3 veckorsuppgift

Github från itslearning

Skall gissa vilka use-cases som fnns i uppgiften.

Baserat på denna listan vi kommer fram till skall man beskriva vad man förväntas kunna göra med systemet. Prioriteringslista av dessa cases.

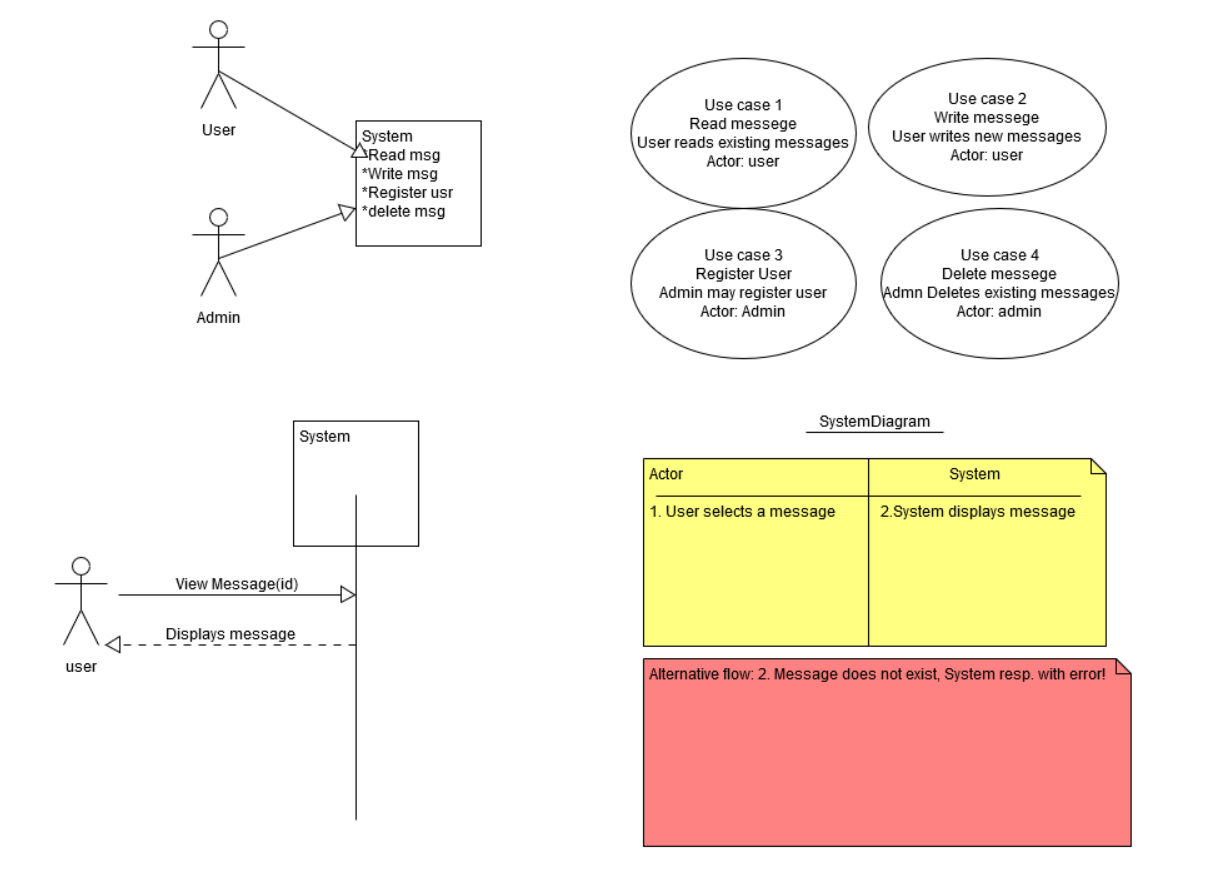
Välj ut vissa cases som man kan tänkas hinna genomföra under kursens gång.

Gruppuppgiften görs i grupper om 3-5.

START KEDJA (Design för online disskusionsforum)

1.Use cases:

* Read msg
* Write msg
* Register usr
* Delete msg
* …



**Föreläsning 2**

User stories – **Template**: As a <***User type***> I want <***Some goal***> so that <***Reason***>.  
Det finns något som kallas för ***Epic***, en Epic är samma sak som en user story, men mycket större.  
Ett fördelaktigt sätt att beskriva arbetsuppgift och vad man vill uppnå i systemet.   
Används oftast i kortare sammanhang, t.ex. för att fylla en sprint.

Ett kravdokument för systemet skall helst inte vara för långt. Man vill gärna fylla det med få liknande stories för att kunna enkelt se vad man måste arbeta med i detta NU.

Skillad på use case och user stories.

Use case: **How?**

User Stories: **What? Why?**

User stories är kraven för en feature.   
As a regular user I want to be able to view my personal calendar and recent

notifications

so that I can make sure that a new notification does not

double-book me on an event

User Stories har så kallade ”Conditions of satisfaction” Man ska kunna skriva en user Story p en post-it lapp med beskrivning. På baksidan av samma lapp ska man kunna se vad man behöver kunna göra för att se så att ”kravet” är uppfyllt.

**Use Case:** View Calendar and Notifications. (And) ska helst inte finnas med.

**Actors:** System Users

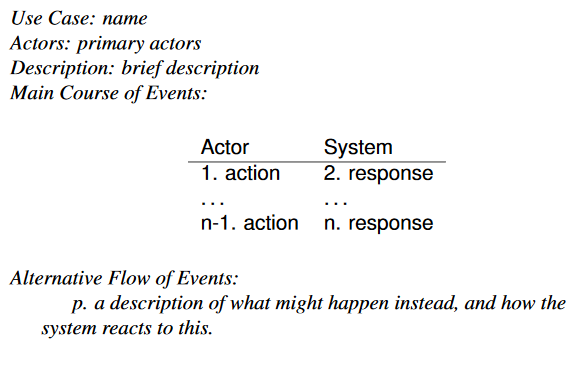
**Description(Use Case description):**

*A user requests to view their personal calendar.*

*The system displays the users’ personal calendar.*

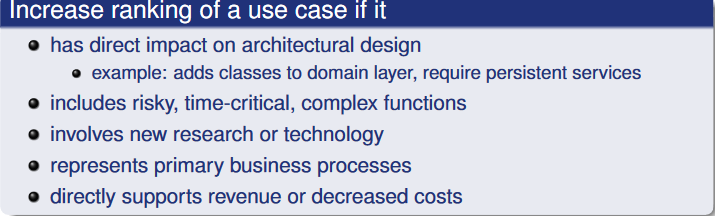
*A user requests to view their recent notifications.*

*The system displays the users’ recent notifications.*

Sträva alltid efter det lättaste sättet att beskriva ett use Case, Helst så att man slipper använda sig av en user Story. Use case ska alltid börja med ett drömscenario. När detta är klart kan man börja med felhantering av systemet. T.ex. låna en bok, boken finns i systemet. Händelsen att boken inte finns i systemet tar man hand om senare.

Tabell över ett use case med vilket kravet är samt hur systemet skall reagera.

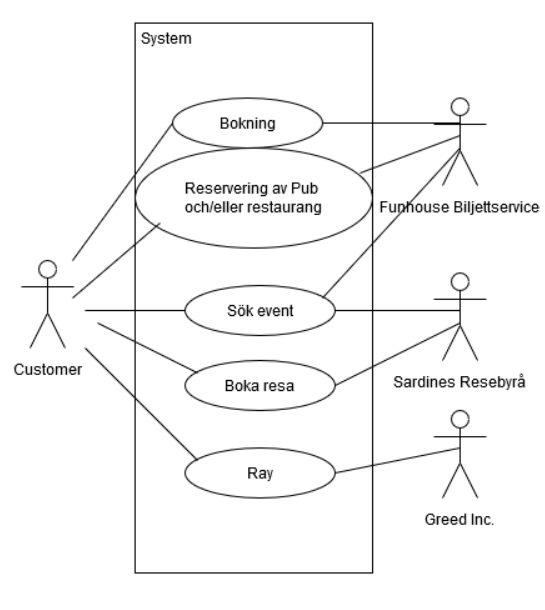
.

STRÄVA efter de mest viktiga egenskaperna för ditt system vid val av Use cases. Se till att grunden är färdig innan man går vidare på detaljnivå. Det finns vissa saker som ökar ett Use case’s prioritet.

Det finns olika prioriteringsmetoder:

* Scored(numrerad prio)
* Discrrete(Hög, Medium, låg)
* Simple Ordering(Bubble sort?)
* MoSCoW(Must have, Should have, Could have, won’t have)
* Cumulative Voting

En egen tanke: Vid genomförandet an scrum Kör man med en mix av discrete och MoSCoW.

**Use Case diagram:**

**Modelling Structure**

* Dynamisk och statisk struktur.
* Statisk: summan av alla dynamiska beteende
* Dynamisk: tar hänsyn till ett attribut.

Attribut och koncept.

Var hittar vi koncepten(leta efter substantiven)

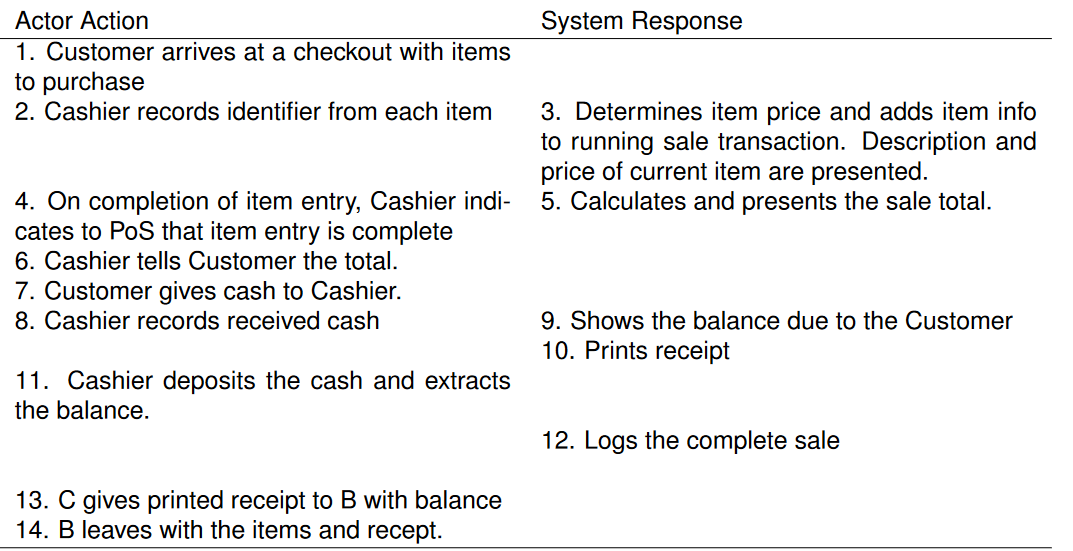
* Fysiska objekt
* Geografiska platser
* Transaktioner
* Behållare(saker som innehåller sake t.ex. flygplan innehåller stolar)
* Händelser
* Beskrivning av saker
* Records, contract(t.ex. kvitton, biljetter)

Försök att hela tiden a en uppdaterad domänmodell. Så fort man upptäckre ett problem i domänmodeller bör man korrigera och lägga till/ ta bort det som behövs för att hålla domänmodellen relevant till projektet.

Kardinalitet mellan koncept. Behöver inte förklaras…Hur ser förhållandet/relationen ut mellan två olika koncept. T.ex. bok och kund.

**Dynamiskt beteende**

**Systemhändelse:**



Ett systemsekvensdiagram för varje use case, sen gör man ett systemsekvensdiagram för varje systemhändelse. Försök att se till att varje sekvens i diagrammet har en optimal fördelning av ”arbetsuppgifter”. På så vis kan man minska antalet kopplingar mellan de olika sekvenserna.

**State diagram(tillståndsdiagram):**

Ett sätt att beskriva en händelse i en viss ordning. När man inte riktigt vet hur hela sekvensen ser ut men man har klart för sig vad man vill ska hända. typ..

**Nested states**

System med komplexa tillstånd, man vill veta vilka tillstånd som finns och vad man kan göra där för att komma vidare till nästa tillstånd. Det finns ingen koppling mellan dessa och hur man ska bygga sitt program.

Vid skapandet av ett tillståndsdiagram, tänk efter, var detta ett tillstånd eller var det en händelse. Det är tillståndet vi är ute efter.

I ett State diagram pekar pilen på de plats där metoden skall ligga.   
Det är enbart de tillstånden i ett state diagram vars klasser vi behöver ändra eller bygga på och det är enbart de funktioner som används i ett state diagram som behövs för att genomföra vårt use case.

**Design Patterns:**

**Reesponsibility driven design**

*Resp for DOING*

* Vilken klass har ansvar för att utföra beräkningar?
* Vilken klass har ansvar för att skapa vissa objekt
* Vilken klass har ansvar för att koordinera olika objekt?

*Resp for Knowing*

* Vilken klass har koll på privat data?
* Vilken klasss håller koll å temperaturer etc..

Levels of patterns

**Different levels:**

* Architecture
  + Systems, subsystems
* Design
  + Klasser och/eller grupper av klasser
* Idiom
  + En hel klass eller funktioner i en klass
* GRASP
  + Låg-nivå design. Finns sammanfattade i kursboken.

GRASP patterns:

* Information expert -
* Creator
* Controller
* Low couplng – lågt antal kopplingar mellan klasser.
* High Cohesion – Väl avgränsade ansvarsområde mellan klass och andra klasser
* Polymorphism – En basklass med flera subklasser, vanligt förekommande.
* Pure fabrication
* Indirection – Man tar en mer indirekt väg för att ta hand om ett problem på ett mer löst sätt(skapar fler kopplingar men ett system som är mer lättskött)
* Protected Variations

Design patterns + Bibliotek.

