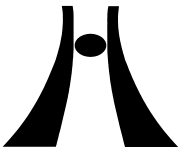
Beteckning:_____



HÖGSKOLAN I GÄVLE

Akademin för teknik och miljö

Release, deploy och distribution vid plugin-utveckling med Eclipse.

Hur detta kan stödjas av en modern utvecklingsmiljö för Java.

Johan Nordlinder
juni 2010

Examensarbete, 15 högskolepoäng, C
Datavetenskap

Dataingenjörsprogrammet
Examinator: Jonas Boustedt Medbedömare: Bengt Östberg
Handledare: Erik Norfelt, Jonas Boustedt

Release, deploy och distribution vid plugin-utveckling med Eclipse.

Hur detta kan stödjas av en modern utvecklingsmiljö för Java.

av

Johan Nordlinder

Akademin för teknik och miljö
Högskolan i Gävle

S-801 76 Gävle, Sweden

Email:
Ndv07jnr@student.hig.se

Abstrakt

Utveckling av olika typer av påbyggnadskod till programvaror blir allt vanligare. Dessa som går under samlingsnamnet plugins skiljer sig från vanliga applikationer då de har en annan struktur samt speciella beroenden till applikationsspecifika moduler. Problem uppstår när denna typ av utveckling inte stöds av de vanliga utvecklingsmiljöer som finns ute på företagen och delar som borde vara automatiserade måste utföras manuellt. Syftet med studien är att undersöka hur utvecklingsmiljön för Java på Sandvik IT Services kan anpassas för att stödja plugin-utveckling för IBM Lotus Notes. I denna studie undersöks skillnaden mellan plugin-utveckling och den vanliga Java-utvecklingen på företaget samt hur detta påverkar verktygen i utvecklingsmiljön. Resultatet beskriver hur utvecklingsmiljön kan anpassas för att stödja plugin-utveckling och en lösning för detta föreslås. Slutligen visas en implementation av lösningen i form av en prototyp där utvecklingsmiljön anpassas för plugins med Maven pluginet Tycho.

Nyckelord: Byggautomatisering, Java, Kontinuerlig Integration, Lotus Notes, Maven, Plugin, Tycho.

Inledning

1.1 Problem

På Sandvik IT Services[1] används idag ett verktyg kallat Lotus Domino/Notes för kollaboration, schemaläggning, e-post samt övrig kommunikation anställda emellan. Verktyget är klient/server baserat; Domino är serverns benämning medan klienten kallas för Notes.

Företaget arbetar mycket med Javautveckling och till detta används utvecklingsverktyget Eclipse. Detta består av en samling bibliotek och plugins som tillsammans formar ett dynamiskt verktyg för programvaruutveckling. Eclipse har anpassats för ett flertal programmeringsspråk och användningsområden, ett av dessa är följaktligen att bygga på själva Eclipse genom så kallad plugin-utveckling. Det finns idag en uppsjö av plugins avsedda att utöka funktionaliteten hos Eclipse på ett flertal områden. Det kan till exempel röra sig om plugins för revisionshantering eller plugins som förenklar användandet av databaser.

Det som är grunden för detta examensarbete är sambandet mellan dessa två delar, närmare bestämt mellan Lotus Notes och Eclipse. Från och med version 8.x av Notes finns här en koppling då biblioteken från Eclipse ligger som grund för Notes. Man kan alltså numera utveckla plugins till Lotus Notes på samma sätt som man tidigare utvecklat plugins till Eclipse.

Förutom användandet av Eclipse som utvecklingsverktyg har Sandvik idag även en lösning för kontinuerlig integration, revisionshantering samt distribution av sin Java-kod. Vid bygge av javaapplikationer sker detta på ett standardiserat sätt och byggprocessen använder sig av centralt delade referensbibliotek.

Dock skiljer sig utveckling av plugins mot utveckling av applikationer som tillhör Java SE[2] och passar följaktligen inte in i den befintliga lösningen för release, deploy och distribution[3]. Detta innebär kortfattat att man måste hantera filer manuellt och därmed får problem vid kollaboration. Dessutom försvinner mycket tid vid installationsprocessen av verktygen och risken finns att situationer med missmatchade versioner uppstår.

---

[1] Hädanefter förkortat som SITS

[2] Java Standard Edition, är en version av Java som används för desktop- och klient-applikationer

[3] Begreppen release, deploy, distribution förklaras i sina sammanhang i avsnitt 2.9

1.2 Syfte

Syftet med studien är att undersöka vilka möjligheter som finns för att förenkla och automatisera release, deploy och distribution vid utveckling av plugins till Lotus Notes, och att följaktligen undersöka huruvida de befintliga verktygen för denna process kan anpassas till att stödja även plugin-utveckling. Det övergripande målet är att om det är möjligt, visa hur utvecklingsmiljön på SITS kan anpassas för plugin-utveckling genom att sätta upp en prototyplösning.

Fördelarna för SITS är många och desamma som fördelarna med den befintliga utvecklingsmiljön för Java, vilka beskrivs utförligare i avsnitt 2.

1.3 Frågeställningar

- Hur kan man automatisera och därmed förenkla stegen release, deploy och distribution vid utveckling av plugins för Lotus Notes?
- Vad skiljer utveckling och distribution av plugins från utveckling av vanliga[4] javaapplikationer till den grad att den befintliga utvecklingsmiljön inte stöder detta?
- Vilka verktyg ingår i utvecklingsmiljön för vanliga javaapplikationer, och hur samspelar dessa?
- Kan dessa verktyg anpassas till att även fungera med utveckling och distribution av plugins för Lotus Notes?
- Har detta problem utforskats av någon annan aktör. Finns tidigare lösningar som kan vara till hjälp?
- Hur kan distribution/installation till klienten skötas?

1.4 Avgränsningar

Avgränsningar vid val av verktyg
Då denna studie fokuserar på utvecklingsmiljön vid SITS, kommer studier av för lösningen aktuella verktyg inte bara att bedömas utifrån ett funktionellt och tidmässig hållbart perspektiv, utan även hur väl de kan passa in bland företagets övriga IT-verktyg.

Avgränsningar vid genomförande
Då detta arbete har en strikt tidsbegränsning kommer en funnen lösning endast att implementeras i form av en prototyp, vars syfte är att visa att processen är möjlig och att ge underlag för resultatutvärdering.

[4] Ordet *vanliga* avser den typ av applikationer som tillhör Java SE

2 Teoretisk bakgrund, förutsättningar

Sandviks utvecklingsmiljö består av en samling verktyg som tillsammans erbjuder en effektiv lösning för release, deploy och distribution. Detta avsnitt förklarar relevanta begrepp samt belyser vilka problem som denna miljö bemöter och vilka verktyg som är involverade.

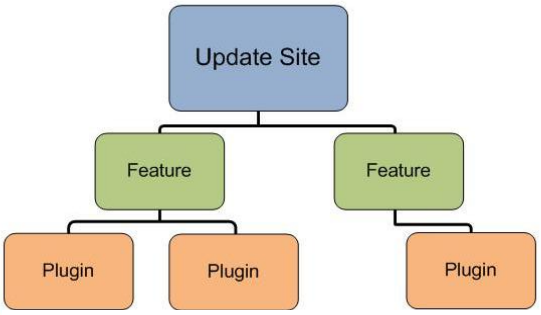
2.1 Proxy

Proxy direkt översatt till svenska som *ställföreträdare* är en applikation som agerar som en sådan vid förfrågningar. På företaget finns en www-proxy som all Internettrafik passerar igenom och förfrågningar riktade mot olämpliga sidor kan blockeras.

2.2 Plugin Development Environment (PDE) och Expeditor Toolkit

Liksom vid övrig javautveckling används Eclipse som utvecklingsverktyg. För plugin-utveckling finns en specifik utgåva av Eclipse anpassad för detta ändamål kallad Eclipse PDE. Till detta adderas det så kallade IBM Expeditor Toolkit som innehåller bibliotek och inställningar för utveckling av Lotus Notes plugins i Eclipse PDE. Lotus Expeditor är det som förenar Eclipse med Lotus Notes. Det är ett ramverk för klientintegration som kan användas för att utöka funktionaliteten i en rad produkter som liksom Lotus Notes kommer från företaget IBM.

Traditionellt sett är ett plugin ett mindre program som kan ”pluggas in” i ett större för att utöka dess funktionalitet. Ofta kan flera plugins samverka för att tillsammans fungera som en större funktion eller egenskap. Vid utveckling av Java-plugins kallas en sådan samling av plugins för en feature. En feature kan innehålla en eller flera plugins som tillsammans tillför den funktionalitet denna feature beskriver och utlovar.



Figur 1 - Visar hur ett större plugin-projekt kan vara organiserat.

För att distribuera features och därmed plugins används idag Equinox P2 [1]. Det är en platform för Eclipse-baserade projekt som konkret gestaltar sig som så kallade ”update sites”. En *update-site* som det hädanefter kommer att kallas i denna rapport består av tre beskrivande XML-filer samt de plugins och de features den aktuella update-siten kan erbjuda. Figur 1 visar hur strukturen kan se ut när ett antal plugins finns tillgängliga i en update-site.

Att göra en release av ett plugin innebär i detta fall att bygga en update-site och därmed även bygga de plugins och features som ska finnas på denna. En update-site kan sparas och användas på två olika sätt. Antingen ligger dess filer fritt i en katalog och sökvägen till denna anges i programmet som ska hämta och installera plugins, eller så är hela denna site paketerad som en standard Jar-fil och sökvägen till denna anges istället i programmet.

2.3 Versionshantering

Versionshantering avser att bemöta problematiken som uppstår när flera utvecklare arbetar med och därför behöver göra ändringar i delade kodfiler. En utvecklares ändringar kan skriva över någon annans om filen använts av denne under tiden det tog att göra ändringen. Det är således viktigt att alla parter blir meddelade om det skett en förändring i en delad resurs.

Versionshanteringsverktyg löser detta genom att tillhandahålla en övervakad server för dessa filer. Vid varje uppdatering tvingas användaren att först hämta hem den senaste versionen av filerna från servern. Sedan sköts integrationen av de nya ändringarna lokalt och automatiskt och den nya versionen av filerna kan sedan laddas upp. Skulle en överlappande ändring trots detta påträffas meddelas användaren och verktyget tillhandahåller hjälp för att lösa konflikten manuellt.

Tack vare denna lösning är det omöjligt för en utvecklare att av misstag lägga in uppdateringar som skriver över någon annans. Versionshanteringsverktyg erbjuder även andra fördelar:

- Kollaboration - Det är enkelt att dela kod och se vem som gjort vilka ändringar;
- Backup – En revisionshanteringsserver kan innehålla flera uppsättningar av samma fil i äldre versioner. Skulle servern av någon anledning krascha är dessutom sannolikheten stor att någon användare har en lokal kopia sparad.
- Integration – Användare uppmuntras till att utföra frekventa uppdateringar vilket gör att alla så gott som alltid arbetar med den senaste versionen av koden.

Sandvik använder sig av versionshanteringsverktyget Subversion [2] samt dess tillhörande Eclipse-plugin Subversive[3].

2.4 Continuous integration

Continuous integration[4], direktöversatt till svenska som *kontinuerlig integration*, är ett uttryck som föddes ur den agila metoden Extreme Programming i slutet av 90-talet. Denna fritt översatt som *Extrem Programmering* bygger bland annat på täta återkopplingar och små delleveranser där grundläggande delar är prioriterade. Det är således viktigt att alla uppdateringar integreras i samma takt som de utvecklas, samt att denna integrationsprocess är överskådlig för alla och uppföljs av motsvarande enhetstester.

Det är detta som verktyg för Continuous Integration underlättar. Tack vare automatiska byggprocesser som startas vid bestämda tidpunkter eller vid uppladdning till projektets versionshanteringsserver, kan de tester som ingår i byggprocessen fort finna felgenererande kod. Därmed kan denna korrigeras direkt medan utvecklaren fortfarande har problemet kvar i sitt minne, samt innan felet hinner generera följdfel. För att organisera dessa automatiska byggen och övervaka pågående projekt använder sig SITS av verktyget Hudson[5].

2.5 Automatic Build

Fördelarna med att ha en automatisk, generell och central byggprocess för alla projekt som är oberoende av utvecklingsverktyget är många. Att automatisera bygg- och test-processen sparar inte bara tid för programmeraren utan medför även att risken för felkonfiguration minskar. Tid sparas även då referensbibliotek kan lagras på en central server istället för att på varje enskild dator behöva installera dessa artefakter manuellt. Projekt som passerar alla steg i byggprocessen utan fel kan sedan själva laddas upp till denna server för att sedan vara tillgängliga för andra projekt och så vidare. Ett verktyg för automatiska byggprocesser är Apache Maven[6], SITS använder sig av version 2.x av verktyget och ett bygge startas antingen manuellt via ett tillhörande Eclipse-plugin kallat m2eclipse[5][7] eller automatiskt av Hudson.

När Maven får till uppgift att bygga ett projekt med referenser till ett tidigare okänt plugin eller en artefakt söks detta i alla tillgängliga repositories[6], och om det hittas laddar Maven automatiskt ner det, bygger projektet, och lagrar sedan artefakten i sitt lokala repository för framtida byggen.

2.6 Fördjupning av byggprocess för Maven

Maven använder sig av något kallat Build Lifecycle[8], direktöversatt som *livscykel för ett bygge*. Det är en samling klart definierade *faser* eller *mål* som ett projekt genomgår när det byggs av Maven. Varje fas är ett fristående plugin och kan bytas ut för att passa den aktuella byggnationen. En standard Maven build Lifecycle definieras som:

- process-resources
- compile
- process-test-resources
- test-compile
- test
- package
- install
- deploy

Varje fas beror på den tidigare och kommer, om den körs, automatiskt kräva att ovanstående fas körs först. Således kan hela byggprocessen genomföras genom att specificera deploy som mål. Deploy innebär att den så kallade artefakten, Jar-filen som innehåller det färdigbyggda projektet, läggs in i ett för ändamålet specificerat repository. Detta innebär att den nu är tillgänglig att fritt refereras till av alla andra Maven-installationer inom nätverket som också har tillgång till detta repository.

---

[5] Maven to Eclipse

[6] Repositories är pluralis av det engelska ordet Repository som förklaras i avsnitt 2.8.

2.7 Maven och POM[7]-filer

POM-filer [9] utgör en central funktion för Maven. Det är i dessa filer allt som Maven behöver veta om det aktuella projektet specificeras, vilka beroenden som finns, vilka Maven-plugins som ska användas, vilken gruppidentitet projektet har med mera. Ett exempel på hur en POM fil kan se ut visas i Figur 2. Även information om tillgängliga repositories som normalt sett lagras i konfigurationsfilen för Maven kan explicit specificeras i POM filen. Och även om POM filer generellt sett existerar per projekt finns även så kallade ”förälder-POM”-filer som innehåller information delad av många ”barn”-projekt.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                      http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>johan.exjobb.test</groupId>
  <artifactId>testprojekt_2</artifactId>
  <version>1.0</version>
  ...
  <dependencies>
    <dependency>
      <groupId>johan.exjobb.test</groupId>
      <artifactId>testprojekt_1</artifactId>
      <version>1.0</version>
      <type>jar</type>
      <scope>test</scope>
      <optional>true</optional>
    </dependency>
    ...
  </dependencies>
  ...
</project>
```

Figur 2 - Visar grundläggande delar i en Maven POM fil för projektet testprojekt_2 som har ett beroende till testprojekt_1.

2.8 Artifact Repository

Maven behöver en lagringsplats för färdigbyggda artefakter samt för att lagra artefakter från andra källor, närmare bestämt ett så kallat repository. Detta är vad verktyget Nexus[10] tillhandahåller. Förutom att fungera som lagringsplats för Jar-filer erbjuder Nexus gedigna säkerhetsinställningar, bland annat möjligheten att begränsa importer från Internet, vilket är uppskattat då detta endast bör skötas av administrationspersonal.

2.9 Release, Deploy och Distribution.

När en applikation anses vara tillräckligt färdig för att en första utgåva ska ges ut är den redo för *release*. I detta steg byggs och slutpaketeras applikationen till en exekverbar fil eller artefakt som den ofta kallas, vilket för Java är av typen Jar. Denna tilldelas slutligen ett versionsnummer och grupptillhörighet.

---

[7] POM är en förkortning av Projekt Object Model, direktöversatt till svenska som *objektmodell för projekt.*

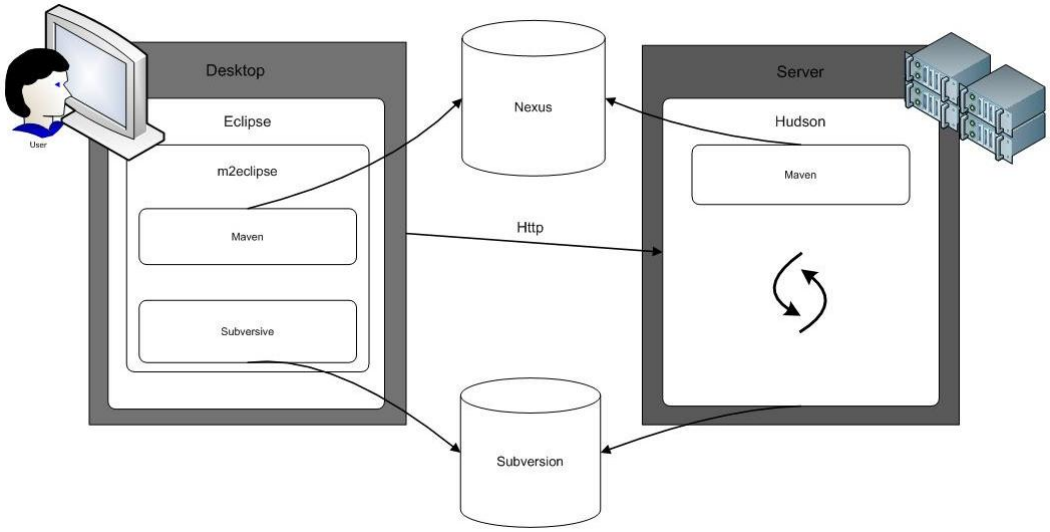
Steget *deploy* avser sedan processen att flytta denna artefakt till ett repository eller annan central plats där den kan indexeras och nås av andra personer.

Slutligen avser begreppet *distribution* det sätt på vilket en applikation kan hämtas och installeras av en slutanvändare.

Utvecklingsmiljöns verktyg kan avlasta arbetet med dessa delar av utvecklingen genom att automatisera delar av processen, eller tillhandahålla mallar och strukturer som underlättar hanteringen.

2.10 Sammanfattning av för undersökningen relevanta delar av utvecklingsmiljön för Java

Sammanfattningsvis används Eclipse som programmeringsverktyg och versionshantering sköts av subversion via Eclipse-pluginet subversive. Projekt byggs med Maven via Eclipse-pluginet kallat m2eclipse, eller med Maven via Hudson automatiskt när de laddas upp till en versionshanteringsserver. När Maven bygger projekt hämtas nödvändiga artefakter från Nexus repositories, och vid deploy läggs de in där. Hudson och Nexus konfigureras genom webbgränssnitt medans Mavens inställningar ändras manuellt i en lokal konfigurationsfil. En överblick av detta finns i Figur 3.



Figur 3 - Visar en överblick av relevanta delar av utvecklingsmiljön för Java på SITS.

Teoretisk Ansats

Då Lotus Notes 8.5 är byggt på Eclipse och plugin-utveckling sker på identiskt sätt, endast med skillnader i referensbibliotek, antas att en lösning för release, deploy och distribution för plugins till Eclipse även kommer att passa för plugins till Lotus Notes.

3 Metod

Efter intervjuer på företaget står det klart att uppgiften består av två distinkta problemdelar.

- Hur kan vi bygga och hantera plugins på ett automatiskt och generellt sätt med delade referensbibliotek/artefakter vid release och deploy.

- Hur kan plugins distribueras till slutanvändaren och kan installationen automatiseras eller på annat sätt förenklas?

Beslut tas att prioritera den första problemdelen av de två både i tid och arbetsföljd. Detta med motivering att den senare kan komma att bero av den tidigare och det därmed är logiskt att de behandlas i samma följd, samt att företaget anser denna vara viktigast.

Det framgår vid ytterligare kommunikation med anställda på företaget att det skulle vara fördelaktigt om lösningen kunde integreras i den befintliga utvecklingsmiljön för Java. Motiveringen till detta är att det anses krångligt att ha någon typ av separat system just för hantering av plugins. Installation och vana hos utvecklare som ska jobba med plugins minimeras om befintliga välkända system kan användas.

3.1 Metodbeskrivning

Med bakgrund av detta skapas en logisk karta över de frågeställningar som måste besvaras, och hur dessa beror av varandra och därmed i vilken ordning de måste behandlas.

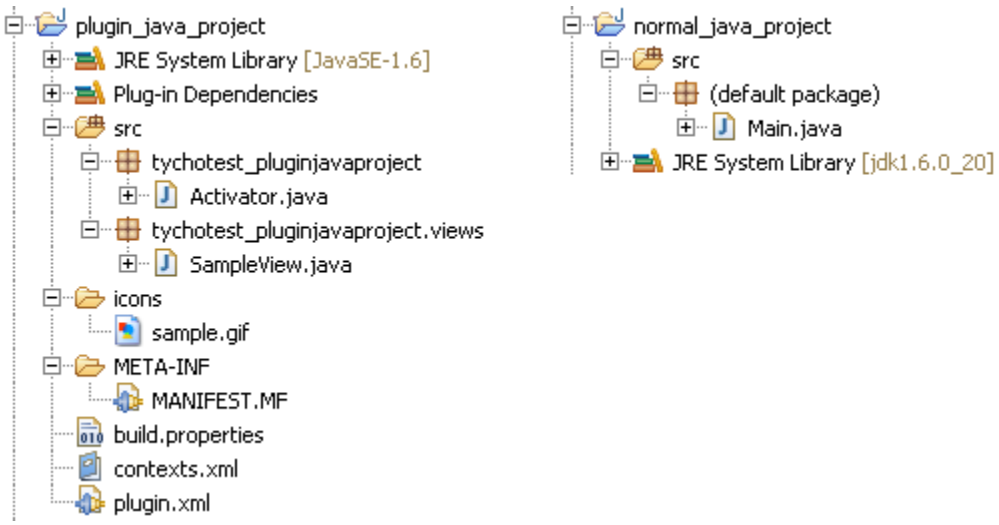
- Hur kan en utvecklingsmiljö se ut som stödjer release, deploy och distribution för plugin-utveckling?
    - Kan den befintliga miljön för release och deploy anpassas och existerar alternativa verktyg som kan erbjuda en bättre lösning?
        - Vad innebär det att integrera stöd för plugin-utveckling i den befintliga miljön?
            - Vilka delar är det som inte är kompatibla i nuvarande system?
                - Vad är skillnaden mellan ett vanligt javaprojekt och ett plugin-projekt?
        - Vilka andra verktyg finns tillgängliga för denna typ av utveckling?
    - Hur kan en distribution av plugins se ut som fungerar med vald lösning för release och deploy?

Tillvägagångssättet för att besvara dessa frågor är efterforskningar i litteratur och communities inom ämnet såväl som intervjuer med kunnig personal på företaget.

4 Genomförande

I detta avsnitt behandlas de frågeställningar från avsnitt 3.1 som utgör metoden för denna studie. De besvaras i hierarkiskt stigande grad och varje frågas svar utgör därmed en grund för svaret på nästkommande fråga.

4.1 Vad är skillnaden mellan ett vanligt javaprojekt och ett plugin-projekt?



Figur 4. Ett plugin projekt respektive ett vanligt Java SE projekt utforskat i Eclipse PDE Package Explorer.

Ett plugin-projekt är i grunden lika som ett vanligt Java-projekt men innehåller dessutom en rad andra filer, bland annat beskrivande XML-dokument där pluginets version med mera sparas. Dessa filer och skillnaden i struktur mot ett vanligt Java-projekt kan urskiljas i Figur 4. Förenklat skulle man kunna säga att ett plugin-projekt innehåller fler beroenden än ett vanligt projekt, och att dessa specificeras på ett speciellt sätt.

4.2 Vilka delar är det som inte är kompatibla i nuvarande system?

Detta avsnitt besvarar hur skillnaden mellan ett plugin-projekt och ett vanligt projekt påverkar de olika verktygen som ingår i utvecklingsmiljön, närmare bestämt vilket eller vilka av verktygen som är inkompatibla med plugin-utveckling och vad som är orsaken till detta.

Eclipse

Då Eclipse är utvecklingsverktyget som används för att skriva plugin-koden är det självklart fullt kompatibelt med plugin-utveckling. Däremot kan det uppstå konflikter mellan eventuella stöd-plugins som måste installeras i själva Eclipse.

Subversive

Subversive arbetar mot en Subversion-server och hanterar bara filer. Alla filtyper i klartext och strukturer stöds. Därmed är plugin-utveckling fullt kompatibelt med subversive.

m2eclipse

M2eclipse bygger POM-filer och strukturerar upp projekt enligt Mavens standard. Då POM-filerna skapas utifrån javaprojektets biblioteksberoende förefaller det som att det är här det blir problem. M2Eclipse vet inte hur det ska förbereda ett plugin-projekt för Maven.

Hudson

Hudsons interaktion med koden består endast i att schemalägga Maven att bygga denna. Att plugins har en annan struktur påverkar således inte Hudson direkt, och därmed är densamme relativt ointressant för detta problemsteg så länge dess tillhörande Maven-installation stöder plugins.

Nexus

Då nexus endast är ett så kallat artifact repository som lagrar JAR-filer är detta inte strukturmässigt berört av skillnaden mellan plugins och vanliga projekt. Däremot kan avsaknaden av plugin-specifika artefakter medföra att Maven inte kan bygga dessa. Därmed är Nexus intressant.

Maven

Maven kan i princip bygga vad som helst så länge det finns passande instruktioner för byggcykeln, en POM-fil som beskriver projektets beroenden korrekt samt att nödvändiga artefakter finns tillgängliga i något repository. Dock är detta inte fallet för plugins, Maven vet inte hur pluginets beroenden ska tolkas och kan därför inte genomföra ett lyckat bygge.

Sammanfattning

Det förefaller vara Maven och m2eclipse som inte är anpassat för hantering av plugins. Orsaken till detta är att dessa två inte kan läsa de beroendereferenser ett plugin-projekt har och inte heller känna igen dess katalogstruktur.

4.3 Vad innebär det att integrera stöd för plugin-utveckling i den befintliga miljön?

Att få den befintliga miljön att stödja plugin-utveckling är givetvis att anpassa de delar som för tillfället inte är kompatibla med plugins till att bli det. Då problemet förefaller vara Mavens del i processen är det således där anpassningar måste ske. Detta leder till frågeställningen vad det innebär att anpassa Maven, vilka problem som konkret måste lösas. För att besvara denna fråga eftersöks information om ämnet på Eclipse Community[11].

Enligt Petersen och Gupta [12] finns ett antal problemområden som kräver åtgärder:

Maven måste ha tillgång till målplattformen

Vid plugin-utveckling kräver Eclipse att en så kallad ”target platform” eller ”målplattform” specificeras. Detta är den katalog där alla plugin-specifika bibliotek finns. På samma sätt som man anger den lokala installationskatalogen för Eclipse vid utveckling av Eclipse-plugins, anger man för plugins till Lotus Notes dess bibliotekskatalog eller ”framework” som den kallas. För att Maven ska kunna hitta de bibliotek eller ”artefakter” som krävs för att bygga ett plugin måste således dessa filer tillgängliggöras på något sätt.

Eclipse plugin har beroenden specificerade i manifest-filen

Eclipse-plugins måste ha sina beroenden angivna i en så kallad manifest-fil. Dessa beroenden måste även Maven känna till för att kunna lyckas med bygget.

Eclipse-plugins måste paketeras på ett speciellt sätt

Katalogstrukturen för ett plugin skiljer sig från den Maven förväntar sig. Vid utveckling av ett vanligt Java-projekt kommer m2eclipse ta hänsyn till detta och därför ändra katalogstrukturen, men detta kommer inte ske på ett korrekt sätt vid plugin-utveckling.

Förutom dessa tre problem behandlar artikeln även ytterligare två problempunkter som berör testning samt hämtning av befintliga Maven-projekt. Samtidigt som artikeln dessutom beskriver hur alla dessa problem kan lösas, visar den även prov på komplexiteten i problemet samtidigt som dess existens bekräftar att efterforskningar gjorts i ämnet.

Med bakgrund av ovanstående dras följande slutsatser:

- Problemet är för komplext för att en egenutvecklad lösning ska kunna produceras under den tid projektet är begränsat till.

- Då problemet tidigare belysts och minst en lösning existerar, även om den numera är utdaterad (2006), är det troligt att mer moderna verktyg finns tillgängliga.

Ytterligare efterforskningar i olika communities på Internet bekräftar att det finns ytterligare lösningar inom detta område i form av plugins eller så kallade Mojos till Maven [13]. Ett plugin kan innehålla en samling Mojos vilka är implementationer av enskilda byggmål i en Maven-byggcykel.

Även om det förefaller finnas mer än ett alternativ inräknat den tidigare refererade artikeln där ett antal Mojos skapades, så är det ett flertal communities som förespråkar Maven-pluginet Tycho. Tycho-projektet lämnade in en ansökan[14] om att bli ett Eclipse-projekt under Eclipse Technology Project[8] cirka 2 veckor innan denna studie påbörjades.

Även om denna ansökan fortfarande inte behandlats stödjer den bilden av Tycho som ges ute i communities, nämligen att det är ett högst seriöst projekt med många intressenter. Dock är Tycho fortfarande i betastadiet och i skrivande stund är 0.8 senaste versionen. Den första officiella versionen 1.0 väntas enligt projektförslaget komma någon gång under tredje kvartalet 2010.

[8] Eclipse Technology Projekt är avsett att fungera som en inkubator eller teknologisk undersökningsplattform för mindre projekt. Ofta leder dessa projekt till en vetenskaplig artikel eller att de integreras i andra huvudprojekt för Eclipse. [15]

4.4 Vilka andra verktyg finns tillgängliga för denna typ av utveckling?

Förutom andra plugins till Maven, liknande Tycho, som antingen är utdaterade eller erbjuder mindre funktionalitet, finns en fåtal liknande projekt som ofta nämns i samma mening som Tycho.

Buckminster [16] är ett alternativ till Maven och har liknande funktionalitet även om det används i mindre utsträckning. Det finns ett tillhörande Eclipse-plugin och möjligheter att utöka detta för plugin-utveckling även om det kräver mycket manuell konfiguration. En fördel är att Buckminster liksom Maven kan användas tillsammans med Hudson.

Athena [17] är inget Eclipse-plugin men en samling av verktyg och skript som används med Eclipse för att utöka PDE-delen, fungerar med Hudson och planeras ha stöd för Maven samt Buckminister. Dock är även detta under utveckling.

B3[18] strävar efter att ersätta Eclipse PDE i framtiden och erbjuda ett bättre sätt att utveckla plugins. Målet är att ta det bästa från Buckminister och PDE och kombinera dessa. B3 anger Athena som relaterat projekt och det spekuleras om ett framtida Maven-stöd.

Plugin Builder [19] klarar tillsammans med Cruise Control [20] automatiska byggen. Cruise Control erbjuder funktioner liknande de hos Hudson. Oklart om det även fungerar med Hudson. Dock tycks inte Plugin Builder stödja kontinuerlig integration.

4.5 Bör den befintliga miljön anpassas eller existerar alternativa verktyg som kan erbjuda en bättre lösning?

För att bestämma vilken lösning som är den mest fördelaktiga utvärderas dessa utifrån kriterierna;

- Teknisk potential
- Långsiktighet
- Anpassning till nuvarande system
- Möjlighet till distributionslösningar

Efterforskning inom Communitys för problemområdet samt kontakt med inom ämnet kunniga personer leder till slutsatsen att Tycho är det bästa alternativet.

Detta motiveras genom att Tycho är ett Maven-plugin och således kan användas med de befintliga verktygen som redan finns på plats och personalen är vana vid. Tycho ska enligt uppgifter från olika communities vara enklare och mindre krångligt att konfigurera. Tack vare att Tycho är en del av Maven, som är mycket omtyckt samt det goda stöd det får i diverse artiklar och Communities, anses användandet av detta plugin med Maven vara den hållbaraste lösningen för framtiden.

4.6 Hur kan en distribution av plugins se ut som fungerar med vald lösning för release och deploy?

Det finns flera alternativ för distributionen av plugins till Lotus Notes. I detta avsnitt lämnas förslag på olika lösningar och dess eventuella för- och nackdelar. För vissa lösningar kan det vara relevant hur byggda artefakter sparas. Då Tycho föreslås för att få release och deploy kompatibel med plugin-utveckling kommer byggda artefakter lagras i ett Nexus repository. Detta kan nås via en webbläsare eller annan programvara som klarar att göra http-förfrågningar.

Sandvik Advertised Program

Sandvik använder idag ett program kallat *Run Advertised Program* för mjukvaruinstallationer över nätverket. Det tillhandahåller en databas av programvaror tillgängliga och kan starta installationsprocesser för dessa. Det är dessutom möjligt att via detta system schemalägga och annonsera installationer för användarna. Det är teoretiskt möjligt att detta system även skulle kunna användas för distribution av plugins. Dock skulle installationsprogram behöva skapas för dessa plugins, då de till skillnad från vanliga program saknar detta. Installationsprocessen skulle kunna vara i enkel form då en plugin kan installeras genom att kopiera dess filer till katalogen avsedd för plugins i Lotus Notes.

Lotus Notes databas

IBM som äger Lotus Notes/Domino föreslår att man kan distribuera plugins genom att bygga en speciell databas för Lotus Notes kallad Widget Catalog [21]. I denna databas kan användare bläddra bland plugins och enkelt installera dessa manuellt. Plugins finns här representerade av XML-filer som förutom att beskriva pluginet innehåller en länk till dess update-site. Denna länk skulle då direkt kunna peka på pluginets sökväg i det Nexus Repository som det flyttats till av Maven vid deploy-fasen.

Lotus Notes policy

Genom att använda så kallade policys kan Lotus Domino-administratörer skicka ut inställningar till alla de Lotus Notes-klienter som ingår i organisationen eller enbart till bestämda grupper. Policys kan även användas för att automatiskt installera en plugin från en Widget Catalog i valda klienter. På detta sätt kan man snabbt och relativt enkelt installera plugins till samtliga klienter även om lite manuellt arbete krävs med att sätt upp policyn, databasen samt de beskrivande XMLfilerna. Eventuellt kan dessa XML-filer automatiskt genereras genom någon typ av script eller manipulation av Mavens deploy-plugin och dess innehåll kan då hämtas från beskrivningen som finns i pluginets feature.

4.7 Hur kan en utvecklingsmiljö se ut som stödjer build, release och deploy för plugin-utveckling?

Här beskrivs arbetet med att sätta upp de verktyg som krävs för att anpassa utvecklingsmiljön till plugin-utveckling.

För att prova Tycho med plugin-utveckling för Lotus Notes krävs en Eclipse-version som både har stöd för plugin-utveckling samt m2eclipse för Maven-funktionalitet. Följande program installeras och konfigureras för detta ändamål:

Lotus Notes 8.5.1
Lotus Notes innehåller ett Eclipse-ramverk med de Notes-specifika bibliotek som Eclipse behöver vid plugin-utveckling. Detta ramverk används av Eclipse som ”Target Platform”.
Eclipse 3.4 SDK-PDE
Plugin: Lotus Expeditor Toolkit 6.2.1
Plugin: Sonatype m2eclipse 0.10.0.20100209-0800

Det visade sig dock vara mer problem än väntat med att få Eclipse och dess plugins att samverka harmoniskt. Anledningen till detta var främst att Eclipse i stor grad stötte på konflikter mellan befintliga plugins och de plugins som skulle installeras, samt att företagets www-proxy begränsade möjligheterna att kontakta Internet och externa update-sites där stödbibliotek kunde nås. För att lösa dessa problem användes Eclipse 3.4 istället för 3.5 samt att verktygen flyttades ut ur den virtuella maskin de tidigare använts i vid plugin-utveckling.

För att testa Tycho och verifiera att det tillåter oss att bygga update-sites korrekt och därmed plugins lades en plan upp;

1. Bygga ett Eclipse plugin med Eclipse PDE.
2. Bygga ett Notes plugin update-site med Eclipse PDE
3. Använda m2eclipse för att bygga ett projekt i Eclipse PDE
4. Använda Tycho (Maven) för att bygga en plugin update-site .
5. Använda Tycho (Maven) för att bygga en Lotus Notes plugin update-site.
6. Använda Tycho med m2eclipse i Eclipse PDE för att bygga en Lotus Notes update-site

Steg 1-3 utfördes med hjälp av guider och dokumentation på Internet och relativt få fel uppstod.

Steg 4 erbjöd däremot en rad motgångar, främst med användandet av Tycho och därmed Maven.

För att använda Tycho i dagsläget måste det ske manuellt i en kommandotolk och med en lokal Maven-installation som stöd, till skillnad från m2eclipse som inte kräver någon installation av Maven utan istället använder en egen inbyggd Maven-distribution. Tycho som är paketerat i form av en artefakt i ett repository hämtas automatiskt av Maven om det anges som ett nödvändigt plugin vid byggnationen.

Detta skiljer sig emellertid från det sätt man var tvungen att använda äldre versioner av Tycho. Det var då inte paketerat som en artefakt utan integrerat i en befintlig Maven-distribution. För att bruka Tycho krävdes att man övergav sin befintliga Maven-installation och istället använde denna Tycho-version av Maven.

Förvirring uppstod när dessa två olika utseenden av Tycho påträffades, dels den gamla i form av en komplett Maven-distribution och dels den nyare i form av endast en artefakt. Samtidigt har Tycho-projektet en stark relation till m2eclipse-projektet, och dess två verktyg i framtiden förväntas fungera tillsammans [22] .

Upptäckten av två guider[23][24] som beskrev hur Tycho i form av en artefakt kunde användas av Maven, samt det faktum att det var den senaste versionen tillgänglig, ledde tillsammans till slutsatsen att det var denna version av Tycho som borde användas.

Dessa två guider förutsatte emellertid att den lokala Maven-installationen hade kontakt med det så kallade ”Central Repository”[25]. Denna, direktöversatt till svenska som *den centrala förvaringsplatsen*, är ett repository som ägs av utvecklarna av Maven. Det innehåller plugins och andra stödartefakter för Maven och följaktligen den senaste exekverbara versionen av Tycho.

På företaget var Maven emellertid inte standardkonfigurerat, utan hade endast interna Sandvik-specifika repositories tillgängliga. Detta ledde till att Maven ej kunde hitta och därmed använda Tycho, istället gjordes försök att manuellt installera Tycho i Mavens lokala repository utan framgång, dels eftersom att en färdigbyggd Jar version av Tycho inte kunde laddas ner på någon funnen webbsida bundna till projektet, samt att det vid denna tidpunkt var okänt att filer manuellt kunde hämtas från repositories via webbläsaren.

Försök genomfördes att bygga Tycho från källkods-filer men även dessa misslyckades på grund av att stödartefakter inte kunde hämtas från Central Repository. Av samma anledning misslyckades installationen av Tycho-artefakten i Mavens lokala repository när det väl blev känt att den kunde laddas ner från Central Repository.

Problemet bestod alltså i att Maven behövde omkonfigureras för att få tillgång till fler repositories, och okunskap om att detta var orsaken till problemen. För att lösa detta kontaktades en anställd på företaget med erfarenhet av Maven och de Sandvikspecifika inställningarna byttes ut mot en standardkonfiguration, dock med nödvändiga inställningar för Proxy och därmed Internetanslutning.

När dessa problem löstes genomfördes steg 4, dvs. ett lyckat bygge av ett Eclipse-plugin med Tycho. Projektet skapades och sattes upp i Eclipse PDE precis som vid tidigare utveckling och sedan användes Tycho i en kommandotolk för att generera dess POM-filer och sedan bygga det. Detta plugin testades genom installation i Eclipse och testkördes utan problem.

Steg 5 genomfördes sedan och ett fungerade plugin för Lotus Notes byggdes med Tycho. Pluginet testades genom installation i Lotus Notes och betedde sig enligt förväntan. En mer detaljerad genomgång av hela bygget i detta steg finns i bilaga 1.

Steg 6 följde sedan. Ett plugin projekt med tillhörande feature och update-site skapades i Eclipse PDE. Sedan användes Tycho för att generera POM-filer för projektet. Slutligen skapades en anpassad m2eclipse byggkonfiguration som använde sig av dessa POM-filer och bygget av plugin-projektet kunde genomföras med stöd av Tycho av m2eclipse i Eclipse PDE.

5 Resultat

Det går att anpassa Java-utvecklingsmiljön på företaget till att stödja plugin-utveckling. Dessutom kan detta realiseras genom påbyggnad av de befintliga verktygen som utgör utvecklingsmiljön och därmed behöver inga nya fristående programvaror installeras. Sammanfattningsvis har alla frågeställningar besvarats samt har den teoretiska ansatsen bevisats vara korrekt.

5.1 Valet av verktyg

Det finns ett flertal lösningar som erbjuder möjligheter för kontinuerlig integration samt automatiska byggprocesser för plugin-utveckling. Dessa har bedömts utifrån följande aspekter:

- Teknisk potential
- Hållbarhet i tid
- Anpassningsbarhet till nuvarande system
- Möjlighet till distributionslösningar

Av de lösningar som granskats har Maven-pluginet Tycho förefallit som det bästa alternativet. Studien har sedan visat hur Tycho kan användas för plugin-utveckling av Eclipse-plugins samt bekräftar att detta även gäller för plugins för Lotus Notes.

Tack vare att Tycho är ett plugin till Maven kan plugin-utveckling ske med befintliga verktyg för automatiska byggen samt kontinuerlig integration. Fördelen med detta är att inga nya programvaror måste tillföras som man måste lära personal att hantera, samt att endast ett verktyg för byggprocessen krävs istället för två.

5.2 Begränsningar i den aktuella versionen av Tycho

Tycho är i skrivande stund fortfarande under utveckling och därmed är viss funktionalitet begränsad. Tycho väntas slutföra sin betafas och därmed utkomma i en färdig första version under tredje kvartalet 2010. När detta inträffar förväntas dessa begränsningar vara upplösta.

Paketering av Update-Site

Ett problem som för tillfället begränsar lösningen är en bugg i den version av Tycho som använts i genomförandet. Normalt innebär steget deploy i en Maven standard lifecycle att koden paketeras till en artefakt och sedan lagras i för ändamålet avsett repository. När detta steg körs av Tycho med en update-site som mål för bygget paketeras denna ej korrekt. Denna bugg[26] har anmälts till Tycho’s bugghanteringssystem och ska enligt uppgift där vara korrigerad i nästkommande version av Tycho.

Interaktion mellan Tycho och m2eclipse

Tycho har starka kopplingar till m2eclipse-projektet och dessa två planeras när Tycho är färdigt helt fungera tillsammans. För tillfället är detta inte fallet, även om m2eclipse kan instruera Maven att bygga ett projekt som fått sina POM-filer genererade av Tycho, och detta bygge kan startas på lika sätt som byggen av vanliga Java-applikationer. Ändå är det en begränsning att inte m2eclipse automatiskt kan identifiera ett plugin-projekt och initiera Tycho för att generera POM-filer. Detta måste ske manuellt via kommandorad och en separat installation av Maven 3 krävs följaktligen.

6 Diskussion

6.1 Av resultatet

Studien har visat hur en utvecklingsmiljö för plugins till Lotus Notes kan se ut och därmed uppfyllt sitt grundläggande mål. Först har problemet brutits ned och orsaken till dess uppkomst har identifierats. För att sedan finna en lösning har ett antal alternativ studerats och efter utvärdering av dess specifikationer har den bäst passande valts. Denna har sedan testats genom implementation av en prototyplösning.

Studien har konkret visat hur man kan anpassa utvecklingsmiljön för Java till att stödja plugin-utveckling till Lotus Notes genom att använda Tycho. Det är tveksamt om lösningen är direkt applicerbar i dagens läge, då Tycho inte utkommit i en officiell version och därmed ännu inte kan användas utan förarbete med konfiguration samt manuell hantering. Emellertid har studien visat varför Tycho är det bästa alternativet för SITS. Detta i samband med prognosen för Tycho-projektet med en skarp release vid tredje kvartalet 2010 gör sammantaget att inget annat alternativ bedöms attraktivare.

Möjligheterna till distribution har undersökts, dock inte lika grundligt som release- och deploy-delen på grund av den prioritering som beskrivs i avsnitt 3. Utforskningen av detta område kunde varit grundligare om det legat inom tidsramen för studien och möjligheter till ytterligare automatisering bedöms existera.

6.2 Av metoden

För att besvara studiens frågeställningar gjordes efterforskningar i elektroniska dokument och intervjuer på företaget. Dessa ledde fram till att en metod för att lösa den övergripande tekniska frågeställningen kunde tas fram, nämligen hur en utvecklingsmiljö för plugin-utveckling kan se ut. Metoden bestod av en logisk *karta* med ett antal frågeställningar ordnade i ett hierarkiskt system. Dessa frågeställningar var avsedda att behandlas nedifrån och upp i hierarkin, och om alla underordnade frågeställningar besvarats var teorin att den övergripande frågan likväl kunde besvaras.

Metoden följdes och visades vara sund. Efter att ha besvarat alla frågor i kartan kunde ett resultat i förväntad form erhållas.

6.3 Av genomförandet

Upplägget för genomförandet har visat sig vara lyckat. Förväntat resultat har uppnåtts. Dock har mycket tid lagts på inlärning av begrepp och involverade verktyg. De större problemsituationer som har uppstått har delvis berott på okunskap. Detta kan emellertid ses som naturligt då området är svårpenetrerat för nybörjare. Dålig dokumentation och avsaknaden av officiella guider har varit ett stort problem, i synnerhet när det gäller Tycho. Ofta har lösningar på problemsituationer eller guider återfunnits på privata webbplatser eller inom närliggande communities.

7 Slutsatser

I detta avsnitt besvaras studiens frågeställningar kortfattat i form av slutsatser som studiens resultat lett fram till.

Det går att sätta upp en utvecklingsmiljö för plugins till Lotus Notes som har stöd för release, deploy och distribution. Genom att använda ett Maven-plugin med namnet Tycho är det möjligt att anpassa utvecklingsmiljön för Java SE applikationer till att även hantera plugin-applikationer. Tack vare detta kan företagets befintliga verktyg för automatiska byggprocesser samt kontinuerlig integration utnyttjas.

Skillnaden mellan ett vanlig Java SE-projekt och ett plugin-projekt är att det senare har referenser specificerade på ett annorlunda sätt samt att extra resursfiler existerar. Detta medför att m2eclipse och därmed Maven som används vid utveckling av Java SE applikationer inte utan hjälp kan bygga ett plugin-projekt korrekt.

En utvecklingsmiljö som stödjer utveckling av plugins till Eclipse stödjer även utveckling av plugins till Lotus Notes. Den enda skillnaden är att andra referensbibliotek måste användas vid bygge av koden.

Studien har visar hur relevanta delar av utvecklingsmiljön för Java-applikationer på SITS ser ut, samt hur dessa verktyg samverkar och kan erbjuda en lösning för release, deploy och distribution.

Problemområdet för kontinuerlig integration och automatiska byggen av plugins har utforskats tidigare och det existerar vissa lösningar. Dock förefaller detta vara första gången det görs med plugins för Lotus Notes i fokus.

Vid större omfattande programvaruutveckling anses det idag vara ett måste med någon typ av lösning för versionshantering, automatiska byggprocesser, kontinuerlig integration och automatiska enhetstester. När marknaden nu även växer för utveckling av plugins är det naturligt att samma funktionaliteter efterlyses där. Det är vidare fördelaktigt om samma verktyg som används till den vanliga utvecklingen även kan användas för plugin-utveckling.

Det finns ett antal lösningar för hur distributionen av plugins till Lotus Notes kan skötas. Lösningen för utvecklingsmiljön ger nödvändigtvis inte något direkt stöd för dessa men begränsar heller inte möjligheterna.

8 Tack

Jag skulle vilja tacka mina två handledare Erik Norfelt på SITS och Jonas Boustedt vid Högskolan i Gävle för det stöd de tillsammans gett mig, Daniel Nilsson på SITS för det han lärt mig om utvecklingsmiljön och den hjälp jag fått med Maven. Ett speciellt tack riktar jag till Mattias Holmqvist som varit ett ovärderligt bollplank för mig vid problem med Tycho.

Sist men främst tackar jag mina nära och kära för den energi och det stöd de givit mig under perioden för detta examensarbete.

9 Referenser

[1] The Eclipse Foundation. ***'Equinox p2'***, *(2010)* <http://eclipse.org/equinox/p2/> (2010-5-25).

[2] The Apache Software Foundation. ***'Apache Subversion'***, *(*2010)<http://subversion.apache.org/> (2010-5-25).

[3] The Eclipse Foundation. ***'Subversive - SVN Team Provider'***, *(*2010)<http://www.eclipse.org/subversive/> (2010-5-25).

[4] Fowler. ***'Continuous Integration'***, (2006)<http://martinfowler.com/articles/continuousIntegration.html> (2010-5-25).

[5] Sun Microsystems,Inc. ***'what is Hudson?'***, *(*2009)<http://wiki.hudson-ci.org/display/HUDSON/Meet+Hudson> (2010-5-25).

[6] The Apache Software Foundation. ***'Welcome to Apache Maven '***, *(*2010)<http://maven.apache.org/> (2010-5-25).

[7] Sonatype. ***'M2eclipse: Project Home'***, *(*2010)<http://m2eclipse.sonatype.org/> (2010-5-25).

[8] The Apache Software Foundation. ***'Introduction to the Build Lifecycle'***, *(*2010)<http://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html> (2010-5-25).

[9] The Apache Software Foundation. ***'POM Reference'***, *(*2010)<http://maven.apache.org/pom.html> (2010-5-25).

[10] Sonatype,Inc. ***'Nexus | the Repository Manager'***, *(*2010)<http://nexus.sonatype.org/> (2010-5-25).

[11] ***'Eclipse Community'***, *(2010)* <http://www.eclipse.org/community/> (2010-5-25).

[12] Peter H. Petersen, Princeton Softech, Inc, and Sumit Gupta, Princeton Softech, Inc. ***'Eclipse Corner Article: Building Eclipse Plugin with Maven 2'***, *(*2006)<http://www.eclipse.org/articles/article.php?file=Article-Eclipse-and-Maven2/index.html> (2010-5-25).

[13] The Apache Software Foundation. ***'Maven - Introduction to Maven 2.0 Plugin Development'***, *(*2010)<http://maven.apache.org/guides/introduction/introduction-to-plugins.html> (2010-5-25).

[14] The Tycho Project. ***'Tycho - Eclipse Technology Project Proposal'***, (2010)<http://www.eclipse.org/proposals/tycho/> (2010-5-25).

[15] The Eclipse Foundation. ***'Eclipse Technology Project'***, *(*2010)<http://www.eclipse.org/projects/project_summary.php?projectid=technology> (2010-5-25).

[16] The Eclipse Foundation. ***'Eclipse Buckminster Project'***, *(*2010)<http://www.eclipse.org/buckminster/> (2010-5-25).

[17] The Eclipse Foundation. ***'Athena Common Build'***, *(*2010)<http://wiki.eclipse.org/Common_Build_Infrastructure> (2010-5-25).

[18] The b3 Project. ***'B3'***, *(*2010)<http://www.eclipse.org/modeling/emft/b3/> (2010-5-25).

[19] Markus Barchfeld. ***'Pluginbuilder - Build Automation for Eclipse Plug-Ins'***, *(*2008)<http://www.pluginbuilder.org/> (2010-5-25).

[20] CruiseControl Project. ***'CruiseControl'***, *(*2010)<http://cruisecontrol.sourceforge.net/> (2010-5-25).

[21] IBM Corporation. *'IBM Lotus Domino and Notes Information Center'*, 2008)<http://publib.boulder.ibm.com/infocenter/domhelp/v8r0/index.jsp> (2010-5-25).

[22] Jason van Zyl. ***'Tycho - Index'***, (2009)<https://docs.sonatype.org/display/TYCHO/Index> (2010-5-25).

[23] O'Brien. *'Building* ***Eclipse Plugins with Maven: Tycho'***, (2008)<http://www.sonatype.com/people/2008/11/building-eclipse-plugins-with-maven-tycho/> (2010-5-25).

[24] Mattias Holmqvist. ***'Building with Tycho – Part 2 (RCP Applications)'***, (2010)<http://mattiasholmqvist.se/2010/03/building-with-tycho-part-2-rcp-applications/> (2010-5-25).

[25] The Apache Software Foundation. ***'Maven - Introduction to Repositories',*** 2010)<http://maven.apache.org/guides/introduction/introduction-to-repositories.html> (2010-5-25).

[26] The Tycho Project. ***'Tycho Issue Tracking - Bug TYCHO-258'***, 2010)<https://issues.sonatype.org/browse/TYCHO-258> (2010-5-25).

10 Bilaga 1: Bygga plugin-projekt för Lotus Notes med Maven-Tycho

Denna bilaga beskriver i detalj hur ett lyckat bygge av ett plugin-projekt för Lotus Notes med Tycho-pluginet med Maven genomfördes.

Först installerades Eclipse 3.4, Lotus Notes 8.5.1, och Expeditor Toolkit 6.2.1. För att sedan kunna köra Tycho installerades även Maven 3-alpha-6. Tycho kräver en Maven 3 version, till skillnad från den Maven 2 version som för tillfället används på företaget (om man inte räknar med den inbäddade Maven 3 distributionen som m2eclipse innehåller).

Pluginet och dess tillhörande feature samt update-site skapades sedan i Eclipse PDE. En kommandotolk startades och arbetskatalogen ändrades till projektkatalogen för Eclipse, ofta kallad ”Workspace”. Sedan kördes kommandot i Figur 5 för att generera POM filer. Kommandot anger att det är Maven-pluginet Tycho som ska användas och att målet är att generera POM-filer för projekten. Dessutom angavs vilken grupptillhörighet projekten ska ha, samt vilken målplattformen som skulle användas för bygget. Lägg märke till att denna var specificerad för Lotus Notes, skulle det varit ett bygge av ett plugin för Eclipse hade målplattformen istället varit sökvägen till Eclipse.

Vid körning av kommandot kommer Maven automatiskt att söka efter Tycho i det lokala repositoriet. Skulle det inte finnas där, vidgas sökningen till alla tillgängliga repositories som kan nås via det interna nätverket eller internet, och skulle Tycho istället påträffas i något av dessa laddas det automatiskt hem och installeras i det lokala. För att göra Tycho tillgängligt måste antingen Tycho-projektets repository läggas till i listan över repositories Maven får arbeta med, eller så måste artefakten installeras Manuellt i ett av de centrala repositories som finns inom företagets nätverk. Det senare är att föredra för att inte vara beroende av resurser som företaget inte själva kan styra över. För denna prototyp valdes dock den tidigare lösningen då endast en dator behövde få Tycho-pluginet installerat i sitt lokala repository.

```
c:\javadev\bin\eclipse_from_VM\_WS>mvn3 org.sonatype.tycho:maven-tycho-plugin:ge
nerate-poms -DgroupId=x.test -Dtycho.targetPlatform="c:\Program Files\IBM\Lotus\
Notes\framework\rcp\eclipse"

[INFO] Scanning for projects...
Downloading: http://java.sandvik.com/nexus/content/groups/public/org/sonatype/tycho/maven-tycho-plugin/maven-metadata.xml
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building Maven Stub Project (No POM) 1
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- maven-tycho-plugin:0.8.0:generate-poms (default-cli) @ standalone-pom ---
[INFO] Scanning C:\javadev\bin\eclipse_from_VM\_WS basedir
[WARNING] Not a directory C:\javadev\bin\eclipse_from_VM\_WS\.tests
[WARNING] Could not determine bundle dependencies: Bundle x.test.plugin cannot be resolved
Resolution errors:

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 7.773s
[INFO] Finished at: Tue Jun 01 10:13:00 CEST 2010
[INFO] Final Memory: 3M/15M
[INFO] ------------------------------------------------------------------------
```

Figur 5 - Visar kommando och komplett logg för bygge av POM-filer med Tycho.

När kommandot i Figur 5 kördes skapades POM-filer för alla projekt i underliggande projektmappar samt en förälder-POM i projektkatalogen, innehållet i denna kan betraktas i Figur 6. I denna förälder-POM finns alla påträffade projekt i projektmappen deklarerade som moduler. De projekt man inte vill arbeta med kan raderas från deklarationen. I förälder-POM filen finns även deklarationen som anger att det är Tycho-pluginet som ska användas vid bygge av projekten. I denna prototyp gjordes inga ändringar i filen, det innebär att projektet plugin_java_projekt även kommer ingå i bygget även om det endast är x.test.plugin/feature/updatesite vi är intresserade av i detta prototyptest.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd" xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <modelVersion>4.0.0</modelVersion>
  <groupId>x.test</groupId>
  <artifactId>_WS</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>pom</packaging>
  <modules>
    <module>plugin_java_project</module>
    <module>x.test.feature</module>
    <module>x.test.plugin</module>
    <module>x.test.updatesite</module>
  </modules>
  <build>
    <plugins>
      <plugin>
        <groupId>org.sonatype.tycho</groupId>
        <artifactId>tycho-maven-plugin</artifactId>
        <version>0.8.0</version>
        <extensions>true</extensions>
      </plugin>
    </plugins>
  </build>
</project>
```

Figur 6 - Visar förälder-POM-filen som genererats av kommandot i figur 5.

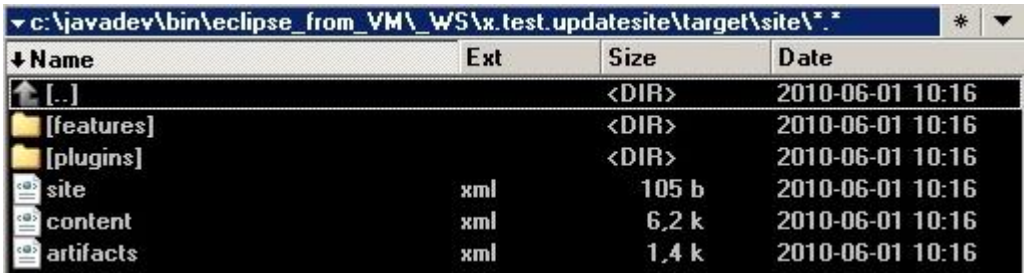
Tack vare att Tycho är specificerat som ett nödvändigt plugin för bygget i förälder-POM-filen spelar det sedan ingen roll om vi använder Maven i kommandotolken eller m2eclipse för att bygga projektet. Maven kommer i båda fallen leta reda på Tycho bland de repositories som finns tillgängliga och använda sig av det vid bygget. Ska bygget genomföras med m2eclipse måste dock målplattformen specificeras i bygg-konfigurationen. Figur 7 visar hur bygget på kommandoraden med den lokala Maven-installationen gick till.

```
c:\javadev\bin\eclipse_from_VM\_WS>mvn3 clean install -Dtycho.targetPlatform="C:
\Program Files\IBM\Lotus\Notes\framework\rcp\eclipse" -Dosgi.os=win32 -Dosgi.ws=
win32 -Dosgi.arch=x86

[INFO] Scanning for projects...
[WARNING] No explicit target runtime environment configuration. Build is platform dependent.
[INFO] tycho.targetPlatform=C:\Program Files\IBM\Lotus\Notes\framework\rcp\eclipse overrides project target platform resolver=local
[WARNING] No explicit target runtime environment configuration. Build is platform dependent.
[INFO] tycho.targetPlatform=C:\Program Files\IBM\Lotus\Notes\framework\rcp\eclipse overrides project target platform resolver=local
[INFO] Resolving target platform for project MavenProject: x.test:plugin_java_project:1.0.0 @ C:\javadev\bin\eclipse_from_VM\_WS\plugin_java
[INFO] --- maven-install-plugin:2.3:install (default-install) @ x.test.updatesite ---
[INFO] Installing C:\javadev\bin\eclipse_from_VM\_WS\x.test.updatesite\target\site\site.xml to c:\lib\mavenrepo2\x\test\x.test.updatesite\0.
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Summary:
[INFO]
[INFO] _WS ................................................ SUCCESS [4.520s]
[INFO] plugin_java_project ................................ SUCCESS [27.134s]
[INFO] x.test.plugin ...................................... SUCCESS [3.226s]
[INFO] x.test.feature ..................................... SUCCESS [1.496s]
[INFO] x.test.updatesite .................................. SUCCESS [5.455s]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 1:06.893s
[INFO] Finished at: Tue Jun 01 10:16:13 CEST 2010
[INFO] Final Memory: 31M/80M
[INFO] ------------------------------------------------------------------------
```

Figur 7- Visar kommando samt valda delar av loggen för ett bygge av ett par plugin-projekt med Tycho.

Det färdigbyggda projektet sparas sedan av Maven i en underkatalog till projektet som Figur 8 visar, och eventuellt även i ett repository beroende på vilket byggsteg som specificerats. I detta exempel specificerades byggstegen *clean* samt *install* som innebär att alla tidigare byggda filer ska raderas, och att en artefakt av projektet även ska installerats i vårt lokala repository vid bygget, som visas i figur 9.

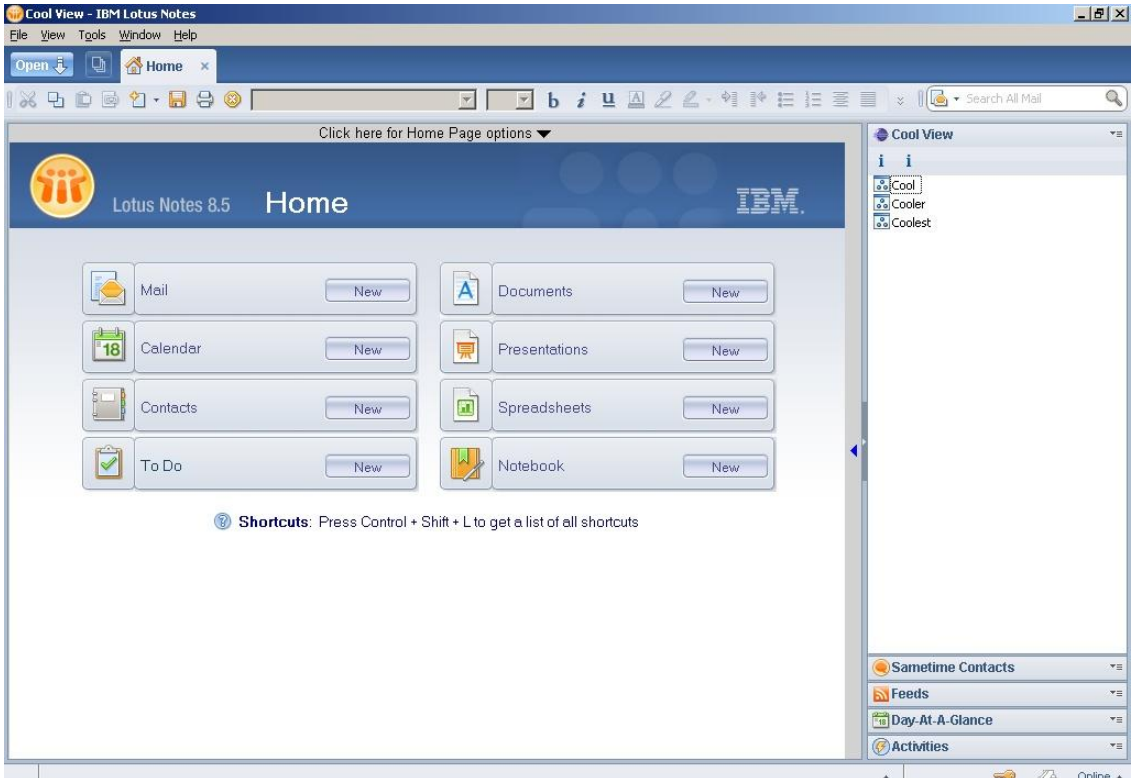


Figur 8 - Visar av Tycho byggda filer i målmappen för bygge av x.test.updatesite.

Figur 9 - Visar av Tycho byggd artefakt i local repository för x.test.updatesite.

Den artefakt som visas vi Figur 9 är emellertid inte korrekt genererad, det är endast filen site.xml från Figur 8 som kopierats med det nya namnet x.test.updateiste-0.0.1-SNAPSHOT.jar. En korrekt genererad artefakt ska innehålla alla filer som tillhör update-siten. Detta är en bugg med den betaversion av Tycho som används för denna prototyp. Buggen ska enligt Tycho-projektets bugghanteringssystem vara fixad i nästkommande version av Tycho. På grund av detta är det endast den färdigbyggda update-siten i Figur 8 som kan användas.

För att sedan testa att pluginet byggts korrekt gjordes en testinstallation i Lotus Notes. Programmet startades, menyn för installation av plugins togs fram, som sökväg angavs katalogen i Figur 8. Update-siten kunde läsas korrekt av Lotus Notes och den feature som innehöll pluginet kunde markeras. Installationen slutfördes sedan utan några problem och programmet krävde som väntat en omstart för att aktivera det nya pluginet. Efter omstarten kunde man som Figur 10 visar använda sig av den meny pluginet skapat.



Figur 10 - Visar det nybyggda plugin som installerats i Lotus Notes från den nybyggda update-site som visas i Figur 8. Det går under benämningen "Cool View" och återfinns till höger i bilden.