Hjemme-Eksamen

Emnekode: *MPSE-2201*

Emnenavn: *Systems Design and Engineering*

Kandidatnr:*130*

May 31, 2019

Contents

List of Figures

0.1 Modeller og Diagrammer

Modell i Systems Engineering beskriver og representerer et system. En modell kan ta for seg flere punkter som for eksempel, planlegging, implementering av systemet, dokumentering, krav og struktur på systemet. Modell beskriver hvordan systemet vil oppføre seg når den er ferdig laget. Vi kan analysere systemet ut fra modellen som er laget. Siden modell representerer et system så fokuserer systems ingeniører på å lage en god modell for systemet slik at det viser hele systemet hvordan funksjonaliteter fungere osv. En modell kan defineres på ulike måter og det er laget for å håndtere kompleksiteter.

Diagrammer viser en visuell modell av et system og samspill mellom komponenter. Det finnes mange ulike diagrammer som for eksempel kontekst diagram som er en høynivå diagram og viser all eksterne fakturer som samspiller(interact) med systemet. Kontekst diagrammen brukes i start fasen når man lager et system og det er på grunn av at man skal se hvem som interacting med systemet og se på miljøet rundt systemet.

0.1.1 V Modell

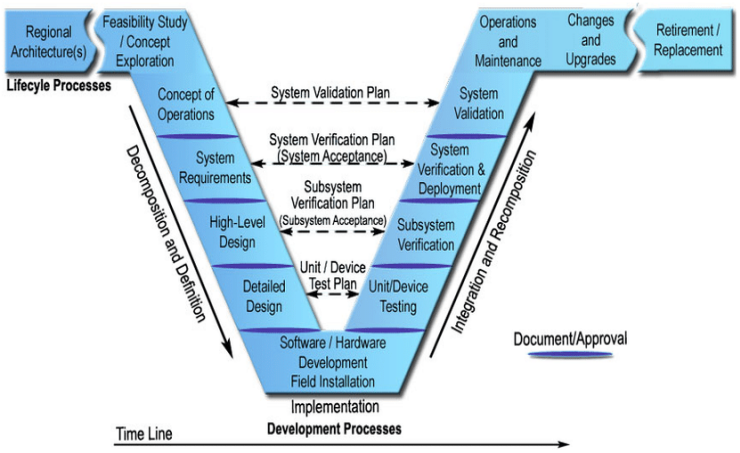

Figure 1: V modell,
hentet fra[4]

Vee modell er modell som grafisk representerer et systemsutviklings livssyklus og denne modellen er også kalt for verifisering og validerings modell eller

Waterfall modell. Denne modellene har forskjellige faser hvor det hver av de fasene må bli ferdig før man går videre til neste fase og viser sammenheng mellom faser og dokumenttyper. V modellen starter øverst på venstre fra kundens ønske og fortsetter mot høyre for godkjenning av systemet og de forskjellige stegene er paralle. Den høyre side av V modellen viser dekomponering av krav og oppretting av systemsspesifikasjoner, mens venstre viser validering og integrasjons delen.

0.1.2 CAFCR

I prosjektet brukte vi CAFCR modellen og denne viser fem ulike måter vi kan se på systems engineering på. De forskjellige metoder er som følgende: Customer objectives,application, functional, conceptual and realization.

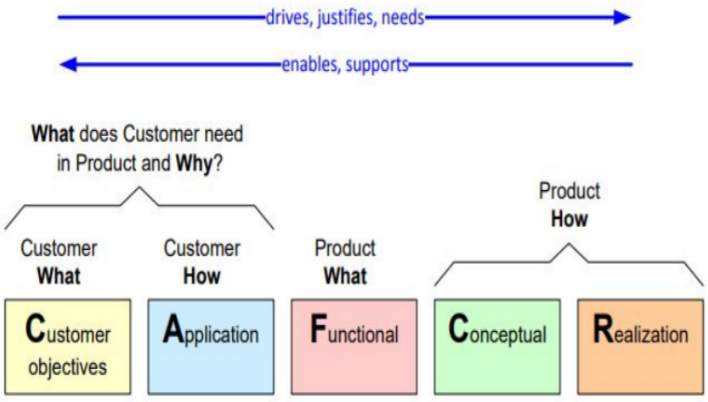

Figure 2: CAFCR modell,
hentet fra [3]

Alle de har ulike hensikter å oppfylle. Denne modellen viser hva kunden vil oppnå og hvordan kunden vil realisere det målet sitt. Funksjonell bekriver HVA av produktet som inkluderer ikke funksjonelle kravene. Hvordan av produktet er beskrivet inn i konseptuelle og realisering tenkning og det er delt i to på grunn av stabilitet. Målet med konseptuell er å kartlegge de ulike aspektene av systemet og målet med realization view er å analysere realtion aspektet av systemet. Siden teknologi endrer seg konstant og gamle blir utdatert og derfor konseptuell view is maintained over lengre tidsperiode enn relation.

0.1.3 Decision making matrix

Dette er en matrise som er basert på kriterier og brukes til å ta beslutninger mellom flere alternativer ut ifra scoring på kriterier. I prosjektet vårt har vi brukt denne matrisen for evaluere mellom konseptet vårt mot vektprosent og vi hadde tre konsepter og vi valgte det konseptet som scoret best på de viktige kriteriene som vi hadde.

CRITERIA	WEIGHT %	CONCEPTS					
		TRASH TERMINATOR 2000		RIVER CLEANING SYSTEM		PACMAN	
		RATING	WEIGHT SCORE	RATING	WEIGHT SCORE	RATING	WEIGHT SCORE
COST	20	4	0.80	3	0.6	2	0.4
PROTECTING ANIMAL LIFE	10	4	0.4	5	0.5	5	0.5
COLLECTING	15	3	0.45	4	0.6	3	0.45
ENERGY CONSUMPTION	5	5	0.25	3	0.15	2	0.1
STURDINESS	10	3	0.3	5	0.5	4	0.4
GARBAGE DISPOSAL	5	3	0.15	4	0.2	3	0.15
ENVIRONMENTALLY FRIENDLY	10	5	0.5	5	0.5	5	0.5
MAINTENANCE	10	4	0.4	5	0.5	3	0.3
INSTALLATION	5	2	0.1	4	0.2	4	0.2
SAFETY	10	5	0.5	5	0.5	5	0.5
TOTAL SCORE			**3.85**		**4.25**		**3.5**

Figure 3: Decision making matrix,
hentet fra[2]

0.1.4 Kontekst diagram

Kontekst diagram er et diagram som definerer grenser mellom systemet og miljø rundt den og viser enheter som har interaksjoner med det. Dette er et høyt nivå diagram av systemet og viser ikke noe detaljer om systemet. For oss var det viktig å forstå i hvilket miljø vil vårt system operere, siden elver varier både med vannstrømmer og elv kanter og vi vi måtte også tenke på sikkerhet på de som bruker elva og ikke bli skade av systemet vårt. Denne kontekst diagrammet som vi laget viser en innblikk i hvilket eksterne deler/faktorer som vi må ta i betrakning til når vi utvikler systemet vårt og hvordan disse faktorene vil samspille med sytemet vårt.

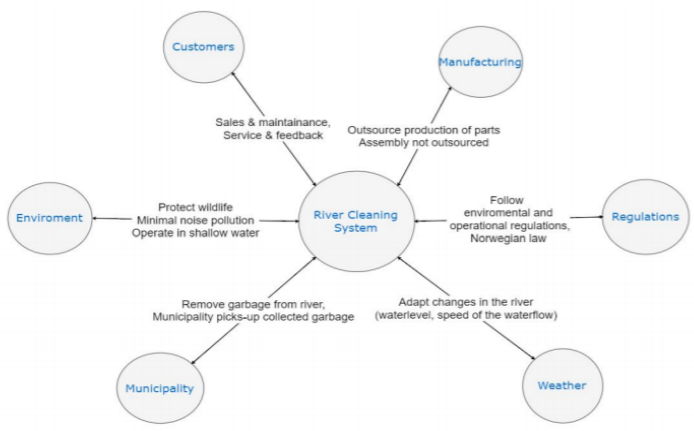

Figure 4: Kontekst diagram, hentet fra[2]

0.1.5 Use case diagram

Use case diagram er et diagram som beskriver hvordan et system vil samhandle med eksterne aktører. Vi valgte å lage et use case for vårt system for å få en bedre forståelse av systemet og satt noen av kravene til systemet i en use case modell og vi fant også noen tilleggskrav til systemet vårt og utfra denne diagrammen designet vi sequence diagrammer.

0.1.6 Sequence Diagram

Sekvensdiagrammer representerer hvordan flere objekter samarbeider med hverandre for å løse en spesifikk oppgave. I prosjektet valgte vi å lage en sekvensdiagram for manuelloverstyring av systemet vårt for å se hvilket steg må gjøres først og diagrammet viser en en sekvensiell rekkefølge mellom objekter og interaksjoner. Diagrammet viser en manull styring av systemet fra en smarttelefon og hvilket steg må skje for at brukeren av systemet skal få styret det manuelt.

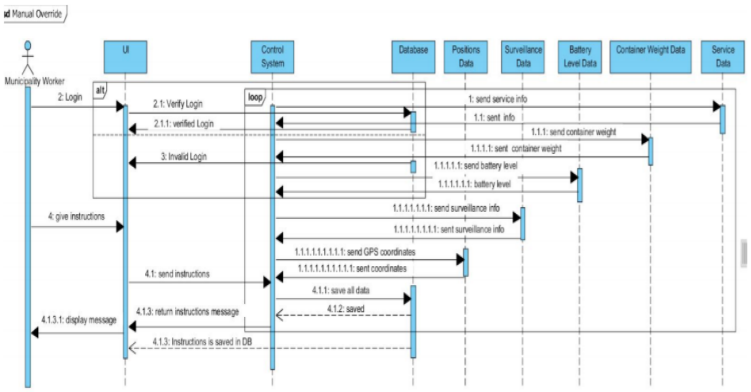

Figure 5: Sequence diagram av manuell styring,
hentet fra[2]

0.2 Prototype og Simulering

0.2.1 Prototype

Prototype er en første/tidlig utgave av et produkt som lages i en utviklingsprosess. I industribransjer er det en viktig steg å utvikle og implementere en tidlig utgave et produktet slik at de kan teste og få tilbakemeldinger på det. Når man skal utviklet et system så det hjelper mye med å lage prototyper slik at man skal teste og se om designet skal endres eller noen andre viktig ting som funksjonaliteter skal endres osv.

En prototype er en iterative design prosess hvor vi går i en loop fra design til tesing og tilbakemeldinger. Vi starter med å designe først så går vi til å lage en modell av designet og da kan vi bygge en prototype. I prosjektet vi startet med å designe og laget modell av designet men vi konstruerte ikke prototype siden det var valgfritt. prototyper brukes som en mal for videreutvikling av systemet og det en modell som representerer en visjon.

0.2.2 Simulering

Simulering handler om å lage en modell som kan manipuleres logisk og bestemme hvordan den fysiske systemet vil fungere og man også si at det

er en teknikk for optimalisere prosess. Simulering hjelper med å ta gode valg ved å bruke virtuell representasjon for å teste prosessendringer og simulering svarer på spørsmål.

Ved å simulere systemet vil det hjelpe med å senke kostandene og vekta og oppforselen til systemet og man kan ta solid beslutninger allerede tidlig i prosessen. Det er en prosess med å teste utvikling av et komplekse systemer uten å teste det i det virkelig problemet.

0.3 Systemstenkning

Systemtekning er systematikk-tekning, disiplin og kreativitet for å sette gode og nye ideer ut. En system designer må tenke langt flere linjer og her må vi se for inspirasjon på den ene siden og mulige løsninger på andre siden. Systems ingeniører kontinuerlig bruker forskjellige tenke måter for å evaluere styre utviklingsprosess. Det å ha en god strategi hjelper mye med å gå fra det ene tenkemåten til den andre. Under er et eksempel på når et system er under design hvordan en systems ingeniør bør tenke.

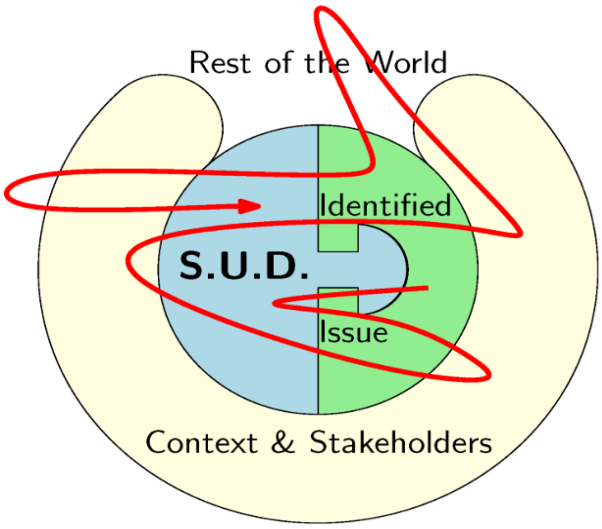

Figure 6: System Under Design,
hentet fra [1]

0.3.1 Hierarkisk Tenkning

Hierarkisk tenkning er at man skal få samme resultat ved forskjellige løsninger, hvor noen av løsningene oppfyller flere kravene til stakeholderne. En system designer må vurdere flere ting som for eksempel: fasiliteter og prioritere systemets subsystemer. Siden vi lever i en kompleks verden så må vi tenke hierarkisk tenkning for å fange det kompleksitet. Som en Systemsingeniør må man tenke abstrakt for å fange de mest viktige egenskapene til systemet som det kreves av systemet og få en oversikt over systemet. Systems ingeniører med hierarkisk tenking måte er mer komfortable med autoritet og foretrekker å jobbe når det er en klar struktur og de tar forskjellige forutsetninger og foretrekker formelle interaksjoner.

Siden en system er bygget opp av subsystemer så må vi tenke hierarkisk struktur og en hierarkisk struktur er en universal fenomen. Hierarki har fire terminologi og det mest grunnlegende er elementer, så neste er subsystemer og de består av elementer. Forskjellige systemer er bygget på ulike hierarkisk struktur.

I prosjektet vi har tenkt hierarkisk fordi vi så på flere løsninger som ga samme resultat, men kostanden var forskjellige og derfor valgte vi den som kostet minst og ga det samme resultatet som de andre skulle gi. Vi har også vurdert hvilket del av systemet av viktig og derfor prioriterte den delen.

0.3.2 Dekomponering og Sammensetning Tenkning

Det handler om hvordan en systems ingeniør eller designer skal dele en hel system i subsystemer og komponenter og her er det viktig å tenke hvordan all dise kan bli integrert til en velfungerende system. Denne type tenkning er vanskelig og må trenes for at en skal bli god på dette. Siden et system består av mange små deler så en systems ingeniør må tenke på å finne grensesnitt mellom de subsystemene.

Når vi skulle lage en autonom system i prosjektet vårt, så valgte vi å dele systemet i en hierarkisk struktur top-down hvor systemet var på toppen og subsystemer var delt i flere grener. Dekomponering tenkning hjalp oss med å dele systemet og subsystemer i små deler hvor vi kunne se hvilke part av systemet var viktig for å prioritere det. Det hjalp også med å finne grensesnitt mellom de subsystemene og hva skjer når funksjoner blir delt og hvilket effekt det gir til systemet. Vi måtte tenker underveis hvordan vi skulle sette de subsystemene sammen og vi måtte sjekke om det passer i systemet

eller ikke og kan vi test det før vi integrerer det i systemet. Vi måtte stille mange spørsmål når vi delte systemet i subsystemer osv.

Det handler om å organisere utviklingsprosessen og vurdere å tenke på den måten som de fleste designere og organisasjoner tenker. Når det er ulike fagfelt som er involvert i et prosjekt så måtte vi ha en konkret bildet av systemet slik at hver og en kunne jobbe med i systemet og hvilket part er de spesialisert i.

0.3.3 Spesifikk-Generisk Tenkning

Dette handler om omfanget til problemet og løsningen. Vi må se på hovedproblemet og forstå den, så kan vi se om det kan løses med en generell løsning eller må vi finne en spesifikk løsning til problemet. Vi må tenke om det er godkjent med en rask løsning eller må vi dokumentere det grundig og det tar tid selvfølgelig.

I prosjektet hadde vi et stort og spesifikk problem som vi skulle løse. Siden problemet var stort å finne en generell løsning var det beste alternativet, slik at vi kunne bruke denne løsningen andre steder også for å rense elver fra søppel og løsningen måtte ha en grundig dokumentasjon. For at vi skulle løse et plast problemet fra elver så trengte vi å tenke spesifikk på hvordan vi skulle løse problemet på en best mulig måte. Vi måtte tenke på hva slags søppel som finnes i elver, trenger vi sorteringsstasjon, hvordan skal vi frakte til land, hvor effektiv systemet skal være, hvordan unngå å ikke skade fisker og andre dyr. Vi måtte tenke veldig spesifikk på det siden vi skulle selge systemet til norske kommuner og Norge er veldig strengt på det.

Hovedproblemet var veldig stort og det hadde mange sub-problemer som vi måtte finne en spesifikk løsning på det som for eksempel: sikkerhet er veldig viktig i Norge og siden vi skulle lage et autonum-system som skulle samle søppel fra elva og ikke skade folk som svømmer rund der og eller ikke kræsje med andre båter som kjører der og hvordan unngå hærvek.

0.4 Grensesnitt

Grensesnitt er en forbindelse mellom to eller flere systemer og eller aspekter og grensesnittstrying spiller en viktig rolle i utviklingen av komplekse systemer og det er på grunn av at et kompleks system er sammensatt av mange sub-systemer fra forskjellige engineering områder som for eksempel: software, elektro osv. I komplekse systemer grensesnittstyring er kjempe utfordrende siden systemer er utviklet av forskjellige utviklerteam, men når denne utfordringen er over så mange av prosjekt konflikter er redusert. Det hjelper med å forbedre kvalitet til systemet og forhindre feil i designet.

Siden et system er bygget opp av mange sub-systemer og samhandler med andre systemer. Det er viktig å indentifisere grensesnitt i starten av prosjektet slik at man finner grensen i systemet man skal utviklet og det hjelper med å forstå hvordan systemet er avhengig av andre systemer. Dersom man ikke identifiserer grensesnitt og det vil påvirke på systemet og kan hende at systemet ikke møter kundens behov og feil grensesnitt vil også skape store problemer og systemet vil feilet. Ved å identifisere grensesnitt vil også hjelpe med å avsløre potensielle problemer og risiko i prosjektet. Det er viktig å identifisere, definere og utvikle grensesnittskrav.

I vårt tilfelle så valgte vi å dele systemet i subsystemer for å identifisere grensesnitt mellom systemet og subsystemer og definere disse grensesnittene og utvikle grensesnittskrav. Når man deler et system i små deler så er det lettere å forstå hvordan disse henger sammen og hvilket deler må er avhengig av hverandre og hvilket deler må prioriteres for andre deler. Det interne grensesnitt er hvordan båten vår skal kommunisere med en subsystem som for eksempel: båten kommuniserer med navigasjonsystemet eller med sorteringsystemet som vil sortere søpla i båten. Det eksterne grensesnitt er hvordan båten skal kommunisere med dock stasjonen for å lade eller levere søppel.

Diagrammet under viser kommunikasjon mellom seleskapet vårt og noen av lifecycle stakeholers. Det viser at hvis batterinivået på båten så skal båten returne til dock stasjonen for å lade og hvis senur har registeret max vekt på containern så skal båten gå til dock stasjon for å levere fra søpla. Dock stasjon kommuniserer med seleskapet som tar imot søpla dersom dock stasjon er fult så dock stasjon vil sende beskjed til selskapet for å komme å hente søpla. Alt dette skjer via interfacer.

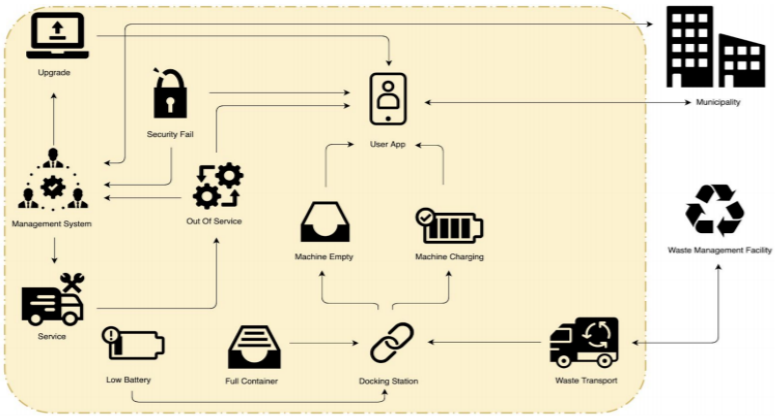

Figure 7: Physical block diagram,
hentet fra[2]

0.5 Traceability requirement

Tacing requirements handler om evnen til å beskrive og sørge for at kravene blir fulgt både framover og backover fra start til slutt. For å gjennomføre en tracing requirement analysering er en viktig del av engineering prosess for å sørge for at alle kravene har blitt vurdert tilstrekkelig for hvert av prosjekt faser. En vanlig måte å få overskit over alle kravene er å en tracing requirement matrise og det helper med å verifisere at alle de angitte kravene er tilknyttet til tilsvarende design elemente. Vi har to type klassifisering av requirement traceability og de forward and backward.

0.5.1 Forward

Linker kravene til design og implementasjon av komponenter også linker til kilder. Det er brukt for sjekke om utviklingen til produktet går i riktig vei og sørger for at hvert krav er testet grundig.

For at vi skulle være sikre på at vi dekket de funksjonelle og system kravene til systemet, så trengte vi å lage en use case diagram for å validere de systemkravene. Krav er hovedfokuset i systems engineering før man går til konseptuelle fasen og disse kravene kan uttrykkes ved både modell og tekst og noen av modeller som kna brukes er use case, Business prosess og funksjonelle analyses.

ID	System requirement	Use case	Test case ID	Status
FR9	Ability to manual override	UC1	TC9	Unknown
FR10	Ability to charge with dock station	UC1	TC9, TC10	Unknown

Figure 8: Forwards functional requirements traceability matrix, hentet fra[2]

0.5.2 Backward

[h] Linker kravene til kilder som dokumenter og folk som har laget systemet og den også linker design og implementasjon av komponenter til kravene. Det er brukt for å sørgefor at det nåværende produkt er på riktig spor og hensikten med denne traceability er å bekrefte at vi ikke utvider omfanget til prosjektet ved å legge til noe som for eksempel arbeid som ikke er spesifisert i kravet eller kode. Siden stakeholderne har forskjellige krav og det er viktig å lage en RT-matrise som kan spores opp til hver enkelte kunder.

ID	System Requirement	Stakeholders	Test case ID	Status
SHR1	Ability to be autonomous	Teacher	TC1	Unknown
SHR2	Ability to collect, contain and dispose garbage	Municipality	TC2	Unknown
SHR4	Ability to not harm or disrupt wildlife	Fishermen Media	TC1, TC2	Unknown

Figure 9: Reverse stakeholders requirements traceability matrix, hentet fra [2]

0.6 Verifisering og validering

Verifisering og validering er uavhengige prosedyrer som brukes til å kontrollere om at et system møter kravene og spesifikasjoner for en bestemt formål. Validering er en bekreftelse fra en test og fremskaffing av beviser på at systemet vil oppfylle de kravene og spesifikasjonene. Verifisering i system engineering er en generell betingelse for å sjekke systemsytelsen mot kravene og vi kan gjøre det ved å simulere, anslysere eller vurderinger og målet med er å levere et system til kunden som er akseptert av kunden og systemet løser kundens problem.

0.6.1 Verifisering

Verifisering er en formell bevis på at systemetdesignet oppfyller alle de spesille kravene og spesifikasjonene og verifisering er en av de grunnene til at kostnadene til et system er høyt. Det finnes metoder som man bruke for å verifisere et system mot de bestemte kravene og mododene er som følgende: Vurdering av design, analysering og test.

I prosjektet vi måtte lage en system verifisering plan for at vi skulle sjekke design og system kravene til systemet vårt mot kundens krav, dersom det ikke gjorde så måtte vi endre design. Her vi måtte teste implementerte løsning designet vårt og sjekke om systemet gir de tilfredsstille de kravene som vi har skrevet ned og om vi har oppnådd disse kravene.

0.6.2 Validering

Dette er en prosess for å evalurere det finalet produktet for å sjekke om produktet oppfyller kundens forventninger og kravene til kunden. Validering prosessen skjer etter verifisering prosessen og validering prosessen kan fange feil som verifisering ikke fanger det opp.

Vi har ikke testet systemet vårt i en realistisk miljø så vi vet ikke hvordan systemet vil tilfredsstille seg og derfor vet ikke om det oppfyller det originale hensikten eller ei.

0.7 Risiko

Risiko er sannsynligheten for at et program eller et prosjekt vil oppleve noen uønsket effekt eller hendelse og konsekvensene av det det uønsket hendelse som påvirker på prosjektet eller programmet. For eksempel de uønsket hendelse kan være som følgende: planlegging, kostnadsoverskridelser eller systemet ikke er pålitelig osv.

0.7.1 Teknisk risiko

En av de store tekniske risiko som er relatert til vårt system er at sytemet ikke virker. Når systemet ikke virker så det skaper andre problemer og konsekvensene vil bli stor. Denne vil forskinke leveranse av systemet til kunden og kostnaden vil overstige kostandene til systemet og kan hende at seleskapet mister kunder på grunn av slikke gjentatte feil. Andre tekniske risiko er at systemet funker, men er ikke effektiv nok til å løse kundens problem eller at det systemet er ikke nøyaktig.

Det er viktig at man skal lage en god plan for prosjektet slik at prosjektet medlemmer vet i hver tid hva de skal jobbe med og hvilke deler må prioriteres. Når man lager komplekse systemer så det ulike team er som er involvert i prosjektet eller kanskje noen deler av det komplekset systemet bygges i helt annen bedrift og det viktig å kommunisere sammen og sette opp møter for test plan osv.

Siden alle systemer har risiko og det veldig lurt å identifisere de avvikene, jo tidligere man gjør det jo bedre er det og enklere å finne løsninger på dem. Hvis oppdager man avvikene forseint, så vil det skape problemer og det gir store konsekvenser for begge parter og risiko har alltid negativ påvirkninger til systemet.

I prosjektet vårt har vi brukt risikovurderingsprosess for å oppdage de uønskede hendelser for systemet vårt som ville oppstå. Det er seks steg som vi følger når vi identifiserer en fare og sjekker om det har stor, liten eller ingen påvirkning til systemet vårt. Dersom det har påvirking til systemet så dokumenterer vi det og finner en løsning til det og hvis det ikke har noe påvirkning dokumenterer vi det også bare for sikkert dersom det vil oppstå noen spørsmål rundt det. Figuren under risiko vurderingsprosessen.

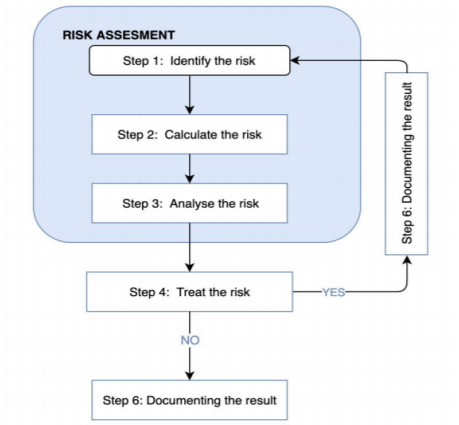

Figure 10: Risk management process,
hentet fra [2]

0.8 Vårt system

Systemet vårt er en autonom system som kan kjøre selvstendig og kan styres manuelt via en app dersom brukeren ønsker å gjøre det eller brukeren ønsker å sjekke tilstanden til systemet. Systemet vårt har mange gode fordeler og skaper verdi for kunden. Den unike systemet vårt kan kjøre 24/7 for å rense elver for søppel og den skader ikke dyr, andre fordeler er at systemet vårt er elektrisk som ikke gir utslipp når den kjører. Verdien som den skaper for kunden er at elver blir renset for søppel som er en trussel mot dyr. Vår autonome båt lager lite lyd som gjør at lokale beboere ikke blir forstyrret av den og andre ved elva. Den oppdager mennesker og andre kjøretøy, så derfor tar den hensyn til folk som er i vannet og svømmer

All teknologien som finnes i markedet nå er enten manuelt og krever arbeidere som styrer maskinen, eller er stasjonære og har et begrenset samlings radius på grunn av dette. Noe som ikke finnes i markedet enda, er et fullt autonomt system som er selvstyrende og detekterer, samler og sorterer søppel selv.

Mange av de maskinene som styres av arbeidere er heller ikke energieffektive siden de er store maskiner av metall som veier bortimot en tonn eller mer. Energien som blir forbrukt av disse maskinene, og kostnadene av å ansette arbeidere som styrer disse, fører til at disse løsningene ikke er lev-

edyktige.

Et fult autonomt system vil koste mer på starten men vil være billigere å holde i drift. Dette er fordi størrelsen på maskinen kan da bli skalert ned som fører til at maskinen veier mindre og det brukes mindre energi til framdrift. I tillegg så sparer man også penger på å ikke trenge å ansette manuelle arbeidere. Alt dette gjør et fullt autonomt system en mer attraktiv løsning på problemet.

Figuren under viser noen av de ytelsesparameterne til systemet vårt og de ytelsesparameterne må oppfylles for at systemet ikke skal feilet.

Size	L = 2.2m W = 1.2m H = 1m
Weight	166kg
Cost	300K
Garbage weight capacity	500kg
Maximum speed	6km/h
Maximum consumption	1.5KW

Figure 11: Key performance parameter,
hentet fra[2]

Parameterne som vi valgte for systemet vårt er knyttet til forretningsplanen vår. Siden bedrifter konkurrer mot hverandre så valgte vi at kostnaden skal være lav, men skal ha høy kvalitet, at produktet skal løse kundens problem.

Kilder

[1] Jan F. Broenink G. Maarten Bonnema Karel Th. Veenvliet.
Systems Design and Engineering: Facilitating Multidisciplinary Development Projects.
14.10.2015.

[2] *Gruppe Prosjekt Rapport for Systems Design og Engineering.*
08.04.2019.

[3] Gerrit Muller.
Overview of CAFCR and Threads of Reasoning.
September 9, 2018
. URL: `https://www.gaudisite.nl/ArchitectingMethodOverviewPaper.pdf`.

[4] Paul Ryus.
ReseachGate, V Model.
Dec 2014
. URL: `https://www.researchgate.net/figure/1-Systems-Engineering-V-Diagram_fig1_301771084`.