Software Design Essentials

Opgave 1

1 Use Cases

Doel

In een Use Case richten we ons op functionele vereisten van een systeem. Bekijk de onderstaande vereisten en bepaal welke geschikt zijn als Use Case.

Stap 1: Lees elke vereiste

Stap 2: Bepaal of het een echte Use Case is

Ja → De vereiste beschrijft een interactie tussen een actor en het systeem en levert een waardevol resultaat.

Nee → Het is een niet-functionele vereiste of te technisch.

Welke vereisten zijn een Use Case?

De gebruiker logt in op het systeem. (Login is meestal een preconditie, geen op zichzelf staande Use Case.)

1. De klant plaatst een bestelling
2. De student registreert zich voor een vak.
3. De gebruiker logt in op het systeem.
4. Het systeem moet 1000 aanvragen per seconde kunnen verwerken.
5. De klant plaatst een bestelling en kiest een betaalmethode.
6. Het systeem slaat automatisch elke minuut een back-up op.
7. Een docent voegt cijfers toe voor een ingediende opdracht.
8. De gebruiker kan zijn profielinstellingen wijzigen.
9. Een bestelling wordt automatisch naar het magazijn doorgestuurd.
10. De website moet laden binnen 2 seconden.
11. Een klant bekijkt zijn bestelgeschiedenis.
12. Het systeem moet voldoen aan GDPR-voorschriften.

13. Een beheerder verwijdert een gebruiker uit het systeem.
14. Het systeem genereert automatisch een factuur.
15. Het systeem verzendt een notificatie bij een nieuwe bestelling.
16. Het betaalsysteem valideert een transactie.
17. De student downloadt zijn studiepuntenrapport.
18. Een arts raadpleegt het patiëntendossier.

2 ATM Adventures!

Je hebt zojuist een contract getekend als ontwerper van de bankautomaat van de toekomst! Jouw missie?

- Bepaal wie (actoren) met de ATM interageert
- Identificeer welke acties (use cases) mogelijk zijn
- Teken een UML Use Case Diagram om alles in kaart te brengen!

Scenario: De Slimme ATM 3000

Je krijgt de taak om een moderne ATM te ontwerpen met extra functionaliteiten. Dit is geen gewone bankautomaat meer – hij kan veel meer! Naast geld opnemen en saldo bekijken, biedt hij de volgende opties:

Standaard bankfunctionaliteiten

- Geld opnemen
- Saldo bekijken
- Geld overschrijven tussen eigen rekeningen
- Geld overschrijven naar andere rekeningen
- Facturen betalen

Extra mogelijkheden

- Aankopen van diensten of producten (bv. belwaarde kopen)
- Cash geld en cheques op je rekening zetten

Onderhoud en beheer

- Aanvullen van geld in de ATM
- Software-updates uitvoeren
- ATM status controleren en storingen melden

2.1 Stap 1: Bepaal de Actoren

Wie gebruikt de ATM en voert acties uit? Denk aan:

- Bankklant
- Geldtransporteur (aanvullen van geld in de ATM)
- Technische Beheerder (software-updates en onderhoud)
- Bank Systeem (externe actor, verwerkt transacties)

2.2 Stap 2: Identificeer de Use Cases

Een Use Case beschrijft een functie van de ATM die een waardevol resultaat oplevert voor een actor.

2.3 Stap 3: Teken het Use Case Diagram

Maak een UML Use Case Diagram waarin je:

- Actoren plaatst (mensen of systemen die met de ATM werken).
- Use Cases tekent als ovalen met de juiste functionaliteiten.
- Verbindingen aangeeft tussen actoren en de bijbehorende Use Cases.

3 Schrijf een Use Case Specification voor het inschrijven op een vak

Situatie:

De hogeschool heeft een nieuw studentenportaal waarin studenten zich zelf kunnen inschrijven op vakken voor het komende semester. Dit proces moet soepel en zonder fouten verlopen, en er moeten ook uitzonderingen voorzien worden (zoals vakken die volzet zijn of studiepunten die overschreden worden).

Denk na over:

- Welke stappen doorloopt een student om zich in te schrijven?
- Wat als een vak volzet is?
- Wat als de student onvoldoende studiepunten over heeft?

Jouw uitdaging:

- Bepaal de actoren (Wie gebruikt het systeem?).
- Schrijf een duidelijke en gestructureerde Use Case Specification voor de functie "Student schrijft zich in op een vak".
- Denk aan alternatieve scenario's!

4 Niet-Functionele vereisten

Naast de functionele vereisten van een softwaretoepassing, spelen niet-functionele vereisten een cruciale rol. Deze bepalen hoe goed het systeem werkt in plaats van wat het doet.

Opdracht:

Geef drie voorbeelden van niet-functionele vereisten die geen directe functionaliteit beschrijven, maar wél essentieel zijn voor de softwarekwaliteit.

Denk aan aspecten zoals:

- Prestaties (bv. responstijd, verwerkingssnelheid)
- Beveiliging (bv. gegevensversleuteling, toegangscontrole)
- Gebruiksvriendelijkheid (bv. mobiele compatibiliteit, intuïtieve interface)

Schrijf drie concrete niet-functionele vereisten op die relevant kunnen zijn voor softwareontwikkeling!

5 Domeinmodel voor een Schooladministratiesysteem

We gaan een domeinmodel opstellen voor een eenvoudig schooladministratiesysteem voor het middelbaar onderwijs. Laten we ons concentreren op basisentiteiten en hun onderlinge relaties.

Stap 1: Identificeer de basisentiteiten en hun attributen

Hieronder staan de belangrijkste entiteiten met hun kenmerken:

Entiteiten:

- School heeft een: Adres, Telefoonnummer, Directeur
- Klas heeft een: Naam, Jaar
- Student heeft een: Voornaam, Achternaam, Geboortedatum, Adres, Email
- Leraar heeft een: Voornaam, Achternaam, Vakgebied
- Vak heeft een: Naam, Beschrijving

Stap 2: Definieer de relaties tussen de entiteiten

- Een school heeft meerdere klassen.
- Een klas heeft meerdere studenten.
- Een klas heeft meerdere leraren.

- Een leraar kan meerdere klassen lesgeven.
- Een klas kan meerdere vakken hebben.
- Een vak kan in meerdere klassen worden onderwezen.

Kan je extra entiteiten bedenken die nuttig kunnen zijn in dit systeem? Teken het domeinmodel als een UML-diagram.