* **Software Design and Architecture** - Design ne ajuta sa ne putem impartasi idea cu alti software developers
* Un good design face ca software:

1. Sa fie Mai usor de implementant
2. Sa permita modificari viitoare fara a trebui sa modificam prea mult din code
3. E mai usor de testat

* **Software Designer** – adesea e responsabil sa determine o software solution pentru o specfifica problema prin a proiecta detaliile componentelor individuale si responsabilitatile lor.
* **Software Architect** – este responsabil sa se uite la intregul sistem si sa aleaga frameworks potrivite, data storage, solutii si sa determine cum componentele interactioneaza intre ele. El este ca o legatura intre client si software si development team. Anume el analzieaza problema clientului si vine cu aspecte tehnice despre cum urmeaza sa rezolve problema. Iata de ce, el e ca un arhitect de cladiri.

****

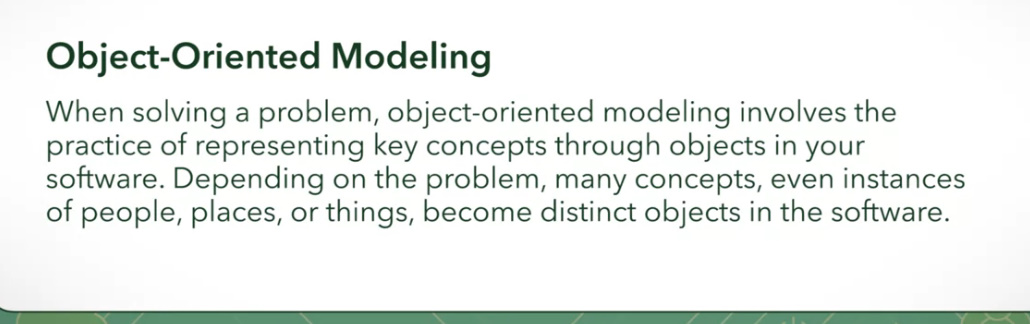
E ceva asemanator ca crearea unei cladiri

**Arhitect** se focuseaza pe structura majora a cladirii

**Interior Designer** se focuseaza pe spatiile mici din cladire, particulare

* **Software Design** – poate fi vazut ca un proces de a transforma dorintele si cererile clientului in working code care este durabil, maintenable pe mult timp si care poate evolua si poate fi integrat intr-un system mai larg
* **Software Architecture** – se bazeaza pe a intelege bussiness problems a clientului, de a gasi solutie la problema sa

**Object Oriented Modeling**



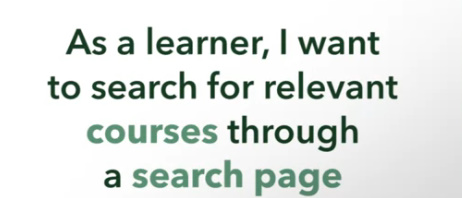
**Software Requirements, Conceptual and Technical Designs**

* OOD include 3 etape de baza:

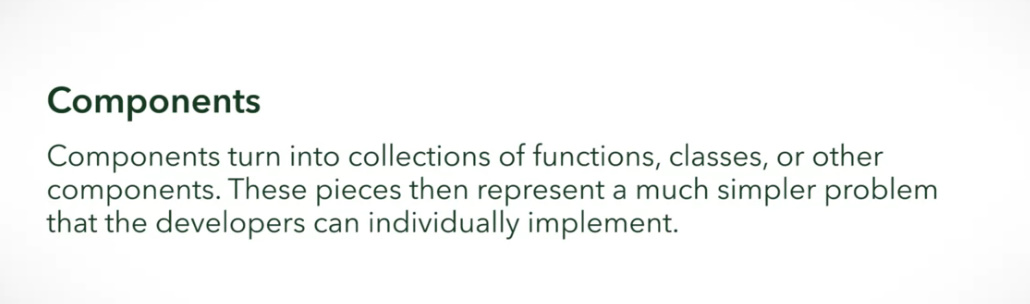
1. Stabilim requirments
2. Conceptual Design
3. Tehnical Design

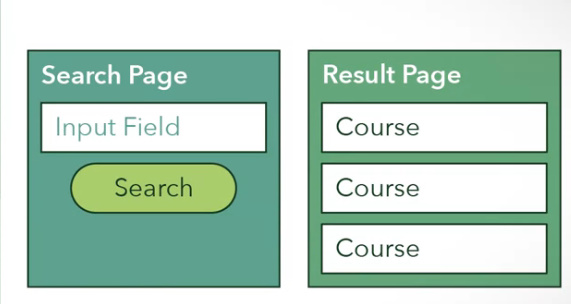
* **Eliciting(obtinerea) details** – faza in care stabilim detaliile necesare pentru a intelege ce urmeaza sa proiectam, ce detalii vor fi, si asta spune clientul deobicei
* **Conceptual design** – Stabilim pe baza la requirments toate conmponents si conections si responsability la components.El este bun pentru a face userilor clar ce vrem sa facem. Foloseste **Object Oriented Analyse**.Produce Conceptual Mockups

De ex:



* **Conceptual Mockups** – ofera solutii, in mdo grafic, despre cum vor fi satisfacute cerintele. Stabilim aici **components** si **connections.** In software, poate fi un model de proiect realizat in tooluri speciale, ca figma sau niste diagrame, scheme etc. Ar fi ca si cum am face proiectul pe deasupra, o schema, de ex, cand construim o casa, facem o schema, cu camerele, intrarile, baia, scarile etc. dar nu ne ocupam, de ex, de scheme pentru unde vor fi tevile, firele electrice etc., asta deja intra in tehnical design, acolo se iau detaliile in considerare

****

****

* **Responsability –** fiecare component are un task ce trebuie sa il faca, numit responsability
* **Tehnical design**– produce Tehnical Diagrams.Anume crearea la Tehnical design e rolul la **Object Oriented Design**. Stabilim ce responsability sunt facute. Specificam detaliile tehnice ale fiecarui component. Fiecare component e despartit in componente tot mai mici si mai mici pana il putem cream din unitati mici de tot. Asta se face deja dupa ce am facut conceptual design si tot pe baza lui se face. El este bun pentru ca developerii sa inteleaga ce trebuie facut.
* **Techinal Diagrams** –
* Defapt, conceptual mock-ups si tehnical diagrams sunt artefacte ce trebuie produse

**Cerinte la soft ce trebuie obtinute cu un bun design in OOD:**

- **Flexible** – putem adauga in soft noi capabilitati

- **Reusable** – codul actual poate fi folosit in noi chestii, deci nu trebuie sa cream deja lucruri ce au fost create, ci sa le folosim pe cele existente in ceea ce cream

- **Maintainable** – code usor de inteles si modificat

**Process and Implementation**

* **Process** – a lua o problema si a crea o solutie care implica software
* Un process este iterativ.
* **Software implementation** – Iteratiile processului constau in a lua cerintele bazate pe problema si sa le folosim conceptual design mockups si tehnical design diagrams care trebuie inca sa treaca teste.Acest proces e repetat pentru fiecare set de cerinte aparute pentru a crea solutia pentru proiect.

**Compromise in Requirments and Design and Quality Attributes**

* uneori, este nevoie de a face anumite sacrificii, din partea la client, pentru a asigura calitatea
* Qulity Attributes pot fi de ex Security, Performance, Convience etc.
* Unele chestii, desi cerute de client, pot scadea din quality attributes, si atunci trebuie sa vb cu client ca sa faca anumite sacrificii sau compromise pentru a mentine qulaity atttributes ridicate si balansate.
* De ex, daca el vrea mai multe lacate la o usa, asta va creste Security, dar va scadea din Convience. Trebuie cumva sa gasim echilibrul la atribute.

**User Story**

* Este o tehnica de a exprima requirments din perspectiva la end user in limbaj natural
* Are cam asa model:

As a \_\_\_\_\_\_, I want to \_\_\_\_\_\_ so that \_\_\_\_\_\_.

* Intai punem cine face actiunea, apoi ce anume vrea sa obtina userul si in “so that” de ce userul vrea acest rezultat. Dar, “so that” poate fi omis daca si asa e clar ce anume vrea sa faca.
* Un exemplu:

As an **online shopper**, I want to add an ***item*** to my ***shopping cart***, so that I can purchase it.

* Vedem ca putem stabili prin substantive care vor fi componentele si deci obiectele
* Verbele ne ajuta sa stabilim requirments:

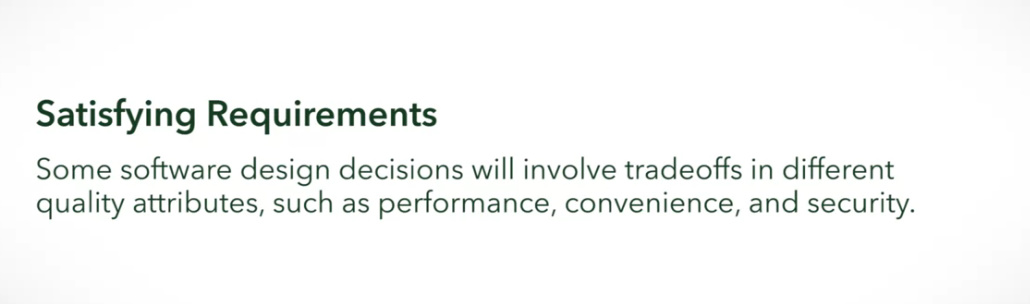
As an online shopper, I want to ***add*** an item to my shopping cart, so that I can ***purchase*** it.

* Connection deja le cam stabilim singuri. Aici e clar ca un shopper e asociat un un shopping cart

**Categories of objects in design**

* **Entity Objects** – adesea corespund unor entitati din lumea reala.
* **Boundary(la limita) objects** – stau la limita intre 2 sisteme. Poate fii un object care are de a face cu un alt software system ca un obiect care obtine date din internet. Poate fi si un obiect cu responsabilitatea de arata informatii unui user si de a lua input de la el.Orice obiect care interactioneaza cu un alt sistem, ca un user, alt software sistem, Internet etc. este considerat boundary object
* **Control Object** – sunt obiecte responsabile de coordonate. Le descoperim cand un obiect are mai multe obiecte in el si ar fi folositor sa avem un alt obiect care le controleaza.

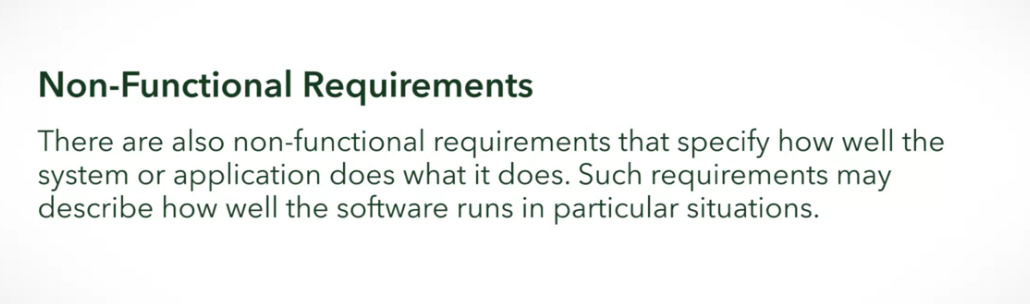
**Qualities concurente si sacrificii**



Requriments sunt luate de la client si sunt elementare pentru un proeict.



De ex, trebuie sa cream un buton de a permite unui user sa dea downloand la un clip video



De ex, stabilim o viteza anumita si o limita la memorie

Alt exemplu, toate masinile indeplinesc Function Requirments de a oferi transportation, dar fiecare model de masina indeplinesc non functional requirments diferit, ca accelarea, viteza etc.

* **Qualities to satisfy for software, alt tip de non functional requirments**



# Record, Organize, and Refine Components

* Conceptual design,care este primul pas, implica stabilirea la Components, Connections si Responsabilities
* Totusi, trebuie cumva sa le putem reprezenta.
* Reprezentarea anume o facem prin CRC Cards

C – class (pentru components)

R - responsability

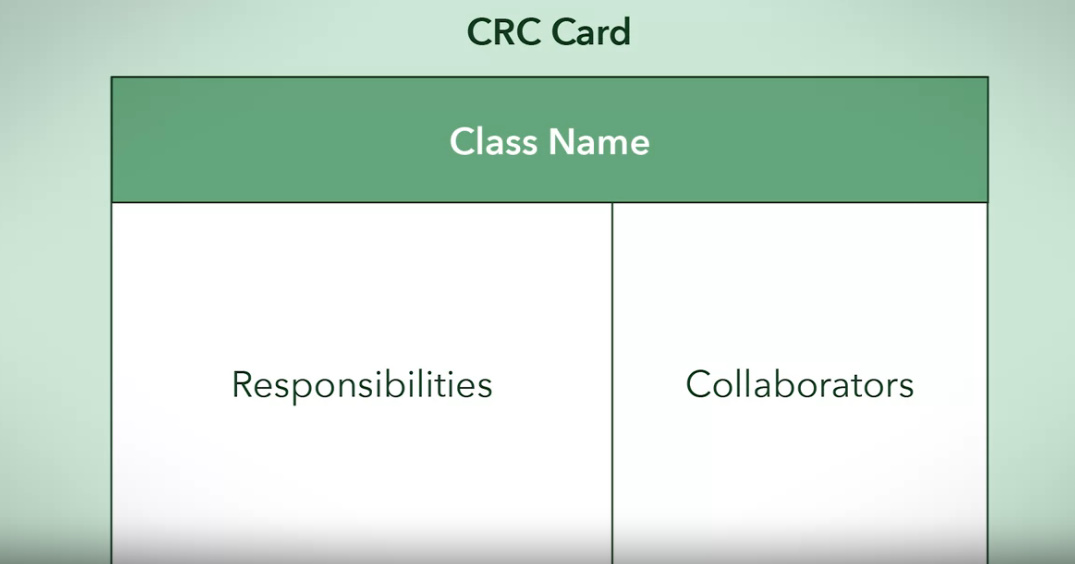
C – collaborator (connection)

* Un CRC Card are 3 elemente:

Class Name – e component name

Responsabilities

Collaborattors – e pentru connections si sunt clase cu care clasa data interactioneaza



* Exemplu pentru Aparat de extragere a banilor la banca:

