CHALMERS

Processer och metoder för utveckling av inbyggda system

DAVID SVÄRD

Examensarbete

Civilingenjörsprogrammet för Elektroteknik

CHALMERS TEKNISKA HÖGSKOLA
Institutionen för datorteknik
Göteborg 2003

Innehållsförteckning

Sammanfattning

Det finns olika processer och metoder för utveckling av programvaruintensiva inbyggda system. Många processer är standardiserade av någon av de många standardiseringsorganisationer som finns i världen. Dessa organisationer kan delas in i grupperna branschspecifika, nationella och internationella där ISO är den främsta av dem.

Det finns ett flertal generella processtandarder för systemutveckling och programvaruutveckling. De främsta är ISO/IEC 15288 System Life Cycle Processes och ISO/IEC 12207 Software Life Cycle Processes.

Vid utveckling av programvara behöver man följa en lämplig process, men också använda lämpliga metoder. eXtreme Programming (XP) är en metodik för utveckling av programvara som tillhör en ny generation av lättviktsprocesser och –metoder. Praktikerna och strategierna i XP baserar sig på principer och värderingar som kommunikation, enkelhet, återkoppling och mod.

Vid jämförelse mellan processtandarden ISO/IEC 12207 och XP visar det sig att de inte är förenliga på ett antal punkter.

Med hjälp av en informationsmodell för en generell utvecklingsprocess kan datorbaserade verktyg som visualiserar, modifierar, anpassar och exekverar processen konstrueras. Appendix 2 innehåller konstruktionsunderlaget för en demonstrator för ett sådant verktyg.

Abstract

There are different processes and methods for development of software intensive embedded systems. Many processes are standardized by any of the many standardization organizations in the world. These organizations can be divided into trade specific, national and international, where ISO is the most prominent one.

There are several generic process standards for systems and software engineering. Most important are ISO/IEC 15288 System Life Cycle Processes and ISO/IEC 12207 Software Life Cycle Processes.

When developing software you need to follow an appropriate process, but also use appropriate methods. eXtreme Programming (XP) is a software development methodology that belongs to a new generation of lightweight processes and methods. The practices and strategies of XP are based on the values, communication, simplicity, feedback and courage.

A comparison between the process standard ISO/IEC 12207 and XP shows that they are inconsistent in a number of ways.

With help from an information model for a generic development process computer based tools that visualize, modify, tailor and execute the process can be designed. Appendix 2 contains a design specification for a demonstrator for such a tool.

Förord

Denna rapport är ett av resultaten av mitt examensarbete som utförts på IVF Industriforskning och utveckling AB. Arbetet har varit mycket lärorikt och gett mig många insikter kring utveckling av inbyggda system.

Ett stort tack till min handledare Håkan Edler, IVF och tidigare Chalmers tekniska högskola, som har hjälpt mig på många sätt. Tack också till medarbetarna på IVF som har bidragit med hjälp och trevlig gemenskap.

1 Inledning

1.1 Bakgrund

De senaste årens tekniska utveckling har resulterat i att industrin producerar alltmer komplexa produkter. Produkterna innehåller ofta teknik från flera olika ingenjörsdiscipliner till exempel mekanik, elektronik och programvara. Traditionellt har dessa discipliner utvecklat sina produkter med sina egna processer, metoder och verktyg. En fråga som industrin står inför idag är om man kan uppnå fördelar, såsom minskade kostnader, kortare utvecklingstider och förbättrade möjligheter till projektstyrning, om de olika disciplinerna samordnar sina processer, metoder och verktyg.

1.2 Syfte

Syftet med denna rapport är att undersöka vilka processer och metoder som bör användas för att utveckla inbyggda system. Ett inbyggt system är ett system vars funktionalitet realiseras genom en samverkan av flera olika teknologier[1]. Inriktningen i denna rapport är på inbyggda system som består av mekanik, elektronik och programvara och som dessutom är programvaruintensiva.

Undersökningen av vilka processer som bör användas kommer att begränsas till att kartlägga vilka standarder för produktutvecklingsprocesser som finns för mekanik, elektronik, programvara och sammansatta system.

Undersökningen av vilka metoder som bör användas kommer att begränsas till ett närmare studium av en modern lättviktsmetod för utveckling av programvara, eXtreme Programming (XP). Lärdomar från detta studium kan säkert också tillämpas inom andra teknikdiscipliner än programvaruutveckling.

Ett ytterligare syfte med rapporten är att undersöka möjligheterna att finna en modell för utvecklingsprocessen som går att tillämpa inom alla teknikdisciplinerna. Detta syfte kommer att uppnås genom att beskriva en informationsmodell för en generell utvecklingsprocess. Informationsmodellen utgör en god grund för implementation av ett system för datorstödd processhantering.

1.3 Metod

Stoffet till den här rapporten har tillkommit genom litteraturstudier, sökningar på internet och samtal med forskare på IVF. Rapporten har tagits fram med vägledning av en utvecklingsprocess för programvara – TSPi[2] (Team Software Process – introduction).

1.4 Disposition

Kapitel 2 beskriver viktiga aspekter vid val av utvecklingsprocess. Därefter ges en introduktion till standarder och standardiseringsorganisationer. Kapitlet avslutas

[1] Definition som används av IVF Industriforskning och utveckling AB.

[2] TSP är en programvaruprocess framtagen av W. S. Humphrey.

med en kartläggning av standarder som kan användas vid utveckling av inbyggda system, antingen på systemnivå eller för någon specifik teknikgren.

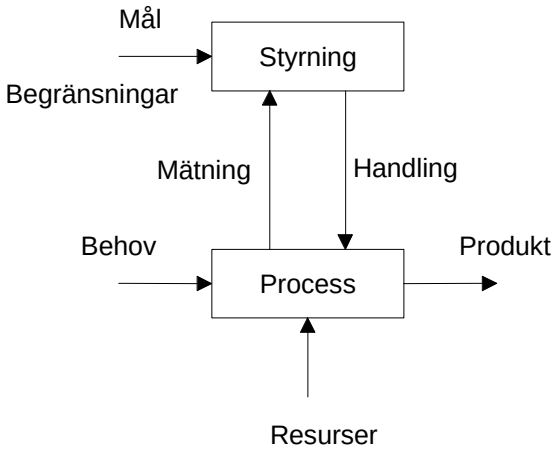
I kapitel 3 studeras metodik för programvaruutveckling, i huvudsak XP. Först ges en introduktion till XP. Därefter görs en jämförande analys av XP och processtandarden ISO/IEC 12207 Software Life Cycle Processes.

I kapitel 4 beskrivs en informationsmodell för en allmän utvecklingsprocess.

2 Standarder för utveckling av inbyggda system

I industrin idag låter man oftast de olika ingenjörsdisciplinerna arbeta var för sig. Därefter integreras de olika lösningarna för mekanik, elektronik och programvara. Utveckling av inbyggda system är i huvudsak ett tekniskt problem och kan lösas med hjälp av ingenjörskonst. Moore beskriver en modell för ingenjörsarbete återgiven i Figur 1.

I den här rapporten fokuseras processens roll för att åstadkomma produkter. Avsnitt 2.1 beskriver några olika sorters produktframtagningsprocesser som finns att välja mellan. Där nämns bland andra processer som är standardiserade (t ex ISO/IEC 12207) och processer som finns i litteraturen (t ex TSP). Avsnitt 2.2 ger en introduktion till standarder och standardiseringsorganisationer. Avsnitt 2.3 beskriver ett antal standarder inom området inbyggda system.



Figur 1. Modell för ingenjörsarbete enligt Moore [Moore98]

2.1 Val av utvecklingsprocess

Det första och allt för vanliga sättet att välja utvecklingsprocess är att välja att arbeta utan en medveten process, att arbeta godtyckligt. Oftast är detta inte ett aktivt val, utan ett passivt val. Eftersom ingen gemensam idé finns för hur man skall arbeta i företaget eller i projektet, lämnas den enskilde ingenjören att arbeta efter eget huvud. En skicklig ingenjör kan ofta åstadkomma mycket när han eller hon får arbeta efter eget huvud, men när produkterna som skall tas fram blir allt mer komplexa och involverar allt större resurser blir risken för misslyckande mycket stor. Även den mest kompetenta ingenjör har nytta av att arbeta efter en

väl anpassad utvecklingsprocess. Det finns stora fördelar med att dokumentera processen.

Det andra sättet, även detta mycket vanligt, är att inom organisationen ta fram en egen utvecklingsprocess. I detta arbete kan företagets alla egna erfarenheter av framgångar och misslyckanden användas.

Istället för att endast lära av sina egna misstag kan man även lära av andras. Man kan utnyttja den erfarenhet som andra skaffat sig genom att använda en kommersiell utvecklingsprocess. Man kan även finna mycket användbart i litteraturen. På programvarusidan finns väl etablerade utvecklingsprocesser såsom Rational Unified Process (RUP) och SEI:s Team Software Process (TSP).

Ett fjärde och ofta underskattat sätt att välja utvecklingsprocess är att utnyttja standardiserade utvecklingsprocesser. Det finns internationella, nationella och branschspecifika standarder som innehåller mycket av industrins samlade kunskaper kring hur man bäst bör arbeta för att producera produkter med hög kvalitet, pålitlighet och säkerhet.

När utvecklingsprocess är vald krävs det dessutom att man väljer vilka metoder man ska använda för att genomföra de aktiviteter som processerna kräver. Dessa metoder kan vara standardiserade, egenutvecklade eller hämtade från litteraturen.

Många metoder kan effektiviseras med hjälp av datorbaserade verktyg. Verktygen kan utvecklas av organisationen själv eller köpas in. Det finns standarder som hjälper till att välja rätt verktyg.

I detta kapitel kommer jag att koncentrera mig på det fjärde sättet att välja utvecklingsprocess, genom att kartlägga vilka standarder som kan vara till nytta. Först vill jag emellertid ge en introduktion till standarder och de organisationer som tar fram dem.

2.2 Introduktion till standarder och standardiseringsorganisationer

Enligt ISO (International Organization for Standardization) är standarder ”dokumenterade överenskommelser innehållande tekniska specifikationer eller andra precisa kriterier för konsistent användning såsom regler, riktlinjer eller definitioner av karaktäristik, för att säkerställa att material, produkter, processer och tjänster är anpassade för sina syften” [ISO].

Det förekommer olika beteckningar för standarder beroende på innehållet och sättet på vilket den tillkommit. Några exempel är: standard, rekommendation, handbok, dokument, specifikation och guide.

Orsaken till att man gör standarder är att de gör livet enklare för många människor. Det underlättar för dem som producerar varor och tjänster genom att resurser sparas när samarbete inom branschen blir möjligt. Det underlättar också för konsumenterna eftersom de får billigare, kompatibla och säkrare produkter.

Exempelvis är ANSI:s uppdrag att höja amerikanarnas levnadsstandard och värna USA:s internationella affärsintressen [ANSI].

2.2.1 Standardiseringsorganisationer

Det finns några olika typer av organisationer som gör standarder. De allra flesta är frivilliga initiativ från industrin som drivs utan vinstintresse. Jag har valt att kategorisera standardiseringsorganisationerna i tre olika grupper: nationella, interna-

tionella och branschspecifika. Eftersom de allra flesta branscher är internationella så finns det en viss redundans i den uppdelningen.

Nästan varje nation i världen har ett särskilt organ som på industrins och myndigheternas uppdrag samordnar nationens standardiseringsaktiviteter. Dessa organ fastställer nationella standarder och deltager i internationella aktiviteter.

USA är det land som är mest framstående vad gäller framtagande av standarder. Deras standardiseringsaktiviteter samordnas av American National Standards Institute (ANSI). ANSI gör inga standarder själva, istället har de auktoriserat ett stort antal andra organisationer att publicera nationella standarder. Det finns särskilda krav att uppfylla för att bli auktoriserad av ANSI.

I Sverige finns på liknande sätt Sveriges standariseringsråd (SSR) som auktoriserar svenska standardiseringsorgan och publicerar en lista över svenska standarder. SSR har auktoriserat Swedish Standards Institute (SIS), Informationstekniska standardiseringen (ITS) och Svenska Elektriska Kommissionen (SEK) som svenska standardiseringsorgan. [SS]

Det största internationella standardiseringsorganet är ISO. I det ingår nationella standardiseringsorgan från nästan alla länder som medlemmar. ISO har även samarbete med andra standardiseringsorgan. ISO:s standarder täcker alla tekniska fält. Standarder inom området elektroteknik och elektronik har emellertid överlåtits åt International Electrotechnical Commission (IEC) som har en liknande struktur som ISO. Inom området informationsteknik har ISO och IEC valt att samarbeta ingående genom att bilda en gemensam teknisk kommitté Joint Technical Commission 1 (JTC1). Subkommitté 7 (SC7) ansvarar för programvaru- och systemutveckling. De har för närvarande ca 100 standarder i sin samling.

ISO har också ett samarbete med International Telecomunication Union (ITU) kring standarder inom telekommunikation och informationsteknik. ITU är till skillnad från de andra organen ett samarbete mellan ländernas myndigheter.

I Europa finns också gemensamma standardiseringsinitiativ. European Committee for Standardization (CEN) har här en liknande roll som ISO har globalt.

2.2.2 Branschspecifika standardiseringsorganisationer

Det finns ett stort antal branschorganisationer som producerar standarder. I USA är många av dessa aukoriserade av ANSI och deras standarder har därmed status som ”US standards”. ANSI har över 270 auktoriserade organisationer, men det finns många fler organisationer i USA som tar fram standarder. I denna rapport kommer några av dem som har störst relevans för utvecklandet av inbyggda system att nämnas.

Världens största tekniska professionella sällskap med mer än 320 000 medlemmar i nära 150 länder är Institute of Electrical and Electronics Engineers (IEEE). Inom IEEE är Computer Society (CS) med huvudkontor i Washington, DC den största grenen. Inom CS är Software Engineering Standards Committee (SESC) ansvarigt för standarder inom programvaruutveckling. SESC och JTC1/SC7 är de två tyngst vägande instanserna för standarder inom programvaruutveckling i världen.

Electronic Industries Alliance (EIA) är en handelsorganisation som omfattar sex mindre sammanslutningar och representerar ett stort antal amerikanska företag inom elektronikbranschen. Företag som konstruerar och tillverkar produkter som främst är ämnade för myndigheterna är sammanslutna i Government Electronics

and Information Association (GEIA). GEIA:s tekniska kommitté med beteckningen G-47 ansvarar för standarder inom systemutveckling och relaterade områden.

International Council on Systems Engineering (INCOSE) är en internationell organisation som har till syfte att främja systemutvecklingsområdet och är starkt involverad i standardiseringsaktiviteter även om inga egna standarder tas fram.

American Institute of Aeronautics and Astronautics (AIAA) heter det främsta professionella sällskapet inom rymdbranschen. Organisationen har 30 000 medlemmar över hela världen.

European Cooperation for Space Standardization (ECSS) är ett europeiskt initiativ för att skapa en samling koherenta standarder för rymdindustrin.

Radio Technical Commission for Aeronautics (RTCA) är ett privat, icke vinstdrivande aktiebolag som utvecklar rekommendationer för kommunikation, navigation, övervakning och lufttrafik baserade på konsesus mellan intressenterna. Mer än 250 myndigheter, industriella och akademiska organisationer från USA och runt om i världen ingår i RTCA.

Object Management Group (OMG) är ett icke vinstdrivande konsortium, som producerar och underhåller specifikationer för företagsapplikationer som fungerar tillsammans. Så gott som alla stora företag inom datorindustrin och hundratals små är medlemmar i OMG.

I appendix 2 har jag inkluderat en lista på organisationer som jag kommit till kännedom om genom mitt arbete.

De flesta organisationer nämnda i detta avsnitt har inom sig bildat tekniska kommittéer (TC) vilka ansvarar för olika teknikområden. I kommittéerna ingår tekniska experter, ofta från hela världen. Vidare kan en teknisk kommitté vara indelad i subkommittéer (SC) och arbetsgrupper (WG).

2.2.3 Hur görs en standard?

Standarder inom nämnda organ kommer till stånd genom samverkan mellan experter. När internationella standarder tas fram följs en väldefinierad process som kännetecknas av att konsensus mellan alla intressenter måste uppnås, att lösningarna fungerar globalt, samt att arbetet är frivilligt och drivs av marknadskrafterna. [ISO]

En standard som inte är helt färdig kan publiceras som utkast (Draft) för granskning.

2.2.4 När görs en standard?

Vanligtvis uppstår behovet av en standard inom någon industrisektor, som gör en förfrågan till ett nationellt standardiseringsorgan. Denna gör i sin tur en förfrågan till den aktuella internationella organisationen om en internationell standard. På internationella standarder görs revisioner åtminstone vart femte år.

2.3 Standarder för utveckling av inbyggda system

I detta kapitel omnämner jag ett antal standarder och ger beskrivningar av ett mindre antal. Det är nödvändigt att kraftigt begränsa antalet standarder som tas upp med tanke på det stora antalet som existerar kombinerat med mina begränsningar i tid. Trots att jag ägnat mycket tid åt att försöka finna så många relevanta standarder som möjligt kan jag ha missat någon som borde ha varit med.

Jag har försökt finna standarder för utvecklingsprocesser som beskriver den högsta nivån – ramverk inom vilka utveckling sker.

Jag har funnit ett flertal standarder för systemutveckling. Här betraktas systemutveckling som en egen ingenjörsdisciplin, som behövs för att hantera komplexiteten som uppstår i tekniska system. Dessa standarder är gjorda för att kunna hantera utvecklingen av produkter som innehåller teknik från flera ingenjörsdiscipliner såsom elektronik, mekanik och programvara. Just den typen av standarder borde vara utmärkta att använda vid utveckling av inbyggda system.

Det är emellertid inte nog då det också behövs standarder för de ingående disciplinerna, t ex programvaruutveckling. Därför har jag sökt efter standarder som beskriver utvecklingsprocessen för produktframtagning inom de olika disciplinerna. När det gäller elektronik- respektive mekanikutveckling har jag inte funnit mycket. På programvaruområdet däremot har det funnits och finns fortfarande många processtandarder. Många branscher har t ex tagit fram sina egna standarder för programvaru- och systemutveckling.

Det finns även standarder inriktade på att produkterna skall uppnå höga krav på kvalitet, pålitlighet eller säkerhet. Dessa standarder beskriver ofta hela eller större delen av processerna för produktens livscykel. Där jag funnit det lämpligt har jag även tagit med standarder på en lägre nivå som beskriver processer, metoder, tekniker och verktyg.

En populär standard som kan användas som skal kring utvecklingen av nästan vilken produkt som helst är ISO 9001. Den ger emellertid inte den ingående vägledning som behövs för framtagning av ett inbyggt system. Mer information om ISO 9001 återfinns i avsnitt 4.3.2.

2.3.1 Standarder för utveckling av generella system

System- respektive programvaruutveckling är två discipliner som ligger mycket nära varandra. Moderna IT-produkter är emellertid ofta så komplicerade att de kräver tillämpning av tekniker från båda disciplinerna [Moore98]. Tabell 1 visar tre olika standarder för ett systems livscykelprocesser och dessutom en guide till en av dem.

ISO/IEC 15288, EIA-632 och IEEE 1220 kan ses som alternativ till varandra att välja mellan. Några av deltagarna i utvecklingen av standarden önskar se en hierarki mellan dessa tre, där 15288 är den mest generella och 1220 den mest branschberoende.

Det finns antagligen fler specialiserade standarder än de som jag omnämner i detta avsnitt, men dessa verkar vara de viktigaste för en internationell publik. [Moore], [Martin]

Tabell 1. Standarder för utvecklingsprocesser för generella system som är programvaruintensiva. (Kursivering betyder att standarden ifråga är under utveckling.)

Standard	Titel	År
ISO/IEC 15288	System life cycle processes	2002
ISO/IEC DTR 19760	Systems engineering – Guide for ISO/IEC 15288 (System life cycle processes)	2003
ANSI/EIA-632	Processes for engineering a system	1999
IEEE Std 1220	Application and management of the systems engineering process	1998

ECSS-E-10A	System engineering	1996

Livscykelprocesser

Ett produktsystem genomgår olika faser under sin livstid, t ex utveckling, operation, underhåll och urdrifttagande/upphörande. Detta kallas för produktens livscykel. Under hela denna tid finns det ett antal processer som beskriver vilka aktiviteter leverantören skall genomföra och vad de skall resultera i. Vissa av processerna är endast intressanta under vissa faser av produktens livscykel andra pågår hela tiden.

ISO/IEC 15288

Denna nya standard har tagits fram av ISO/IEC JTC1/SC7/WG7 så sent som i november 2002. Standarden är baserad på en ny modell och arbetet har skett i nära samarbete med INCOSE [Coallier]. Arbetet påbörjades 1996 och har alltså tagit sex år. En stor fördel med utvecklingen av denna standard har varit att WG7 samtidigt har kunnat revidera ISO/IEC 12207 för att uppnå en god anpassning mellan de båda.

Syftet med standarden är att etablera ett ramverk för att beskriva livscykeln för människogjorda system och en mängd väldefinierade processer, samt tillhörande terminologi.

Standarden beskriver systemutveckling på en mycket hög nivå. Standarder på en lägre nivå som är knutna till ett visst systemelement är tänkta att användas för att implementera elementet ifråga. För systemelement som implementeras i programvara tillämpas processerna i ISO/IEC 12207.

Processerna i 15288 delas in i fyra grupper av processer enligt nedan. Varje process definieras av dess syfte, resultat och aktiviteter. Processerna är:

- Agreement Processes
  - Acquisition
  - Supply
- Enterprise Processes
  - Enterprise Environment Management
  - Investment Management
  - System Life Cycle Process Management
  - Resource Management
  - Quality Management
- Project Processes
  - Project Planning
  - Project Assessment
  - Project Control
  - Decision-making

  - Risk Management
  - Configuration Management
  - Information Management
- Technical Processes
  - Stakeholder Requirements Definition
  - Requirements Analysis
  - Architectural Design
  - Implementation
  - Integration
  - Verification
  - Transition
  - Systems Analysis
  - Validation
  - Operation
  - Maintenance
  - Disposal

Processmodellen som beskrivs av 15288 kan användas för att stödja processutvärdering i enlighet med ISO/IEC 15504. Beslut har fattats om att harmonisera 15288 med ersättaren till ISO 9000-3. Dessutom har beslut fattats om att koordination med internationellt erkända standarder för systemutveckling, framtagna av exempelvis IEEE och EIA, ska företas. [Software]

I början av 2003 planeras en guide till ISO/IEC 15288, för närvarande med namnet ISO/IEC DTR[3] 19760, att släppas som en teknisk rapport.

EIA-632

I mitten av 90-talet beslutade amerikanska försvarsdepartementet (Department of Defense) att man skulle satsa mindre på egenutvecklade särskilda militära standarder och satsa mer på kommersiella. Detta ledde till att försvarets nyutvecklade systemutvecklingsstandard Mil-Std 499B aldrig publicerades. Det dokumentet transformerades istället av EIA till EIA/IS-632[4] som är befriad från den militära terminologin. IS-632 har senare omarbetats till en högre abstraktionsnivå för att kunna rymma fler tillämpningsområden och släpptes 1999 som en full standard, EIA-632.

Syftet med EIA-632 är att den ska förmå en organisation att förbättra sin konkurrenskraft på globala marknader genom att utveckla och producera kvalitetsprodukter och leverera dem i tid till överkomliga priser eller kostnader.

[3] DTR – Draft Technical Report

[4] IS – Interim Standard

Standarden innehåller tretton processer som behandlar alla steg i utvecklingen av ett system. Alla aspekter av utvecklingens livscykel hanteras. Processerna beskriver arbetsuppgifter och resultat och de är uppdelade i följande grupper:

- Acquisition and Supply Processes
  - Supply
  - Acquisition
- Technical Management Processes
  - Planning
  - Assessment
  - Control
- System Design Processes
  - Requirements Definition
  - Solution Definition
- Product Realization Processes
  - Implementation
  - Transition to Use
- Technical Evaluation Processes
  - System Analysis
  - Requirements Validation
  - Product Verification
  - Product Validation

Författarna till EIA-632 vill se IEEE 1220 som en tillämpning av 632 för elektronik- och elektroteknikindustrin. Eventuellt kommer en guide till 632 att tas fram i samarbete mellan GEIA och INCOSE.

IEEE 1220

IEEE 1220 utvecklades parallellt med att amerikanska försvarsdepartementet utvecklade Mil-Std-499B och bygger på samma modell som denna.

Målet för IEEE 1220 är att definiera kraven på en organisations totala tekniska ansträngningar relaterade till utvecklingen av system och konsumentprodukter (inklusive datorer och programvara), liksom även processerna som tillhandahåller stöd under produkternas livscykel. Standarden omfattar all faser av produktens liv från första koncept till utveckling, operation och bortskaffande.

IEEE 1220 kan ses som en tillämpning av hög-nivåstandarder för elektronikindustrin. Till skillnad från andra systemutvecklingsstandarder saknar IEEE 1220 betoningen på ansvarförhållanden mellan beställare och leverantör.

Standarden kan tillämpas för stegvis påbyggnad av existerande produkt, eller för nyutveckling. Den ska fungera både för engångsprodukter, såsom en satellit, och för massproducerade produkter.

ECSS-E-10A

Det europeiska rymdorganet ESA har tagit fram ett antal standarder för utveckling av rymdapplikationer. Dessa har haft beteckningen PSS (Planning Standards Subcommittee) men fasas nu ut och ersätts av standarder framtagna av det nya initiativet ECSS (European Cooperation for Space Standardization). [ESA]

ECSS-E-10A är tänkt att vägleda utvecklingen av system (inklusive elektronik, programvara, man-in-the-loop, faciliteter och tjänster) för rymdtillämpningar. Standarden specificerar implementationskrav för den ansvariga systemutvecklingsorganisationen med utgångspunkt från att systemutvecklingsprocessen definierad i standarden ECSS-E-10-01 tillämpas [ECSS].

Processutvärdering

Det finns standarder som kan användas för att utvärdera mognadsnivån hos en organisations processer för systemutveckling. För att kunna göra det krävs att man utgår från en referensprocess som är erkänt bra. Tre olika standarder för utvärdering av processer för systemutveckling återfinns i Tabell 2.

Tabell 2. Standarder för utvärdering av processer för systemutveckling

Standard	Titel	År
EIA 731-1	Systems engineering capability model	2002
ISO/IEC TR 15504	Information technology – Software process assessment	1998
CMMI	Capability maturity model integration	2000

EIA-731-1

Denna standard är en integration av de två tidigare utvärderingsstandarderna SE-CMM (från EPIC) och SECAM (från INCOSE).

Den beskriver en två-dimensionell modell med fokuserade områden (Focus Areas) på ena axeln och mognadsnivå på den andra axeln.

Modellen är speciellt framtagen för att kunna utvärdera organisationer vilkas processer definierats utifrån EIA-632 eller IEEE 1220. Standarden kommer emellertid att tas ur bruk i takt med att användarna övergår till CMMI [Denham02].

ISO/IEC 15504

Detta är en standard, även känd som SPICE (Software Process Improvement and Capability dEtermination), för utvärdering av programvaruprocesser, men den omarbetas nu av JTC1/SC7 för att bli mer generell och kunna appliceras på vilken process som helst. Man planerar att ta bort den information som är specifik för programvaruprocessen [Martin03]. För närvarande är standarden uppdelat i nio olika delar.

Processmodellen i SPICE är uppdelad i fem processkategorier och det ingår totalt runt 40 processer. Varje process beskrivs med ett antal base-practices som skall följas för att processen skall anses vara mogen. I SPICE utvärderas en organisations processer gentemot de enskilda processerna i en tvådimensionell skala.

Processerna i ISO/IEC 15288 kan användas som referensmodell vid processutvärdering med SPICE. Processerna i SPICE är också harmoniserade med processerna som beskrivs av ISO/IEC 12207 Life Cycle Processes. SPICE är också avsedd att vara harmoniserad med ISO 9000.

CMMI

I USA har representanter för myndigheterna och industrin har gått samman för att ta fram en modell för utvärdering av en organisations totala tekniska och ledningsmässiga ansträngningar i samband med produktutveckling. SEI:s (Software Engineering Institute) olika modeller för utvärdering av processer och EIA/IS-731 har integrerats i detta arbete som följaktligen kallas CMM Integration.

Elementen i CMMI kallas processområden (Process Areas) och utvärderingen kan representeras på två olika sätt. Dels med en kontinuerlig vy, som är en tvådimensionell modell hämtad från EIA/IS 731, och dels med en nivåindelade vy, som är hämtade från SEI SW-CMM.

En möjlighet är att CMMI antas som en full standard av EIA [Martin03]. För dem som använder EIA 731 idag kommer övergången till CMMI att bli mjuk [CMMI].

Det finns ett antal instanser av CMMI anpassade till olika området, t ex CMMI-SW (Software) och CMMI-SE/SW (Systems Engineering/Software).

Tabell 3. Standarder relevanta för systemutveckling. (Kursivering betyder att standarden ifråga är under utveckling.)

Standard	Titel	År
ANSI/AIAA G-043	AIAA guide for the preparation of operational concept documents	1992
IEEE 1362	IEEE guide for information technology system definition concept of operation document	1998
IEEE 1233	IEEE guide for developing system requirements specifications	1998
IEEE 1471	IEEE recommended practice for architectural description of software-intensive systems	2000
ISO/AWI 10303-233	Industrial automation systems and integration – Part 233: Systems engineering data representation	?
UML for SE	UML for systems engineering	?

Övriga relaterade standarder

För att utveckla komplicerade system som innehåller programvara, krävs förutom kunskaper i systemutveckling kunskaper i bl a programvaruutveckling. För att programvaruutvecklarna ska kunna genomföra sitt arbete behöver de viss information om systemet från systemutvecklarna [Moore98]. Vanligen är den information som krävs ”concept of operations”, systemkrav och arkitektur. I Tabell 3 återfinns bl a standarder som tillhandahåller vägledning för hur informationen kan dokumenteras.

Konceptanalys handlar om att undersöka en systemlösning i dess operationella kontext. Resultatet av analysen formuleras i ett ”ConOps”-dokument. IEEE 1362 föreskriver önskvärt innehåll och format för ett sådant dokument. Ett alternativ är AIAA G-043-1992 som har större fokus på framtagandet av dokumentet och är särskilt lämpligt för rymdapplikationer. AIAA G-043 uppgraderas nu i partnerskap mellan AIAA och INCOSE [Martin03].

Vid framtagningen av specifikationen av systemkraven finns vägledning att få från IEEE Std 1233. Standarden åberopar IEEE 1220 för processen som resulterar i en kravspecifikation, men tillhandahåller även ytterligare information.

Arkitektur handlar om hur den allra högsta nivån av en systemlösning utformas. IEEE 1471 ger rekommendationer för hur ett programvaruintensivt systems arkitektur kan beskrivas.

ISO/AWI 10303-233 är ett nytt projekt inom ramen för ISO-programmet STEP (Standard for the Exchange of the Product model data). Projektet skall resultera i en standard för systemutvecklingsdata.

Unified Modeling Language (UML) från Object Management Group (OMG) har varit en succé när det gäller att modellera programvarusystem. OMG samarbetar nu med INCOSE för att utvidga UML till att kunna modellera ett stort antal system innehållande hårdvara, programvara, data, personal och faciliteter.

OMG SE DSIG (System Engineering Domian Special Interest Group) har följande mål för sitt arbete med att utvidga UML:

– Skapa ett modelleringsspåk för specifikation, design, och verifikation av komplexa system.
– Främja integration av disciplinerna programvaruutveckling och systemutveckling.
– Främja kontroll vid överföringen av information mellan discipliner och verktyg för utveckling av system.

2.3.2 Standarder för programvaruutveckling

Mycket av materialet i detta avsnitt är hämtad från [Moore98].

Eftersom många olika organisationer har gjort sina egna standarder finns det många standarder under denna rubrik. De har inte alltid varit konsistenta, ibland har de rent av varit motsägande. Detta har givetvis varit ett problem för den organisation som har velat arbeta på ett strukturerat sätt. Idag finns en ambition att samordna insatserna för att göra standarderna mer enhetliga, inte minst vad gäller terminologi. Livscykeln beskriven av ISO 12207 ser ut att få en central roll i detta arbete.

Standarderna för programvaruutveckling kan kategoriseras på olika sätt. Ett sätt som skiljer standarder åt är vilken nivå av specialisering/generalisering de har. Vissa standarder behandlar allmänna principer medan andra är specifika och behandlar detaljer. SESC har anammat en lagervy för standarder som innehåller dokument med ökande specialerising: terminologi, översiktlig guide, principer, grunder, applikationsguider och tekniker.

2.3.3 Sammanhang för programvaruutveckling

Ett annat sätt att kategorisera standarder är att utgå från vilken vetenskaplig eller teknisk disciplin de tillhör. James W. Moore som skriver för SESC:s räkning identifierar sex disciplinära sammanhang för programvaruutveckling och beskriver vilka standarder som hör hemma inom respektive sammanhang. Jag presenterar nedan en kort sammanfattning av kontexterna som är: datavetenskap, kvalitetsstyrning, projektledning, systemutveckling, pålitlighet och säkerhet. Givetvis är dessa kontexter, datavetenskap undantaget, relevanta för de andra disciplinerna (system, elektronik och mekanik) också.

Sammanhangen täcker många behov som finns inom industrin. Vissa av aspekterna är ointressanta för vissa. Den som gör COTS-produkter för ”skrivbordsapplikationer” behöver inte tänka på särskilt mycket på ett större system än själva programvaran. Många däremot som utvecklar systemprodukter (inbyggda system) behöver tänka också på pålitlighet och kanske säkerhet.

Datavetenskap

I denna kategori finns standarder som innehåller taxonomi för datavetenskap, terminologi för datavetenskap, samt standarder som beskriver datavetenskapliga tekniker.

Kvalitetstyrning

ISO 9000-3

Kvalitetsledningssystemet ISO 9000 tillhandahåller standarder som är relevanta för utveckling av programvara. ISO 9000-3 är en handledning för hur ISO 9001 ska användas vid utveckling av programvara. Eftersom ISO 9000 är skriven med inriktning på tillverkande industri har det funnits ett behov av en särskild anpassning till programvaruutveckling, som nästan uteslutande består av konstruktion. Följande rubriker beskrivs av ISO 9000-3:

- Management responsibility
- Quality system
- Contract review
- Software development and design
- Document and data control
- Purchasing requirements
- Customer-supplied products
- Product identification and tracing
- Process control requirements
- Product inspection and testing
- Control of inspection equipment
- Inspection and test status of products
- Control of nonconforming products
- Corrective and preventive action
- Handling, storage, and delivery
- Control of quality records
- Internal quality audit requirements
- Training requirements
- Servicing requirements

– Statistical techniques

ISO 9000-serien är i första hand skriven för beställaren av en produkt och inte för leverantören. Många leverantörer väljer att arbeta efter standarden och att dessutom bli certifierade. Beställaren får hjälp att försäkra sig om att det certifierade företaget arbetar på ett strukturerat sätt, genom att standarden specificerar vad som ska göras. Leverantören måste emellertid bestämma hur saker och ting ska göras. Det ger standarden ingen hjälp med. Standarden är till hjälp vid kontraktsskrivning, men för leverantören är dess vägledning inte tillräcklig för att kunna producera högkvalitativ programvara.

IEEE Std 730-2002 specificerar vilket format och innehåll som Software Quality Assurance Plans bör ha.

Den kanadensiska standarden CAN/CSA-Q396 föreskriver hur kvalitetssäkring av programvara skiljer sig åt beroende på hur kritisk programvaran är.

Projektledning

Att leda och styra ett projekt involverar ett antal principer och aktiviteter oavsett vilken typ av projekt som genomförs. Olika teknikdiscipliner och projekt kräver emellertid att metoder, tekniker, etc. är anpassade till aktiviteten ifråga. PMBOK (Guide to the Project Management Body of Knowledge) från Project Management Institute (PMI) är en standard som uppfyller det första behovet genom att tillhandahålla generell kunskap om projektledning. Tillämpningen för programvara finner vi i IEEE 1058.1 Software Project Management Plans. PMBOK beskriver fem typer av ledningsprocesser [Moore98]:

– Initiating
– Planning
– Executing
– Controlling
– Closing

IEEE 1058.1 skrevs utan PMBOK som mall, men kan ändå ses som en specialisering av processerna i den. Skillnaden mellan att leda ett programvaruprojekt och projektledning i allmänhet är inte så stor som en programmerare till en början kan tro.

Standarder för konfigurationsstyrning passar också in under denna rubrik. Det finns ett flertal sådana.

Pålitlighet

Pålitlighet är tänkt att vara ett samlande icke-kvantitativt begrepp som inbegriper en produkts karaktäristik på följande områden:

– Tillgänglighet avgör hur redo produkten är att användas på begäran.
– Tillförlitlighet bestämmer livslängden för produktens prestation.

– Underhållbarhet beskriver hur enkelt det är att underhålla och uppgradera produkten.
– Stöd för underhåll tillämpar kontinuerligt stöd för att uppnå tillgänglighetsmål.

IEC TC56 är en teknisk kommitté som tar fram generella standarder för pålitlighet som går att använda inom alla industrisektorer.

Systemsäkerhet

Traditionellt har säkerhetsstandarder skrivits av specifika branschgrupper. Det har förekommit debatt om huruvida det är lämpligt att skriva generiska standarder för systemsäkerhet och även om huruvida begreppet ”software safety” har någon egentlig innebörd eller inte. Det har uppstått ett stort antal standarder på området eftersom de skrivits av så många grupper.

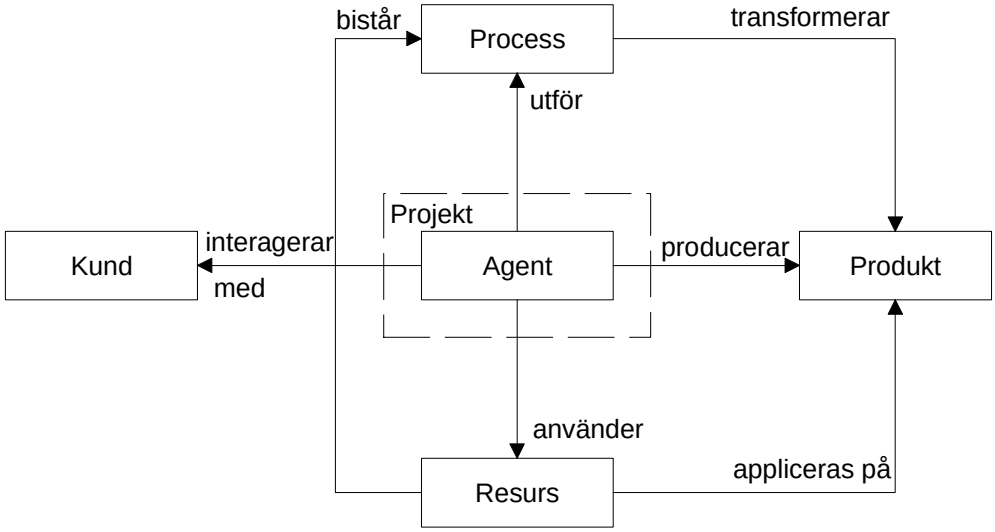
Nedan finns dock några standardsamlingar från grupper som har verkat för att uppnå en mer enhetlig syn på systemsäkerhet genom att söka harmonisering med andra standarder.

– Standarderna för funktionell säkerhet från IEC SC65A började som sektorspecifika, men blir alltmer generella. SC65A har tagit fram standarden IEC 61508, Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems som består av sju delar. [IEC61508]
– Ett exempel på en sektorspecifik samling som formar relationer till en mer generell grupp är standarderna för programvara i kärnkraftsreaktorers säkerhetssystem från IEC SC45A.
– En annan samling som är harmoniserad med andra nationella och internationella standarder är IEEE Power Engineering Society:s standarder för programvara i kärnkraftsreaktorers säkerhetssystem.

Medan pålitlighetsaspekten handlar om att få system att fungera i enlighet med sina krav, så handlar systemsäkerhet om att systemet inte ska orsaka skada oavsett om det fungerar som det ska eller inte.

2.3.4 Objekt för programvaruutveckling

Ett tredje sätt att kategorisera standarder är att identifiera ”objekten” för programvaruutveckling (se Figur 2). Enligt detta synsätt genomförs utveckling av programvara av ett projekt som innehåller ett antal olika agenter. Dessa agenter interagerar med kunder och använder resurser för att utföra processer som innebär produktion av produkter.



Figur 2. Objekten för programvaruutveckling enligt Moore [Moore98]

Resurser

Enligt Moores objektorienterade modell av programvaruutveckling i Figur 2 används resurser för att producera produkter. Det finns inom programvaruområdet standarder för resurser såsom:

- Terminologi och taxonomi
- Notation
- Tekniker
- Produkter för processinformation
- Bibliotek för återanvändning
- Verktyg

Produkter

Målet för en stor del av programvaruindustrin är att producera produkter av hög kvalitet. Att avgöra programvaruprodukters kvalitet är emellertid så svårt att de flesta har undvikit att försöka. De allra flesta standarder har istället fokuserat på att göra utvecklingsprocessen så bra som möjligt och att den sedan ska resultera i produkter av hög kvalitet. JTC1/SC7 har tagit fram en samling standarder som dock inte är helt okontroversiell. Standarderna kan grupperas enligt nedan:

- Standarder för produktevaluering
- Standarder för produktkaraktäristik
- Standarder för paketering av programvaruprodukter

Processer

Processer för programvaruutveckling bör ses som specialfall av processer för generell systemutveckling. Det finns standarder som beskriver hela livscykeln för

programvaran, med alla dess processer, och standarder som beskriver enskilda processer i större detalj.

Amerikanska militären var först ut med att framställa standarder för programvarans livscykel. Efter att man under 90-talet fattat beslut om att i stor utsträckning gå över till kommersiella standarder, så finns det idag inga militära standarder för programvaruutveckling kvar. Den kunskap som fanns lagrad i standarder som Mil-Std-498 har varit input vid framtagningen av den amerikanska anpassningen av ISO/IEC 12207. Idag handlar det mycket om 12207 (se Tabell 4) och standarder som stöder dess olika processer.

Det finns liksom för systemutveckling även standarder för utvärdering av programvaruprocesser. SEI SW-CMM och tidigare nämnda SPICE är två av de mest erkända för detta ändamål. För att stödja komponentbaserad utveckling pågår nu ett projekt med namnet OOSPICE. Projektet fokuserar på processer, teknologi och kvalitetsaspekter av komponentbaserad programvaruutveckling [OOSpice].

I Tabell 5 finns ett antal intressanta alternativ till 12207 uppräknade.

Tabell 4. Standarder för 12207:s livscykelprocesser

Standard	Titel	År
ISO/IEC 12207	Information technology – Software life cycle processes	1995
ISO/IEC 12207 /Amd 1	Information technology – Software life cycle processes – Amendment 1	2002
ISO/IEC TR 15271	Information technology – Guide for ISO/IEC 12207 (Software Life Cycle Processes)	1998
ISO/IEC 14759	Software engineering – Mock up and prototype – A categorization of software mock up and prototype models and their use	1999
IEEE/EIA 12207.0	Information technology – Software life cycle processes	1996
IEEE/EIA 12207.1	Guide for information technology – Software life cycle processes – Life cycle data	1997
IEEE/EIA 12207.2	Guide for information technology – Software life cycle processes – Implementation considerations	1997

ISO/IEC 12207

Utvecklandet av ISO/IEC 12207 utgick från IEEE 1074 och Mil-Std-498 och var färdigt 1995. 12207 beskriver de olika ingående processerna i en komplett programvarulivscykel och hög-nivårelationerna som styr processernas samverkan. Standarden täcker programvarans livscykel från konceptualisering av idéer till tillbakadragande. 12207 är tänkt att vara oberoende av teknologier, metodik och användbar för alla sorters livscykelmodeller. Istället föreskrivs användaren att själv välja en processmodell [Moore98].

Alla processer är hierarkiskt uppbyggda med aktiviteter som ingår i processerna och arbetsuppgifter som ingår i aktiviteterna. Följande processer finns:

- Primary Processes
  - Acquisition
  - Supply
  - Development
  - Maintenance

  - Operation
- Supporting Processes
  - Documentation
  - Configuration Management
  - Quality Assurance
  - Verification
  - Validation
  - Joint Review
  - Audit
  - Problem Resolution
- Organizational Processes
  - Management
  - Infrastructure
  - Improvement
  - Training

Standarden är skriven på ett sätt så att den lämpar sig väl för två-partsöverenskommelser, men ska även vara användbar vid de minst formella överenskommelserna [Moore98].

Det finns även två guider till ISO/IEC 12207. Den allmänna ISO/IEC TR 15271 är till hjälp vid applicerandet av 12207 i en organisation. Guiden beskriver instanser för vattenfallsmodellen, spiralmodellen och evolutionär modell.

ISO/IEC TR 14759 är speciellt inriktad på tillämpning av 12207 på fullskale-modeller och prototyper.

ISO/IEC 12207 Amd 1

Under 2002 publicerades ISO/IEC 12207 / Amd 1, som är en tillfällig revision av ISO/IEC 12207. Revisionen sammanjämkar krav från standarder som redan finns eller som är under utveckling av SC7, exempelvis ISO/IEC TR 15504. Dessutom har erfarenheter från användandet av ISO/IEC 12207 har resulterat i lärdomar som är värdefulla i uppdateringsprocessen. [Techstreet]

IEEE/EIA 12207.0, .1, .2

IEEE/EIA 12207 är den amerikanska anpassningen av den internationella ISO/IEC 12207. Man har i den valt att lägga mer fokus på organisationsnivån och mindre på projektnivå. Dessutom har all erfarenhet från de militära standarderna använts, nu senast från den demilitariserade J-Std-016 [G-34] som nyligen drogs tillbaka. Den amerikanska anpassningen är emellertid fullt kompatibel med den internationella.

Tabell 5. Alternativa standarder för livscykelprocesser för programvara

Standard	Titel	År
ECSS-E-40A Software	Software engineering standards	1999
IEC 60880	Software for computers in the safety systems of nuclear power stations	1986
IEEE Std 7-4.3.2	Standard criteria for digital computers in safety systems of nuclear power generating stations	1993
RTCA DO-178B	Software considerations in airborne systems and equipment certification	1992

ECSS-E-40A Software

Eftersom standarderna från ECSS ska ersätta standarderna med beteckningen PSS (Procedures Standards and Specifications) ersätter ECSS-E-40A motsvarande PSS-05-0 Software Engineering Standards.

Denna standard är avsedd för utveckling av programvara som ingår i rymdtillämpningar, antingen det gäller mark, uppskjutning eller rymd. Standarden beskriver en livscykelmodell, som anpassats till livscykelprocesserna i ISO/IEC 12207 i uppföljaren ECSS-E-40B Software.

Det aktuella projektet har ansvar för att anpassa standardmodellen genom att ansätta en livscykelmodell. Tre exempel på ansatser finns beskrivna: vattenfall, inkrementell leverans och evolutionär utveckling. Standarden baserades från början på ett antal IEEE-standarder. Den är inte avsedd att användas för framtagning av produkter till en massmarknad.

IEC 60880

I syfte att ta fram programvara med hög tillförlitlighet för användning i kärnkraftverks säkerhetssystem kan IEC 60880 användas. Standarden ställer krav på varje steg i utvecklingen av programvara, inklusive design, utveckling, kvalitetssäkring och drift, men även på dokumentation för varje steg. Det finns även en andra del som tar upp skydd mot vanliga felorsaker, användning av programvarubaserade verktyg och användning av förutvecklad programvara.

IEEE 7-4.3.2

Denna standard spelar en central roll i att integrera amerikanska standarder för datorer i kärnkraftsverks säkerhetssystem. Den innehåller hänvisningar till andra standarder för programvaruutveckling som skall, alternativt bör, användas.

RTCA DO-178B

Syftet med denna standard är att tillhandahålla riktlinjer för produktion av programvara för luftburna system och utrustning som utför sin avsedda funktion med ett beroende av en säkerhetsnivå som överensstämmer med flygduglighetskrav. Den behandlar processerna som bör användas för konstruktion av programvara som används i luftburna system [Moore98].

Denna standard är särskilt avsedd att användas vid certifiering av flygburen utrustning och används för detta ändamål både i USA och Europa.

Kunder

Standarderna som passar under denna rubrik passar i regel även under någon annan rubrik. De modeller för programvaruutveckling som antagits av IEEE SESC och av JTC1/SC7 utgår från att det alltid finns en kund som beställare av en programvaruprodukt. 12207 är också baserad på förutsättningen att det finns en beställare och en leverantör.

2.3.5 Standarder för mekanik- respektive elektronikutveckling

Som tidigare nämnts är de sammanhang och objekt som identifierades för programvaruutveckling säkerligen relevanta även för mekanik- och elektronikutveckling. Inom dessa discipliner har jag haft mycket svårt att finna ramverk för produktutveckling med betoning på högnivå-beskrivningar av livscykelprocesser.

Det finns säkert ett flertal orsaker till det. Standardisering av utvecklingsprocesser har inom de aktuella disciplinerna gjorts på bransch- och ännu mer på företagsnivå. Därför finner man inte några generiska processtandarder hos de största standardiseringsorganisationerna.

Programvaruutveckling som är en jämförelsevis omogen ingenjörsdisciplin har upplevt ett akut behov av att strukturera sitt arbete efter principer som bevisligen fungerar. Orsakerna till det är bland andra att programvaruprodukter är mer komplexa än de flesta andra produkter. Ansvaret att skapa standarder för programvaruutveckling har i stor utsträckning hamnat på standardiseringsorganisationerna. Att det blivit så beror antagligen mycket på kultur och tradition.

Nedan nämner jag några exempel på standarder som jag funnit och som har åtminstone någon relevans.

Elektronikutveckling i syfte att uppnå höga krav på tillförlitlighet kan göras i enlighet med MIL-HDBK-338B Electronic Reliability Design Handbook.

ECSS har tagit fram standarder för mekanikutveckling med inriktning på rymdapplikationer.

Världens tre största bilkoncerner, DaimlerChrysler, Ford och General Motors använder en gemensam standard för leverantörer som heter Quality System Requirements QS-9000, eller kort och gott QS-9000. Standarden är baserad på 1994 års version av ISO 9001 och innehåller dessutom särskilda krav som gäller för bilindustrin. ISO-samhället betraktar tilläggen i QS-9000 som bilindustrins tolkningar av ISO 9001. [QS-9000]

Nedan räknar jag upp några exempel på standarder, som man kan undersöka närmare. Alla har någon anknytning till säkerhet.

– IEC 62061, Safety of Machinery – Functional Safety of Safety-related Electrical, Electronic and Programmable Electronic Control Systems
– SS-EN 954-1:1996 Maskinsäkerhet – Säkerhetsrelaterade delar i styrsystem – Del 1: Allmänna principer för konstruktion
– EN/ISO 13849-1:1999 Safety of Machinery – Safety-related Parts of Control Systems – Part 1: General Principles for Design (Det finns även en guide vid namn EN/ISO 13849-100:2002.)

2.3.6 Branschspecifika standarder

Många standarder som berör produktutvecklingsprocessen är mycket generella. Det finns därför många gånger ett behov att anpassa dem [Moore] till den specifika världsdelen, nationen, branschen, företaget eller projektet. Detta är orsaken till att många branscher tar fram egna standarder. Att ta fram internationella standarder tar ofta väldigt lång tid, och därför vill man heller inte alltid vänta.

I vissa fall har man utnyttjat och samlat den industriella erfarenheten när det t ex gäller att bygga bilar. Vissa branscher har mycket höga krav på att deras produkter ska vara pålitliga eller säkra, t ex flyg- och rymdindustrin.

3 Metodik för programvaruutveckling

Det finns idag delade meningar om hur programvara bäst bör utvecklas. De traditionella processerna har de senaste åren fått konkurrens av en ny skola som kallar sina processer och metoder för lättviktiga eller snabba. En av de centrala tankarna i dessa metoder är att allt som görs i en utvecklingsprocess inte måste dokumenteras. Endast de dokument som är verkligt värdefulla ska tas fram.

Kanske går dessa metoder att tillämpa inom andra teknikgrenar. Kanske bör de inte tillämpas alls. För att ta reda på detta börjar detta kapitel med att beskriva innehållet i XP. Därefter görs en jämförande analys av utvecklingsarbetet i processtandarden ISO/IEC 12207 och XP.

3.1 Introduktion till eXtreme Programming

Detta avsnitt innehåller ett referat av boken eXtreme Programming Explained av Kent Beck [Beck00]. Beck är skaparen av programvaruutvecklingsmetoden eXtreme Programming (XP). Orsaken till att metoden har fått namnet extrem programmering är att den är baserad på förnuftiga principer, som tillämpas på en extrem nivå.

Om granskningar är bra, så varför inte granska hela tiden (parprogrammering). Om testning är bra, så varför inte testa hela tiden (enhetstestning), till och med kunderna testar (funktionella tester). Om design är bra, så varför inte göra design till en daglig syssla för programmerarna (refaktorisering). Etcetera.

XP består av tolv praktiker och ett antal strategier. Praktikerna härleds från fyra grundläggande värderingar, femton principer som är härledda från värderingarna, samt fyra basaktiviteter. Detta kapitel är uppdelat i de tre avsnitten problemet, lösningen och implementation. I det första beskrivs grundläggande frågeställningar för programvaruutveckling, värderingarna, principerna och basaktiviteterna. I det andra avsnittet beskrivs XP:s praktiker och strategier. Det tredje avsnittet ger kunskaper om hur XP bör implementeras i en organisation eller ett projekt.

3.1.1 Problemet

Detta avsnitt innehåller bakgrundsinformation till XP:s metodik. Först beskrivs vad som enligt Beck är det grundläggande problemet vid programvaruutveckling nämligen risk. Därefter beskrivs XP:s fokus som är utvecklingsarbetet. Vidare beskriver Beck två sätt att se på ekonomin i ett projekt. XP ger utvecklaren stora valmöjligheter att styra projektet så att det blir lönsamt.

Därefter beskrivs betydelsen av de fyra variabler som styr ett programvaruprojekt nämligen kostnad, tid, kvalitet och omfång. I efterföljande avsnitt beskrivs

Becks syn på kostnaden för ändringar. Därefter tas Becks liknelse mellan bilkörning och programvaruutveckling upp. Vidare beskrivs och motiveras de fyra värderingar som ligger till grund för XP: kommunikation, enkelhet, återkoppling och mod. Från dessa värderingar härleder Beck 15 principer som även de ligger till grund för XP.

I det följande avsnittet beskrivs XP:s huvudsakliga aktiviteter som enligt Beck är eller borde vara de grundläggande aktiviteterna vid all programvaruutveckling: koda, testa, lyssna och designa.

Det grundläggande problemet

Det grundläggande problemet vid programvaruutveckling är projektets risker. Det finns många saker som kan gå snett i ett programvaruprojekt. Det finns olika risker som innebär problem med produkten och dess leverans. XP är en metodik som oftast kan förhindra att riskerna blir verklighet.

XP:s fokus

XP:s hjärtslag är utvecklingsarbetet. Det är när en ingenjörsmässig arbetsuppgift utförs. Arbetsuppgifter[5] är den minsta enheten som schemaläggs. Utvecklingscykeln innebär att:

– Par av programmerare programmerar tillsammans.

– Utvecklingen drivs av tester. Först skriver man tester, sedan programmerar man och därefter kör man testerna. Alla eventuella defekter rättas så att testerna går bra. När man inte kan komma på några fler viktiga tester är man färdig.

– Paren skriver inte bara tester, de ligger också bakom systemets design. De genomför analysen, designen, implementationen och testningen av systemet.

– Integration och integrationstestning följer omedelbart efter utveckling av en funktion.

Ekonomi för programvaruutveckling

Det finns enligt Beck några strategier för att maximera vinsten i ett programvaruprojekt.

– Man kan spendera mindre pengar, vilket är svårt eftersom alla startar med ungefär samma verktyg och kompetenser.

– Man kan tjäna mer pengar, vilket endast är möjligt om man har en excellent marknads- och försäljningsorganisation.

– Man kan tjäna pengar tidigare och spendera pengar senare i projektet.

– Man kan öka sannolikheten att projektet överlever så att man ökar chanserna att få den stora förtjänsten i slutet.

Ett annat sätt att se på ekonomin i programvaruprojekt är att se på de olika valmöjligheter som finns. Dessa är:

– Valmöjligheten att överge – om man kan få en förtjänst av projektet även om det avbryts är det bra. Ju större den förtjänsten är desto bättre.

[5] Task

– Valmöjligheten att ändra – en projektledningsstrategi som tillåter att inriktningen på projektet ändras är bra. Ju kraftigare och frekventare projektet kan ändras desto bättre.
– Valmöjligheten att skjuta upp – möjligheten att skjuta upp investeringar är bra. Ju mer man kan avvakta med och ju längre desto bättre.
– Valmöjligheten att växa – om en marknad skjuter i höjden vill man kunna växa snabbt. En projektledningsstrategi som ger möjligheten att skala upp produktionen är bra. Ju längre och snabbare projektet kan växa desto bättre.

XP är tänkt att vara en strategi som ger projektet stora möjligheter att göra val under tiden det fortlöper. Följaktligen är XP lämpligt att använda när kraven är vaga eller föränderliga. Först implementeras de funktioner som man vet att kunden vill ha. När kravbilden klarnar kan ytterligare funktioner implementeras. Om det är osäkert vilket värde en viss funktion ger kunden kan det vara mer lönsamt att vänta med att implementera den, annars riskerar det att bli bortkastat arbete.

Variabler

Det finns fyra av varandra beroende variabler som styr utvecklingen av programvara: kostnad, tid, kvalitet och omfång. Ofta finns det externa krav på någon av dessa variabler. Exempelvis kan kunden sätta upp ett leveransdatum som inte får överskridas. Chefen kan sätta upp ett kvalitetskrav som måste uppfyllas. Den variabel som är enklast att variera är omfånget, som avgör vilken funktionalitet som skall levereras.

I XP bestäms vilken funktionalitet som ryms i projektet utifrån kraven på kostnad, tid och kvalitet. Om tiden och pengarna inte räcker till för att leverera som planerat ska inte kvaliteten sänkas, utan funktionaliteten ska minskas. På grund av detta är det viktigt att implementera den högst prioriterade funktionaliteten först. Erfarenhet av projektplanering bör också resultera i ständigt bättre planer.

Kostnad för ändring

En klassisk tanke från programutvecklingens område är att ändringar kostar mer pengar ju senare i projektet de kommer. Kostnaden för ändringar stiger exponentiellt med tiden. Att finna buggar så tidigt i projektet som möjligt blir därför en prioriterad aktivitet.

XP är baserat på förutsättningen att detta inte längre är sant. Ny teknologi har enligt Beck resulterat i att kostnaden för ändringar planar ut med tiden och når en nivå då de inte längre ökar. Objektteknologi har lett till designer som är mycket mer flexibla och förändringsbara. Även objektdatabaser har ökat flexibiliteten.

Beck har erfarenhet av den utplattade kostnad/tid-kurvan i projekt när följande principer följts:

– En enkel design, som inte innehåller något extra för tänkta tillägg och ändringar.
– Automatiserade tester, som säkerställer att ändringar blir grundligt testade.
– En positiv inställning till ständiga förbättringar av designen, så att man vågar göra ändringar när det behövs.

Att köra bil

Ett programvaruprojekt är enligt Beck som att köra bil. Det räcker inte att i början ställa in ratten i rätt riktning och hålla den där. Man måste vara uppmärksam hela tiden och ständigt korrigera riktningen för att hålla sig på vägen. Förändringar av krav, design, affärer, teknologi och människor kommer att infinna sig i ett programvaruprojekt. Det viktiga är att hantera förändringarna på rätt sätt.

Värderingar

XP är baserat på fyra värderingar: kommunikation, enkelhet, återkoppling och mod.

När problem uppstår i ett projekt är det ofta på grund av bristande kommunikation. Någon glömde att berätta något viktigt för någon annan. Någon ställde fel frågor. XP använder sig av särskilda praktiker för att få till stånd kommunikation. Effekten av enhetstestning, parprogrammering och storleksuppskattning av arbetsuppgifter är att programmerare, kunder och chefer måste kommunicera med varandra.

Det är bättre att göra en så enkel design som möjligt för dagens behov, än att göra en komplicerad för morgondagens behov, som riskerar att aldrig användas. Enkelhet främjar och främjas också av god kommunikation.

Återkoppling är avgörande för att få reda på hur projektet fortlöper. Genom att skriva och köra tester fås kortsiktig återkoppling inom minuter eller dagar. Återkoppling i det längre tidsperspektivet, veckor och månader, fås t ex genom att sätta systemet i produktion så fort som möjligt. Först när det är gjort kommer vissa insikter om systemet att kunna göras. Dessa kan leda till förbättringar.

Mod är den fjärde av värderingarna. Mod att pröva nya idéer, mod att slänga dålig kod, etcetera. Utan de andra värderingarna låter det som hacking, men i samklang med dem är mod oerhört värdefullt. Det finns inget som säger att man aldrig programmerar in sig i ett hörn som man inte kan komma ut ur. Då krävs mod för att gå en annan väg.

En djupare liggande värdering med avgörande betydelse för vilket projekt som helst är respekt. Om inte teammedlemmarna bryr sig om varandra eller projektet kommer det inte att gå bra.

Grundläggande principer

De lite vaga värderingarna i XP konkretiseras med hjälp av ett antal principer. Nedan följer först de fem mest centrala principerna, därefter ytterligare tio.

Snabb återkoppling

Ju snabbare man kan få återkoppling desto större blir möjligheterna att lära sig något av den. Det är bättre att få återkoppling efter sekunder och minuter än efter dagar, veckor eller månader.

Antag enkelhet

Behandla varje uppgift som om den vore enkel att lösa. Den tid som man sparar på flertalet av uppgifter, som faktiskt är enkla att lösa, resulterar i massor av tid till de få uppgifter som verkligen är svåra. Programmerare har fått lära sig att planera sitt arbete och designa för återanvändning. XP förespråkar att lösa dagens problem i dag och lita på att man kan lösa morgondagens problem imorgon.

Inkrementell förändring
Små förändringar har större chans att bli lyckosamma än stora. I XP förändras designen i små steg, likaså planen. Även införandet av XP bör ske inkrementellt.

Välkomna förändring
Att välkomna förändring är att anta en strategi som innebär att så många valmöjligheter som möjligt bibehålls, samtidigt som de mest pressande problemen löses först.

Kvalitetsarbete
Ingen tycker om att göra ett slarvigt jobb. Arbetet måste vara av hög kvalitet. Om inte ser det mörkt ut för projektet.

Lär ut lärande

Fokus ligger inte på att lära ut att man skall testa XYZ utan att lära ut strategier för hur man lär sig hur mycket testning man bör göra.

Liten initial investering

För mycket resurser för tidigt i ett projekt är en bra förutsättning för katastrof. Det är inte bra att ha för små resurser, men en begränsad budget motiverar till att göra ett bra jobb. Ofta klarar man sig på mindre än vad som känns helt tillfredsställande.

Spela för att vinna

Mycket av dagens metoder för programvaruutveckling går ut på att inte förlora snarare än på att vinna. Man följer sina processer noggrant för att undvika att få skulden för ett eventuellt misslyckande. I XP görs allt som hjälper teamet att vinna och inget som inte hjälper teamet att vinna.

Konkreta experiment

Varje gång ett beslut fattas bör det testas om det fungerar. Om många beslut fattas utan att testas ökar riskerna. Därför bör t ex en designomgång avslutas med en serie experiment kring de frågor som uppstått och inte med en färdig design.

Öppen, ärlig kommunikation

Fundamentalt. Om teammedlemmarna är rädda för konflikter riskerar det att resultera i stora problem.

Arbeta med människors instinkter, inte emot dem

Om programmeraren inte får någon kortsiktig vinst av att utföra en viss aktivitet spelar det mindre roll om det är fördelaktigt i det långa loppet. När tidspressen sätter in är risken stor att det blir bortprioriterat.

Accepterat ansvar

Människor som blir åtsagda vad de ska göra kommer att bli frustrerade och uttrycka den frustrationen till skada för projektet. Det är bättre att låta programmerarna själva acceptera ansvaret för olika uppgifter än att ge dem ansvaret.

Lokal anpassning

Principerna bakom XP måste anpassas till det specifika teamet som ska använda processen. Man kan inte bara läsa en bok och sedan veta hur man utvecklar programvara.

Res lätt

Ett XP-team är alltid redo till snabba förändringar. Om man snabbt ska kunna ge sig ut på en resa kan man inte ha för mycket bagage. Därför måste artifakterna vara få, enkla och värdefulla. Det är framförallt tester och kod som ger kunden värde.

Uppriktiga mätningar

Mätningar ska göras till den nivå av noggrannhet som mätinstrumenten tillåter. Det är bättre att säga att det här tar ungefär två veckor än att säga att det tar 14.176 timmar om man inte har något tillförlitligt sätt att estimera så noggrant.

Grundläggande aktiviteter

Beck beskriver de fyra grundläggande aktiviteterna vid programvaruutveckling: koda, testa, lyssna och designa.

Kodning

Det viktigaste med att koda är att man lär sig saker. Att formulera tankar och idéer i kod är det bästa sättet att lära sig om de fungerar i verkligheten. Kod är också utmärkt för att kommunicera sina idéer till andra. När de ser koden förstår de vad man försöker förklara. Eftersom kod är den enda artefakt ett programvaruprojekt absolut inte kan vara utan, är det viktigt att försöka utnyttja den till så mycket som möjligt.

Testning

Det är inte lätt att veta om man har implementerat det man verkligen tänkte implementera. Eller om det man tänkte var det man borde ha tänkt. Tester är det bästa sättet att försäkra sig om saken. Tester öppnar också möjligheten för att kontrollera vad som har gjorts utan att låsa fast vid en viss implementation. Automatiserade tester kan inte bara göras för funktionella krav, utan även för en del icke-funktionella krav, t ex prestanda.

Tester är både en resurs och ett ansvar. Programmeraren har ansvar att skriva tester för alla tänkbara testfall. När programmeraren inte längre kan komma på några tester som skulle fallera är kodningen färdig. Att skriva tester innan man programmerar, ger fördelar både på lång och kort sikt. På lång sikt visar det sig att programmet kan överleva längre om testerna körs och underhålls. Med tester kan man göra ändringar i programkoden under en längre tidsperiod än om man saknar tester. Detta med bibehållet förtroende för systemet.

På kort sikt märker man också fördelar. Det är roligare att programmera med testerna på plats. Man kan programmera med större självförtroende. Det går alltid att trycka på en knapp som kör alla automatiserade testerna som bekräftelse på att man gjort rätt. Att testa och programmera ihop går också fortare än att bara programmera. Orsaken till den förhöjda produktiviteten är att tiden för debugging reduceras kraftigt när testerna körs.

Det finns givetvis en fara med att förlita sig på sina tester, om de inte är tillräcklig bra eller heltäckande. Med tiden lär man sig emellertid hur mycket tester som behövs för att uppnå önskad kvalitet på programvaran. I XP används framförallt två typer av tester. Dels enhetstester som skrivs av programmerarna för att övertyga sig själva om att programmet fungerar, dels funktionella tester som skrivs (eller åtminstone specificeras) av kunderna för att de ska kunna försäkra sig om att programvaran fungerar som tänkt.

Lyssnande

Programmerares kunskaper om affärer är begränsad. Därför måste programmerare lyssna till kunderna så att de förstår vilka affärsbehoven är. Denna kommunikation behöver styras upp på något sätt, så att den blir av och sker på rätt sätt.

Design

Design innebär att skapa en struktur som organiserar logiken i systemet. Bra design organiserar logiken i systemet så att en ändring i en del av systemet inte kräver ändringar på andra delar. Bra design ser till att varje logikdel endast återfinns på en plats i systemet. Bra design placerar logiken nära det data som den opererar på. Bra design tillåter utökningar av systemet eftersom ändringar bara behöver göras på ett ställe.

Om designen inte är bra kommer det till slut att bli omöjligt att bygga vidare på systemet. En ändring på ett ställe kommer att kräva ändringar på andra ställen givna av komplexa samband. I XP ingår design i programmerarnas dagliga arbete.

”Så du kodar eftersom om du inte kodar, har du inte gjort någonting. Du testar eftersom om du inte testar vet du inte när du är färdig med kodningen. Du lyssnar eftersom om du inte lyssnar vet du inte vad som ska kodas eller vad som ska testas. Och du designar så att du kan fortsätta koda och testa och lyssna i det oändliga.” [Beck00]

3.1.2 Lösningen

I detta avsnitt beskrivs vad XP består av. De praktiker som tillämpas i XP bygger på de värderingar, principer och aktiviteter som presenterades i föregående avsnitt. Vanliga invändningar mot praktikerna beskrivs, men också argumenten för att de verkligen fungerar. Därefter genomgås XP:s strategier för ledning och faciliteter. Vidare att affärsmässigt och tekniskt ansvar skall delas upp. Därefter XP:s strategier för planering, utveckling, design och testning.

Praktiker

Råmaterialet till en nya programvarudisciplin (XP) är enlig Beck:

– Berättelsen om att lära sig köra bil

– De fyra värderingarna – kommunikation, enkelhet, återkoppling och mod

– Principerna
– De fyra basaktiviteterna – kodning, testning, lyssnande och design

Dessa element ligger till grund för praktikerna i XP. Det finns svagheter hos de flesta av dessa praktiker, men de samverkar så att svagheter hos en praktik uppvägs av styrkan hos andra.

Planeringsspelet

Bestäm snabbt innehållet i nästa release genom att kombinera tekniska uppskattningar och affärsmässiga prioriteringar. Uppdatera planen när den inte längre stämmer med verkligheten.

Små releaser

Börja med att sätta programvaran i produktion så fort som möjligt. Släpp sedan regelbundet nya versioner mycket tätt.

Metafor

Allt utvecklingsarbete ska utgå från en gemensam ”metafor”, vilket är en beskrivning av hur hela systemet är tänkt att fungera. I XP ersätter metaforen mycket av det som brukar kallas arkitektur.

Enkel design

Systemet skall alltid vara designat så enkelt som möjligt. All extra komplexitet som upptäcks tas bort. Om man vet att framtiden är osäker är det galet att implementera funktionalitet som man inte vet om man kommer att behöva. Implementera istället för dagens behov.

Testning

Programmerare skriver enhetstest som måste klaras av för att utvecklingsarbetet skall fortsätta. Det finns inget sådant som en funktion utan ett automatiserat test. Kunderna skriver funktionella tester som visar att en funktion är färdig. Man måste inte skriva ett test för varje metod man skriver. Det räcker med de produktionsmetoder som kan krascha.

Refaktorisering

Programmerare omstrukturerar program för att göra dem enklare, ta bort redundans, förbättra kommunikationen och utöka flexibiliteten. Refaktorisering ska göras när systemet ber om det. Om systemet exempelvis kräver att du kopierar kod för att få en ny funktion att fungera då är det dags att refaktorisera.

Parprogrammering

All produktionskod skrivs av två programmerare framför en dator. Den person som handhar tangentbordet och musen för tillfället tänker på hur den aktuella funktionen bäst skall implementeras. Den andre tänker mer strategiskt på om det finns något annat sätt att lösa problemet och om det finns några fler testfall som

behöver skrivas. Vid parprogrammering arbetar man inte alltid med samma partner utan byter ofta och arbetar kanske med alla.

Kollektivt ägande

Vem som helst kan ändra vilken kod som helst i systemet när som helst och har dessutom ansvar att göra det. Jämfört med att ingen äger koden och med individuellt ägande av koden leder kollektivt ägande till att koden utvecklas snabbare samtidigt som den förblir stabil.

Kontinuerlig integration

Integration bör ske efter några timmar eller efter en dag, så fort en arbetsuppgift är slutförd. Ett sätt att göra det på är att ha en dator dedikerad till integration. När den är ledig sätter sig ett par som slutfört ett kodavsnitt och lägger in det i systemet. Därefter körs de automatiska testerna. Om testerna inte klaras av är det ingen diskussion om vem som ska rätta buggarna. Man lämnar inte maskinen förrän 100% av testerna fungerar.

40-timmars vecka

Den enkla regeln i XP är att inte arbeta mer än ungefär 40 timmar varje vecka och att aldrig jobba övertid mer än en vecka i taget. Om man ser att man inte kan nå sina mål utan att jobba övertid ännu en vecka har man ett problem i projektet som inte kan lösas med mer övertidsarbete.

Kund på platsen

Utvecklingsteamet måste ha en kund som sitter tillsammans med teamet på heltid för att svara på frågor. Kunden ska vara en person som kommer att använda systemet när det är färdigbyggt. Eftersom det är svårt att generera 40 timmar frågor varje vecka kan personen ändå hinna med mycket av sina normala arbetsuppgifter.

Kodningsstandarder

Om programmerarna ska kunna läsa och ändra i varandras kod är det nödvändigt att komma överens om kodningsstandarder. Dessa ska kräva minimalt arbete och betona kommunikation.

Praktikernas svagheter och hur de övervinns

Ingen av ovan nämnda praktiker är ny. Alla har använts tidigare, men ersatts av andra overhead-krävande på grund av deras svagheter. Kollapsen av den exponentiella kostnadskurvan har fört dessa praktiker tillbaka till spelplanen. Detta avsnitt kommer att beskriva invändningarna mot dem, men också hur deras svagheter kan kompenseras av styrkorna hos andra praktiker. Detta avsnitt visar hur XP skulle kunna vara möjligt.

Planeringsspelet

Man kan inte starta utvecklingen av ett programvarusystem endast utifrån en grov plan. Man kan inte göra ständiga uppdateringar av planen, det skulle ta för lång tid och reta upp kunderna. Om inte:

– Kunderna uppdaterade planen själva utifrån uppskattningar från programmerarna.
– Man hade tillräckligt med plan för att visa kunden ungefär vad som var möjligt att hinna med.
– Man släppte versionerna tätt, så att en ändring i planen inte betydde mer än några veckor eller månaders förskjutning.
– En kund satt med teamet och snabbt kunde upptäcka tänkbara ändringar och tillfällen till förbättringar

Små releaser

Man kan inte starta produktionen av systemet redan efter några månader och sedan släppa nya versioner på daglig eller månatlig basis. Om inte:
– Planeringsspelet hjälpte till att prioritera de storys som var viktigast först, så att redan ett litet system skulle innebära värde för kunden.
– Man integrerade kontinuerligt så att kostanden för paketering skulle bli minimal.
– Testningen resulterade i ett ständigt buggfritt system, så att det inte krävdes en lång testperiod inför varje release.
– Man kunde göra en enkel design, som var tillräcklig för nuvarande system, men inte för resten.

Metafor

Man kan inte starta utvecklingen med endast en metafor. Den innehåller för lite detaljer och dessutom kan man ha fel idé. Om inte:
– Man snabbt kan få konkret återkoppling från tester och kod om metaforen fungerar i praktiken eller inte.
– Kunden är tillfreds med att prata om systemet som en metafor.
– Man refaktoriserar hela tiden för att förbättra förståelsen av metaforens innebörd.

Enkel design

Man kan inte utgå från en design som endast täcker dagens kod. Man skulle designa in sig själv i ett hörn utan möjlighet att ta sig ut. Om inte:
– Man var van vid refaktorisering, så att det inte var ett problem att göra ändringar.
– Man har en klar metafor, så att man kunde vara säker på att framtida ändringar konvergerade i rätt riktning.
– Man programmerade tillsammans med en partner så att man var säker på att man gjorde en enkel design och inte en dum design.

Testning

Det är omöjligt att skriva alla testerna som behövs. Programmerarna kommer inte att göra det. Om inte:

– Designen är så enkel som den kan vara, så att det inte är så svårt att skriva testerna.
– Man programmerade tillsammans med någon annan, som kom på fler tester när man själv inte gjorde det. Och som kunde rycka till sig tangentbordet när man vill skippa testerna.
– Man mår bra när man ser att alla testerna klaras av.
– Kunderna mådde bra när de såg alla testerna köras.

Refaktorisering

Det går inte att refaktorisera hela tiden. Det skulle ta för mycket tid, vara för svårt att kontrollera och få systemet att gå sönder. Det skulle vara möjligt om:

– Man är van vid kollektivt ägande så att man inte har något emot att göra ändringar överallt där det behövs.
– Man har kodningsstandarder så att man inte behöver formatera innan man refaktoriserar.
– Man programmerar i par, så man har större mod att ge sig på en svår ändring och sannolikheten att man förstör något är mindre.
– Man har en enkel design så att refaktoriseringen är enklare.
– Eftersom man har testerna, är det mindre risk att man förstör något utan att veta om det.
– Man har kontinuerlig integration så att om man förstörde någonting av misstag eller någon av refaktoriseringarna inte fungerade med någon annans arbete, skulle man få reda på det inom några timmar.
– Man är utvilad så man har mer mod och är mindre benägen att göra något fel.

Parprogrammering

Det går inte att skriva all produktionskod i par. Det skulle gå för långsamt. Tänk om två personer inte kom överens. Om inte:

– Kodningsstandarderna reducerade smågnabbet.
– Alla var pigga och utvilade, vilket skulle reducera olönsamma diskussioner.
– Paren skriver tester tillsammans först, vilket ger dem en gemensam förståelse för problemet innan de tar itu med det på allvar.
– Paren har metaforen att utgå ifrån när de ska göra grundläggande design.
– Paren arbetar inom en enkel design, så bägge kan förstå vad det är som pågår.

Detta ska jämföras med nackdelarna med att programmera på egen hand.

Kollektivt ägande

Man kan inte ha folk att ändra vad som helst. De skulle förstöra saker och kostnaden för integration skulle stiga i höjden. Om inte:

– Man integrerar ofta så att riskerna för problem minskar.
– Man skriver och kör tester så att risken att förstöra något av misstag minskar.
– Man programmerar i par vilket minskar risken för skador.
– Man följer en kodningsstandard, så att man inte börjar bråka om krullparenteserna.

Kontinuerlig integration

Man kan inte integrera efter endast några timmar. Det tar för lång tid och är förenat med alltför stora möjligheter att förstöra något. Om inte:

– Man kan köra testerna snabbt så att man vet att man inte har förstört något.
– Man programmerar i par så antalet strömmar att integrera är hälften så stort.
– Man refaktoriserar så det finns fler små delar vilket reducerar riskerna för konflikter.

40-timmarsvecka

Det räcker inte att arbeta 40 timmar i veckan. Man kan inte producera tillräckligt mycket affärsvärde på 40 timmar. Om inte:

– Planeringsspelet matar dig med mer värdefullt arbete att göra.
– Kombinationen av planeringsspelet och testerna minskar risken för otrevliga överraskningar som innebär att du får mindre tid än tänkt.
– Praktikerna i sin helhet hjälper dig att programmera i din toppfart, så att det inte går att göra det snabbare.

Kund på platsen

Man kan inte ha en kund som sitter med teamet heltid. Kunden kan göra större nytta någon annanstans i företaget. Om inte:

– De kan producera värde genom att skriva funktionella tester.
– De kan producera värde för projektet genom att göra småskalig prioritering och beslut kring omfånget åt programmerarna.

Kodningsstandarder

Programmerare är individualistiska och skulle inte ställa upp på att koda efter en gemensam standard och sätta krullparanterserna annorstädes än vanligt. Om inte:

– Chansen finns att genom XP vara med i ett vinnande lag.

Strategi för ledning

Det finns två diken vad gäller ledningsstrategier. Det ena är att ha en väldigt centraliserad styrning av projektet, vilket innebär att en person har visionen och fattar

alla beslut. En sådan strategi medför att det blir väldigt enkelt att fatta beslut, men det blir väldigt svårt att genomföra dem.

Det andra diket är att låta alla leda sig själva. Det skapar en kaotisk situation där alla riskerar att dra åt olika håll. En balans mellan dessa två extremer vore det bästa. Nedan beskrivs hur XP:s principer ligger till grund för ledningsstrategin.

Accepterat ansvar – Det är chefens ansvar att lyfta fram vad som behöver göras men inte att bestämma vem som ska göra det.

Kvalitetsarbete – Ett samspel mellan programmerare och chefer som är baserat på förtroende. Cheferna litar på att programmerarna vill göra ett bra jobb. Deras uppgift är att försöka hjälpa dem att gör det ännu bättre.

Inkrementell förändring – Chefen tillhandahåller vägledning hela tiden och inte bara ett tjockt dokument i början av projektet.

Lokal anpassning – Chefen bör ta initiativ till att anpassa XP till de lokala förutsättningarna och övervinna hinder i den lokala kulturen.

Res lätt – Chefen skall inte introducera massor av overhead, långa möten och statusrapporter. Det chefen kräver av medarbetarna ska gå fort för dem att fullfölja.

Uppriktiga mätningar – Chefen ska göra mätningar på en realistisk nivå av noggrannhet.

XP:s ledningsstrategi är snarare decentraliserad än centraliserad. Chefens uppgift är att leda planeringsspelet, insamla mätningar, se till att resultatet av mätningarna ses av rätt personer och ibland gå in och lösa situationer som inte går att lösa på ett decentraliserat sätt.

Mått

Ledningens grundläggande verktyg är mått. Det bästa sättet att förmedla information om enskilda projektmått till programmerarna är att sätta upp en tydlig plansch på väggen. Det är inte bra att ha för många mått på tapeten samtidigt, högst tre till fyra stycken. Några viktiga mått är projekthastigheten som är förhållandet mellan estimerad utvecklingstid och kalendertid, samt resultatet av enhetstesten (som alltid bör vara 100%).

Coachning

I XP är chefens uppgifter uppdelade i två roller: coachning och spårning. Coachens uppgift är inte att fatta alla beslut, utan att förmå medarbetarna att fatta kloka beslut. Coachen ska stötta de andra programmerarnas arbete, ta tag i svåra tekniska uppgifter, tänka på den långsiktiga refaktoriseringen och förklara processen för den övre ledningen.

En annan viktig uppgift är att införskaffa prylar och mat. Meningslösa prylar kan tilldelas viktiga meningar och mat är en bra kortsiktig lön för avklarade uppgifter.

Spårning

Den andra delen av ledning i XP är spårning. Det räcker inte att göra uppskattningar i början av projektet. Projektets status ska följas upp så att planeringsspelet kan fungera som tänkt.

Ingripande

Ibland kan chefen bli tvungen att gripa in och göra besvärliga förändringar i projektet. Det är viktigt att chefen är ödmjuk när han eller hon har tillåtit att något gått snett. Chefen kan bli tvungen att avskeda någon från projektet, initiera ändring av processen eller avsluta projektet.

Strategi för faciliteter

För att XP ska fungera bra ställs vissa krav på faciliteter och lokaler. Det ideala är att ha ett stort, öppet rum med bord för datorer mitt i, där programmeringen sker. På väggarna kan man ha vita tavlor och viktiga budskap upphängda. En liten personlig vrå, med utrymme för personliga saker och tillgång till telefon är bra för att tillgodose medarbetarnas behov av avskildhet. Ett konferensbord är bra och en hörna med soffor och leksaker för avkoppling. Meningen med att samla programmerarna nära varandra är att främja kommunikationen mellan dem.

Uppdelning av affärsmässigt och tekniskt ansvar

Om affärsfolket får bestämma över projektet kommer de att vilja styra de fyra variablerna som styr utvecklingsarbetet: tid, kostnad, kvalitet och omfång. Det kommer leda till att programmerarna får för mycket att göra och inte vet vad de ska prioritera.

Om däremot teknik-folket får bestämma kommer resultatet att bli det samma. Satsningen på omfånget kommer att bli för stort i förhållande till vad det ger tillbaka. Teknik-folket vill gärna testa nya teknologier vilket tar tid och innebär stora risker. Ansvaret bör fördelas så att de olika grupperna kan bidra med sin expertis.

Strategi för planering

De principer som ligger till grund för planeringsstrategin i XP är:

- Planera endast den närmaste iterationen detaljerat. Efterföljande iterationer vinner inget på detaljerad planering, utan kan planeras i grova drag.
- Eftersom ansvar inte kan ges utan måste accepteras kan inte projektledaren göra en plan och säga, detta ska vi göra och det tar så här lång tid. Projektledaren måste fråga teamet om att ta ansvar och sedan lyssna på svaret.
- Den som är ansvarig för att implementera är också ansvarig för att uppskatta hur lång tid det tar.
- Ignorera beroenden mellan olika delar av systemet. Planera som om delarna kunde göras i godtycklig ordning.
- Planera för prioriteringar eller för utveckling. När man planerar för att kunden ska kunna göra prioriteringar krävs betydligt lägre detaljnivå än vid planering för implementation.

Planeringsspelet

Planering involverar två deltagare, Affär och Utveckling. Relationerna mellan dessa grupper kan många gånger vara dåliga. Det bästa är om relationen bygger på ömsesidigt förtroende. Planeringsspelet innehåller ett antal regler för att underlätta relationerna.

Målet för planeringsspelet är att maximera värdet av programvaran. Från det beräknade värdet dras utvecklingskostnaden och risken av.

Strategin är att investera så lite pengar som möjligt för att få den mest värdefulla funktionaliteten i produktion så fort som möjligt.

Delarna i planeringspelet är story-korten.

Spelarna som deltar är Affär och Utveckling. Affär kan utgöras av beställaren av systemet eller andra som representerar användarna av det tänkta systemet.

De olika dragen som kan göras i spelet beror på vilken fas man befinner sig i. Det finns tre cykliska och överlappande faser i spelet. Utforskning – som undersöker vilka nya saker systemet skulle kunna göra. Åtagande – då en delmängd av alla möjliga funktioner väljs ut för att implementeras. Styrning – som innebär att ändra inriktningen när verkligheten krossar planen.

Under utforskningsfasen finns det tre drag som kan göras:

– Affär kan skriva en story på ett indexkort, med en kort beskrivning av en feature.
– Utveckling kan uppskatta arbetsåtgången för att implementera storyn.
– Affär kan dela upp en story i flera mindre storys, om Utveckling inte klarar av att tidsuppskatta den eller de inser att någon del av storyn har högre prioritet än de övriga

Under åtagandefasen finns det fyra drag som kan göras:

– Affär sorterar alla storys efter värde i tre olika högar: (1) de som krävs för att systemet ska fungera, (2) de som är mindre nödvändiga men som skulle ge stort affärsvärde och (3) de som skulle vara trevliga att ha.
– Utveckling sorterar alla kort efter risk i olika högar: (1) de vilkas tidsåtgång kan uppskattas precist, (2) de som kan uppskattas någorlunda väl och (3) de vilkas tidsåtgång inte kan uppskattas.
– Sätt hastighet. Utveckling berättar för Affär hur fort teamet kan programmera varje kalendermånad.
– Affär bestämmer omfång för nästa release utifrån release-datum och utvecklingshastighet. Alternativt kan omfånget först bestämmas och releasedatumet beräknas utifrån det.

Under styrningsfasen finns fyra drag som kan göras:

– Iteration – I början av varje iteration, varje eller var tredje vecka, utväljer Affär en iteration som innehåller de storys som är mest värdefulla att implementera. Den första iterationen måste resultera i ett körbart program.

– Återställande – Om Utveckling inser att de har överskattat sin utvecklingshastighet ber de Affär att göra en ny prioritering.
– Affär kan skriva en ny story om de inser att det behövs. Efter uppskattning kan den bytas in mot motsvarande krävande storys.
– Om Utveckling märker att planen inte längre stämmer med verkligheten kan uppskattningarna göras om och hastigheten sättas på nytt.

Iterationsplanering

Iterationsplanering har tydliga likheter med planeringsspelet men här görs planeringen på kortare tidshorisont och spelarna är de individuella programmerarna. Faserna och dragen är liknande de i planeringsspelet.

Utforskningsfasen:
– Skriv en arbetsuppgift. En story kan i regel delas upp i ett antal mindre arbetsuppgifter. Vissa arbetsuppgifter är inte knutna till någon särskild story.
– Dela upp arbetsuppgifter om de inte kan göras på några dagar. Slå samman flera mindre arbetsuppgifter till en större om de bara tar en timme var.

Åtagandefasen:
– En programmerare accepterar ansvaret för en viss arbetsuppgift.
– Uppskatta tidsåtgången att utföra en arbetsuppgift.
– Sätt belastningsfaktor som förtäljer hur stor andel av iterationen som väntas bli ideal programmeringstid i procent. I en treveckorscykel borde ingen hinna med mer än 7-8 dagar effektiv programmering.
– Balansering innebär att programmerarna multiplicerar deras belastningsfaktor med uppskattningarna av åtagna arbetsuppgifter. Om någon har blivit överbelastad, måste han/hon lämna ifrån sig uppgifter.

Styrningsfasen:
– Implementera en arbetsuppgift. En programmerare tar ett kort, hittar en kollega, skriver testfallen, skriver koden och kör testerna tills de fungerar. Därefter integreras koden med övriga systemet.
– Varannan eller var tredje dag frågar någon i teamet hur mycket tid programmerarna har lagt ner på sin arbetsuppgift och hur lång tid det är kvar tills den är färdig.
– Återhämtning innebär att programmerare som visar sig vara överbelastade kan minska sina åtagande på ett antal olika sätt.
– Verifiera story innebär att så snart de funktionella testerna för en story är färdiga och aktuella arbetsuppgifter är gjorda kan testerna köras för att kontrollera att implementationen av storyn fungerar.

Utvecklingsstrategi

Om planeringsstrategin stämmer väl med traditionella idéer från programvaruområdet så kan man konstatera att utvecklingsstrategin i XP skiljer sig från mer etablerade strategier.

Kontinuerlig integration innebär att inget kodavsnitt hinner bli äldre än några timmar. Kontinuerlig integration kräver tillgång till verktyg som gör att jobbet kan göras fort. En smidig testmiljö behövs också, så att testerna kan köras snabbt. En stor fördel är att risken i projektet minskar. Om två personer har tänkt olika kring hur något ska fungera upptäcks det efter bara några timmar. Man behöver aldrig sitta och leta efter en bugg som introducerades någon gång under de senaste veckorna. Dessutom får man nyttig träning i integrationsprocessen och är väl förberedd när en release ska släppas. Kontinuerlig integration hjälper också programmeraren att hitta en bra rytm i arbetet. Efter integration kan programmeraren släppa tankarna på den förra arbetsuppgiften och sätta igång med nästa.

Kollektivt ägande innebär att vem som helst kan ändra vilken kod som helst i systemet. Detta skulle inte fungera om man inte hade de automatiserade testerna som kunde ge omedelbar återkoppling på ändringarna. En annan effekt av kollektivt ägande är att komplexiteten i systemet avtar. Den som hittar något som verkar onödigt komplext gör förenklingar av koden. Därmed behöver heller aldrig någon köra fast på grund av andras dåliga arbete, det är bara att ändra det.

Parprogrammering är något av det som är svårast att ta till sig i XP. Om någon vägrar att deltaga ska man veta inte alla passar för XP. Den största fördelen med parprogrammering är att det främjar kommunikation. Ny kunskap om någon del av systemet kommer att sprida sig till hela teamet när man byter par. Detta leder till att inte en person blir ensam om att kunna delar av systemet. Becks erfarenhet är att parprogrammering är effektivare än individuell programmering. Det krävs emellertid mycket träning för att bli verkligt bra på det. Även om man inte skulle uppnå en högre produktivitet så är det ändå värt att arbeta i par på grund av den högre kvalitet man uppnår. Parprogrammering handlar inte om att en person sitter och tittar på när den andre programmerar, bägge ska vara aktiva i arbetet. Med pressen från sin partner minskar också risken att man hoppar över något steg i processen för att göra kortsiktiga tidsvinster.

Designstrategi

XP:s alla fyra värderingar ligger till grund för designstrategin.

Kommunikation – En design ska vara kommunikativ och ge information om systemet. Enkelhet främjar kommunikation.

Enkelhet – Designstrategin ska vara enkel att formulera och resultera i en enkel design. Att genomföra strategin är däremot inte enkelt eftersom bra design alltid är svårt.

Återkoppling – Ett stort problem vid design är att veta om det verkligen fungerar eller inte. XP:s designstrategi innebär därför att snabbt ta fram en enkel design och sedan börja implementera för att få reda på om den fungerar.

Mod – Det krävs mod för att sluta designa tidigt och lita på att den dag det kommer att krävas mer kommer man att kunna lägga till det.

Även XP:s principer stämmer väl med designstrategin.

Liten initial investering – Vi vill göra en så liten investering i designen som möjligt innan vi får lön för den.

Antag enkelhet – Eftersom vi designar så enkelt som möjligt, får vi tid över till ändringar, om den enklaste designen inte skulle fungera.

Inkrementell förändring – Det kommer aldrig att finnas ett tillfälle då designen är gjord. Den kommer hela tiden att förändras i små steg.

Res lätt – Designstrategin ska inte resultera i någon ”extra” design. Endast den som fyller våra nuvarande syften.

XP:s designstrategi kan formuleras i fyra punkter.

1. Börja med ett test, så att vi vet när vi är färdiga. Det krävs en del design bara för att kunna göra det testet: vilka är objekten och vilka synliga metoder har de?
2. Designa och implementera precis så mycket som behövs för att få testet att fungera.
3. Upprepa
4. Så fort man ser möjligheten till att göra designen enklare – gör det.

Systemets flexibilitet utökas med tiden. Förutsättningen för denna strategi är att kostnaden för förändring förblir någorlunda konstant med tiden.

Refaktorisering

Att designa genom refaktorisering innebär att börja med det första testfallet och implementera det. Det kanske räcker med ett objekt och två metoder. Därefter tas nästa testfall fram och designen byggs ut för att kunna hantera det. Så här kan arbetet fortsätta ett par veckor. Efter ett tag kommer vissa delar av systemet inte att kännas bra. Då är det dags att sammankalla teamet och ta en dag av omstrukturering av systemet. Riktigt stora omstruktureringar som tar mycket tid, bör göras i små steg, inte allt på en gång, så att man kan fortsätta att släppa realeaser under tiden.

Om man ska göra den enklaste designen som klarar av alla testfallen, måste man veta vad som menas med enklaste designen. Enligt Beck är det att:

1. Systemet (koden och testerna) måste kommunicera allt som programmeraren vill kommunicera.
2. Systemet får inte innehålla dubbletter av koden. (Punkt 1 och 2 utgör tillsammans En och Endast En gång-regeln)
3. Systemet ska ha så få klasser som möjligt.
4. Systemet ska ha så få metoder som möjligt.

XP:s designstrategi skiljer sig mycket från traditionellt tänkande kring design. En nyckel i sammanhanget är att förstå att det inte bara är tid som är pengar. Även risk kostar pengar. Design är inte heller gratis. Med en mer komplex design finns det hela tiden mer som måste förstås och testas. Overheaden blir större.

Att designa med hjälp av olika visuella representationer går snabbt och gör det enkelt att jämföra olika designalternativ. Nackdelen är att visuella representationer inte ger konkret återkoppling utan innebär risk. Det är först när en design har kodats som man kan avgöra om den verkligen fungerar. Några av XP:s principer kan användas för vägledning när det gäller att använda bilder.

Små initiala investeringar – Lär oss att endast göra några få bilder i taget.
Spela för att vinna – Använd inte bilder av rädsla.
Snabb återkoppling – Ta snabbt reda på om bilderna är riktiga.
Arbeta med människors instinkter – Låt framförallt dem som är duktiga på bilder ta fram dem.
Välkomna förändring och *res lätt* – Spara inte bilderna efter att de har påverkat koden eftersom de ändå kommer att förändras imorgon.

Teststrategi

De allra flesta ogillar att tala om testning eftersom de vet att det är viktigt och att de inte gör tillräckligt. Om programmerarna kunde välja bort att göra tester skulle de göra det. I XP har man valt en strategi för testning som utgår från principen att arbeta med människors instinkter och inte emot dem.

Testerna i XP är isolerade och automatiska. Att de är isolerade innebär att testerna är oberoende av varandra.

Det finns ingen mening med att testa precis allt som går att testa i ett program. Det viktiga är att testa de saker som kan fallera. Ju fler tester man skriver ju mer lär man sig vilken typ av tester som ger värdefull återkoppling

De som skriver tester i XP är programmerarna och kunderna. Programmerarna skriver tester för en metod i taget när något av följande fall föreligger:

– Om en metods gränssnitt är det minsta oklart skrivs ett test innan metoden skrivs.
– Om metodens gränssnitt är klart men metodens implementation är oklar skrivs också ett test för metoden.
– Om man kommer på ett ovanligt fall då koden borde fungera skrivs ett test för att kontrollera det.
– Om man kommer på ett problem senare skrivs ett test som isolerar problemet.
– Om man ska refaktorisera ett kodavsnitt och inte känner till en viss aspekt av dess beteende skrivs ett test för att undersöka det, om det inte redan finns ett sådant test.

Enhetstesterna ska alltid klaras av till 100%. Skulle något test fallera är det teamets högsta prioritet att få det att fungera.

De funktionella testerna som tas fram av kunden måste inte alltid fungera till 100%. Dessa testerna skrivs för varje story. Oftast kan inte kunderna skriva testerna helt på egen hand därför har XP-team oavsett storlek åtminstone en dedike-

rad testare. Testarens uppgift är att omvandla kundens kanske vaga idéer till konkreta, automatiserade och isolerade tester.

Enhetstesterna och de funktionella testerna är hjärtat i XP:s teststrategi. Vid behov kan även andra tester genomföras såsom parallelltester för att jämföra två systems likheter, stresstester för att simulera högsta tänkbara belastning och aptester för att simulera irrationellt användande av systemet.

3.1.3 Implementation

I detta avsnitt tas praktisk information kring implementation av XP upp. Först förklaras hur XP bör införas i en organisation. Därefter beskrivs livscykeln för ett idealt XP-projekt. De roller som ingår i metodiken beskrivs. Vidare ett avsnitt om när XP är olämpligt att använda.

Införa XP

XP bör införas en praktik i taget. Lämpligen börjar man med det största problem man har och försöker lösa det på XP:s sätt. Mest naturligt är att börja med testerna eller planeringsspelet. Eftersom som man tar en praktik i taget har man gott om tid att lära sig den ordentligt. Man bör inte heller underskatta betydelsen av den fysiska miljöns utseende.

Att införa XP i ett redan existerande team är ännu svårare än att börja med ett nytt team. Detta avsnitt innehåller särskilda råd angående testning, design, planering, ledning och utveckling.

Livscykel för ett idealt XP-projekt

Utforskning

Den första fasen i ett idealt XP-projekt är utforskningsfasen. I den experimenterar programmerarna med olika teknologier och olika arkitekturer för systemet. Kunden tränar sig samtidigt på att skriva storys. Denna fas kan ta några veckor för ett team som har erfarenhet av teknologin och några månader för ett team som saknar sådan erfarenhet. Fasen är slut när kunden har skrivit så många storys att det finns substans till en första release och när programmerarna inte kan göra bättre tidsuppskattningar utan att implementera systemet.

Planering

Planering går ut på att komma överens om ett datum när en minsta möjliga men mest värdefulla release ska släppas. Planeringsspelet (se avsnitt 4.4.2) beskriver hur detta går till. Planeringsfasen bör ta en eller två dagar. Planen för den första releasen bör vara mellan två och sex månader lång.

Iterationer till första releasen

Schemat för åtagandet, bryts ner i en till fyra veckor långa iterationer som resulterar i funktionella testfall som täcker de storys som ingår i iterationen. I den första iterationen väljs storys ut som tvingar programmeraren att implementera helst hela arkitekturen för systemet.

Att välja storys för nästkommande iterationer är helt upp till kunden. De storys som ger störst värde för kunden bör väljas först. När iterationerna genomförs bör

man kontrollera hur man ligger till enligt planen. Om det finns stora avvikelser bör man planera om.

I slutet av iterationen har kunden gjort färdigt de funktionella testerna och de fungerar. Varje iteration bör avslutas med en liten fest med pizza och fyrverkerier.

Produktion

Inför en release är det lämpligt att korta ner cyklerna till en vecka. Dagliga möten där alla talar om vad de jobbar med kan vara en bra sak. Typiskt finns någon process som verifierar att systemet är redo att gå i produktion. Nya tester kan behöva skrivas.

Systemets prestanda kan också behöva justeras här. Vilket är lämpligt när man känner systemet som bäst och sannolikt har produktionsmaskinvaran tillgänglig. Hastigheten på utvecklingen av systemet minskar i denna fas eftersom riskerna ökar och varje ändring som ska ingå i releasen behöver övervägas noggrannare.

Många projekt misslyckas och går aldrig in i produktion och därför finns det god anledning att ställa till med en stor fest när ett projekt lyckas och programvaran börjar produceras.

Underhåll

Det normala tillståndet för ett XP-projekt är underhåll. Så fort den första releasen har släppts så finns det programvara som ska underhållas även om utvecklingen går vidare. Varje ny release börjar med en utforskningsfas. I underhållsfasen är man mer försiktig med ändringar.

Död

Förr eller senare är det tid för projektet att dö. Det bra sättet att dö på är när kunden inte kommer med några nya storys. Då är det dags att sätta systemet i malpåse, skriva en fem till tio sidor lång beskrivning av systemet som man gärna vill ha när man ska göra en ändring efter fem år.

Det tråkiga sättet att dö på är när systemet inte längre kan leverera värde för kunden. Avsluta projektet med en fest, med alla utvecklare som varit med under projektets gång.

Roller

I lagidrotter har spelarna ofta olika roller med särskilda ansvarsområden. Spelarna är duktiga på olika saker. En bra coach kan se vilka spelarkaraktärer han har och kan använda en taktik som passar spelarmaterialet. Det är bättre att ändra spelsystemet än spelarna eftersom det senare i regel inte går. I XP finns det några roller som man inte kan klara sig utan: programmerare, kund, coach och spårare.

Programmerare

Programmeraren är hjärtat i XP. Om de kunde göra balanserade beslut om lång- och kortsiktiga prioriteringar skulle det inte behövas någon mer teknisk personal i projektet. Det som främst driver programmeraren är XP:s värdering – kommunikation (se avsnitt 4.4.1) Om programmet är färdigt men det saknas någon del i kommunikationen så är programmerarens arbete ofullständigt.

I XP betonas vissa kompetenser hos programmerarna som inte betonas inom andra utvecklingsstilar. Parprogrammering kräver förmåga att prata, vilket programmerare inte alltid är bra på.

En extrem programmerare måste också ha vanan att styras av enkelhet. Att se igenom kundens önskningar för att hitta det viktigaste och att göra en enkel lösning. Tekniska förmågor som behövs är att han/hon är ganska duktig på att programmera, kan refaktorisera och enhetstesta sin kod. Programmeraren måste också övervinna sina rädslor för misslyckande med hjälp av ett uppmuntrande team.

Kund

En annan roll som är nödvändig i ett XP-projekt är den som kunden har. Det är inte enkelt att vara kund i ett XP-projekt. Det finns mycket som man behöver lära sig, t ex att skriva storys och funktionella tester. Något av det svåraste för kunden är att fatta beslut om prioriteringar, samtidigt som möjligheterna att styra projektet är begränsade.

Testare

Mycket av testningen genomförs av programmerarna själva. Därför är testarens arbete inriktat mot kunden och att få de funktionella testerna att fungera.

Spårare

Spårarens uppgift är att sköta teamets planering och uppföljning. Spåraren använder programmerarnas tidsuppskattningar för att göra den övergripande planeringen. Därefter gör spåraren mätningar av verklig tids- och resursåtgång och meddelar om det behövs några förändringar för att man ska klara av nästa release.

Coach

Coachen har ansvar för att processen efterföljs och måste ha ingående kunskaper i idéerna bakom XP och hur XP ska tillämpas. Coachen ska inte styra för mycket utan arbeta på ett sätt som får programmerarna att själva fatta beslut. Om det blir nödvändigt måste emellertid coachen kunna ingripa med skärpa. Coachen måste inte vara en teknisk expert.

Konsult

Ibland kan teamet få behov av ingående teknisk kunskap för att lösa ett visst problem. Då kan det vara lämpligt att ta in en konsult som hjälper till att lösa det. Under tiden studerar en eller två medlemmar ur teamet hur konsulten löser problemet. Efteråt kastar de antagligen lösningen och gör om den själva.

Big Boss

Det som teamet framförallt behöver från chefen är mod, förtroende och ibland tillrättavisning. Medlemmarna i XP-teamet kommer att vara måna om att i god tid berätta för chefen hur projektet ligger till.

När XP är olämpligt

Det finns vissa omständigheter som gör XP olämpligt att använda. Det största hindret är en företagskultur som innebär att man sätter riktningen för projektet i början och korrigeringar av riktningen ses som svaghet.

Ett exempel på detta är när det krävs en komplett specifikation av programvaran innan programmeringen kan påbörjas. En annan kultur som inte fungerar med XP är när det krävs långa arbetsveckor av medarbetarna. XP kan inte genomföras trött.

XP bör inte användas i stora projekt. Ungefär tio programmerare är en övre gräns. Det som framförallt är problematiskt med stora projekt är integrationen. Det är inte möjligt att kontinuerligt integrera så många kodströmmar.

Om man vet att man har en exponentiell kurva för kostnaden av ändringar bör man inte heller använda XP.

En miljö som inte ger möjlighet till snabb återkoppling är inte heller lämpad för XP. Exempelvis om det tar 24 timmar att kompilera och länka systemet eller om programvaran måste genomgå en två månader lång kvalitetsförsäkring innan den kan släppas.

Ibland kan det vara omöjligt att genomföra tester på grund av svårtillgänglig eller dyr hårdvara.

Om den fysiska miljön inte är rätt är det också omöjligt att utveckla programvara på ett extremt sätt.

3.2 Jämförande analys av utvecklingsarbetet i ISO/IEC 12207 och XP

Detta avsnitt innehåller en jämförande analys av den internationella standarden ISO/IEC 12207:1995 och eXtreme Programming (XP) så som den beskrivs i [Beck00]. I 12207 kan man läsa att standarden är ämnad att vara ett generellt ramverk som är oberoende av livscykelmodell, metoder, tekniker och verktyg.

XP har kritiserats för att innehålla idéer som strider mot erkända och etablerade sätt att utveckla programvara. Ett sätt att ta reda på om så är fallet är att jämföra XP med en av standardvärldens mest erkända processtandarder för programvaruutveckling, ISO/IEC 12207. Eftersom jag inte har haft tillgång till tillägget Amendment 1 från 2002, kan det finnas inaktuell information i beskrivningen av 12207.

Intressanta frågor är till exempel om det finns några konflikter mellan processerna – om de motsäger varandra. Vidare om XP kan rymmas inom ramverket 12207 – vara en tillämpning av det.

ISO/IEC 12207 Information Technology – Software Life Cycle Procesess definierar en process med namnet Development Process. Denna process är en av 17 processer i 12207 och beskriver aktiviteter och arbetsuppgifter för utveckling av ett programvarusystem. Eftersom 12207 inte beskriver hur arbetsuppgifterna skall genomföras finns det stort utrymme för utvecklaren att välja metod och teknik. Att välja just utvecklingsprocessen var naturligt eftersom det är den mest omfattande av 12207:s processer och eftersom XP starkt betonar utvecklingsarbetet. I Tabell 6 räknar jag upp de olika ingående aktiviteterna och arbetsuppgifterna. För att förenkla har jag tagit bort de aktiviteter som fokuserar systemaspekter (5.3.2, 5.3.3, 5.3.10 och 5.3.11)

Den första kolumnen i Tabell 6 innehåller den aktuella paragrafen i ISO/IEC 12207. Den andra kolumnen innehåller:

– Paragraftexten från 12207 översatt till svenska. Processer, aktiviteter och arbetsuppgifter.
– XP:s riktlinjer inom motsvarande område.
– Min bedömning av huruvida 12207 och XP är förenliga inom aktuellt område, samt eventuell motivering till bedömningen.

Den tredje kolumnen innehåller bedömningen en gång till. Jag använder en fyrgradig skala för att beskriva huruvida aktiviteterna eller arbetsuppgifterna i 12207 och är förenliga med XP eller inte.

Ja – XP är förenlig med 12207.
Ok – XP motsäger inte 12207.
Tveksamt (?) – Det är tveksamt om XP är förenlig med 12207.
Nej – XP är inte förenlig med 12207.

Paragrafhänvisningar inne i texten och i den första kolumnen refererar till någon av paragraferna i 12207. Det första siffran pekar ut processgrupp: primära livscykelprocesser (5), stödjande processer (6) eller organisatoriska livscykelprocesser (7). Den andra siffran pekar ut vilken process det gäller, till exempel utvecklingsprocessen (5.3). Den tredje siffran pekar ut vilken aktivitet inom processen som avses, till exempel Design av programvaruarkitektur (5.3.5). Den fjärde siffran pekar ut en arbetsuppgift inom aktiviteten, till exempel: ”Utvecklaren skall utveckla och dokumentera en top-nivådesign för programvarudelens externa gränssnitt och mellan programvarans komponenter för programvarudelen.” (5.3.5.2)

När någon av 12207:s processer omnämns i texten är de kursiverade. När ”utvecklaren” omnämns i texten motsvarar det den organisation som utvecklar programvaran.

Tabell 6. Utvecklingsprocessen i 12207 jämförd med XP

5.3	**U T V E C K L I N G S P R O C E S S E N**	
5.3.1	**IMPLEMENTATION AV PROCESSEN.**	
	XP: Metodiken XP ska implementeras i ett verkligt projekt.	**Ja**
	Ja.	
5.3.1.1	Om inte fastställt i kontraktet, skall utvecklaren definiera eller välja en programvarulivscykel lämplig för omfånget, betydelsen av och komplexiteten i projektet. *Utvecklingsprocessens* aktiviteter och arbetsuppgifter skall väljas och fördelas på livscykelmodellen.	
	XP: XP ska anpassas till det specifika projektet enligt Beck, som beskriver en ideal livscykel för ett XP-projekt. Denna beskrivning är inte särskilt detaljerad, vilket leder till ett behov hos den som vill implementera metodiken att göra en noggrannare beskrivning. Det skapar också utrymme för en viss variation av modeller [Bunse]. Exempelvis kan XP implementeras som en minimal tillämpning av RUP. [Booch98]	Ok

	Ok.	
5.3.1.2	Utvecklaren skall:	
	a) Dokumentera utdata i enlighet med *dokumentationsprocessen* (6.1).	
	XP: Följ XP:s riktlinjer för dokumentation. Dessa innebär att endast skapa sådan dokumentation som är kritisk för projektet, främst kod och tester.	Nej
	Nej. Om 12207 kräver att det är dess egen process för dokumentation som ska följas.	
	b) Placera utdata under *konfigurationsstyrningsprocessen* (6.2) och utföra ändringsstyrning i enlighet med den.	
	XP: XP kräver mycket lite konfigurationsstyrning eftersom all kod ägs gemensamt och integreras kontinuerligt. Viss grundläggande styrning behövs givetvis för att kunna ångra ändringar och hantera de olika releaserna.	Nej
	Nej. Om 12207 kräver att det är dess egen process för konfigurationsstyrning som måste följas.	
	c) Dokumentera och lösa problem och disharmonier funna i programvaruprodukter och arbetsuppgifter i enlighet med *problemlösningsprocessen* (6.8).	
	XP: Lös problem på lägsta möjliga nivå i organisationen i enlighet med XP:s riktlinjer för problemlösning. Det viktigaste verktyget för problemlösningen är kommunikation. Kunden på plats kan omedelbart svara på frågor. Vissa problem måste lösas av coachen.	Nej
	Nej. Om 12207 kräver att det är dess egen process för problemlösning som måste följas.	
	b) Utföra de stödjande processerna (paragraf 6) som specificerade i kontraktet.	
	XP: XP beskriver strategier för att stödja utvecklingsarbetet, exempelvis strategier för ledning, faciliteter och planering.	Nej
	Nej. Om inte 12207 kräver att det är dess egna stödjande processer som måste följas.	
5.3.1.3	Utvecklaren skall välja, anpassa och använda de standarder, metoder, verktyg och programspråk (om inte fastställt i kontraktet) som är dokumenterade, lämpliga och fastställda av organisationen för att utföra aktiviteterna i *utvecklingsprocessen* och i de stödjande processerna (paragraf 6).	
	XP: XP-praktiken *kodningsstandarder* föreskriver val av gemensamma kodningsstandarder, som kräver minimalt arbete och betonar kommunikation. För övrigt skall standarder, metoder och verktyg som är lämpliga och förenliga med XP väljas, anpassas och användas i projektet.	Ok
	Ok.	
5.3.1.4	Utvecklaren skall utveckla planer för utförandet av aktiviteterna i utvecklingsprocessen. Planerna bör inkludera specificerade standarder, metoder, verktyg, hand-	

	lingar och ansvar associerade med utvecklingen och acceptansen av alla krav inklusive systemsäkerhet och intrångssäkerhet. Om nödvändigt får separata planer utvecklas. Dessa planer skall dokumenteras och genomföras.	
	XP: Utvecklaren skall utföra praktikerna i XP på ett genomtänkt sätt.	?
	Tveksamt. I XP ska inga planer som inte är kritiska för projektet utvecklas och dokumenteras.	
5.3.1.5	Icke levererbara delar kan brukas i utvecklingen eller underhållet av programvaruprodukten. Likväl, skall det säkerställas att driften och underhållet av den levererbara programvaruprodukten efter dess leverans till beställaren är oberoende av sådana delar, annars bör de delarna betraktas som levererbara.	
	XP: Testerna i XP ska leva med koden och utvecklas i takt med den. På så vis förlängs programvarans livslängd.	?
	Tveksamt. Enligt XP inte bara kan utan ska testerna underhållas.	
5.3.4	**ANALYS AV PROGRAMVARUKRAV.** För varje programvarudel (eller konfigurerad programvarudel, om definierad) består denna aktivitet av arbetsuppgifterna nedan:	
	XP: Kundens krav analyseras i XP genom kommunikation med kunden och skrivandet av storys och funktionella tester.	**Ok**
	Ok. Om 12207 inte endast avser krav formulerade på ett visst sätt, till exempel i en traditionell kravspecifikation, uppstår ingen konflikt med XP.	
5.3.4.1	Utvecklaren skall fastslå och dokumentera programvarukrav, inklusive specifikationerna av kvalitetskaraktäristika, beskrivna nedan. Vägledning för specificerandet av kvalitetskaraktäristika kan återfinnas i ISO/IEC 9126. a) Funktionella och duglighetspecifikationer, inklusive prestanda, fysiska karaktäristika, och omgivande förutsättningar under vilka programvarudelen ska fungera; b) Programvarudelens externa gränssnitt; c) Acceptanskrav; d) Specifikationer av systemsäkerhet, inklusive de relaterade till metoder för drift och underhåll, miljöpåverkan och personalskador; e) Specifikationer av intrångssäkerhet, inklusive de relaterade till äventyrande av känslig information; f) Ergonomi, inklusive de relaterade till manuella operationer, människa-maskininteraktion, begränsningar på personal och områden som behöver koncentererad mänsklig uppmärksamhet, som är känsliga för mänskliga fel och träning; g) Datadefinition och databaskrav; h) Installations- och acceptanskrav för programvaruprodukten vid drifts- och underhållsplats(erna); i) Användardokumentation; j) Användardrift och utförandekrav; k) Krav på användarunderhåll.	
	XP: Funktionella krav och acceptanskrav ska fastslås och dokumenteras i XP	Nej

	genom storys respektive funktionella tester. Alla övriga krav skall också tas i betraktande i utvecklingsarbetet, men ska inte fastslås eller dokumenteras.	
	Nej. Enligt XP bör inte alla uppräknade specifikationerna av kvalitetskaraktäristika fastslås och dokumenteras.	
5.3.5	**DESIGN AV PROGRAMVARUARKITEKTUR.** För varje programvarudel (eller konfigurerad programvarudel, om definierad) består denna aktivitet av arbetsuppgifterna nedan:	
	XP: Programvaruarkitekturen får en första grov design i XP genom metaforen. I utforskningsfasen experimenterar man med olika arkitekturer. Därefter sker designen av arkitekturen dag för dag i samband med kodningen av programvaran. Den första releasen bör emellertid innehålla huvuddelen av arkitekturen. Arkitekturen fastställs aldrig i ett XP-projekt utan förändras och förfinas under systemets uppbyggnad.	**Ja**
	Ja. All programvara behöver en bra arkitektur.	
5.3.5.1	Utvecklaren skall omvandla kraven för programvarudelen till en arkitektur som beskriver dess top-nivåstruktur och identifierar programvarukomponenterna. Det skall säkerställas att alla kraven för programvarudelen allokeras till sina programvarukomponenter och förfinas ytterligare för att underlätta detaljerad design. Programvarans arkitektur skall dokumenteras.	
	XP: XP:s storykort säkerställer att alla krav specificerade på dem implementeras i systemet. Delar av arkitekturen dokumenteras genom metaforen.	?
	Tveksamt. XP ställer inga explicita krav på att programvarans olika komponenter skall identifieras. XP föreskriver inga metoder för att säkerställa att kraven allokeras till komponenterna.	
5.3.5.2	Utvecklaren skall utveckla och dokumentera en top-nivådesign för programvarubitens externa gränssnitt och mellan programvarans komponenter för programvarubiten.	
	XP: Designen skall utvecklas inkrementellt i samband med implementationen. Koden ska vara enkel och kommunicera den viktiga designinformationen.	Nej
	Nej. XP motsätter sig att göra en färdig design och att dokumentera denna. Designen skall inte dokumenteras på annat sätt än genom testerna och koden.	
5.3.5.3	Utvecklaren skall definiera och dokumentera en top-nivådesign för databasen.	
	XP: Se kommentar till 5.3.5.2.	Nej
	Nej. Se kommentar till 5.3.5.2.	
5.3.5.4	Utvecklaren bör utveckla och dokumentera preliminära versioner av användardokumentation.	
	XP: Användardokumentation ingår inte som en del av XP:s metodik.	Ok
	Ok. XP motsätter sig inte att preliminära versioner av användardokumentation utvecklas.	
5.3.5.5	Utvecklaren skall definiera och dokumentera preliminära testkrav och schemat för	

	integration av programvaran.	
	XP: Utvecklaren skall skriva tester för alla tänkbara fall som behöver testas. Testerna sparas. I XP sker integration kontinuerligt.	Nej
	Nej. I XP definieras och dokumenteras inte testkrav eller schema för integration av programvaran.	
5.3.5.6	Utvecklaren skall utvärdera arkitekturen för programvarudelen och designerna av gränssnitt och databas i betraktande av kriterierna uppräknade nedan. Resultaten av utvärderingarna skall dokumenteras. a) Spårbarhet till programvarudelens krav; b) Extern konsistens med programvarudelens krav; c) Intern konsistens mellan programvarukomponenterna; d) Lämplighet av designmetoder och –standarder som används; e) Genomförbarhet av detaljerad design; f) Genomförbarhet av drift och underhåll.	
	XP: XP kräver att alla programmerare i projektet utvärderar och förbättrar arkitekturen genom refaktorisering närhelst behovet dyker upp.	?
	Tveksamt. Om 12207 kräver en formell utvärdering kan det inte ske inom ramen för XP. Formell dokumentation av utvärderingar passar inte in i XP. XP kan inte garantera att samtliga kriterierna tas i betraktande.	
5.3.5.7	Utvecklaren skall förrätta gemensamma granskningar i enlighet med 6.6.	
	XP: I XP sker granskning hela tiden genom parprogrammering. Vid särskilda behov kan utvecklingsteamet sammankalla till möte.	?
	Tveksamt. Formella granskningar ersätts i huvudsak av parprogrammeringen.	
5.3.6	**DETALJERAD DESIGN AV PROGRAMVARAN.** För varje programvarudel (eller konfigurerad programvarudel, om definierad) består denna aktivitet av arbetsuppgifterna nedan:	
	XP: I XP används ingen representation av programvaran som ligger mellan övergripande idéer, såsom storys, och implementationen i kod. Detaljerad design av programvaran återfinns endast tillfälligt någon annanstans än i programmerarens huvud.	**?**
	Tveksamt. Om det i 12207 avses att en mellanliggande representation av programvaran ska tas fram innebär det en konflikt med XP. Om detaljerad design kan förstås som den småskaliga strukturen på koden skulle det kunna gå att jämka 12207 och XP på denna punkt.	
5.3.6.1	Utvecklaren skall utveckla en detaljerad design för varje programvarukomponent hos programvarudelen. Programvarukomponenterna skall förfinas till lägre nivåer innehållande programvaruenheter som kan kodas, kompileras och testas. Det skall säkerställas att alla programvarukraven allokeras från programvarukomponenterna till programvaruenheterna. Den detaljerade designen skall dokumenteras.	
	XP: I XP påbörjas kodningen så fort som möjligt för att testa om designidéerna verkligen håller.	Nej
	Nej. Enligt XP skall ingen detaljerad design utvecklas och dokumenteras.	

5.3.6.2	Utvecklaren skall utveckla och dokumentera en detaljerad design för programvarudelens externa gränssnitt, mellan programvarukomponenterna och mellan programvaruenheterna. Den detaljerade designen av gränssnitten skall tillåta kodning utan behovet av ytterligare information.	
	XP: I XP påbörjas kodningen så fort som möjligt för att testa om designidéerna verkligen håller.	Nej
	Nej. Enligt XP bör ingen detaljerad design av gränssnitt färdigställas och dokumenteras före implementationen påbörjas.	
5.3.6.3	Utvecklaren skall utveckla och dokumentera en detaljerad design för databasen.	
	XP: Se kommentar till 5.3.6.1.	Nej
	Nej. Se kommentar till 5.3.6.1.	
5.3.6.4	Utvecklaren skall uppdatera användardokumentation om nödvändigt.	
	XP: Användardokumentation ingår inte som en del av XP:s metodik.	Ok
	Ok. XP motsätter sig inte att användardokumentation uppdateras vid behov.	
5.3.6.5	Utvecklaren skall definiera och dokumentera testkrav och schema för testning av programvaruenheter. Testkraven bör inkludera att stressa programvaruenheten vid gränserna för dess krav.	
	XP: Utvecklaren skall skriva och utföra tester för alla tänkbara fall som behöver testas.	Nej
	Nej. I XP definieras och dokumenteras inte testkrav eller schema för testning av programvaruenheter.	
5.3.6.6	Utvecklaren skall uppdatera testkraven och schemat för *integration av programvaran*.	
	XP: Utvecklaren skall uppdatera testerna vid behov.	Nej
	Nej. I XP definieras och dokumenteras inte testkrav eller schema för integration av programvaran.	
5.3.6.7	Utvecklaren skall utvärdera programvarans detaljerade design och testkraven i betraktande av kriterierna medtagna i listan nedan. Resultaten av utvärderingarna skall dokumenteras. a) Spårbarhet till programvarudelens krav; b) Extern konsistens med arkitekturell design; c) Intern konsistens mellan programvarukomponenter och programvaruenheter; d) Lämplighet av designmetoder och –standarder använda; e) Genomförbarhet av testning; f) Genomförbarhet av drift och underhåll.	
	XP: XP kräver att alla utvecklare i projektet ständigt förbättrar designen genom	?

	refaktorisering, närhelst behovet dyker upp.	
	Tveksamt. Eftersom ingen detaljerad design tas fram behöver den inte heller utvärderas. Detaljerad design i betydelsen av den småskaliga strukturen i koden ska däremot utvärderas. Testkrav tas inte heller fram. Dokumentation av utvärderingar är överflödigt.	
5.3.6.8	Utvecklaren skall förrätta gemensamma granskning(ar) i överensstämmelse med 6.6.	
	XP: I XP sker granskning hela tiden genom parprogrammering. Vid särskilda behov kan utvecklingsteamet sammankalla till möte.	?
	Tveksamt. Formella granskningar ersätts i huvudsak av parprogrammeringen.	
5.3.7	**KODNING OCH TESTNING AV PROGRAMVARAN.** För varje programvarudel (eller konfigurerad programvarudel, om definierad) består denna aktivitet av arbetsuppgifterna nedan:	
	XP: Utvecklaren skall skriva tester först, därefter skall koden skrivas och testas.	**Ja**
	Ja. Kodning och testning av programvara tillhör XP:s grundläggande aktiviteter.	
5.3.7.1	Utvecklaren skall utveckla och dokumentera följande: a) Varje programvaruenhet och databas; b) Testprocedurer och data för testning av varje programvaruenhet och databas.	
	XP: Utvecklaren skall utveckla och dokumentera, det vill säga koda, alla delar av programvaran. Innan kodningen sker utvecklas tester för koden.	?
	Tveksamt. Testprocedurer och data för testning bör inte dokumenteras enligt XP.	
5.3.7.2	Utvecklaren skall testa varje programvaruenhet och databas säkerställande att den uppfyller sina krav. Testresultatet skall dokumenteras.	
	XP: Utvecklaren skall testa alla delar av programvaran säkerställande att den uppfyller kundens behov.	?
	Tveksamt. Testresultat bör inte dokumenteras enligt XP.	
5.3.7.3	Utvecklaren skall uppdatera användardokumentationen om nödvändigt.	
	XP: Användardokumentation ingår inte som en del av XP:s metodik.	Ok
	Ok. XP motsätter sig inte att användardokumentation uppdateras vid behov.	
5.3.7.4	Utvecklaren skall uppdatera testkraven och schemat för *integration av programvaran*.	
	XP: Utvecklaren skall uppdatera testerna vid behov.	Nej
	Nej. I XP definieras och dokumenteras inte testkrav eller schema för integration av programvaran.	
5.3.7.5	Utvecklaren skall utvärdera programvarukod och testresultat i betraktande av krite-	

	rierna medtagna på listan nedan. Resultaten av utvärderingarna skall dokumenteras. a) Spårbarhet till programvarudelens krav och design; b) Extern konsistens med programvarudelens krav och design; c) Intern konsistens mellan enhetskrav; d) Testtäckning av enheter; e) Lämplighet av kodningsmetoder och –standarder använda; f) Genomförbarhet av programvaruintegration och –testning; g) Genomförbarhet av drift och underhåll.	
	XP: XP kräver att alla programmerare i projektet utvärderar och förbättrar koden genom refaktorisering närhelst behovet dyker upp. Om något test visar dåligt resultat skall problemet åtgärdas.	?
	Tveksamt. Testkrav tas inte fram i XP. Formell dokumentation av utvärderingar passar inte in i XP. XP kan inte garantera att samtliga kriterierna tas i betraktande.	
5.3.8	**INTEGRATION AV PROGRAMVARAN.** För varje programvarudel (eller konfigurerad programvarudel, om definierad) består denna aktivitet av arbetsuppgifterna nedan:	
	XP: Programmerarna integrerar programvaran kontinuerligt.	**Ja**
	Ja. Integration av programvaran görs i både 12207 och XP.	
5.3.8.1	Utvecklaren skall utveckla en integrationsplan för att integrera programvaruenheterna och programvarukomponenterna till en programvarubit. Planen skall inkludera testkrav, procedurer, data, ansvar och schema. Planen skall dokumenteras.	
	XP: Programmerarna integrerar programvaran kontinuerligt, dagligen, eller så ofta en arbetsuppgift är slutförd.	?
	Tveksamt. Någon särskild plan för integration upprättas inte i XP.	
5.3.8.2	Utvecklaren skall integrera programvaruenheterna och programvarukomponenterna och testa då sammansättningarna utvecklas i överensstämmelse med integrationsplanen.	
	XP: Utvecklaren skall integrera programvarans olika delar och testa sammansättningarna.	Ja
	Ja.	
5.3.8.3	Utvecklaren skall uppdatera användardokumentationen om nödvändigt.	
	XP: Användardokumentation ingår inte som en del av XP:s metodik.	Ok
	Ok. XP motsätter sig inte att användardokumentation uppdateras vid behov.	
5.3.8.4	Utvecklaren skall utveckla och dokumentera, för varje av programvarudelens acceptanskrav, en mängd av tester, testfall (indata, utdata, testkriterier) och testprocedurer för genomförande av programvarans acceptanstestning. Utvecklaren skall säkerställa att den integrerade programvarudelen är redo för programvaruacceptanstestning.	

	XP: Kunden utvecklar och dokumenterar funktionella tester som avgör när en story är färdig. Inför en release kan ytterligare tester behöva skrivas.	?
	Tveksamt. Om det måste ske på ett mycket formellt sätt blir det kanske konflikt med XP.	
5.3.8.5	Utvecklaren skall utvärdera integrationsplanen, design, kod, tester, testresultat och användardokumentation med hänsyn till kriterierna medtagna på listan nedan. Resultaten av utvärderingen skall dokumenteras. a) Spårbarhet till systemkraven; b) Extern konsistens med systemkraven; c) Intern konsistens; d) Testtäckning av programvarubitens krav; e) Lämplighet av teststandarder och -metoder som används; f) Överensstämmelse med förväntade resultat; g) Genomförbarhet av acceptanstestning av programvaran; h) Genomförbarhet av drift och underhåll.	
	XP: XP kräver att alla utvecklare i projektet ständigt förbättrar design, kod, och tester genom refaktorisering, närhelst behovet dyker upp. Om något test visar dåligt resultat skall problemet åtgärdas.	?
	Tveksamt. Integrationsplan görs ej i XP. Formell dokumentation av utvärderingar passar inte in i XP. XP kan inte garantera att samtliga kriterierna tas i betraktande.	
5.3.8.6	Utvecklaren skall förrätta gemensamma granskningar i enlighet med 6.6.	
	XP: I XP sker granskning hela tiden genom parprogrammering. Vid särskilda behov kan utvecklingsteamet sammankalla till möte.	?
	Tveksamt. Formella granskningar ersätts i huvudsak av parprogrammeringen.	
5.3.9	**ACCEPTANSTESTNING AV PROGRAMVARAN.** För varje programvarudel (eller konfigurerad programvarudel, om definierad) består denna aktivitet av arbetsuppgifterna nedan:	
	XP: Acceptanstestning i XP består av funktionell testning och drift av de tidiga versionerna av programvaran.	**Ok**
	Ok.	
5.3.9.1	Utvecklaren skall uträtta acceptanstestning i överensstämmelse med acceptanskraven för programvarudelen. Det skall försäkras att implementationen av varje programvarukrav testas för uppfyllelse. Acceptanstestningens resultat skall dokumenteras.	
	XP: I XP utförs funktionella tester för att kontrollera att alla storys är implementerade.	?
	Tveksamt. Formell dokumentation av testresultat passar inte in i XP.	
5.3.9.2	Utvecklaren skall uppdatera användardokumentationen om nödvändigt.	

	XP: Användardokumentation ingår inte som en del av XP:s metodik.	Ok
	Ok. XP motsätter sig inte att användardokumentation uppdateras vid behov.	
5.3.9.3	Utvecklaren skall utvärdera designen, koden, testerna, testresultaten och användar-dokumentationen i betraktande av kriterierna medtagna på listan nedan. Resultaten av utvärderingarna skall dokumenteras. a) Testtäckning av programvarudelens krav; b) Överensstämmelse med förväntade resultat; c) Genomförbarhet av systemintegration och –testning, om utfört; d) Genomförbarhet av drift och underhåll.	
	XP: XP kräver att alla programmerare i projektet utvärderar och förbättrar design, kod och tester genom refaktorisering närhelst behovet dyker upp. Om något test visar dåligt resultat skall problemet åtgärdas.	?
	Tveksamt. Formell dokumentation av utvärderingar passar inte in i XP. XP kan inte garantera att samtliga kriterierna tas i betraktande.	
5.3.9.4	Utvecklaren skall stödja revision(er) i överensstämmelse med 6.7. Resultaten av revisionerna skall dokumenteras. Om både maskinvara och programvara är under utveckling eller integration, kan revisionerna bli uppskjutna till *acceptanstestningen av systemet*.	
	XP: Utvecklaren skall upprätthålla intensiv kommunikation med kunden.	Nej
	Nej. Formella revisioner ska inte genomföras i XP.	
5.3.9.5	Vid framgångsrikt fullgörande av revisionerna, om utförda, skall utvecklaren: a) Uppdatera och förbereda den levererbara programvaruprodukten för *systemintegration*, *acceptanstestning av systemet*, *installation av programvaran* eller *stöd för acceptans av programvaran* som tillämpbart; b) Fastställa en baseline för programvarudelens design och kod.	
	XP: Inför en release kan nya tester tas fram för att kontrollera att programvaran är redo för leverans.	?
	Tveksamt. XP motsätter sig inte att utvecklaren uppdaterar och förbereder för utvecklingenprocessens avslutande aktiviteter. Inte heller att en baseline för koden fastställs, däremot skall ingen baseline för designen fastställas.	
5.3.12	**INSTALLATION AV PROGRAMVARAN.**	
	XP: Installation av programvaran ingår inte i XP.	**Ok**
	Ok. XP motsätter sig inte att utvecklaren helt eller delvis installerar programvaran.	
5.3.13	**STÖD FÖR ACCEPTANS AV PROGRAMVARAN.**	
	XP: Stöd för acceptans av programvaran ingår inte i XP.	**Ok**
	Ok. XP motsätter sig inte att utvecklaren ger stöd för acceptans av programvaran.	

Resultatet av analysen (se Tabell 7) är att av nio aktiviteter har jag funnit fyra som helt klart är förenliga med XP, fyra som inte motsäger XP och en som det är tveksamt om den är förenlig med XP.

Vidare ger analysen av totalt 37 arbetsuppgifter att endast en fullt ut är förenlig med XP, sex motsäger inte XP, 18 är tveksamma och tolv är inte förenliga med XP. Orsaken till att aktiviteterna är mer förenliga med XP än arbetsuppgifterna är att aktiviteterna är allmänt beskrivna och ger därför mycket större utrymme för olika lösningar. Arbetsuppgifterna är mer detaljerat beskrivna och kräver därför mer exakt överensstämmelse. Elva av de oförenliga och elva av de tveksamma arbetsuppgifterna är svåra att harmonisera med XP helt eller delvis för att de kräver extra dokumentation. XP:s princip *res lätt* innebär att all dokumentation som inte är kritisk för projektet ska undvikas.

Några av arbetsuppgifterna kräver också att de ska genomföras på ett formellt sätt som ibland strider mot XP:s praktiker.

Min slutsats av analysen är att ISO/IEC 12207:1995 och eXtreme Programming motsäger varandra på några punkter. Det kan dock vara möjligt att rymma XP:s strategi för utveckling inom 12207:s utvecklingsprocess genom att följa 12207:s anpassningsprocess[6] (Annex A) som innebär att enskilda aktiviteter och arbetsuppgifter kan väljas bort och att andra kan läggas till. Frågan är hur mycket av 12207 som blir kvar.

Tabell 7. XP:s förenlighet med 12207

Paragraf	**Beskrivning**	**Förenlig**
5.3	**UTVECKLINGSPROCESSEN**	
5.3.1	**Implemenation av processen**	**Ja**
5.3.1.1		Ok
5.3.1.2		Nej
5.3.1.3		Ok
5.3.1.4		?
5.3.1.5		?
5.3.4	**Analys av programvarukrav**	**Ok**
5.3.4.1		Nej
5.3.5	**Design av programvaruarkitektur**	**Ja**
5.3.5.1		?
5.3.5.2		Nej
5.3.5.3		Nej
5.3.5.4		Ok
5.3.5.5		Nej
5.3.5.6		?
5.3.5.7		?
5.3.6	**Detaljerad design av programvaran**	**?**
5.3.6.1		Nej
5.3.6.2		Nej
5.3.6.3		Nej
5.3.6.4		Ok
5.3.6.5		Nej

[6] Tailoring process

5.3.6.6		Nej
5.3.6.7		?
5.3.6.8		?
5.3.7	**Kodning och testning av programvaran**	**Ja**
5.3.7.1		?
5.3.7.2		?
5.3.7.3		Ok
5.3.7.4		Nej
5.3.7.5		?
5.3.8	**Integration av programvaran**	**Ja**
5.3.8.1		?
5.3.8.2		Ja
5.3.8.3		Ok
5.3.8.4		?
5.3.8.5		?
5.3.8.6		?
5.3.9	**Acceptanstestning av programvaran**	**Ok**
5.3.9.1		?
5.3.9.2		Ok
5.3.9.3		?
5.3.9.4		Nej
5.3.9.5		?
5.3.12	**Installation av programvaran**	**Ok**
5.3.13	**Stöd för acceptans av programvaran**	**Ok**

I Tabell 8 återges de arbetsuppgifter som enligt analysen inte är förenliga med XP. De konflikter mellan ISO/IEC 12207 och XP som jag bedömer vara mest signifikanta är att:

– 12207 kräver att mycket information ska definieras och dokumenteras.

– 12207 kräver att en arkitekturell design färdigställs innan ytterligare utvecklingsarbete får företas.

– 12207 kräver att en detaljerad design tas fram.

– 12207 kräver att testkrav och schema för testning och integration ska tas fram.

Tabell 8. Arbetsuppgifter i 12207 som inte är förenliga med XP.

Paragraf	**Motivering**
5.3.1.2	Om 12207 kräver att det är dess egna stödjande processer som ska följas.
5.3.4.1	Enligt XP bör inte alla uppräknade specifikationerna av kvalitetskaraktäristika fastslås och dokumenteras.
5.3.5.2	XP motsätter sig att göra en färdig design och att dokumentera denna. Designen skall inte dokumenteras på annat sätt än genom testerna och koden.

5.3.5.3	XP motsätter sig att göra en färdig design och att dokumentera denna. Designen skall inte dokumenteras på annat sätt än genom testerna och koden.
5.3.5.5	I XP definieras och dokumenteras inte testkrav eller schema för integration av programvaran.
5.3.6.1	Enligt XP skall ingen detaljerad design utvecklas och dokumenteras.
5.3.6.2	Enligt XP bör ingen detaljerad design av gränssnitt färdigställas och dokumenteras före implementationen påbörjas.
5.3.6.3	Se kommentar till 5.3.6.1.
5.3.6.5	I XP definieras och dokumenteras inte testkrav eller schema för testning av programvaruenheter.
5.3.6.6	I XP definieras och dokumenteras inte testkrav eller schema för integration av programvaran.
5.3.7.4	I XP definieras och dokumenteras inte testkrav eller schema för integration av programvaran.
5.3.9.4	Formella revisioner ska inte genomföras i XP.

4 Informationsmodell för utvecklingsprocessen

Om utvecklingsarbetet inom de olika teknikdisciplinerna kan samordnas på ett effektivare sätt borde det kunna resultera i minskade kostnader, kortare utvecklingstid och förbättrade möjligheter till projektstyrning. Inom området för produktdata pågår idag experiment med att samordna data från de olika teknikdisciplinerna. PDM-system har länge använts för att hantera data för mekaniska produkter. Inom programvaruområdet finns SCM-system. Att integrera dessa system vore värdefullt.

På samma sätt som man kan bygga modeller av en produkt, kan man bygga modeller av en process. Målsättningen för mitt arbete har varit att ta fram en processmetamodell som kan användas inom alla teknikdisciplinerna. Ett av de största hindren för att lyckas med det är att alla disciplinerna använder sig av sin egen specifika terminologi. Även inom disciplinerna varierar terminologin. Detta arbete rymmer inte någon ingående analys av den terminologi som finns, men genom analys av några processtandarder, [WFMC] och [Svensson03] har jag kunnat göra ett val av den metainformation som jag anser bör finnas med i en beskrivning av en produktframtagningsprocess. Jag vill dock inte hävda att de val som jag gjort är de enda möjliga.

Nyttan med informationsmodellen är att den sätter fokus på vilken information som en process egentligen bör beskriva. När detta är känt kan de olika disciplinerna anpassa sina processmodeller till den modellen. Informationsmodellen är också utgångspunkten för att kunna skapa gemensamma datorbaserade verktyg för hantering av processen. Det finns ett stort antal tänkbara funktioner för sådana verktyg, varav några räknas upp nedan:

– Visualisering av processen

Att få visuella beskrivningar av en process är viktigt för att ge en ökad förståelse av processen.

– Modifiering av processen

  Fördelen med att hantera processer i ett datorverktyg är att ändringar och tillägg kan göras på ett smidigt sätt.

– Manuell och intelligent anpassning av processen

  Generella utvecklingsprocesser behöver i regel anpassas till lokala förutsättningar. Att genom ett antal musklick få fram en skräddarsydd instans av processen ger stora möjligheter att på ett tidseffektivt sätt hantera sina processer.

  Integrerat med ett kunskapssystem kan intelligenta anpassningar göras åt användaren. Anpassningarna görs baserat på ett antal parametervärden som användaren matar in i systemet.

– Exekvering av processen

  Denna funktion vägleder användaren under exekveringen av processen. Verktyget håller reda på var i processen användaren befinner sig och vilka aktiviteter som ska genomföras härnäst.

  Denna funktion kan byggas ut med till exempel datainsamling. Verktyget samlar då in data från användaren och kan sammanställa rapporter som är användbara vid processförbättring.

Informationsmodellens innehåll

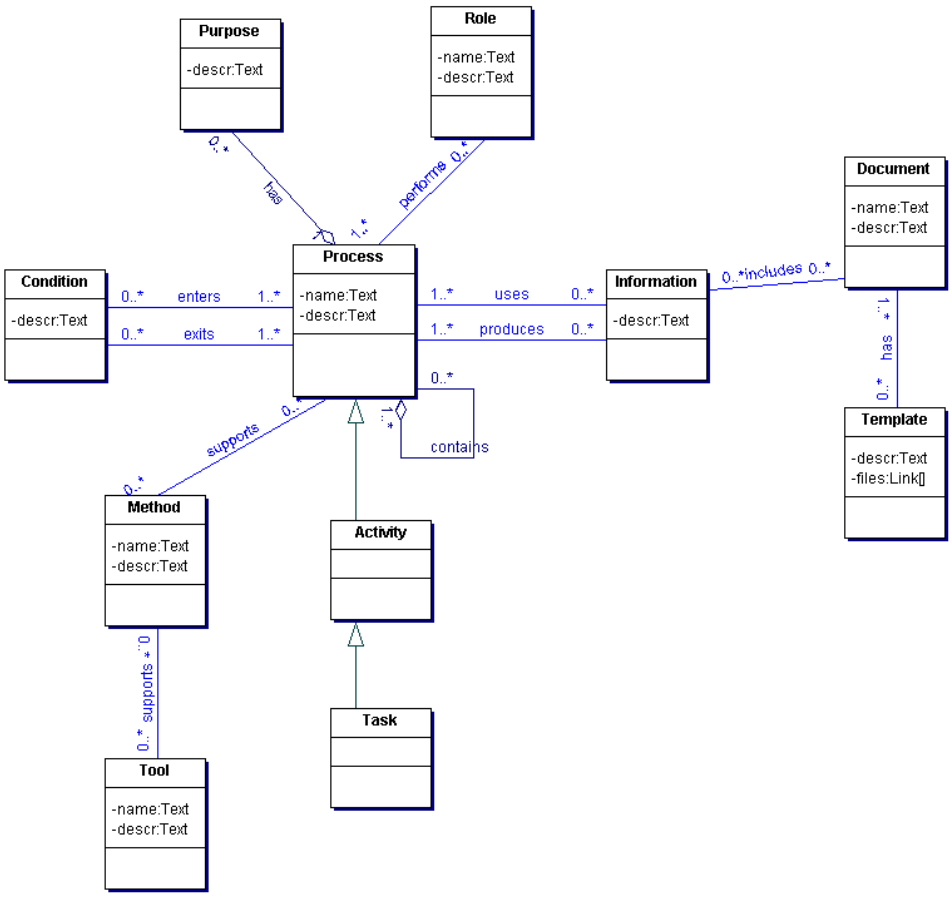
Nedan kommenterar jag samtliga entiteter i informationsmodellen i Figur 3.

Process
Processen utgör den viktigaste entiteten, runt den kretsar nästan alla andra. Varje process kan innehålla ett antal subprocesser. På det sättet byggs en trädstruktur av processer upp. Processen har namn, beskriving och syfte. Dessutom kan processen ha:

– roll
– ingångsvillkor
– utgångsvillkor
– konsumerad information
– producerad information
– metoder

Activity & Task
För att kunna hantera processer på lägre nivåer annorlunda än de på högre nivåer finns specialiseringarna aktivitet och arbetsuppgift med.

Purpose
Varje process kan ha ett syfte. Syftet har en textbeskrivning.



Figur 3. Informationsmodell för utvecklingsprocessen (UML klassdiagram)

Role
Varje process kan ha en roll som utför processen. Samma roll kan utföra flera processer. Rollen har ett namn och en textbeskrivning.

Condition
Varje process kan ha flera ingångsvillkor som måste vara uppfyllda för att processen ska påbörjas. På samma sätt kan varje process ha flera utgångsvillkor som måste vara uppfyllda för att processen ska avslutas. Ett villkor kan samtidigt utgöra utgångsvillkor för en process och ingångsvillkor för en annan. Villkor har ett namn och en textbeskrivning.

Information
Varje process kan ha flera informationselement. Information har en textbeskrivning och kan också ha flera dokument.

Document
Ett dokument kan ingå i flera informationselement. Dokument har ett namn, en textbeskrivning och kan också ha flera mallar.

Template
En mall har en textbeskrivning och kan ha flera länkar till fysiska mallar.

Method
En metod har namn och beskrivning och kan stödja flera processer. En metod kan också stödjas av flera verktyg.

Tool
Ett verktyg har namn och beskrivning och kan stödja flera metoder.

5 Slutsatser

Det finns inte ett sätt som är rätt när det gäller att utveckla inbyggda system eller dess ingående komponenter. Standardsamhället som är mycket stort har emellertid tagit fram många standarder som är användbara för att definiera en lämplig utvecklingsprocess.

Vid utveckling av produkter som innehåller flera teknikdiscipliner kan det vara lämpligt att använda flera processtandarder som beskriver arbetet med olika detaljnivå. Exempelvis kan det vara lämpligt att använda en övergripande processstandard för systemutveckling och en annan processtandard för utveckling av programvaran i systemet. Om man utvecklar en produkt inom en bransch som har tagit fram egna anpassade standarder kan det vara ett bättre alternativ att använda dem.

Vid utveckling av mekanik eller elektronik finns inga generella processtandarder som är direkt applicerbara, men inom särskilda branscher kan sådana återfinnas.

De flesta processtandarder som jag studerat innehåller likheter som gör att de delvis passar till produktutveckling i allmänhet. Det förekommer emellertid skillnader i terminologi även hos standarder som beskriver utveckling av i stort sett samma system. Inom programvaruutveckling har det funnits en rik flora av standarder, ibland med inbördes motsättningar. Det har resulterat i att organisationer som velat utveckla produkter på bästa möjliga sätt, har varit tvungna att jämföra standarderna på marknaden och ta det bästa ur varje.

För att komma till rätta med denna oönskade situation arbetar nu de på programvaruområdet två mest inflytelserika standardiseringsorganisationerna ISO/IEC JTC1 SC7 – Software & System Engineering och IEEE Computer Society – Software Enginering Standards Committee (SESC) på att få till stånd en harmoniering mellan olika standarder. Främst ska de två viktigaste processtandarderna ISO/IEC 12207 Software Life Cycle Process och ISO/IEC 15288 System Life Cycle Processes harmonieras med varandra när det gäller processer, terminologi, dokumentstruktur, detaljnivå, etcetera. Dessa ska även harmoniseras med andra näraliggande standarder.

Den teknik som är svårast att utveckla i ett inbyggt system är sannolikt programvaran. Programvara är i regel oerhört komplexa system med intrikata kopplingar mellan subsystemen. eXtreme Programming (XP) är en metodik för pro-

gramvaruutveckling. Den tillhör en ny generation av processer och metoder som kallas lättviktiga och snabba[7]. XP är en disciplinerad och omfattande metodik som beskriver ett antal praktiker och strategier för programvaruutvecklings flesta områden. Praktikerna och strategierna är baserade på underliggande värderingar, principer och upphovsmannens erfarenheter.

Processtandarden ISO/IEC 12207 representerar traditionell kunskap kring hur programvara bör utvecklas. På ett antal punkter finns det motsättningar mellan XP och 12207. Framförallt är det kravet i 12207 på att definiera och dokumentera ett stort antal informationselement som skapar konflikt med XP.

Medan processtandarder är skrivna i punktform finns XP formulerad i brödtext i litteraturen, vilket gör det svårare att hitta den information man söker.

En annan skiljelinje att ta fasta på är att XP uppskattar programkod medan traditionella processer förespråkar att modeller av programvaran ska konstrueras. Eftersom programkod är svårbegriplig är det en eftersträvansvärd ansats. De modelleringsspråk som finns idag, till exempel UML[8], kräver kompetens för att kunna utnyttjas på bästa sätt. Om programvaruutvecklare skall använda dessa modelleringsspråk i högre utsträckning än idag tror jag att de behöver få mer träning i att använda dem. Kanske behöver helt nya modelleringsspråk också tas fram.

Att arbeta mycket disciplinerat efter en utvecklingsprocess riskerar att ta arbetsglädjen från ingenjören, vilket kan resultera i stora förluster i produktivitet. Disciplin är emellertid fortfarande det enda kända sättet att försäkra sig om hög kvalitet.

6 Referenser

Litteratur

[Beck00] Beck, K. (2000) *Extreme programming explained: embrace change.* Addison-Wesley. ISBN 0-201-61641-6

Humphrey W. S. (2000) Introduction to the team software process. Reading: Addison-Wesley. ISBN 0-201-47719-X

[Moore98] Moore, J.W. (1998) *Software engineering standards: a user's road map.* Los Alamitos: IEEE Computer Society. ISBN 0-8186-8008-3

[Oskarsson95] Oskarsson, Ö. (1995) *ISO 9000 i programutveckling.* Lund: Studentlitteratur. ISBN 91-44-49531-5

[Pressman00] Pressman, R.S. (2000) *Software engineering: a practitioner's approach: European adaptation* (5. ed.) London: McGraw-Hill. ISBN 0-07-709677-0

[Sommerville01] Sommerville, I. (2001) *Software engineering.* (6. ed.) Harlow: Addison-Wesley. ISBN 0-201-39815-X

[Svensson03] Svensson, D. (2003) *Towards Product Structure Management in Heterogeneous Environments*. Göteborg: Chalmers University of Technology. ISBN 91-7291-275-8

[7] Engelskans "agile" används.

[8] UML – Unified Modeling Language

Workflow Management Coalition (1998) Interface 1: Process Definition Interchange Q&A and Examples (TC00-1016)

Elektroniska källor

[Bunse] Bunse C. (2002) Out of the dark: Adaptable process models for XP (2003-03-20)

[Booch98] Booch G. (1998) The process. URL: (2003-03-20)

[Martin03] Martin J. (2003) Slides från INCOSE möte. (2003-03-20)

[Martin98] Martin J. (1998) Tutorial G. URL: (2003-03-20)

[ANSI] *ANSI.* URL: http://www.ansi.org/ (2003-03-20)

[CMMI] (1999) URL: http://www.sei.cmu.edu/cmmi/presentations/euro-sepg-tutorial/sld099.htm (2003-02-27)

[Coallier] Coallier, F. (2003) International standardization in software and systems engineering. *STSC CrossTalk,* February. URL: http://www.stsc.hill.af.mil/crosstalk/2003/02/coallier.html (2003-02-21)

[ECSS] Collaboration website of the ECSS. (2002) *The European Space Agency*. URL: http://www.ecss.nl/ (2003-03-20)

[ESA] ESA standards information. (2002) *The European Space Agency*. URL: http://www.estec.esa.nl/pr/estecinfo/standards.php3 (2003-03-20)

[G-34] Software Sub-committee. (2003) *Electronic Industries Alliance.* URL: http://www.geia.org/sstc/G-34/g34.htm

[IEC61508] Functional safety zone (2003) *International Electrotechnical Commission.* URL: http://www.iec.ch/61508 (2003-03-20)

[ISO] International organization for standardization. (2003) URL: http://www.iso.org/ (2003-03-20)

GEIA. URL: http://www.geia.org/ (2003-03-20)

IEEE. URL: http://www.ieee.org/ (2003-03-20)

MIL-Standards.com. ULR: http://store.mil-standards.com/ (2003-03-20)

[OOSpice] OOSpice. URL: http://www.oospice.com/ (2003-03-20)

[QS-9000] ASQ: Standards. URL: http://qs9000.asq.org (2003-03-20)

[Software] Software productivity consortium. (2003) URL: http://www.software.org/ (2003-03-20)

[SS] URL: http://www.svenskstandard.org.se (2003-03-20)

[Techstreet] Information technology: software life cycle processes: amendment 1. (2003) *Techstreet.* URL: http://www.techstreet.com/cgi-bin/detail?product_id=1035809 (2003-03-20)

15288 Home Page. (1999) URL: http://www.15288.com/ (2003-03-20)

Appendix 1: Standardiseringsorganisationer

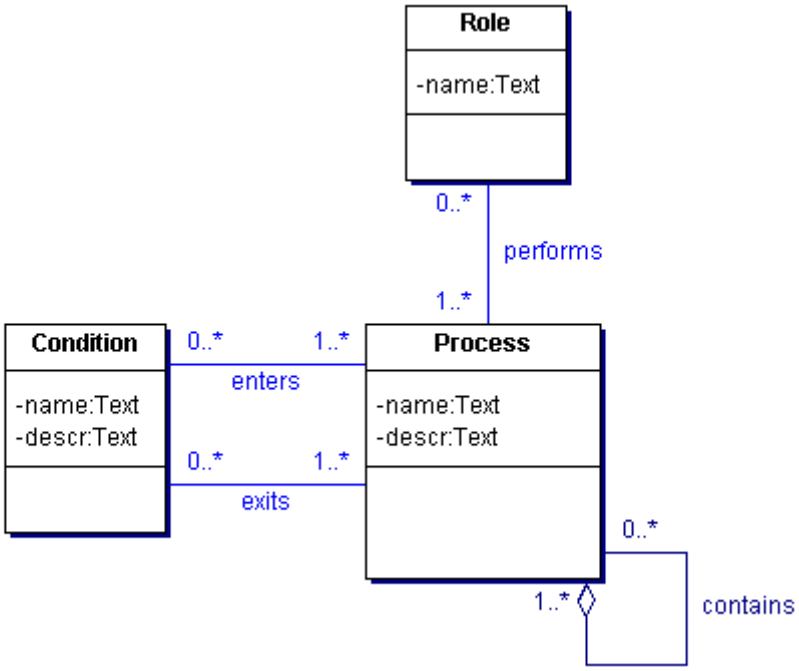
American Institute of Aeronautics and Astronautics (AIAA)
American National Standards Institute (ANSI)
Automotive Industry Action Group (AIAG)
Canadian Standards Association (CSA)
Electronic Industries Alliance (EIA)
European Association of Aerospace Industries (AECMA)
European Committee for Electrotechnical Standardization (CENELEC)
European Committee for Standardization (CEN)
European Cooperation for Space Standardization (ECSS)
European Telecommunications Standards Institute (ETSI)
Institute of Electrical and Electronic Engineers (IEEE)
(IEEE Computer Society CS Store - IEEE Standards)
Government Electronics and Information Technology Association (GEIA)
Inter-American Telecommunication Commission (CITEL)
International Automotive Sector Group (IASG)
International Council on Systems Engineering (INCOSE)
International Electrotechnical Commission (IEC)
International Organization for Standardization (ISO)
International Telecommunication Union (ITU)
Object Management Group (OMG)
Project Management Group (PMI)
Radio Technical Comission for Aeronautics (RTCA)
Society of Automotive Engineers (SAE)
Swedish Standards Institute (SIS)
Telecommunications Industry Association (TIA)
World Wide Web Consortium (W3C)

Appendix 2: Konstruktionsunderlag för demonstrator

Utvecklingen av demonstratorn har gjorts i enlighet med den objektorienterade metod som ingår i RUP[9] och som beskrivs av C. Larman i *Applying UML and Patterns*[10].

Domänmodell

Den förenklade domänmodell som beskrivs nedan innehåller endast de klasser som är implementerade i demonstratorn.



[9] RUP – Rational Unified Process

[10] Larman C. (2002) *Applying UML and Patterns*. (2. ed.) Upper Saddle River: Prentice Hall. ISBN 0-13-092569-1

Krav

Alla projektintressenters överenskomna krav på programvaruprodukten specificeras här i detta avsnitt.

Problemformulering

Det finns ett behov av ett datorbaserat utvecklingsverktyg som stödjer framtagning och anpassning av en organisations eller ett projekts produktutvecklingsprocess. Produktkategorin som här avses är inbyggda system som är programvaruintensiva.

Funktionella krav

De funktionella kraven beskrivs genom ett antal användningsfall. Användningsfallen har tilldelats en prioritet som är hög (H), medium (M) eller låg (L). De användningsfall som har högst prioritet bör implementeras först i systemet.

Användningsfall	*Prioritering*
1. Skapa ny process	H
2. Infoga process	H
3. Välj processens namn	M
4. Välj processens beskrivning	M
5. Välj processens roll	M
6. Välj processens ingångsvillkor	M
7. Välj processens utgångsvillkor	M
8. Öppna fil	H
9. Spara fil	H
10. Utläs information om process	H
11. Välj processens använda information	L
12. Välj processens producerade information	L
13. Välj processens metoder	L
14. Välj metodens verktyg	L
15. Välj informationens dokument	L
16. Välj dokumentets mallar	L
17. Utläs information om entitet	L
18. Ändra information om entitet	L
19. Ta bort process	L
20. Ta bort entitet	L

Nedan följer detaljerade beskrivningar av de användningsfall som har tilldelats hög eller medium prioritet.

USE CASE 1	Skapa ny process	
Preconditions	Systemet är inte låst i någon dialogruta.	
Success End Condition	Det finns ett tomt processträd (endast rotnod) i trädfönstret.	
Primary Actor	Användaren	
DESCRIPTION	**Step**	**Action**
	1	Användare begär kommando *New File.*
	2	Ett tomt träd (endast rotnod) blir synligt i trädfönstret.
EXTENSIONS	**Step**	**Branching Action**
	1a	En processfil är redan öppen: 1a1. Systemet frågor om processenfilen ska sparas. 1a2. Användaren väljer *Yes*, *No* eller *Cancel*. 1a3. Om användaren väljer *Yes* sparas den öppna processfilen (use case 9)

USE CASE 2	Infoga process	
Preconditions	Ett processträd är öppet. Systemet är inte låst i någon dialogruta.	
Success End Condition	Processträdet innehåller en ny process på önskad position.	
Primary Actor	Användaren	
DESCRIPTION	**Step**	**Action**
	1	Användaren begär kommando *Insert Process*.
EXTENSIONS	**Step**	**Branching Action**
	1a	En processnod är markerad: 1a1. Processen infogas på nivån under den markerade noden.
	1b	Ingen processnod är markerad: 1b1. Processen infogas på nivån under rotnoden.

USE CASE 3	Välj processens namn	
Preconditions	Processens informationsfönster är öppet. Systemet är inte låst i någon dialogruta.	
Success End Condition	Processens namn har ändrats i informationsfönstret och i trädföntstret.	
Primary Actor	Användaren	
DESCRIPTION	**Step**	**Action**
	1	Användaren klickar i processens namnfält.
	2	Användaren matar in processens namn.
	3	Användaren konfirmerar det nya namnet.
SUB-VARIATIONS		**Branching Action**
	3	Användaren kan konfirmera nytt namn genom att: trycka Enter på tangentbordet, klicka på annat objekt i framen

USE CASE 4	Välj processens beskrivning	
Preconditions	Processens informationsfönster är öppet. Systemet är inte låst i någon dialogruta.	
Success End Condition	Processens beskrivning har ändrats i informationsfönstret.	
Primary Actor	Användaren	
DESCRIPTION	**Step**	**Action**
	1	Användaren klickar i processens beskrivningsfält.
	2	Användaren matar in processens beskrivning.
	3	Användaren konfirmerar den nya beskrivningen.
SUB-VARIATIONS		**Branching Action**
	3	Användaren kan konfirmera nytt namn genom att: trycka Enter på tangentbordet, klicka på annat objekt i framen

USE CASE 5	Välj processens roll	
Preconditions	Processens informationsfönster är öppet. Systemet är inte låst i någon dialogruta.	
Success End Condition	En process har fått sin utförande roll definierad.	
Primary Actor	Användaren	
DESCRIPTION	**Step**	**Action**
	1	Användaren klickar på rollväljarsymbolen.
	2	Systemet visar rollväljardialogen med namnen på de roller som finns inmatade i systemet.
	3	Användaren markerar och väljer roll.
	Börja om på steg 2.	
	4	Användaren stänger rollväljardialogen.
EXTENSIONS	**Step**	**Branching Action**
	3a	Ingen av rollerna passar processen: 3a1. Användaren väljer *New Role*. 3a2. Systemet begär inmatning av den nya rollens namn. 3a3. Användaren matar in namn.

USE CASE 6	Välj processens ingångsvillkor	
Preconditions	Processens informationsfönster är öppet. Systemet är inte låst i någon dialogruta.	
Success End Condition	En process har fått sina ingångsvillkor definierade.	
Primary Actor	Användaren	
DESCRIPTION	**Step**	**Action**
	1	Användaren klickar på symbolen för att välja ingångsvillkor.
	2	Systemet presenterar namnen på de villkor som finns inmatade i systemet.
	3	Användaren markerar och väljer villkor.
	Börja om på steg 2.	
	4	Användaren stänger villkorväljardialogen.
EXTENSIONS	**Step**	**Branching Action**
	3a	Villkoren passar inte processen: 3a1. Användaren väljer *New Condition.* 3a2. Systemet begär inmatning det nya villkorets namn. 3a3. Användaren matar in namn.

USE CASE 7	Välj processens utgångsvillkor	
Preconditions	En processnod är markerad.	
Success End Condition	En process har fått sina utgångsvillkor definierade.	
Primary Actor	Användaren	
Trigger		
DESCRIPTION	**Step**	**Action**
	1	Användaren klickar på symbolen för att välja utgångsvillkor.
	2	Systemet presenterar namnen på de villkor som finns inmatade i systemet.
	3	Användaren markerar och väljer villkor.
	Börja om på steg 2.	
	4	Användaren stänger villkorväljardialogen.
EXTENSIONS	**Step**	**Branching Action**
	3a	Villkoren passar inte processen: 3a1. Användaren väljer *New Condition.* 3a2. Systemet begär inmatning det nya villkorets namn. 3a3. Användaren matar in namn.

USE CASE 8	Öppna fil	
Preconditions	Systemet är inte låst i någon dialog.	
Success End Condition	En kopia av sparad processfil finns öppen i trädfönstret.	
Primary Actor	Användaren	
DESCRIPTION	**Step**	**Action**
	1	Användaren begär kommando *Open File*.
	2	Systemet presenterar värddatorns existerade mappstruktur och begär val av processfil.
	3	Användaren väljer fil.
	4	Systemet öppnar fil.
EXTENSIONS	**Step**	**Branching Action**
	2a	En processfil är redan öppet: 2a1. Systemet frågor om processfilen ska sparas. 2a2. Användaren väljer *Yes, No* eller *Cancel*. 2a3. Om användaren väljer *Yes* sparas den öppna processfilen (use case 9)

USE CASE 9	Spara fil	
Preconditions	Det finns ett öppet processträd. Systemet är inte låst i någon dialog.	
Success End Condition	En kopia av det öppna processträdet har sparats i ett icke-flyktigt minne.	
Primary Actor	Användaren	
DESCRIPTION	**Step**	**Action**
	1	Användare begär kommando *Save File*.
	2	Systemet presenterar värddatorns existerade mappstruktur och begär inmatning av filnamn.
	3	Användaren väljer mapp och matar in filnamn.
EXTENSIONS	**Step**	**Branching Action**
	3a	Användaren matar in felaktigt filnamn: 3a1. Systemet begär ny inmatning av filnamn.

USE CASE 10	Utläs information om process	
Preconditions	Ett processträd är öppet. Systemet är inte låst i någon dialogruta.	
Success End Condition	Processträdet innehåller en ny process på önskad position.	
Primary Actor	Användaren	
DESCRIPTION	**Step**	**Action**
	1	Användaren bläddrar i processträdet och markerar en nod.
	2	Systemet presenterar information om processen i informations-fönstret.

Utökad funktionalitet

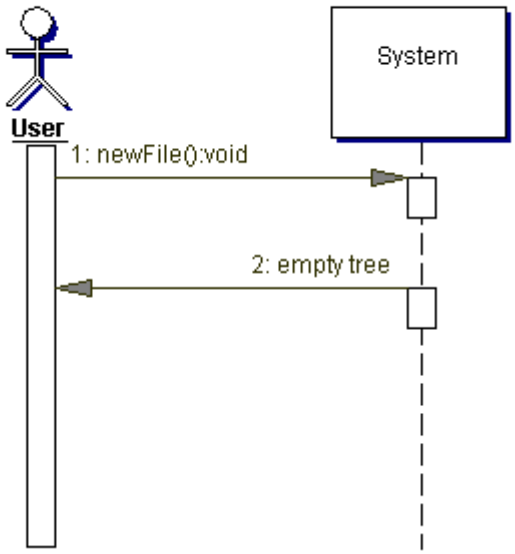
Systemet skulle kunna byggas ut med den funktionalitet som beskrivs nedan.

- Använda olika processmodeller
- Skräddarsy processer
- Spara skräddarsydda processer
- Öppna skräddarsydda processer
- Modifiera skräddarsydda processer
- Ta bort skräddarsydda processer
- Skriva ut processbeskrivningen
- Exekvera processen
- Aktivitetsvägledning
- Datainsamling: kompileringsdefekter, testningsdefekter, tidtagning
- Datarapportering
- Möjlighet att hantera flera användare i ett distribuerat system

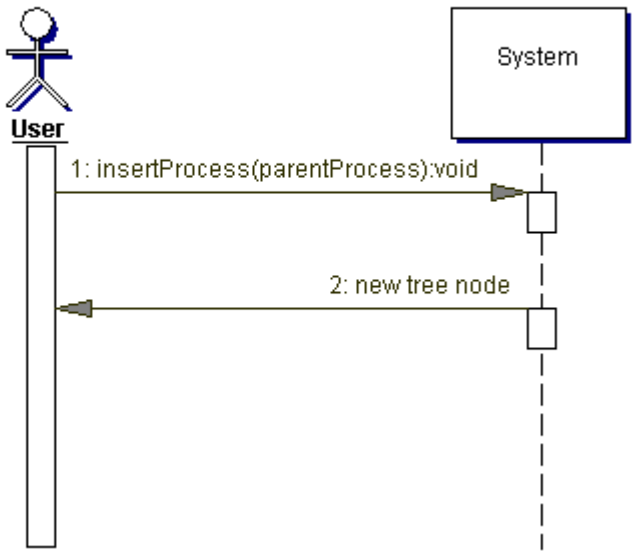
Systemsekvensdiagram

Nedan beskrivs de olika operationer som användaren kan begära genom att generera olika systemhändelser.

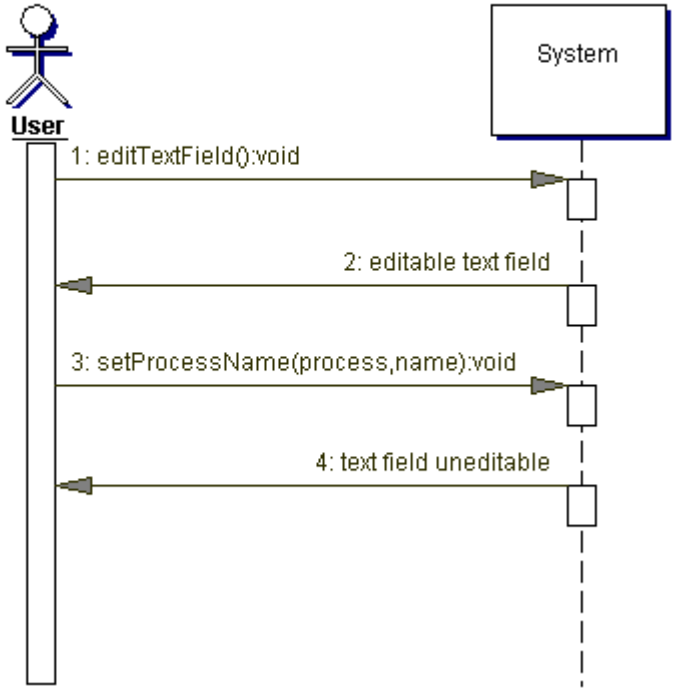
1. Skapa ny process



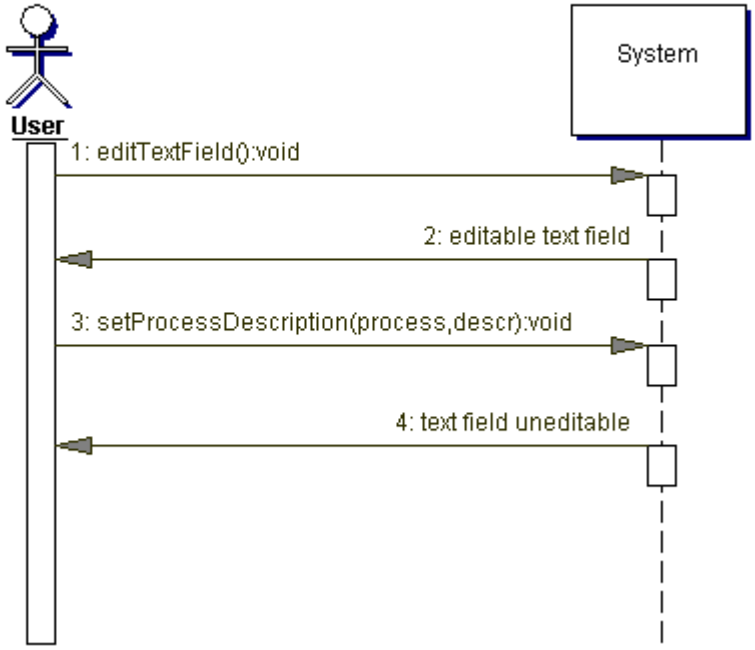
2. Infoga process



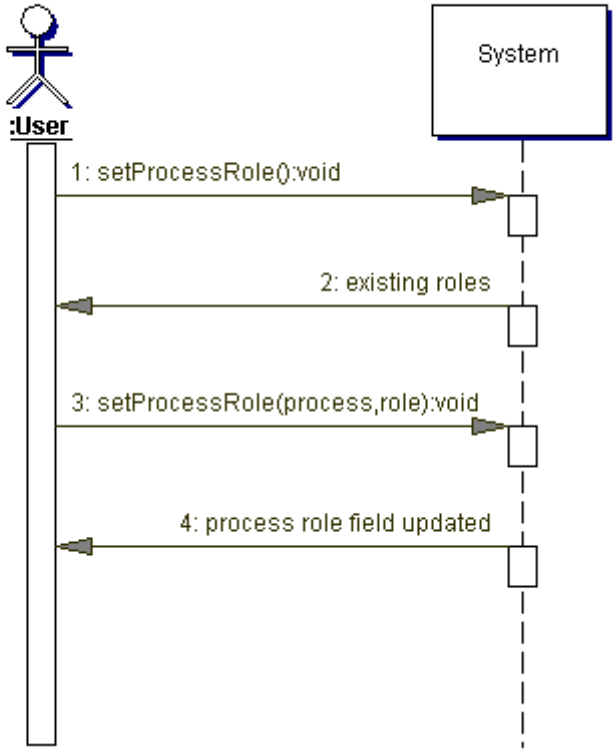
3. Välj processens namn



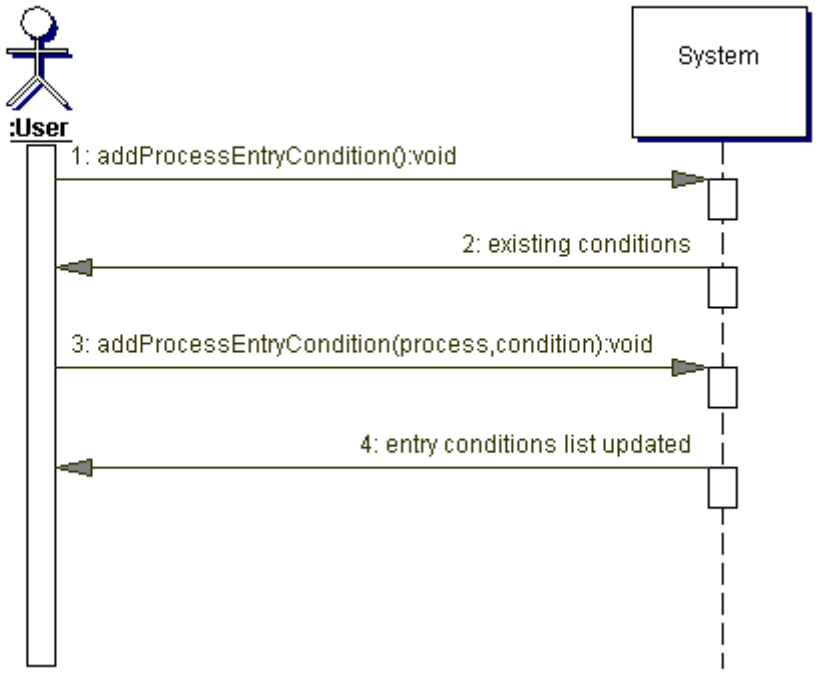
4. Välj processens beskrivning



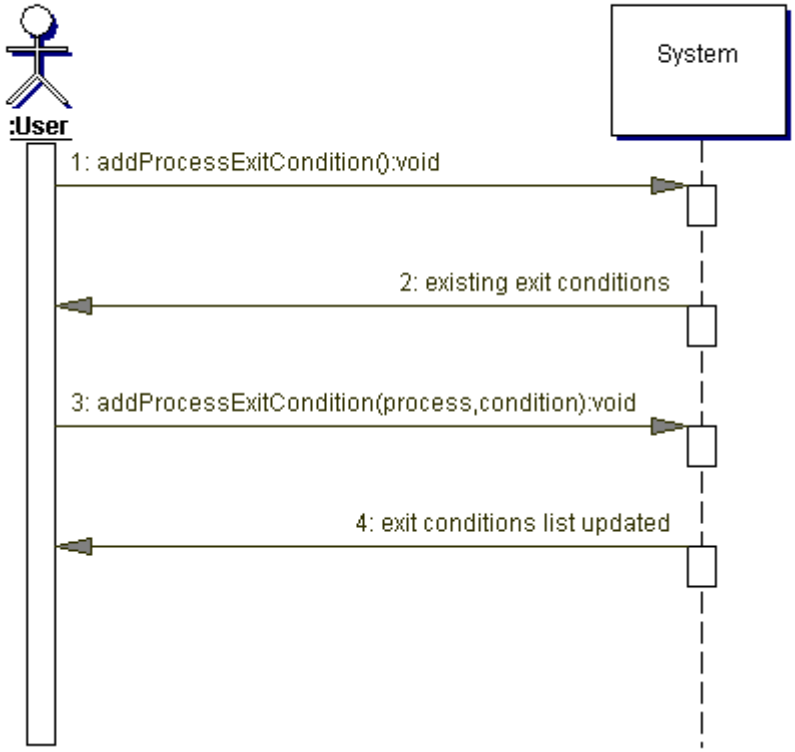
5. Välj processens roll



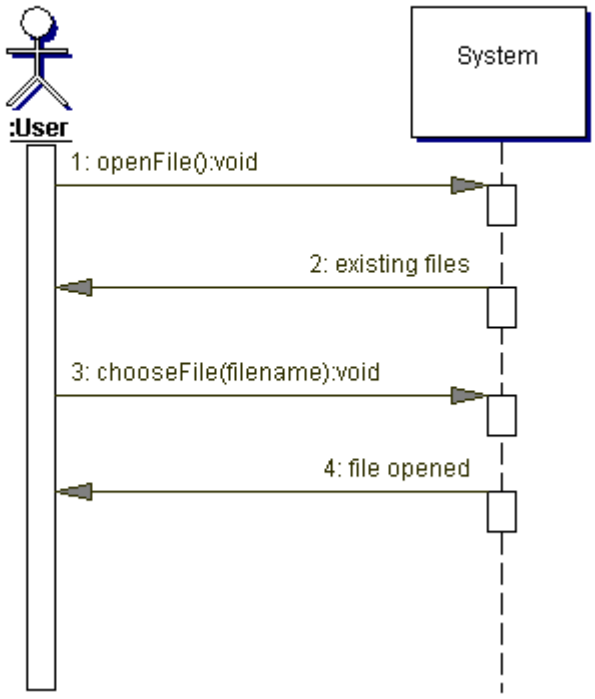
6. Välj processens ingångsvillkor



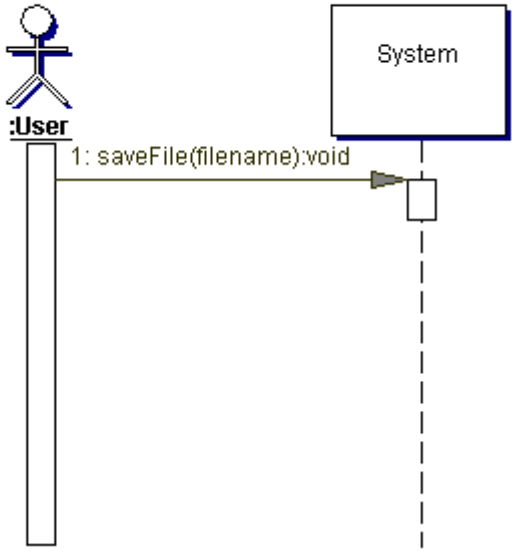
7. Välj processens utgångsvillkor



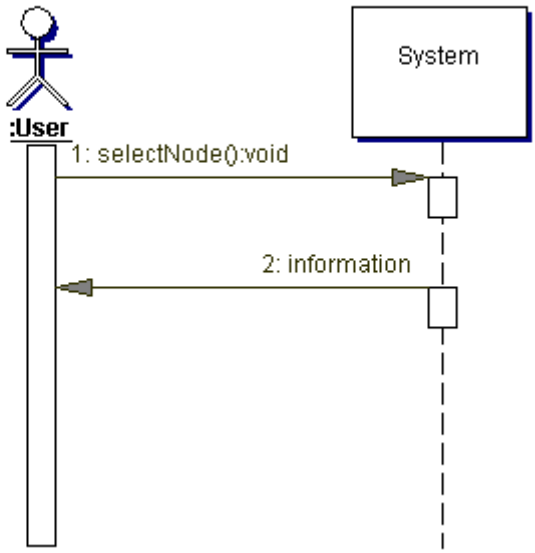
8. Öppna fil



9. Spara fil



10. Utläs information om process



Kontrakt för systemoperationer

I detta avsnitt beskrivs kontrakt för systemets operationer. Kontrakten beskriver tillståndförändringar för objekten i domänmodellen. Kontrakten är indelade efter användningsfallen och ger ytterligare information om de systemoperationer som beskrivs av systemsekvensdiagrammen.

1. Skapa ny process

Contract C1.1: *newFile*

Operation: newFile()
Preconditions:
Postconditions: - En instans p av Process skapades.
- p:s attribut initierades.

2. Infoga process

Contract C2.1: *insertProcess*

Operation: insertProcess(parent: Process)
Preconditions: Det finns ett processträd (minst en instans av Process).
Postconditions: - En instans p av Process skapades.
- p:s attribut initierades.
- p associerades med parent

3. Välj processens namn

Contract C3.1: *setProcessName*

Operation: setProcessName(p: Process, name: Text)
Preconditions: Karaktäristika för en process matas in
Postconditions: - p.name blev name

4. Välj processens beskrivning

Contract C4.1: *setProcessDescription*

Operation: setProcessDescription(p: Process, descr: Text)
Preconditions: Karaktäristika för en process matas in
Postconditions: - p.descr blev descr

5. Välj processens roll

Contract C5.1: *setProcessRole*

Operation: setProcessRole()
Preconditions: Karaktäristika för en process matas in
Postconditions:

Contract C5.2: *setProcessRole*

Operation: setProcessRole(p: Process, r: Role)
Preconditions: En roll läggs till processen
Postconditions: - r associerades med p

Contract C5.3: *newRole*

Operation: newRole(descr: Text)
Preconditions: En ny roll skapas.
Postconditions: - En instans r av Role skapades.
- r.descr blev descr

6. Välj processens ingångsvillkor

Contract C6.1: *addProcessEntryCondition*

Operation: addProcessEntryCondition()
Preconditions: Karaktäristika för en process matas in
Postconditions:

Contract C6.2: *addProcessEntryCondition*

Operation: addProcessEntryCondition(p: Process, entryC: Condition)
Preconditions: Ett ingångsvillkor läggs till processen
Postconditions: - entryC associerades med p

Contract C6.3: *newCondition*

Operation: newCondition(descr: Text)
Preconditions: Ett nytt villkor skapas.
Postconditions: - En instans c av Condition skapades.
- c.descr blev descr

7. Välj processens utgångsvillkor

Contract C7.1: *addProcessExitCondition*

Operation: addProcessExitCondition()
Preconditions: Karaktäristika för en process matas in
Postconditions:

Contract C7.2: *addProcessExitCondition*

Operation: addProcessExitCondition(p: Process, exitC: Condition)
Preconditions: Ett utgångsvillkor läggs till processen
Postconditions: - exitC associerades med p

8. Öppna fil

Contract C8.1: *openFile*

Operation:	openFile()
Preconditions:	En fil öppnas.
Postconditions:	

Contract C8.2: *chooseFile*

Operation:	chooseFile(filename: FileName)
Preconditions:	En fil öppnas.
Postconditions:	- Ett antal instanser av entiteter (processer, roller, m.m.) återskapas. - Instansernas attributvärden återskapas. - Instansernas associationer återskapas.

9. Spara fil

Contract C9.1: *saveFile*

Operation:	saveFile(filename:FileName)
Preconditions:	En fil sparas.
Postconditions:	

10. Utläs information om process

Inga kontrakt skrivs för detta användningsfall eftersom det inte medför några förändringar i domänmodellen.

Design

Designen av demonstratorn är baserad på en skiktad arkitektur. Det finns tre skikt eller lager som alla utgör var sitt paket: presentation, applikation och lagring. Presentationslagret har ansvar för interaktionen mellan användaren och systemet. Användaren kan begära olika systemoperationer som presentationslagret hanterar självt eller skickar vidare till applikationslagret.

Presentationslagret avspeglar grafiskt tillståndet hos applikationslagret. För att implementera den avspeglingen har designmönstret Observer använts. I enlighet med Observer observerar presentationslagret applikationslagret. Eftersom applikationslagret har designats med en applikationsfasad observerar presentationslagret endast applikationsfasaden.

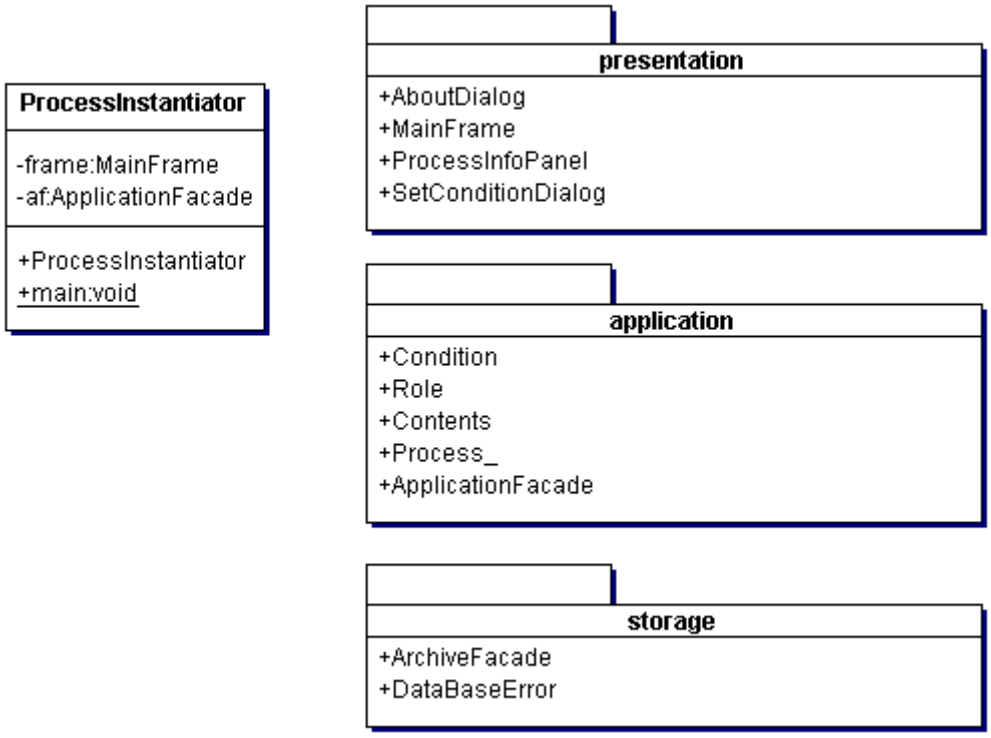
Designmönstret Facade innebär att alla funktioner de olika delsystemen i en komponent tillhandahåller samlas i ett gemensamt gränssnitt. Det blir då mycket enklare att använda komponenten eftersom inte alla dess delsystem måste förstås.

Applikationslagret ansvarar för den logik som utgör själva applikationen medan lagringslagret ansvarar för lagring av data.

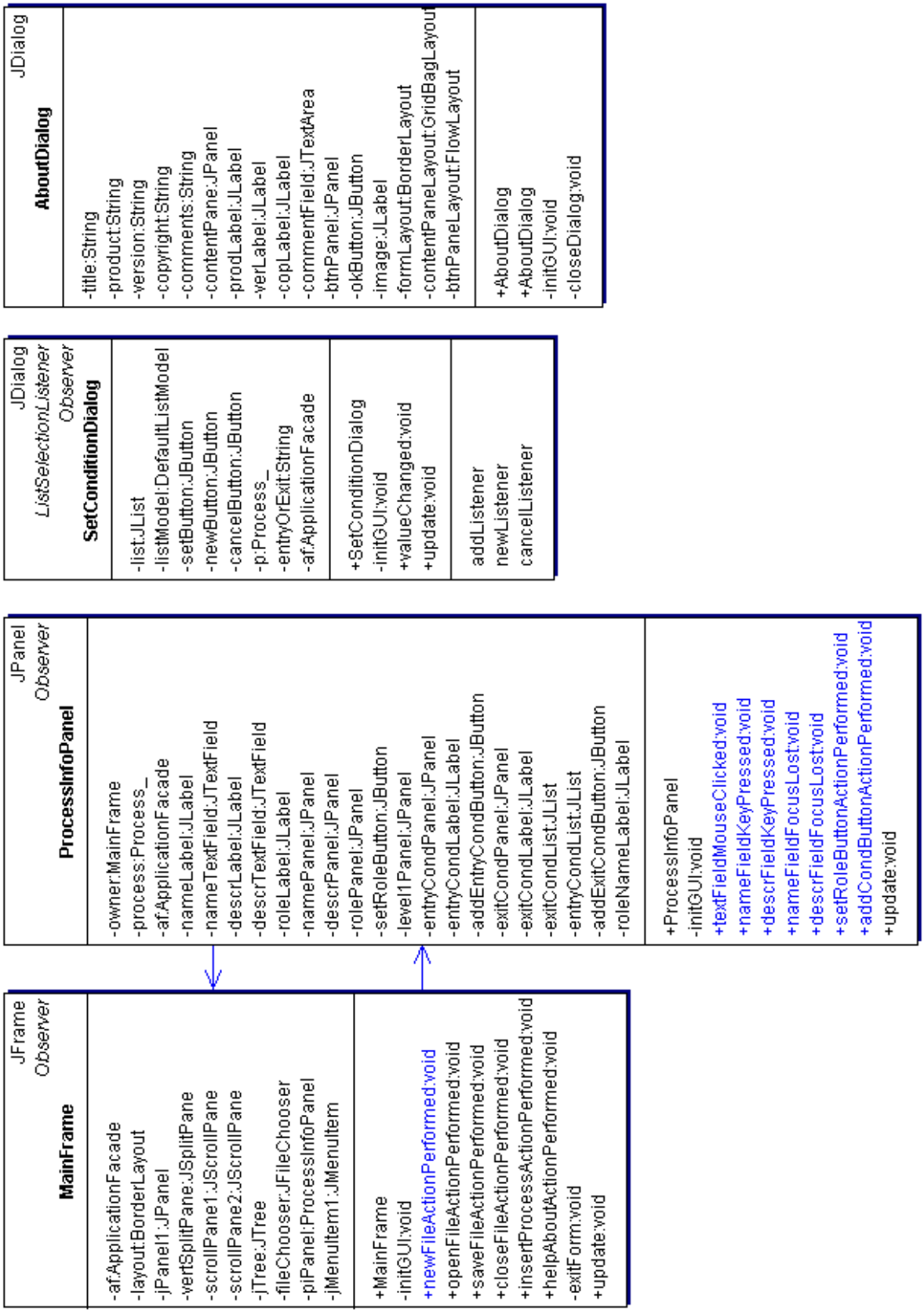
Applikationsfasaden har designats som en Singleton, vilket är ett designmönster som garanterar att klassen alltid har en och en endast en instans. Dessutom tillhandahåller den en global referens till denna instans, vilket i detta fall behövs för att presentationslagret ska kunna nå den.

I detta avsnitt presenteras designen av demonstratorns statiska och dynamiska beteende. Den statiska vyn av systemet består av UML-klassdiagram, som visar vilka attribut och metoder klasserna innehåller. Den dynamiska vyn av systemet är inte komplett men består av ett antal UML-sekvensdiagram som visar i vilken ordning objekt skapas och metoder anropas i applikationslagret. Främst visas sekvensdiagram för metoder som motsvarar systemoperationer (se Kontrakt för systemoperationer).

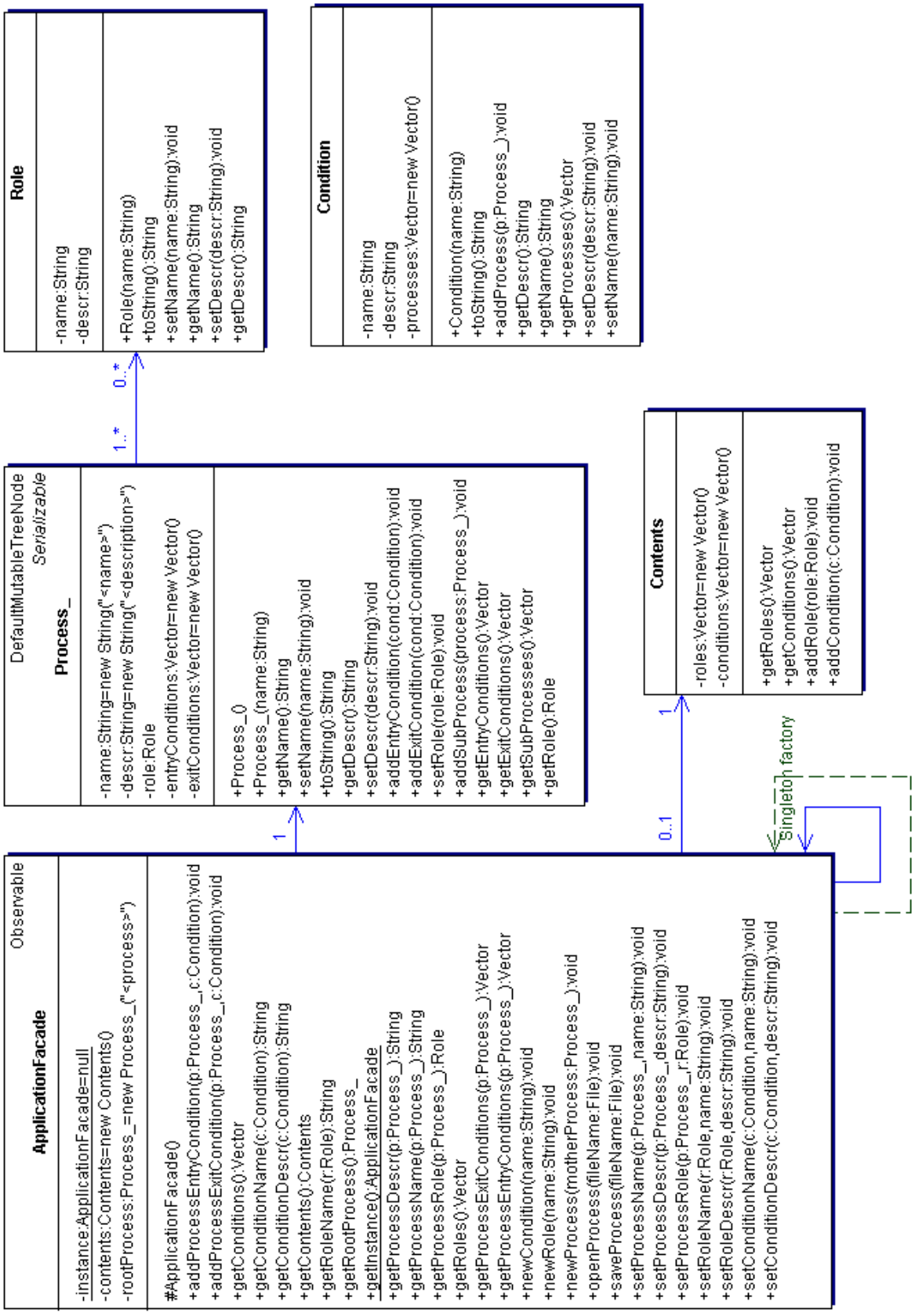
Statisk vy


ProcessInstantiator
-frame:MainFrame
-af:ApplicationFacade
+ProcessInstantiator
+main:void
presentation
+AboutDialog
+MainFrame
+ProcessInfoPanel
+SetConditionDialog
application
+Condition
+Role
+Contents
+Process_
+ApplicationFacade
storage
+ArchiveFacade
+DataBaseError

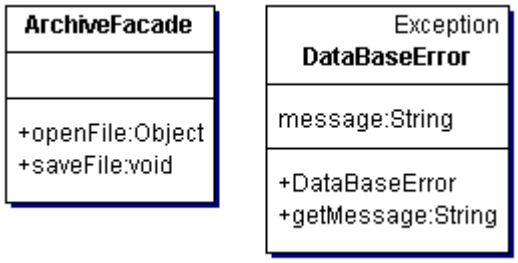
Presentation



Application

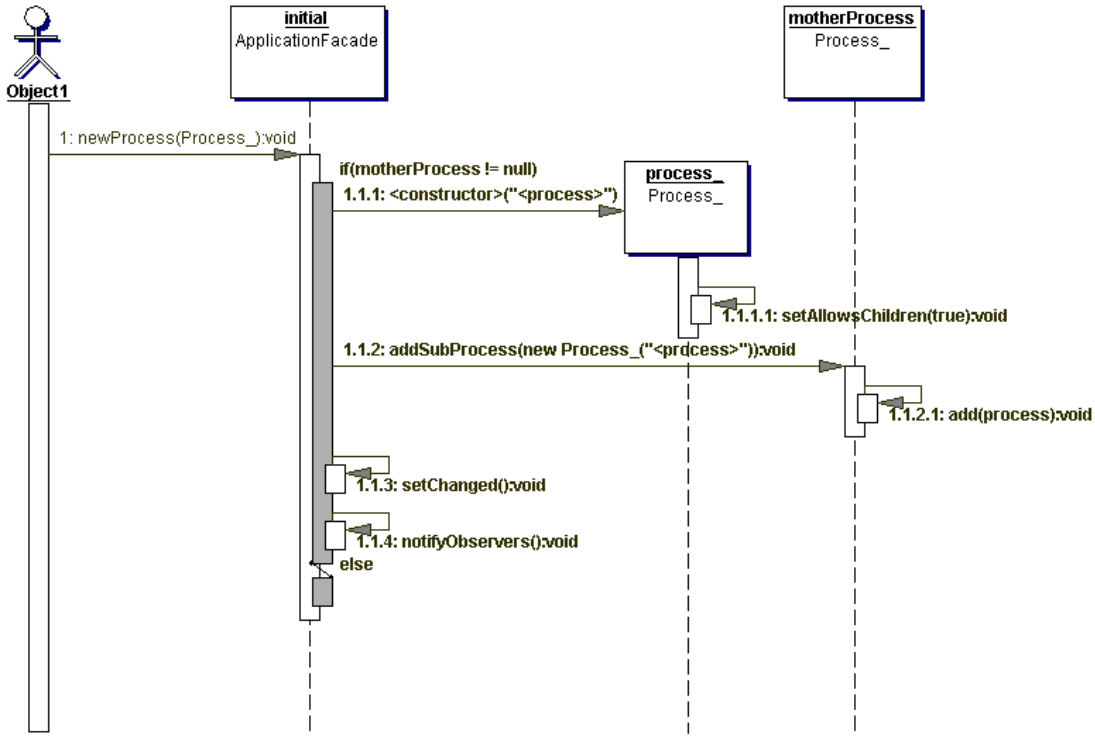


Storage

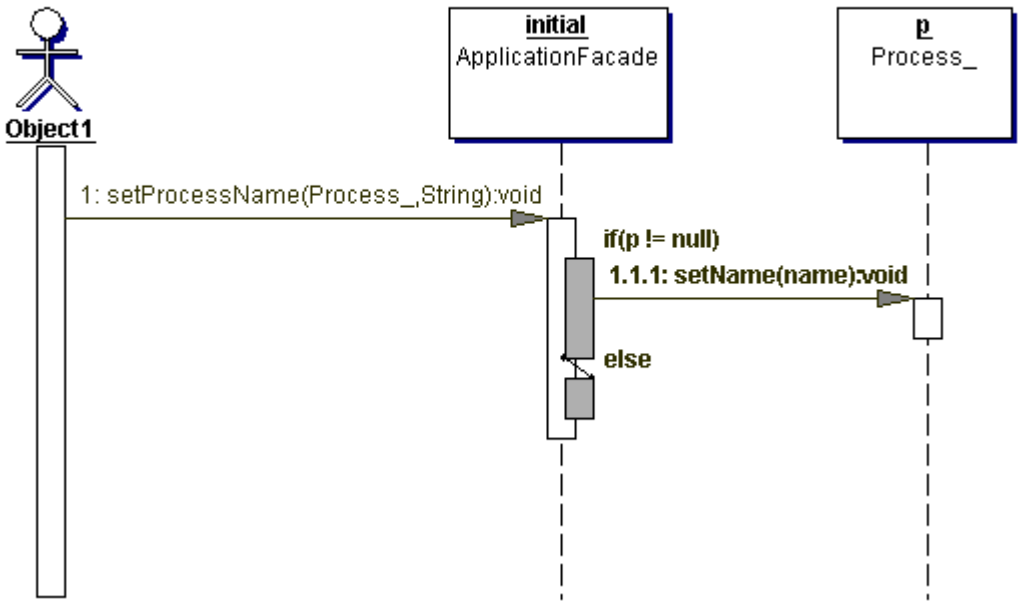

ArchiveFacade
+openFile:Object
+saveFile:void
Exception
DataBaseError
message:String
+DataBaseError
+getMessage:String

Dynamisk vy

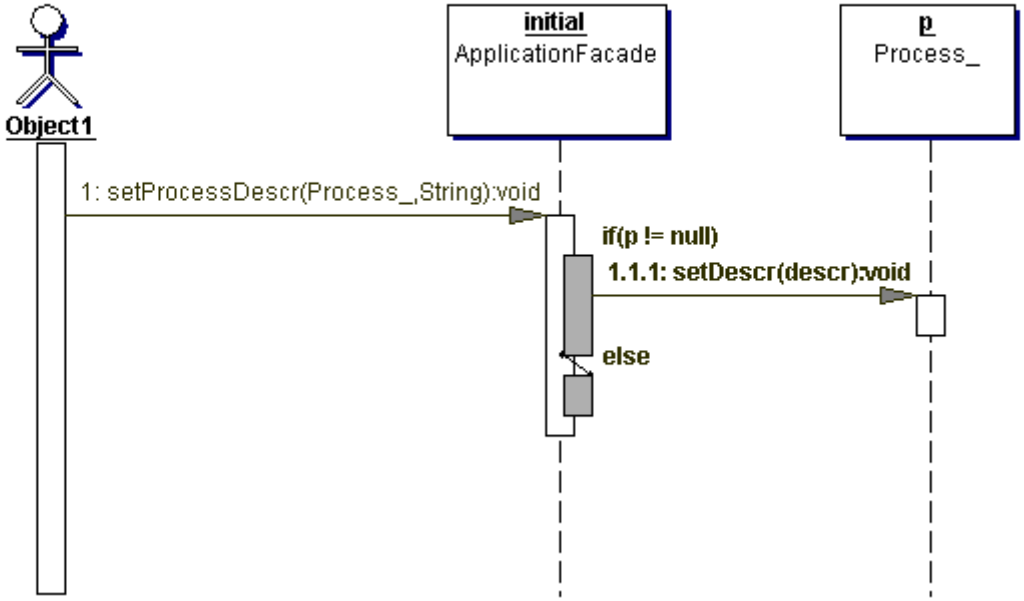
2. Infoga process



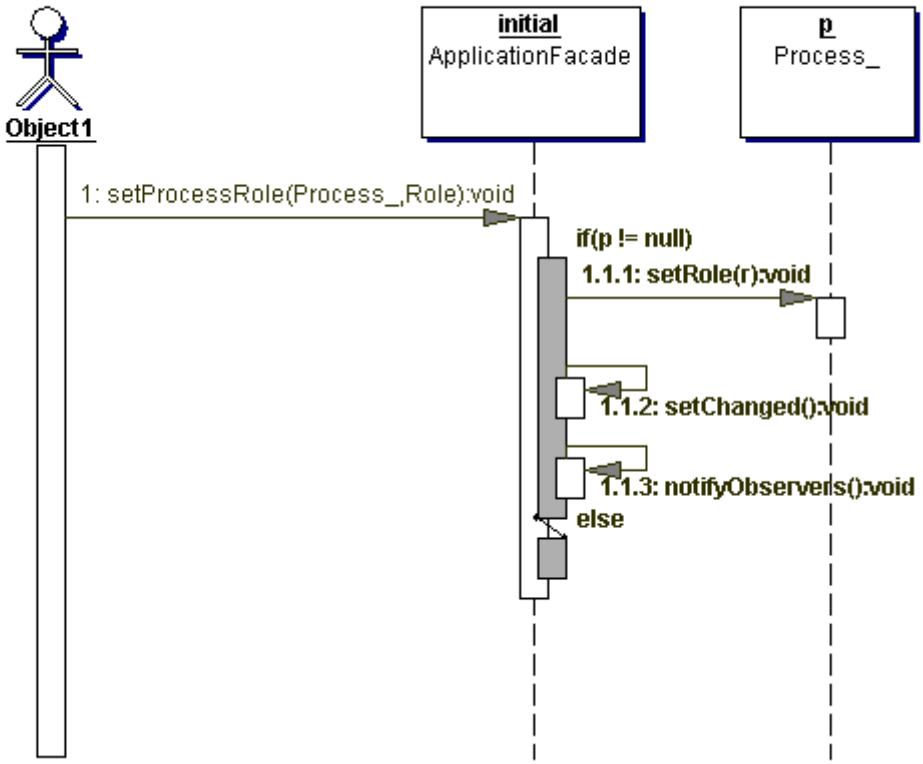
3. Välj processens namn

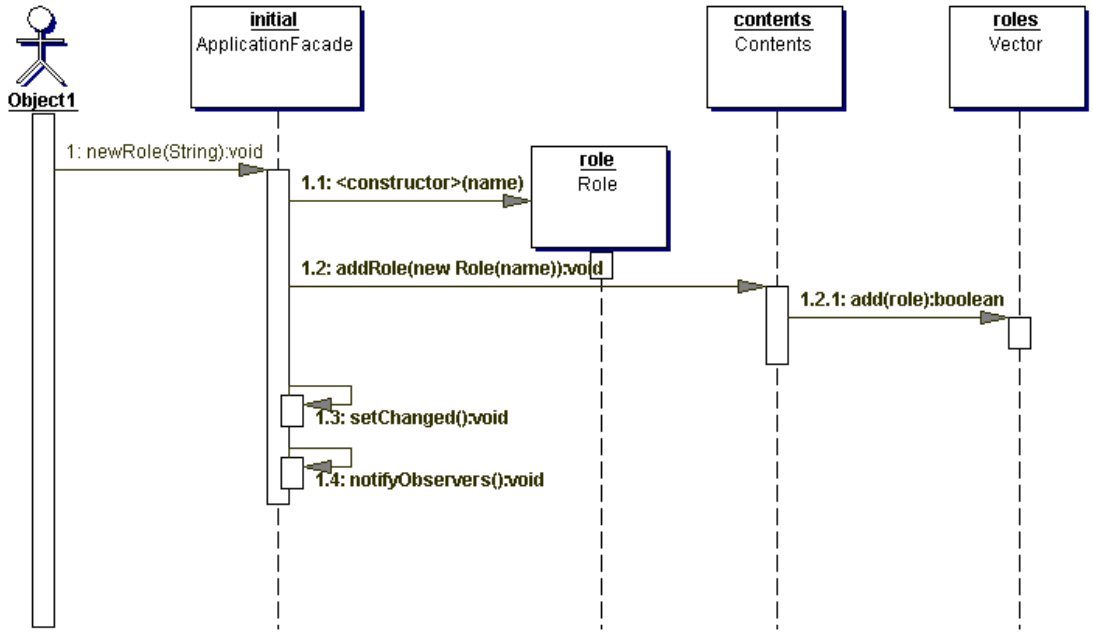


4. Välj processens beskrivning

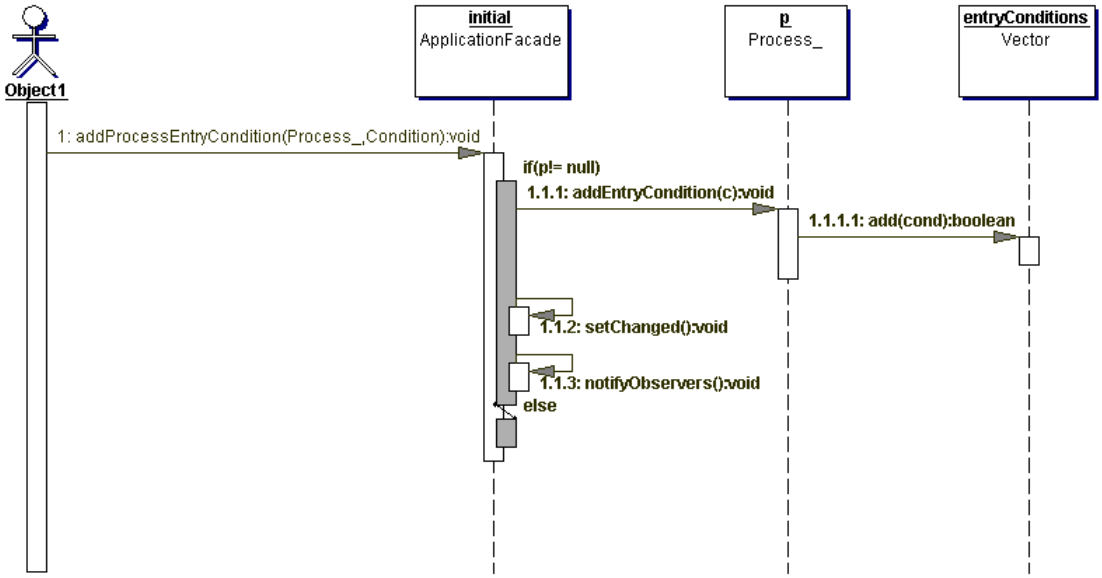


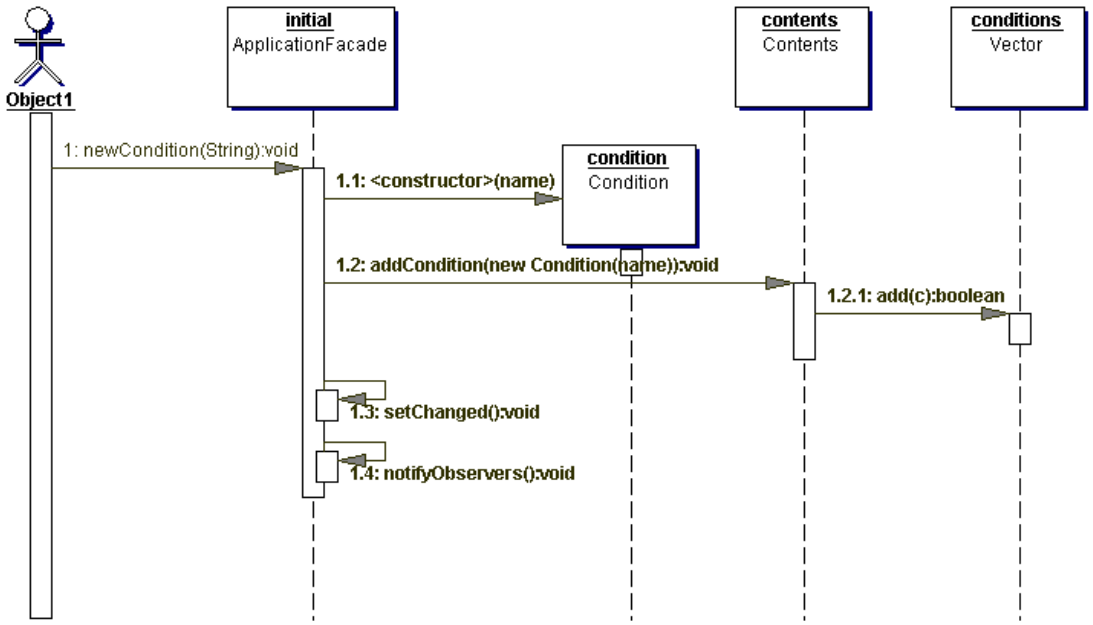
5. Välj processens roll



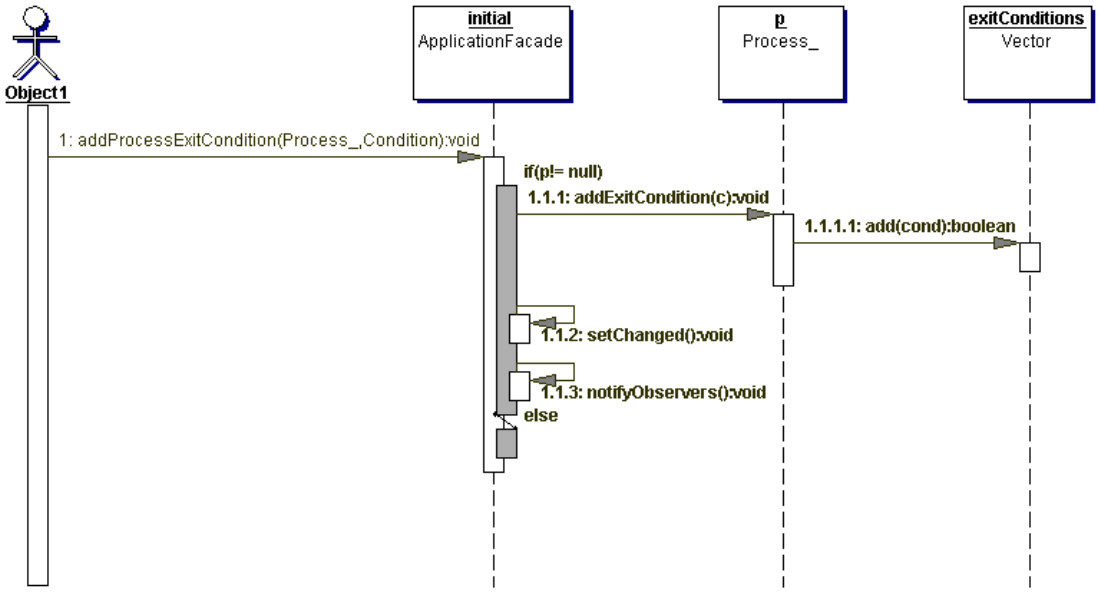


6. Välj processens ingångsvillkor

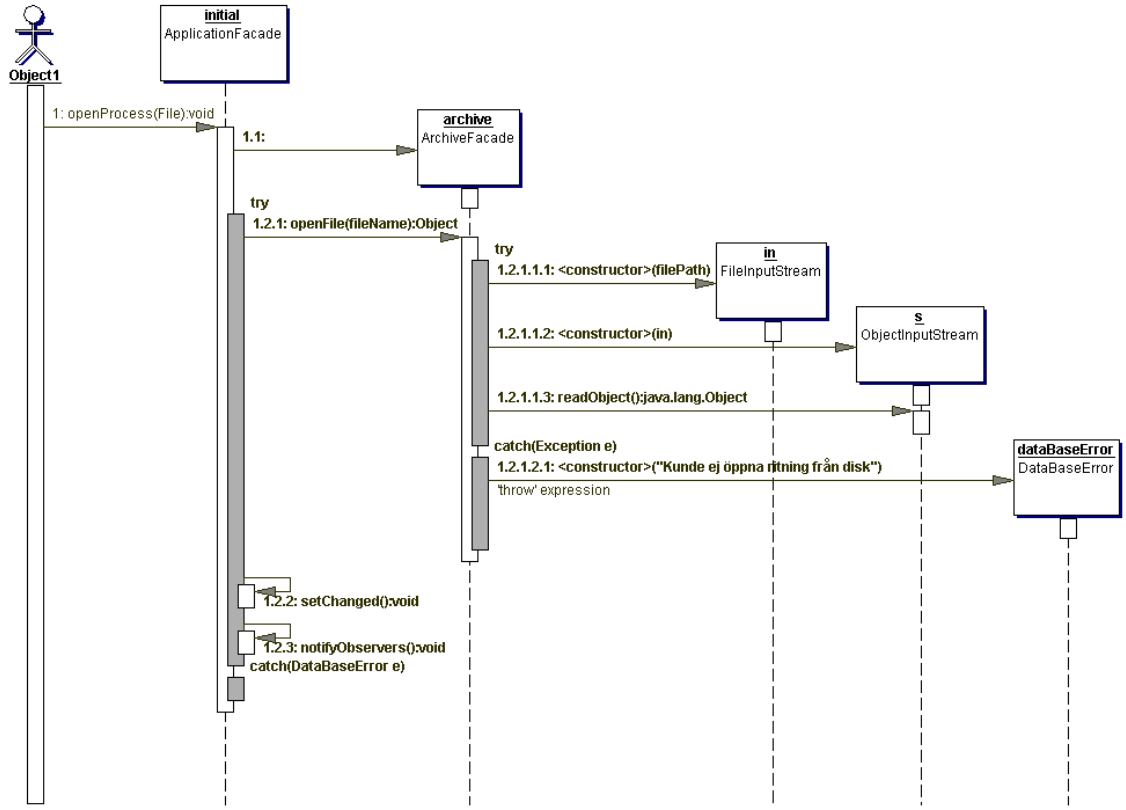




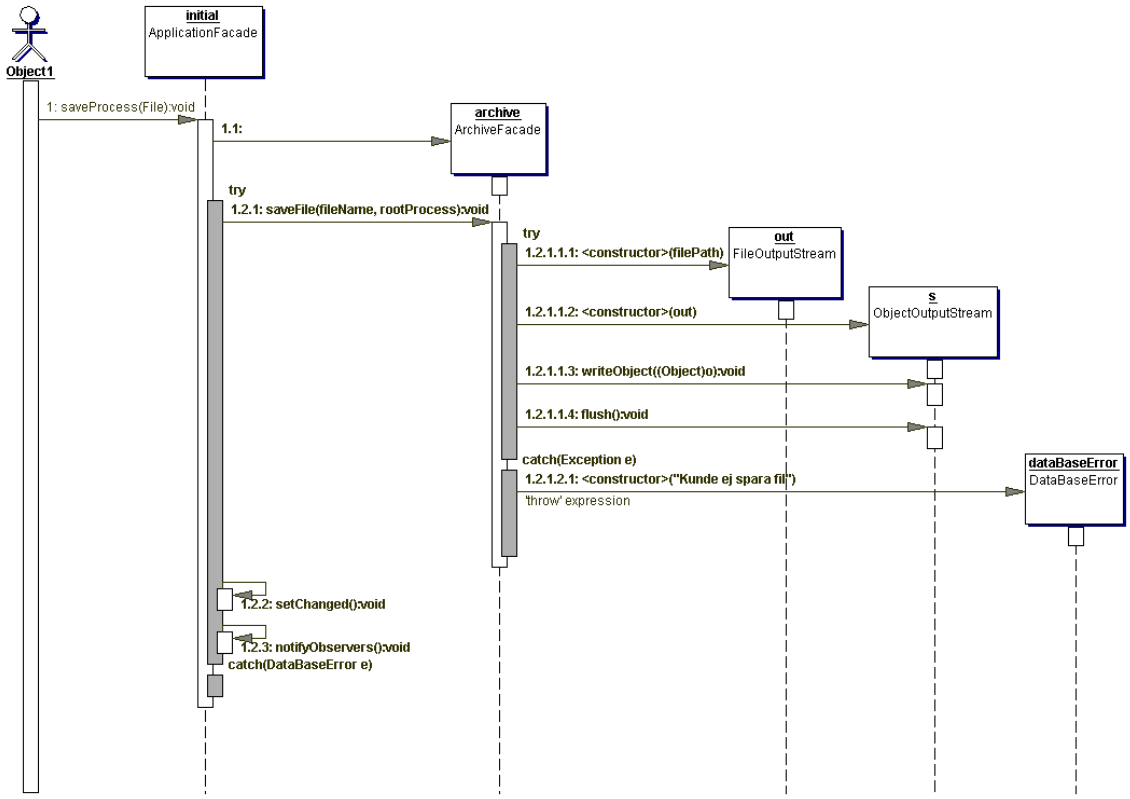
7. Välj processens utgångsvillkor



8. Öppna fil


Object1
initial
ApplicationFacade
1: openProcess(File):void
1.1:
archive
ArchiveFacade
try
1.2.1: openFile(fileName):Object
try
1.2.1.1.1: <constructor>(filePath)
in
FileInputStream
1.2.1.1.2: <constructor>(in)
s
ObjectInputStream
1.2.1.1.3: readObject():java.lang.Object
catch(Exception e)
1.2.1.2.1: <constructor>("Kunde ej öppna ritning från disk")
'throw' expression
dataBaseError
DataBaseError
1.2.2: setChanged():void
1.2.3: notifyObservers():void
catch(DataBaseError e)

9. Spara fil


Object1
initial
ApplicationFacade
1: saveProcess(File):void
1.1:
archive
ArchiveFacade
try
1.2.1: saveFile(fileName, rootProcess):void
try
1.2.1.1.1: <constructor>(filePath)
out
FileOutputStream
1.2.1.1.2: <constructor>(out)
s
ObjectOutputStream
1.2.1.1.3: writeObject((Object)o):void
1.2.1.1.4: flush():void
catch(Exception e)
1.2.1.2.1: <constructor>("Kunde ej spara fil")
dataBaseError
DataBaseError
'throw' expression
1.2.2: setChanged():void
1.2.3: notifyObservers():void
catch(DataBaseError e)