейс (UI)
  - Външни интерфейси към други системи, устройства, мрежи или други производители или консуматори на информация
  - Вътрешни интерфейси между различни проектни компоненти.
- Компонентни елементи
- Deployment elements

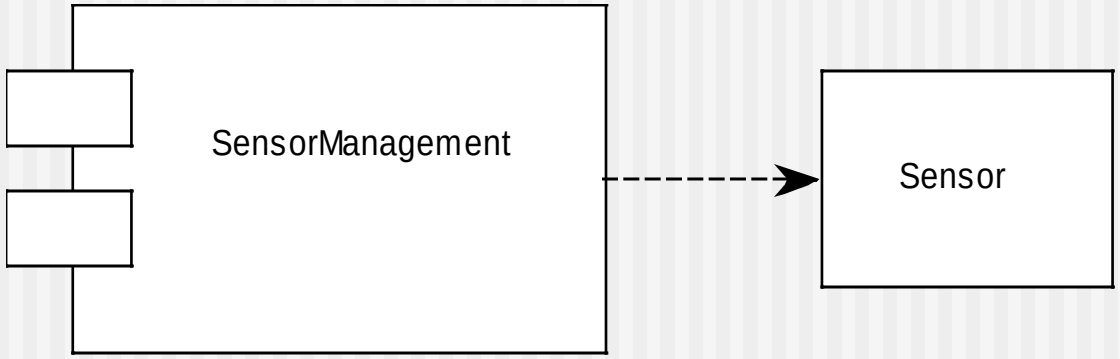
Архитектурни елементи

- Архитектурния модел [Sha96] е разделен на три:
  - Информация за приложния домейн за софтуера, който да бъде разработен;
  - Специфични елементи на модела на изискванията такива като диаграми на потока на данните или аналитични класове, техните връзки и сътрудничество
  - Налични архитектурни шаблони

Елементи на интерфейса



Component диаграми



Deployment диаграми