A brief exploration of the XP planning process “The planning game”

Master Thesis in Computing Science and Engineering, 20p

Författare:
Robert Jonsson
c98rjn@cs.umu.se

Handledare:
Vitec Fastighetssystem AB*:*
Lars Stenlund
lars.stenlund@vitec.se

Umeå Universitet:
Jürgen Börstler
jubo@cs.umu.se

Abstract

All activities performed in conjunction between individuals or organisations demand strategies for how the work is to be carried out. The software industry is no exception and up till now several methodologies and strategies have been developed, among them *Extreme Programming – XP*. XP is a lightweight methodology for software developing; it addresses teamwork, communication and feedback. The methodology provides techniques and work procedures that makes software development a less risky business.

This paper is a theoretical study of the XP planning process and explains how XP planning is performed utilizing *The Planning Game*. The study and a short software development project, together represent a level D thesis in Computing Science and Engineering.

The study tries to give answers to the following problems:

- What is the main purpose of *The Planning Game*?
- How does the game apply?
- Which events do you plan with the game?
- Who are the game players?
- Of what size should a *user story* be?
- Who writes the *user stories*?
- How do you use story cards? Are there different types of cards?
- What is the main difference between light- and heavy-weight methodologies?
- Who should use the planning game?
- What are the advantages and disadvantages?

The purpose of the software project is to exemplify the XP planning process and provide some practical experience.

Sammanfattning

Verksamhet som utförs i samförstånd mellan flera individer eller organisationer kräver strategier för hur arbetet skall fortlöpa. Mjukvaruindustrin är inget undantag där ett stort antal metoder och strategier utvecklats, däribland *Extreme Programmering - XP*. XP är en lättviktsmetodik för mjukvaruutveckling som baseras på enkelhet, feedback och kommunikation. Metoden tillhandahåller arbetssätt och tekniker för mjukvaruutveckling som minskar osäkerhetsfaktorer och därmed risken att misslyckas med sitt projekt.

Detta arbete är en teoretisk studie av planeringsprocessen i XP, ”*The Planning Game*”. Studien och ett praktiskt exempel utgör tillsammans ett examensarbete på D – nivå. Arbetet har genomförts som en artikel- och litteraturstudie där bl.a. följande frågeställningar undersöks:

- Syftet med *The planning game*?
- Hur tillämpas spelet?
- Vad planeras med planeringsspelet?
- Vilka är deltagarna i spelet?
- Hur lång/kort bör en berättelse vara?
- Vem skriver berättelserna?
- Hur används s*tory cards?* Finns det olika typer av kort?
- Hur skiljer sig tungvikts- och lättviktsmetodiker åt?
- Vilka bör/bör inte använda spelet?
- Fördelar/Nackdelar

Det praktiska exemplet är ett mindre utvecklingsprojekt där teorin bakom XP och planeringsspelet tillämpas.

Innehållsförteckning

1. Inledning

Utveckling av mjukvaror för affärsändamål har blivit en mycket viktig del av informationsteknologin under de senaste årtiondena. Men att skapa bra programvaror är långt ifrån en problemfri företeelse. Kunden kanske säger en sak menar en annan och vill en tredje, att identifiera vad denne egentligen vill ha är en av svårigheterna. Det kan dessutom vara rätt svårt att beskriva stora system vilket ytterligare försvårar [10]. Men om vi fick önska så skulle alla system och programvaror som skapas inneha följande egenskaper:

• Nyttigt och användbart	•*Utför det man önskar*
• Pålitligt	•*Inte fallerar*
• Flexibelt	•*Anpassnings och förändringsbart*
• Överkomligt	•*Ej för kostsamt*
• Tillgängligt	•*Lätt att använda*
• Enkelt att underhålla	•*Enkelt att anpassa till nya behov eller krav*

Men hur kan vi uppnå dessa egenskaper?

För att uppnå målen krävs bl.a. insikt i kundens behov och rådande förhållanden men också metoder, teknologier och system för själva utvecklingsarbetet. En del företag utvecklar sina egna metoder och processer medan andra använder befintliga såsom RUP [11].
Egenutvecklade metoder och processer består ofta av regler som tillkommit för att undvika misstag från tidigare projekt. Tyvärr händer det att omfattningen av dessa i rädsla för att göra samma fel växer och gör metoden ohanterlig. Då riskerar själva processen att ge upphov till de problem den egentligen var till för att lösa [1].
RUP är en metod för mjukvaruutveckling som kräver omfattande dokumentarbete vilket gör kravhantering och planering mer tungarbetad och komplex. *Extreme Programming - XP* är en lättviktig metodik som bl.a. avser minska överflödig dokumenthantering vid planering. Därför används enkla beprövade metoder i XP som exempelvis förespråkar muntlig kommunikation över skriftlig, vilket är en av anledningarna att kunden ingår i projektet som en deltagare. Kunden kan styra utvecklingen på nära håll och kommunikationen kund och utvecklare emellan stärks [2].

1.1. Arbetet, Syfte och Mål

Huvudsyftet med examensarbetet är att utreda hur planeringsprocessen i ett projekt som använder Extreme Programmering fungerar och vad som skiljer den mot traditionell planering. Målet med arbetet är att bilda en uppfattning om styrkor och svagheter i metoden, samt få erfarenhet av projektplanering med XP och planeringsspelet.

1.2. Material och Metod

Teoriarbetet har genomförts som en artikel/litteratur studie där materialet hämtats från böcker, webbsidor och artiklar. Litteraturen i bokform är i huvudsak hämtad ur bokserien *The XP Serie*s [20].

2. Problembeskrivning

2.1 Specifikation av uppgiften

Vitec Fastighetssystem AB jobbar med utveckling av programvaror och system riktat mot fastighets- och energibranschen [21]. Företaget levererar främst IT-lösningar anpassade för Windows och webbmiljö. Idag är utvecklingen helt inriktad mot plattformen Microsoft .NET vilket medför att Microsoft Visual Studio .NET är huvudverktyget. På företaget tillämpas metodiken *Extreme Programming* i utvecklingsarbetet och man följer huvuddelen av stegen i processen. Planering av projekt sker idag enligt gamla principer, men företaget kommer inom en snar framtid att införa planeringsspelet som planeringsmetod.

Teoridel
I XP är ett viktigt moment planeringen, dels för att bilda sig en uppfattning om vad kunden vill ha dels för att kunna uppskatta tidsåtgången för projektet. Teoridelen av examensarbetet utgörs av en studie i vilken planeringsstrategin i XP, *The Planning Game* granskas. Följande delar kommer att beaktas:

- Vem/vilka skall använda planeringsspelet
- Hur tillämpas det praktiskt
- *User stories* och s*tory cards*
- Skillnaden *use case* och *user story*
- Typer av *story cards.*
- Alternativa planeringsmetoder
- För- och nackdelar med planeringsspelet
- Hur planeras projekt på Vitec AB
- Hur passar planeringsspelet in i deras verksamhet

Praktik del
Den praktiska delen av arbetet är ett mindre utvecklingsprojekt där ett verktyg för datafångst, rapport och analys skapas. Planeringen av projektet sker med hjälp av planeringsspelet för att ge erfarenhet av planeringsmetoden. Projektets deltagare består av en kund och en utvecklare, extern handledare agerar kund enligt XP, *On-site Customer* [22]. Programmets funktion beskrivs i korta berättelser med *story cards*.

Utvecklingsarbetet inriktas mot plattformen Microsoft .NET [23] och språket Visual Basic .NET. Applikationen som skapas arbetar som en *Rich Client* anpassat för *No-Touch deploy* [24]. Vilket innebär att distribution, uppdatering och exekvering kan utföras i webbmiljö.

2.2 Avgränsning

Omfattningen av det teoretiska arbetet har begränsats till att innefatta planeringsprocessen i XP, *The Planning Game.*

Utvecklingsarbetet har avgränsats och funktionaliteten har utvecklats i nedanstående ordning.

- Applikation som *Rich Client* i webbmiljö.
- Funktionellt och anpassningsbart gränssnitt (*Syncfusion*).
- Automatisk hantering av persondata, datum och tid. (Inloggad person, kategori, mm.)
- Kommunikation med databas, SQL server och vyer.
- Inmatning och lagring av ny data i databas.
- Redigering av befintlig lagrad data.
- Begreppshantering
- Tidtagningsfunktion.
- Spårbarhet
- Rapportgenerering
- Offlinefunktion.
- Webbservice för översättning av XML –data och lagring i databas.

2.3. Informationskällor

I utvecklingsarbetet har information och hjälp i första hand hämtats från artiklar ur MSDN – *Microsoft System Developer Network*, samt utvecklare på Vitec AB.

3. Metoder, verktyg och förutsättningar i praktikdelen

3.1. Plattform för utvecklingsarbetet

Programmet är skrivet för operativsystemet Windows XP/2000 och utvecklingsarbetet har inriktats mot Microsofts plattform .NET. För att applikationen ska fungera krävs därför att ramverket .NET finns installerat på värddatorn som exekverar programmet.

Applikationen har anpassats för en teknologi i .NET som kallas *No-Touch Deploy*[1]. Tekniken innebär att distribution, uppdatering och exekvering av program kan skötas från en webbserver. Applikation startas via en hänvisning till en webbserver varifrån programmet automatiskt hämtas hem till klienten där det exekveras lokalt. På så vis fungerar applikationen som en r*ich client* vilket innebär att Windowsspecifika funktioner exekveras lokalt vilket reducerar belastningen på serversidan. Motsatt förhållande gäller för *thin-clients* där all logik finns på serversidan.

3.2. Utvecklingsmiljö, Visual Studio .NET

Programmet är skapat i Microsofts utvecklingsmiljö Visual Studio .NET® och källkoden är skriven i Visual Basic .NET.

3.3. Databashantering, SQL-Server 2000

Systemet är uppbyggt kring en databas (se bild 1) som finns tillgänglig via en SQL-server. Inloggning till servern och databasen sker automatiskt genom auktorisation via användarens Windowskonto.
Den information som registreras i programmet lagras i databasen på servern. Kontakten med databasen sköts med hjälp av klassen `SQLhelper` som automatiskt hanterar öppning och stängning av databasen.

[1] *Kallas ibland ”Zero deploy in the .NET Framework”.*

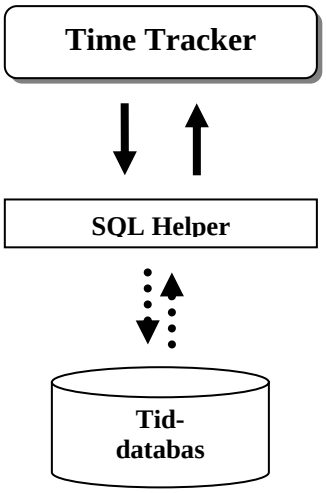


Bild 1, Datalagring

3.3.1 Vyer, Lagrade procedurer och SQL-frågor

I databasen finns möjlighet att skapa och använda lagrade procedurer och vyer [25] för dataåtkomst. Dessa förenklar åtkomst, uppdatering och tillägg av data i tabeller och ökar generellt sett prestandan i arbetet mot databasen. När utvecklingsarbetet startade var tanken att allt informationsutbyte med databasen skulle ske med lagrade procedurer. I applikationen används istället vyer och SQL-frågor och bara ett fåtal lagrade procedurer.

3.4. Enhetstestning med verktyget NUnit 2.0

I XP bygger utvecklingen på att man kontinuerligt kan kontrollera funktionaliteten hos den kod man skriver. Denna teknik kallas enhetstestning [26] och är ett av momenten i XP. Testerna används för att kontrollera att den kod som skapas verkligen löser rätt problem. Testerna skapas kontinuerligt före, under och efter inplementation av *user stories*.
Enhetstestning ger utvecklaren en bra bild av problemet som skall lösas och delar upp det i mindre delar som kan testas individuellt. Testerna fungerar som en typ av skyddsnät när flera utvecklare arbetar med samma kod s.k. ”kollektivt ägande av kod” [27]. Då utgör de en kontroll att de ändringar som införs inte påverkar eller förändrar tidigare funktionalitet. Därigenom stärks förtroendet för koden som utvecklats och deltagarna ”vågar” ändra i kod som någon annan skrivit.

Verktyget NUnit [28] är ett ramverk som används för att testa kod utvecklad i .NET miljö. Idén med verktyget är att testerna som programmet skall klara skapas i förväg och integreras med programkoden som skall klara testerna.

I praktikdelen av examensarbetet var tanken att nyttja verktyget NUnit genom hela utvecklingsarbetet för att skapa enhetstester. Det visade sig kräva mer resurser än vad som var tänkt, varför det sedermera ströks.

3.5. Projektets planeringsstrategi, *The planning game*

Arbetet med applikationen för datafångst och analys har planerats utifrån planeringsprocessen i XP. Med planeringsspelet har en iterativ utvecklingsplan skapats, releaseplaner, iterationsplaner, värdering och prioritering har utarbetats i samråd med kunden. Kundens krav på systemet har formulerats i korta berättelser som anger vad systemet skall utföra och hur det ska se ut. Prioritering av berättelser har skett ur aspekten att ”värdefullast funktionalitet” skall implementeras först (sett ur kundens perspektiv).
Kompletterande fakta om vad systemet skall uträtta har beskrivits i en, *system methafor* [29] som skapats genom att kunden med stöd av sina berättelser förklarat målbilden. Tvetydigheter har kontinuerligt korrigerats genom ständig kontakt med kunden.

4. *The Planning Game*, planeringsprocessen i Extremprogrammering

4.1 Introduktion – Extremprogrammering [XP]

Extremprogrammering är en hårt disciplinerad metodik för mjukvaruutveckling där enkelhet, feedback, teamwork, kommunikation och problemlösning betonas [10]. Metodiken består av tolv processer där några är obligatoriska och måste ingå i projekt som tillämpar XP. Det är också en av anledningarna att många ser XP som en disciplin inom mjukvaruutveckling. Målet med metodiken är att minimera vanligaste riskerna inom mjukvaruutveckling såsom missade tidsramar, buggiga system och dålig funktionalitet [3].
Arbetssättet i XP gör att deltagarna får bättre kontroll på att utvecklingen sker i enlighet med kundens krav och att det är rätt problem som systemet löser. XP är alltså en lättviktig metodik som genomsyras av kommunikation människor emellan istället för dokumentform.

XP består av följande tolv processer:

- The Planning Game — *Planering med hjälp av ett rollspel.*
- Små releaser: — *Många men korta releaser, varje release levererar ett körbart system med ny funktionalitet.*
- System Metaphor: — *En gemensam och övergripande berättelse som beskriver systemets funktion.*
- Enkel design: — *Enklast möjliga designen i alla lägen.*
- Kontinuerlig testing: — *Tester av systemet med hjälp av "Unit testing".*
- Refactoring: — *Snygga till koden, ta bort duplicerade metoder mm.*
- Parprogrammering: — *All programmering sker två och två, vid en dator.*
- Kollektivt ägande — *Alla i teamet har rättigheter att ändra överallt i koden.*

- Kontinuerlig integration: *Nya delar av systemet integreras med övriga delar minst en gång per dag.*
- 40-timmars-veckor: *All övertid minimeras.*
- On-site Customer: *En representant för kunden eller kundens intressen finns med i projektet hela tiden.*
- Standardiserad kod: *Regler för hur koden skall se ut.*

Metodiken grundades 1996 av Kent Beck, som sedan början av 90-talet funderat på nya metoder för mjukvaruutveckling. Vid ett utvecklingsprojekt hos Daimler Chrysler bestämde sig Kent för att införa ett antal nya koncept i utvecklingsarbetet, bl.a. par-programmering. Det visade sig fungera bra och blev startskottet för utvecklingen av lättviktsmetodiken extrem programmering.
Metoden är anpassad för att fungera i utvecklingsteam av storleken 2 till 10 personer. Denna gräns behöver inte ses som absolut utan är en rekommendation.

Enligt *Kent Beck* är idén med XP att företag i mjukvaruindustrin ska öka sin produktivitet, bli mer flexibla och på ett bättre sätt hantera otydliga krav och snabba förändringar [3]

4.1.1 Lätt- vs. tungviktsmetodik

De flesta metoder och modeller för programvarukonstruktion liknar varandra i något avseende eftersom de hanterar dokumentation och specifikation av utvecklingsarbete. Den främsta skillnaden mellan tung- och lättviktsmetodiker ligger i hur dokumentationsprocessen behandlas. De lättviktiga processerna förespråkar kommunikation människor emellan och anser att koden som utvecklas är projektets dokumentation. De tyngre processerna är mycket välstrukturerade och upprätthåller dokument i pappersform som noggrant beskriver vad som skall och har utvecklats. Detta leder till mer arbete vid sidan om själva utvecklingsarbetet, vilket är en nackdel. Fördelen är att metoderna är strikta och ger ett mer förutsägbart resultat. De lättviktiga däremot har en större grad av flexibilitet, vilket leder till att de på ett bättre sätt kan hantera oförutsedda förändringar.
En utförlig analys av skillnaderna mellan metodikerna finns i Jason P. Charvas artikel ”*Heavyweight vs. lightweight methodologies*” [30]

4.1.2 Andra lättviktsmetodiker

De metoder och tillvägagångssätt som återfinns i XP är inga nyheter utan har funnits lika länge som programmering och de flesta programspråken. Därför finns flera andra metoder som fungerar på liknande sätt. Exempel på andra metoder som också klassas som lättviktsmetodiker är:

- ASD — *Adaptive Software Development (Jim Highsmith)*
- CRYSTAL LIGHT — *Alistair Cockburns metodmatris där man själv väljer ut en metodvariant som passar projektet.*
- DSDM — *Dynamic Systems Development Method (www.dsdm.org). Påminner mycket om XP, men saknar XPs tekniska inslag*
- FDD — *Feature Driven Design (Peter Coad)*
- OSS — *Open Source Software*
- SCRUM — *Varje iteration är en rusning med kaos i hälarna. Mellan iterationerna återfinner man en viss ordning (www.controlchaos.com).*

4.2 Planeringsprocessen

Att planera projekt och sedan följa planen är en svår och krävande utmaning som ställer höga krav på teamet och projektledaren. Inom mjukvaruindustrin har det länge funnits olika metoder och tankesätt för att underlätta detta arbete, problemet är att många av dem ofta ger upphov till ökad belastning i form av dokumentation. Ett annat problem är att metoderna har svårt att klara snabba förändringar av förutsättningar och krav. I XP planeras arbetet i korta iterationer, (ca 1-3 veckor) vilket gör att projektet snabbt kan anpassas till rådande omständigheter. Med XP kan alltså kunden förändra, göra tillägg och ta bort krav och funktionalitet kontinuerligt [4], vilket är en av anledningarna till att intresset för XP ökat kraftigt och fler företag vill använda metoden i sin verksamhet.

4.2.1 Översikt av planeringsprocessen i XP

1. Övergripande plan, *Exploring the big plan*
    - Kund beskriver systemet.
    - Diskussionen dokumenteras med *user stories.*
    - Utvecklare analyserar och värderar *user stories.*
    - Är projektet genomförbart?
    - Hur lång tid kommer projektet att behöva?
    - Har vi resurser att gå vidare?
    - Bestäm längden (budget) för en release och en iteration.

2. Releaseplanering, *Planning Game*
    - Kunden sorterar *user stories* i prioritetsordning (tre högar).
    - Bestäm utvecklingsteamets kapacitet.
    - Vad är viktigast att göra först?
    - Kunden väljer *user stories* till ett värde av releasens budget, dvs. antalet *story points* som beräknats ur releasens längd.

3. Iterationsplanering, *Planning Game*
    - Kunden väljer *user stories* till iterationen ur releasens budget.
    - Utvecklarna bryter ned varje *user story* i *tasks.*
    - Utvecklarna åtar sig ansvaret för de tasks som ingår i iterationen.
    - Varje utvecklare uppskattar tidsåtgången för de *tasks* de är ansvariga för.
    - Iteration planeras tillsammans med kund.

4.2.2 Varför behövs planeringen?

Planeringen av ett mjukvaruprojekt i XP kan beskrivas som ett sätt att bilda sig en uppfattning om hur samarbetet med en kund kommer att bli. Planeringen behövs för att lära sig vad kunden efterfrågar och för att avgöra hur lång tid det kommer att ta. Den används också för att dela upp, prioritera och avgränsa utvecklingen, eftersom det är viktigt att arbetet hela tiden inriktas på de viktigaste uppgifterna. Med hjälp av planeringen skapas hållpunkter och milstolpar som används för att stämma av projektet. De fungerar som ett mått på hur mycket jobb som är avklarat och ger en fingervisning om i vilken riktning projektet fortlöper. Om projektet vid något tillfälle ligger utanför planeringen kan projektet snabbt korrigeras vid dessa hållpunkter innan projektet fallerar. I XP får teamet och kunden snabbt feedback om hur projektet ligger till eftersom arbetet sker i korta iterationer. Vid slutet av varje iteration presenteras systemet för kunden varvid han/hon bedömer arbetet i förhållande till planeringen. [4].

Planeringen behövs också för att hantera situationer där människor ställs inför främmande prövningar och osäkerhet. Första reaktionen i dessa fall är ofta närmast är panikartad och kan medföra irrationella handlingar. När man planerar något försöker man förutsäga vad som skall hända och hur man ska agera, för att på bästa sätt klara av situationen. Om man i det verkliga livet vet vad man bör göra i oväntade lägen, dvs. har en plan för hur man ska agera, drabbas man inte lika lätt av panik. På samma sätt fungerar det i ett mjukvaru-utvecklingsprojekt som präglas av ständiga förändringar och störningar. Det gäller därför att redan från början ha en god strategi för oförutsedda händelser och på så vis skapa en trygg tillvaro.
En detalj att reflektera över är att den planering man bygger bara kommer att bli så bra som de uppskattningar den bygger på.

4.3 *User Stories,* Kravdokumentation i XP

Kravhantering handlar om att fastställa kundens önskemål och förväntningar på systemet som skall byggas. Problemet är att det är svårt att beskriva stora system och identifiera vad kunden egentligen vill ha. I traditionell mjukvaruutveckling utförs en kravanalys som fastställer kundens krav i ett kravdokument. Dokumentet omarbetas till dess att båda parter är överens om vad som skall utvecklas. Nackdelen med metoden är att kravhantering blir rätt krånglig och tar lång tid [2].

Planeringsprocessen i XP använder *user stories* för kravhantering. *User stories* är berättelser som med några få rader och på ett enkelt sätt beskriver vad systemet ska uträtta och kan närmast beskrivas som en lättviktig version av traditionella *use cases*. Kunden antecknar några rader beskrivande text i form av en *user story* på ett *story card* som sedan används i planeringsspelet som spelkort. Den funktion som beskrivs måste vara utformad så att den går att testa och värdera, varje berättelse tilldelas ett värde i form av ett antal *story points* [6].
Formatet på korten hjälper till att hålla nere längden av en berättelse till ett par meningar, vilket anses vara en fördel eftersom korta berättelser oftast ger mindre utrymme för onödiga utsvävningar. En annan fördel med kortsystemet är att det är lätt att arbeta med under planeringsfasen, korten kan ligga på ett bord eller hängas upp på en vägg eller bara lagras i en hög.
User stories används alltså för att dela upp kundens önskemål och krav på systemets funktionalitet i mindre delar som är lättare att förstå, värdera och testa. Till skillnad från ett kravdokument innehåller spelkorten få detaljer eftersom *user stories* representerar ett koncept och inte en detaljerad specifikation. Detaljer är viktiga men behövs först vid implementationen och då finns kunden med i projektet och kan svara på detaljfrågor. Det viktiga med *user stories* är att de är enkla och att alla kan förstå dem eftersom texten i någon mening ska återskapa diskussionen som förekom skapandet av berättelsen [6].

4.3.1 Bra och dåliga berättelser

Att skriva berättelser ställer vissa krav på författaren eftersom en berättelse skall skrivas i så enkla termer som möjligt och med ett naturligt språk.

Bra *user stories* utformas enligt nedan:

- Det som beskrivs genererar ett värde för kunden.
- Endast ett fåtal meningar, kort och kärnfullt.
- Testbar utifrån kundens *acceptance tests.*
- Oberoende av andra *user stories*.

Att skriva *user stories* är en iterativ process som kräver mycket feedback för att bli bra.

4.3.2 User stories & story cards, exempel

För att visa principen med berättelser och spelkort beskrivs nedan ett exempel hämtat ur boken *Planning Extreme Programming s.53* [4]. Exemplet kommer från utvecklingen av ett bokningssystem för resor och berättelserna beskriver funktioner som skall finnas i systemet.

1. Hitta lägsta taxan.
Presentera de tio lägsta priserna för en resa mellan A och B.

2. Visa tillgängliga flyg.
Visa vilka flyg som finns för en resa mellan A och B.

3. Sortera bästa flygen.
Sortera flygresorna efter: Flygtid, antal byten, landnings-tid, närmaste flygplats för en resa mellan A och B.

4. Köp av biljett.
Köp av biljett med kreditkort. Kontrollera kreditkortets giltighet.

5. Skapa kundprofil
Lagra kundens personuppgifter och kreditkortsnummer för att snabb åtkomst.

6. Visa kundens resplan.
Presentera alla kundens resplaner som finns lagrade.

7. Avboka resplan.
Avboka en resplan. Avboka tillhörande hotell, taxi, mm.

Figur 3 *User stories* exempel

4.3.3 Story points

Story points är i någon mening ett mått för hur lång tid en *user story* kräver för att färdigställas. Vad som menas med en poäng kan inledningsvis vara rätt diffust när det inte finns något att jämföra med, begreppet är dessutom individuellt och bestäms av utvecklingsteamet. En gemensam nämnare är i alla fall att en berättelse som värderas till fyra poäng kommer att behöva dubbelt så lång tid som en berättelse av värdet två poäng.

4.3.4 Idealveckor, idealdagar och *perfect programming hour*

Dessa begrepp används i planeringen när man vill beskriva hur lång tid en programmerare behöver för att slutföra implementationen av en berättelse. Med ”ideal” menas att programmeraren utan avbrott och störande moment kan använda 100 % av sin tid. Idealveckor och idealdagar överensstämmer således sällan inte med motsvarande kalendertid. När en programmerare säger att han behöver en vecka för att slutföra något, tar det förmodligen det dubbla.

Idealveckor – *Tid uttryckt i antal veckor där programmeraren eller teamet lägger ned 100 % av sin tid på en och samma uppgift.*

Idealdagar – *Tid uttryckt i antal dagar där programmeraren eller teamet lägger ned 100 % av sin tid på en och samma uppgift.*

Perfect programming hour -
En timme där programmeraren eller teamet kan fokusera 100 % av sin tid på en och samma uppgift under en timme.

4.3.5 *Velocity,* Utvecklingsteamets kapacitet

Velocity är ett mått på hur många *story points* ett team hinner implementera under en iteration, dvs. hur mycket arbete som teamet kan uträtta under en viss tid. Första gången en release planeras brukar intuition och gissningar användas för att uppskatta kapaciteten, en vanlig gissning brukar ligga vid ca 1/3 poäng per person under en iteration. Detta är naturligtvis en grov gissning som kan slå väldigt fel men måttet kommer med tiden att stabilisera sig och bli mer korrekt.

4.3.6 Att uppskatta utvecklingstid för berättelser, *story points*.

Ett av de svåraste momenten i planeringsspelet är att uppskatta utvecklingstiden för en berättelse. För att bli bra på det krävs övning och erfarenhet, därför är det viktigt att hela utvecklingsteamet är med i diskussionen och gemensamt skattar tiden för varje enskild berättelse. I boken *Planning Extreme Programming* [4] beskrivs olika sätt för att uppskatta utvecklingstid, bl.a. principen ”gårdagens väder” och jämförelse med tidigare liknande berättelser. Den förstnämnda används vid iterationsplanering och använder antalet poäng som implementerats i föregående iteration som ett värde för kommande iteration. För att teamet ska bli bättre på uppskattningar är det viktigt att den verkliga utvecklingstiden noggrant dokumenteras eftersom teorin då kan avstämmas mot verkligheten varvid betydelsen av poäng kan korrigeras. Första iterationen då det inte finns någon gammal information att jämföra med får teamet och kunden använda intuition och erfarenhet för att bedöma omfattningen av en iteration [7].

4.3.7 Förändra planeringen i efterhand.

En av styrkorna med planeringen i XP-projekt är att nya och förändrade krav kontinuerligt kan hanteras. Nya *user stories* och gamla som behöver modifieras kan enkelt bearbetas och plockas in i kommande iterationer utan att det påverkar det nuvarande arbetet nämnvärt. Motsvarande tillägg eller förändring i ett projekt som planerats på traditionellt sätt skulle sannolikt ha krävt omfattande planeringsinsatser med förseningar som följd.

4.4 Planering av extremeprogrameringsprojekt

4.4.1 Planeringsfaserna

I XP utförs planering på tre plan, övergripande, release och iteration.

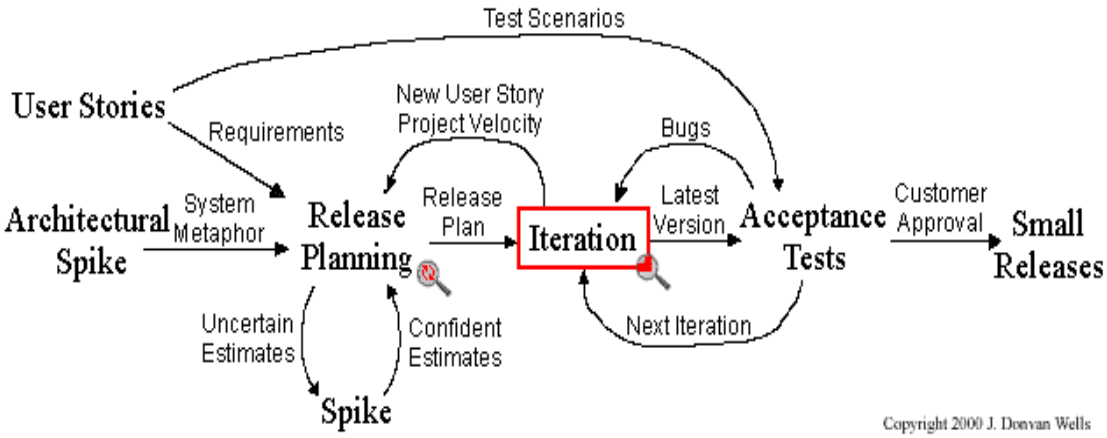


Bild 1. Planeringsfaserna i ett XP projekt. [2]

Övergripande, *Exploring the big plan*

Den inledande delen av planeringen handlar om att förstå vad kunden vill ha och om det finns resurser att genomföra. Tillsammans med kunden skapas korta övergripande berättelser, *user stories* som i icke tekniska termer beskriver systemets funktionalitet. Därefter uppskattas utvecklingstiden mot bakgrund av tillgängliga resurser. Utifrån detta skapas en grov planering, *big plan* för det tilltänkta systemet. Planen innehåller än så länge bara korta berättelser som beskriver systemet, men dessa ligger till grund för ett beslut om projektet är genomförbart eller ej [5].

Releaseplan

När ett beslut att gå vidare med projektet fattats kommer steget att utveckla en releaseplan. Syftet med releaseplaneringen är att dela in utvecklingen av systemets funktionalitet i ett antal releaser som var och en levererar ett körbart system. En release i XP omfattar normalt arbete för ca 3 månader, men omfattningen bestäms slutgiltigt av kunden. Kunden värderar *user stories* utifrån prioritet och sätter ett slutdatum för releasen. Den här delen av planeringen i XP kan liknas vid en synkronisering av projektet mot affärsmässigt viktiga mål, exempelvis förutbestämda datum då systemet skall demonstreras.

[2] XP projektets faser, (2003-02-12), http:// *www.extremeprogramming.org/abcd/planning.htm*

Iterationsplan
Vid releaseplaneringen bestämmer kunden systemets funktionalitet som skall vara klart på releasens slutdatum. Arbetet med att färdigställa en release delas upp i iterationer där varje iteration levererar ett fungerande delsystem. Iterationslängden bestäms av kunden men bruka vara mellan en och tre veckor, längre iterationstid medför ett färre antal iterationer och vice versa. I slutet av iterationerna sker en avstämning mot den ursprungliga planeringen. Kunden kan då ändra sina krav genom att lägga till och ta bort *user stories* eller modifiera befintliga, samtidigt kan teamet korrigera eventuella problem som uppstått. Valet av iterationslängd baseras på hur ofta kunden anser de vara nödvändigt att stämma av mot ursprungliga planen. Av detta förefaller korta iterationer vara att föredra vilket också rekommenderas av XP [3].

Varje ny iteration inleds med att kunden och utvecklarna samlas för att bryta ner de ingående berättelserna i ännu mindre delar *tasks*, varje *task* har en längd av ca en till tre idealdagar (se 4.3.3). Eftersom det är viktigt att alla i teamet förstår innebörden av berättelserna som ingår i iterationen, läser kunden lupp berättelserna, att höra dem ur kundens perspektiv ger en bättre bild av vad han/hon vill ha. Utvecklarna har dessutom möjlighet att ställa frågor direkt till kunden om något är oklart. I slutet av en iteration testas systemet med kundens tester, *acceptance tests,* kod som inte går att testa existerar definitionsmässigt inte i XP.

4.4.2 Kostnad, kvalitet, tid och omfattning

Med planeringen kontrolleras fyra variabler som måste koordineras för att ge bästa möjliga effekt.

Kostnad
– Mer pengar löser vissa problem fast för mycket kan skapa nya problem.
Tid
– Mer tid gör att vi kan leverera mer. Men för mycket tid kan vara till skada.
Kvalitet
– Genom att tumma på kvaliteten kan man göra vissa vinster på kort sikt. Men på längre sikt kostar det för mycket.
Omfattning, *scope*
– Mindre omfattning gör det möjligt att få bättre kvalitet samt leverera tidigare och billigare

4.4.3 Kundens roll i projektet

I jämförelse med andra metoder för projektplanering och utveckling skiljer sig XP i sin syn på kunden. I ”vanliga” fall är kunden den enhet som levererar en lista med krav på funktionalitet och ett datum när det hela skall vara klart. XP anser att kundens närvaro i ett projekt är av högsta vikt för att uppnå ett bra resultat och är alltså alldeles för viktig för att helt utelämnas ur projektet. Arbetssättet som eftersträvas är kund och utvecklare i nära samarbete eftersom det medför att de blir medvetna om varandras svårigheter och problem. En annan poäng med att ha kunden med i projektet från början till slut är att dokumentationen inte behöver vara lika detaljerad. I händelse av tvetydigheter finns kunden på plats och kan förklara, vilket också medför att planeringen tar mindre tid.

4.5 Planeringsspelet

En process som närmast kan beskrivas som en typ av rollspel där syftet är att skapa en givande kommunikation mellan utvecklare och kund för att på så sätt nå målet att leverera mjukvara motsvarande kundens förväntningar. Processen kallas *The Planning Game* och hjälper till att fokusera utvecklingsarbetet på det mest väsentligt för kunden och används både för release- och iterationsplanering. Även om spelet används på olika sätt vid planering av olika delar av projektet är alltid grundprincipen densamma.

4.5.1 Enkelhet och kommunikation grundläggande i planeringsprocessen

När man väljer att spela ett nytt sällskapsspel vill man att reglerna skall vara så enkla som möjligt eftersom deltagarna snabbt ska förstå spelet och komma igång. Planeringsprocessen i XP bygger på öppen kommunikation och kan liknas vid ett rollspel som är enkelt och kraftfullt. Deltagarna använder *user stories* för att illustrera systemet som skall byggas, det väsentliga i spelet är att få igång en diskussion bland deltagarna som leder till en förståelse för vad kunden vill ha.

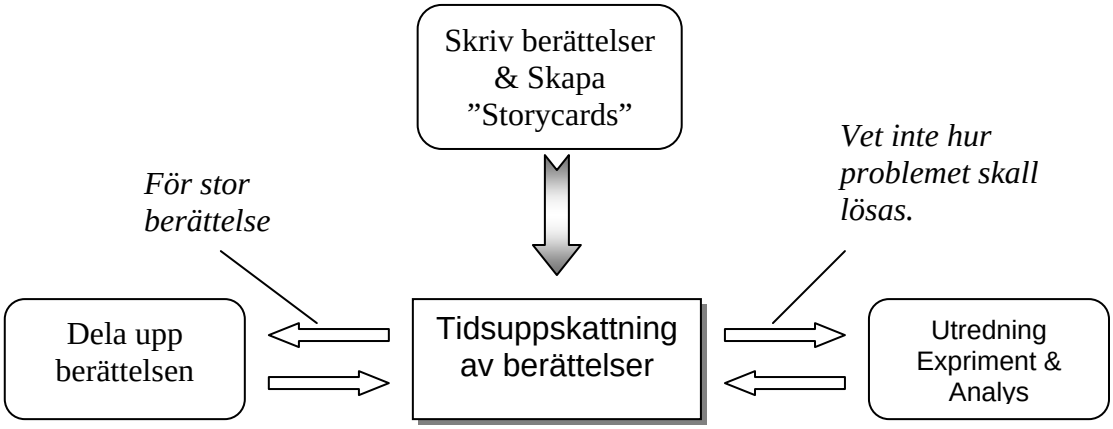
4.5.2 Deltagarna i *The Planning Game*

I spelet finns två sidor representerade, utveckling och kund. Med kund avses en representant för beställaren eller dennes intressen, dvs. den som är upphovet till projektet. I XP används begreppet *On-site customer* vilket innebär att en kund fysiskt deltar i projektet under utvecklingsarbetet från start till slut.
I praktiken kan det vara svårt för en kund att ställa upp och delta till 100 %. Alternativet då är att låta en representant för kundens intressen delta i stället vilket ställer extra höga krav på kommunikation och samarbetet dem emellan, men det är det näst bästa alternativet.

4.5.3 Releaseplanering

Undersökningsfasen.
I den första delen av releaseplaneringen är uppgiften att med hjälp av *planning game* skapa och värdera *user stories* för systemet. Kunden beskriver funktioner och egenskaper med några meningar och dokumenterar på *story cards*. Utvecklarnas uppgift är att tillsammans bedöma och avgöra antal *story points* för utveckling av varje *user story*. Om det är svårt att värdera pga. ny teknik eller annat högriskmoment utförs en *spike,* utvecklarna gör en kort utredning om den nya tekniken genom experiment och analys (se figur 1).

I XP används ”gårdagens väder” principen för att göra en bedömning av utvecklingstider för *user stories*. Det innebär att en liknande berättelse som implementerats tidigare användes som referens i bedömningen. Metoden ger ett bra riktvärde som någorlunda överensstämmer med verkligheten eftersom det är sannolikt att liknande arbete kommer att behöver samma utvecklingstid.
Värderingen noteras på kortet och används sedan i releaseplanen.



Figur 1. Undersökningsfasen

Tilldelningsfasen
I denna fas bestäms arbetet för en release dvs. arbete för ca 1-3 månader framåt. En release är en version av systemet som inte har all funktionalitet implementerad, men är tillräckligt utvecklad för att levereras till användare utanför projektet.

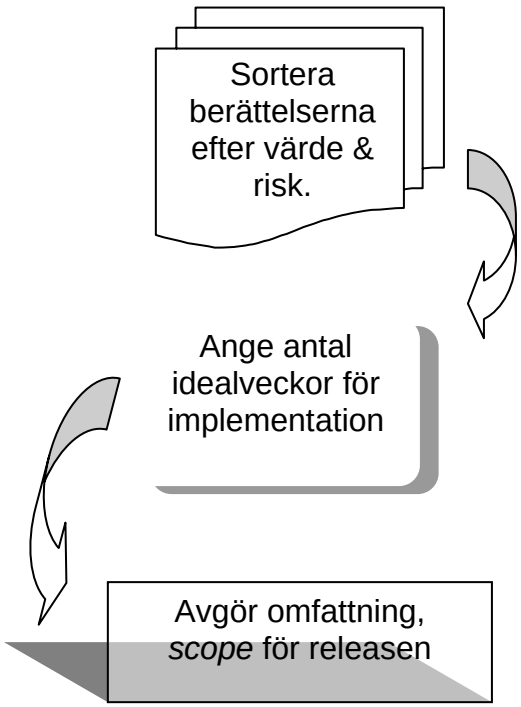
En lista av berättelser som skall ingå i releasen skapas, enligt följande steg:

1. Kunden sorterar berättelserna i ordningen ”viktigast först”, tre högar högst-, mellan- och lägst prioritet.

2. Utvecklarna klassificerar berättelsernas utifrån risk. (detta steg är valfritt ty de uppskattade poängen anger också risk i någon mening.)

3. Utvecklingsteamet anger sin kapacitet, *velocity* uttryckt i idealtid per iteration eller antal poäng per iteration.

4. Kunden väljer omfattningen (*Scope*) för releasen, dvs. ”handlar” *user stories* utifrån en tidbudget baserad på längden av releasen och teamets kapacitet.



Figur 2. Tilldelning

4.5.4 Iterationsplanering

Arbetet med att implementera en release delas upp i ett antal iterationer och för att avgöra hur många iterationer som behövs skapas en iterationsplan. XP förespråkar korta iterationer och det har visat sig att iterationer bör vara max tre veckor [4] men beroende på typ av projekt och organisation kan iterationer bli ända upp till 20 veckor.
Under iterationsplaneringen väljer kunden återigen *user stories* men nu enbart från dem som ingår i releasen och antalet ska motsvara arbete för vald iterationslängd. Exempel, iterationslängden två veckor medför att *user stories* vars sammanlagda antal poäng totalt blir två veckors jobb kan väljas. Syftet med iterationsplanen är alltså att dela upp implementering av releasens berättelser i mindre utvecklings cykler, där varje cykel levererar ett körbart system av värde för kund.

Tasks – *Programmeringsuppgifter*
Iterationsplaneringen startar med att utvecklarna gemensamt bryter ner varje *user story* i *tasks* [1] där varje *task* är mindre och mer detaljrika uppgifter som konkret beskriver vad som skall implementeras. Iterationsplanerings två faser, ”brainstorming” och ansvarsfördelning består av följande delar:

Fas 1: ***Brainstorming***

1. Kunden läser upp en berättelse.
2. Utvecklarna brainstormar tillsammans för att dela in berättelsen i mindre uppgifter (tasks). Varje uppgift noteras på ett indexkort.
3. Fortsätt tills alla berättelser är uppdelade i tasks.
    3.1 Om några av berättelserna inte går att dela upp ber man kunden att vänta med dessa
4. Samla ihop uppgiftskorten och inled den andra fasen av planeringen.

Fas 2: ***Tilldelning***

1. Utvecklarna väljer de *tasks* som var och en vill implementera och åtar sig ansvaret för dessa. Därefter beräknar varje utvecklare utvecklingstiden för sina uppgifter och antecknar det på baksidan av uppgiftskortet.
    1.1 Om någon av uppgifterna värderas till mer än 3 idealdagar/poäng måste uppgiften delas upp i mindre delar och på nytt fördelas.
2. Om några av uppgiftskorten blir över och ingen kan ta ansvar för dem, ber man kunden vänta med dessa.

4.6 *Acceptance tests*, kundens kontroll

I XP skapar kunden formella tester som kontrollerar att en *user story* implementerats korrekt, dvs. löser rätt uppgift. Testerna kallas *acceptance tests* [32] och är kundens kontroll att systemet uppfyller alla krav som förmedlats. Kunden ansvarar för att testerna testar rätt funktioner, men utvecklingsteamet hjälper till att skapa dem. Utvecklarna använder kundens tester för att verifiera att man förstått problemet och löst det på rätt sätt.

4.7 Avstämning av projektet

Under utvecklingen förvaras den nuvarande iterationsplanen väl synligt för projektdeltagarna. När *user stories* implementerats och testats klart dokumenteras den verkliga utvecklingstiden, därefter markeras berättelsen som slutförd i iterationsplanen.
Denna arbetsgång fyller flera funktioner, bland annat blir det lätt för teamet att se hur mycket arbete som kvarstår innan iterationen är klar och det är lätt att se om någon del av utvecklingen behöver extra resurser.
I slutet av varje iteration utförs en uppföljning av implementations-arbetet där det fastställs hur utvecklarna ligger till i förhållande till den uppskattade utvecklingstiden. Det är viktigt att detta görs ordentlig eftersom uppföljning av utvecklingstiden hjälper till att förbättra kommande skattningar av utvecklingstid.

4.8 Fördelar/nackdelar med planeringsmetoden

En av de främsta fördelarna med planeringsmetoden i XP är att den ger stor flexibilitet i hanteringen av krav på systemet, kunden kan när som helst under projektets gång göra förändringar av de befintliga kraven samt lägga till nya. Metoden gör att kostnaden för att ändra funktionalitet i systemet ligger på samma nivå genom hela projektet i stället för att växa exponentiellt med tiden, vilket annars är vanligt vid mjukvaruutveckling [3].

I planeringsmetoden hjälper utvecklingsteamet kunden att formulera *user stories*, kunden kommunicerar då direkt med utvecklarna vilket ger dem möjlighet att ställa frågor när kraven är otydliga. Detta tillsammans med att systemet bryts ner i mindre delar ökar förståelsen för kundens krav och systemet som skall byggas.

Att ha kunden nära knuten till projektet är både en fördel och en nackdel, det hjälper till genom att kunden fokuserar på det som är mest värdefullt men det kan vara en nackdel i beslut av mer teknisk karaktär där kunden inte har kunskap.

Fördelar:

- Mindre formaliteter och dokumentation.
- Hög flexibilitet.
- Mänsklig kommunikation.
- Fokuserar på att generera ”värde” för kunden.
- Kunden finns nära utvecklingsteamet i projektet.
- Bryter ned problem i mindre beståndsdelar.
- Hjälper kunden förstå vad systemet skall utföra.
- Feedback levereras kontinuerligt.

En av nackdelarna med metoden ligger i skapandet av *user stories*. Dessa måste formuleras så att de blir testbara, uttrycker något värdefullt och beskrivs i ett par meningar, en uppgift som inte är helt enkel för en ovan författare, vilket i det här fallet är kunden. En annan svaghet med metoden är att en mycket stor del av ansvaret ligger hos kunden, ett XP projekt där kunden inte är med och styr kommer med största sannolikhet att misslyckas [4]. Detta gör att höga krav ställs på rollen som kund och tyvärr är det den roll där utvecklare kan erbjuda minst hjälp.

Att som nybörjare planera projekt med XP är en stor utmaning, eftersom planeringssättet är ovant och från början ger upphov till stora bekymmer när utvecklingstid skall uppskattas. Att bara använda gissningar och intuition ger inte precis den tillförlitlighet man skulle önska vilket kan ses som något av en svaghet.

Nackdelar:

- Svårt att formulera bra berättelser, kräver erfarenhet.
- Stort styrande ansvar hos kund, ställer höga krav.
- Mycket svårt att bedöma utvecklingstid inledningsvis.

5. Utveckling av tidredovisningsapplikation med hjälp av XP och *the planning game*

5.1 Inledning

I den praktiska delen av examensarbetet har ett system för tidredovisning, *Time Tracker* implementerats. Systemet skall användas av anställda på företaget Vitec AB för att på ett snabbt och enkelt sätt dokumentera tid som investerats hos olika kunder och projekt. Verktyget har ett anpassningsbart gränssnitt där användaren kan ordna de ingående komponenterna efter eget tycke.
Systemet arbetar som en fristående applikation men distribueras via en webbserver vilket gör att applikationen själv kan avgöra när uppdateringar finns tillgängliga och hämta hem dessa vid behov. Metoden som används för detta kallas *No-touch deploy* och ingår i Microsofts ramverk .NET. Programmet betraktas således som en *rich-client* och kan samtidigt nyttja fördelarna hos en win32 applikation och en webbapplikation.

Utvecklingsprojektet har genomförts med en kund och en utvecklare vilket försvårat delar av planeringsspelet och XP, bl.a. värdering av *user stories*. Utöver detta bör projektet av två anledningar ses som ett experiment med *The Planning Game*. Dels har ingen av de inblandade parterna tidigare erfarenhet av projekt som planerats med metoden, och dels är utveckling med XP är anpassat för utvecklingsteam om 2 – 10 deltagare. Upplägget i detta fall rekommenderas alltså inte för XP [2].

5.2 Utvecklingsprocessen

Utvecklingsarbetet speglar tillämpningen av metodiken XP och planeringsprocessen, *The Planning Game*, vilket innebär att all utveckling skett utifrån de regler som gäller för XP. Några av momenten är av tidsbegränsande och av praktiska skäl undantagna, exempelvis parprogrammering.

Med den iterativa utvecklingsprocessen har systemet skapats i tre releaser. Varje release har delats in i iterationer, där iterationslängden anpassats efter rådande omständigheter. Vid slutet av varje iteration har ny funktionalitet integrerats med tidigare versioner av systemet.

För att säkerställa en om inte komplett men fungerande lösning vid projektets stoppdatum har utvecklingen skett utifrån *viktigast först*, sett ur kundens perspektiv.

5.3 Kravhanteringen i projektet

Kundens krav på systemet har utformats med korta övergripande *user stories* (se bilaga). Tillägg och modifiering av krav har skett kontinuerligt och planerats in i kommande releaser. Berättelserna har utarbetats av kund och nyttjats i planeringsspelet vid release- och iterationsplanering.

5.4. Release- och iterationsplanering med planeringsspelet

Releaseplaneringen har skapats med *The planning game* och med kunden och *user stories* som utgångspunkt har tre releaser planerats. Eftersom projektet är ett specialfall har tiden för varje release begränsats till mellan 2 och 5 dagar, vilket inte är normal tid för en release i ett XP projekt. Varje release innehåller berättelser som tillsammans bildar arbete för en förbestämd tidsperiod.
I bilaga A finns alla berättelser som användes i den inledande planeringen av projektet. När arbetet startade hade kunden redan en klar bild av grundfunktionerna i systemet och formulerade dem i korta berättelser. Tillsammans gick vi sedan igenom varje berättelse och diskuterade innebörden, resultatet blev att några ströks och några modifierades men tillräckligt många fanns för planering av release ett.
För att passa ett litet projekt valde vi att i projektets inledande del använda korta releaser och iterationer, releaselängden blev två dagar och iterationslängden blev en dag. Arbetet med att värdera berättelser och sedan bryta ner dem i tasks var väldigt svårt, tiden som skattades blev grovt tilltagen och stämde överhuvudtaget inte med verkligheten, en nyttig erfarenhet. Vid planering av release två försökte vi öka längden för till fyra dagar. De berättelser som valts till denna release värderades med avsikt något högre med tanke på missen vid första planeringen. Jag vet inte om det beror på att vi valt för korta utvecklingscykler men totalt sett tycker jag inte värderingen av berättelserna blev som förväntat, momentet får nog anses som det svåraste i hela planeringsprocessen.

I bilden nedan visas release två uppdelad i iterationer, den första siffran anger den uppskattade tiden och den andra verkliga tiden.
(I bilaga A.2 finns nedanstående *tasks* och *user stories* i helhet.)

Release 2	
Iteration 1:	Iteration 3:
Task 3. [8](10)	Task 5. [4](5)
	Task 6. [4](3)
Iteration 2:	
Task 1. [1](0,5)	Iteration 4
Task 2. [3](3,5)	Task 7. [2](2)+(3)
Task 4. [4](6)	Task 8. [2](2)

Nedan visas ett exempel på *user stories* som ingår i release 3

Userstories, Release 3 Omfattning: 3 dagar. (- = Uppskattad tid i dagar för en berättelse.)
[Användarinfo] - 0,25
Inloggad användares namn skall visas i namnlisten för programmet.

[Tabbordning] - 0,25
Med tab-tangenten skall hopp mellan de olika fälten ske i korrekt ordning för att underlätta inmatning från tgb.
Ordningen skall vara:
Företagslistan -> Aktivitet -> Produkt -> Antal timmar -> Spara

[Spårning av debitering] - 0,25
För att ge en tydlig överblick av vad som debiteras skall det i ett litet textfält framgå vilket företag och projekt som är valt, vilken aktivitet och produkt det gäller.
Förmateringen av detta bör vara i ordningen:
Projektnummer Kund - - - - - - - - - Projekt
Aktivitet --- Produkt

[Validera tidinmatning] - 0,25
Endast siffror skall kunna skrivas in i texfältet för timmar.Antalet inmatningsbara siffror kan begränsas till 2 st.

[Dockningsbara fönster] - 0,6
Listan som innehåller företag och projekt skall ha egenskapen av ett dockningsbart fönster.

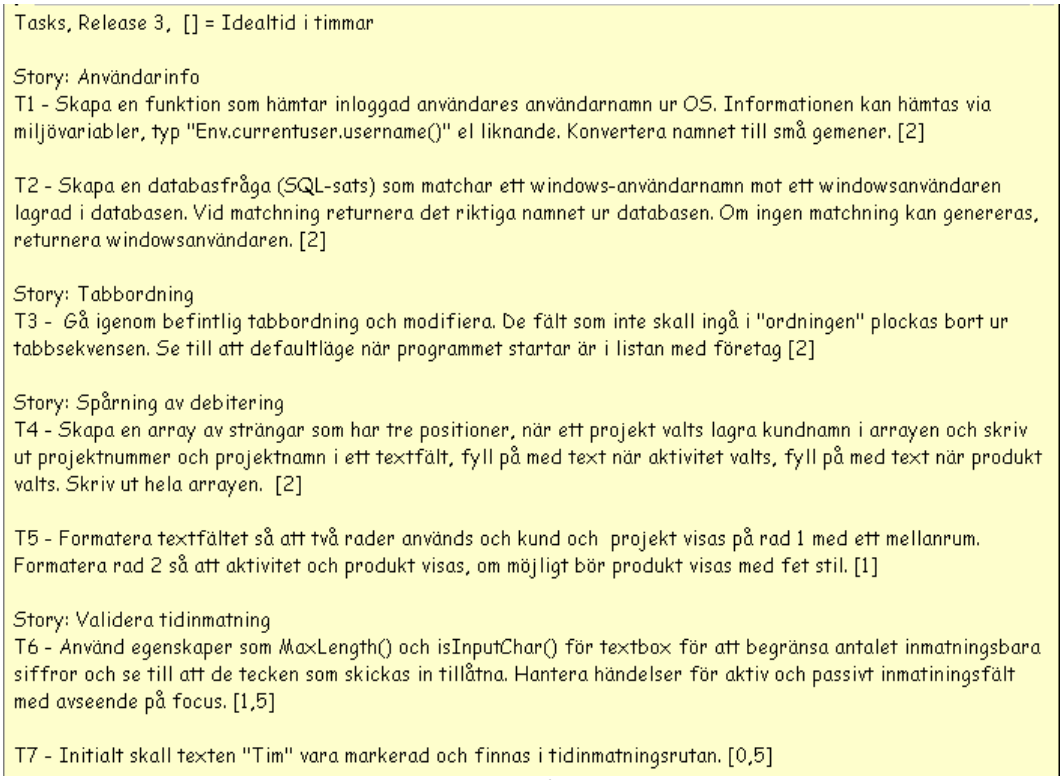
[Favoriter] - 0,4
En lista som kallas favoriter innehållande inloggad persons "favoritprojekt". (Kort lista)
Lägg till alternativ "Lägg till favoriter" i inmatningsfältet.
Skapa tabell "Favoriter" i databasen som lagrar projektnummer, aktivitetsnummer, produktnummer och fakturatext.

[Kortkommandon] - 0,3
Förutom klipp ut, klistra in och kopiera skall det finnas kortkommandon för att hoppa till respektive inmatningsfält i programmet. Ex. ctrl + i skall med automatik hoppa till fältet med "intern text". De fält som skall ha kortkommandon är Aktivitet, Produkt, Fakturatext och Intern text.

[Sparaknapp & Enterslag] - 0,5
Spara knapp eller "enter"-slag.
När alla uppgifter är inmatade skall antalet timmar anges därefter registreras datat mha spara-knappen.
Ett alternativ till sparaknappen är att bara trycka på entertangenten när tiden matats in.
Efter registrering bör data lämnas kvar i fälten för fakturatext, aktivitet, produkt och spårningsfältet.

Bild 2, *User stories*, Release 3

User stories som plockas ut till releasen delas upp i iterationer och därefter i *tasks*. Exempel på tasks och iterationer ur release 3 nedan.



Tasks, Release 3, [] = Idealtid i timmar

Story: Användarinfo
T1 - Skapa en funktion som hämtar inloggad användares användarnamn ur OS. Informationen kan hämtas via miljövariabler, typ "Env.currentuser.username()" el liknande. Konvertera namnet till små gemener. [2]

T2 - Skapa en databasfråga (SQL-sats) som matchar ett windows-användarnamn mot ett windowsanvändaren lagrad i databasen. Vid matchning returnera det riktiga namnet ur databasen. Om ingen matchning kan genereras, returnera windowsanvändaren. [2]

Story: Tabbordning
T3 - Gå igenom befintlig tabbordning och modifiera. De fält som inte skall ingå i "ordningen" plockas bort ur tabbsekvensen. Se till att defaultläge när programmet startar är i listan med företag [2]

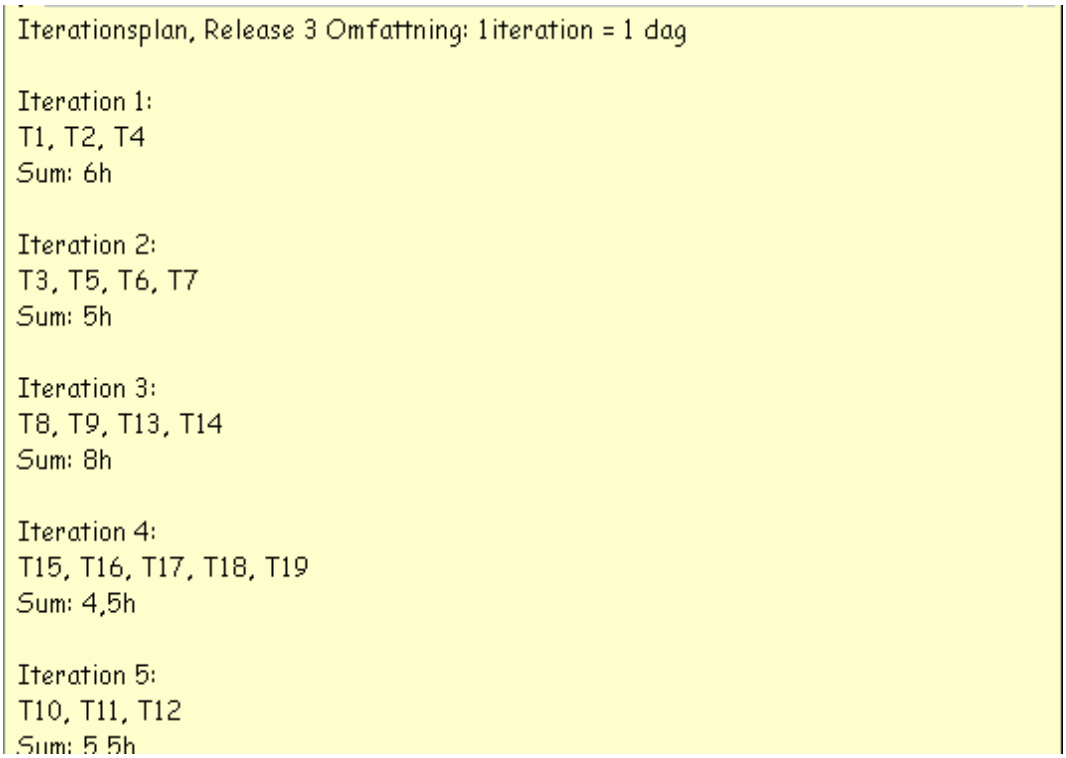
Story: Spårning av debitering
T4 - Skapa en array av strängar som har tre positioner, när ett projekt valts lagra kundnamn i arrayen och skriv ut projektnummer och projektnamn i ett textfält, fyll på med text när aktivitet valts, fyll på med text när produkt valts. Skriv ut hela arrayen. [2]

T5 - Formatera textfältet så att två rader används och kund och projekt visas på rad 1 med ett mellanrum. Formatera rad 2 så att aktivitet och produkt visas, om möjligt bör produkt visas med fet stil. [1]

Story: Validera tidinmatning
T6 - Använd egenskaper som MaxLength() och isInputChar() för textbox för att begränsa antalet inmatningsbara siffror och se till att de tecken som skickas in tillåtna. Hantera händelser för aktiv och passivt inmatiningsfält med avseende på focus. [1,5]

T7 - Initialt skall texten "Tim" vara markerad och finnas i tidinmatningsrutan. [0,5]

Bild 3, Tasks från release 3



Iterationsplan, Release 3 Omfattning: 1iteration = 1 dag

Iteration 1:
T1, T2, T4
Sum: 6h

Iteration 2:
T3, T5, T6, T7
Sum: 5h

Iteration 3:
T8, T9, T13, T14
Sum: 8h

Iteration 4:
T15, T16, T17, T18, T19
Sum: 4,5h

Iteration 5:
T10, T11, T12
Sum: 5,5h

Bild 4, Iterationerna i release 3

För hela planeringen med releaser, iterationer och *user stories*, se bilaga A.

5.5. Applikationen

Programmet har fått namnet Time Tracker 2003. Syftet med programmet är som namnet antyder att ”spåra” tid eller närmare bestämt hålla reda på tid som skall debiteras. När programmet startar identifieras användaren och en koppling till servern med databasen skapas. Från databasen hämtas uppgifter om kunder och projekt som sedan läses in i programmet. Varje användare har möjlighet att lagra egna ”favoriter”, dvs. inställda kombinationer av företag, projekt, aktivitet, fakturatext och debiterbar tid. Dessa läses också in när programmet startar.

Nedan visas en bild på hur programmet ser ut efter uppstart.

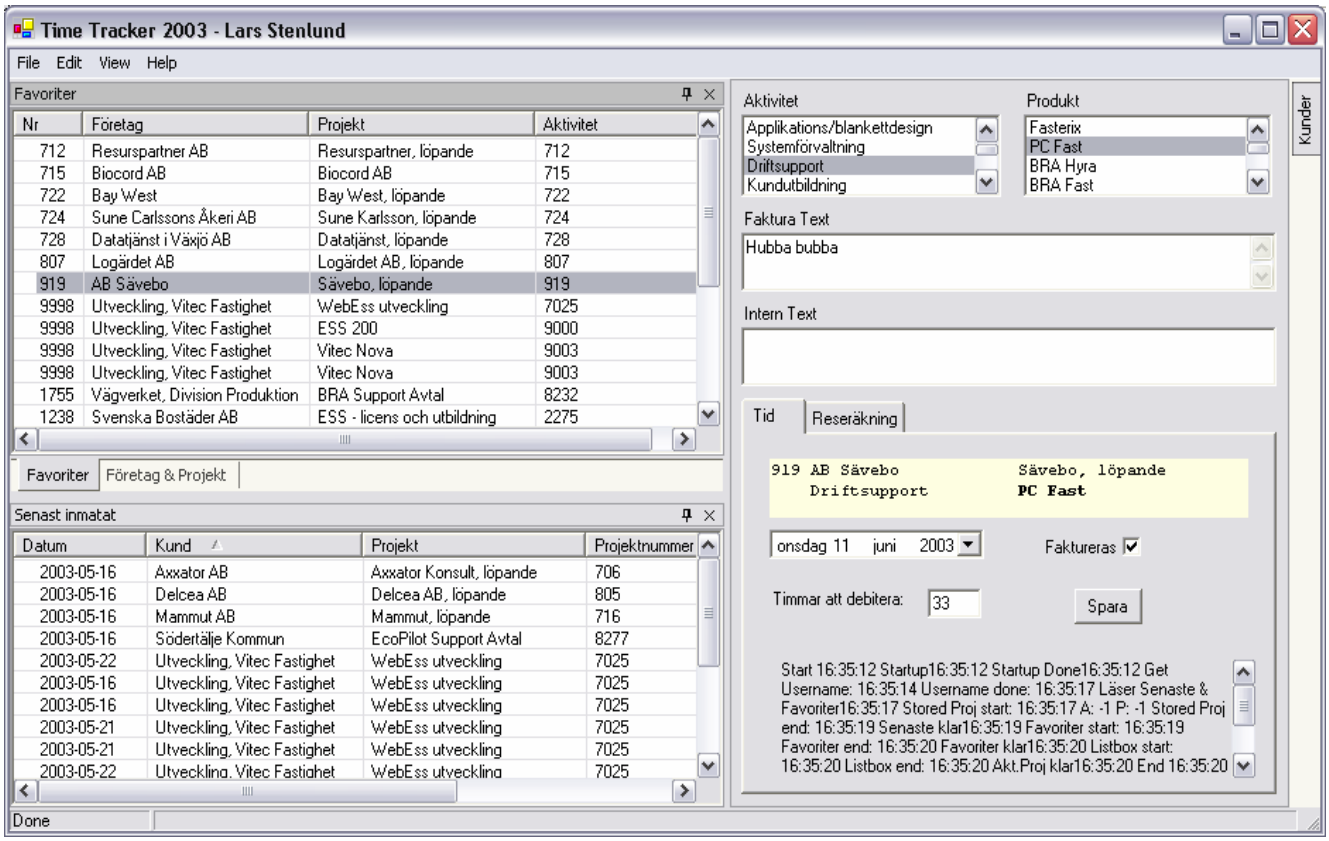


Bild 5, Time Tracker 2003

5.5.1 *Web deploy*

För att applikationen skall fungera krävs att maskinen som exekverar programmet har ramverket Microsoft .NET installerat. Ramverket innehåller en CLR – Common Language Runtime (se bild 6), som används vid exekveringen av applikationer skrivna för .NET. När användaren klickar på länken som hänvisar till det körbara programmet konsulteras ramverket som avgör om det körbara programmet är skrivet för .NET. Därefter exekveras programmet i CLR.

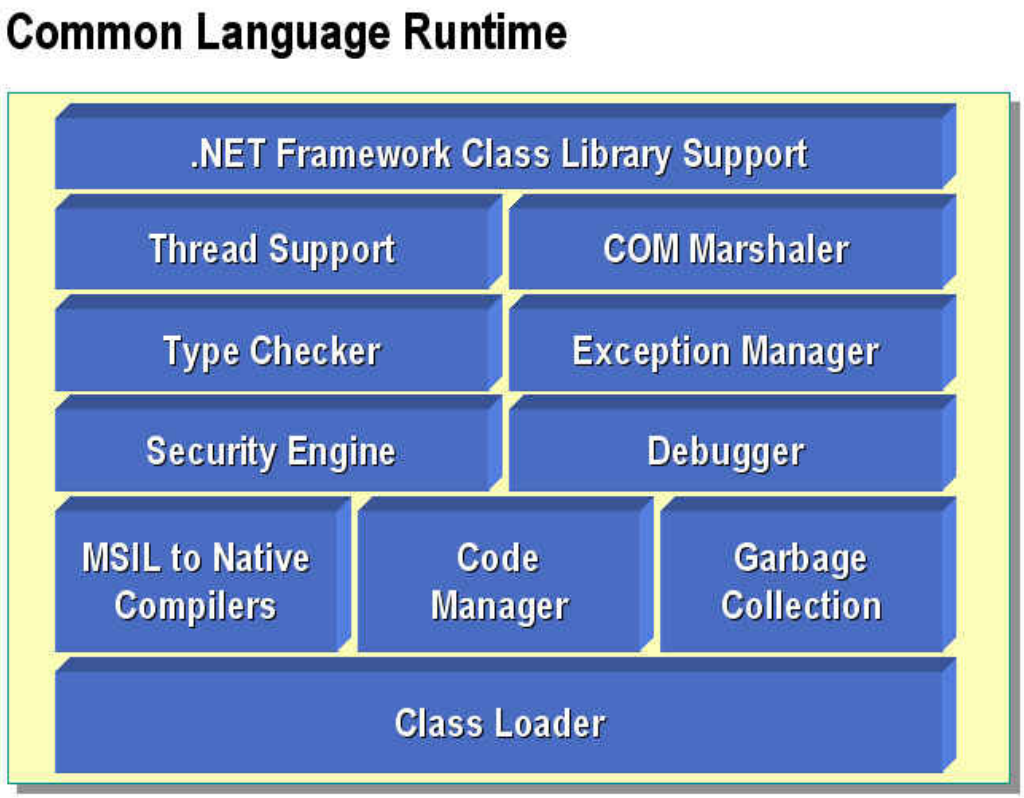


Bild 6, DotNet CLR

5.5.2 Anpassningsbara användargränssnitt med Syncfusion

Ett av kraven på systemet var att gränssnittet skulle vara anpassningsbart för användaren. För att åstadkomma detta skapades användargränssnittet med hjälp av dockningsbara komponenter från utvecklingspaketet Syncfusion. Detta ger användaren möjlighet att själv välja hur fönstren i programmet skall visas. Bilden nedan visar tidredovisningsprogrammet där ett fönster dockats i vänstra och högra programkanten.

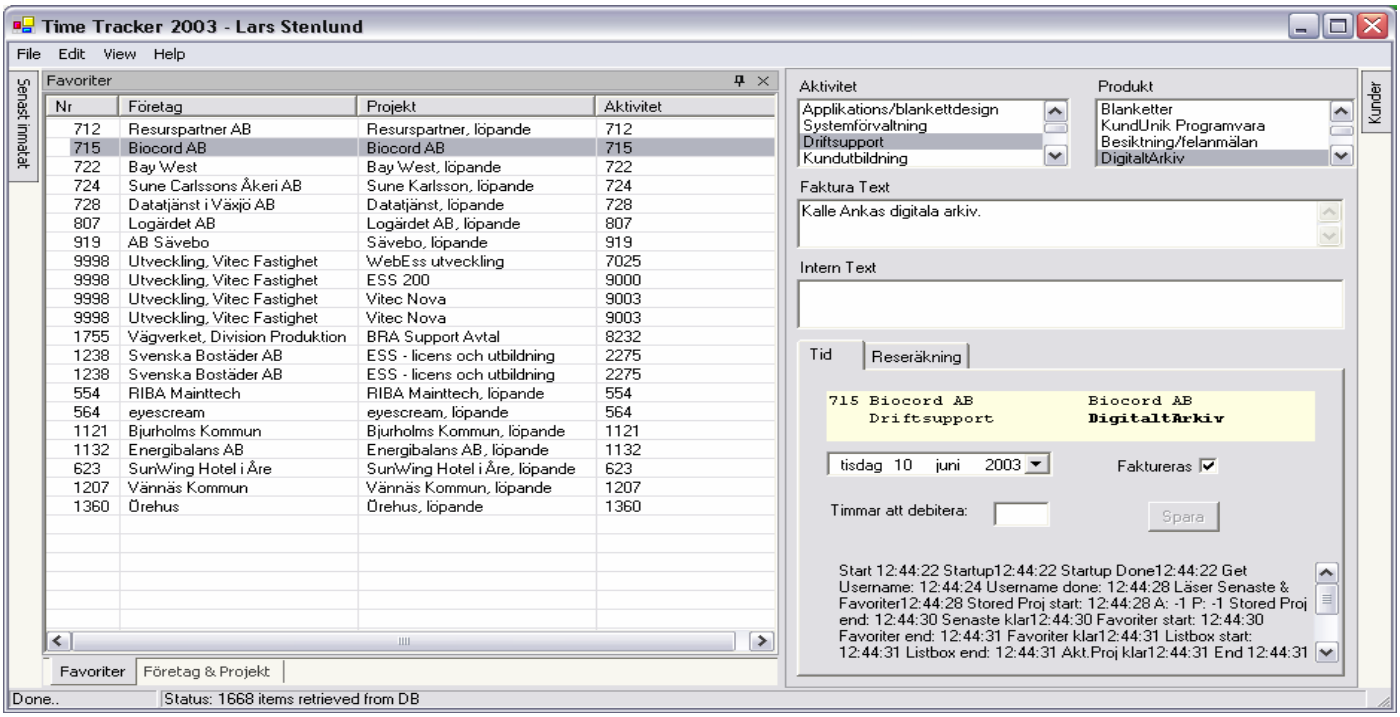


Bild 7, Dockade fönster, (*Syncfusion*)

5.6. Resultat

Systemet som utvecklats kan hantera inmatning av tider, lagring av data i databas, redigering av senast inmatade tider samt hantera favoriter i form av förutbestämda projekt- och aktivitet kombinationer. Gränssnittet är anpassningsbart och användaren kan ändra ordningen på komponenterna som programmet innehåller. Applikationen är utformad som en *rich client* och distribueras via en webbserver.

De kriterier som beskrevs i avgränsningen har realiserats så när som på ett par punkter. Därför bör resultatet av projektet anses stå i proportion till vad som förväntades.

I utvecklingsarbetet saknas framförallt testningen av systemet med hjälp av enhetstester vilka uteblev helt, både enhetstester och kundens tester. Eftersom det är en viktig del av arbetssättet i XP skulle den delen ha beaktats betydligt mer.

5.7. Framtida versioner

Applikationen kommer att slutföras under de närmaste månaderna då ett fullt fungerande system skall stå klart att använda.

6. Diskussion

Metoden för planering av projekt i XP är enkel i jämförelse med traditionell planering, principen att beskriva ett system i form av korta berättelser gör det lättare för kunden att förmedla sina krav. Men enkelheten i metoden är kanske den faktor som gör den relativt svår att ta till sig. Det krävs mycket arbete för att bli bra på att formulera bra *user stories* och sedan värdera dem rätt. Principen för att mäta framgång och motgång i utvecklingsarbetet bygger på att antalet avklarade user stories. Uppföljning av utvecklingen kan tyckas enkel och aningen bristfällig då måttet som används bara är antalet implementerade *user stories*. Men det ger naturligtvis en bild av hur mycket jobb som är kvar men också en bild av utvecklingsteamets kapacitet vilket är mer värdefullt vid nästa planeringsmöte.

Eftersom utvecklingsarbetet pågår parallellt med planeringen i ett XP projekt skapas körbar version av systemet som går att exekvera och provköra redan efter första iterationen. Kunden kan alltså kontinuerligt testa sitt system och bilda sig en uppfattning som sedan används för att påverka fortsatta arbetet. I ett vanligt programutvecklingsprojekt sker implementationsarbetet i slutet av projektet, fram till dess existerar systemet endast i form av diagram och specifikationer. I det avseendet känns det som att metoden i XP ger kunden mer utrymme att skapa ett system som uppfyller önskemålen.

Ett av frågetecken med XP är hur väl det står sig i jämförelse med de långtgående fördelarna som ett väldokumenterat system har. På den punkten hävdar förespråkare av XP att dokumentationen i XP som utgörs av källkod och enhetstester mycket väl matchar ett väldokumenterat system eftersom källkoden och testerna motsvarar den informationen som dokumenteras. Metoden kanske fungerar för mindre system och för dem som varit med i projektet och skapat koden. Men för helt utomstående måste ett dokument som beskriver koden och vad den gör i klartext vara lättare att ta till sig än enhetstester och källkod i XP som fortfarande är kod. De flesta som läst någon annans kod vet att det inte alltid är så lätt att följa direkt.

För projekt som tillämpar XP rekommenderas 2 – 10 deltagare i utvecklingsteamet, den undre gränsen bör absolut ses som ett minimum (enligt egen erfarenhet). I projektet som utförts i detta arbete har den gränsen understigits då bara en utvecklare funnits tillgänglig. Erfarenheten från det är att hela konceptet med *planning game*, enkelhet och framförallt kommunikation fallerar och är beroende av att det finns minst två utvecklare i teamet utöver kunden. Inte minst gäller det i arbetet med att skriva, värdera och dela upp *user stories*.

Svagheterna som finns i metoden ligger främst i de krav som ställs på deltagarna, framförallt kunden. Det går inte att anta att kunden direkt kan skriva bra user stories eller en vill. Det går inte heller ta för givet att beställaren i ett inledande skede av utvecklingsprojektet vet exakt hur de vill att systemet ska se ut och vilka funktioner som ska ingå.
Det första av problemen har ingen självklar lösning, vilket kan ses som något av en osäkerhetsfaktor.

Som nybörjare känns planeringsmetoden något osäker eftersom det är så pass svårt att bedöma utvecklingstid för berättelser, faktum är att planeringsmetoden har en tröskel som måste överkommas innan den blir riktigt användbar. Utifrån egna erfarenheter skulle jag rekommendera att planeringsmetoden används i ett par övningsprojekt innan den appliceras i en skarp situation.

Trots detta tror jag att planeringsmetoden *the planning game* och XP som helhet ska ses som ett av de främsta alternativen till de tyngre metodikerna.
Ett utvecklingsteam som blir duktiga på XP kommer sannolikt att bygga programvaror både snabbare och billigare vilket är något som borde tilltala alla, kunden framförallt. Därför tror jag att lättviktsmetodiker som XP sannolikt kommer att dominera mjukvaruutvecklingen framöver.

7. Tack

Jag vill rikta ett stort tack till Lars Stenlund, VD Vitec Fastighetssystem AB som gett mig möjlighet att utföra ett examensarbete på Vitec AB i Umeå samt ställt upp med handledningsresurser.

Dessutom vill jag tacka Jan Wikström, utvecklare Vitec AB för konsultation i utvecklingsarbetet.

Tack också till övrig personal på Vitec AB som förgyllt fredagarna med gott fredagsfika.

Slutligen vill jag tacka den interna handledaren vid Umeå Universitet Jürgen Börstler.

8. Referenser

8.1. Böcker

[1] C.Martin, Robert, “Agile Software Development”, Prentice Hall Inc. 2003.

[2] Wake William C., “Extreme Programming Explored”, Addison-Wesley, 2000.

[3] Beck Kent, “eXtreme Programming explained”, Upper Saddle River, NJ, Addison-Wesley, 2000.

[4] Martin Fowler, Beck Kent “Planning Extreme Programming”, Addison Wesley Longman, Inc. 2001.

[5] Newkirk, James, C.Martin, Robert, “Extreme Programming in Practice”, Addison Wesley Longman, Inc. 2001.

[6] Jeffries Ron, Anderson Ann, Hendrickson Chet, “Extreme Programming Installed”, Upper Saddle River, Addison-Wesley, 2000.

[7] Auer Ken, Miller Roy, Cunningham Ward, “Extreme Programming Applied: *Playing to Win”*, Addison Wesley Longman, Inc. 2001.

8.2. Artiklar

[10] Eiderbäck, Björn, ”Programutveckling med XP, Vad?, Varför? och Hur?”, *oopsWiiki*, 3 Maj 2001

[11] Smith, John, “A Comparision of RUP and XP”, Rational White Papers, May 2001.

8.3 Internet sidor

[20] The XP Series, (2003-03-10), http://www.aw.com/cseng/series/XP/

[21] Vitec AB, (2003-04-16) http://www.vitec.se

[22] On-Site Customer, (2003-03-04) http://c2.com/cgi/wiki?OnsiteCustomer

[23] What is Microsoft .NET Framework, (2003-04-24) http://www.microsoft.com/net/basics/framework.asp

[24] *No-Touch Deployment in .NET*, (2003-04-24) http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dv_vstechart/html/vbtchno-touchdeploymentinnetframework.asp

[25] SQL Manual, “Vyer och Lagrade procedurer i SQL”, (2003-04-04) http://support.internethotellet.com/hotellnt/sql/manual/

[26] Enhetstestning, (2003-03-04), http://c2.com/cgi/wiki?UnitTest

[27] Collective Ownership, (2003-05-13)
http://www.xp123.com/xplor/xp0206a/index.shtml

[28] NUnit, *Source Code Testing Tool*, (2003-04-24) http://www.nunit.com/

[29] Metaphor, (2003-02-20)
http://www.xprogramming.com/xpmag/whatisxp.htm#metaphor

[30] Jason, P. Charvat, "*Heavyweight vs. lightweight methodologies*" (2003-05-06),http://www.zdnet.com.au/builder/manage/project/story/0,2000035082,20270383,00.htm

[31] Användarfall**,** *Use cases*, (2003-03-10),
http://www3.telge.kth.se/tkn/together/

[32] *Acceptance Test*, (2003-03-10)
http://www.extremeprogramming.org/stories//functionaltests2.html

Bilaga A

A.1 *User stories*

Flerföretag
Stöd för flera ftg map ftg-reg+proj+produkt samt autoval beroende på inloggning.

Tvingande fält
I vissa kombinationer ska man kunna sätta upp tvingande fylleri av ftg/proj/aktivitet/produkt

Import av ftg-register
Ska kunna importera ftg-register från textfil och varna vid uppdatering av befintlig post. Från XOR compact.

GUI constraints
Inmatning skall kunna ske enbart med tgbord och då också effektivt.
Dubbelkommandon typ ctrl+y undviks, enbart tillåtet om Windowsstandard, typ klipp ock klistra.
Favoriter på snabbval med funktionstangenter med tooltip el. liknande label.

Påminnelse
Tidsstyrt jobb som ska kunna påminna folk som inte matat in sina tider senaste veckan. Mailpåminnelse. Ska styras av superuser.

Statusändring
Görs genom att göra ett webfönster i appen som sen anropar befintlig .asp-kod.
Obehandlad, avvakta etc.

Fakturaunderlag
Timmar markerade som "Ska Faktureras" ska kunna väljas i en lista med full GUI-funktionalitet och sen göras preview och utskrift på (även .pdf). Sen ska fakturanummer kunna anges.

Analys
Ska finnas i komplett form utifrån Tidanalys-pgmet. Kan göras på enskild users projekt och egna tider. Superuser har full analys - alla projekt, personer och kunder

Behörighetsnivåer
Två nivåer, user och superuser

Avsluta projekt
Det ska vara enkelt att avsluta ett projekt, dock bara för den som la upp det eller för superuser.

Nytt projekt
Nytt projekt skall kunna läggas upp med automatisk tilldelning av projektnummer av vem som helst.

Inmatningsfält
Alla fält som finns i gamla programmet skall finnas.

Redigering
Man ska kunna redigera effektivt, t.ex. byta status på många poster eller byta projekt på många poster genom att markera en listview och sen utföra redigeringen

Tidmätning
En funktion för tidmätning pss som i ett annat pgm

Favoriter
Man ska kunna lägga upp sina favoriter map kund/projekt

Inloggning
Ska ske med automatik utifrån inloggad NT user

Dubbla textfält
Det ska finnas två textfält, ett för internt bruk och ett som kan visas för kund på fakturaunderlaget.

A.2 Release 1

Userstories, Release 1

[Inloggning]
Ska ske med automatik utifrån inloggad NT user, Användarens användarnamn hämtas ur databasen.

[Tidinmatning]
Användaren ska kunna välja befintliga företag och projekt ur en listview.

[Dubbla textfält]
Det ska finnas två textfält, ett för internt bruk och ett som kan visas för kund på fakturaunderlaget.

Tasks, Release 1

Tasks: [] = estimerad tidsåtgång i idealtimmar.
Story: *Tidinmatning*
T1 - Skapa ett Gui - skal för programmet med menyrad. Designa Gui för inmatningsdel. [8]

T2 - Hämta data ur databas, läs in alla kunder, projekt och projektnummer från databasen och presentera i en "ListView". [4]

Story: Inloggning
T3 - Modifiera databasen så att den innehåller ett fält som lagrar en användares riktiga namn. Pröva att matcha windowsanvändaren mot namnet i det nya fältet. [4]

Story: *Dubbla textfält*
T4 – Skapa inmatningsfält för Intern text och fakturatext.

Iterationsplanering, (Release 1)

Iterationslängden är 1 dag. (Komprimerad för att passa mitt projekt.)

Iteration 1:
Task 1.
Task 4.

Iteration 2:
Task 2.
Task 3.

A.3 Release 2

Userstories, Release 2

[Inmatningsfält]
Alla fält som finns i det gamla programmet skall finnas med i det nya.

[Första bokstaven i projekt- eller företags- namn.]
När data är inläst till en listview ska man kunna markera en av kolumnerna och därefter trycka första bokstaven i projektnamnet eller företagsnamnet, varpå detta bläddras fram.

[No-touch deploy]
Applikationen skall distribueras som en "Rich client" och kunna köras via webben.

Tasks, Release 2

Tasks: [] = Antal idealtimmar

Story: Inmatning
T1 - Skapa och placera in alla fält från gamla programmet i det nya. [1]

T2 - Skapa nya inmatningsfält, textboxes, för inmatning av reseräkning.
Start-tid och slut-tid för resan matas in i var sin textbox.
Övriga utlägg matas in i en lista med två fält, "typ av utlägg" och "belopp". [3]

Story: No-touch deploy
T3 - No-touch deploy, lägg ut programmet så att det går att köra från en webbserver (Hämta hem vid behov), verifiera att uppdatering fungerar.
Lägg ut det befintliga programmet. [8]

Story: Första bokstaven i projekt- eller företags- namn.
T4 - Sortera listan med företag i bokstavsordning, fallande och stigande. [4]

T5 - Skapa en funktion reagerar på knapptryckning i listview och talar om vilken kolumn som är markerad och vilken tangent som trycktes ner. [4]

T6 - Skapa en funktion som har två argument, en sorterad kolumn (index som anger vilken kolumn) och en bokstav. Leta upp första instansen av ord i kolumnen som börjar med denna bokstav. returnera ett index som talar om vilken rad ordet finns på. [4]

T7 - Verifiera att funktionen med bokstäver fungerar, ta hand om eventuella execptions.[2]

T8 - Integrera nya delar med befintligt system. Skapa enhetstester för testningsbara funktioner.[2]

Iterationsplanering, Release 2
Iterationslängd = 1 dag

Iteration 1:
Task 3. [8]

Iteration 2:
Task 1. [1](0,5)
Task 2. [3](3)
Task 4. [4](6)

Iteration 3:
Task 5. [4](5)
Task 6. [4](3)

Iteration 4
Task 7. [2](2)+(3)
Task 8. [2](2)

A.4 Release 3

Userstories, Release 3

(- = Uppskattad tid i dagar för en berättelse.)

[Användarinfo] - 0,25
Inloggad användares namn skall visas i namnlisten för programmet.

[Tabbordning] - 0,25
Med tab-tangenten skall hopp mellan de olika fälten ske i korrekt ordning för att underlätta inmatning från tgb.
Ordningen skall vara:
Företagslistan -> Aktivitet -> Produkt -> Antal timmar -> Spara

[Spårning av debitering] - 0,25
För att ge en tydlig överblick av vad som debiteras skall det i ett litet textfält framgå vilket företag och projekt som är valt, vilken aktivitet och produkt det gäller.
Förmateringen av detta bör vara i ordningen:
Projektnummer Kund - - - - - - - - - Projekt
Aktivitet --- Produkt

[Validera tidinmatning] - 0,25
Endast siffror skall kunna skrivas in i texfältet för timmar.
Antalet inmatningsbara siffror kan begränsas till 2 st.

[Dockningsbara fönster] - 0,6
Listan som innehåller företag och projekt skall ha egenskapen av ett dockningsbart fönster.

[Favoriter] - 0,4
En lista som kallas favoriter innehållande inloggad persons "favoritprojekt". (Kort lista)
Lägg till alternativ "Lägg till favoriter" i inmatningsfältet.
Skapa tabell "Favoriter" i databasen som lagrar projektnummer, aktivitetsnummer, produktnummer och fakturatext.

[Kortkommandon] - 0,3
Förutom klipp ut, klistra in och kopiera skall det finnas kortkommandon för att hoppa till respektive inmatningsfält i programmet. Ex. ctrl + i skall med automatik hoppa till fältet med "intern text". De fält som skall ha kortkommandon är Aktivitet, Produkt, Fakturatext och Intern text.

[Sparaknapp & Enterslag] - 0,5
Spara knapp eller "enter"-slag.
När alla uppgifter är inmatade skall antalet timmar anges därefter registreras datat mha spara-knappen.
Ett alternativ till sparaknappen är att bara trycka på entertangenten när tiden matats in.
Efter registrering bör data lämnas kvar i fälten för fakturatext, aktivitet, produkt och spårningsfältet.

Summa: 2,8 dagar

Tasks, Release 3

[] = Idealtid i timmar

Story: Användarinfo
T1 - Skapa en funktion som hämtar inloggad användares användarnamn ur OS. Informationen kan hämtas via miljövariabler, typ "Env.currentuser.username()" el liknande. Konvertera namnet till små gemener. [2]

T2 - Skapa en databasfråga (SQL-sats) som matchar ett windows-användarnamn mot ett windowsanvändaren lagrad i databasen. Vid matchning returnera det riktiga namnet ur databasen. Om ingen matchning kan genereras, returnera windowsanvändaren. [2]

Story: Tabbordning
T3 - Gå igenom befintlig tabbordning och modifiera. De fält som inte skall ingå i "ordningen" plockas bort ur tabbsekvensen. Se till att defaultläge när programmet startar är i listan med företag [2]

Story: Spårning av debitering
T4 - Skapa en array av strängar som har tre positioner, när ett projekt valts lagra kundnamn i arrayen och skriv ut projektnummer och projektnamn i ett textfält, fyll på med text när aktivitet valts, fyll på med text när produkt valts. Skriv ut hela arrayen. [2]

T5 - Formatera textfältet så att två rader används och kund och projekt visas på rad 1 med ett mellanrum. Formatera rad 2 så att aktivitet och produkt visas, om möjligt bör produkt visas med fet stil. [1]

Story: Validera tidinmatning

T6 - Använd egenskaper som MaxLength() och isInputChar() för textbox för att begränsa antalet inmatningsbara siffror och se till att de tecken som skickas in tillåtna. Hantera händelser för aktiv och passivt inmatiningsfält med avseende på focus. [1,5]

T7 - Initialt skall texten "Tim" vara markerad och finnas i tidinmatningsrutan. [0,5]

Story: Dockningsbara fönster
T8 - Gå igenom (översiktligt) Syncfusion -objekten för dockningsbara fönster och granska "DockingManager-" och "DockingClient-" objekten. Gå igenom eventuella exempel [3]

T9 - Bygg om nuvarande gränssnitt så att listan med företag och projekt visas i ett fönster som är dockningsbart. Anpassa arbetsytan så att befintliga objekt kan växa när ett dockningsbart fönster döljs eller flyttas. [3]

Story: Favoriter
T10 - Skapa ytterligare ett dockningsbart fönster, "Favoriter" som passar ihop med fliken "Företag". [1,5]

T11 - Lägg till ett fält i databasen som indikerar Utöka databasklassen med en fråga som läser in inloggad persons projekt som markerats som favorit. [2]

T12 - Hantera inläsning av projekt som markerats "favorit", sker vid initiering av programmet. Hantera scenario med 0 favoriter, hoppa över inläsning. Hantera uppdatering av favortier vid inmatning av ny favorit, läs in det nya objektet i listan med favoriter. [2]

Story: Kortkommandon
T13 - Skapa till ett "event" som reagerar på tangentkombinationer, ctrl + i, ctrl + a, ctrl + p. [1]

T14 - Vid vald kombination skall respektive fönster hamna i focus. När textfälten hamnar i focus skall befintlig text markeras för att snabbt kunna bytas ut. [1]

Story: Sparaknapp & Enterslag
T15 - Uppdatera databasklassen med ytterligare en SQL-sats som uppdaterar databasen med datat som angivits under inmatning. Verifiera att satsen funktionerar enligt önskemål. [1]

T16 - När inmatningen avslutas med sparaknappen eller entertangent skall data lagras i databasen. Kontrollera om "lägg till favoriter" är markerad. Uppdatera favoriter listan.[2]

T17 - I tidinmatningsrutan, skapa ett event som reagerar på entertangenten.[0,5]

T18 - Vid avslut med enterslag skall en kontroll utföras som bevakar antalet tecken inmatat i tidrutan och hanterar ett "NULL" värde. [0,5]

T19 - Efter avslutad inmatning rensa tidinmatningsfältet, men lämna övriga textfält. Sätt focus på listan med företag.[0,5]

Summa idealtid: 29 timmar

Iterationsplan, Release 3

Omfattning: 1iteration/dag

Iteration 1:
T1, T2, T4
Sum: 6h

Iteration 2:
T3, T5, T6, T7
Sum: 5h

Iteration 3:
T8, T9, T13, T14
Sum: 8h

Iteration 4:
T15, T16, T17, T18, T19
Sum: 4,5h

Iteration 5:
T10, T11, T12
Sum: 5,5h

Summa: 5 Iterationer => 5 Idealdagar