CHALMERS

Visualisering av produktionsförbättringar

En förstudie till framtida demonstrator

Kandidatarbete inom maskinteknik

ALEXANDER BENGTSSON
AXEL ERIKSSON
JOHAN NÄNZÉN
EMELIE ROSLUND
CARL SWANSTRÖM

Institutionen för Material och tillverkningsteknik
CHALMERS TEKNISKA HÖGSKOLA
Kandidatarbete 2014:07

Förord

Detta kandidatarbete erbjöds av institutionen Material- och tillverkningsteknik på Chalmers tekniska högskola och genomfördes under våren 2014. Kandidatgruppen består av studenter i årskurs 3 från teknologsektionen för Automation och mekatronik samt Maskinteknologsektionen.

Kandidatgruppens medlemmar önskar tacka handledare *Richard Hedman* som ständigt varit tillgänglig för konsultation och positiv till gruppmedlemmarnas frågor. Ett tack riktas även till projektets examinator *Peter Almström* som tog sig tid att i början av kandidatarbetet utbilda gruppen i produktionsteknik, produktionshistoria och metodstudier.

Vi vill även tacka de personer som medverkat på intervjuer samt till de företag som på förfrågan lämnat ut kostnadsfria versioner av sina mjukvaror.

Alexander Bengtsson, Axel Ericson, Johan Nänzén, Emelie Roslund och Carl Swanström
Chalmers tekniska högskola
Göteborg, Maj 2014

Abstract

Swedish manufacturing industry is now facing large challenges since the increasing globalization have made it easy to outsource the production. To keep the competitiveness work improvements is necessary. One way to measure productivity is the method PPA, *Productivity Potential Analysis*. The project is carried out at Chalmers University of Technology at the institution for production management.

The starting point of this project is a conceptual model over a production system, which is a further development of the PPA-method. The project is a preparatory study which aims to investigate how to develop a demonstrator that shows the improvements mentioned above. To achieve this, the project has been divided into three separate parts:

- Formulate requirement specifications
- Evaluate existing software
- Develop a prototype

The result of these parts is a detailed software requirement specification for software development, a software requirements specification for evaluation of existing software, an evaluation of existing software according to the requirement specification and finally a verified and validated prototype. The prototype is fully functioning and has a manual which thoroughly explains how to use it. A case has been designed which, together with the manual, can help user who are interested in exploring the possibilities of the prototype. The conclusion is that it is indeed possible to use existing software for functionality. However, the development of a new demonstrator custom made for the model is considered to be the best alternative.

Sammandrag

Tillverkningsindustrin i Sverige står inför stora utmaningar och den ökade globaliseringen har gjort att flera företag flyttat produktion utomlands. För att behålla konkurrenskraft och jobb krävs produktivitetsförbättringar inom svensk tillverkningsindustri. Ett sätt att mäta produktivitet är med metoden PPA, *Productivity Potential Analysis*. Projektet utförs på Chalmers tekniska högskola som även är den plats där upphovsmakarna till PPA verkar.

Projektets utgångspunkt är en konceptuell modell över ett produktionssystem som är en vidareutveckling från PPA-metoden. Projektet är en förstudie som syftar till att undersöka hur en demonstrator kan utvecklas för att realisera denna konceptuella modell. För att uppnå syftet har projektet delats upp i tre delar:

- Utforma kravspecifikationer
- Utvärdera existerande mjukvaror
- Utveckla en egen prototyp

Resultatet från de tre delarna är en kravspecifikation för mjukvaruutveckling, en kravspecifikation för utvärdering av mjukvaror, möjliga kombinationer av existerande mjukvaror och slutligen en verifierad och validerad prototyp. Prototypen är fullt fungerande och till den finns en manual som grundligt förklarar för användare hur den fungerar. Ett case har utformats för att tillsammans med manualen fungera som en grund för nya användare som vill utforska prototypens möjligheter. Slutsatser som dragits är att det bedöms vara möjligt att använda existerande mjukvaror för funktionaliteten. Däremot bedöms framtagning av egenutvecklad demonstrator anpassad efter modellen som det bästa alternativet.

Ordlista

Nedan beskrivs kort de ämnesspecifika ord som används i rapporten. Orden återfinns i bokstavsordning.

Anläggning (facility) – Fabrik, subsystem och arbetsstation är olika typer av anläggningar. Fabriken består av subsystem som består av arbetsstationer där aktiviteter utförs.

Aktivitet – Byggs upp av flera subaktiviteter och utförs av resurser. Aktiviteterna formuleras som tidsekvationer. En aktivitet skulle kunna vara "Montera Produkt A".

Aktivitetsklassifikation – Talar om aktiviteten är cykelberoende, batchberoende eller periodisk. Kommer från produktionen.

Demonstrator – Ett verktyg som används för att visualisera något. I det här projektet innebär en demonstrator en mjukvara (eller en kombination av flera mjukvaror) som visualiserar produktionsförbättringar i industrin.

Direkt aktivitet – Aktiviteter som driver materialet framåt, till exempel montering.

Element – De minsta rörelser som en operatör utför, till exempel att sträcka sig för att ta något eller att ta ett steg. Det är de minsta beståndsdelarna i aktiviteter. Elementen fås från SAM-studier och har fördefinierade namn. De matas in manuellt eller läses in från mjukvara givet SAM-ark. Grundrörelserna förklaras nedan:

- **Get** – att sträcka sig för att greppa
- **Put** – att placera på en plats
- **Step** – att förflytta sig
- **Return** – att sträcka sig för att lämna tillbaka
- **Use** – att använda, till exempel en skruvmejsel
- Tillägg till grundrörelser:
    - **Apply force** – att med kraft trycka eller lägga ner
    - **Handful** – att greppa en handfull, till exempel skruvar
    - **Precision** – att med precision placera eller utföra

Elementartidssystem – Ett system som sätter en standardtid på element. Detta motsvarar den beräknade tid som det tar för en medeloperatör att genomföra detta element. Operatören ska klara av att följa standardtiden åtta timmar om dagen, fem dagar i veckan i hela sitt arbetsliv utan att bli utsliten. MTM och SAM är exempel på elementartidssystem.

Exekvera – Köra en mjukvara.

Faktisk tid (actual time) – Den verkliga tid det tar för en operatör att genomföra en subaktivitet eller aktivitet.

Faktor – Tidsenhet som används i SAM-studier, 1 timme = 3600 s = 20 000 faktorer.

Gränssnitt – Utformningen av en viss förbindelse mellan olika objekt. Kommunikation mellan mjukvaror eller maskin och människa.

Ideal kapacitet, CAP_I – Ideal kapacitet visar hur många enheter per timme som idealt kan tillverkas vid en arbetsstation, det vill säga utan störningar eller annat som hindrar arbetaren från att arbeta. Kapaciteten baseras på den ideala tiden det tar att utföra de element som aktiviteterna i arbetsstationen består av, det bygger alltså på elementartidssystem.

Ideal tid (ideal time) – Den tid en (sub-)aktivitet ska ta enligt SAM-studie och elementartid. Ska dock inte förväxlas med optimal tid. Ideal tid utgår från metoden enligt nuvarande standard medan optimal tid utgår från den optimala metoden.

Indirekt aktivitet – Aktiviteter som krävs men som inte driver materialet framåt, till exempel laddning av maskin eller hämta material.

Informationsmodell – Den konceptuella modell av ett produktionssystem som tagits fram i tidigare forskning. Visar på ett generiskt sätt samband mellan den totala kapaciteten i en fabrik och aktiviteternas uppbyggnad. Se kapitel 2.2 för utförlig förklaring.

Iterera – Upprepa till önskat resultat uppnåtts.

Kompilera – Översätta kod till instruktioner för datorn.

Maskinvara – Samlingsnamn för en dators fysiska delar.

Metod – Den metod som används för tillverkningsprocessen. Det vill säga i vilken ordning aktiviteter och subaktiviteter sker samt hur de är uppbyggda.

Prestationsparameter – Parameter som talar om i vilken takt jämfört med den ideala som operatören arbetar. Anges i procent. Om arbetet utförs på en kortare tid än idealt blir parametern över 100 % och om det utförs på en längre tid blir parametern under 100 %. Denna faktor påverkas endast av arbetarens motivation och fysisk förmåga. Kommer från manuell prestationsbedömning eller ideal tid jämförd med verklig tid.

Process – Aktiviteter och resurser sammankopplade där aktiviteterna är ordnade i den följd som krävs för att skapa produkter. Flera processer kan gå genom samma aktivitet.

SAM – Ett MTM-baserat elementartidssystem. Förkortningen står för sekvensbaserad aktivitet- och metodanalys. Används för att bryta ned aktiviteter i dess minsta beståndsdelar, element, för att på så sätt kunna analysera aktiviteter. Kan endast göras om arbetet är standardiserat.

SAM-ark – Ett Excel-ark som erhålls efter genomförd SAM-studie. Visar hur aktiviteter består av subaktiviteter och hur de i sin tur består av element. Respektive elements tid i faktorer samt antal element och subaktiviteter återfinns även här. Kan vara gjort för hand eller automatiskt genererat från en mjukvara, till exempel AviX.

Subaktivitet – Byggsten i aktiviteterna, till exempel “Hämta Komponent A”.

Tidsdrivare – Tidsdrivare är en multiplikator som används för att ange antalet gånger en (sub-)aktivitet eller ett element utförs. Till exempel blir tidsdrivaren för att skruva på stolsben på en stol fyra eftersom samma procedur utförs fyra gånger.

Tidsekvationer – Visar vilka subaktiviteter en viss aktivitet består av och vilka element en viss subaktivitet består av. Tidsekvationerna visar också antalet likadana subaktiviteter eller element, se tidsdrivare. Ekvationerna kan även visa vilka delar som en anläggning är uppbyggd av. Tidsekvationerna skrivs på formen nedan.

$$Montera\ Produkt\ A\colon Tidsdrivare1 \times Hämta\ Komponent\ A + Tidsdrivare2 \times Hämta\ Komponent\ B$$

Utnyttjandegrad (utilization) – Utnyttjandegraden visar hur arbetstiden fördelas mellan planerat produktionsarbete och förluster i förhållande till planerad produktionstid. Förlusterna kategoriseras som personlig tid, störningar och systemförluster. Denna fördelning fås från en frekvensstudie för manuella arbetsuppgifter och anges i procent (0-100 %). Den totala arbetstiden motsvaras av 100 %.

De olika kategorierna återfinns nedan:

- $\mathbf{U_N}$ – personlig tid, till exempel raster
- $\mathbf{U_S}$ – systemförluster, till exempel balanseringsförluster
- $\mathbf{U_D}$ – störningar i produktionen
- $\mathbf{U_M}$ – planerat arbete

Verklig kapacitet, $\mathbf{CAP_R}$ – Talar om det antal enheter som för tillfället produceras. Siffran ska alltså motsvara det faktiska antalet enheter som varje timme produceras. Denna kapacitet beror på hur situationen ser ut: hur väl metoden fungerar, hur väl arbetarna arbetar, hur väl maskinerna fungerar, hur organisationen ser ut samt hur produktionssystemet är designat.

1 Introduktion

Inledningsvis kommer bakgrunden till varför ämnet är aktuellt presenteras samt bakgrunden till projektet. Därefter presenteras problembeskrivningen, syftet och frågeställningarna i projektet innan avgränsningarna och tillvägagångssättet för projektet beskrivs.

1.1 Bakgrund

Svensk tillverkningsindustri står idag inför stora utmaningar. Den ökande globaliseringen har lett till att svenska industriföretag har flyttat produktion till låglöneländer och endast behåller den mer specialiserade kärnverksamheten och utvecklingen inrikes (Bellgran and Säfsten, 2005, Gamma et al., 1994). Pisano and Shih (2012) poängterar att produktutvecklingen ofta är nära sammankopplad med produktionsverksamheten och att produktutvecklingen därför tenderar att flytta efter. Denna ökande globala konkurrens har därför tvingat den svenska industrin att öka automatiseringsgraden för att kunna bibehålla sin konkurrenskraft trots det höga löneläget i Sverige (Almström and Winroth (2010). Klingstam and Gullander (1999) betonar att större krav på effektivitet, kvalitet och flexibilitet lett till att högre krav ställs på beslutsstödsystemen. Därför förespråkar de en ökad användning av IT-verktyg för beslutsstöd inom tillverkningsindustrin.

Almström and Kinnander (2011) poängterade att det finns en stor outnyttjad förbättringspotential för effektiviteten i svenska industriföretag. För att mäta och beskriva detta utvecklade de metoden Productivity Potential Assessment (PPA). Resultat från studier har visat att flertalet tillverkande företag har avsevärd förbättringspotential, framför allt gällande utnyttjandet av befintliga resurser i form av människor och maskiner, samt produktionsteknisk kompetens (Almström and Kinnander, 2006). För att denna produktionspotential ska kunna realiseras behöver den demonstreras på ett tydligt sätt för beslutsfattare i företaget. Det är viktigt att informationen är tydlig, eftersom människor har en ovilja mot förändring som kan bero på att resultatet av förändringen är okänt eller oönskat. (Rubenowitz, 2004)

Arbetet med PPA-metoden har vidareutvecklats av forskargruppen i Productivity Management vid Chalmers tekniska Högskola, Institutionen för material- och tillverkningsteknik. Utvecklingen har lett till att en konceptuell modell av ett produktionssystem (Hedman, 2013), hädanefter kallad informationsmodell, tagits fram. Med hjälp av modellen kan företagets produktionsförbättringspotential utvärderas genom att utvärdera följande parametrar:

- Produkternas tillverkningsmetod (M)
- Resursernas prestation (P)
- Resursernas utnyttjandegrad (U)

Med hjälp av dessa tre faktorer är det möjligt att demonstrera produktivitetsförbättringspotentialen hos tillverkande företag.

1.2 Problembeskrivning

För att motivera förändringar i produktion behöver en vilja till förändring skapas. För att uppnå detta bör förbättringspotentialen demonstreras på ett tydligt sätt baserat på fakta så att personer med olika bakgrund och kompetens inom ett företag kan ta till sig informationen och förstå möjligheterna. En utredning om hur informationsmodellen kan realiseras i en mjukvara skall göras.

1.3 Syfte och mål

Projektets syfte är att genomföra en förstudie av hur informationsmodellen kan realiseras i form av en demonstrator. Förstudien har delats in i tre delmål:

- Skapa kravspecifikationer
- Genomföra en utvärdering av existerande mjukvaror
- Utveckla en egen prototyp

De tre delarna är tänkta att ligga till underlag för hur en demonstrator bäst kan utformas. Detta ska ske genom att prototypen syftar till att utveckla kraven i kravspecifikationen. Kravspecifikationen i sin tur syftar till att vara underlag för utvärderingen av mjukvarorna.

1.4 Frågeställningar

För att uppnå syftet på bästa sätt ställs följande fråga:

Vilka krav bör ställas på en demonstrator för att produktionsförbättringspotentialen ska visualiseras på ett tydligt sätt?

Baserat på de krav som ställs ytterligare två frågor:

Vilka kommersiellt tillgängliga mjukvaror är möjliga att använda och hur kan en sammansättning av dessa se ut?

Hur kan en potentiell demonstrator utvecklas?

1.5 Avgränsningar

Syftet är att realisera en befintlig informationsmodell och därför kommer ingen verifiering eller testning av logiken och sambanden bakom informationsmodellen ingå i projektet. En framtida demonstrator är tänkt att analysera redan existerande produktionssystem.

Ett av projektets mål är att ta fram en prototyp och inte en felfri kommersiell mjukvara. Mjukvaran utvecklas därför med fokus på interaktion, funktion, design och inte på stabilitet. Vidare kommer den egenutvecklade prototypen utvecklas med objektorienterad programmering eftersom informationsmodellen är objektorienterad.

När olika alternativ beaktas tas ej hänsyn till kostnaden för dessa val, detta då uppskattning för de olika alternativen är ett stort projekt som kräver djupare kunskaper och erfarenhet av mjukvaruutveckling än den som projektgruppen besitter.

Rapporten beskriver framtagningen av två kravspecifikationer, en prototyp samt en utvärdering av kommersiella mjukvaror. Produktionsanalys sker med hjälp av prototypen samt dess case och manual.

1.6 Övergripande tillvägagångssätt

Förstudien har delats in i tre huvudområden; framtagning av två kravspecifikationer, utveckling av en prototyp och utvärdering av existerande mjukvaror.

För att utföra dessa delar har studien varit dels empirisk och dels teoretisk. Projektets tillvägagångssätt förklaras mer ingående i kapitel 3.

1.7 Disposition

Rapporten är tänkt att läsas av en högskolestudent på en teknisk högskola med begränsade datakunskaper. I rapporten finns en teknisk beskrivning av arbetet med och resultatet av att ta fram kravspecifikation, utvärdera kommersiella mjukvaror samt utveckling av prototyp.

I kapitel 2 beskrivs projektets teoretiska ramverk som använts i projektet, både för informationsmodellen och de tre projektleverablerna. Detta följs av kapitel 3 som beskriver metodiken för projektets leverabler och behandlar hur teorin har använts. Resultatet beskrivs i kapitel 4, 5 och 6 som behandlar framtagning av kravspecifikationer, utvärdering av mjukvaror samt prototyputveckling. Resultaten diskuteras i kapitel 7 där även en koppling till och jämförelse med teorin finns. Därefter kommer kapitel 8 med en slutsats kring vad projektet lett fram till samt en rekommendation för framtiden. Rapporten avslutas med en referenslista och bilagor.

2 Teoretiskt ramverk

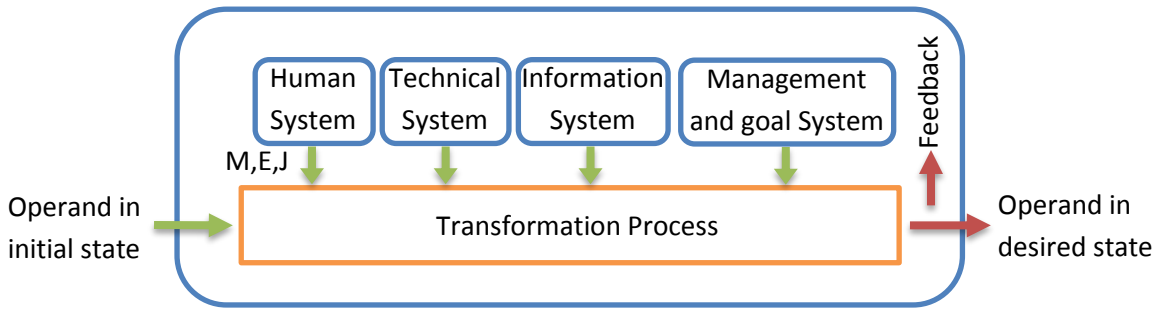
I följande kapitel behandlas teorin för rapportens huvuddelar. Först ges en inblick i produktionssystem samt olika metoder för att ta fram mätetal inom detta område. Detta följs upp med en förklaring av informationsmodellen och dess delar. Kapitlet fortsätter med teoretisk information om projektmodeller, kravhantering, mjukvaruutveckling samt teori kring utvärdering av mjukvaror.

2.1 Analys av produktionssystem

Ett system utgörs av olika komponenter som samverkar och systemteorin baseras på relationerna och samspelet mellan dessa. En del i systemteorin innefattar att definiera systemgränsen för att bestämma vad som utgör in- och utdata till systemet samt vilka komponenter systemet innefattar. En grundläggande utgångspunkt för systemteorin är att ett system ger synergier, vilket resulterar i att det som förs ut ur systemet är värt mer än det som fördes in. I transformationssystemet i figur 1 illustreras transformationssystemets förmåga att generera synergier (Bellgran and Säfsten, 2005).

Figur 1 Visar en bild av ett system där indata (input) transformeras till utdata (output). Inspiration hämtad från Bellgran and Säfsten (2005).

Ett system är en samling personer, maskiner och metoder som används för att utföra aktiviteter (Chisholm, 1990). Churchman (1968) tillägger att ett system ska ha en mängd mål som det arbetar mot. Hubka and Eder (1988) använder systemteorin och kombinerar den med produktionsteori för att skapa en transformationsmodell. I transformationsprocessen av indata till utdata inkluderas mänskliga, tekniska, informations- och ledningssystemet. Det mänskliga och tekniska systemet utgör arbetskraften och maskinerna som utför tillverkningsprocessen. Transformationsprocessen påverkas även av informationssystemet och ledningssystemet beroende på hur det är utformat för att stödja systemet vilket illustreras av figur 2.

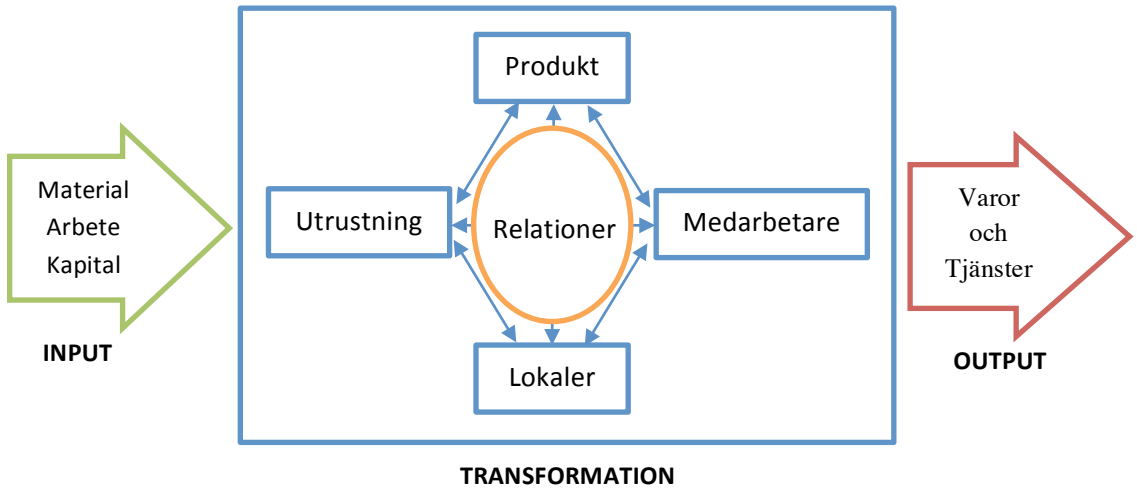


Figur 2. Bild över transformationsprocessen. Bild inspirerad från Hubka and Eder (1988).

Hågeryd et al. (2005) illustrerar ett produktionssystem i figur 3, vilken inkluderar in- och utparametrar. I modellen utgörs indata av material, arbete och kapital medan utparametrarna utgörs av varor och tjänster. Transformation från indata till utdata utgörs av relationerna mellan produkter,

lokaler, medarbetare och utrustning som används inom det definierade produktionssystemet. Hur effektiv transformationen är påverkas av samverkan mellan ovanstående relationer och ett vanligt mått på en organisations produktivitet är:

$$Produktivitet = \frac{Output}{Input}$$



Figur 3. Visar de delarna som ingår i ett produktionssystem. Inspiration hämtad från Hågeryd et al. (2005).

Den här delen täcker vilka verktyg och tekniker som finns tillgängliga för att analysera produktionssystem. De som har identifierats som intressanta för projektet är tidsstudier, SAM-analys och frekvensstudier.

Tidsstudier kan delas in i direkta och elementartidsbaserade. Ett exempel på en direkt tidsstudie är att en aktivitets cykeltid mäts. Direkta tidsstudier syftar till att underlätta planeringsarbetet och för att kunna jämföra faktiska tider med planerade tider för att bedöma produktionssystemets effektivitet (Sundkvist, 2014). Elementartidssystem använder förutbestämda standardtider för vanliga rörelser som till exempel att gå, att sträcka sig och att placera föremål. Rörelserna som har tilldelats standardtider kallas element och att ge rörelser standardtider syftar till att förutspå standardtider för nya eller existerande arbetsaktiviteter Freivalds (2009). Det finns ett flertal olika elementartidssystem som till exempel Methods Time Measurement (MTM). MTM-studier bryter ned rörelser på en väldigt låg nivå vilket medför att det tar lång tid att genomföra en sådan tidsstudie. Det har tillkommit flertalet elementartidssystem sedan MTM och en av dessa är Sekvensbaserad aktivitet- och metodanalys (SAM). Den främsta skillnaden mellan systemen som utvecklats är noggrannheten i nedbrytningen av rörelser där den mest noggranna bryter ned rörelser i ögonrörelser (Sundkvist, 2014).

SAM är ett MTM baserat elementartidssystem som möjliggör att metodmätningen av arbetsaktiviteter utförs på ett standardiserat och objektiv sätt. Systemet är sekvensbaserat där en aktivitet studeras för att identifiera rörelserna som har utförts. Standardrörelserna delas in i elementen Get, Put, Use och Return. Ett exempel på en aktivitet är att sträcka sig efter en kaffekopp, föra den till munnen för att slutligen placeras på bordet igen. Denna sekvens innehåller elementen

hämta, använd, och lämna i den givna ordningen. Resultatet från SAM-studien ges i en tidsenhet som kallas faktor där en faktor är 0,18 sekunder (IMD, 2004).

Frekvensstudier är en statistisk teknik som används för att undersöka hur stor del av företagets resurser inom ett visst område som används till specifika aktiviteter. Undersökningen genomförs genom att en tillverkningsresurs observeras under slumpmässiga tider för att kontrollera vilken aktivitet som utförs. Detta resulterar i en procentuell fördelning av de olika genomförda aktiviteterna och utifrån denna fördelning kan förluster identifieras (Freivalds, 2009).

2.1.1 Produktivitetsdimensioner

För att få en bättre förståelse för vilka faktorer som skall fokuseras på för att förbättra produktiviteten har Saito (2001) och Helmrich (2001) identifierat tre olika dimensioner metod, prestation och utnyttjandegrad.

Metodvariabeln utgörs av arbetssättet och mäts som cykeltiden för en viss aktivitet, variabeln fås alltså från elementartidsstudier, exempelvis SAM. Cykeltiden inverteras och multipliceras med 3600 för att få produktionskapaciteten per timme:

$$M = \frac{3600}{Cykeltid\ [sekunder]}\ [st/h].$$

Genom att jämföra den faktiska tiden med den ideala tiden för produktionen av en viss produkt fås hur snabbt tillverkningsresurserna arbetar jämfört med den ideala situationen, vilket är variabeln för prestation. Detta fås med:

$$P = \frac{Faktisk\ tid}{Ideal\ tid}\ [\%]$$

Utnyttjandegraden illustrerar hur stor del av arbetstiden som spenderats på att utföra planerade aktiviteter jämfört med den totala planerade tiden. Det innebär att utnyttjandegraden aldrig kan bli högre än 100 %. Utnyttjandegraden fås genom:

$$U = 100\% - U_{Förluster}\ [\%]$$

Genom att multiplicera metodvariabeln med den procentuella utnyttjandegraden och prestationen fås den faktiska kapaciteten i systemet. Genom att jämföra den ideala med den faktiska kapaciteten går det att visa på vilken förbättringspotential som finns i produktionen. Detta kan även jämföras över tid för att visa på vilka förbättringar som har gjorts samt hur detta har påverkat förhållandet mellan den ideala och faktiska kapaciteten.

Exempel - Detta kan illustreras med ett exempel på en arbetsstation på ett måleri. På måleriet finns en arbetsstation där en person skall måla en robot. På stationen för målningen är den ideala tiden för att måla roboten satt till 100 sekunder. Dagen som prestationsstudien utfördes gjordes en uppskattning att 200 sekunder var tid som det faktiskt tog att måla roboten. Samtidigt som prestationsuppskattning gjordes det också en analys för utnyttjandegrad. Det sågs att planerade aktiviteter uppmättes till 70 % av den tid målaren arbetade. Detta ger nedanstående differens mellan den ideala och faktiska kapaciteten:

$$Ideal\ kapacitet = M * P_{=1} * U_{=1} = \frac{3600}{100} * 1 * 1 = 36\ stycken$$

$$Faktisk\ kapacitet = M * P * U = \frac{3600}{100} * \frac{100}{200} * 0{,}6 = 36 * 0{,}5 * 0{,}6 = 10{,}8\ stycken$$

Nästa gång stationen analyserar tar det 90 sekunder för montören att utföra stationen och planerade aktiviteter utförs 50 % av tiden. Samtidigt är metoden oförändrad vilket ger följande resultat:

$$Ideal\ kapacitet = M * P_{=1} * U_{=1} = \frac{3600}{100} * 1 * 1 = 36\ stycken$$

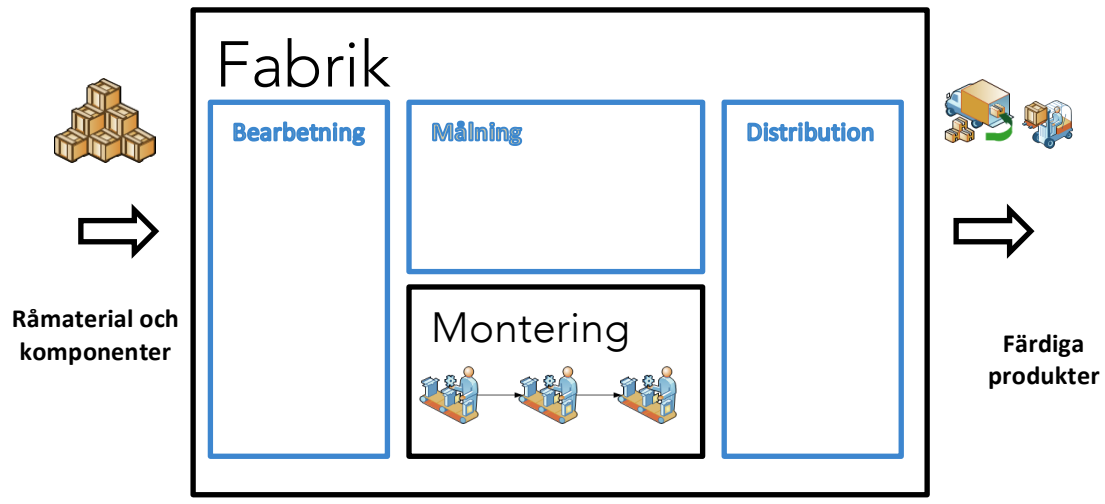
$$Faktisk\ kapacitet = M * P * U = \frac{3600}{100} * \frac{100}{90} * 0{,}5 = 36 * 1{,}11 * 0{,}5 = \approx 20\ stycken$$

Därför kan det konstateras att den ideala kapaciteten är oförändrad medan den faktiska kapaciteten har förbättrats och närmat sig den ideala kapaciteten.

2.2 Metod & modell för att visualisera produktionsförbättringar

Hedman (2013) beskriver en egenutvecklad konceptuell modell av ett produktionssystem som i det här projektet kallas informationsmodellen. Den här delen ämnar förklara logiken och sambanden bakom informationsmodellen, vilket innefattar dess olika delar samt en förklaring av hur dessa hänger ihop. Dess beståndsdelar är anläggningar och tillverkningsprocesser som i sin tur bryts ner i mindre beståndsdelar.

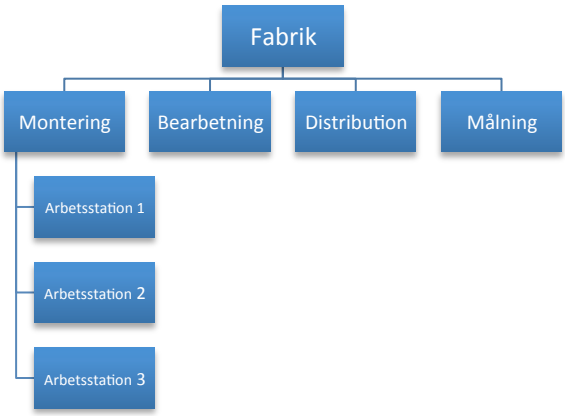
Anläggningen består av en fabrik som utgör området där produktionssystemets tillverkning utförs. Fabriken är den högsta nivån i informationsmodellens hierarki och den har undernivåerna subsystem och arbetsstationer. I figur 4 visas en konceptuell modell över ett produktionssystem där fabriken är den högsta nivån. Fabriken delas i sin tur in i subsystemen montering, målning, bearbetning och distribution. Subsystemet montering kan i sin tur delas in i arbetsstationer.



Figur 4. Principiell bild över en fabrik med tillhörande subsystem. Subsystemet för monteringen har i sin tur delats in i arbetsstationer (Hedman, 2013).

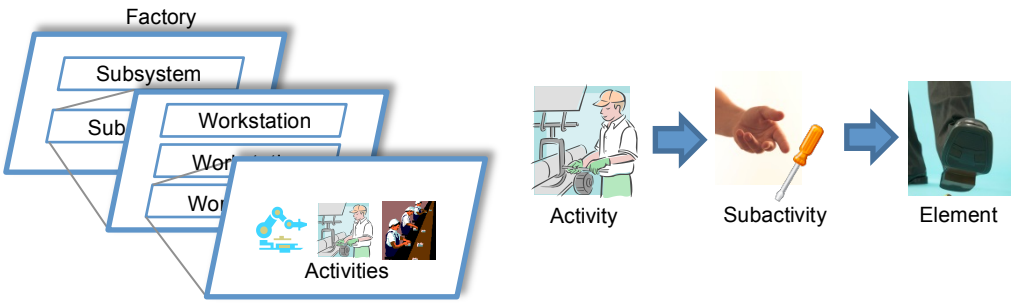
Tillverkningsprocesser utförs i anläggningen och processen kan analyseras från anläggningens samtliga hierarkiska strukturnivåer, se figur 5 för ett exempel. Detta betyder att tillverkningsprocessen på fabriksnivå kan ses som hela processen för att konvertera råmaterial till

slutprodukter. På subsystemsnivå avgränsas tillverkningsprocessen till att innehålla delmoment där en avdelning till exempel kan vara montering. På arbetsstationsnivå delas tillverkningsprocessen in i ännu mindre delar, vilket applicerat på monteringen innebär att monteringen delas in i delmoment.



Figur 5. Figuren beskriver den hierarkiska strukturen som exemplifierades i figur 4.

Aktiviteter är momenten som utförs i tillverkningsprocessen för att transformera till exempel råvaror till färdiga produkter. Aktiviteter kan i sin tur delas in i subaktiviteter som delas in i element, vilket figur 6 visar.

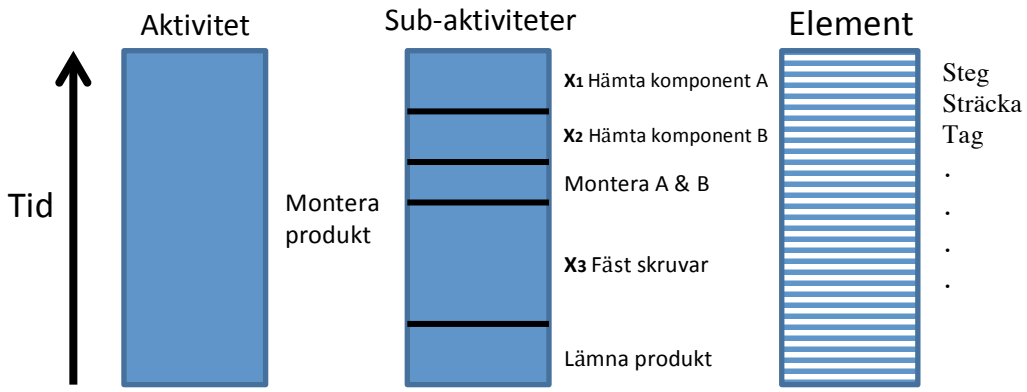


Figur 6. Figuren illustrerar informationsmodellens hierarki. Fabriken är överst i hierarkin med ett subsystem under. Subsystemet har i sin tur arbetsstationer som undernivå. Vid arbetsstationen utförs aktiviteter som i sin tur består av subaktiviteter och element som bryter ned tillverkningsmetoden i mindre rörelser. Anpassad: från Bellgran and Säfsten (2005)

Figur 7 illustrerar det genom att dela upp aktiviteten i subaktiviteterna hämta komponent A, hämta komponent B, montera ihop AB, fäst skruvar och lämna produkten. Tidsdrivare är antalet gånger som respektive subaktivitet utförs och dessa tillsammans med subaktiviteterna utgör en tidsekvation. Tidsekvationer används för att identifiera vilka aktiviteter och subaktiviteter som bidrar till tiden det tar att producera en produkt. Tidsekvationen för produkt AB kan uttryckas:

$$X_1 * Hämta\ A\ komponent + X_2 * Hämta\ B\ komponent + Montera\ ihop\ AB + X_3 * Fäst\ skruvar + Lämna\ produkt$$

X är tidsdrivarvariabeln och illustrerar antalet gånger som subaktiviteten utförs. Tiden det tar för att utföra subaktiviteten baseras på standardtider som i informationsmodellen baseras på SAM-standarden. Därför bryts subaktiviteterna ned i element för att standardtiden ska fås, vilket illustreras i figur 7.



Figur 7. Figuren visar att aktiviteter består av subaktiviteter. Subaktiviteterna bestå i sin tur av element (Hedman (2013).

När en standardtid och den faktiska tiden har räknats ut för en aktivitet kan en prestationsbedömning utföras. Prestationsbedömningen anges i procent och är högre än 100 % om tillverkningsresursen håller ett högre tempo än vad MTM-standardtiden förespråkar. MTM-standarden är baserad på ett tempo som en normalperson ska klara av 8 timmar per dag i resten av sitt liv utan att bli utmattad eller skadad.

Resursutnyttjandegraden fås med hjälp av frekvensstudier för att bedöma hur stor andel av arbetstiden som resursen utför följande aktiviteter:

- U_M - Planerade aktiviteter
- U_N - Personlig tid
- U_S - Systemförluster
- U_D - Störningar

Genom att invertera tiden det tar att utföra aktiviteten, alltså ta fram cykeltiden, samt därefter multiplicera detta med 3600 fås den ideala kapaciteten per timme (M). Om detta kombineras med resursernas utnyttjandegrad (U_M) och prestation (P) fås den faktiska kapaciteten, vilken uttrycks nedan:

$$CAP_R = M * P * U_M$$

Produktionskapaciteten i ett idealt produktionssystem uttrycks:

$$CAP_I = M$$

2.3 Ingenjörsmässig kravhantering

Requirement Engineering är kravhantering med ingenjörsmässig metodik. Det är en kravhanteringsmetod som används för att ta fram och följa upp krav vid mjukvaruutvecklingsprojekt. Kraven skall behandla projektet och den mjukvara som utvecklas. I projektet definieras Requirement Engineering som ingenjörsmässig kravhantering.

Vidare kan Requirement Engineering betraktas som en process som innebär att gå från ett problem till en lösning, vilket syftar till att säkerställa att denna process skall fungera på bästa möjliga vis. Arbetssättet ska säkerställa att lösningen eller den slutgiltiga leveransen är av hög klass. Ett steg i processen är att ställa krav på föremålet som ska utvecklas. Dessa krav skall vara uppfyllda av den slutgiltiga lösningen för att problemet skall anses löst (Docker, 1998). För att framgångsrikt ta fram

en lösning är det till stor hjälp att tidigt urskilja vad mjukvaran utför och varför det utförs, alltså försöka ha en effektiv kravhantering. Om ett projekt har ett krav som lyder ”*Produktleverans skall ske senast 9 mars, 2014*” är detta ett krav på projektet, inte ett krav på produkten. För produkten innebär detta en begränsning (Docker, 1998).

Att utforma en kravspecifikation blir en viktig del av utvecklingsprocessen i nya produkter. Speciellt för att utveckla produkten till det som efterfrågas och önskas utan att få med funktioner som inte är önskade (Lauesen, 2002). Det är inte ovanligt att det blir svårigheter i projekt med kommunikationsgap mellan avdelningar som tillsammans skall utveckla en mjukvara. Kravspecifikationen skall hjälpa till att samla intressenterna i projektet på samma spår, innan dess att mjukvaruutvecklare kommit för långt i processen av att skapa produkten, utan att egentligen veta vad som önskas (Pressman, 2005). Dessutom undviks onödiga kostnader senare i projektet. Kostnaden ökar markant allteftersom projektet fortskrider och fel bör minimeras så tidigt som möjligt i processen (Hood, 2008).

Definitionen av Requirement Engineering är inte helt självklar och oproblematisk. Förr när mjukvaruutvecklare skulle utveckla en mjukvara blev det i extremfall att de behövde gissa vad kunden hade för behov. Detta gjorde att flera program inte kom att användas för att de inte tillförde nytta för kunden och denne hade varit tvungen att ändra sitt arbetssätt alltför mycket för att använda det. Detta ledde fram till att olika metoder utvecklades för att lättare förstå kundens behov och specificera det för programmering (Hood, 2008).

Metodernas utveckling och djupare insikt i Requirement Engineering har resulterat i att man idag ser till variationen av intressenter till skillnad mot tidigare där det var vanligt att bara se till uppdragsgivare eller beställarens krav. Nu är det vanligare att alla som på något sätt berörs av produkten skall inkluderas på ett eller annat sätt redan vid framtagningen av kravspecifikationen. Det kan vara fördelaktigt att tidigt komma överens om vad mjukvaran skall innehålla och inte, kopplad till uppskattad budget för varje önskemål. På så sätt kan mindre viktiga önskemål tidigt elimineras och fokus läggs på projektets huvudfunktioner (Hood, 2008).

2.3.1 Ingenjörsmässig kravhantering i praktiken

Pressman (2005) beskriver Requirement Engineering kort som en brygga mellan design och skapandet av mjukvara. Det innefattas av att formulera, dokumentera och hantera mjukvarukrav. Software Requirements, en gren av Software Engineering, är mer specifikt inriktat på kravställande och arbete med kravspecificering efter intressenternas behov. Genom att samla deras intressen och identifiera vilka de är tidigt förenklar det vidare arbetet med kravspecifikationen (Lauesen, 2002). Inget projekt är det andra helt likt. För att generalisera starten på ett mjukvaruprojekt är det vanlig att en idé tillsammans med ny teknik triggar uppstarten på ett nytt projekt. De drivs på av koncept eller vision från uppdragsgivare, också kallad kravställare. För att få hjälp med första delen av mjukvaruutvecklingen tas hjälp av kravanalytiker som har som uppgift att formulera krav efter kravställarens formulering av problemet, hanterar dem för att till sist lämnas vidare till utvecklare av mjukvara. Samtliga blir intressenter i projektet.

Som hjälp att identifiera intressenter kan man till figur 8 knyta dem till stegen i projektfaserna. Kravställare eller uppdragsgivaren tar hjälp av analytiker för att påbörja arbetet med kravspecifikationen. Dessa tillsammans itererar fram krav genom flera steg av validering. Nästa

person som kopplas till mjukvarans utveckling är mjukvaruutvecklaren som tar hand om design, program och test (ev. tillkommer fler grupper som tar hand om tester, kontrakt i resp. ruta, här förenklas det för förståelse). Projektfaserna kan sammanfattas (Pressman, 2005) med följande sju steg med respektive svårigheter som förklaras nedan:

- Uppstart

  Hur projektet startas. Kan var en händelse eller teknisk utveckling som leder till att uppstarten, ser väldigt olika ut från fall till fall. Uppgiften här är att få så bred förståelse av problemet som möjligt utan att gå in på lösningen. Att tidigt i projektet få klart för sig varför mjukvaran utvecklas, för vem och var det skall användas är exempel så sådana frågor.
- Framtagning

  Att påbörja framtagningen av önskemål på mjukvaran genom att fördjupa frågorna från uppstarten. Bryta ned önskemål och försöka få med praktisks detaljer för att börja arbeta på utformning av krav. Problemområden brukar ligga i att skapa systemgränser genom avgränsningar. Att skapa förståelse mellan intressenter om vad som skall utformas och samla deras åsikter. Att organisera och dokumentera hjälper till med att undvika svårigheter, redan tidigt i processen.
- Utarbetande

  Ytterligare förfina detaljnivån av tidigare insamlad information från intressenter och påbörja utvecklingen av tekniska modeller (Pressman, 2005) med funktioner, särdrag och begränsningar kopplade till krav. Att ta fram modeller över informationsflöden och hur mjukvaran skall bete sig vid interaktion.
- Förhandling

  Vid det här laget har listan på krav vuxit och det är vanligt att det uppstår konflikter mellan krav som motsäger varandra. Då får analytikern se över hur arbetet skall fortskrida, förändra och se över existerade krav för att antingen undvika att de går emot varandra eller kanske ta risken att helt ta bort ett krav. Förhandlingen går till genom att väga krav mot varandra och uppskatta eventuella förändringar i de respektive kraven.
- Specifikation

  Nu specificeras kraven ned till modeller som eventuellt kan vara grafiskt se ut, matematiska lösningar, en grov prototyp, användarscenarion eller olika kombinationer av dessa. Projekt skiljer sig mycket från varandra där mängden självklara och ospecificerbara krav kan variera kraftigt. Olika analytiker gör på olika sätt (Lauesen, 2002) och det är upp till användare och analytiker av kravspecifikationen att bestämma lämplig nivå. Det är en risk att för mycket tas med som får dokumentet att bli svårförståeligt och tappa sin funktion.
- Validering

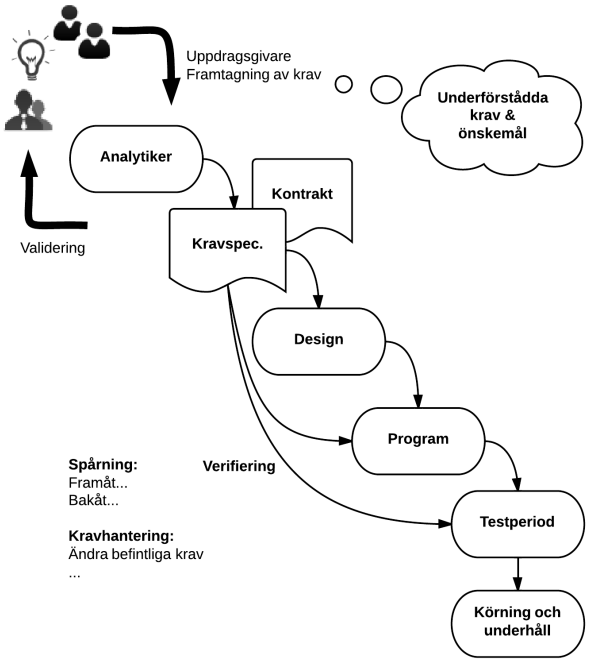
  Hur skall man veta att kraven är rätt ställda och på en lämplig nivå? Det blir viktigt att utvärdera de olika kraven. Försöka eliminera inkonsekvens, felskrivningar som orimliga krav och säkerställa att felaktigheter hittas. Kraven önskas beskriva produkten och med hjälp av intressenter går igenom kraven för att utvärdera och validera kravets betydelse.

- Kravhantering

  Krav förändras, speciellt när det rör sig om mjukvaror som uppdateras. Det blir därför ett processteg att hantera existerande krav för olika produkter för att de ska anpassas till uppdateringar och förändringar i och utanför mjukvaran. För att hantera kraven kan det vara av hjälp att sortera dem efter andra kriterier. Det kan göras genom att länka samband mellan funktioner eller hela grenar i en mjukvara till ett program och redovisa dem i tabeller.

2.3.2 Att kravställa mjukvara

Software Requirements (Lauesen, 2002) berör de olika stilistiska metoder att presentera krav och när de är lämpliga. Användningen av en kravspecifikation varierar och förändras genom produktens livscykel. Beroende på var i processen projektet befinner sig kan kravspecifikationen användas på flera sätt. Att det finns flera användningsområden med en kravspecifikation gör framtagningsprocessen upplevs av analytiker som komplicerad. Det ska tilläggas att utförligheten i kravspecifikation och den tid som läggs ned tidigt i projektet ofta fås igen i senare steg.



Figur 8. Funktioner hos kravspecifikation illustrerad med vattenfallsmodellen. Inspiration från Lauesen (2002)

I figur 8 visas funktionerna hos kravspecifikationen tillsammans med vattenfallsmodellen. Trots att vattenfallsmodellen inte alltid är metoden som används för framtagning av krav går alla utvecklingsprojekt går igenom samma steg. Därför är det en beskrivande figur att använda för att förklara funktionerna hos kravspecifikationen. En summering av rollerna hos kravspecifikationer för en mjukvara är validering, verifiering, spårning, kravhantering och rättsfrågor (Lauesen, 2002). Utöver detta finns fler användningsområden, för dem hänvisas till kapitel 7 i Lauesen (2002).

Första användningsområdet blir validering. Kunden får möjlighet att reflektera över de krav som finns och få känsla ifall dessa är tillräckliga för att börja konstruera produkten, alltså att det förklarar de önskade funktionerna tillräckligt. Om inte itereras processen tillbaka ett steg som tillåter analytiker lägga till eller förtydliga de krav som behövs (Pressman, 2005). Ett problem som vanligt uppstår vid valideringen är att intressenter har olika syn på hur vad som är ett underförstått och självklart krav eller inte. Kravspecifikationer kan i praktiken inte specificera allt (Lauesen, 2002). Kraven blir då ett långt dokument, i princip i punktform, där det dels blir för mycket jobb att ta fram kraven samtidigt som användningen av dokumentet försämras. Det är underlättande för uppdragsgivare och analytiker att tillsammans kommer överens med de som skall utföra data-, design- och mjukvaruutveckling över vad som ses som självklart och inte i kravspecifikationen.

Verifiering kan göras kontinuerligt eller efter det att mjukvaran har utvecklats beroende på vilken utvecklingsmodell projektet följer. Vanligtvis sker detta i en avsedd testperiod där man går igenom kraven från början till slut för att stämma av att mjukvaran faktiskt uppfyller kraven.

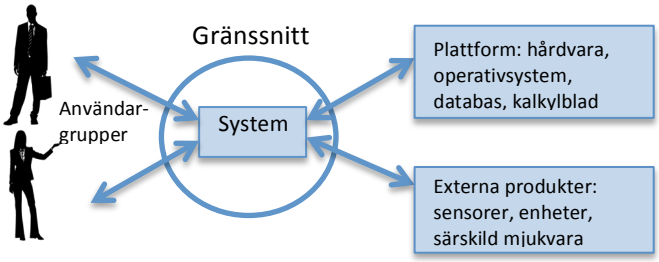
Spårning är rätt likt validering- och verifieringsstegen men innebär att man mer aktivt söker igenom kraven för att se att det finns koppling mellan önskemål- krav- och mjukvara. Framåtspårning går man från krav till mjukvara och från kundönskemål till krav, framåt i utvecklingsprocessen. Bakåtspårning är motsatsen där man strävar bakåt från krav till önskemål och från mjukvara till krav. Det sistnämnda används som verktyg för att undvika att slösa resurser på sådant som inte skapar mervärde för kund.

Kravhantering är precis som namnet antyder hur krav hanteras och organiseras. Det är ovanligt att kraven blir helt rätt från början, att ha en systematik som gör det enkelt att spåra och ändra ett krav effektiviserar processen och underlättar arbetet. Det behandlar också versionsförändringar och förnyelse av mjukvaran över tiden.

Kravspecifikationen skall vara ett dokument som beskriver vad mjukvaran skall utföra. När det är fullt specificerat används en kravspecifikation som underlag vid outsourcing och utvecklingen av själva mjukvaran. Kravspecifikationen kan således också användas som bevisning vid rättsliga tvister. Det händer att mjukvaruutvecklare och uppdragsgivare har meningsskiljaktigheter om en utvecklad programvara innehåller de funktioner som har specificerats eller inte (Lauesen, 2002). Därför blir det viktigt att ett komplett underlag efter standard för att hålla sig borta från sådana tråkigheter. Det kan tilläggas att i stora delar av världen gäller att domstol har bra känsla för vad som räknas som lagom detaljeringsnivå av krav. Beställaren har i rättsfall tendens att vinna sådana fall (Lauesen, 2002).

2.3.3 Innehåll i kravspecifikationer

Utformning av kravspecifikationer är i teorin relativt rakt fram, specificera input-parametrar till systemet och output för respektive input. I praktiken är det svårare att specificera tillräckliga krav i rätt utsträckning och avgränsa detaljeringen. Som hjälp finns de teoretiska modellerna (Lauesen, 2002). I figur 9 visas systemet som ett blackboxsystem och de delar som bör ingå i kravspecifikationen enligt Lauesen (2002):



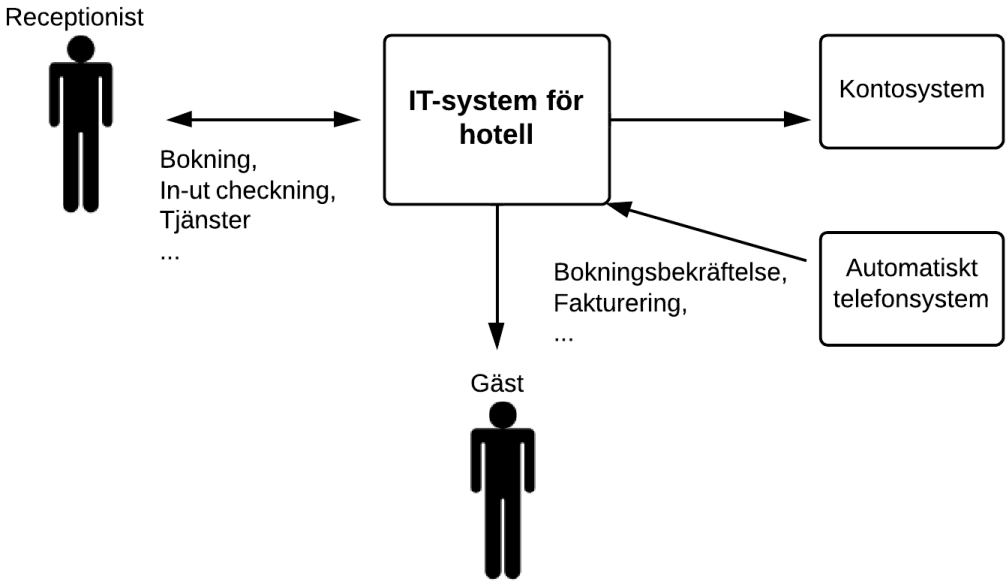
Data krav:
Systemtillstånd: Databas,
kommunikationskanaler, input- output format
Funktionella krav, varje gränssnitt:
Spela in, beräkna, omvandla, överföra
Teori: F(input, tillstånd) -> (output, tillstånd)
Funktionslista, pseudokod, aktivitetsdiagram,
skärmdump, stöduppgifter xx till yy

Kvalitetskrav:	**Organisationskrav:**
Prestanda	Leveranstid
Användarvänlighet	Rättsliga
Underhåll	Utvecklingsprocess
…	…
Övriga resultat:	**Hjälpa läsaren:**
Dokumentera	Företagsmål
Installera	Definitioner
Konvertera	Diagram

Figur 9. Innehåll i kravspecifikationer enligt Lauesen (2002) som blackbox

Genom att se på systemet och förklara systemgränserna för respektive gränssnitt ges förståelse för hur mjukvaran önskas bli. Krav ställs med olika mål. I figur 9 visas typerna som krav på data, funktionella krav, kvalitetskrav, och organisationskrav tillsammans med övriga resultat. Krav på data, funktionella krav och kvalitetskrav som är de krav som generellt brukar vara svårast att ställa för produkten. Lauesen (2002) ger exempel på hur dessa krav kan formuleras. Vanligast förekommande stil på funktionellt krav (Lauesen, 2002) som kallas funktionskrav. Då beskrivs funktionen i text genom att fokusera på att beskriva funktionen i text. Att hålla sig till endast en stil i kravställande blir lätt bristande. Ett funktionellt krav har svårt att ge djupare teknisk detaljering och kan därför kompletteras med flera olika metoder för att ge en bättre förståelse och djup i kravet. Det kan vara algoritmer eller case som beskriver stegen användaren går igenom, eller uppgiftsbeskrivningar som beskriver processen i punktform. Att ge kombinationer av de ovanstående skriver Lauesen (2002) att de tillsammans underlättar att formulera kraven men också för förståelsen.

Att inkludera förklaringar med hjälp av bilder och sammanhangsdiagram underlättar för läsaren och ger bredare förståelse av funktionen och de olika informationskanalerna som önskas. Sammanhangsdiagram beskriver systemgränser, användargrupper, informationskällor så som sensorer. Det kan också gärna belysa interaktionen mellan mjukvara, människa och databas. De finns i olika variationer och detaljeringsgrader där det är analytikerns uppgift att sätta rätt nivå på kravet där de används. Exempel för ett IT-system för hotell kan ses i figur 10 med dess kommunikationskanaler.



Figur 10. Ovan visas kommunikationen för ett hotellsystem, en typ av sammanhangsdiagram med inspiration från Lauesen (2002).

Genom att ha försiktighet vid kravställande och tydlighet i vad mjukvaran skall uträtta med hjälp av diagram, beskrivningar och begränsningar underlättar det för mjukvaruutvecklare i processen att skapa mjukvaran. Specificeras det för mycket om begränsningar och om vilka databaser som skall användas mm. förhindras utvecklingen av smarta system. Specificerar analytikern för lite kan det bli komplicerat att bygga en systemarkitektur som uppnår kravställarens mål. Det är en balansgång för att få rätt detaljnivå som övervinns med hjälp av kommunikation mellan intressenter.

2.4 Mjukvaruutveckling

I kapitlet beskrivs den teori som är relevant för egenutvecklade mjukvaror. Först ges en introduktion till programmering med fokus på de olika grundläggande metoder som finns. Därefter fokuserar teorin kring objektorienterad programmering och de tekniker som kan användas vid mjukvaruutveckling.

2.4.1 Projektmodeller för mjukvaruutveckling

Då en mjukvara ska utvecklas finns det många sätt att gå till väga för att planera och strukturera projektet. Några av dessa möjliga projektmodeller tas upp nedan. Modellerna är uppbyggda av samma steg, men kopplingen mellan dessa steg är det som skiljer modellerna åt.

2.4.1.1 Vattenfallsmodellen

Vattenfallsmodellen är den första publicerade modellen för mjukvaruutveckling (Pressman, 2005, Sommerville, 2011). Det är en av de enklaste modellerna och är uppbyggd av linjärt ordnade faser, som alla hanterar distinkta och separata delproblem. Varje fas påbörjas när föregående fas avslutats, faserna avslutas inte innan de kan verifieras och valideras. Genom att gå till väga på detta sätt bryts utvecklingen av mjukvaran ner i tydliga, mindre steg som underlättar utvecklingsprocessen (Jalote,

2009). Faserna kan sammanfattas i kommunikation, planering, modellering, konstruktion och användning (Pressman, 2005).

Det finns begränsningar med denna modell där två av de viktigaste är att kraven förutsätts vara oförändrade genom hela processen och att alla krav måste specificeras från början. Den första begränsningen beror på att det är ett orimligt antagande vid framtagning av nya produkter att alla krav från början är kända. Nackdelarna hos den andra begränsningen ligger i att det inte får tillkomma några krav eller förändringar av krav efter hand. Det kan innebära att kunden lägger till krav som i början av projektet anses viktiga men som senare visar sig vara överflödiga. (Jalote, 2009)

Enligt Pressman (2005) kan modellens linjära faser också leda till att arbetsgruppen blockeras internt så att gruppmedlemmar tvingas vänta på varandra medan deras arbete färdigställs. Denna väntetid kan överskrida den tid som läggs ned på själva utvecklingsarbetet.

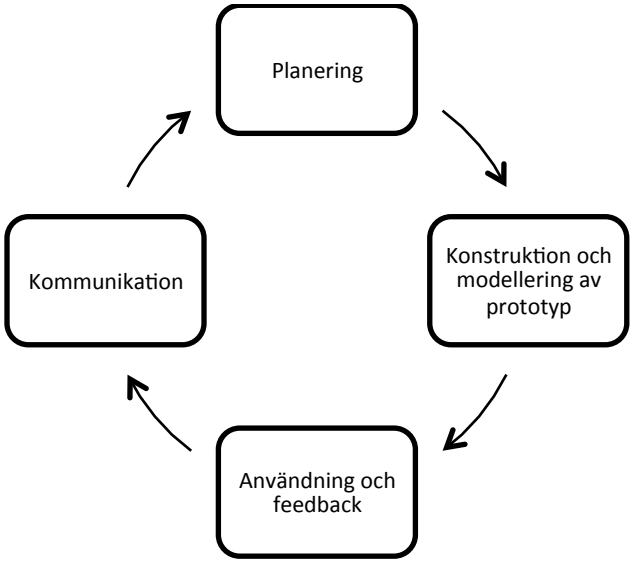
Trots begränsningarna är vattenfallsmodellen enligt Jalote (2009) den modell som genom tiderna använts mest för mjukvaruutveckling. Idag används den mest vid rutinuppdrag eftersom kraven då är lätta att förstå.

2.4.1.2 Prototyping

Prototyping är en evolutionär modell som kan användas ensam men som oftare används tillsammans med en eller flera andra modeller. Definitionen av en evolutionär modell är att det är en iterativ modell som utvecklar mjukvaran så att den gradvis blir mer komplett. Två stora nackdelar med dessa modeller är att projektet är svårt att planera eftersom antalet iterationer är okänt samt att processens fokus mer ligger på flexibilitet än hög kvalitet (Pressman, 2005). Fördelen med en evolutionär modell är däremot att kravspecifikationen gradvis kan utvecklas och förbättras i takt med att användaren/kunden bättre förstår vad denne vill ha ut av mjukvaran (Sommerville, 2011). Det leder till en mjukvara som bättre uppfyller kundens önskemål.

Prototyping är bra för de fall där kraven är oklara och inte helt definierade (Pressman, 2005) och kan vara svåra att ta reda på (Jalote, 2009). Målet med modellen är att motverka vattenfallsmodellens första begränsning; att kraven inte får förändras (Jalote, 2009).

Processen kan ses i figur 11 och startar med kommunikation mellan utvecklare och kund. De definierar tillsammans mjukvarans övergripande mål, identifierar de krav som för tillfället är kända samt identifierar de områden som kräver ytterligare förklaring (Pressman, 2005).



Figur 11. Projektmodellen prototyping. Bild inspirerad från (Pressman, 2005).

Nästa steg är att ta fram en enklare prototyp för att bättre förstå kraven (Jalote, 2009). Planeringen av denna prototyp fokuserar på de aspekter av mjukvaran som kunden ser, kan ta till sig och ge feedback på. Sedan konstrueras prototypen som kunden får använda och utvärdera. Genom att testa denna prototyp kan kunden få en bättre känsla och förståelse för systemet och tala om för utvecklaren vad som bör ändras, läggas till och bevaras (Jalote, 2009). Feedbacken används för att förbättra prototypen vilket görs dels för att komma närmare att uppfylla kundens mål och dels för att utvecklaren bättre ska förstå vad som måste göras. För varje iteration fås en mer utvecklad prototyp (Pressman, 2005).

Att kunden själv får testa och utvärdera resulterar i mer stabila krav som inte ändras lika ofta. De förändringar som kunden ser som önskvärda implementeras i mjukvaran innan kunden återigen testar och utvärderar mjukvaran. Iterationen fortsätter till dess att det inte längre är värt att lägga ner tid och pengar på att förändra prototypen (Jalote, 2009).

En nackdel med denna modell kan vara att kunden inte förstår att prototypen är långt ifrån en färdig mjukvara. Det kunden ser är att prototypen nästan är färdig när utvecklaren ser att prototypen bara är provisoriskt uppbyggd för att visa dess viktigaste funktioner. En annan nackdel kan vara att utvecklaren väljer ett sämre alternativ endast av anledningen att det går fortare att implementera prototypen på detta sätt men att utvecklaren sedan glömmer varför detta sätt var dåligt. De dåliga valen har då blivit en del av den slutliga lösningen (Pressman, 2005).

Trots sina problem är detta ofta en effektiv modell. Det är dock viktigt att kunden och utvecklaren redan från början kommer överens om att prototypen endast kommer vara just en prototyp och till största delen användas för att ta fram krav. Prototypen kommer sedan förkastas, medan kunskapen som erhållits genom mötena med kunden tas med till framtiden. För att utveckla mjukvaran till en slutlig mjukvara krävs omfattande programmering (Pressman, 2005).

2.4.1.3 Spiralmodellen

Även denna modell är en evolutionär modell och har därför samma övergripande för- och nackdelar som prototyping. Modellen kombinerar den iterativa naturen hos prototyping med de kontrollerade och systematiska aspekterna av vattenfallsmodellen. Spiralmodellen är alltså en iteration genom

vattenfallsmodellens olika faser, vilket är mer realistiskt än vattenfallsmodellens linjära arbetssätt (Pressman, 2005).

Varje varv i spiralen motsvarar en fas i utvecklingen av mjukvaran. Ett varv kan till exempel fokusera på kravställning, ett annat på funktionalitet och ytterligare ett annat varv kan fokusera på systemdesign. Varje varv är indelat i fyra olika delar; målsättning, riskanalys och riskreducering, utveckling samt validering och planering. Spiralmodellen trycker på vikten av att identifiera risker (Sommerville, 2011).

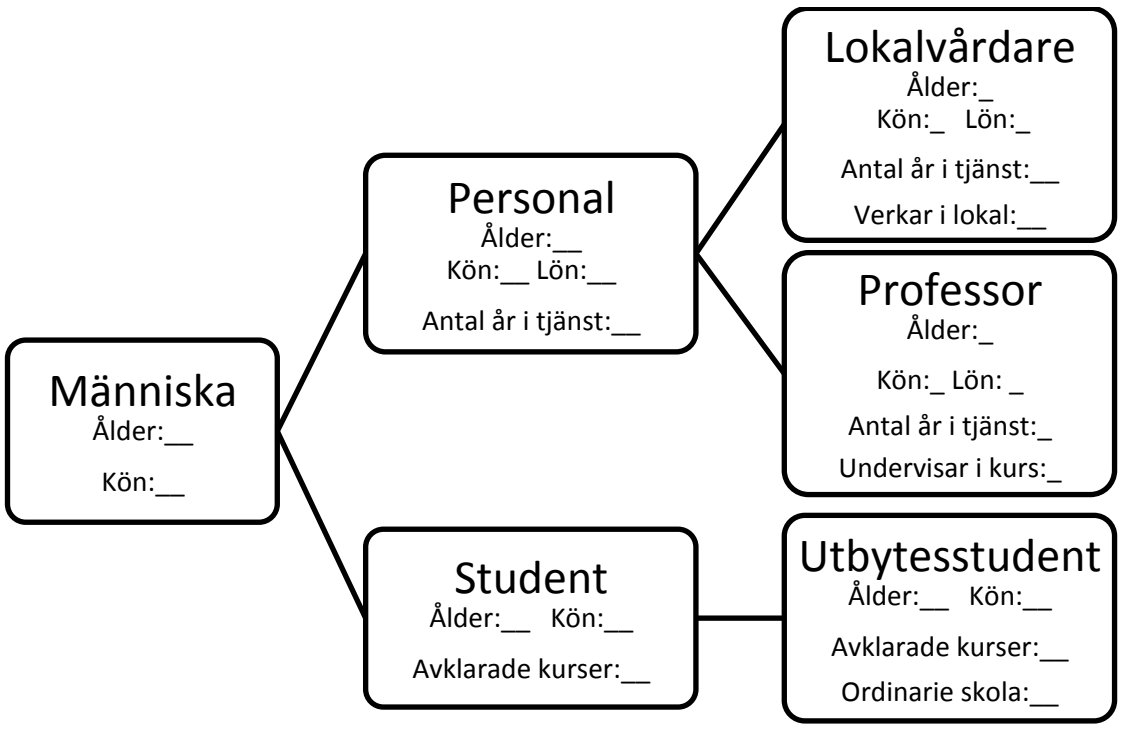
Modellen kan användas genom mjukvarans hela livslängd. Initialt används spiralen till att utveckla mjukvaran medan senare varv används för att förbättra den redan existerande mjukvaran (Pressman, 2005).

2.4.2 Objektorienterad programmering

Den objektorienterade metodiken innebär att koden delas upp i klasser, också kallade objekt. Ett objekt är en gruppering av de egenskaper och funktioner som är relevanta för just det objektet. Vid objektorienterad modellering av ett produktionssystem motsvarar alltså varje del i produktionssystemet ett objekt i modellen med vissa egenskaper såsom namn och plats. Varje specifik instans av dessa delar, till exempel en specifik arbetsstation eller aktivitet skapas sedan som en kopia av det objektet och egenskaperna tilldelas de variabler som gäller för just den instansen. Det gör det enkelt att överblicka mjukvaran, eftersom allt är grupperat. De tydliga gränserna mellan objekten ger också möjlighet till olika abstraktionsnivåer, det vill säga att endast visa relevant information för utomstående. (Clark et al., 2013)

Ett vanligt förfarande vid objektorienterad modellering är att klasser ärver varandra. Att en klass ärver en annan klass innebär att den ärvda klassen, också kallat subklassen, har samma egenskaper och funktioner som den klassen den ärver, också kallat basklassen, och kan lämpa sig väl då flera objekt är nära besläktade. (Clark et al., 2013)

För att modellera människors olika sysselsättning på Chalmers skulle man till exempel kunna utgå från en människoklass som basklass där varje människa har ett antal egenskaper. En personalklass och en studentklass ärver sedan den klassen med ett antal för de kategorierna specifika egenskaper såsom lön för personalen och avklarade kurser för studenten. Ytterligare klasser för de olika yrkeskategorierna som en professorklass och en lokalvårdareklass ärver sedan personalklassen, med tillägg för till exempel kurser som undervisas för professorer och i vilka byggnader lokalvårdaren verkar. En klass för ordinarie studenter och en för utbytesstudenter ärver studentklassen eftersom en utbytesstudent har ett antal andra egenskaper se figur 12 såsom vilken skola denne tillhör.



Figur 12. Ovan illustreras hur en objektorienterad modellering skulle kunna utformas med arvstekniken för att illustrera människors på Chalmers olika sysselsättningar.

När personerna på Chalmers tekniska högskola sedan modelleras kan objekt av valfri typ skapas. För alla människor som bara befinner sig på Chalmers tekniska högskola utan att vara varken anställda eller studenter kan ett objekt av typen "människa" skapas. Det kan då tilldelas alla attribut som är definierade för en människa, till exempel ålder och kön. Personal kan antingen skapas som ett rent personalobjekt med de definierade attributen eller, om det är en professor eller lokalvårdare, skapas ur ett sådant objekt som ärver personalklassen och således även har attributen lön och antal år i tjänst. En student kan antingen skapas ur studentklassen, eller ur den ärvda klassen utbytesstudent om det bättre beskriver personen.

Eftersom alla ovan nämnda klasser ärver människoklassen har alltså även så väl personal som student liksom professor och utbytesstudent alla egenskaper definierade för människoklassen. Professorklassen har utöver det alla egenskaper för personalklassen, men personalklassen har inte de specifika egenskaperna hos professorn eftersom det till exempel inte är logiskt att ett cafébiträde ska ha egenskapen "*kurser denne undervisar i*".

2.4.3 Objektorienterade programmeringsspråk

Det finns idag ett antal vida använda objektorienterade språk såsom C++, Java och C# med olika för- och nackdelar. C++ är en ren påbyggnad av programmeringsspråket C, och låter precis som sin föregångare programmeraren komma väldigt nära maskinvaran som programmeras till exempel i termer av exakt var och hur data ska sparas. Tack vare det är det möjligt att skriva optimerade mjukvaror, men i gengäld kräver det att programmeraren själv tar ansvar för vad som ska hända med till exempel tillfälliga variabler då de inte längre behövs. Java är ansett att vara betydligt lättare att programmera i och kan likt C++ användas för att skriva mjukvaror till en lång rad enheter från datorer till mobiltelefoner. Den mångsidigheten påverkar dock prestandan negativt och de beräkningar och operationer mjukvaran gör tar längre tid med samma processor. Betydligt bättre prestanda har då

språket C# (uttalas c sharp) som i princip kan exekveras på alla datorer med Microsofts operativsystem Windows, men till priset av mångsidigheten och plattformsoberoendet som återfinns Java. (Skansholm, 2010, Clark et al., 2013)

Värt att notera är att en stor del av den funktionalitet som går att realisera i ett datorprogram även går att göra web-baserad. Så är även fallet med C# som delar många funktioner med ASP.NET som även det är utvecklat av Microsoft och som utvecklas i samma miljö som C#. Detta innebär att det är relativt enkelt att realisera ett program skrivet i C# som en websida.

2.4.4 Programmera för robusthet

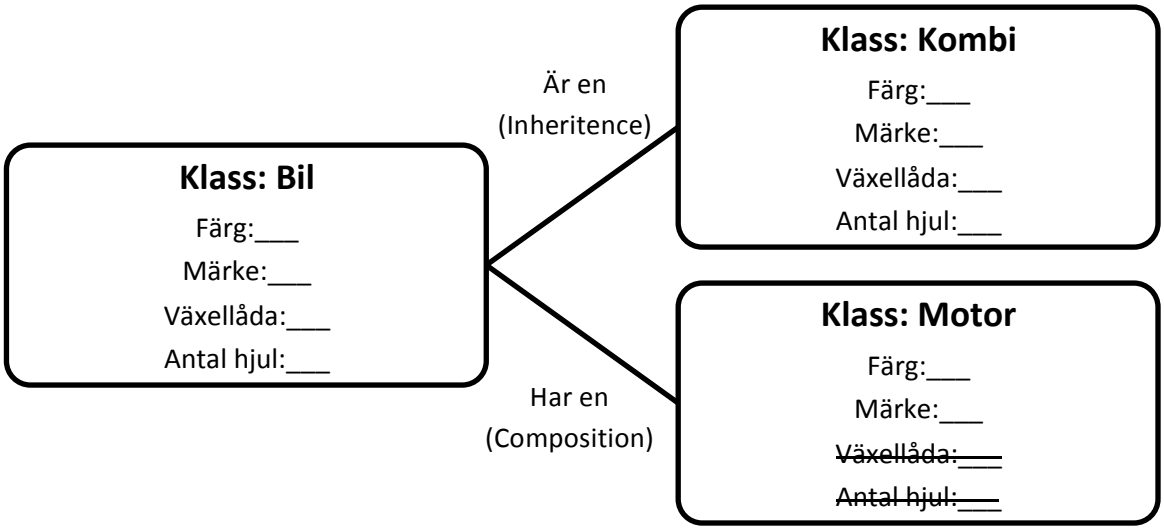
Hur väl mjukvaran klarar av oförutsedda händelser brukar benämnas robusthet. Oförutsedda händelser kan innefatta så väl felaktigt användande som buggar i koden. En god undantagshantering med på förhand specificerade rutiner för vad som hända för givna fel är viktigt för att mjukvaran inte ska krascha eller generera felaktig data. Synligheten hos objekt och variabler för andra delar av mjukvaran är också viktigt att ta i beaktning. I praktiken innebär det att dessa ska deklareras på ett sådant sätt att det inte går att komma åt variabler som endast är aktuella för en specifik klass från en annan klass av misstag.

2.4.5 Designfilosofier

Två olika designfilosofier i objektorienterad programmering är inheritence och composition. Inheritence innebär att klasser ärver varandra som illustrerades i exemplet ovan medan composition innebär referenser mellan fristående klasser för att visa dess relationer.

Även om det kan kännas lockande att låta klasser som är liknande ärva varandra, som nämnts ovan är det inte alltid en god idé. Då relationerna de olika klasserna mellan definieras när mjukvaran kompileras kan de alltså inte ändras när mjukvaran exekveras, vilket gör att flexibiliteten för användaren blir lidande. Banden mellan de olika klasserna är också så starka att en förändring av basklassen kan ge oönskade och oväntade förändringar i beteendet hos subklasserna. Robustheten blir också lidande eftersom variabler i basklassen per automatik blir synliga och ändringsbara för subklassen. (Cooper, 2002)

Composition bygger på referenser mellan olika klasserna istället för arv. Basklassen innehåller då referenser till berörda subklasser, som är helt fristående klasser. Nackdelen med metoden är att mer kod behöver skrivas flera gånger, men å andra sidan är det ofta möjligt att återanvända klasserna genom att utforma metoderna ur ett generellt perspektiv. Det finns dock emellertid mest fördelar med metoden varav flexibiliteten kanske är den största. Klasser tenderar att behöva ändras över tid, och även om klasser initialt kan ha samma egenskaper är risken stor att de sambanden ändras (Cooper, 2002). Hur hanterar man till exempel att ett bemanningsföretag ska sköta en del av lokalvården? Skapar man då ytterligare en klass för bemanningsföretag och sedan subklasser som ärver den för alla berörda yrkeskategorierna? Skall olika bemanningsföretag representera olika klasser? Hur låter man en person gå från anställd till inhyrd? Som synes tenderar klasshierarkin att växa i verkliga problem vilket gör mjukvaran svårt att överblicka.



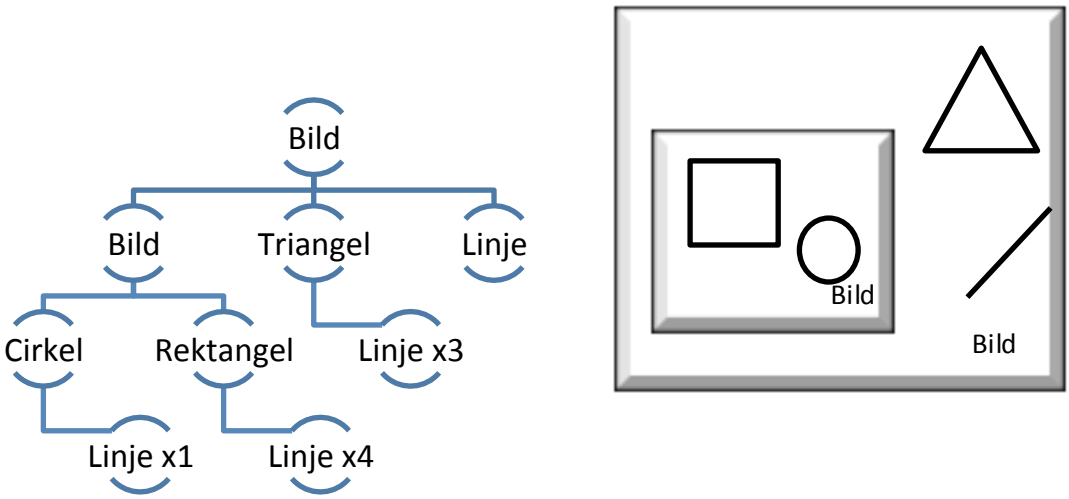
413. I figuren illustreras när det är bra att använda en metod baserad på Inheritence eller Composition.

Figur 13 beskriver ett exempel på när det kan vara bra respektive dåligt att använda arv som metod för att återanvända kod. Eftersom en kombi är en typ av bil kan det vara motiverat att låta klassen kombi ärva klassen bil då alla egenskaper som är relevanta för bilen också är relevanta för en kombi. De ytterligare egenskaper en kombi kan tänkas ha jämfört med en "standardbil", låt säga lastutrymme, kan sedan implementeras endast i kombiklassen. En motor är däremot ingen bil även om en bil innehåller en motor och vissa egenskaper hos bilklassen såsom växellåda och antal hjul är därför irrelevanta för motorklassen. Programmeraren behöver då själv hålla reda på vad som är relevant för respektive klass och risken för fel är överhängande.

2.4.6 Designmönster

Sedan den objektorienterade programmeringen skapades på 80-talet har det tillkommit en lång rad designmönster som underlättar modelleringen av komplexa system. Några av de vanligaste och mest använda publicerades 1994 i boken *Design Patterns* av en grupp kallad *Gang of Four* (Gamma et al., 1994).

Ett designmönster som introducerades i boken är det så kallade composite mönstret som är en praktisk tillämpning av designfilosofin composition nämnd ovan. Boken ger som två exempel på användningsområde dels modellering av hierarkiska strukturer över företag med förhållandet mellan chefer och anställda samt grafiska komponenter som illustreras nedan. Composite mönstret beskriver de ingående delarna som composites och leafs. Varje composite kan ha underordnade i form av andra compositer och leafs. Leafs aldrig har några objekt under sig i den hierarkiska strukturen oavsett vilken nivå de befinner sig på. (Cooper, 2002)



Figur 14. Illustration över exempel inspirerat från exempel i boken Design Patterns. (Gamma et al. (1994)

I figur 14 visas en variant av det exempel som introduceras i boken Design Patterns (Gamma et al. (1994) En figur kan här innehålla antingen andra bilder eller geometriska figurer såsom en triangel, en kvadrat, en cirkel eller en linje. De geometriska figurerna är i det här exemplet leafs, eftersom de i sin tur endast består av koordinater för de vektorer som representerar figurerna och därför inte kan ha några underordnade.

2.4.7 Sparande av användardata

Två principer för att spara användardata är främst förekommande, att spara ned till en databas eller som en datafil. Fördelarna med en databas är att metoden är effektiv även för stora datamängder. Databasen uppmuntrar även till att bara plocka ut relevanta delar snarare än att läsa och skriva alla data varje gång en ändring görs.

Att spara ned användardata till en fil kallas i dagligt tal för att serialisera data och går ut på att mjukvaran skriver användardata till en fil enligt förutbestämd standard, också kallat protokoll. Bortsett från en stor mängd specifika protokoll för en viss datatyp såsom Mp3 för musik eller JPG för bilder förekommer främst två standarder, XML-filer och binära filer. XML står för Extensible Markup Language och har den stora fördelen att data är organiserat och skrivet på ett sådant sätt att den är läsbar för människor. Binära datafiler är däremot omöjliga att tolka för en människa, men det har i gengäld en högre prestanda (MSDN, 2014). Det är också specifikt för .NET applikationer och det går således inte att läsa in information serialiserad i en C# applikation i till exempel en Java-mjukvara.

2.5 Utvärdering av mjukvaror

Utvärdering av mjukvaror är en process för att bedöma hur väl mjukvaran möter slutanvändarens krav eller förmåga att konstruera om mjukvaran så att kraven uppfylls (Punter et al., 2004). Tidigare studier visar att en stor andel av mjukvaruutvärderingsprojekt gett undermåliga resultat (Punter and Lami, 1998). En svårighet med utvärderingsprocessen är att formulera mätbara men förståeliga utvärderingskriterier för att samtliga intressenters förväntningar på projektet ska vara kongruenta (Punter et al., 2004). Wanyama and Far (2008) konstaterar att det finns en stor mängd tillgängliga mjukvaror med olika kvalitet, förmågor och svårigheter med att få mjukvarorna att fungera med

varandra. Detta försvårar utvärderingsprocessen och trots det används många ostrukturerade metoder, vilket skapar behovet för en heltäckande utvärderingsmetod. Comella-Dorda et al. (2002) förklarar PECA-metoden som innefattar stegen planera undersökningen, etablera kriterium, samla in och analysera data som i figur 15.

Figur 15. Figuren beskriver en metodik för att utvärdera mjukvaror Comella-Dorda et al. (2002).

2.5.1 Planera undersökningen

Comella-Dorda et al. (2002) betonar vikten av att i planeringsstadiet av ett utvärderingsprojekt etablera ett utvärderingsteam där deltagarna har blandade kunskaper. För att bästa resultat ska ges behöver även målet och syftet med utvärderingen identifieras samt hur resultatet från utvärderingen ska mätas. Därtill behöver utvärderingens intressenter identifieras samt att projektet planeras efter vilka resurser som står till förfogande.

2.5.2 Etablera kriterium

För att etablera kriterium skiljs definitionerna krav och kriterium åt samt att tillvägagångsättet för prioritering av kriterium tas upp. Kraven som ställs anger intressenternas förväntningar på produkten. En skillnad mellan kriterium och krav är att kraven generellt har en abstrakt karaktär för att förhindra att möjliga lösningar förkastas på grund av för hårt ställda krav. Kriterium ska vara tydliga och mätbara eftersom dessa syftar till att en jämförelse ska kunna utföras mellan olika mjukvaror. Detta medför även att översättningen från krav till kriterium anses vara problematisk Comella-Dorda et al. (2002).

De två viktigaste egenskaperna för ett bra kriterium är att den önskade förmågan enkelt ska kunna mätas samt att det ska finnas en metod för att mäta förmågan. För prioriteringen är det lämpligt att genomföra genom viktning av krav. Detta kan till exempel genomföras med hjälp av en ostrukturerad viktning vilket uppges vara den vanligaste metoden. Det systemet bygger på att utvärderingspersonerna bestämmer vikterna baserat på deras personliga åsikter Comella-Dorda et al. (2002).

2.5.3 Insamling av data

För att samla in data tar Comella-Dorda et al. (2002) upp ett flertal tekniker i vilka litteraturstudier och egenutformade tekniker ingår. I litteraturstudier ingår sökningar efter produktens prestanda och dessa kan till exempel genomföras på internet eller undersökningar av tidigare utvärderingar. I egenutformade tekniker ingår till exempel att utvärderingsteamet själva tester programmets egenskaper för att samla in data som används för analysen.

2.5.4 Analysera data

Det sista steget i modellen är att analysera data, vilket innefattar sammanställning och analys av data. För sammanställningen av data har All-to-Dollars Technique och Weighted Aggregation identifierats. I den förstnämnda summeras alla fördelar som plus medan nackdelarna resulterar i minus. I den sistnämnda viktas kraven för att viktigare krav ska ge större effekt på resultatet.

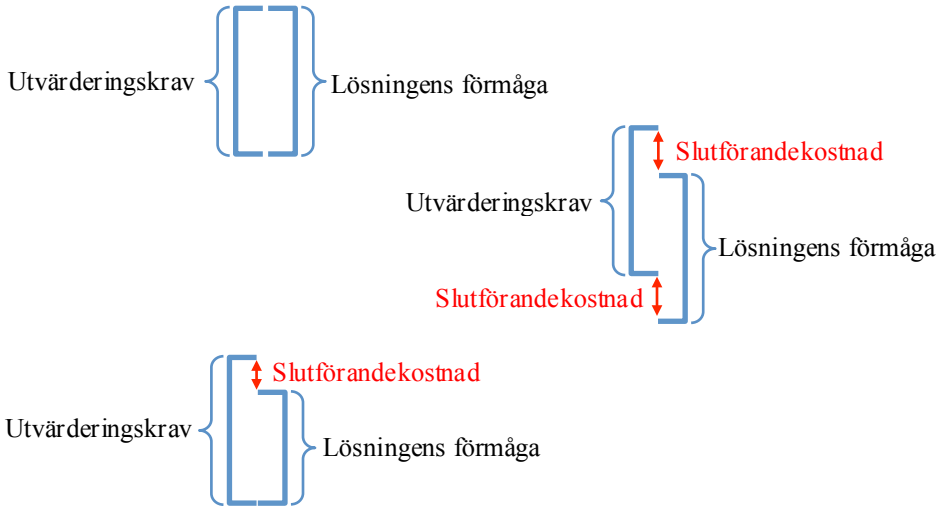
För att analysera data tas känslighetsanalys, gap-analys och analys för fullgörande upp. Känslighetsanalys innefattar att genomföra en analys om vad konsekvenserna blir om variabler som mjukvaruutvärderingen baseras på förändras.

Gap-analyser analyserar differensen mellan kraven som ställts på en mjukvara och mjukvarans faktiska funktionalitet. Ett gap uppstår när mjukvaran inte når upp till kravet som ställts. Figur 16 visar ett exempel på ett gap för kriterium K5 och mjukvara P3 eftersom mjukvaran endast kan användas i Windows-baserade system. Detta utgör ett gap eftersom kravet var att mjukvaran även skulle kunna användas i ett Linux-baserat system.

Produkt / Krav	P1	P2	P3
K1	Stöder ej Java	Komplett lösning	Komplett lösning
K2	Inexakta beräkningar	Endast 2 decimaler	Kan inte räkna
K3	10% < Krävd tillförlitlighet	Ingen tillförlitlighetsdata	Komplett lösning
K4	Komplett lösning	Leverantör utomlands	Leverantör i Canada
K5	Komplett lösning	Kräver Linux	Endast för Windows

Figur 16. Figuren illustrerar olika mjukvarors kravuppfyllnad. Om mjukvaran uppfyller kravet anges fullständig lösning. I andra fall anges orsaken till varför kravet inte är uppfyllt Comella-Dorda et al. (2002).

Fullgörandeanalysen analyserar vad som krävs för att anpassa en mjukvara som ej uppnår samtliga ställda krav se figur 17. Exempel på aspekter som analyseras är vidareutveckling av programmet för krav som ej är uppfyllda, arkitektur, underhållskostnader och affärsmässiga överväganden. Slutligen kräver denna analys att gapen identifieras och att mjukvarans förmåga att uppnå kraven utvärderas se figur 17, samt att strategier för att åtgärda mjukvarans gap identifieras.



Figur 17. Den översta delen av figuren illustrerar fallet då en mjukvara uppnår samtliga krav. De andra anger fallet när kraven skiljer sig från mjukvaran samt visar hur stor del av mjukvaran som behöver åtgärdas och hur krävande det är. Figuren är hämtad från Comella-Dorda et al. (2002).

Med tanke på att beslutsfattare är olika bör en fakta baserad rekommendation. Därför är det viktigt att utvärderingsprocessen dokumenteras noggrant för att beslutsfattaren ska förstå processtegen och följa med i resonemanget. Detta bör leda till en summering med rekommendationer innehållande de olika produkternas kravuppfyllande, svagheter, fördelar och dokumentation av allmän fakta Comella-Dorda et al. (2002).

3 Metod

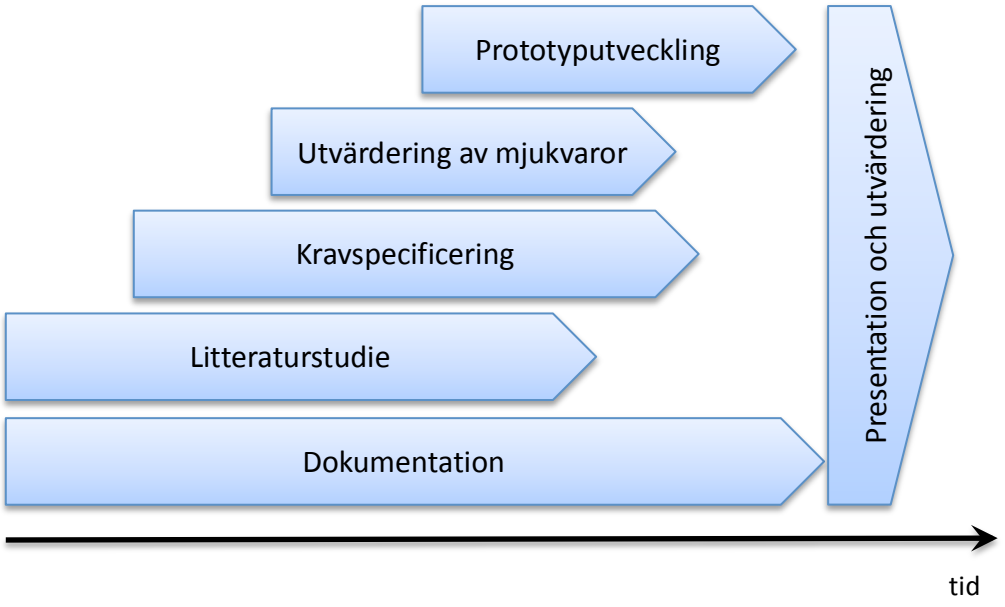
Kapitlet inleds med en övergripande förklaring av arbetsgång och fortsätter därefter med beskrivning och motivering av den valda projektmodellen. Därefter följer metodvalen för de tre delarna i projektet; kravspecifikation, utvärdering av existerande mjukvaror samt mjukvaruutveckling. De tre delarna är i rapporten uppdelade för att öka läsbarhet och förståelse men skedde under projektets gång så gott som parallellt, som kan ses i figur 18. Kapitlet avslutas med en beskrivning över hur informationsinhämtningen gick till i projektet. Projektets syfte är att utföra en förstudie av hur informationsmodellen kan realiseras i form av en demonstrator. I kapitlet förklaras hur projektet har lagts upp för att uppfylla syftet och svara på följande frågor:

Vilka krav bör ställas på en demonstrator för att produktionsförbättringspotentialen ska visualiseras på ett tydligt sätt?

Vilka kommersiellt tillgängliga mjukvaror är möjliga att använda och hur kan en sammansättning av dessa se ut?

Hur kan en potentiell demonstrator utvecklas?

Figur 18 visar visuellt hur projektets arbetsgång har sett ut. Inledningsvis gjordes en litteraturstudie som följdes upp med dokumentation som sedan fortlöpte genom hela projektet. När gruppen var tillräckligt inlästa på aktuellt område påbörjades arbetet med att utforma en kravspecifikation. Först och främst utvecklades en kravspecifikation för utvärdering av mjukvaror för att kunna påbörja en utvärdering av aktuella mjukvaror. Kravspecifikationsarbetet fortsatte sedermera med utformning av kravspecifikationen för utveckling av en mjukvara, när denna kravspecifikation blivit påbörjad inleddes arbetet med att skapa en prototyp. När alla tre projektleverabler färdigställts slutfördes aktuell rapport och projektet avslutades därefter med en presentation.

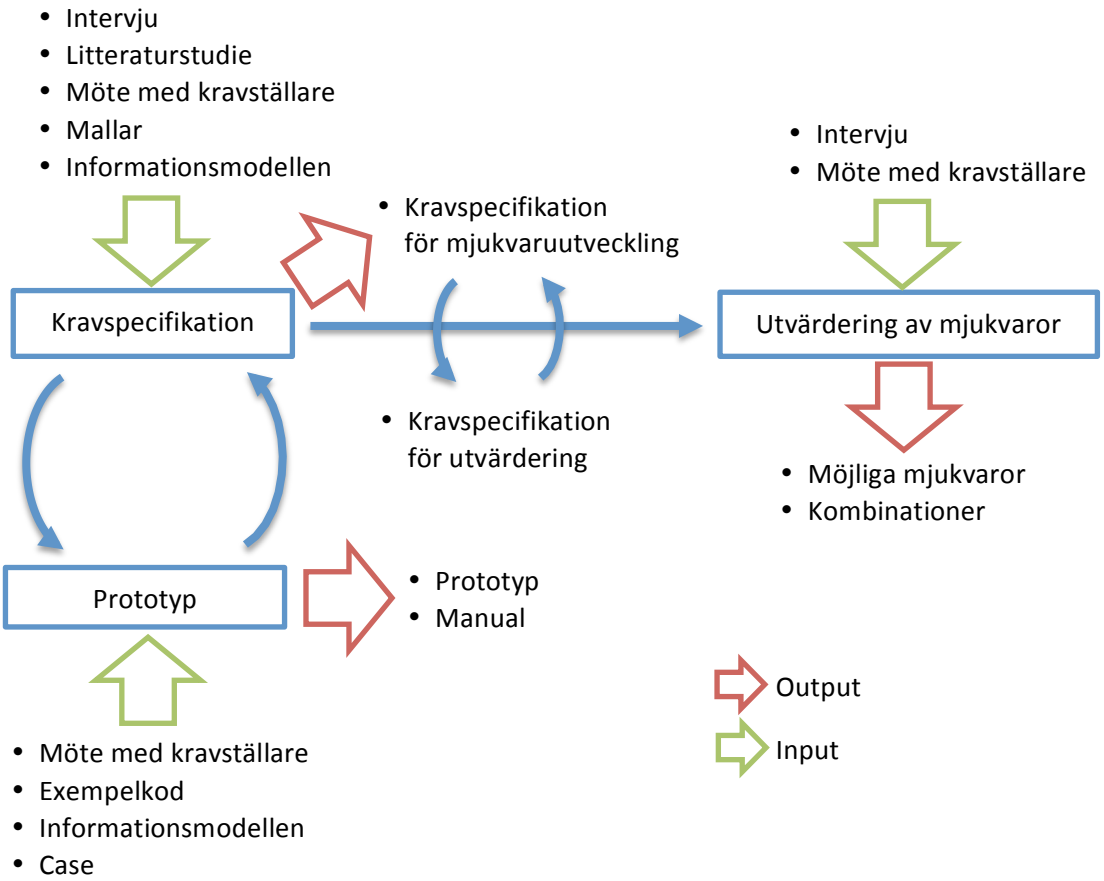


Figur 18. Projektets tidsplan

Den övergripande projektmodellen som valdes är prototyping vilket är en iterativ modell. Den innefattar främst framtagningen av kravspecifikation och utvecklingen av prototypen eftersom

iteration mellan dessa två delar hela tiden utfördes. Från prototypen erhölls ytterligare förståelse för vilka krav som bör ställas på en demonstrator och från kravspecifikationen erhölls förståelse för vilka funktioner som bör implementeras i demonstratorn och som testades i prototypen. Andra möjliga projektmodeller är vattenfallsmodellen och spiralmodellen. Med vattenfallsmodellen hade dock ingen iteration tillåtits varför funktioner och krav hade varit svåra att ta fram och förstå. Spiralmodellen, som är en iterativ modell precis som prototyping, är uppbyggd av varv där varje varv fokuserar på sin del i utvecklingen. Detta hade medfört att de krav och funktioner som tagits fram skulle ha grupperats och tagits fram tillsammans. Detta ansågs inte gynna projektet utan det var mer önskvärt att först ta fram de krav och funktioner som var mest grundläggande för att sedan ta fram de andra mindre vitala funktioner och krav.

Prototypingmodellen valdes eftersom den förespråkar att mjukvaruutvecklingen sker iterativt med utvecklingen av kravspecifikationen. Dessutom sker kontakten frekvent med kunden för att stämma av hur kunden uppfattar prototypen. Det underlättade för både kravställare, utvecklare och utfärdare av kravspecifikationen. Dessutom är prototyping bra för de projekt där kraven som ställs på produkten är oklara och svåra att definiera vilket de i detta projekt var. I figur 19 beskrivs projektets arbetsgång med kopplingar mellan respektive område. Dessutom sker kontakten frekvent med kravställarna Peter Almström och Richard Hedman.



Figur 19 Beskriver arbetssättet kopplat till resultatet

De blå pilarna förtydligar vilka delar av projektet som har påverkat varandra, de gröna pilarna visar vad som är input till respektive projektområden och de röda pilarna visar det resultat som fås från

respektive projektområde. Nedan beskrivs de metoder och tekniker som använts för att ta fram de tre resultaten.

3.1 Informationshämtning

Litteraturstudier, intervjuer och diskussioner har genomförts under hela projektets gång för att erhålla nödvändig information samt kunna väga olika former av källor mot varandra. Initialt riktades informationssökningen och informationsinhämtningen endast mot att förstå området och projektet medan det senare utvecklades till att omfatta all teori som krävdes för att uppfylla projektets syfte.

Intervjuer kan delas upp i ostrukturerade intervjuer och semistrukturerade intervjuer. Vid ostrukturerade intervjuer ställer intervjuaren frågor med grund i olika teman och frågorna är inte bestämda i förväg. I semistrukturerade intervjuer är frågornas följd inte bestämda men däremot är de frågor som skall ställas bestämda i förväg (Bryman and Nilsson, 2002). Alla intervjuer valdes att utföras i form av ostrukturerade intervjuer. Inledningsvis beskrevs och presenterades projektet inför informanten, därefter tilläts informanten begrunda olika alternativ för att sedan ge en beskrivning på dennes synsätt kring olika möjliga tillvägagångsätt.

Kring projektmodeller, kravspecifikationer, mjukvaruutveckling samt olika mjukvaror och dess egenskaper har litteraturstudie genomförts. Litteraturen hittades genom att söka på nyckelord, så som “Software Engineering”, “Requirements Engineering”, “Systems Engineering”, “Requirements Management”, ”Simulation Software” och ”Production Software”.

Information kring mjukvaror samt deras egenskaper och möjligheter har även erhållits, utöver litteratur, genom intervjuer med sakkunniga personer och diskussioner med kravställare.

Från framtagningen av kravspecifikation erhölls två kravspecifikationer, från mjukvaruutvecklingen erhölls en prototyp med tillhörande manual och från utvärdering av existerande mjukvaror erhölls en lista med möjliga kombinationer av mjukvaror.

3.2 Kravspecifikation

Projektet har utvecklat två kravspecifikationer. Den ena kravspecifikationen, som hädanefter kallas ”*Kravspecifikation för utvärdering*”, är inte ett av projektets huvudresultat. Den används som ett verktyg internt i projektet för att utvärdera i vilken utsträckning existerande mjukvaror uppfyller kraven som ställs på demonstratorn. Den är skriven på en högre nivå med fokus på funktionalitet och mindre detaljspecifika krav för att inte komplicera användningen under utvärderingsprocessen. Den andra kravspecifikationen, som hädanefter kallas ”*Kravspecifikation för mjukvaruutveckling*”, är ett av projektets tre resultat och har utformas med målet att vara tillräckligt utförlig för att kunna lämnas över till ett mjukvaruutvecklingsföretag för programmering.

I takt med att litteraturstudierna tog fart breddades perspektivet på hur kravspecifikationer för mjukvaror skall se ut. Efter en ostrukturerad intervju med Richard B. Svensson, universitetslektor vid institutionen för data-och informationsteknik på Chalmers tekniska högskola erhölls input som vägdes mot litteraturen (Lauesen, 2002). Därefter valdes funktionskrav (Lauesen, 2002) som huvudsaklig metod att presentera krav. I ett funktionskrav förklaras objektet i textform med en funktionell beskrivning. Negativa aspekter med metoden är att det är svårt att få med teknisk detaljering med enbart funktionskrav. Som komplement där högre detaljering behövs införs sammanhangsdiagram och uppgiftsbeskrivningar (Lauesen, 2002).

Grunden för kravspecifikationen lades genom arbete med kravställare, att tydligt stolpa de grundkrav som behövdes för att inleda arbetet med demonstratorn och utvärderingen av möjliga mjukvaror. Intressenterna till projektet identifierades som kravställare som också är handledare och examinator, analytiker som utfärdar kravspecifikationen, mjukvaruutvecklare för prototyp och användare som representeras av testpersoner av prototypen. Grundkrav verifieras och resulterade i ”*Kravspecifikation för utvärdering*”. Därefter omformulerades krav till funktionskrav tillsammans med teorins input för vad som bör innefattas av kravspecifikationer (Lauesen, 2002). Som komplement där det ansetts vara nödvändigt har funktionskrav kompletterats med andra metoder enligt litteraturen. Med ökad detaljeringsnivå genom iterering och verifiering med kravställare har målet varit att få fram realistiska krav i rätt detaljnivå för kandidatarbetet. Detta resulterade i ”*Kravspecifikationen för mjukvaruutveckling”*. Slutligen byggdes listan med krav på parallellt imed utveckling av prototypen genom att identifiera nödvändiga krav genom prototyping. Under processen har kravfunktionen aktivt spårats genom att gå fram och tillbaka mellan prototyp, kravspecifikation och kravställare och söka efter konflikter och fel.

Resultatet med funktionskrav kombinerat med övriga metoder tillsammans med strukturen på dokumentet blir kravspecifikationen i helhet mer hands-on jämfört med teoretiska alternativen. Det är också den mest använda metoden (Lauesen, 2002), med krav som skall vara förståeliga för intressenterna av projektet.

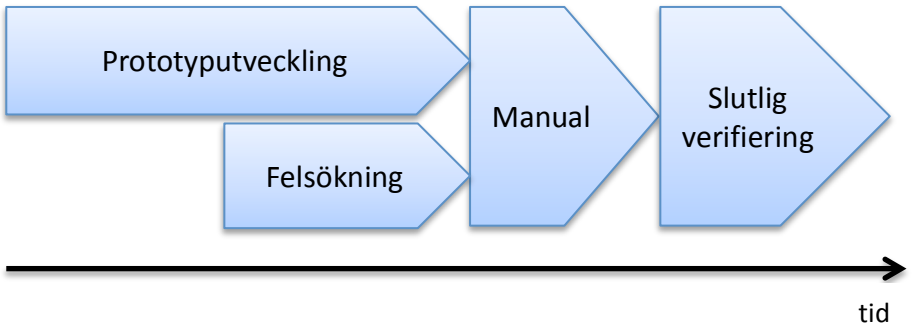
3.3 Prototyp

Ett av projektets syften är att utveckla en prototyp, som visar hur demonstratorn kan se ut. Prototypen möjliggör även underlag till kravspecifikationen och kan påvisa brister. Förhoppningen är därmed att de krav som framställs ska bli så väl mer relevanta som bättre preciserade. Den kan även ge inspiration till den slutgiltiga mjukvaran, och skulle eventuellt kunna vidareutvecklas till en fungerande mjukvara.

För projektet i sin helhet valdes som sagt den iterativa utvecklingsmetoden prototyping utvecklingen. De avgränsningar som följer med prototyping-metoden innebär att fokus från start varit att ta fram en prototyp i vilken krav och önskemål kan testas och utvärderas under projektets gång enligt ett iterativ tillvägagångssätt (se figur 20). Samtidigt är det nödvändigt att ständigt avväga noggrannhet vid utveckling kontra utvecklingshastighet eftersom sämre, men för stunden fungerande lösningar, i dagligt tal kallade fulhack, tenderar att ge stora problem och mer arbete efterhand som utvecklingen fortlöper.

Då prototyping valts som projektets modell valdes ett objektorienterat programmeringsspråk samt en utvecklingsmiljö och därefter modellerades en första version av prototypen. Den fungerade endast som en skiss över ett tänkbart användargränssnitt samt beskrivning av vilken funktionalitet slutprodukten skall ha, men utan någon implementerad funktionalitet. Utifrån denna version utvecklades prototypen sedan successivt, i samråd med kravställare och kravspecifikation, till den slutgiltiga prototypen erhölls.

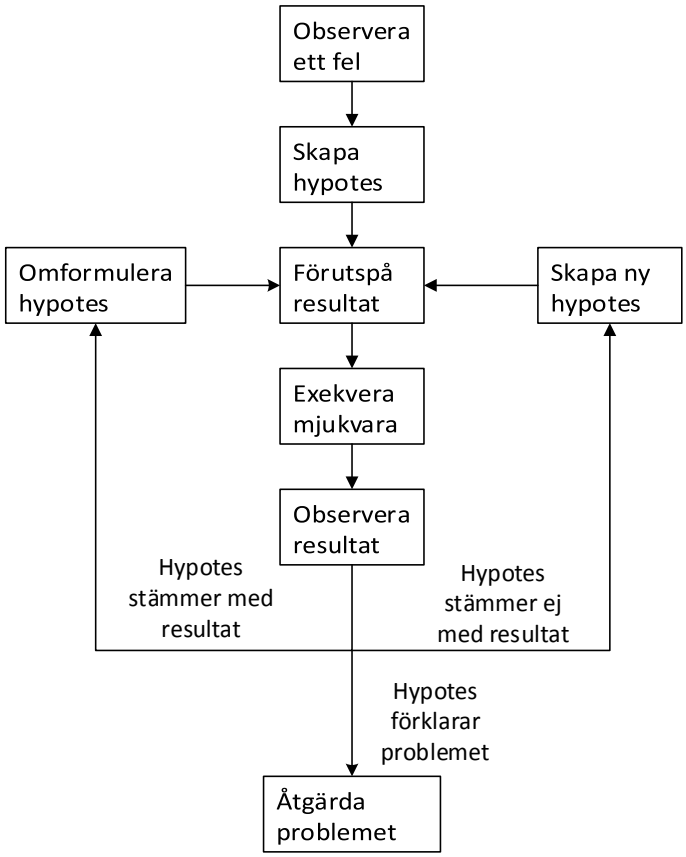
Processen inleddes med att prototypen konstruerades och modellerades och då grundfunktionerna implementerats startade en felsökning. Då prototyputvecklingen och felsökningen avslutades skrevs en manual och med hjälp av denna manual genomfördes en slutgiltig verifiering av prototypen, allt enligt figur 20. Manualen har skapats för att de som läser rapporten skall ha möjlighet att använda prototypen om så önskas.



Figur 20. Beskriver arbetssättet kopplat till resultatet

3.3.1 Verifiering och validering

Verifiering innebär kontroll av prototypens kravuppfyllnad medan validering innebär kontroll av hur väl prototypen beskriver verkligheten. Verifieringen har i detta projekt skett genom felsökning av prototypen som tillsammans med testning av prototypen genomförts parallellt med utvecklingen av prototypen, från den tidpunkt då de grundläggande funktionerna fungerade. Validering har skett genom test av prototypen med manual och case. För felsökningen finns en metod som bygger på en vetenskaplig metod för experiment (Zeller, 2006). Den vetenskapliga metoden kan anpassas till felsökning av mjukvaror och ser då ut som i figur 21. Mjukvaran körs och ett fel upptäcks och observeras, till exempel ett beräkningsfel. Då skapas en hypotes till varför detta fel uppstår. Med hypotesen förutspås mjukvarans nästa resultat och hypotesen testas genom att exekvera mjukvaran och observera resultatet. Om mjukvarans resultat ej uppfyller förutsägelsen skapas en helt ny hypotes. Mjukvaran exekveras och resultatet observeras. Då förutsägelse och resultat stämmer överens formulerar man om hypotesen för att undersöka varför problemet uppstår. Mjukvaran exekveras och resultatet observeras. Iterationen genomförs till dess att hypotesen förklarar problemet till de tidigare och eventuella framtida fel.



Figur 21. Diagram som visar hur felsökningsprocessen gått till. Inspiration hämtad från Zeller (2006)

I detta projekt följde felsökningen efter beräkningsfel metoden ovan. Prototypens funktioner undersöktes och kontrollerades manuellt, men följde metoden ovan så gott det gick. Även designen och estetiken i prototypen undersöktes och värderades men detta följde av naturliga skäl inte metoden beskriven ovan. Då möjliga förbättringsförslag hittades, som inte var beräkningsfel, diskuterades de innan de implementerades eller avfärdades. För varje ny implementering, förbättring och förändring genomfördes systematiska tester av prototypen för att detektera eventuella nya fel, både funktionsfel och beräkningsfel.

Då utvecklingen och felsökningen av prototypen avslutades startade arbetet med att ta fram en manual till prototypen samt ett case att användas till användartester av prototypen. Manualen och caset togs fram för att validera prototypen, vilket innebär att kontrollera så att prototypen beskriver verkligheten på ett tillfredställande sätt. Den metod som användes för att skriva manualen i detta projekt beskrivs av Robinson et al. (2000). Metoden startar med att en planering genomförs där huvuddragen i manualen tas fram vilket i projektet gjordes genom att systematiskt använda prototypen så som en framtida användare kan tänkas göra. Sedan skrivs och redigeras manualen innan den slutliga manualen erhålls. I projektet skrevs instruktionerna tydligt och på ett så enkelt språk som möjlig samt att det finns skärmdumpar av prototypen där speciellt viktiga delar är markerade för att tydliggöra instruktionerna.

För att hjälpa användaren att orientera sig i manualen är det viktigt att ha en tydlig struktur (Robinson et al., 2000) vilket implementerades i prototypens manual genom att ha tydliga rubriker och så informativa bildtexter att bilderna ska kunna förstås utan att läsa brödtexten.

För att kunna genomföra användartester skapades ett case då manualen var färdigskriven. I detta case ingick en beskrivning av en problematisk del i en fiktiv fabrik. Tillsammans med detta case och manualen genomfördes användartester av prototypen för att dels undersöka prototypens och manualens användarvänlighet men även hur väl en verklig situation går att införliva i prototypen. De som testade prototypen var dels kravställare och dels gruppmedlemmar som inte varit delaktiga i utvecklingen av prototypens uppbyggnad. Kommentarer och synpunkter på prototyp, manual och case togs emot, analyserades och implementerades eller avfärdades.

Som ett sista steg i valideringen av prototypen genomfördes ett test som kan likställas med ett riktigt fall ute i produktion. Underlaget som användes till testet kommer från rapporten ”*Saving P&C Inc.*”(Basoukos et al., 2014) som togs fram i kursen Production Ergonomics and Work Design på Chalmers tekniska högskola och omfattar SAM-analyser samt filmer. Med hjälp av detta undersöktes hur väl prototypen är applicerbar på en verklig situation. Samtidigt som testet genomfördes gjordes även en sista utvärdering av prototypens korrekthet, användarvänlighet och utvecklingsmöjligheter genom att ständigt kommentera för- och nackdelar med prototypen då den används på detta omfattande sätt.

3.4 Utvärdering av existerande mjukvaror

Den här delen av metoden syftar till att beskriva tillvägagångssättet för utvärderingen av mjukvaror. Efter att teorin analyserats har metoden som använts för utvärdering av mjukvaror delats in i tre olika steg. Det första steget var att identifiera de mjukvaror som kan vara aktuella att studera, därefter utvärderades dessa mjukvaror efter uppställda krav. Efter utvärderingen gjordes även olika kombinationer av mjukvarorna för att undersöka möjligheten att kombinera olika mjukvaror för ett bättre resultat. De tre stegen beskrivs av figur 22.

Figur 22. Figuren beskriver metodiken för utvärderingen av existerande mjukvaror.

3.4.1 Identifiering

För att identifiera tänkbara mjukvaror som skulle kunna användas för att designa demonstratorn utfördes en informationssökning. De identifierade mjukvarorna delades in i kategorier, vilket senare möjliggjort en analys av varje kategoris funktionalitet för att uppnå projektets önskemål. De kategorier som tagits fram är programmerings-, simulerings-, visualiserings- och produktionsmjukvaror.

En intervju hölls med tekniklektor Per Medbo på institutionen Teknikens ekonomi och organisation för att skapa en övergripande förståelse för de olika typerna av kategorierna. För att förenkla utvärderingen av mjukvaror beslutades att utvärderingen av existerande mjukvaror enbart skulle innehålla mjukvaror där projektgruppen själva hade kunskaper. Senare genomfördes intervjuer på en mer specifik nivå där samtliga kvarvarande mjukvaror inom produktionskategorierna utvärderades för att få en övergripande bild av för- respektive nackdelar med de olika mjukvarorna.

3.4.2 Utvärdering

Nielsen (1994) anger att användartester är en metod för att testa och undersöka en produkt, exempelvis en mjukvara. Processen syftar till att undersöka hur riktiga användare är tänkta att använda mjukvaran. Myers et al. (2011) betonar vikten av att användartesterna planeras och struktureras i förväg för att mjukvarans funktionalitet ska utvärderas mot samtliga funktioner som ska vara uppfyllda. Fokus vid användartestet är mjukvarans funktionalitet och inte hur mjukvaran är uppbyggd.

Användartesterna för att bedöma mjukvarornas egenskaper har därför planerats i förväg där det första användartestet baserades på *”Kravspecifikationen för utvärdering av mjukvaror”*. Analysen syftade till att innehålla personliga övergripande reflektioner samt till att identifiera positiva och negativa aspekter med respektive mjukvara.

Kraven som ställts på demonstratorn omarbetades för att översätta kraven till funktioner. Funktionsuppdelningen använde produktionssystemteorin för att dela upp informationsmodellens funktioner i in- och utdata samt operationerna som krävs för att överföra indata till utdata.

Utvärderingen genomfördes med hjälp av användartester genom att mjukvarorna undersöktes funktion för funktion för att se vilka funktioner som uppfylldes respektive inte.

3.4.3 Kombination av mjukvaror

Olika mjukvarukombinationer som uppfyllde kraven på demonstratorn identifierades med hjälp av en uppdelning enligt gap-analys teori. Eftersom det skulle vara väldigt tidskrävande att analysera samtliga kombinationer valdes 2 olika kombinationer som baserades på gruppens tidigare analys av mjukvarorna. Detta resulterade slutligen i 2 olika förslag på hur kombinationer av mjukvaror kunde användas för att skapa en demonstrator.

4 Kravspecifikationer

Inför projektet var en av frågeställningarna hur krav bör ställas på en demonstrator som realiserar informationsmodellen för att visualisera produktionsförbättringar på ett tydligt sätt. För att underlätta arbetet gjordes två kravspecifikationer. Den första, ”*Kravspecifikation för utvärdering*”, utvecklades för att vara ett hjälpmedel i utvärderingsprocessen. Där listas i korthet de krav som av uppdragsgivare anses vara nödvändiga att uppfylla för att behålla funktionalitet i informationsmodellen. Den andra kravspecifikationen ”*Kravspecifikation för mjukvaruutveckling*” är ett fristående dokument som förklarar grunderna i informationsmodellen med krav formulerade efter så som beskrivet i teorin (Lauesen, 2002)(Lauesen, 2002). Denna kravspecifikation har en hög detaljeringsgrad med hjälp av omskrivningar till funktionskrav och sammanhangsdiagram för att skapa en helhetsbild efter projektets ramar.

4.1 Kravspecifikation för utvärdering

Utdrag från ”*Kravspecifikation för utvärdering”* visas i tabell 1. Kraven är skrivna på en hög nivå och beskriver funktionerna i korthet. De är utformade så att det dels ska vara möjligt att utvärdera redan existerande programvaror i utvärderingen och dels lätt spåra och validera huvudfunktionerna hos prototypen. Hela ”*Kravspecifikation för utvärdering”* finns att se i bilaga A.

Tabell 1. Utdrag från kravspecifikation för utvärdering av mjukvaror.

Funktionalitetskrav	**Kravuppfyllande**
1. Funktionalitet	
1.1 Skall definiera fabriksnivåer och aktiviteter för M-parametern	
1.2 Definiera aktiviteter som subaktiviteter	
1.2.1 Skall kunna ange verklig tid	
1.2.2 Skall kunna definiera ideal tid	
1.2.3 Skall kunna definiera tidsdrivare för aktivitet	
…	
1.4 Skall kunna ange prestationsparameter per resurs, P-parametern [%]	
1.4.1 Möjlighet till prestationsmått per aktivitet	
1.5 Skall ange uttnyttnandegradsparameter för processen, U-parametern [%]	
1.5.1 Skall kunna definiera utnyttjandegrad	
1.5.1.1 Skall kunna välja processnivå	
…	
1.5.2.4 Skall visa förluster: personlig tid, systemförluster, störningar/haverier	
1.6 Skall kunna visa kopplingar av aktiviteter via precedensdiagram	
1.6.1 Annat sätt att visa flödet på kvalificerat vis	
1.7 Skall kunna exportera data till PowerPoint, PDF eller alt. Presentationsform	
1.7.1 Skall kunna exportera grafer	
…	
1.7.8 Precedensdiagram	
1.7.10 Hierarkisk struktur	
2.1 Skall generera tidsekvationer	
2.1.1 Beräkna produktivitetsmått för varje aktivitet, M-Parametern [enh/tid]	
2.1.2 Addera Aktiviteter och subaktiviteter	
2.1.3 Användning av tidsdrivare	
2.2 Beräkna CAP	
2.2.1 Visualisera ideal och verklig kapacitet	
2.3 Möjlighet att spara användardata	
3. Önskvärda utvecklingsmöjligheter	
…	

4.2 Kravspecifikation för mjukvaruutveckling

Från ”*Kravspecifikationen för utvärdering*” bygger ”*Kravspecifikationen för mjukvaruutveckling”* vidare med en högre detaljeringsgrad och är uppbyggt för att kunna läsas som ett fristående dokument. För att ge förståelse för de som står utanför projektet behöver vissa delar, och då främst informationsmodellen, förklaras mer ingående och på ett sätt så att användaren skall förstå sambanden bakom kravet och dess funktionalitet. ”*Kravspecifikation för mjukvaruutveckling*” har delats upp i kapitel för att göras överskådligt och hjälpa läsaren att förstå innehållet. Tabell 2 visar kapitelindelningen som anpassats för att få med beskrivning av informationsmodellen och den önskade slutliga demonstratorn. Kravspecifikationen inleds med att ge en introduktion och förklaring till informationsmodellen, tillhörande definitioner och beskrivning av demonstratorn innan krav ställs. På så sätt får läsaren grepp om vad produkten skall mynna ut till innan dess att krav som ökar förståelse.

Tabell 2. Kapitelindelning i ”*Kravspecifikation för mjukvaruutveckling*” med respektive delar som hjälper läsaren förstå informationsmodellen.

1	Introduktion
.	
2	Definitioner, förkortningar och akronymer
.	
3	Beskrivning
3.1	Avgränsningar
3.2	Produktperspektiv
3.3	Övergripande krav
3.4	Användarkaraktäristik
3.5	Funktionalitet
4	Kravspecifikation
.	
4.1	Funktionalitet och uppbyggnad av modell

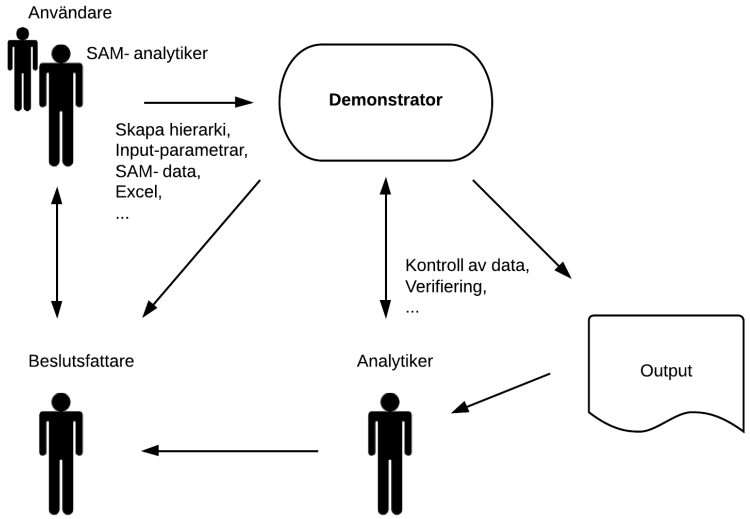
Det är först i kapitel 3.3 som de första kraven ställs, efter det att läsaren har fått en känsla för vilka avgränsningar som gjorts och att produktperspektivet har förklarats. Då ges övergripande krav som är frånkopplade informationsmodellen men viktiga för mjukvaruutvecklingen och för stabilitet i mjukvaran. Exempel på vad de beskriver är noggrannhet i antal decimaler vid beräkning, användarvänlighet, säkerhetsuppdatering och robusthet i mjukvaran. Det en balansgång för att inte överspecificera mängden allmänna krav för mjukvaruutvecklingen. Det gäller att identifiera viktiga grundfunktioner som underlättar användning för slutanvändaren av mjukvaran. Det skall vara standardfunktioner så som autosparande och starttider för mjukvaran. Krav sorteras efter kapitelindelning och rubriceras med en unik kravkod. Det förenklar arbetet med kravsökning och hantering. Se tabell X för de övergripande kraven.

Tabell 3. Övergripande krav från *"Kravspecifikation för mjukvaruutveckling"*. Hjälpmedel till utvecklare för att mjukvaran skall vara användbar enligt användarvänlighet för mjukvaran.

K3.1.1	Tider, både i sekunder och faktorer begränsas till maximalt två decimaler i utskrift, medan alla decimaler används vid beräkning.
K3.1.2	Användare skall kunna ändra i befintlig modell, lägga till, ta bort eller flytta komponenter. För både ovanliggande och underliggande nivåer i hierarki- uppbyggnad.
K3.1.3	Det skall gå att köras på operativsystemet Windows XP framåt. Den skall även vara gå att använda på alla plattformar från Windows XP och framåt.
K3.1.4	En sparad fil från en användare skall gå att öppna och använda hos en annan användare på en annan dator som uppfyller systemkraven.
K3.1.5	Skall automatiskt spara en säkerhetskopia av förändringar var 5:e minut.
K3.1.6	Vid avstängning skall användaren tillfrågas att spara modellen innan avstängning.
K3.1.7	Skall vid uppstart tillfråga användaren att öppna befintlig modell eller skapa ny.
K3.1.8	Vid uppstart skall det inte ta mer än fem sekunder till det att användaren får välja hur den skall arbeta enligt krav **K3.3.6.**
K3.1.9	Skall tillåta flera importeringar av SAM- ark till samma modell.
K3.1.10	Mjukvaran skall vara robust för inmatning av data och skall ha fastslagna rutiner vid händelse av felaktig syntax.
K3.1.11	Vid inmatning av olika parametrar skall det bara accepteras korrekt syntax definierat enligt metoden.
K3.1.12	Mjukvaran skall följa metoder och använda sig av termer och förkortningar från informationsmodellen (Hedman 2013).

I kapitel 3.4 beskrivs användarkaraktäristiken. Att beskriva den i kravspecifikationen är viktigt för att ge förståelse för vem det är som skall använda mjukvaran och hur den används. Där förklaras användningen med text tillsammans med sammanhangsdiagram som förbereder läsaren för vad funktionaliteten skall kopplas till och vad som skall hända i kulisserna efter att mjukvaran fått sin input. Användarna av mjukvaran beskrivs i kravspecifikationen som:

"*Huvudanvändaren av mjukvaran blir gruppen själva likaså intressenter som handledare och examinator. Det förväntas att användaren är insatt i användning av informationsmodellen. Det skall gå att visa funktionerna och användandet av metoderna i mjukvaran för nybörjare och de skall få en inblick i hur modellens data länkas samman och beskriva beräkningarna i form av tidsekvationer med fler. Det skall finnas möjlighet att anpassa output från demonstratorn efter behov att redovisa för olika beslutsfattare se figur 23*."



Figur 23 - Illustration av samband för informationskanaler mellan demonstrator och användare

Här visas användarna av mjukvaran tillsammans med sammanhangsdigram för att visa hur demonstratorn skall användas. Detta länkas samman med krav på format på indata och utdata, beskrivningar för funktioner och funktioner som är önskvärda. I kapitel 3.5 förklaras funktionalitet som innefattar förklaring av transformationen från indata till utdata och mer ingående användningen av tidsekvationer. Det beskrivs hur tidsekvationerna byggs upp från element till subaktivitet till aktivitet.

I kapitel 4 börjar kravställningen där funktionskrav används som sätt att presentera kraven. Nedan följer exempel på hur några av funktionskraven i kravspecifikationen ut:

K.4.1.36 - *Skall kunna skapa tillverkningsprocesser genom att välja ingående aktiviteter i rätt ordning. Till varje aktivitet skall en resurs med tillhörande egenskaper kopplas*.

K.4.1.37 - *Skall kunna spara modellen av produktionssystemet enligt ISO 15531 MANDATE*.

En del av kraven i kravspecifikationen har utvecklats från den tidigare ”*Kravspecifikationen för utvärdering”.* I vissa fall räcker det med att översätta ett krav från den tidigare kravspecifikationen till ett krav i ”*Kravspecifikationen för mjukvaruutveckling”*, men ofta resulterar det i att krav behöver förtydligas. I dessa fall resulterar det i att ett krav expanderas till flera för att få en mer utförlig beskrivning. Det visas i nedanstående exempel där första kravet är hämtat från ”*Kravspecifikation för utvärdering*” och det andra kravet är det första kravets motsvarighet i ”*Kravspecifikation för mjukvaruutveckling*”:

1.2.2 *Skall kunna definiera ideal tid.*

K.4.1.3 - *Skall kunna ange data för en subaktivitet eller aktivitet så som: verklig tid, ideal tid, typ, direkt eller indirekt aktivitet.*

I andra fall där krav behöver utvecklas, så som när M-parametern beskrivs, resulterar ett krav i ”*Kravspecifikation för utvärdering*” till flera krav i ”*Kravspecifikation för mjukvaruutveckling*”. Just M-parametern har flera krav i den senare kravspecifikationen eftersom det är en grundsten i

informationsmodellen. Det behövs krav för inmatning, transformering och avläsning. Nedan ses kravet för inmatning av M-parameter från ”*Kravspecifikation för utvärdering*”:

1.1 Skall definiera fabriksnivåer och aktiviteter för M-parametern.

Kravet översätts till flera krav i ”*Kravspecifikation för mjukvaruutveckling*” varav några kan ses nedan:

K.4.1.1	Skall låta användaren bygga systemlayout där användaren skapar fabriksnivåer och aktiviteter. Användaren ska själv välja hur omfattande modellen skall bli och vilka systemgränser det är som gäller. Det ska vara hierarkisk uppbyggnad där det tydligt ska framgå vilken/vilka aktiviteter och subaktiviteter som ingår i de olika nivåerna. Modellen ska synas i takt med att det byggs ut och efteråt för redigering och felsökning. Se ”Bilaga 1 – Process: skapa fabrikshierarki” för beskrivning med hjälp av algoritm.
Syfte:	Att användaren skall själv bygga systemet som skall användas i programmet
Utlösare:	Öppnar programmet och väljer ”Skapa ny”
Förutsättning:	Att användaren har eller skall utföra en SAM- och frekvensanalys över ett ex. tillverkningssystem (eller uppskattar tider och förstår innebörden av det).
Frekvens:	Varje gång ny analys över system skall göras, nytt system skapas oftast en gång per subsystem/process.
Handling:	Skapa anläggning. Lägga till fabrik. Lägga till subsystem. Lägga till arbetsstation. Lägga till aktivitet. Lägga till subaktivitet. Lägga till standard- element.
Varianter:	1b: Ladda befintligt system. 5b: Ange ideal tid istället för att använda subaktiviteter. 6b: Ange ideal tid istället för att använda element 7b: Definiera element som inte är med i mjukvaran.

Genom prototyping uppmärksammades det att hierarkins storlek efter inmatning av M-parameter snabbt blir ohanterlig. Därför arbetades följande krav fram:

K.4.1.12 - *Skall ha möjlighet att välja vad som skall visas och inte i systemets olika nivåer, kunna välja att gömma subaktiviteter i ett fall och i ett annat välja att visa alla nivåer ned till elementnivå.*

För att ytterligare specificera hur användaren ska definiera aktiviteterna tillkom kravet:

K.4.1.2 - *Vid uppbyggnad av modellen skall det finnas en undergrupp till aktivitet som kallas subaktivitet. Flera subaktiviteter bildar aktiviteter. Varje subaktivitet ska bestå av SAM- element eller en förutbestämd tid som ska användas för beräkningar*.

Caset tydliggjorde behovet av att kunna ange den ideala tiden manuellt för att slippa det extraarbete som en digitalisering av en befintlig analog SAM-studie innebär. Det resulterade i kravet:

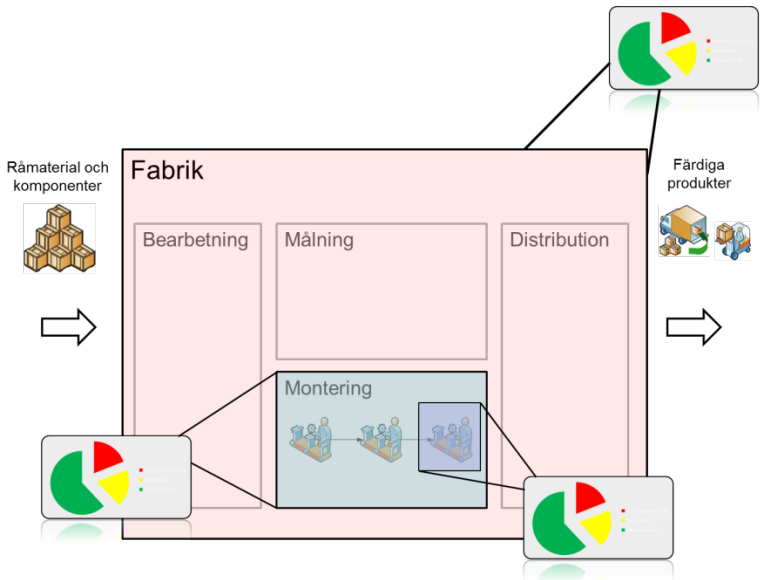
K.4.1.5 - *Skall kunna definiera ideal tid utifrån en SAM-analys. När SAM-analys inte är möjlig att genomföra skall en ideal tid för en subaktivitet eller aktivitet kunna definieras manuellt.*

Nedan visas ett exempel på krav som beskriver exportering och prioritering av nyckeltal som utvecklats från kravet ***1.7.1*** i *"Kravspecifikation för utvärdering"*. Här visas hur utförligheten i kravet ges av en beskrivning i vad som önskas och hur det skall göras genom att ha utvecklat detaljnivån i det tidigare kravet. Från *"Kravspecifikation för mjukvaruutveckling"* återfinns följande krav som motsvarar ***1.7.1*** i *"Kravspecifikation för utvärdering":*

K.4.1.25 - *Skall kunna exportera data från programmet till dokument läsbara för både Windows och Mac- användare. Skall exportera filer i format så som PDF, .docx, .jpeg och .ppt m.fl. Vid val av exportering skall användaren tillfrågas vilken data som skall exporteras och till vilken typ av dokument.*

K.4.1.26 - *Skall kunna exportera grafer på nyckeltal så som CAP_I, CAP_R och produktionsparametrar per process och per produktionsgrupp. Skall ha möjlighet att exportera CAP_I och CAP_R i de fabriksnivåer som ligger över aktivitet i fabrikshierarkin. Skall kunna få ut $CAP_{I/R}$ för system se figur 24 för prioritering.*

K.4.1.27 - *Skall kunna exportera U-parametern som pajdiagram och siffervärde för arbetsstation, subsystem och fabrik.*



Figur 24. Hämtad från Hedman (2013). Beskriver hur parametrar kan hämtas på olika nivåer i produktionssystemet och dess prioritering när det finns flera parametrar att hämta. Bilden visar att enskild monteringsstation prioriteras under monteringsavdelning.

Mängden krav har anpassats efter informationsmodellen utan att ställa för mycket krav som kan ses som självklara och överflödiga. Resultatet med dels en kravspecifikation för utvärdering och dels för mjukvaruutveckling har gjort att projektet har kunnat fortsätta arbeta efter projektmodellen

prototyping för att iterativt förfina kravspecifikationen, vidareutveckla prototyp och att utvärdera existerande mjukvaror..

5 Resultat från utvärdering av existerande mjukvaror

Det här kapitlet presenterar vilka krav på demonstratorn som existerande mjukvaror uppfyller samt hur mjukvarorna kan kombineras för att utveckla demonstratorn. Resultatet från utvärderingsprocessen delas in i nedanstående delar enligt figur 25:

Figur 25. Figuren illustrerar metodiken som använts för utvärderingen av existerande mjukvaror.

5.1 Resultat från identifiering av mjukvaror

I den här delen presenteras mjukvarorna som identifierats som intressanta för projektets syfte, vilka även har delats in i kategorier. Delen innefattar även resultatet från intervjun som genomfördes för att få en övergripande bild av för- respektive nackdelar med de olika mjukvarukategorierna.

De mjukvaror som har identifierats och anses vara aktuella att använda i utvärderingen är följande:

- Arena – demonstrera, förutse och mät för effektiv optimering av tillverkningssystem
- AviX – videobaserad mjukvara för att stödja produktionstekniskt arbete
- Casat NX – beredning och balansering av tillverkningsprocesser
- Microsoft Excel – formatera, ordna och visualisera data
- FlexSim – analysera produktionssystem i 3D grafer
- Matlab – numerisk beräkning, visualisering och programmering
- Microsoft Visio – skapa och dela diagram
- Simul8 – simulering och analys av processer och produktionsflöden

5.1.1 Kategoriindelning

Mjukvarorna delades in i kategorierna simulerings-, programmering-, och produktions-flödesmjukvaror.

5.1.1.1 Simuleringsmjukvaror

Vid användning av flödessimuleringsmjukvaror mäts cykeltiden upprepade gånger för att ta fram en teoretisk statistisk fördelningsfunktion över cykeltidsvariationen, om detta är möjligt. Därefter är det nödvändigt att bestämma en matematisk sannolikhetsmodell för att kunna använda simuleringsmjukvaran (Ross, 2012). Matematiska modeller för om maskiner eller andra definierade enheter går sönder kan även utföras för att simuleringen skall vara så exakt som möjligt.

Simuleringsmjukvaror används främst för att utifrån verkliga komplexa system kunna ta fram information om systemet och utifrån detta kunna fatta ett beslut. Därför skapas en simulationsmodell för att systemet ska kunna studeras vidare och ett rationellt beslut ska kunna fattas (Giaglis, 2001). Exempelvis skulle detta kunna innebära att en avdelning i en produktionsanläggning analyseras station för station för att identifiera cykeltider per aktivitet samt

hur dessa varierar för att till exempel kunna bestämma flaskhalsar, buffertstorlekar och avdelningens output. Utvalda program inom aktuell kategori är följande:

- Arena
- FlexSim
- Simul8

5.1.1.2 Produktionsmjukvaror

I det här projektet definierades produktionsmjukvaror som mjukvaror som är specifikt anpassade för att användas i produktionsmiljöer och är tänkta att fylla en funktion inom området. Dessa mjukvaror innehåller ofta någon form av metodstudieverktyg med funktioner som bör underlätta för önskemålen. Utvalda program inom aktuell kategori är följande:

- Avix
- Casat NX

5.1.1.3 Programmeringsmjukvaror

Denna kategori med mjukvaror kräver mer kunskaper om mjukvaruutveckling än övriga mjukvaror. Det är dock möjligt att utnyttja dessa mjukvaror i ett bredare perspektiv då dessa är tillverkade för att hantera data. Hur datahanteringen utförs kan utvecklas fritt vilket gör att mjukvarorna bör vara fullt möjliga att använda i projektets syfte. De kommer dock krävas mycket arbete att bygga upp modeller och mallar inuti dessa mjukvaror. Utvalda program inom aktuell kategori är följande:

- Microsoft Excel
- Matlab

För att undersöka vilka krav som Microsoft Excel förväntades att uppnå utfördes därför noggrannare studier av programmet. Excel består av olika ark där data kan läggas in och eftersom informationsmodellens indata kan variera beslutades att modulen Visual Basic (VBA) skulle undersökas. Detta eftersom VBA-script gör det möjligt att automatisera hur företagets hierarkiska struktur läggs in, utföra simulationer och få ut resultat. (Sundkvist et al. 2012).

5.1.1.4 Visualisering/flödesmjukvaror

Eftersom Microsoft Visio skiljde sig skarpt från övriga mjukvaror kategoriserades detta som en visualiserings/flödesmjukvara. Mjukvaran är som kategoriseringen antyder specialiserat mot visualisering av data och är lämpligt att skapa till exempel precedens- och flödesdiagram med. Utvalda program inom aktuell kategori är följande:

- Microsoft Visio

5.1.2 Intervjuresultat

I intervjun med Per Medbo framgick det att mjukvaror inom simuleringskategorin inte är optimala att använda i projekt med aktuell typ av frågeställning och syfte. Simuleringsmjukvaror syftar främst till att ge ett beslutsunderlag om var buffertar ska placeras, buffertstorlekar, flaskhalsar och output för given sekvens. Informationsmodellen fokuserar istället på att mäta metodtiden (läs framtagning av faktisk och ideal cykeltid) samt hur störningar och resursernas prestation påverkar produktionen för att därefter jämföra den ideala med den optimala kapaciteten. Således krävs verktyg där ideala tider

kan tas fram och då simuleringsmjukvaror bygger på att dessa tider redan är framtagna klarar simuleringsmjukvaror ej den typen av krav. Vidare finns det ingen funktion i simuleringsmjukvaror som möjliggör framtagning av CAP, tidsdrivare och frekvensstudieresultatet.

I likhet med teorin om simuleringsmjukvaror konstaterade Per Medbo att det inte är lämpligt att bygga upp ett befintligt system i en simuleringsmjukvara och därefter använda simuleringsmjukvaran för att ta fram ledtider. Tvärtom bygger simuleringsmjukvaran på att all data redan är insamlad. I simuleringsmodeller är det även önskvärt att utnyttja fördelningsfunktioner. Detta då inga monteringstider sker exakt lika snabbt utan det finns en viss avvikelse som inte går att komma från. Den modell som används i detta arbete tar inte hänsyn till tidsvariationer i utförandet av monteringen Alla tider anses vara absoluta och ske under samma tid vid varje upprepning och dessa tider kommer istället resultera i en högre mängd störningar.

Till följd av resultatet från intervjun med Per Medbo, och den litteraturstudie som utförts parallellt, beslutades att följande mjukvaror ej skulle analyseras vidare:

- Arena
- FlexSim
- Simul8

5.2 Resultat från utvärdering av mjukvaror

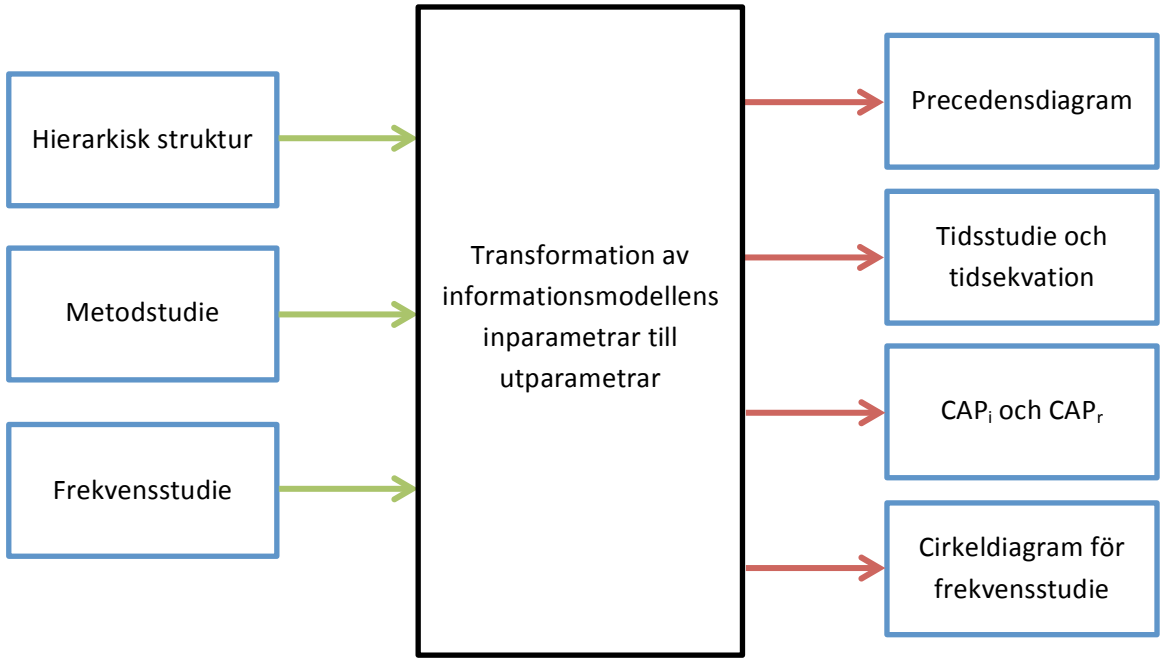
Resultatet från utvärderingen innehåller en generell beskrivning av för- och nackdelar med olika mjukvaror, se tabell 4. Den här delen innehåller även resultatet från den egenutformade systemmodellen av informationsmodellen. Denna resulterade i en förklaring av vilka funktioner som demonstratorn ska ha uppfyllt och resultatet från vilka funktioner som respektive mjukvara uppfyllt.

Tabell 4. Nedan visas en tabell som innehåller en generell kommentar, för- och nackdelar från utvärderingen av nedanstående mjukvaror.

Mjukvara	Mjukvarukategori	Kommentar	Fördelar	Nackdelar
AviX	Produktion	Eftersom en grund finns för att utföra SAM-studier men mjukvaran själv inte utnyttjar detta för uträkningar av ideal tid fallerar mjukvaran i stort sett på samtliga punkter.	Specialutformat för tidsstudier	Uträkning av ideal tid, Prestationsbedömning och CAP blir fel.
Casat NX	Produktion	Anses vara en väl fungerande mjukvara som är relativt intuitivt uppbyggt. Dock avsaknas kopplingen till hur individer arbetar och möjlighet att mäta dessa	Relativt intuitivt Det som saknas bör enkelt kunna byggas in	Avsaknad av individmätning Möjligheter till importering och exportering är relativt låga
Microsoft Excel	Data/Programmering	Genom Excel är det möjligt att på ett flexibelt sätt hantera data och utforma dokument och mallar för att behandla data efter önskemål.	Uppfyller i stort sett samtliga krav	För utförandet av metodstudie krävs det att ideal tid redan identifierats Kräver att ett VBA-script skapas vilket tar tid
Matlab	Data/Programmering	Utvecklat för numerisk beräkning, visualisering och programmering, känt globalt och används på flertalet universitet och högskolor.	Stor potential till personlig anpassning Möjlighet att vidareutveckla på egen hand	Komplicerat att strukturera informationsmodellen
Microsoft Visio	Visualisering/Flödesprogram	Mjukvaran är tänkt att användas för att skapa och dela diagram och processkartor på ett sätt som gör komplex information enklare att förstå.	Goda visualiserings-möjligheter	Möjligheten att mata in data och uppnå kandidatarbetets syfte finns ej

5.2.1 Systemmodell av informationsmodellen

I systemmodellen har kraven som ställts på demonstratorn översatts till funktioner. Detta resulterade i att indataparametrarna till systemet definierades och systemmodellen illustrerar de funktioner som krävs av demonstratorn för att transformera indata till utdata. Se figur 26 för vilka in- och utdataparmetrar som demonstratorn behöver stödja.



Figur 26. Transformationsmodell anpassad efter krav som ställs på informationsmodellen.

5.2.2 Parameterkoppling

Koppling mellan vilken indata som krävs för respektive utdataparameter beskrivs i aktuellt kapitel, för att förtydliga innebörden av figur 26.

Precedensdiagram

Precedensdiagrammet utgår ifrån vilken produkt som ska produceras samt vilka aktiviteter som krävs för att få fram produkten. Precedensdiagrammet ska förklara i vilken ordning de olika aktiviteterna, underaktiviteterna eller elementen ska genomföras. Därför är det viktigt vid inläggningen av data att det tydligt framgår i vilken ordning de olika aktiviteterna utförs för att mjukvaran ska kunna generera precedensdiagrammet.

Tidsdrivare och tidsekvationer

På samma sätt som det vid precedensdiagrammet krävs ett standardiserat sätt för inläggningen av data kommer detta behövas för att få fram tidsdrivare och tidsekvationer. För att få fram dessa kommer det vara nödvändigt att det i mjukvaran ska kunna gå att lägga in hur många aktiviteter en genomförs för en produkt. Därutöver ska tidsåtgången, standardelementtiden eller bådadera behöva läggas in för att få fram tidsekvationen.

Kapacitet

För att få fram kapacitet kommer ekvationerna nedan att användas:

$$Ideal\ kapacitet = CAP_I = M * P_{=1} * U_{=1}$$

$$Verklig\ kapacitet = CAP_R = M * P * U$$

Dessa ekvationer används för att räkna ut kapacitet på fabriks-, subsystems-, arbetsstations-, tillverkningsprocess- och aktivitetsnivå. För att räkna ut kapaciteten krävs det därför att metod-, prestations- och utnyttjandegradsparametrarna länkas samman. Detta kräver en strukturerad metod för hur indata ska läggas in i mjukvaran.

Cirkeldiagram för frekvensstudien

Beroende på vilka nivåer som frekvensstudien lagts in på kommer frekvensstudie cirkeldiagrammen anpassas på dessa nivåer. Dessa fås fram genom att ekvationen nedan ska stämma:

$$100\% - (U_N + U_S + U_D) = U$$

$$0 \leq U_N, U_S, U_D, U \leq 100\%$$

5.2.3 Mjukvarornas måluppfyllnad

En undersökning genomfördes för att kunna identifiera om det finns någon typ av mjukvarukategori som är att föredra för utvecklingen av demonstratorn. Nedan presenteras resultatet från utvärderingen i tabell 5 för mjukvarorna där de olika kategorierna diskuteras separat för att sedan jämföras mot varandra.

Tabell 5 Resultatet visar vilka funktioner som respektive mjukvara uppfyller

Funktioner	Avix	Excel	Visio	Casat	Matlab
Input					
Hierarkisk Struktur	Y	Y	Y	Y	Y
Metodstudie	N	N	N	N	N
Frekvensstudie	N	Y	N	N	Y
Transformationer					
Precedensdiagram	N	Y	Y	Y	N
Tidsekvationer	Y	Y	N	Y	Y
CAP	N	Y	N	N	Y
Cirkeldiagram frekvensstudie	Y	Y	N	N	Y
Antal	3	6	2	3	5

Programmeringsmjukvaror - Denna mjukvarukategori är fullt möjlig att använda, den har dock flera nackdelar. Vid användande av de två mjukvaror som valts ut i denna kategori krävs att mjukvaran utvecklas. Detta medför att det krävs att mjukvaran är egenutvecklad, vilket gör att det krävs mer tid att utforma demonstratorn i denna mjukvarukategori.

- **Excel –** Det är mjukvaran som uppfyllt flest krav, dock har ett fåtal nackdelar identifierats. Den främsta nackdelen med Excel framför AviX är att AviX ger ett betydligt mer visuellt fulländat uttryck. Ytterligare nackdel med Excel är att dess användarvänlighet försämras med antalet anläggningar som analyserats eftersom en större datamängd och ett större antal avdelningar bidrar till mer scrollande och ett större antal flikar. Fördelen med Excel är främst att det med modulen VBA går att utveckla en mjukvara som i hög utsträckning kan användas

för att uppfylla de funktioner som ställs på prototypen. Som tabell 5 illustrerar är den enda funktionen som ej uppfylls av Excel att metodstudie ska kunna läggas in som indata till mjukvaran vilket ej är möjligt på grund av att Excel ej stödjer en SAM-analys.

- **Matlab –** Mjukvaran är inte ämnad för den typ av uppgifter som krävs av demonstratorn. Matlab anses problematisk att använda främst eftersom visualiseringsmöjligheterna är begränsade. Detta illustreras även av att mjukvaran ej uppfyller kraven för precedensdiagram. Kriteriet för att klara av metodstudier var att en metodstudie skulle kunna genomföras direkt i mjukvaran. Eftersom det inte bedömdes möjligt att spela upp en film i Matlab klarar inte mjukvaran metodstudiekravet.

Produktionsmjukvaror - Samtliga produktionsmjukvaror har varit utmärkta att använda till det syfte som dem har utvecklats för. Dock fyller ingen av mjukvarorna det syfte som önskas i detta projekt. Det är möjligt att arbeta med produktionsmjukvarorna för att realisera vissa delar av informationsmodellen Det är möjligt att arbeta med produktionsmjukvarorna i vissa delar av processen, men det är inte möjligt att rakt igenom använda samma mjukvaror för att visualisera produktionsförbättringarna på det sätt som önskas.

- **AviX -** Som tidigare nämnts bedöms AviX vara en kompetent mjukvara att utföra SAM-analyser i. AviX uppfyller dock ej kraven på att frekvensstudien ska kunna delas in i kategorierna som informationsmodellen kräver. Eftersom AviX inte klarar av att dela in elementen efter SAM-standard kommer inte mjukvaran att kunna användas för att räkna ut ideal tid, vilket medför att CAP_I ej kan räknas ut.
- **Casat -** På samma sätt som AviX är uppbyggt med fel definitioner av data äasat också uppbyggt på ett annorlunda sätt än informationsmodellen. Som en konsekvens av detta genomförs metod- och frekvensstudien på ett annat sätt varför Casat fallerar på de krav som ställs på dessa områden. Mjukvaran är därför olämplig att använda för projektets syfte.

Visualiserings/Flödesmjukvaror – Den här kategorin är specialiserad mot visualisering av data och är lämplig för att skapa till exempel precedens- och flödesdiagram med.

- **Microsoft Visio** - Möjligheten att använda denna mjukvara enligt önskemål finns ej. För att skapa precedensdiagram är Microsoft Visio den mjukvara som är lämpligast, men att hantera data i mjukvaran går ej på det sätt som önskas. Därför är denna mjukvarukategori inte bra att använda.

5.3 Kombination av mjukvaror

De mjukvaror som valdes att vidare analyseras genom att kombineras med varandra var AviX, Excel och Visio. Med dessa mjukvaror skapades följande kombinationer: AviX+Excel och Visio+AviX+Excel.

5.3.1 AviX + Excel

I den här delen presenteras ett egenutvecklat konceptuellt förslag på hur Excel och AviX skulle kunna kombineras för att nå samtliga krav i *"Kravspecifikationen för utvärderings"*. Genom att kombinera mjukvarorna på detta sätt skulle Excels brist med att utföra metodstudien lösas eftersom en metodstudie först skulle genomföras i AviX. Därefter skulle ett Excel-ark exporteras ifrån AviX, vilket illustreras av figur 27.

		Reg.nr
Object	Date	DWG No.
Operation	Issued by	Page of

Method description	GET					PUT							USE					RETURN PUT								Summing up Factors		
	Step	GS			Add. for Handful	Weight >5 kg	Step	PD			Add. for Precision	Apply Force	No.of strokes, grips etc..	No.of places	Time of stroke,prip etc		Apply Force	Weight > 5 kg	Step	PD			Add. for Precision	Apply Force	Bend+Arise			
	S	80	45	10	-H	AW	S	80	45	10	-P	AF					AF	AW	S	80	45	10	-P	AF	B			
Underaktivitet	3	5	4	2	6	2	3	5	4	2	3	3	f	n	t	=	3	2	3	5	4	2	3	3	12	F	f	Total
Montera skruv 111	2				1			4																		7		
Montera skruv 1111	2				1			4																		7		
Montera skruv 1	2				1			4																		7		
Montera skruv 11	2				1			4																		7		
Montera skruv 222			4							6		1	1													12		
Montera skruv 333			3				3			3					4											13		
Montera skruv 2222			4							6		1	1													12		
Montera skruv 3333			3				3			3					4											13		
Montera skruv 2			4							6		1	1													12		
Montera skruv 3			3				3			3					4											13		
Montera skruv 22			4							6		1	1													12		
Montera skruv 33			3				3			3					4											13		
Total	8	0	28	0	4	0	12	16	0	36	0	4	4	0	16	0	0	0	0	0	0	0	0	0	0	128	0	0

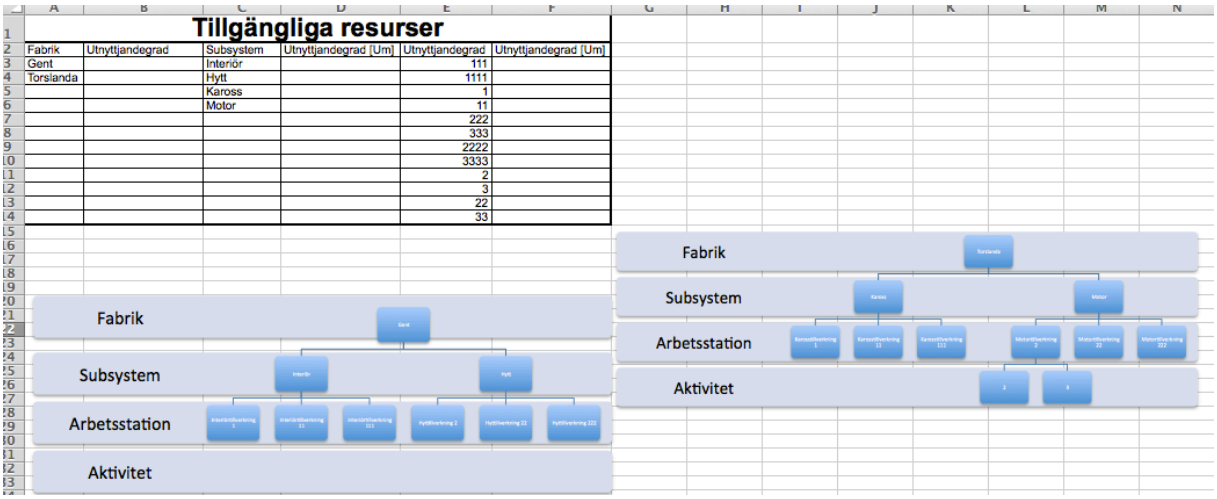
Figur 27. Figuren föreställer ett Excel-ark där en SAM-studie utförts.

I Excel-arket ovan skulle en ruta skapas där användaren skulle behöva lägga in den faktiska tiden. Excel-arken behöver dessutom sparas i en gemensam mapp där de behöver sparas så att fabriken, subsystemet, arbetsstationen, tillverkningsprocessen och aktiviteten kan spåras. Detta behövs för att det egenutvecklade VBA-scriptet ska kunna koppla subaktiviteterna till anläggningens samtliga hierarkiska strukturnivåer och till tillverkningsprocessen. Detta för att ideal tid, hierarkisk struktur, tidsdrivare och tidsekvationer på de hierarkiska strukturnivåerna samt tillverkningsprocess ska kunna identifieras. Ovanstående skulle resultera i ett Excel-ark där kolonnerna arbetsstation, subsystem, fabrik, tillverkningsprocess och aktivitet skulle adderas. Detta illustreras av figur 28.

Object					
Operation					
Method description					
Underaktivitet	**Tillverkningsprocess**	**Aktivitet**	**Fabrik**	**Subsystem**	**Arbetsstation**
Montera skruv 111	*Bil*	*111*	*Gent*	*Motor*	*Motortillverkning*
Montera skruv 1111	*Bil*	*1111*	*Gent*	*Interiör*	*Interiörtillverkning*
Montera skruv 1	*Bil*	*1*	*Torslanda*	*Hytt*	*Hyttillverkning*
Montera skruv 11	*Bil*	*11*	*Torslanda*	*Kaross*	*Karosstillverkning*
Montera skruv 222	*Bil*	*222*	*Torslanda*	*Motor*	*Motortillverkning*
Montera skruv 333	*Bil*	*333*	*Gent*	*Motor*	*Motortillverkning*
Montera skruv 2222	*Bil*	*2222*	*Gent*	*Interiör*	*Interiörtillverkning*
Montera skruv 3333	*Bil*	*3333*	*Gent*	*Interiör*	*Interiörtillverkning*
Montera skruv 2	*Bil*	*2*	*Gent*	*Hytt*	*Hyttillverkning*
Montera skruv 3	*Bil*	*3*	*Torslanda*	*Hytt*	*Hyttillverkning*
Montera skruv 22	*Bil*	*22*	*Torslanda*	*Kaross*	*Karosstillverkning*
Montera skruv 33	*Bil*	*33*	*Torslanda*	*Kaross*	*Karosstillverkning*
Total					

Figur 28. Figuren visar att aktiviteterna har förflyttats från att ligga i separata Excel-dokument till att ligga i ett enskilt ark. Exempelfabriken producerar bilar i fabriker i Gent och Torslanda. Subsystemen inkluderar motor, interiör, hytt och kaross.

Den hierarkiska strukturen skulle förslagsvis kunna automatgenereras från metodstudieanalysen som generats från figur 29. Där skulle även utnyttjandegraden (U_M) kunna läggas in från resultatet av frekvensstudien.

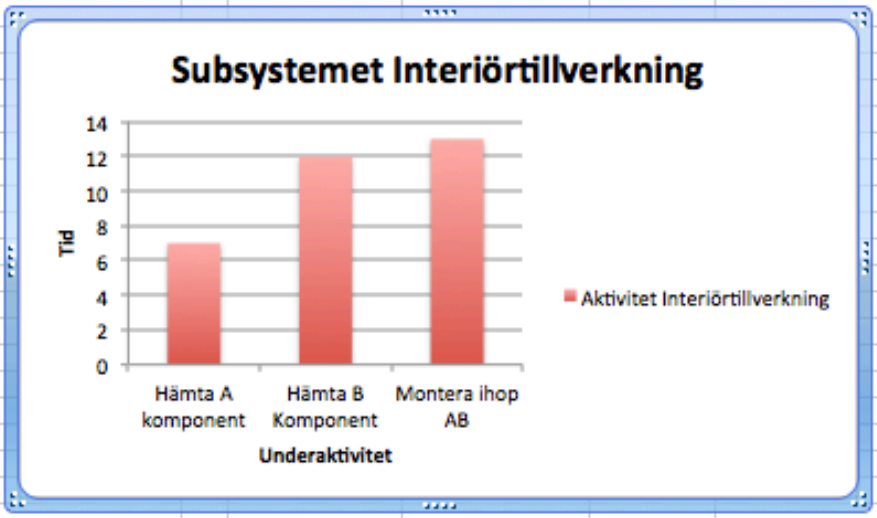


Figur 29. Bilden visar ett förslag på hur den hierarkiska strukturen samt hur resursutnyttjandet läggs in för varje nivå skulle kunna ske.

Därifrån skulle Excel utvidgas till att innefatta olika flikar där CAP_I, CAP_R, tidsekvationer, tidsdrivare, utnyttjandegrad och prestation illustreras för samtliga hierarkiska nivåer. Figur 30 illustrerar dessa tal för subsystemet interiörtillverkning.

Underaktivitet	Tid	Ideal Kapacitet	Utnyttjandegrad	Prestation	Faktisk kapacitet	CAPr	CAPi
Hämta A komponent	7	514					
Hämta B Komponent	12	300					
Montera ihop AB	13	277					
Summering	32	277					

$$X_1 * Hämta\ A\ komponent + X_2 * Hämta\ B\ komponent + Montera\ ihop\ AB$$



Figur 30. Figuren illustrerar hur CAP_I, CAP_R, prestation, utnyttjandegrad och tidsdrivare synliggörs för det exemplifierade subystemet interiörstillverkning.

Metodstudien skulle även vara möjlig att utföra i Excel men då krävs en separat videouppspelningsmjukvara för att visuellt kunna ta fram tider. Att ha möjligheten att föra in data i samma program som videon spelas upp i är en fördel då detta kan utföras effektivare tidsmässigt än vid skiftning mellan två olika mjukvaror. Därefter kan all data transformeras så att önskad output från mjukvaran ges. Med hjälp av detta skulle det även vara möjligt att skapa en färdig rapportmall där önskad feedback kan ges. Det är även möjligt att automatiskt generera grafer i PowerPoint då VBA-script möjliggör detta.

5.3.2 AviX + Excel + Visio

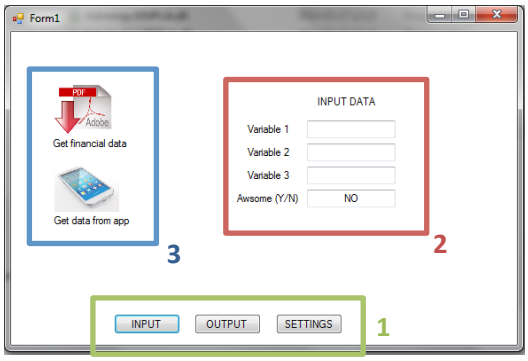
Genom att lägga Visio till ovanstående paket skulle även grafer som precedensdiagram och cirkeldiagram kunna automatgenereras från mjukvaran. Detta gör att den här kombinationen uppfyller kraven och ses som ett möjligt sätt att utföra uppgiften på. I föregående lösning skulle det behöva utföras manuellt av analytikern. I övrigt skulle den här kombinationen fungera likadant som kombinationen ovan.

6 Prototyp

Baserat på de krav som tagits fram i kravspecifikationen var en av frågeställningarna *”Hur kan en potentiell demonstrator utformas?”.* Det är den frågan som kommer besvaras i detta kapitel. Kapitlet inleds med de allra första versionerna av prototypen och avslutas med att presentera den slutliga.

6.1 De första versionerna

Nedan beskrivs de första versionerna av prototypen som främst användes som ett verktyg för att medla mellan kravställare och utvecklare, för att underlätta kommunikationen dem emellan samt säkerställa att de båda siktade mot samma mål.



Figur 31 Första konceptets inmatning av data.

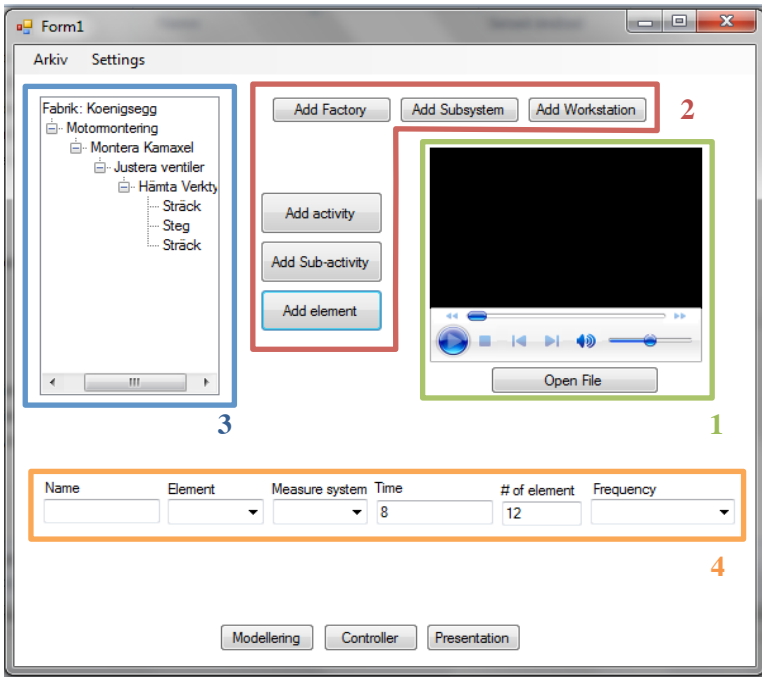
Det första konceptet hade ingen implementerad funktionalitet utan syftade endast till att visa möjliga funktioner. Konceptet hade ett enkelt användargränssnitt med en manöverpanel för att navigera mellan prototypens olika delar (ruta 1). I figur 31 ses konceptets layout då inmatning av parametrar sker och i figur 31 ses konceptets layout då resultat genererats. Inmatning av data sker i huvudsak genom textrutor (ruta 2). Om data finns tillgänglig för inläsning från till exempel PDF, från mobilapplikation eller liknande (ruta 3) kan fälten fyllas i automatiskt.

Figur 32. Första konceptets presentation av data.

Resultaten presenteras med lämpliga interaktiva grafer eller figurer där användaren kan zooma och panorera för att fokusera på relevant information i figur 32 (ruta 4) samt numeriskt (ruta 5). Figurerna kan till exempel bestå av en duPont-modell för att visualisera hur lönsamheten hos företaget påverkas av förändringarna eller hur mängden personal vid en given tid kan ändras, här gestaltade av Tintin-figurerna Dupond och Dupont.

Prototypen kan sedan generera en PDF alternativt en Powerpoint-presentation med informationen enligt förutbestämda mallar (ruta 6).

Då denna första version tagits fram utvecklades den andra versionen av prototypen, nu med viss implementerad funktionalitet.



Figur 33. Andra konceptets inmatning av data.

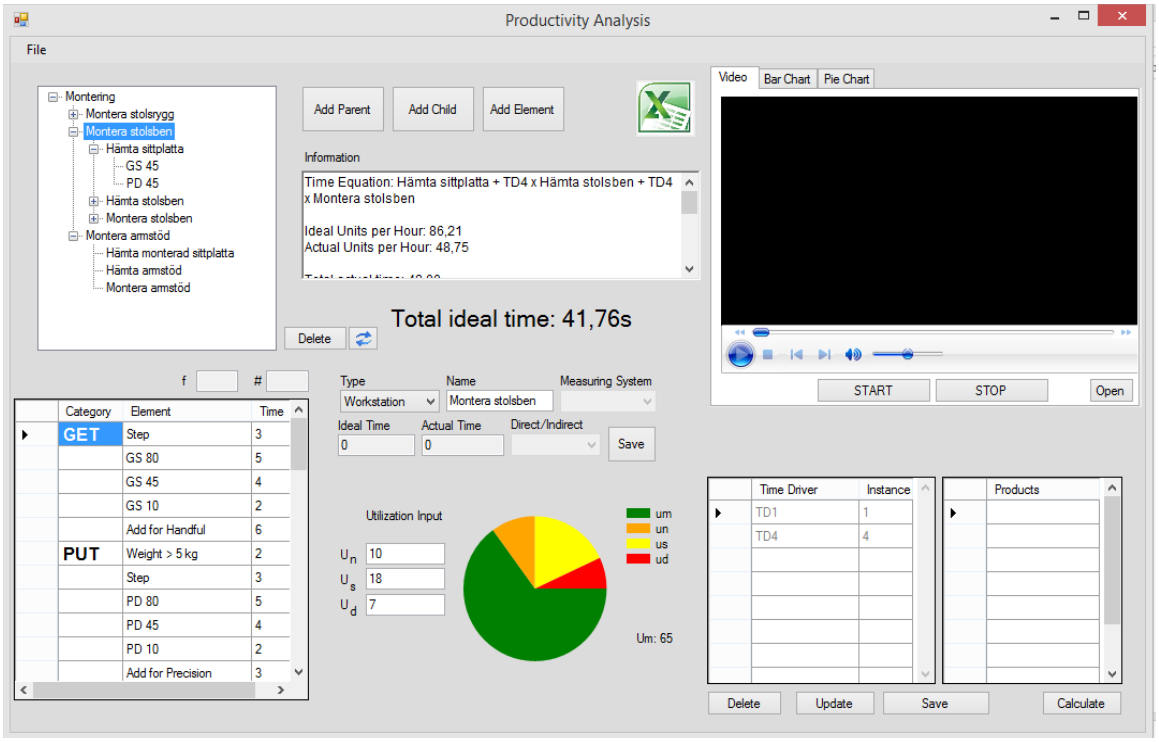
Fabrik, subsystem, arbetsmoment och dess beståndsdelar ned till element kartläggs och byggs upp steg för steg med hjälp av knapparna i figur 33 (ruta 2) till ett hierarkiskt system. Varje steg i hierarkin har en egen knapp.

Träddiagrammet (ruta 3) visar strukturen och gör det möjligt att enkelt redigera och ändra de olika beståndsdelarnas egenskaper (ruta 4) så som namn, standardtid och om det förekommer cykelvis, batchvis eller periodiskt. Som alternativ till att arbeta i realtid alternativt överföra sina egna anteckningar kan användaren utgå från en film så att prototypen automatiskt räknar ut tiden baserat på filmens speltid (ruta 1).

6.2 Slutlig prototyp

Efter utvärdering av funktionerna hos den andra versionen ändrades designen något för att bättre uppfylla kraven och bli mer intuitivt och användarvänlig. Bland annat byttes panelen för inmatning av produktionsmodellens delar (ruta 2) ut, inmatningen av element blev mer lättöverskådlig och mediaspelaren fick fler funktioner.

Det resultat som till sist erhölls är en verifierad och validerad prototyp som realiserar informationsmodellen. Det är ett förslag på hur en framtida demonstrator skulle kunna utformas.



Figur 34. Skärmdump av den slutliga prototypen med det case som användes för utvärderingen.

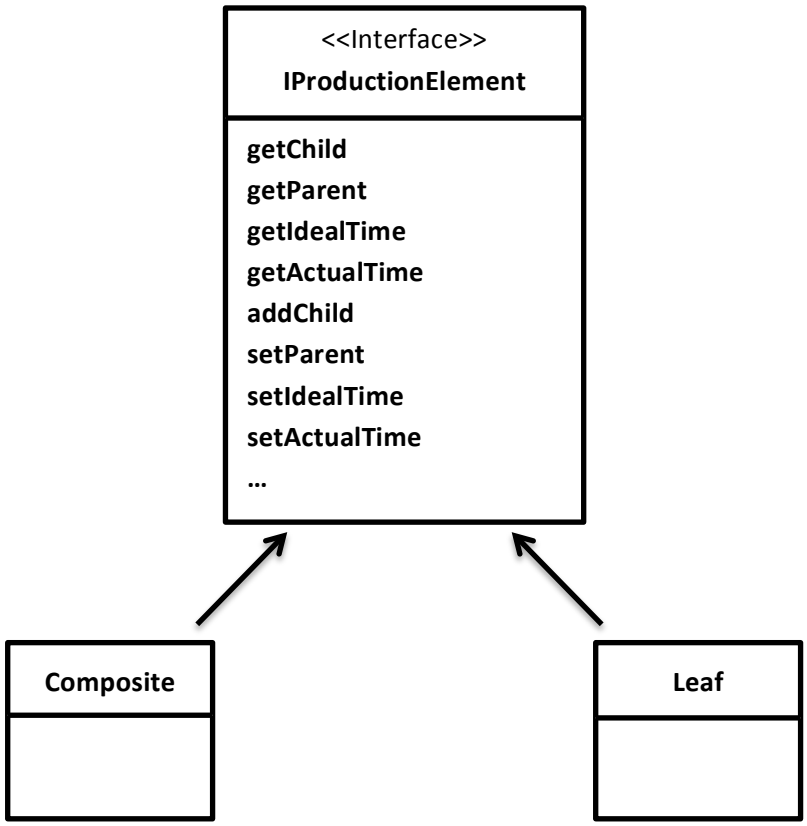
Som kan ses i figur 34 så bär den slutliga prototypen stora likheter med de tidigare versionerna av prototypen.

Prototypen låter användaren bygga upp produktionssystemet enligt informationsmodellen. Tack vare objektens generella egenskaper finns det dock möjlighet att hoppa över, lägga till eller omdefiniera steg i hierarkin efter behov. Det är alltså fullt möjligt att inte skapa en fabrik, till exempel, och istället börja med att skapa ett subsystem.

Elementen utgår från tidsstudieverktyget SAM och nyttjar tidsenheten faktorer medan aktiviteter och subaktiviteter har tiden definierad som sekunder. Att de har olika tidsenheter beror på att elementen kan läsas in från Excel-ark genererade från produktionsmjukvaran AviX där tiden anges i faktorer, medan tiden för aktiviteter och subaktiviteter läggs till manuellt. Till det uppbyggda produktionssystemet är det möjligt att mata in utnyttjandegrad på önskad anläggningsnivå samt erhålla prestationsparametrar. Nedan presenteras prototypens funktioner.

6.2.1 Produktionssystemets uppbyggnad

Den filosofi som valdes för att i prototypen bygga upp produktionssystemets hierarki är composite pattern som är en praktisk tillämpning av composite-filosofin. Valet gjordes med stöd av boken ”*Intoduction to Design Patterns in C#*” (Cooper, 2002). Composition är en av två huvudsakliga filosofier för att bygga upp hierarkiska strukturer, där den andra är arv. Enligt teorin kring relationerna “has a” och “is a” för composition respektive arv valdes en struktur byggd på det förstnämnda då en aktivitet inte är en arbetsstation, men arbetsstationen innehåller aktiviteter. Beslutet stöds även av den andra principen för objektorienterad programmering fastslagen av *Gang of Four* i boken *”Design Patterns: Elements of Reuseable Object-Oriented Software”* (Gamma et al., 1994).



Figur 35. UML-diagram över de två klasserna som modellerar produktionssystemet.

Klassdiagrammet, se figur 35, beskriver de ingående klasserna. Composite-klassen används för att beskriva fabrik, subsystem, arbetsstation, aktivitet samt subaktivitet eller kort och gott allting som kan ha underordnade. Eftersom den enda skillnaden mellan de olika komponenterna är fördefinierade textsträngar kan användaren enkelt definiera egna komponenter och delsystem om det underlättar hanteringen. Leaf-klassen används i dagsläget endast för elementen, men kan även användas till andra typer av komponenter (till exempel andra typer av elementartidssystem) som aldrig är överordnade en annan komponent i enlighet med composite-mönstret.

För att underlätta vid iterering över trädstrukturen har klasserna implementerats med så kallade doubly linked lists. Det innebär att varje nod i trädet har en referens till dess överordnade, och det är alltså enkelt att gå från ett element upp till dess subsystem. Alternativet, att iterera genom hela trädet med varje nods underordnade till rätt element är funnet är visserligen någonting som med dagens datorer och effektiva algoritmer går snabbt för mindre strukturer, men vars beräkningsintensitet växer snabbt i takt med antalet noder. Att använda dubbla referenser är dock någonting som kan generera stora problem om inte referensen för den överordnade uppdateras vid en eventuell ändring. Detta är inget frekvent förekommande i ett produktionssystem, men en klar nackdel att ta i beräkning.

6.2.2 Inmatning av data

Inmatning av data från tidsstudierna kommer främst ske från ett förutbestämt Excel-ark då det är standardförfarandet idag och således någonting användarna är familjära med. Det finns dock möjlighet att bygga upp modellen över produktionssystemet direkt i prototypen och att efter eget tycke redigera modellen i prototypen efter inläsningen. Inläsningen från Excel sker med biblioteket *SpreadsheetLight* som lyder under en MIT-licens för Open Source mjukvara.

Om användaren väljer att bygga upp och tilldela alla värden direkt i prototypen sker detta i huvudsak med vanliga textrutor som prototypen konverterar till för variabeln aktuell datatyp. Av hänsyn till prototypens robusthet genereras endast ett felmeddelande om konvertering inte är möjlig, till exempel om textsträngen som ska konverteras till ett tal innehåller andra tecken än siffror alternativt decimalpunkt istället för decimalkomma. För data som är av typen listor såsom element, tidsdrivare och produkter används standardverktyget GridView. I objektet sparas då namnet, numret och radnumret i GridView:n.

På samma sätt som för den andra versionen av prototypen kan användaren använda en film som stöd vid inmatningen av data för arbetsmomentet. Denne kan då enkelt definiera när i filmen momentet startar och slutar, varefter prototypen automatiskt räknar ut tiden baserat på filmens speltid vid de båda tillfällena. Alternativt används filmen endast som stöd vid SAM-studie.

6.2.3 Utmatning av data

All utdata genereras varje gång den efterfrågas av användare eller prototyp och "caching results" som nämns i teorikapitlet förekommer alltså inte.

För de diagram prototypen genererar används standardfunktionen för diagram i C#, klassen *Chart*. Övrig information presenteras i textetiketter, också kallade labels. I de fall där behovet för användaren att presentera mer text funnits har standardklassen RichTextBox använts. Den sköter till exempel radbrytningar automatiskt och har en inbyggd funktion som låter användaren scrolla i de fall textens antal rader överskrider textrutans storlek vilket sparar mycket plats jämfört med alternativet att skriva ut all text med endast labels.

6.2.4 Resultat och analys av data

Då all indata är inlagd är det sedan möjligt att använda prototypen som ett verktyg för att analysera produktionssystemet med hjälp av de utdata som genereras. De viktigaste resultaten som visas i prototypen är ideal och verklig kapacitet, prestationsparameter, utnyttjandegrad samt tidsekvationer.

Genom att studera utdata är det möjligt att undersöka vilka aspekter som skulle kunna förbättras. Sedan kan användaren genomföra förändringar i produktionssystemet i prototypen och analysera resultatet av dessa förändringar för att sedan kunna dra slutsatser. Prototypen ger användaren möjlighet att spara sitt produktionssystem som en binär fil för att öppnas vid ett senare tillfälle. Det är även möjligt att spara om produktionssystemet som en ny fil då förändringar gjorts. Detta innebär att användaren kan gå tillbaka till äldre scenarier för att till exempel jämföra nuläget med ett eventuellt framtida system. En mer utförlig beskrivning av hur prototypen fungerar och kan användas finns i Bilaga C.

Till prototypen finns en manual som användare kan använda som ett hjälpmedel för att bygga upp sitt produktionssystem för analys. Förklaringar till begrepp och funktioner omfattas av manualen, samt steg-för-stegbeskrivningar i hur man går till väga. Det finns även ett kapitel som beskriver hur man kan använda resultaten som genereras från prototypen för att analysera sitt produktionssystem. Manualen är alltså en utförlig beskrivning i hur prototypen fungerar samt en hjälp på vägen för att användningen av prototypen ska gå så smidigt som möjligt. Manualen i sin helhet kan ses i Bilaga C.

7 Diskussion

Enligt projektets tidsupplägg, vilket visar att dokumentation och litteraturstudie påbörjades direkt vid projektets uppstart. Dokumentationen fortsatte genom hela projektet fram till presentationen medan litteraturstudien tog slut ungefär då en fjärdedel av projektet återstod. Efter att litteraturstudierna påbörjats startade arbetet med att ta fram kravspecifikationen. Därefter startade utvärderingen och sedan startade prototyputvecklingen. Projektets olika delar pågick till stora delar parallellt och utvecklades iterativt.

Det här upplägget hade kunnat se annorlunda ut beroende på vilka beslut som tagits initialt. Till exempel hade någon del kunnat startat tidigare i projektet, hållit på längre eller avslutats senare. Starten av projektet gick åt till att få inblick i området för projektet genom övergripande och ingående litteraturstudier inom produktivitetsteori och Software Engineering. Det krävde ingående förklaringar och beskrivningar av informationsmodellen för att förstå vad demonstratorn ska realisera. Det var i samband med att informationsmodellen introducerades som arbetet med kravspecifikationen startade. Det möjliggjorde att ytterligare frågor kunde ställas för att förtydliga informationsmodellens funktioner och samband. Dessa funktioner och samband skrevs sedan ned som krav i kravspecifikationen. Eftersom de resultat som projektet genererade kan anses användbara och uppfylla frågor som initialt ställdes bör den modell och det upplägg som valdes för projektet anses som lämpliga och effektiva.

Som tidigare nämnts är prototyping bra för projekt där kraven är oklara och svåra att definiera. Den beskrivningen stämmer in på detta projekt varför prototyping har valts som övergripande projektmodell. Dess främsta egenskap är iterering mellan olika projektfaser vilket i detta projekt använts för att hålla bra kommunikation mellan de olika leverablerna. Detta för att kontinuerligt kunna validera och kontrollera utvecklingen av kravspecifikationen och prototyputvecklingen mot kravställare. Itereringen går mellan kommunikation, planering, modellering och konstruktion av prototyp samt användning och feedback. Detta följdes i projektet i och med att det första steget i varje iteration var att kommunicera med kravställare för att förståelse för en specifik önskad funktion i demonstratorn. Därefter planerades hur funktionen skulle realiseras innan den implementerades i prototypen. Därefter utvärderades funktionen, både med avseende på relevans och hur väl den fungerade, för att sedan ta ställning till om funktionen skulle formuleras i ett krav eller avfärdas.

Fördelen med prototyping i jämförelse med vattenfallsmodellen är att vattenfallsmodellen inte tillåter iteration mellan olika projektfaser. Vid användande av vattenfallsmodellen görs antagandet att inga krav förändras under projektets gång. Det försvårar arbetet med att sätta krav när kravanalytiker behöver vara säker på att kravet gäller. Det hade troligtvis lett till att kravspecificeringen drar ut på tiden och fördröjt starten av övriga projektdelar. Alternativet hade varit att en kravspecifikation hade hetsats fram för att kunna starta de andra delarna i projektet. Kravanalytiker ställs då inför ett dilemma där det blir en balansgång mellan att få en utförlig kravspecifikation och att starta nästkommande steg i projektet.

Spiralmodellen är en projektmodell som är snarlik prototyping i och med att de båda itererar genom projektets faser. Dock är denna modell uppbyggd i varv och ett varv fokuserar på en del av utvecklingen, till exempel kravställning, funktionalitet eller systemdesign. I projektet var det dock mer önskvärt att fokusera på de delar i utvecklingen som det för tillfället fanns mest information om eller som var mest relevant. Initialt låg fokus på att implementera de mest grundläggande funktioner

för att allt eftersom projektet fortskred flytta fokus mot mindre vitala funktioner. Om spiralmodellen hade valts hade slutresultatet troligtvis inte skiljt sig så mycket från de resultat som erhållits med prototyping. Däremot ansågs prototyping bättre följa projektets utformning.

7.1 Kravspecifikation

Under projektets gång har synen på kravspecifikationens relevans och omfattning ändrats radikalt. På grund av det krävdes en omfattande litteraturstudie samt konsultation med Dr Richard Berntsson Svensson vid Chalmers tekniska högskola för att på ett tillfredsställande sätt kunna specificera kraven för mjukvaran. Framförallt visade Svensson på vilken typ av krav som är av störst vikt vid mjukvaruutveckling och hur krav ställs då det inte alltid finns fysiska mätbara attribut. Med hjälp av kravställare har sedan de två kravspecifikationerna arbetats fram iterativt genom test med prototypen.

Den iterativa modell som användes ökade kraven på kommunikationen då de olika delprojekten snabbt växte och blev svåra att överblicka eftersom de pågick samtidigt. Detta talar för att vattenfallsmodellen möjligen vore bättre lämpad för detta projekt då den koncentrerar arbetet till en fas i taget. Samtidigt är den modellen mer tidskrävande då arbetet tenderar att bli mer ineffektivt då den medför att teamen internt blockerar de andra, vilket sannolikt hade gjort kraven mindre specificerade då detta omöjliggjort itereringen mellan prototypen och kravhanteringen.

Att sätta sig in i ett så omfattande område som Software Engineering och få en överskådlig bild tog längre tid än vad som reflekteras i rapporten. I början blev informationssökningen något för bred och det blev svårt att konkretisera litteraturstudien för just kravspecifikationen. Men i takt med att litteraturstudien bredde ut sig uppenbarades det att det behövdes verktyg för att hjälpa till och ge tydlighet i kravarbetet. Som hjälp genom Lauesen (2002) metoder ställt mot Pressman (2005) blev det en klarhet i arbetet. Att lyckas med mjukvaruutveckling bygger på att kravanvändningen går till på ett sätt där analytiker hanterar och aktivt söker funktioner i krav mot mjukvaran. Tydlighet i krav fås genom att variera sättet att framföra det och anpassa metoden efter komplexiteten hos funktionen.

”Kravspecifikationen för utvärdering” av program utvecklades i tidigt skede under projektet och behövde inte förklaras ytterligare eftersom det var projektteamet som själva skulle använda den som hjälpmedel. Den förenklade arbetet med att undersöka befintliga mjukvaror dels då den utgjorde standarden för vad mjukvarorna skulle klara av och den underlättade dokumentationen av mjukvarornas potential. På grund av att den var framtagen för eget bruk innehas en djupare förståelse av varje krav eftersom det är de involverade i projektet står för framtagning, användning och utvärdering.

Den mer utförliga ”*Kravspecifikationen för mjukvaruutveckling”* hade helt andra krav på utförligheten och tydligheten eftersom det var ett externt dokument som skall läsas av personer ej insatta i produktionsteknik. Med hjälp av Software Engineering och Software Requirements började transformeringen av kraven. Tillvägagångssätten från (Pressman, 2005) och kravställning i form av funktionella krav tillsammans med de grafiska hjälpmedlen presenterade av Lauesen (2002). Genom tester av den egna prototypen och genom specificering av redan ställda krav växte kravspecifikationen. Då ”*Kravspecifikationen för mjukvaruutveckling”* har fått krav från den egna prototypen kan det vara så att den är vinklad efter projektets andra resultat. Det skulle kunna resultera i att fel från prototypen överförs till slutgiltig demonstrator och till kraven ställda i

kravspecifikationen. Detaljeringen i kraven kommer till viss del från de tester som utförts på prototypen. Det kan liknas med spårning av kravens funktion som tidigare refererat (Lauesen, 2002).

Kraven visas också för kravställare och valideras kontinuerligt vilket gör att både prototyp och kravspecifikation får krav med hög precision, både gällande detaljnivå och korrekthet på kravet. En svårighet har varit att ställa kraven på ett sådant sätt att fristående mjukvaruutvecklare skall förstå innebörden. Informationsmodellen upplevs av många som komplex varför den är svår att förklara i krav för utomstående. Dessutom är det svårt att avgöra vilka delar i informationsmodellen som kräver mest förtydligande och är mest relevanta för en mjukvaruutvecklare.

För att väga upp för detta har kravspecifikationen strukturerats i enlighet med vad som bör infattas av en kravspecifikation enligt Lauesen (2002). Hade det funnits mer tid för fortsatt arbete hade en möjlighet för ytterligare validering varit att göra tester där dokumentet lämnas över till exempelvis studenter på institutionen för datavetenskap. Eftersom detta inte gjorts skulle en lösning för bättre förståelse av informationsmodellen vid ett utvecklingsprojekt vara att kravspecifikationen överlämnas tillsammans med projektrapporten. Då beskrivs delar som kan vara till hjälp vid utvecklingen av den slutliga demonstratorn, både teoretisk bakgrund liksom de delar som beskriver hur prototypen har utformats. För att ytterligare underlätta vid ett framtida utvecklingsprojekt bör även prototypen med tillhörande källkod bifogas.

Målet har varit att kunna lämna vidare kravspecifikationen till en mjukvaruutvecklare för utveckling av en slutgiltig demonstrator. För att detta ska vara möjligt behöver den kompletteras med:

- Kvalitetskrav: Prestanda
- Datakrav: Databas och kommunikationskanaler
- Organisationskrav: Rättsliga och leveranstid

Varför har inte detta gjorts? Att sätta prestandakrav på mjukvara beror mycket på vad det är för hårdvara (dator) användaren av mjukvaran sitter med. Det skall alltså först sättas standarder för vad som är rimligt att slutanvändaren skall använda för system. Detta blir en första test av informationsmodellen med virtuellt hjälpmedel där slutproduktens krav kan komma att skilja en del från prototypens. För att undvika otydlighet med underkvalificerade krav som behövs sättas med tester för eller uppskattningar tagna från luften har detta avgränsats. Prestanda bygger till stor del på databaser och kommunikationskanaler. Projektet lämpar sig mot att testa kvalitet och utformande av informationsmodellerna och inte vilken typ av databas som skall användas enligt modellerna (Lauesen, 2002). Därför utesluts även dessa delar från kravspecifikationen som gjorts i projektet. Det är delar som är mer lämpade till teknologer med bakgrund inom datorvetenskap.
De rättsliga- och leveranskrav som inte specificerats i den utförliga kravspecifikationen kommer först vid beställandet av ett utvecklingsteam för mjukvara. De kan ta upp rättsskyddsfrågor, beställningskrav och sådant som kommer vid senare skeden i utvecklingen.

Med det sagt presenteras ”Kravspecifikationen för mjukvaruutveckling” i bilaga B.

7.2 Programutvärdering

Inledningsvis konstaterades i teorin att ett utvärderingsteam med blandad bakgrund och kompetens är mest fördelaktig. Teamet i projektet är homogent, vilket kan ses som en nackdel eftersom det gör att vissa viktiga aspekter såsom potentialen med till exempel mjukvaror och tillhörande egna script

kan ha missat. Det har försökt åtgärdats genom att intervjua sakkunniga personer som till exempel tekniklektor Per Medbo för att få en övergripande bild om mjukvarukategorierna.

En annan problematik som konstaterades är att det finns en stor mängd möjliga mjukvaror med skiftande kvalitet. Detta ökar betydelsen av en utförlig informationssökning för att sedan kunna eliminera lösningar som inte förväntas klara kraven. Notera att en mer övergripande litteraturstudie eller fler intervjuobjekt eventuellt hade resulterat i fler eller alternativa mjukvaror att utvärdera. Detta begränsar studiens generaliserbarhet men det möjliggör en snabbare utvärderingsprocess där mer tid kan riktas på att utvärdera de utvalda mjukvarorna.

Med en kort tidsperiod krävs att utvärderingen av mjukvaror går relativt fort. Då majoriteten mjukvaror är sedan tidigare okända för gruppen och framförallt produktionsmjukvarorna har ett upplägg som är nytt och okänt. Därför är det tänkbart att möjligheten att utnyttja mjukvarorna på ett fungerande sätt förbisetts. Detta medför att mjukvarornas potential missbedömts och att väsentliga brister ej uppmärksammats. Testaren har i vissa fall haft tillgång till manual men har alltid varit tvungen att lära sig mjukvaran själv från grunden. Detta är även applicerbart på utvecklingen av VBA-script i Excel. Det finns risk att utvärderingsteamet missat vissa funktioner som skulle ha varit användbara för att konstruera en konceptuell mjukvara som bättre möter kraven men kan likväl vara så att funktioner överskattas.

I det teoretiska ramverket konstaterades svårigheten med att översätta krav till kriterier. Detta har även upplevts i projektet eftersom *"Kravspecifikationen för utvärdering"* av redan existerande mjukvaror har gjorts generell för att inte vara så strikt att inga mjukvaror kan uppfylla kraven. Detta har medfört att kraven är utformade så att det finns utrymme för tolkning. Ur ett utvärderingsperspektiv hade det därför varit önskvärt att kraven definierats tydligare, det vill säga att dessa översatts till kriterier.

Vissa mjukvaror uppfyller kravet, men om sättet som detta utförs på inte är acceptabelt enligt informationsmodellen har kravet ändå bedömts att inte uppfylls. Detta gäller även motsatsen. Hur kraven har bedömts har alltså med tanke på kravspecifikationens utformning varit godtyckligt, vilket motiverats med att projektet endast utgör förstudien till hur en demonstrator ska utvecklas. Projektet har syftat till att undersöka hur alternativa demonstratorer kan utvecklas och det är därför inte önskvärt att utesluta mjukvaror på grund av för strikt ställda krav. Målet har således inte varit att komma med en exakt på lösning på problemet utan syftar istället till att undersöka hur en möjlig lösning kan utformas.

Kombinationslösningarna som valdes skulle vara möjliga att uppfylla de funktionella krav som ställts. En för alltför stor skillnad mellan mjukvarans förmåga och kraven som ställts skulle därför innebära att det skulle krävas mycket resurser för att åtgärda felen, vilket är i enlighet med gap-analysen. Därför (Comella-Dorda et al., 2002)valde utvärderingsteamet att bortse från Casat när alternativa lösningskombinationer utformades. Den främsta orsaken var att Casats definition av utnyttjandegrad skiljer sig från informationsmodellens. Det antogs därför att det skulle krävas att Casat omformas efter informationsmodellen, vilket ej bedöms vara möjligt.

Lösningarna som presenterades valdes genom värdera vad respektive mjukvara har för konkurrensfördelar. Bedömningssystemet baserades i första hand på om mjukvaran uppfyllde kravet eller inte och i andra hand på hur mjukvaran upplevdes generellt. Eftersom det finns svårigheter i att

sätta konkreta kriterier som överträffade målen valdes bedömningssystemet beskrivet ovan. Att två kombinationer valdes att analyseras berodde dels på vilka resurser som fanns tillgängliga men desto mer på att de valda kombinationerna upplevdes som de enda möjliga. Casat föll bort eftersom gapet till en fullgjord mjukvara var för stort medan Matlab föll bort eftersom hur det än kombinerades inte skulle kunna uppnå metodstudiekravet bättre än någon annan lösning. Detta berodde på att AviX var enda programmet av de kvarvarande som uppfyllde kravet för metodstudie och att metodstudien gavs i Excel-format. Eftersom Matlabs förmåga att utföra enkla matematiska operationer inte bedömdes vara bättre än Excels förmåga bedömdes det som att Matlab inte skulle vara meningsfull att använda.

Slutligen kan följande brister kopplas till de konceptuella lösningarna Avix + Excel och Avix + Excel + Visio som är snarlika varandra. Kombinationerna uppfyller samtliga funktionella krav men upplevs ha bristande användarvänlighet, robusthet och de kräver att flera mjukvaror kombineras. Användarvänligheten vid kombinationer av mjukvaror tenderar att bli försämrad då många program som ska samverka ökar komplexiteten för användaren. Uppstartsfasen och installationen bedöms ta längre tid eftersom flera mjukvaror kräver flera mjukvarulicenser. Därutöver anses användarvänligheten problematisk eftersom sparandet av Excel-dokument skulle kräva att dessa sparas korrekt i en särskild mapp. Därutöver har utvärderingsteamet inte lyckats hitta någon bra lösning som skulle lösa robusthetsproblemet om till exempel en bokstav väljs istället för siffra eller om en siffra är inom fel intervall. Slutligen kan Excel upplevas som simpelt som program eftersom den mesta logiken bakom programmet blir relativt synlig.

7.3 Prototyp

Nedan diskuteras prototypen både gällande utvecklingsarbetet och utvecklingsarbetet. Diskussionen går även in på hur iteration mellan prototyputveckling och framtagning av krav.

7.3.1 Utvecklingsarbetet

UML-diagrammet låg till grund för den modell av systemet som prototypen realiserar. Stor vikt lades initialt på flexibilitet för framtida ändringar varpå ett antal förenklingar av modellen gjordes. En inledande litteraturstudie resulterade i det designmönster som senare kom att realisera informationsmodellen, det så kallade composite-mönstret som likt informationsmodellen bygger på composition och inte arv mellan klasser vilket gör den flexibel för framtida ändringar. Vid närmare inspektion av egenskaperna för hierarkins ingående delar upptäcktes en rad likheter både vad gäller dess attribut och funktioner. Tack vare detta kunde hela modellen realiseras med endast två olika klasser. Den ena klassen utvecklades för elementartidssystemet, i det här fallet SAM-elementen, och den andra klassen utvecklades för de delar i hierarkin som kan ha underordnade såsom anläggning, aktivitet och subaktivitet. Mängden kod kunde på så sätt hållas nere vilket snabbade på utvecklingsprocessen avsevärt.

Ett potentiellt önskemål från kravställare för den framtida demonstratorn är att den ska vara web-baserad. Det är möjligt att genomföra med prototypen då C# delar många funktioner med ASP.NET, som relativt enkelt kan realisera ett program skrivet i C# som en websida. Att göra prototypen web-baserad är inget som tagits hänsyn till i detta projekt men det skulle alltså vara fullt möjligt att göra det för den framtida demonstratorn.

7.3.2 Utvärdering av prototypen

Prototypen som tagits fram visade sig underlätta kommunikationen mellan kravställare och projektgrupp genom att båda parter kunde uttrycka sina funderingar och önskemål visuellt. Framtagningen av prototypen och kravspecifikationerna underlättades av iterationen dem emellan. Utvärderingen av prototypen utgjorde underlag för kravspecifikationen på två sätt. För det första utvärderades kraven genom att implementera dem i prototypen och för det andra genom att funktionalitet i prototypen resulterat i nya krav. Med hjälp av prototypen kunde även initialt ställda krav förtydligas och förbättras. Några av de krav där prototypen varit till hjälp för att utforma och förbättra krav presenteras nedan.

Att användning av prototypen och caset hjälpt till att utforma nya krav visas med exemplet som följer. Från kravställaren var det ej givet att användaren manuellt skulle kunna mata in en ideal tid för en aktivitet eller en subaktivitet. Istället var det endast tänkt att den ideala tiden skulle fås fram genom att bygga upp aktiviteten eller subaktiviteten med hjälp av element. Vid användning av prototypen och caset upptäcktes det däremot att det skulle vara önskvärt för användaren att den möjligheten finns. Detta behov kan till exempel uppkomma när en SAM-studie har utförts manuellt och det inte är önskvärt att lägga in alla element i prototypen eller när en aktivitets eller subaktivitets ideala tid har uppskattats baserat på verklig tid och resursens prestationsförmåga. Detta önskemål resulterade i följande två krav:

”Skall kunna definiera ideal tid” som återfinns i ”*Kravspecifikation för utvärdering*” som krav **1.2.2**

”Skall kunna ange data för en subaktivitet eller aktivitet så som: verklig tid, ideal tid, typ, direkt eller indirekt aktivitet.” som återfinns i ”*Kravspecifikation för mjukvaruutveckling*” som **K.4.1.3**.

Prototypen och caset har utöver att bidra till framtagning av nya krav även varit ett hjälpmedel för att precisera redan framtagna krav. Ett krav som togs fram initialt var ”*Modellen skall presenteras medan den hierarkiska strukturen byggs upp”*. Vid användandet av prototypen och caset blev det tydligt att antal delar i hierarkin snabbt växer och blir svåra att överblicka. Därför uppmärksammades behovet av att kunna gömma för stunden irrelevanta delar i hierarkin. Resultaten av detta blev följande krav:

”*Skall ha möjlighet att välja vad som skall visas och inte i systemets olika nivåer, kunna välja att gömma subaktiviteter i ett fall och i ett annat välja att visa alla nivåer ned till elementnivå.”* som återfinns i ”*Kravspecifikation för mjukvaruutveckling*” som **K.4.1.12**.

Exemplen som presenterats ovan visar tydligt prototypens roll i framtagningen och preciseringen av krav till kravspecifikationerna. Somliga krav hade inte blivit lika utförliga och tydliga utan prototypen eftersom förståelsen för önskad funktionalitets betydelse inte varit lika ingående. Kring andra krav hade det inte funnits en förståelse att det var nödvändigt att förtydliga dem om inte användandet av prototypen visat detta. Ytterligare andra krav hade inte alls tagits fram då insikten i att kraven behövde läggas till inte funnits om de inte upptäckts vid användadet av prototypen. Utvärderingen av prototypen tydliggjorde också behovet av ytterligare funktionalitet i prototypen som mynnade ut i fler krav till kravspecifikationerna.

Prototypen är inte lika robust som en kommersiell mjukvara bör vara men den visar ändå tydligt hur en kommersiell mjukvara som ska realisera infomationsmodellen skulle kunna se ut och hur den

skulle kunna användas. I prototypen finns de allra viktigaste funktionerna implementerade. Det är möjligt att bygga upp ett produktionssystem enligt informationsmodellens uppbyggnad, både genom att skapa element i prototypen och genom att läsa in SAM-ark från Excel. Prototypen är flexibel på det sättet att användaren lätt kan ändra och ta bort en tillagd del i hierarkin. Det är även upp till användaren att bestämma vilka hierarkinivåer som ska respektive inte ska finnas med. I prototypen ser användaren inte någon av den bakomliggande programkoden utan ser endast det som användaren har nytta av i användandet av prototypen. Av de existerande mjukvaror som utvärderats i detta projekt finns det ingen enskild mjukvara som klarar av de ovan nämnda funktioner hos prototypen.

I flera av de andra mjukvarorna som testats fås inte den utdata som önskas eller önskad indata går inte att mata in. Dessa brister finns inte hos prototypen eftersom den realiserar informationsmodellen. Detta gör att prototypen blir väldigt lätt att använda för de användare som önskar arbeta enligt informationsmodellen. Det behövs inga ”speciallösningar” för att lägga till och generera det som önskas i prototypen. Men detta innebär också att prototypen inte är anpassad till något annat tillvägagångssätt än informationsmodellen.

Många av de funktioner som finns i prototypen är sådana som är bra att föra vidare även till den slutliga demonstratorn. Viktigast av dessa funktioner är att hierarkins uppbyggnad är intuitiv och lättförståelig i och med att den byggs upp som en trädstruktur vilket många torde vara vana vid att handskas med. Att det genereras grafer i prototypen istället för bara text är nog ett bra sätt att fånga de flesta användares intresse. Speciellt utnyttjandegraden visualiseras på ett mer förståeligt sätt än om det bara hade varit text i och med cirkeldiagrammet där den värdeadderande tiden är grön; ju mer grönt desto större andel arbetstid går åt till planerat arbete.

Prototypen har också vissa funktioner som i dagsläget inte utnyttjas samt en del funktioner som saknas för att prototypen fullt ut skulle kunna användas som demonstrator enligt kravställares önskemål. Både dessa kategorier är möjliga att åtgärda om så önskas tack vare programmeringsspråket och utvecklingsmiljön som används. Nedan beskrivs kortfattat några av de funktioner som skall eller bör utvecklas för att få en fullt utvecklad demonstrator. De funktioner som *skall* utvecklas är krav som ställs på den framtida demonstratorn i ”*Kravspecifikation för mjukvaruutveckling*” i Bilaga B och de funktioner som *bör* utvecklas är önskemål. Notera att önskemål ej återfinns i kravspecifikationen då författandet av dessa skulle kräva en omfattande insikt i arbetsmetodiken för användandet av dylik mjukvara.

Funktioner som finns men inte används i prototypen:

- Möjlighet att välja om en aktivitet är indirekt eller direkt. Implementering av detta krav skulle kunna innebära att användaren har möjlighet att sortera aktiviteterna i dessa kategorier för att skapa processer med endast direkta aktiviteter.
- Möjligheten att välja hur tiden tagits fram. Om denna funktion implementeras möjliggör det en distinktion mellan aktiviteter vars tider är säkrade genom klockning eller mer osäkra eftersom de tagits fram genom uppskattning.

Funktioner som skall utvecklas och implementeras i prototypen för att erhålla demonstrator:

- Möjlighet att skapa tillverkningsprocesser genom att koppla aktiviteter och resurser i enlighet med krav **K.4.1.36**. Aktiviteterna ska läggas till i den ordning de utförs i processen och till varje aktivitet ska en resurs med en prestationsparameter kopplas.
- Möjlighet att spara modellen över anläggningen som byggs i ett standardformat som kan öppnas i andra mjukvaror enligt **K.4.1.37.**
- Möjlighet att generera till exempel PDF, PowerPoint, Excel eller motsvarande med relevant information såsom diagram enligt **K.4.1.25**. Detta möjliggör att användare kan distribuera informationen genererad från demonstratorn och att denna informations kan läsas oberoende av operativsystem.

Funktioner som bör utvecklas och implementeras i slutlig demonstrator:

- Databaser för resurser, element, uppbyggd fabrik samt skapade tillverkningsprocesser för att underlätta lokalisering av data för användare. I dessa databaser ska det vara möjligt för användaren att lägga till ny data, till exempel nya resurser med tillhörande namn och prestationsförmåga. Men det ska också finnas data fördefinierat i databaserna, till exempel bör alla element med tillhörande tider vara importerade från elementartidssystem.
- Alternativt inmatningssystem vid SAM-studie. Detta skulle möjliggöra för användaren att i realtid utföra en SAM-studie genom att elementen finns fördefinierade och användaren lägger till element med kortkommandon såsom ”3S” för tre element av typen step, ”45GS”

Trots dessa funktioner som bör utvecklas är prototypen möjlig att redan nu använda för att analysera produktionssystem. Analysen kan göras tillsammans med manual och case om så önskas. Prototypen är också ett väldigt bra exempel på hur en demonstrator skulle kunna utformas och vilken funktionalitet den bör ha. Den visar också hur demonstratorn skulle kunna ligga till grund för beslut som fattas kring produktionens uppbyggnad genom att på ett tydligt sätt visualisera produktivitetspotentialen.

8 Slutsats

Forskargruppen i Productivity Management på institutionen Material- och tillverkningsteknik vid Chalmers tekniska högskola har identifierat behovet av att på ett enkelt och pedagogiskt sätt illustrera förbättringspotentialer i produktionen enligt en framtagen modell. Projektet syftar till att illustrera hur en sådan demonstrator kan se ut, vilka funktioner den bör innefatta och eventuellt ge vägledning inför ett framtida utvecklingsprojekt.

De krav som bör ställas på en demonstrator som ska visualisera produktionspotentialen på ett tydligt sätt sammanställdes i två kravspecifikationer. Dessa kravspecifikationer har olika abstraktionsnivåer för att kunna användas inom olika områden. Den med högst abstraktionsnivå, det vill säga den minst detaljerade, har främst utgjort stöd vid undersökning av funktionaliteten hos befintliga mjukvaror. Den beskriver de grundfunktioner demonstratorn bör uppfylla. Kravspecifikationen med lägst abstraktionsnivå innehåller med sin högre detaljnivå de krav som bör ställas på en egenutvecklad demonstrator.

För att undersöka vilka kommersiella mjukvaror som skulle kunna vara möjliga att använda som demonstrator utvärderades tänkbara kandidater med hjälp av den minst detaljerade kravspecifikationen. Utvärderingen visade att ingen enskild mjukvara kan uppfylla de krav som ställs på demonstratorn, men att en kombination av AviX, Microsoft Excel och Microsoft Visio uppfyller kraven.

För att ytterligare specificera hur en demonstrator kan utformas och konkretisera dess grundfunktioner har en prototyp utvecklats i programspråket C#. Dess behov aktualiserades ytterligare av att det inte fanns några enskilda befintliga mjukvaror som uppfyller demonstratorns syfte. Kombinationen av kommersiella mjukvaror uppfyller kraven men upplevs som mer invecklad, mindre robust och mindre tilltalande än den egenutvecklade prototypen.

Prototypen är verifierad och validerad och kan tillsammans med tillhörande manual användas för att bygga upp och analysera ett produktionssystem. Tack vare prototypen har kraven kunnat realiseras och testas under projektets gång vilket resulterat i mer relevanta och detaljerade krav. Prototypen i sig ger även en god indikation på hur en slutgiltig mjukvara kan se ut då dess funktionalitet och användarvänlighet ansetts vara mycket god. Den stämmer även väl överens med den slutgiltiga demonstratorns syfte och mål enligt den case-studie som genomförts med prototypen.

Referenser

ALMSTRÖM, P. & KINNANDER, A. 2006. Att mäta produktivitetspotentialen - Rapport om PPA-metoden. Instutionen för Material- och Tillverkningsteknik, Tillverkningsteknik.

ALMSTRÖM, P. & KINNANDER, A. 2011. Assessing and benchmarking the improvement potential in manufacturing systems at shop-floor level. *International Journal of Flexible Manufacturing Systems*.

ALMSTRÖM, P. & WINROTH, M. Why is there a mismatch between operation times in the planning systems and the times in reality? Proceedings of APMS2010, 2010.

BASOUKOS, I., CRONRATH, C., JOHANSSON, J. & SIMONSSON, E. 2014. Saving P&C Inc.**,** 7.

BELLGRAN, M. & SÄFSTEN, K. 2005. *Produktionsutveckling: utveckling och drift av produktionssystem,* Lund, Studentlitteratur.

BRYMAN, A. & NILSSON, B. 2002. *Samhällsvetenskapliga metoder*, Liber ekonomi.

CHISHOLM, A. 1990. Nomenclature and definitions for manufacturing systems. *CIRP annals,* 39**,** 735-742.

CHURCHMAN, C. W. 1968. *Challenge to reason*, McGraw-Hill New York.

CLARK, D., MEISTER, T. & BOOKS24X, I. 2013. *Beginning C# object-oriented programming, second edition,* New York, Apress.

COMELLA-DORDA, S., DEAN, J., MORRIS, E. & OBERNDORF, P. 2002. A process for COTS software product evaluation.

COOPER, J. W. 2002. *Intoduction to Design Patterns in C#,* eBook, J Addison-Wesley Professional.

DOCKER, T. 1998. Successful requirements management. *Requirements Engineering*.

FREIVALDS, A. 2009. *Niebel's methods, standards, and work design*, McGraw-Hill Higher Education Boston, Mass.

GAMMA, E., HELM, R., JOHNSON, R. & VLISSIDES, J. 1994. *Design Patterns: Elements of Reuseable Object-Oriented Software*, Addison-Wesley.

GIAGLIS, G. M. 2001. A Taxonomy of Business Process Modeling and Information Systems Modeling Techniques. *International Journal of Flexible Manufacturing Systems,* 13**,** 209-228.

HEDMAN, R. 2013. Manufacturing Resource Modelling for Productivity Management. 101 pages.

HELMRICH, K. 2001. *Productivity Processes, methods and experiences of measuring and improving*, Informationsgruppen.

HOOD, C. 2008. *Requirements management: the interface between requirements development and all other systems engineering processes,* Berlin; London, Springer Berlin Heidelberg.

HUBKA, V. & EDER, W. E. 1988. Theory of technical systems: a total concept theory for engineering design. *Berlin and New York, Springer-Verlag, 1988, 291 p.,* 1.

HÅGERYD, L., BJÖRKLUND, S. & LENNER, M. 2005. *Modern produktionsteknik: D. 2,* Stockholm, Liber.

IMD 2004. SAM - Sequential Activity - and Methods Analysis - System description.

JALOTE, P. 2009. Jalote, Pankaj. A concise introduction to software engineering. American Library Association.

KLINGSTAM, P. & GULLANDER, P. 1999. Overview of simulation tools for computer-aided production engineering. *Computers in Industry,* 38**,** 173-186.

LAUESEN, S. 2002. *Software requirements: styles and techniques,* Harlow, Addison-Wesley.

MSDN. 2014. *Basics of .NET Framework Serialization* [Online]. Microsoft. Available: <http://msdn.microsoft.com/en-us/library/ms233836(v=vs.90).aspx> [Accessed 2014-05-18 2014].

MYERS, G. J., SANDLER, C. & BADGETT, T. 2011. *The art of software testing*, John Wiley & Sons.

NIELSEN, J. 1994. *Usability engineering*, Elsevier.

PISANO, G. & SHIH, W. 2012. *Producing prosperity: why America needs a manufacturing renaissance*, Harvard Business Review Press.

PRESSMAN, R. S. 2005. *Software engineering: a practitioner's approach,* Boston, Mass, McGraw Hill.

PUNTER, T., KUSTERS, R., TRIENEKENS, J., BEMELMANS, T. & BROMBACHER, A. 2004. The W-Process for software product evaluation: a method for goal-oriented implementation of the ISO 14598 standard. *Software Quality Journal,* 12**,** 137-158.

PUNTER, T. & LAMI, G. Factors of software quality evaluation. Proceedings European Software Control and Metrics Conferences (ESCOM), 1998. Citeseer, 257-266.

ROBINSON, P. A., ETTER, R. & CRCNETBASE 2000. *Writing and designing manuals: operator manuals, service and maintenance manuals, manuals for international markets,* Boca Raton, CRC Press.

ROSS, S. M. 2012. *Simulation,* San Diego, Academic Press.

RUBENOWITZ, S. 2004. *Organisationspsykologi och ledarskap,* Lund, Studentlitteratur.

SAITO, S. 2001. *Case study: reducing labor cost using industrial engineering techniques*, McGraw-Hill, New York.

SKANSHOLM, J. 2010. *Java direkt med Swing,* Lund, Studentlitteratur.

SOMMERVILLE, I. 2011. *Software engineering,* Harlow, Addison-Wesley.

SUNDKVIST, R. 2014. *Financial benefits of shop floor productivity improvements.* Chalmers University of Technology.

WANYAMA, T. & FAR, B. 2008. An Empirical Study To Compare Three Methods For Selecting Cots Software Components. *International Journal of Computing and ICT Research,* 2**,** 34-41.

ZELLER, A. 2006. *Why programs fail: a guide to systematic debugging,* Heidelberg; Amsterdam, Elsevier Morgan Kaufmann Publ.

Bilaga A - Kravspecifikation för utvärdering av mjukvaror

Funktionalitetskrav	**Kravuppfyllande**
1. Funktionalitet	
1.1 Skall definiera fabriksnivåer och aktiviteter för M-parametern	
1.2 Definiera aktiviteter som subaktiviteter	
1.2.1 Skall kunna ange verklig tid	
1.2.2 Skall kunna definiera ideal tid	
1.2.3 Skall kunna definiera tidsdrivare för aktivitet	
1.2.4 Skall kunna definiera tidsdrivare för subaktivitet	
1.3 Skall vissa koppling mellan subaktivitet med dess SAM- element	
1.3.1 Skall kunna använda förutbestämda element	
1.3.2 Möjlighet att definiera egna element	
1.3.3 Möjlighet till skalning i systemnivåer	
1.4 Skall kunna ange prestationsparameter per resurs, P-parametern [%]	
1.4.1 Möjlighet till prestationsmått per aktivitet	
1.5 Skall ange uttnyttnandegradsparameter för processen, U-parametern [%]	
1.5.1 Skall kunna definiera utnyttjandegrad	
1.5.1.1 Skall kunna välja processnivå	
1.5.1.2 Skall visa U-parametrar per aktivitet	
1.5.2.3 Skall visa värdeadderande tid	
1.5.2.4 Skall visa förluster: personlig tid, systemförluster, störningar/haverier	
1.6 Skall kunna visa kopplingar av aktiviteter via precedensdiagram	
1.6.1 Annat sätt att visa flödet på kvalificerat vis	
1.7 Skall kunna exportera data till PowerPoint, PDF eller alt. Presentationsform	
1.7.1 Skall kunna exportera grafer	
1.7.2 Skall kunna exportera värden	
1.7.3 Utnyttjandegrad	
1.7.4. Tidsekvationer	
1.7.5. Prestation	
1.7.6 Exportera listor	
1.7.7 Listor	
1.7.8 Precedensdiagram	
1.7.10 Hierarkisk struktur	
2. Mjukvarans operationer	
2.1 Skall generera tidsekvationer	
2.1.1 Beräkna produktivitetsmått för varje aktivitet, M-Parametern [enh/tid]	
2.1.2 Addera Aktiviteter och subaktiviteter	
2.1.3 Användning av tidsdrivare	
2.2 Beräkna CAP	
2.2.1 Visualisera ideal och verklig kapacitet	
2.3 Möjlighet att spara användardata	
3. Önskvärda utvecklingsmöjligheter	
3.1 Användning	
3.1.1 Möjlighet till automatisk inmatning från SAM- ark (exempelvis Excel)	
3.1.2 Möjlighet till att använda filmfil för att analys	
3.1.3 Möjlighet att ange om aktivitet är direkta eller indirekt	
3.1.4 Möjlighet till personlig anpassning av till	

Bilaga B - Kravspecifikation för utvärdering av mjukvaror

Kravspecifikation för mjukvaruutveckling av demonstrator:

-En kravspecifikation med anknytning till kandidiatarbetet MTTX02-14-07

1. Introduktion

Detta dokument är del av kandidatarbetet "*Visualiseringar av produktionsförbättringar*" som görs på Institutionen för material & tillverkningsteknik, Chalmers under våren 2014. Projektets mål är att göra en förstudie för en demonstrator för att testa modell från tidigare forskning på institutionen (Hedman 2013). Modellen är i kort, ett redskap som skall förtydliga förluster i industrimiljöer. Fördelen är att den jämför verklig kapacitet i produktion jämförd med ideal kapacitet. Jämförelse görs baserat på produktionssystemets metod, prestation och utnyttjandegrad. För att förstå hur modellen fungerar och dess styrkor föreslås att antingen läsa slutrapporten eller för mer ingående se informationsmodellens ursprung Hedman (2013).

Dess syfte är att validera och kontrollera framtagningen av mjukvara för informationsmodellen. Kravspecifikationen skall vara ett hjälpmedel för utformandet av demonstratorn i projektet, verifiera och validera funktionen, för att till sist vara grund för utveckling av slutgiltig demonstrator. Den skall specificera de övergripande kraven på demonstratorn i kandidatprojektet och utvecklingsbehoven för slutgiltiga demonstratorn. Kravspecifikationen är framtagen för att vara en del av utvecklingsarbetet för de krav som ligger på modellen. Det riktar sig till samtliga involverade i projektet för att ta upp och bepröva integration med ett framtida digitalt redskap för modellen. När det väl är dags för ett team mjukvaruutvecklade att ta över och skapa en kommersiell produkt är målet att kravspecifikationen härifrån skall vara ett hjälpmedel i det fortsatta arbetet.

2. Definitioner, förkortningar och akronymer

Nedan beskrivs kort de ämnesspecifika ord som används i rapporten. Orden återfinns i bokstavsordning.

Anläggning (facility) – Fabrik, subsystem och arbetsstation är olika typer av anläggningar. Fabriken består av subsystem som består av arbetsstationer där aktiviteter utförs.

Aktivitet – Byggs upp av flera subaktiviteter. Aktiviteterna formuleras som tidsekvationer. En aktivitet skulle kunna vara "Montera Produkt A".

Aktivitetsklassifikation – Talar om aktiviteten är cykelberoende, batchberoende eller periodisk. Kommer från produktionen.

Demonstrator – Ett verktyg som används för att visualisera något. I det här projektet innebär en demonstrator en mjukvara (eller en kombination av flera mjukvaror) som visualiserar produktionsförbättringar i industrin.

Direkt aktivitet – Aktiviteter som driver materialet framåt, till exempel montering.

Element – De minsta rörelser som en operatör utför, till exempel att sträcka sig för att ta något eller att ta ett steg. Det är de minsta beståndsdelarna i aktiviteter. Elementen fås från SAM-studier och har fördefinierade namn. De matas in manuellt eller läses in från mjukvara givet SAM-ark. Grundrörelserna förklaras nedan:

- **Get** – att sträcka sig för att greppa
- **Put** – att placera på en plats
- **Step** – att förflytta sig
- **Return** – att sträcka sig för att lämna tillbaka
- **Use** – att använda, till exempel en skruvmejsel
- Tillägg till grundrörelser:
    - **Apply force** – att med kraft trycka eller lägga ner
    - **Handful** – att greppa en handfull, till exempel skruvar
    - **Precision** – att med precision placera eller utföra

Elementartidssystem – Ett system som sätter en standardtid på element. Detta motsvarar den beräknade tid som det tar för en medeloperatör att genomföra detta element. Operatören ska klara av att följa standardtiden åtta timmar om dagen, fem dagar i veckan i hela sitt arbetsliv utan att bli utsliten. MTM och SAM är exempel på elementartidssystem.

Exekvera – Köra en mjukvara.

Faktisk tid (actual time) – Den verkliga tid det tar för en operatör att genomföra en subaktivitet eller aktivitet.

Faktor – Tidsenhet som används i SAM-studier, 1 timme = 3600 s = 20 000 faktorer.

Gränssnitt – Utformningen av en viss förbindelse mellan olika objekt. Kommunikation mellan mjukvaror eller maskin och människa.

Ideal kapacitet, CAP_I – Ideal kapacitet visar hur många enheter per timme som idealt kan tillverkas vid en arbetsstation, det vill säga utan störningar eller annat som hindrar arbetaren från att arbeta. Kapaciteten baseras på den ideala tiden det tar att utföra de element som aktiviteterna i arbetsstationen består av, det bygger alltså på elementartidssystem.

Ideal tid (ideal time) – Den tid en (sub-)aktivitet ska ta enligt SAM-studie och elementartid. Ska dock intte förväxlas med optimal tid. Ideal tid utgår från medotden enligt nuvarande standard medan optimal tid utgår från den optimala metoden.

Indirekt aktivitet – Aktiviteter som krävs men som inte driver materialet framåt, tex laddning av maskin eller hämta material.

Informationsmodell – Den konceptuella modell av ett produktionssystem som tagits fram i tidigare forskning. Visar på ett generiskt sätt samband mellan den totala kapaciteten i en fabrik och aktiviteternas uppbyggnad. Se Hedman (2013) för utförlig beskrivning.

Iterera – Upprepa till önskat resultat uppnåtts.

Kompilera – Översätta kod till instruktioner för datorn.
Maskinvara – Samlingsnamn för en dators fysiska delar.
Metod – Den metod som används för tillverkningsprocessen. Det vill säga i vilken ordning aktiviteter och subaktiviter sker samt hur de är uppbyggda.
Prestationsparameter – Parameter som talar om i vilken takt jämfört med den ideala som operatören arbetar. Anges i procent. Om arbetet utförs på en kortare tid än idealt blir parametern över 100 % och om det utförs på en längre tid blir parametern under 100 %. Denna faktor påverkas endast av arbetarens motivation och fysisk förmåga. Kommer från manuell prestationsbedömning eller ideal tid jämförd med verklig tid.
Process – Aktiviteter och resurser sammankopplade där aktiviteterna är ordnade i den följd som krävs för att skapa produkter. Flera processer kan gå genom samma aktivitet.
SAM-ark – Ett Excel-ark som erhålls efter genomförd SAM-studie. Visar hur aktiviteter består av subaktiviteter och hur de i sin tur består av element. Respektive elements tid i faktorer samt antal element och subaktiviteter återfinns även här. Kan vara gjort för hand eller automatiskt genererat från en mjukvara, till exempel AviX.
SAM-studie – Ett MTM-baserat elementartidssystem. Förkortningen står för sekvensbaserad aktivitet- och metodanalys. Används för att bryta ned aktiviteter i dess minsta beståndsdelar, element, för att på så sätt kunna analysera aktiviteter. Kan endast göras om arbetet är standardiserat.
Subaktivitet – Byggsten i aktiviteterna, till exempel "Hämta Komponent A".
Tidsdrivare – Tidsdrivare är en multiplikator som används för att ange antalet gånger en
(sub-)aktivitet eller ett element utförs. Till exempel blir tidsdrivaren för skruva på stolsben på en stol fyra eftersom samma procedur utförs fyra gånger.
Tidsekvationer – Visar vilka subaktiviteter en viss aktivitet består av och vilka element en viss subaktivitet består av. Tidsekvationerna visar också antalet likadana subaktiviteter eller element, se tidsdrivare. Ekvationerna kan även visa vilka delar som en anläggning är uppbyggd av. Tidsekvationerna skrivs på formen nedan.

$$Montera\ Produkt\ A\colon Tidsdrivare1 \times Hämta\ Komponent\ A + Tidsdrivare2 \times Hämta\ Komponent\ B$$

Utnyttjandegrad (utilization) – Utnyttjandegraden visar hur arbetstiden fördelas mellan planerat produktionsarbete och förluster i förhållande till planerad produktionstid. Förlusterna kategoriseras som personlig tid, störningar och systemförluster. Denna fördelning fås från en frekvensstudie för manuella arbetsuppgifter och anges i procent (0-100 %). Den totala arbetstiden motsvaras av 100 %.
De olika kategorierna återfinns nedan:

- $\mathbf{U_N}$ – personlig tid, till exempel raster
- $\mathbf{U_S}$ – systemförluster, till exempel balanseringsförluster
- $\mathbf{U_D}$ – störningar i produktionen
- $\mathbf{U_M}$ – planerat arbete

Verklig kapacitet, $\mathbf{CAP_R}$ – Talar om det antal enheter som för tillfället produceras. Siffran ska alltså motsvara det faktiska antalet enheter som varje timme produceras. Denna kapacitet beror på hur situationen ser ut: hur väl metoden fungerar, hur väl arbetarna arbetar, hur väl maskinerna fungerar, hur organisationen ser ut samt hur produktionssystemet är designat.

3. Beskrivning

Här listas mål för demonstratorn, funktioner och andra faktorer som påverkar produkten. Det blir en brygga för att få förståelse för projektet och vart siktet är ställt på. Här fås en kortare beskrivning för vad demonstratorn skall göra för att sedan kompletteras i kravspecifikationen i följande kapitel.

3.1 Avgränsningar

Kravspecifikationen är tänkt att fungera som stöd vid utveckling av gruppens egen mjukvara och slutligen den färdiga demonstratorn. Det inte är meningen att en utomstående person, utan insikt i modeller och metoder, att läsa och dra nytta av kravspecifikationen. Kopplingar mellan kapacitet och finansiell data har inte implementerats vid projektets slutförande (maj 2014). Det pågår förvisso forskning i området, men för att implementera det i programvaran krävs ytterligare arbete i metoden och blir inte en del av arbetet.

Det listas inga krav på dataflöden eller hur programmeringstekniska problem skall lösas. I området Software Requirements finns modeller hur kravhanteringen fungerar (Lauesen 2002), eftersom det inte är ett arbete utfört av datateknologer görs det här en avgränsning för att anpassas efter projektets bakgrund på maskinteknik.

3.2 Produktperspektiv

Kravspecifikationen är grund för utformandet av en mjukvara i kandidatarbetet "*Visualiseringar av produktionsförbättringar*". Kraven kommer användas vid test och validering av den mjukvara som kandidatarbetet skall resultatera i tillsammans med slutrapport.

Målet att göra kvalitativa tester för utformningen av ett hjälpmedel till informationsmodellen för att slutligen mynna ut i en mjukvara för kommersiellt bruk. Slutmålet för demonstratorn är att få fram ett verktyg som med precision bestämmer CAP_l och CAP_r. Det skall också ingå verktyg för att visar hur mjukvaran används i form av en manual för att hjälpa nya användare. Demonstratorns uppgift är att belysa fördelarna i informationsmodellen att nedifrån-och-upp analysera fabriksmiljöer genom tidsstudier. Demonstratorn är till för att testa och implementera metoderna digitalt för att belysa förbättringspotentialen hos flera användargrupper och intressenter.

3.3 Övergripande krav

Mjukvaran skall spara data lokalt på användarens dator eller av användaren avsedd destination. När användaren sparar för första gången skall det frågas efter destination på det sparade objektet.

K3.1.1 Tider, både i sekunder och faktorer begränsas till maximalt två decimaler i utskrift, medan alla decimaler används vid beräkning.

K3.1.2 Användare skall kunna ändra i befintlig modell, lägga till, ta bort eller flytta komponenter. För både ovanliggande och underliggande nivåer i hierarki-uppbyggnad.

K3.1.3 Det skall gå att köras på operativsystemet Windows XP framåt. Den skall även vara gå att använda på alla plattformar från Windows XP och framåt.

K3.1.4 En sparad fil från en användare skall gå att öppna och använda hos en annan användare på en annan dator som uppfyller systemkraven.

K3.1.5 Skall automatiskt spara en säkerhetskopia av förändringar var 5:e minut.

K3.1.6 Vid avstängning skall användaren tillfrågas att spara modellen innan avstängning.

K3.1.7 Skall vid uppstart tillfråga användaren att öppna befintlig modell eller skapa ny.

K3.1.8 Vid uppstart skall det inte ta mer än fem sekunder till det att användaren får välja hur den skall arbeta enligt krav **K3.3.6**.

K3.1.9 Skall tillåta flera importeringar av SAM- ark till samma modell.

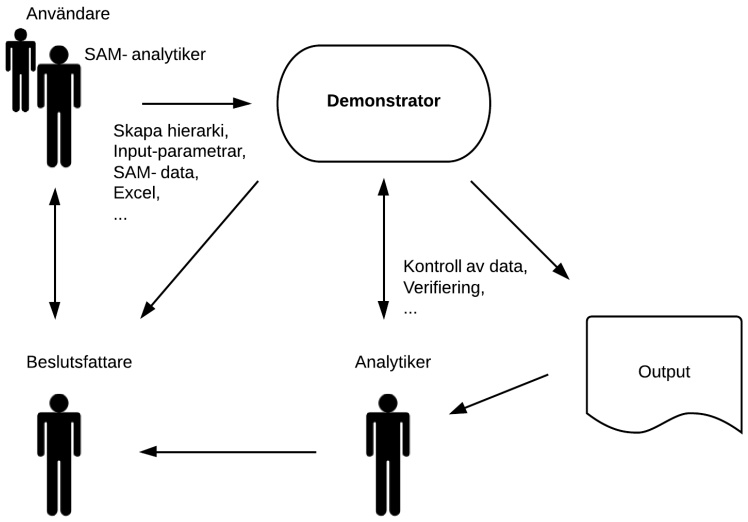
K3.1.10 Mjukvaran skall vara robust för inmatning av data och skall ha fastslagna rutiner vid händelse av felaktig syntax.

K3.1.11 Vid inmatning av olika parametrar skall det bara accepteras korrekt syntax definierat enligt metoden.

K3.1.12 Mjukvaran skall följa metoder och använda sig av termer och förkortningar från informationsmodellen (Hedman 2013).

3.4 Användarkaraktäristik

Huvudanvändaren av programmet blir gruppen själva likaså intressenter som handledare och examinator. Det förväntas att användaren är insatt i användning av informationsmodellen. Det skall gå att visa funktionerna och användandet av metoderna i mjukvaran för nybörjare och de skall få en inblick i hur modellens data länkas samman och beskriva beräkningarna i form av tidsekvationer med fler. Det skall finnas möjlighet att anpassa output från demonstratorn efter behov att redovisa för olika beslutsfattare se figur 1.



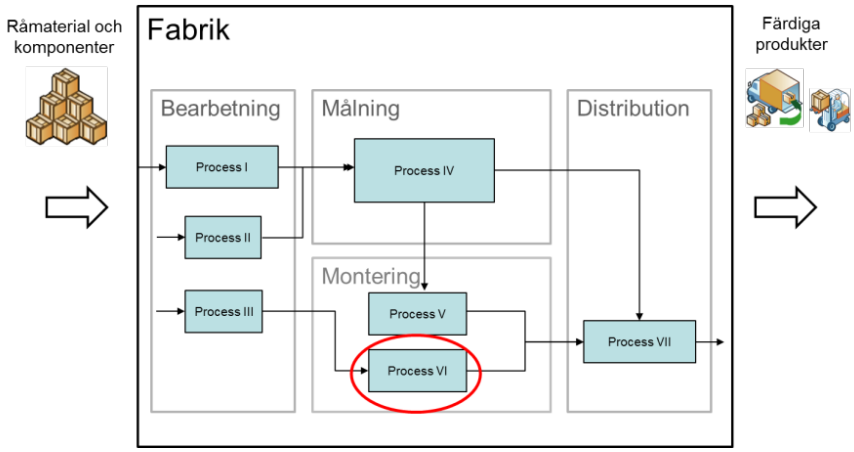
Figur 1 - Illustration av samband för informationskanaler mellan demonstrator och användare

För att behärska programmet behövs grundläggande kunskap av metoderna (Hedman 2013) innan användning. Om användaren inte har dessa kunskaper riskerar materialet att bli missvisande och det skall informeras vid felinmatning av data enligt krav **K.3.3.11**.

I metoden används verktyg som Sekvensbaserad Aktivitets- och Metodanalys (SAM) och frekvensstudier. För att kunna utföra analyserna på rätt sätt och få rätt inparametrar till mjukvaran behövs specifik utbildning och erfarenhet för att beräkningarna som står som grund skall bli korrekta. Resultatet i slutändan är högst beroende av detaljeringen nedifrån och upp för att metoderna skall behålla kredibilitet. Se informationsmodellens ursprung för bakgrund och samband i modellen från Hedman (2013).

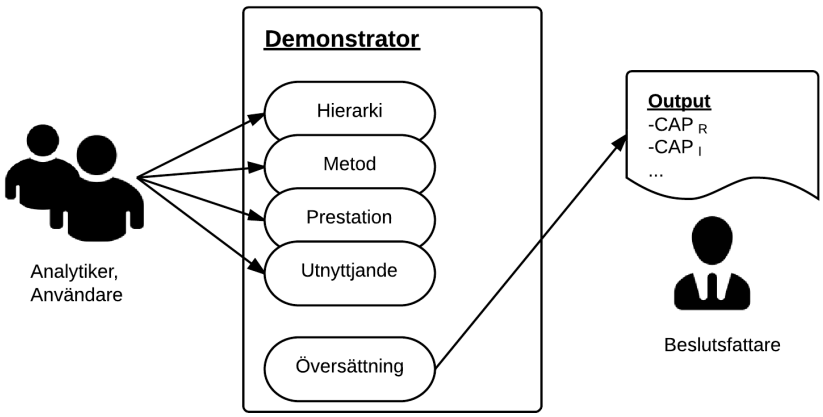
3.5 Funktionalitet

Start av programmet får användaren bygga en hierarkisk struktur av fabriken, eller av användaren önskad produktionsnivå, se **K4.1.1**. Hedman (2013) beskriver en produktionsprocess enligt figur 2. Modellen förklarar steg av produkten från råmaterial till färdig produkt utifrån från en given input. Dessa består av tillverkningsprocesser, tillverkningsresurser och fabriker. Det är upp till användaren att göra analys på process, önskad nivå och föra över det man undersökt till mjukvaran.



Figur 2 - Visar produktionsprocesser avgränsade inom avdelningar. Här process VI inom montering.

I figur 2 underliggande *Process VI* kan processens ytterligare delas in i aktivitet och sedan i flera underaktiviteter och element. Tillverkningsresurser som finns i en anläggning utgörs av människor och maskiner. Därutöver kopplar modellen tillverkningsresurser till tillverkningsprocessen då det är tillverknings-resurserna som utför aktiviteter. En fabrik kan delas in i olika undersystem som i sin tur delas in i arbetsstationer som sedan tillverkningsresurserna kan kopplas till. Modellen kan således säga beskriva att det dels finns en geografisk aspekt som beskriver var anläggningen finns och var aktiviteter utförs, men den vill även koppla resurser och de aktiviteter som resursen utför till fabriker, undersystem och arbetsstationer. Till respektive nivå matar användaren in data vilket involverar tider från metod-, prestation- samt utnyttjandegradsstudier. Dessa tre datatyper kommer dels från SAM-studier (Metodparameter), dels från manuell prestationsbedömning av resurser i systemet (Prestationsparameter) och dels data från frekvensstudier (Utnyttjandeparameter). Dessa förkortas härefter som M-, P- och U- parametrar enligt forskning (Hedman 2013).



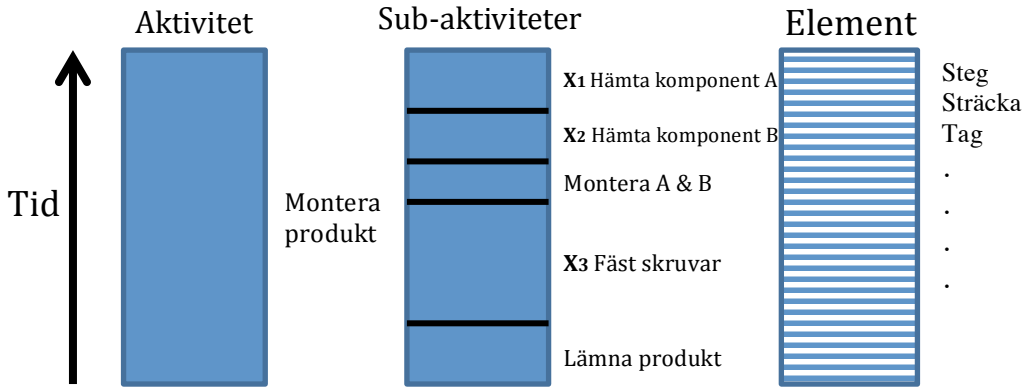
Figur 3 - Användning av demonstrator

Mjukvaran ämnas för användning enligt figur 3 där analytiker och användare utför SAM-analys på ett produktionssystem och för in dess data i demonstratorn enligt krav **K.4.1.1** Mjukvaran skall översätta och beräkna för att få önskad output enligt krav. Skall formulera tidsekvationer för de olika arbetsstationerna från SAM-studien. En tidsekvation är en summering av det som behövs för att utföra exempelvis en aktivitet. Om man skall göra en tidsekvation för att dricka kaffe skulle den kunna se ut enligt exempel att dricka kaffe:

$$\text{DrickaKaffe} = 1\ x\ \textit{Sträcka sig efter koppen} + 1\ x\ \textit{Föra koppen till munnen} + \textit{Antal klunkar}\ x\ \text{Dricka kaffe} + 1\ x\ \textit{Sätta ned koppen}$$

Dricka kaffe är aktiviteten som består av subaktiviteter som ex. dricka kaffe och sträcka sig efter koppen. Dessa består i sin tur av element som var och ett har förutbestämda tider för mänskliga rörelser (SAM element). Dessa förklarar alla de rörelser som ingår i aktiviteten. Element adderas för att skapa subaktiviteter, och subaktiviteter adderas upp till aktiviteter enligt figur 4. Genom att göra en SAM- analys är det lättare att identifiera onödiga rörelser. Man får därigenom resurser för att arbeta ut en förbättringsplan när man väl analyserat vad som behövs för att här dricka kaffet. Antal klunkar reglerar hur mycket kaffe som dricks per gång testpersonen dricker per kopplyft. Nu är det inte meningen att förbättra sättet man just dricker kaffe. Ser till hur kaffe köps, hur man bryggt det och allt kring huvudaktiviteten går det med säkerhet att hitta onödiga rörelser som kan standardiseras och förbättras.

Det är tidsekvationerna som bygger upp M-parametern i metoden. Genom att härleda dessa fås nyttorna som att analysera grundligt respektive aktivitet egentligen har för aktiviteter. Användaren skall ha valmöjlighet att härleda tidsekvationer stegvis genom systemet som är skapat. Exempelvis från aktivitet ned till tidselement.



Figur 4 - Summering av element och subaktivitet till aktivitet

Det är upp till användaren av programmet att strukturera indelningen av subsystem och aktiviteter för att återge en verklig bild av produktionen. Efter att det är gjort fås möjlighet att spara ned förändringar av produktionsparametrar och fabriksuppbyggnaden i olika versioner enligt krav **K4.1.22**. På så sätt kan man dels se dagslägets kapacitet men också testa sig fram med förändringar i M-parametern och spåra vilka utslag det skulle ge på ideal kapacitet i jämförelse med den faktiska kapaciteten i dagsläget. Att användaren kan experimentera fram förändringsförslag på exempelvis en monteringsstation ger enorm nytta. Att digitalt testa och experimentera med ändringar i montörens rörelser på elementnivå kan ligga som motiv för att förändra arbetssättet. Vare sig det gäller nya verktyg eller få bort onödiga rörelser i en monteringsstation är tidsekvationerna effektiva. Dels att räkna ut tiden för arbetet

samtidigt som man som analytiker får en bild av möjliga förbättringar, gäller både industriarbete bottom-up eller kaffedrickande.

Demonstratorn skall ge värden på tidigare nämnda metod, prestation och utnyttjandegrad, som förändras när given indata förändras. Mjukvaran skall visualisera hur systemet är uppbyggt samt hur utnyttjandegrader ser ut på olika nivåer. Det är upp till användaren att själv dra slutsatser, men programmet ska vara ett hjälpmedel för att peka ut vilka delar som med fördel kan förändras och med vilken effekt.

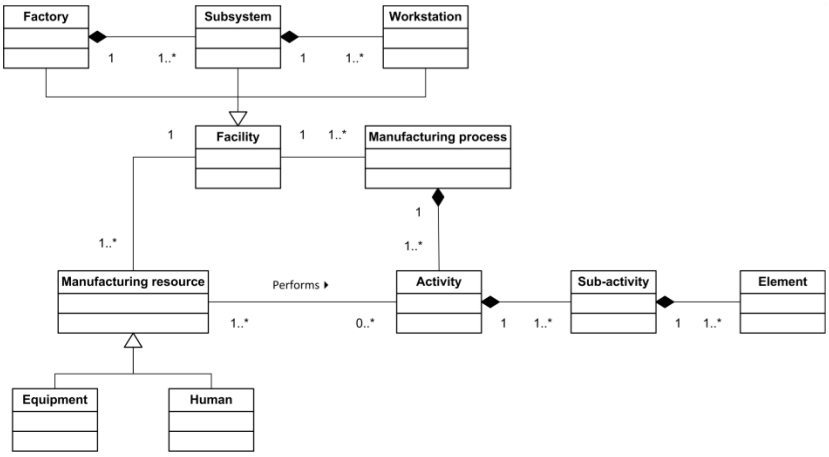
4. Kravspecifikation

Kraven på mjukvaran har tagits fram genom arbete med forskare och utvecklare av modellen. De listas beroende på vad det är för typ av krav.
Materialet har granskats och utifrån metoder och material från ”Software Requirements: Styles & Techniques” av Lauesen (2002).

4.1 Funktionalitet och uppbyggnad av modell

K.4.1.1	Skall låta användaren bygga systemlayout där användaren skapar fabriksnivåer och aktiviteter. Användaren ska själv välja hur omfattande modellen skall bli och vilka systemgränser det är som gäller. Det ska vara hierarkisk uppbyggnad där det tydligt ska framgå vilken/vilka aktiviteter och subaktiviteter som ingår i de olika nivåerna. Modellen ska synas i takt med att det byggs ut och efteråt för redigering och felsökning. Se bilaga 1 – ”*Process: skapa fabrikshierarki*” för beskrivning med hjälp av algoritm.
Syfte:	Att användaren skall själv bygga systemet som skall användas i programmet
Utlösare:	Öppnar programmet och väljer ”Skapa ny”
Förutsättning:	Att användaren har eller skall utföra en SAM- och frekvensanalys över ett ex. tillverkningssystem (eller uppskattar tider och förstår innebörden av det).
Frekvens:	Varje gång ny analys över system skall göras, nytt system skapas oftast en gång per subsystem/process.
Handling:	Skapa anläggning. Lägga till fabrik. Lägga till subsystem. Lägga till arbetsstation. Lägga till aktivitet. Lägga till subaktivitet. Lägga till standard- element.
Varianter:	1b: Ladda befintligt system. 5b: Ange ideal tid istället för att använda subaktiviteter. 6b: Ange ideal tid istället för att använda element 7b: Definiera element som inte är med i mjukvaran.

K.4.1.2. Vid uppbyggnad av modellen skall det finnas en undergrupp till aktivitet som kallas subaktivitet. Flera subaktiviteter bildar aktiviteter. Varje subaktivitet ska bestå av SAM- element eller en förutbestämd tid som ska användas för beräkningar (se **K.4.1.11**).



Figur 5 - Modell av produktionssystem

K.4.1.3. Skall kunna ange data för en subaktivitet eller aktivitet så som: verklig tid, ideal tid, typ, direkt eller indirekt aktivitet.

K.4.1.4. Skall kunna ange resurser så som operatörer och maskiner. Operatör tilldelas namn, arbetsstation och prestationsförmåga per arbetsstation. Maskiner tilldelas namn, aktiviteter möjliga att utföra, stoppdata per aktivitet enligt figur 5.

K.4.1.5. Skall kunna definiera ideal tid utifrån en SAM-analys. När SAM-analys inte är möjlig att genomföra skall en ideal tid för en subaktivitet eller aktivitet kunna definieras manuellt.

K.4.1.6. Skall kunna definiera enskild arbetsstation som flaskhals hos en produktfamilj som begränsar beräkning av CAP_i hos produkten. Det är upp till användaren, och inte mjukvaran, att göra denna definiering.

K.4.1.7. Skall kunna definiera variabel för tidsdrivare i modellen. Tidsdrivare skall kunna knytas till aktiviteter och subaktiviteter. Skall kunna ändra egenskaperna för tidsdrivare i efterhand så som namn och upprepning.

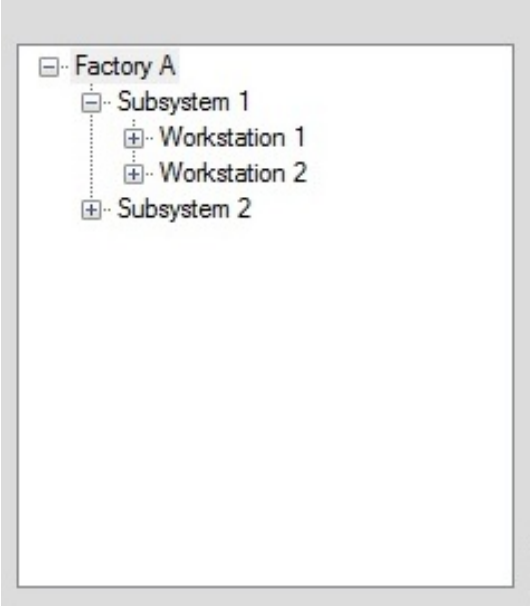
K.4.1.8. Skall visa koppling mellan subaktivitet och dess SAM- element. När användaren markerar eller dubbelklickar på subaktiviteten skall en lista med dess element visas tillsammans med dess tider.

K.4.1.9. Skall kunna använda förutbestämda SAM- element för att bygga tidsekvationer för subaktiviteter. Element har tidsenheten ”faktor” där en faktor översätts som: $1\ [faktor] = \frac{3600}{20\,000}\ [s]$.

K.4.1.10. Skall generera tidsekvationer baserat på aktiviteter och subaktiviteter. Summering av antalet handlingar multiplicerat med upprepningar för respektive aktivitet eller subaktivitet. Därefter summeras antalet aktiviteter i en arbetsstation också som egen tidsekvation.

K.4.1.11. Möjlighet att definiera egna element när det inte finns standardelement som passar. Det är upp till användaren att välja hur det skall gå till, antingen att själv definiera ett element och hur lång tid det tar att utföra och vilken enhet den skall ha (faktor eller sekunder).

K.4.1.12. Skall ha möjlighet att välja vad som skall visas och inte i systemets olika nivåer, kunna välja att gömma subaktiviteter i ett fall och i ett annat välja att visa alla nivåer ned till elementnivå. Se figur 6.



Figur 6 - Skalning av fabriksnivåer

K.4.1.13. Skall kunna ange prestationsparameter per arbetsstation[1] eller aktivitet som procent. Värdet på P-parametern som tillhör aktiviteten skall visas och vara spårbart. Tilldelas inte ett P-värde på en aktivitet skall värdet för arbetsstationens användas, dvs. värdet i den högre nivån i hierarkin. Prestationsparametern skall användas vid uträkning av kapacitet.

K.4.1.14. Om det ligger flera prestationsmått per aktivitet (ett i själva aktiviteten och ett i subsystemet är ett exempel) i olika nivåer skall det lägsta värdet i hierarkin användas för beräkning av lägstanivån. Saknas värde tas värde från högre nivå vid beräkning och ger ett felmeddelande där det:

- Säger att det saknas värde
- Värde för beräkning tas från respektive nivå

K.4.1.15. Skall kunna ange utnyttjandegrad för en process.

K.4.1.16. Utnyttjandegrad (U-parametern) skall anges som: “need based-”, “system designed-”, och “disturbance affected-” utilisation rate. Dessa skall visas som paj- diagram tillsammans med värdeadderande tid med färgkodning (värdeadderande-grönt, U_N-orange, U_S-gult och U_D-rött). Utnyttjandegrad skall användas vid uträkning av kapacitet.

K.4.1.17. Skall finnas möjlighet att användaren själv ska välja var utnyttjandegraden matas in. Inmatad utnyttjandegrad skall sättas som defaultvärde för lägre nivåer i hierarkin.

K.4.1.18. Skall visa kopplingar mellan aktiviteter och materialflöden för ett subsystem, skall visa ett precedensdiagram där det skall framgå ordningen på aktiviteter och materialflödet genom subsystem, arbetsstation eller fabrik. Användaren skall kunna göra indelning baserat på direkt eller indirekta aktiviteter

K.4.1.19. Mjukvaran skall utföra beräkningar och operationer enligt modellen (Hedman 2013). Om det uppstår fel i beräkningar eller beräkningar inte går att utföra skall användaren få felmeddelande.

[1] En avgränsning som görs till demonstratorn i kandidatarbetet, för slutprodukt görs det per resurs.

K.4.1.20. Skall beräkna M-Parametern [enh/h] per aktivitet baserat på underliggande subaktiviteter. Addering av subaktiviteter multiplicerat med tidsdrivare skall ge aktivitetens tid. På samma sätt skall addering av aktiviteters tid och tidsdrivare per aktivitet ge processens tid enligt figur 4.

K.4.1.21. Skall beräkna CAP_I baserat på tider för underliggande system. CAP_R skall beräknas från M-, P- och U-parametrarna. Skall kunna sparas undan som [%] av CAP_I och som [enh/h]. Skall visa ideal och verklig kapacitet per arbetsstation vid markering. Skall kunna expandera för att visa aktiviteterna för en arbetsstation. Ideal kapacitet skall kunna visas på följande sätt:

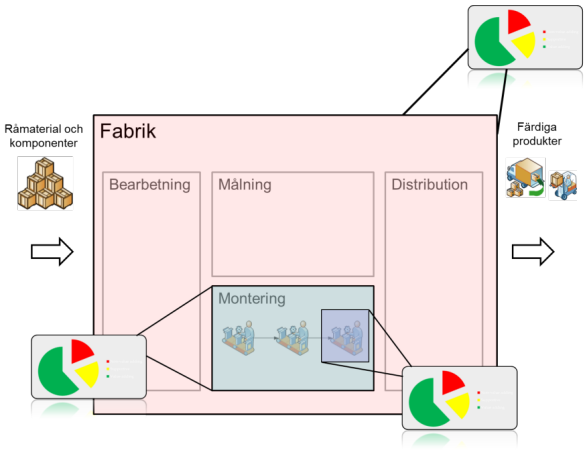
K.4.1.22. Skall kunna spara användardata med fabriksstrukturen med möjlighet att välja namn och köra analys av flera olika modeller samtidigt. Skall kunna öppna upp och fortsätta analysera/ arbeta med modellen vid senare tillfälle.

K.4.1.23. Bör kunna importera filmfiler för uppspelning. Skall kunna pausa, stoppa starta filmen och ändra uppspelningshastighet. Samtidigt som filmen spelas skall man kunna bygga upp hierarkin och lägga till verkliga tider enligt krav K4.1.23. De filformat som filmspelaren skall stödja skall inkludera avi, mp4 och wmv.

K.4.1.24. Verklig tid skall kunna definieras genom att en film studeras där programmet automatiskt ska kunna skicka över tiden för en viss aktivitet eller underaktivitet.

K.4.1.25. Skall kunna exportera data från programmet till dokument läsbara för både Windows och Mac- användare. Skall exportera filer i format så som PDF, .docx, .jpeg och .ppt m.fl. Vid val av exportering skall användaren tillfrågas vilken data som skall exporteras och till vilken typ av dokument.

K.4.1.26. Skall kunna exportera grafer på nyckeltal så som CAP_I, CAP_R och produktionsparametrar per process och per produktionsgrupp. Skall ha möjlighet att exportera CAP_I och CAP_R i de fabriksnivåer som ligger över aktivitet i fabrikshierarkin. Skall kunna få ut $CAP_{I/R}$ för system (se figur 5 för prioritering).



Figur 7 - Prioritering av parametrar

K.4.1.27. Skall kunna exportera U-parametern som pajdiagram och siffervärde för arbetsstation, subsystem och fabrik. (se figur 7 för prioritering).

K.4.1.28. Skall kunna exportera tidsekvationer för alla steg upp till och med aktivitet i hierarkin samt processer. Ekvationerna skall redovisas som "*Tidsdrivare x subaktivitet + Tidsdrivare2 x subaktivitet*". Skall kunna ses som följd i en process.

K.4.1.29. Skall kunna exportera P-parametern i alla nivåer ovan aktivitet eller den nivå det först anges. Skall exporteras som ett diagram där procentsatsen jämförs mot 1 se figur 7 för prioritering av parametrar.

K.4.1.30. Skall kunna exportera listor på antalet upprepningar av element i respektive subaktivitet. Skall exporteras som lista med möjlighet till val att också exportera som diagram av antalet upprepningar.

K.4.1.31. Skall kunna exportera listor på antalet upprepningar av subaktiviteter per aktivitet. Skall exporteras som lista med möjlighet till val att också exportera som diagram av antalet upprepningar.

K.4.1.32. Skall kunna exportera listor på antalet upprepningar av aktiviteter i respektive process. Skall exporteras som lista med möjlighet till val att också exportera som diagram av antalet upprepningar.

K.4.1.33. Skall kunna exportera precedensdiagram för de nivåer där användaren gjort en aktivitetskoppling och nivåkopplingar i systemstrukturen.

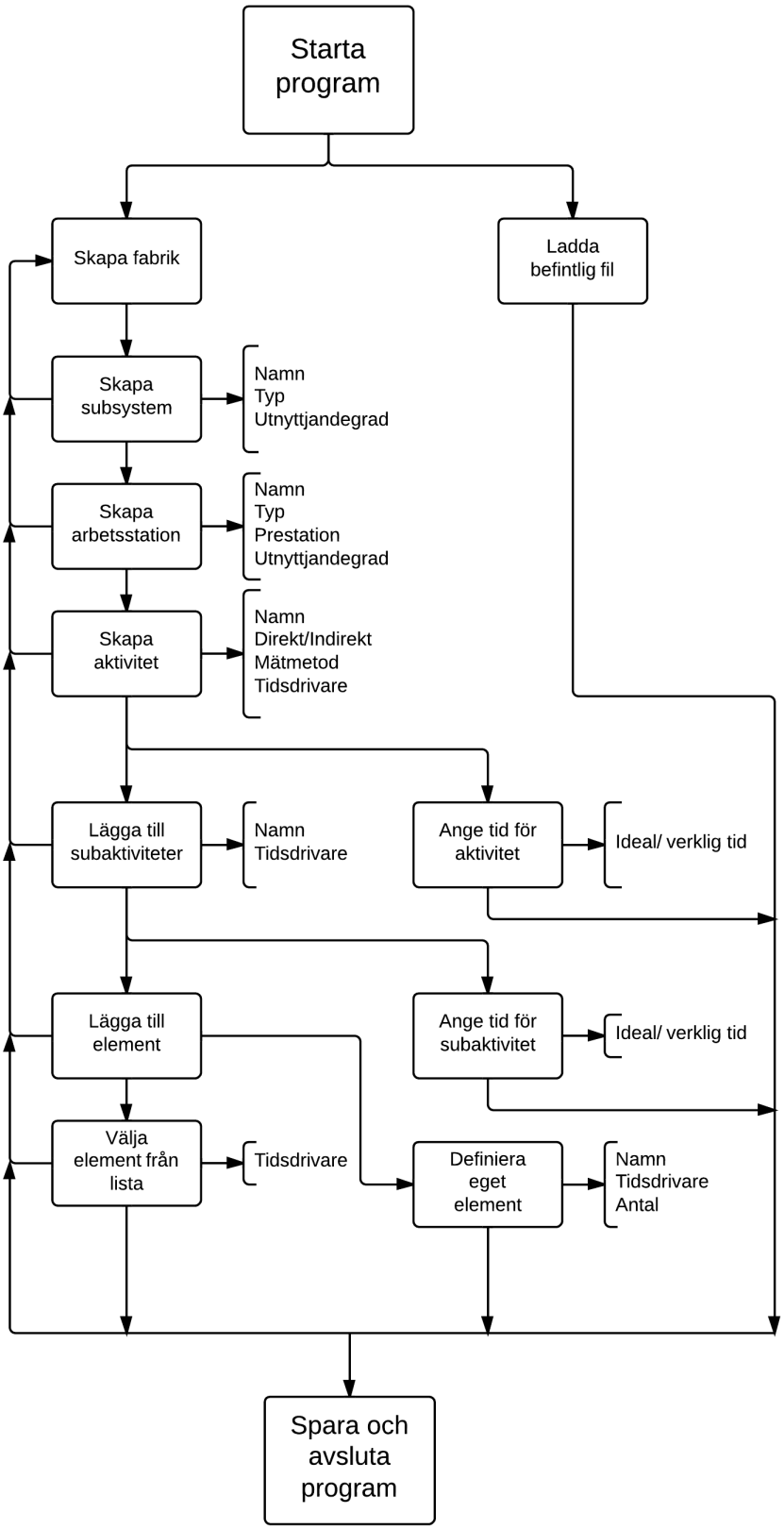
K.4.1.34. Skall kunna exportera den hierarkiska strukturen på fabriken från SAM-ark. Det skall framgå vilken nivå det är och vilka som är underliggande moment i strukturen.

K.4.1.35. Skall kunna definiera om en verklig tid för en aktivitet tagits fram genom klockning eller uppskattning.

K.4.1.36. Skall kunna skapa tillverkningsprocesser genom att välja ingående aktiviteter i rätt ordning. Till varje aktivitet skall en resurs med tillhörande egenskaper kopplas.

K.4.1.37. Skall kunna spara modellen av produktionssystemet enligt ISO 15531 MANDATE.

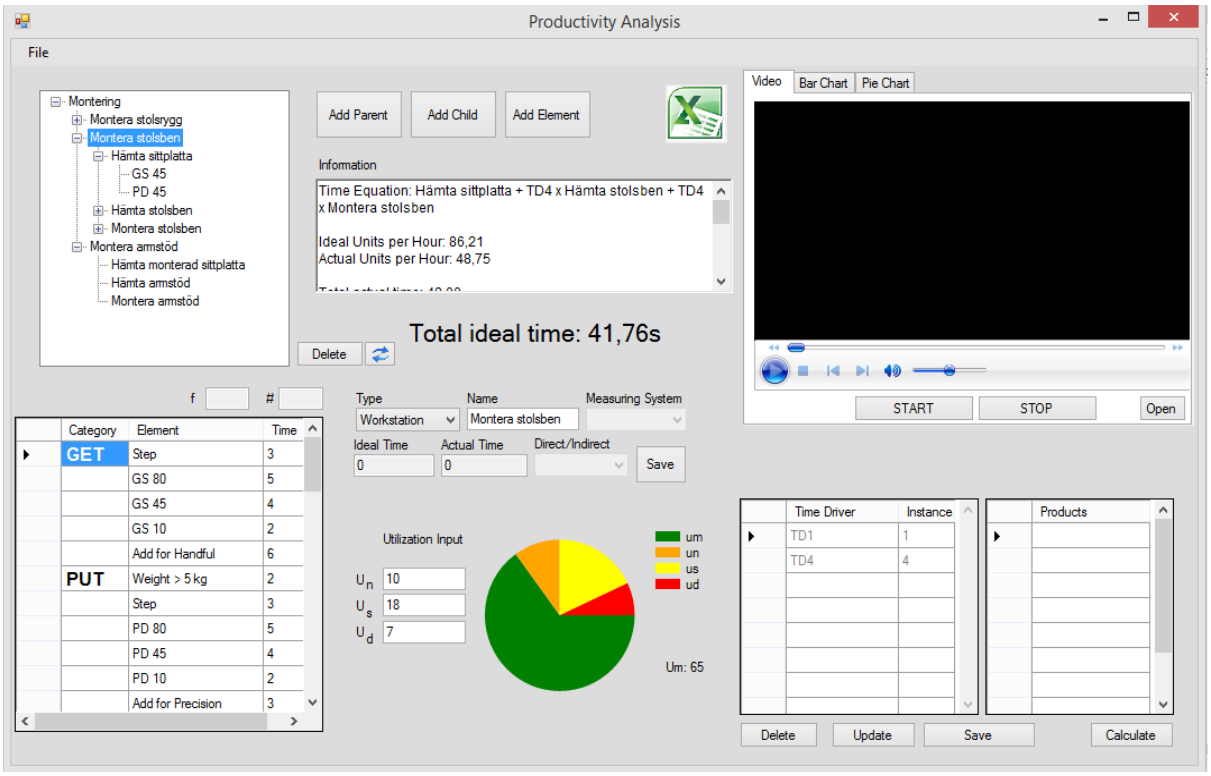
Bilaga 1 – Process: skapa fabrikshierarki



Bilaga C – Manual C#

Manual

ACE - prototyp i C#



INNEHÅLLSFÖRTECKNING

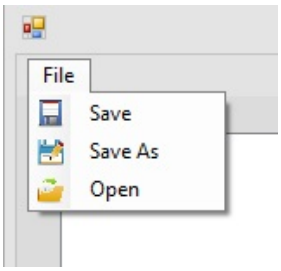
FÖRORD

Denna manual ska hjälpa dig som användare att få ut så mycket som möjligt av prototypen. Då prototypen är på engelska och manualen på svenska finns det därför nedan en ordlista över de ord som används i prototypen respektive manualen.

Prototypen	Manualen
Facility	Anläggning
Factory	Fabrik
Subsystem	Subsystem
Workstation	Arbetsstation
Activity	Aktivitet
Subactivity	Subaktivitet
Element	Element
Actual time	Faktisk tid
Ideal time	Ideal tid
Utilization	Utnyttjandegrad

För förklaring av ordens innebörd se slutrapporten "Visualisering av produktionsförbättringar".

1. SPARA OCH ÖPPNA



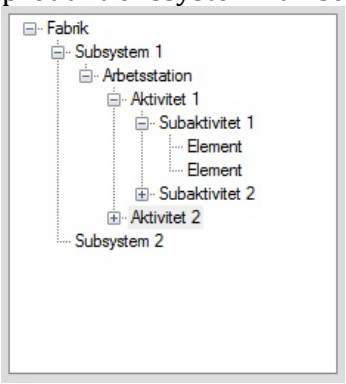
Figur 1. File, där Save, Save As och Open kan hittas.

I övre vänstra hörnet under fliken "File" finns det möjlighet att spara data och öppna tidigare sparad data, se figur 1.
För att spara med ett nytt namn används "Save as". Det kommer då upp en ruta.
Där väljer du den sökväg där filen ska sparas, det önskade filnamnet fylls i och sedan trycker du på knappen "Spara".
För att spara över den senast sparade filen används "Save" under fliken "File".
För att öppna sparad data trycker du på "Open" under fliken "File". En dialogruta dyker upp där önskad fil väljs innan du trycker på "Öppna" (eller "Open") i dialogrutan.
Alla ovan nämnda operationer kan göras vid valfritt tillfälle under användning av prototypen.

2. BYGGA UPP HIERARKI

3.1 Lägga till

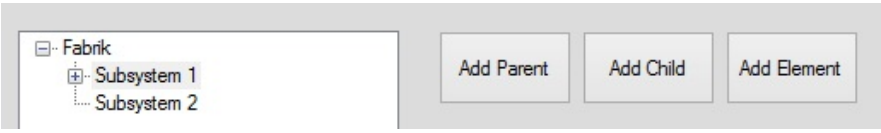
Det är viktigt att vara medveten om i vilken ordning hierarkin är tänkt att byggas upp för att strukturen på fabriken ska bli korrekt. Ett exempel på hur en hierarki för ett produktionssystem kan se ut ses på figur 2 nedan.



Figur 2. Exempelhierarki.

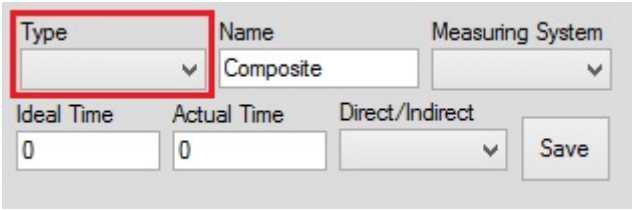
Du som användare har friheten att själv välja vilka delar av hierarkin som ska byggas upp samt möjligheten att i efterhand ändra detta. Det finns dock några begränsningar; en aktivitet består av en eller flera subaktiviteter som i sin tur består av ett eller flera element. Det är däremot inte ett krav att lägga till alla tre typerna av anläggning; prototypen fungerar lika bra utan till exempel subsystem och fabrik. Dock kan det underlätta förståelsen för hur de olika delarna hänger ihop om alla beståndsdelar i hierarkin finns med.

3.1.1 Anläggning



Figur 3. De tre knapparna Add parent, Add child och Add element.

1. Klicka på knappen ”Add Parent” som kan ses i figur 3.. En första gren i trädet till vänster kommer då skapas, med namnet ”Composite”.



Figur 4. Val av typ.

2. Markera anläggningen i trädet till vänster och välj typ i rullistan, som är markerad med rött i figur 4. Det som finns att välja på är: fabrik, subsystem och arbetsstation. Prototypen tar inte hänsyn till vilket val som görs utan du som användare måste se till att anläggningarna skapas i rätt ordning.
3. Bredvid val av anläggning skrivs, istället för “Composite”, önskat namn in. Inget annat än typ och namn kan ändras med en anläggning.
4. Tryck på ”Save”.

Det är möjligt att lägga till fler än en anläggning. Om fler anläggningar på samma nivå som den redan skapade ska läggas till ska steg 1-4 ovan följas. Om däremot en underrubrik till en redan existerande anläggning ska skapas, till exempel ett subsystem under en fabrik, blir tillvägagångssättet något annorlunda:

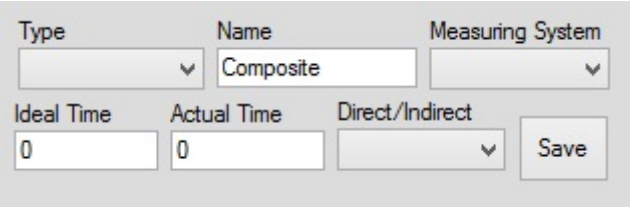
a. Markera den önskade anläggningen i trädet till vänster som den nya anläggningen ska hamna under.
b. Klicka på knappen ”Add Child”, som kan ses i figur 4. Då skapas en underrubrik till den markerade anläggningen.
c. Följ sedan steg 2-4 ovan.

3.1.2 (Sub-)aktivitet

Om ett Excel-ark kompatibelt med prototypen innehas och önskas läsas in, var vänlig och hoppa då ned till rubriken “Inläsning från Excel”.

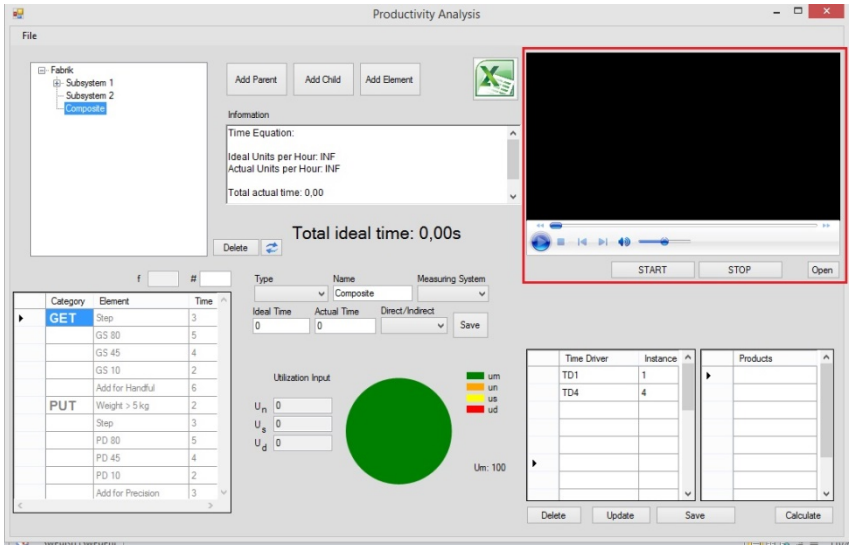
Om ett kompatibelt Excel-ark inte innehas, vänligen följ stegen nedan.

1. För aktivitet: markera önskad anläggning som aktiviteten ska ingå i.
   För subaktivitet: markera önskad aktivitet som subaktiviteten ska ingå i.
2. Klicka på knappen ”Add child”. En ny gren på lägre nivå i trädet skapas.



Figur 5. De egenskaper som ska väljas och fyllas i för respektive (sub-)aktivitet.

3. Välj typ i rullistan - subaktivitet eller aktivitet.
4. Fyll i valfritt namn på (sub-)aktiviteten, istället för “Composite”.
5. Välj den mätmetod som använts för att ta fram den faktiska tiden, SWAG eller stoppklocka.
6. Om ideal tid redan är känd fyll då i tiden i rutan märkt “Ideal time”. Om (sub-)aktiviteten är uppbyggd av element, och den ideala tiden därmed inte är given, hoppa då över detta steg (steg 6) och gå till nästa.
7. Då den faktiska tiden ska fyllas i finns det två alternativ.
   a. Om tiden tagits fram via SWAG eller stoppklocka fyll då i tiden i rutan märkt “Actual time” (observera att tiden ska anges i siffror och inte i bokstäver!).



Figur 6. Filmspelaren makrerad i rött i övre högra hörnet.

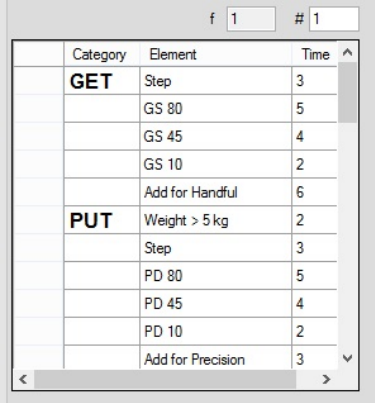
b. Tiden tas fram genom att använda filmspelaren, som kan ses i figur 6.
   i. Börja med att trycka på “Open” under filmspelaren.
   ii. En dialogruta kommer då upp där du kan leta upp den önskade filmen och trycka på “Öppna” (eller “Open”) i dialogrutan.
   iii. När filmen finns i filmspelaren trycker du på Play-knappen i filmspelaren.
   iv. När önskad (sub-)aktivitet startar trycker du på “Start” under filmspelaren och ”Stop” då (sub-)aktiviteten är slut.
   v. Tiden för (sub-)aktiviteten läggs då automatiskt in i rutan märkt “Ideal time”.

8. Välj om (sub-)aktiviteten är direkt eller indirekt.
9. Om (sub-)aktiviteten upprepas se rubriken ”Tidsdrivare” nedan.
10. Tryck på knappen ”Save”.

3.1.3 Element

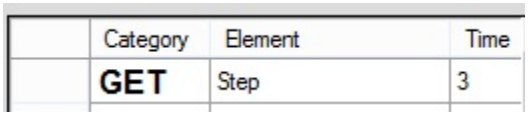
Om elementen finns definierade i ett Excelark kompatibelt med prototypen som önskas läsas in, vänligen se rubriken “Inläsning från Excel” nedan. Om ett sådant dokument inte finns, vänligen följ stegen nedan.

1. Markera önskad subaktivitet som elementen tillhör. Observera att subaktiviteter, och inte aktiviteter, byggs upp av element. Den närmaste nivån över element i hierarkin måste därför vara en subaktivitet.
2. Tryck på knappen ”Add Element”. Detta kommer lägga till ett element med namn “Element”. Det är inte möjligt att addera underrubriker till dessa delar i trädet eftersom element är den minsta beståndsdelen.
3. Då tiden för elementet ska fyllas i finns två alternativ.



Figur 7. Elementlistan.

a. Välja ett befintligt element. Typen av element väljs i listan i nedre vänstra hörnet, som kan ses i figur 7, genom att först markera elementet i trädet och sedan markera önskad elementtyp i listan. Detta görs genom att trycka på den vita rutan till vänster om elementtypen och sedan ska antalet likadana element fyllas i i rutan markerad med "#".
b. Lägga till eget element. Om elementet som ska läggas till inte finns i den befintliga listan går det bra att lägga till ett eget.



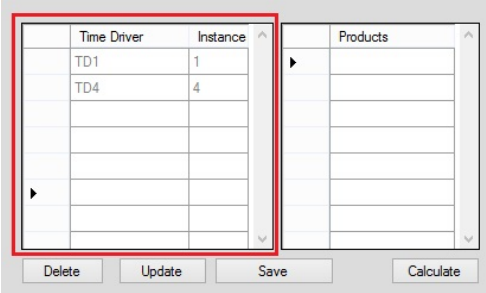
Figur 8. Närbild på elementlistan.

i. Under rubriken "Element" i listan i nedre vänstra hörnet fyller du i namnet på elementet, se figur 8.
ii. Under rubriken "Time" fylls tiden för elementet i. Observera att tiden ska anges i faktorer, vilket är samma enhet som används för de fördefinierade elementen.
iii. I rutan märkt "#" som finns ovanför listan fylls antalet element i.
iv. I rutan märkt "f" står det från början 1. Denna siffra ska endast ändras om det element som läggs till är ett element som innefattar någon sorts skruvande. Rutan märkt "f" ska då innehålla antalet grepp som tas om till exempel skruvmejseln.

4. Tryck på knappen "Save".

3.1.4 Tidsdrivare

Tidsdrivare är en multiplikator som används för att ange antalet gånger en (sub-)aktivitet utförs. Till exempel blir tidsdrivaren för skruva på stolsben på en stol fyra eftersom samma procedur utförs fyra gånger. För att lägga till en tidsdrivare, följ stegen nedan.



Figur 9. Listan för tidsdrivare markerat i rött.

1. Välj en tom rad i listan i prototypens nederdel som kan ses i figur 9.
2. Ange namn under rubriken ”Time Driver” i listan nere till höger.
3. Ange antalet tidsdrivare under rubriken ”Instances” i samma lista. Observera att antalet ska anges i siffror och inte i bokstäver.

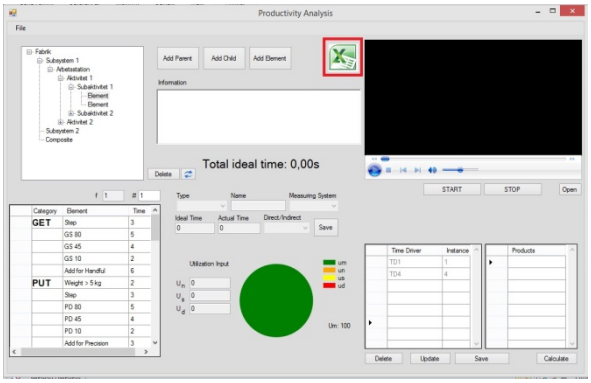
För att koppla en tidsdrivare till en (sub-)aktivitet, följ stegen nedan. Observera att samma tidsdrivare kan användas till flera aktiviteter, men att det bara är möjligt att lägga till en tidsdrivare till en (sub-)aktivitet i taget.

a. Markera den (sub-)aktivitet i trädet som ska ha en tidsdrivare.
b. Önskad tidsdrivare markeras i listan i prototypens nederdel genom att trycka på den tomma rutan till vänster om tidsdrivaren.
c. Tryck på knappen “Save”.

3.1.5 Inläsning från Excel-ark

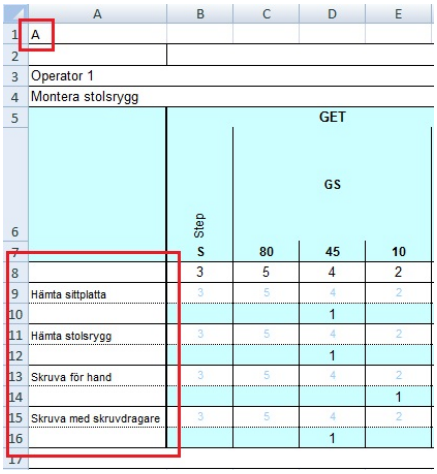
Om ett standardiserat SAM-ark som är kompatibelt med prototypen innehas går det bra att läsa in detta ark. Om ett SAM-ark genererat från AviX innehas krävs en del modifieringar för att prototypen ska kunna läsa in det, se steg 1-4 nedan.

Då SAM-arket endast innehåller element och subaktiviteter måste först anläggning, subsystem, arbetsstationer och aktiviteter skapas i trädet, med valfriheten att inte lägga till alla nyss nämnda delar. Detta görs enligt 3.1.1-3.1.2 ovan.



Figur 10. Inläsning från Excel sker med hjälp av den markerade knappen.

Då trädet är skapat trycker du på knappen med Excel-ikonen, se figur 10, som genomför inläsningen av all information till programmet.
Programmet kommer då skapa ett träd utifrån den informationen som finns i Excel-arket.



	A	B	C	D	E
1	A				
2					
3	Operator 1				
4	Montera stolsrygg				
5				GET	
6		Step		GS	
7		S	80	45	10
8		3	5	4	2
9	Hämta sittplatta	3	5	4	2
10				1	
11	Hämta stolsrygg	3	5	4	2
12				1	
13	Skruva för hand	3	5	4	2
14					1
15	Skruva med skruvdragare	3	5	4	2
16				1	
17					

Figur 11. Ett Excel ark från AviX ska modifieras till detta utseende.

1. I Excel-dokumentet från AviX kommer det finnas en flik för varje aktivitet. Innehållet på dessa flikar måste läggas in i var sitt dokument eftersom programmet inte klarar av att läsa in flikar. Varje dokument ska se ut enligt figur 11.
2. Kolumnerna till vänster ska raderas till dess att subaktiviteternas namn står i första kolumnen i dokumentet, som i figur 11.
3. Rad 1 ska vara tom, men det ska vara den enda tomma raden ovantill, se figur 11.
4. I övre vänstra hörnet, i cell A1, ska du skriva “A”, utan citationstecken. Detta görs för att programmet ska förstå att dokumentet kommer från AviX.
5. Slutligen måste eventuella inringade siffror i Excel-dokumentet manuellt ersättas med nollor. Att en siffra i SAM-dokumentet är inringad innebär att tiden för detta element inte ska läggas till den totala tiden för subaktiviteten då elementet utförs samtidigt som ett annat element. För att programmet inte ska lägga till denna tid måste därför siffrorna ersättas med nollor. Ringarna kring siffrorna behöver ej tas bort.

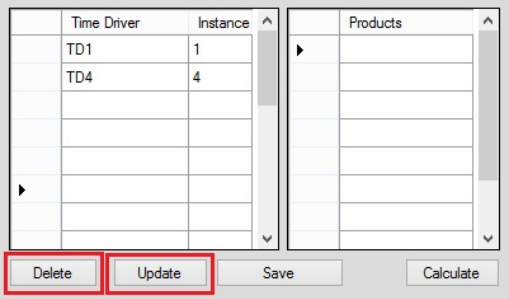
3.2 Ändra/ta bort

2.2.1 Hierarkin

För att ta bort en gren i trädet markeras önskad gren och sedan trycker du på knappen ”Delete” till höger under rutan med hierarkin. Observera att alla underrubriker som tillhör den grenen också kommer att tas bort.
För att ändra någon del i strukturen markeras den önskade delen i trädet innan önskad ruta eller rad i någon av rullisterna ändras. Tryck sedan på knappen “Save”.
För att ändra elementtyp till något annat av alternativen i listan i programmets nedre vänstra hörn markeras först önskat element att förändra i trädet och sedan markeras önskad elementtyp att byta till. Tryck sedan på knappen ”Save”.

3.1.1 Tidsdrivare

För att ta bort en tidsdrivare från en (sub-)aktivitet markeras först önskad (sub-)aktivitet i trädet och sedan trycker du på knappen “Delete” under rutan för tidsdrivare, se figur 12.

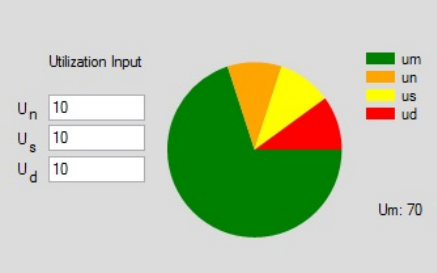


Figur 12. Knapparna som används då en tidsdrivare ska raderas eller uppdateras.

För att ändra en redan existerande tidsdrivare markeras önskad tidsdrivare innan namn eller antal ändras i listan, precis på samma sätt som då tidsdrivaren skapades. Då ändringen är genomförd trycker du på knappen "Update", se figur 12. I och med denna knapptryckning kommer tidsdrivaren ändras för alla (sub-)aktiviteter den är kopplad till.
Det är även möjligt att ändra flera tidsdrivare samtidigt. Börja med att ändra namn eller antal på de aktuella tidsdrivarna. Tryck Ctrl och markera sedan alla tidsdrivarna som ändrats och tryck sedan på "Update".

3. UTNYTTJANDEGRAD

Utnyttjandegraden visar hur arbetstiden fördelas mellan värdeadderande arbete och förluster i form av personlig tid, störningar och systemförluster. Denna fördelning fås från en frekvensstudie för manuella arbetsuppgifter. Då frekvensstudien gjorts och utnyttjandegraden ska fyllas i, vänligen följ stegen nedan. Observera att utnyttjandegraden endast fylls i på anläggningsnivå, det vill säga utnyttjandegraden kan endast mätas på nivåerna arbetsstation, subsystem och fabrik.



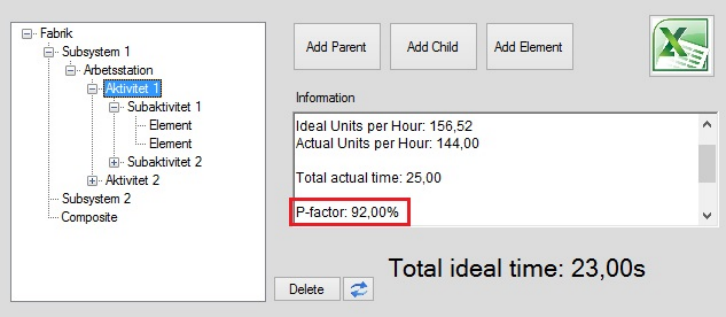
Figur 13. Diagram som visar utnyttjandegraden, samt de rutor där utnyttjandegraden matas in.

1. Markera önskad anläggning i trädet. Den anläggning som markeras ska motsvara den nivå som frekvensstudien gjordes på. Om frekvensstudien gjordes för en arbetsstation ska utnyttjandegraden också fyllas i på arbetsstationsnivå.
2. Fyll i värdena för de tre förlusterna i respektive ruta där Un är personlig tid, Us är systemförluster och Ud är störningar.
3. Tryck på "Save".

Om frekvensstudier gjorts på olika nivåer, till exempel på en arbetsstation samt på hela subsystemet, ska utnyttjandegraden matas in på respektive nivå. De arbetsstationer där separata frekvensstudier inte genomförts ska få samma utnyttjandegrad som nivån ovanför, i det här fallet subsystemet. Detta gör programmet i dagsläget inte automatiskt utan du som användare måste själv fylla i rätt utnyttjandegrad på rätt nivå.

4. P-FAKTORN

P-faktorn, där P står för performance eller prestation, talar om i vilken takt det verkliga arbetet på en arbetsstation eller i en aktivitet förhåller sig till den ideala tiden. För att få fram denna faktor krävs det därför att både ideal och verklig tid för önskad aktivitet eller arbetsstation definieras. För att lägga till dessa tider måste vissa steg följas, se rubriken "Lägga till" ovan.



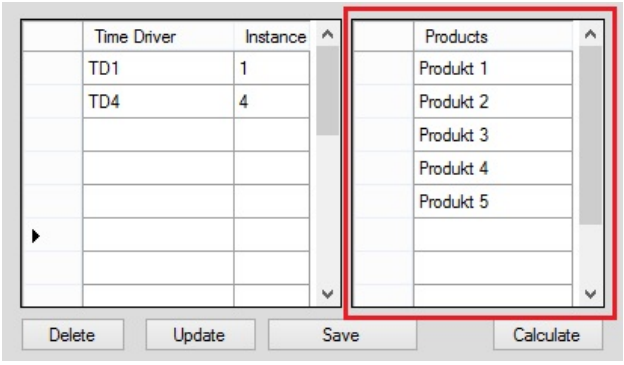
Figur 14. P-faktorn markerad i Informationsfönstret.

Programmet beräknar själv P-faktorn och visar denna i fönstret märkt "Information", se figur 14.

5. PROCESS

För att skapa olika produktflöden används processer. Detta innebär att olika produkter kan kopplas ihop med olika aktiviteter för att på så sätt ta reda på specifik information om en viss produkten. Denna funktion är ännu inte fullt fungerande men det är möjligt att skapa processer och se vilka aktiviteter som krävs för en viss produkt. För att göra detta, följ stegen nedan.

1. En hierarki måste byggas upp enligt rubriken "Lägg till".



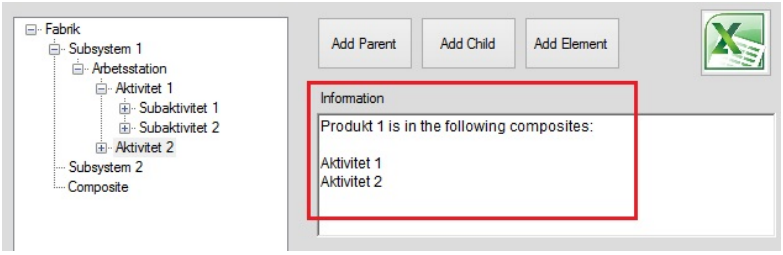
Figur 15. Listan för produkter markerat i rött. Under Listan ses knappen Calculate.

2. En eller flera produkter måste skapas i listan programmets nedre högra hörn, se figur 15. Lägg till ett namn och tryck "Enter".
3. Markera en aktivitet i trädet i programmets övre vänstra hörn.
4. Markera den produkt som kräver den valda aktiviteten genom att trycka på den vita rutan till vänster om produktnamnet. Om flera produkter kräver samma aktivitet trycker du på Ctrl och markerar sedan alla berörda produkter.

5. Tryck på “Save” för att spara tilldelningarna.

För att se vilka aktiviteter en produkt måste genomgå följs stegen nedan:

a. Markera önskad produkt genom att trycka på den vita rutan till vänster om produktnamnet.
b. Tryck på “Calculate”, se figur 15.

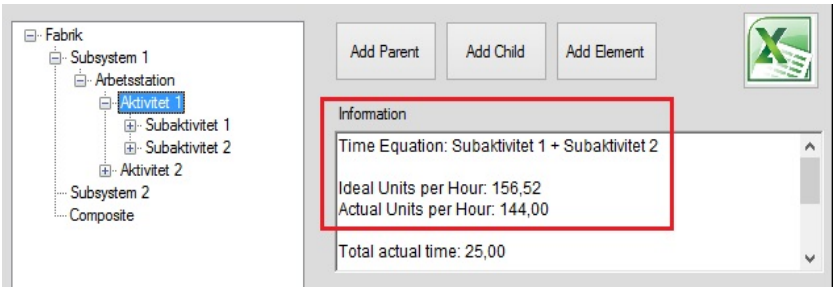


Figur 16. Bild över hur informationsfönstret ser ut då du tryckt på Calculate.

c. I Informationsfönstret kan du nu se vilka aktiviteter som krävs för att tillverka produkten, se figur 16.

6. RESULTAT OCH ANALYS

Det finns flera olika parametrar som skapas som ett resultat av de parametrar som tidigare matats in.



Figur 17. De resultat som genereras från programmet visas i Informationsrutan.

Dessa resultat visas i rutan märkt "Information", se figur 17, förklaras nedan för att under rubriken "Analysmöjligheter" gå igenom hur resultaten kan analyseras.
Beroende på vilken gren i trädet som är markerad kommer informationen vara olika.

6.1 Tidsekvationer

För att se vilka subaktiviteter en viss aktivitet består av och vilka element en viss subaktivitet består av finns tidsekvationerna. Tidsekvationerna visar också antalet subaktiviteter eller element, som motsvaras av tidsdrivare. Ekvationerna kan även visa vilka delar som en anläggning är uppbyggd av. Tidsekvationerna skrivs på formen nedan.
Time Equation: Tidsdrivare 1 x Subaktivitet 1 + Tidsdrivare 2 x Subaktivitet 2

7.2 P-faktorn

P, som nämnts innan, står för performance och talar om i vilken takt jämfört med den ideala hastigheten som arbetet utförs. Om arbetet utförs på en kortare tid än idealt blir P-parametern över 100 % och om det utförs på en längre tid blir P-parametern under 100 %. Denna faktor påverkas endast av arbetarens motivation och fysisk förmåga.

7.3 Ideal kapacitet, CAP_I

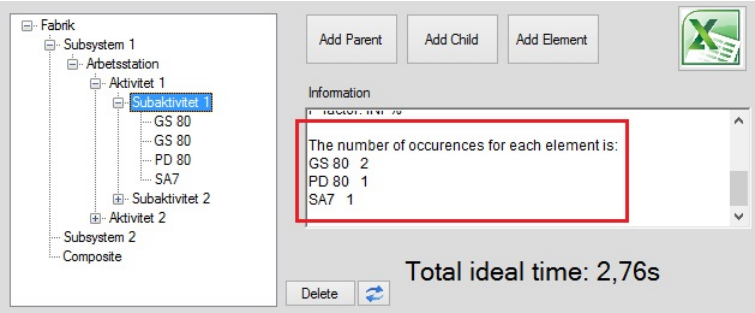
Förklaringen nedan utgår för enkelhetens skull från att en arbetsstation är markerad.
Ideal kapacitet visar hur många enheter per timme som idealt kan tillverkas vid en arbetsstation, det vill säga utan störningar eller annat som hindrar arbetaren från att arbeta. Kapaciteten baseras på den ideala tiden det tar att utföra de element som aktiviteterna i arbetsstationen består av, det bygger alltså på elementartidssystem.
Det som är viktigt att tänka på är att den ideala kapaciteten utgår från de förutsättningarna som finns för tillfället, med bland annat nuvarande metod och kunskapsnivå. Om metoden skulle förändras, till exempel genom att en manuell skruvmejsel byts ut mot en skruvdragare, kommer även den ideala kapaciteten förändras eftersom aktivitetens uppbyggnad kommer bli annorlunda.

7.4 Verklig kapacitet, CAP_R

Förklaringen nedan utgår för enkelhetens skull från att en arbetsstation är markerad.
Verklig kapacitet talar om hur många enheter arbetsstationen för tillfället klarar av att producera. Siffran ska alltså motsvara det faktiska antalet enheter som varje timme lämnar arbetsstationen.

Denna kapacitet beror helt på hur situationen på arbetsstationen och dess aktiviteter ser ut, hur väl metoden fungerar, hur väl arbetarna arbetar, hur väl maskinerna fungerar och så vidare. Ändras någon av parametrarna, till exempel att arbetarna utbildas så att de kan mer och därmed kan arbeta snabbare, kommer även antalet producerade enheter att förändras.

7.5 Antal upprepningar av varje element



Figur 18. Antalet upprepningar av elementen i någon del i hierarkin visas i Informationsrutan.

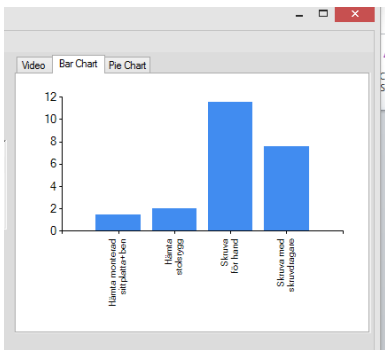
Längst ned i Informations-rutan finns "The number of occurences for each element is" som kan ses i figur 18. Detta visar vilka element och hur många av varje som ingår i en valfri del i hierarkin. Det är till exempel möjligt att markera en arbetsstation för att se vilka element som kan utföras på stationen.

7.6 Utnyttjandegrad

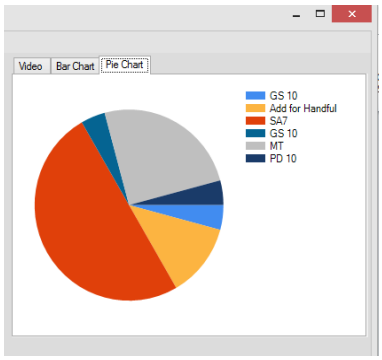
Cirkeldiagrammet talar om hur arbetarnas tid fördelas mellan de fyra olika kategorierna som nämnts ovan. Detta är en inparameter men även ett resultat värt att analysera vid produktionsförbättringar.

7.7 Tidsdiagram

Det finns i övre högre hörnet en ruta med två diagram, ett stapeldiagram och ett cirkeldiagram, se figur 19 och 20. Båda diagrammen visar hur tiden för till exempel en aktivitet fördelas över de olika subaktiviteterna. Den kan även visa hur tiden för en subaktivitet fördelas över olika element, beroende på vilken del i hierarkin som är markerad. I stapeldiagrammet ses hur lång tid respektive subaktivitet eller element tar och i cirkeldiagrammet ses fördelningen av tid mellan subaktiviteterna eller elementen.



Figur 19. Stapeldiagram över tidfördelning.



Figur 20. Cirkeldiagram över tidsfördelning.

7.8 Analys av förbättringspotential

Utifrån de resultat som fås från programmet kan analyser göras för att undersöka förbättringspotentialen. Vilka analyser som görs beror på vad man vill åstadkomma och kan variera från fall till fall. Nedan kommer några förslag på möjliga slutsatser som kan dras och analyser som kan göras med programmet som verktyg.

7.8.1 Jämförelse av ideal och verklig kapacitet

Hur mycket skiljer sig dagens tillverkningshastighet från den ideala hastigheten? Om differensen är stor kan det vara en god idé att fortsätt analysera, bland annat med förslagen nedan. Om differensen inte är så stor kan det betyda att den markerade delen i fabriken redan är tillräckligt optimerad och att förändringar kanske inte är nödvändiga på just det området.

7.8.2 Analys av P-faktorn

Genom att analysera arbetarnas prestation är det möjligt att upptäcka var i flödet arbetet utförs snabbt respektive långsamt. Om prestationen är tillräckligt låg för att förbättring är önskvärt bör man fundera på hur förbättringen ska genomföras. Kanske behövs det utbildning eller metodförklaring för personalen så att de lär sig mer och därigenom blir bättre eller något sätt att höja motivationen för att genom att öka deras motivation öka arbetshastigheten.

7.8.3 Analys av utnyttjandegraden

Utnyttjandegraden talar om hur mycket av arbetstiden vid en anläggning som går åt till att utföra planerade aktiviteter. Det visar också fördelningen av förlusterna; störningar, den tid arbetarna står och väntar samt personlig tid. Detta kan jämföras mellan olika anläggningar inom samma fabrik eller mellan olika fabriker för att se hur de skiljer sig åt. Genom att undersöka utnyttjandegraden får man reda på vad som kan förbättras för att öka den värdeadderande tiden. Till exempel ger höga störningssiffror en indikation på att maskinerna kan behöva underhållas, lagas eller kanske till och med ersättas. Hög andel personlig tid indikerar att personalen tar många eller långa raster och att strängare kontroll av rasterna kanske bör införas. Om arbetarna står och väntar en stor del av tiden, utan att ha något arbete att göra, kan det vara en indikation på att anläggningen är felbalanserad och att den bör balanseras om.

7.8.4 Analys av tidsdiagrammen

I dessa diagram finns tidsfördelningen för en aktivitet eller subaktivitet. Genom att studera dessa diagram, både stapel- och cirkeldiagrammet, är det möjligt att undersöka hur lång tid i förhållande till varandra som subaktivteterna eller elementen tar. Observera att om en elementtyp upprepas så kommer deras tider ej att summeras, utan de visas var för sig i diagrammen.

Om en aktivitet är markerad är det då möjligt att se hur lång tid de olika subaktiviteterna tar och till exempel undersöka om några subaktiviteter skiljer sig allt för mycket åt i tid jämfört med hur det borde vara. Du kan då närmare undersöka en eller flera av dessa subaktiviteter och reda ut varför differensen av deras tider är så stor.

7.8.5 Analys av M-parametern

Genom att analysera den metod som används för det manuella arbetet kan den förbättras och därigenom kan både den ideala och den verkliga kapaciteten förbättras. Det är alltså möjligt att öka antalet producerade enheter per timme enbart genom att titta på och förändra den nuvarande metoden. Det finns i programmet två snarlika sätt att göra detta på som förklaras nedan.

7.8.5.1 Tidsekvationerna

Tidsekvationerna visar hur arbetet genomförs, vilka delar som ingår, hur många av varje ingående del samt dess tid. Tidsekvationerna kan visa om det är möjligt att förbättra metoden genom att till exempel byta ut, ta bort eller ändra några subaktiviteter eller element. Om en subaktivitet innehåller många element av typen "Step" kanske det går att förbättra metoden genom att flytta objektet i fråga närmare arbetsstationen.

Då metoden förändras behöver en ny analys av dagsläget göras. Analysen förändras genom att mata in de nya värdena eller förändra hierarkin så att informationen i programmet stämmer överens med den nya metoden. Att denna förändring i programmet måste göras beror på att den ideala tid som programmet tagit fram från elementen inte längre stämmer med den nya metoden. Det är viktigt att tänka på att så fort metoden förändras kommer även analysen behöva förändras.

Eftersom det är möjligt att spara olika scenarier i programmet går du som användare aldrig miste om någon information du tidigare gjort så länge du sparar en fil för varje scenario, se kapitel 2. Det är även möjligt att i programmet förändra metoden enbart för att testa och se vad resultatet av en viss förändring blir. Detta resultat kan sedan jämföras med dagsläget och en utvärdering kan göras om förändringen är värd att genomföra.

7.8.5.2 Antal upprepningar av varje element

Analysen av detta resultat genomförs på samma sätt som för tidsekvationerna ovan med den skillnaden att det här är möjligt att se alla de element som utförs på till exempel en arbetsstation, och inte bara antalet element i en subaktivitet. Om det totalt sett finns många element av typen "Step" på en viss arbetsstation kan det vara en indikation på att arbetsstationen bör ses över och eventuellt förändras för att minska antalet upprepningar av elementet.
Det ger en överblick över hur arbetet är utformat på en högre nivå än subaktivitetsnivå. Det rekommenderas dock att titta på varje subaktivitet för sig för att dra slutsatser. Men att se antalet upprepningar av element på en arbetsstation kan visa var närmare undersökning bör göras.