Avdelningen för Datavetenskap och Informatik

Anton Danielsson och Niklas Kling

Presentation av projektstatus samt design av automatiska tester

Presentation of Project status and design of automated tests

Examensarbete 15 hp

C-uppsats Datavetenskap

Datum/Termin: 2012-06-07

Handledare: Thijs-Jan Holleboom

Examinator: Donald F. Ross

Löpnummer: C2012:06

Karlstads universitet 651 88 Karlstad

Tfn 054-700 10 00 Fax 054-700 14 60

Information@kau.se www.kau.se

Presentation av projektstatus samt design av automatiska tester

Presentation of Project status and design of automated tests

Niklas Kling

Anton Danielsson

Denna rapport är skriven som en del av det arbete som krävs för att erhålla en kandidatexamen i datavetenskap. Allt material i denna rapport, vilket inte är mitt eget, har blivit tydligt identifierat och inget material är inkluderat som tidigare använts för erhållande av annan examen.

Niklas Kling

Anton Danielsson

Godkänd, 5 juni 2012

Opponent: Vincent Thuning

Opponent: Björn Nordström

Handledare: Thijs Jan Holleboom

Examinator: Donald F. Ross

Sammanfattning

Denna rapport beskriver det arbete vi gjorde hos Ninetech i Karlstad. Målet med vårt arbete var att skapa en applikation som hämtar information rörande resultat av automatiska byggen i pågående projekt. Sammanställningen av olika projekt skulle visas löpande. Vårt arbete kan beskrivas som två delar, en praktisk och en teoretisk del. Den praktiska delen bestod av att skapa en applikation som visar status på de olika projekt som Ninetech för tillfället arbetar med. Denna applikation är tänkt att köras dagligen på en skärm synlig för personalen. Skärmen är också tänkt att visa gästande kunder statusen på deras projekt. Applikationen visar bl.a. information om tester som körs på de olika projekten.

I den teoretiska delen skapades ett dokument som Ninetech kan använda för att introducera sin personal till att arbeta med automatiska tester.

Presentation of Project status and design of automated tests

Abstract

This report describes the work we did at Ninetech in Karlstad. The purpose of our work was to create an application that collects information about the results of automated builds in ongoing projects. The combined information of projects should be presented continuously.

Our work can be described as two parts, one practical and one theoretical part. The practical part consisted of creating an application that shows the status of Ninetechs current projects. This application is supposed to run daily on a screen viewable for the employees. The screen will also show visiting customers the status of their project. The application shows information about tests in the different projects among other information.

In the theoretical section a document was created. This document can later be used by Ninetech to introduce their personnel on how to work with automated tests.

Innehållsförteckning

Figurförteckning

1 Inledning

Detta kapitel ger en grundläggande uppfattning om vad programutveckling är, samt ger några exempel på olika utvecklingsmodeller som tillämpas.

1.1 Vad är programutveckling?

Programutveckling syftar till processen som genomförs vid utveckling och underhåll av en mjukvaruprodukt. Genom design, analys och konstruktion är det möjligt att skapa en produkt för praktisk användning. Detta gäller för alla typer utav utveckling, och därmed även programutveckling. IEEE, ett institut som bl.a. tillhandahåller tekniska standarder, definierar programutveckling enligt följande [1]:

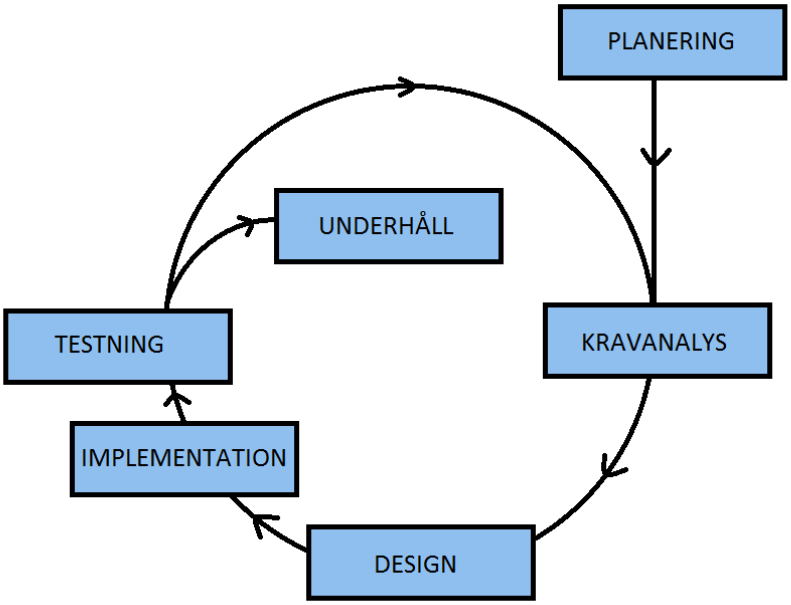
1. The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software, that is, the application of engineering to software. 2. The study of the approaches as in (1).

I enlighet med definitionen är alltså programutveckling ett sätt att skapa mjukvaruprodukter på ett systematiskt, disciplinerat och betydande sätt.

Utvecklingen kan delas upp i ett antal faser. Dessa faser bör passeras oavsett vilken utvecklingsmodell som utvecklaren använder, se figur 1.

De olika faserna är:

- Planering – I denna fas bestäms hur projektets upplägg och hur det ska genomföras. En tidsplan tas även fram.
- Kravanalys – Här bestäms de mål och krav som projektet bör uppfylla när det är färdigställt.
- Design – I denna fas bestäms systemets struktur och hur det bör se ut för att nå målen.
- Implementation – Här sker utvecklingen av systemet. Programmering utförs, databaser skapas och visuella objekt ritas upp m.m.
- Testning – Systemet testas här för att försäkra att målen har uppfyllts.
- Underhåll – Efter att systemet har använts en tid kan brister som kräver vidare arbete upptäckas.



Figur 1: De faser som passeras vid programutveckling

Många av de programutvecklings-projekt som påbörjas misslyckas tyvärr. Orsakerna till detta är ofta en missad deadline eller överskriden budget. Dessa orsaker beror i sig på undermålig planering eller testning.

Ett exempel på ett misslyckat projekt är projektet Bolit. Bolit var tänkt att förbättra arbetssättet på Patent och Registreringsverket (PRV). Projektet startades under våren 1997 och var planerat att vara klart i maj 1999. Utvecklingen gick tyvärr inte som man tänkt sig. Viktiga beslut om arkitektur förpassades till framtiden och kravspecifikationen var bristande. Efter att grovt överskridit budget samt deadline lanserades dock projektet under sommaren 2000. Vid en första anblick såg systemet bra ut, men det visade sig senare att systemet inte fungerade som tänkt och kunde därför inte användas. Tio år senare stod PRV fortfarande utan ett fungerande system.[2]

Detta misslyckande ger indikation på undermålig planering och användande av tester.

1.2 Olika utvecklingsmodeller

För att minska chansen till ett misslyckande används ofta någon typ av utvecklingsmodell för att strukturera upp arbetet med utveckling av programvara. Det finns flera olika modeller

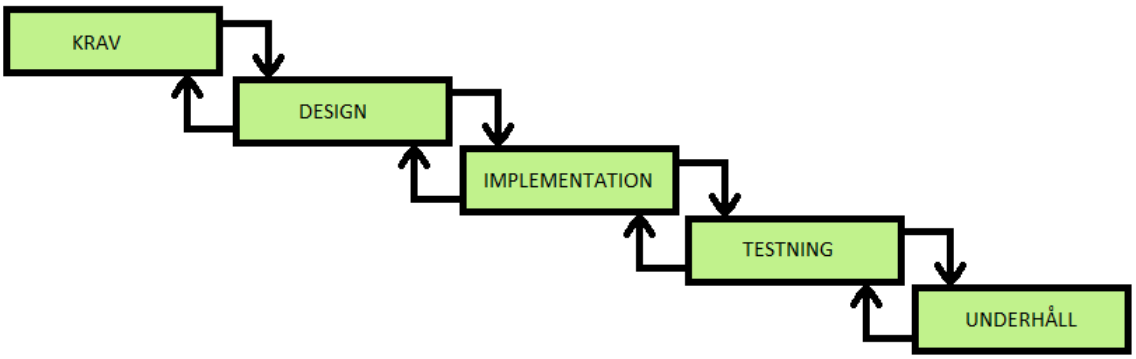
att följa, alla med sina för- och nackdelar. Några exempel på modeller som används idag är vattenfallsmodellen, agila modeller samt extremprogrammering.

1.2.1 Vattenfallsmodellen

Tanken med denna modell är att man först ska göra färdigt ett steg innan man fortsätter med nästa. Som i ett vattenfall, se figur 2.

Ett exempel på hur denna modell kan användas är enligt följande:

1. Ett möte hålls med en kund. Tillsammans skapar man en kravspecifikation.
2. En design som beskriver strukturen över systemet tas fram.
3. Systemet skapas. Inga ändringar i strukturen görs.
4. Systemet testas för att se om det uppfyller kravspecifikationen. Om ett krav ej uppfylls går man tillbaka till motsvarande steg.
5. Systemet levereras och eventuellt underhåll utförs.



Figur 2: En illustration över vattenfallsmodellen

Modellens kanske största nackdel är att testningen sker först i slutet av utvecklingen. Detta medför att felaktigheter är svåra att lösa, detta eftersom systemets design kan vara svår att ändra i ett senare skede [3].

Modellen är en av de äldsta och presenterades redan år 1956 av Herbert D. Benington [4]. Trots sin ålder och sina många nackdelar används denna modell fortfarande brett inom branschen idag. En anledning till detta är att strukturen är lätt att förstå.

1.2.2 Agila utvecklingsmodeller

Ett antal industriexperter träffades år 2001 för att diskutera hur programvaruutveckling kunde förbättras. Målet var att hitta ett sätt att snabba upp utvecklingen och att enklare kunna

göra förändringar i systemet. Gruppen kallade sig själva ”Agile Alliance” och tillsammans skapade de ”Agile Manifesto”, ett manifest innehållande principerna vid agil utveckling [3].

Till skillnad från vattenfallsmodellen arbetar man här inte med varje steg för sig. Utvecklingen sker istället genom att man gör små, samtidiga, ändringar i varje steg istället för att fokusera på en stor förändring. Detta medför att det hela tiden finns en fungerande version av systemet.

Agila metoder tillämpar ofta parprogrammering, där flera utvecklare arbetar tillsammans på samma dator. För att upprätthålla en god kvalitet används också ofta enhetstester.

Generellt sett är denna sorts modeller svårare att överblicka jämfört med vattenfallsmodellen men dock mer effektiva vid felhantering.

Exempel på agila utvecklingsmodeller är Scrum och Extrem programmering.

1.2.3 Extrem programmering och testdriven utveckling

Extrem programmering, även kallat XP, är en agil utvecklingsmodell som är skapad av en av ursprungsförfattarna till ”Agile Manifesto”, Kent Beck. Extrem programmering syftar till att hålla nere utvecklingskostnader genom att använda korta utvecklingscykler. I denna modell är ändringar en naturlig, önskad del av utvecklingen. Vid tillämpning av denna modell bör det därför redan från början vara inplanerat att ändringar kommer att ske.

Vid arbete med extrem programmering är det viktigt att ha kunden lättillgänglig. Kunden är den som bestämmer vad som ska uppfyllas och prioriteras i systemet. Genom flitig användning av enhetstester och täta testkörningar hålls kunden uppdaterad och kan direkt ge feedback.

En av grundpelarna i extrem programmering är testdriven utveckling (TDD). Detta innebär att tester skrivs innan tillhörande funktionell kod skrivs. När en modul är avklarad körs alla tester för att försäkra att systemet fungerar. För mer information om TDD, se kapitel 2.4.

1.3 Projektets mål

Målet med projektet är att skapa en applikation som visar status på de projekt Ninetech jobbar med. Denna är tänkt som en morot för att få personalen på Ninetech att jobba mer testdrivet vid utveckling.

En lathund som informerar hur ett projekt läggs till i applikationen ska också tas fram.

1.4 Rapportens upplägg

Kapitel 1: Introduktion: Kapitlet ger en grundläggande uppfattning om vad programutveckling är, kunskap som kan vara bra att ha vid andra delar av rapporten.

Kapitel 2: Bakgrund: En beskrivning av arbetets bakgrund ges. Syfte och relevanta begrepp förklaras.

Kapitel 3: Ninetech TestBoard – Implementation och design: Arbetet vid skapandet av applikationen beskrivs. Slutresultatets design visas även.

Kapitel 4: Skapandet av en Lathund: Arbetet vid skapandet av lathunden beskrivs.

Kapitel 5: Resultat: Kapitlet redogör de resultat vi upplevde när program och lathund lanserats.

Kapitel 6: Slutsats: Här skildras våra tankar och åsikter på arbetet. Detta kapitel innehåller även tankar om vad som kan förbättras i framtiden.

1.5 Summering

Det finns alltså flera olika modeller att gå efter vid utveckling av programvara. Oavsett vilken modell som följs verkar det vara en bra idé att använda sig utav tester. Ett projekt som misslyckas redan vid planeringen kan mycket väl kosta stora summor pengar.

Eftersom det är bra att upptäcka fel i ett tidigt skede är det viktigt att köra tester ofta. För att ytterligare effektivisera processen, kan det vara bra att köra dessa tester automatiskt.

2 Bakgrund

Detta kapitel kommer att beskriva vad det är för sorts arbete som utförts och vad det hade för mål. Kapitlet börjar med en presentation utav företaget för att sedan övergå till en beskrivning utav målet med projektet. Den resterande delen av kapitlet beskriver olika begrepp som berör arbetet.

2.1 Ninetech som företag

Ninetech startades år 1993 och sysselsätter idag ca 130 personer. Under år 2011 utsåg Dagens Industri Ninetech till årets ”Gasell” i Värmland, något som vittnar på att de är ett växande företag [5].

Ninetechs verksamhet kan delas in i fyra marknadsområden:

1. **Extern webb**

Extern webb syftar på att Ninetech bistår med hjälp när företag vill öka sin webbnärvaro. Just detta område hjälper företag med deras webbsatsningar utåt mot kunder. En sådan satsning kan vara t.ex. en ny webbplats, en kampanjsajt eller en e-handelslösning. Ninetech kan även bistå organisationer med strategisk rådgivning inom webb och hur man kan stärker sitt varumärke med hjälp utav den.

2. **Intern webb**

Detta område syftar till att hjälpa organisationer att stärka dess interna kommunikation med hjälp utav webben. Ninetech kan utveckla lösningar som ger medarbetarna information och känslan av engagemang och som samtidigt kan effektivisera processer och produktion. Exempel på denna typ av lösningar är intranät och stödsystem för projekthantering.

3. **Affärslösningar**

I detta område infinner sig lösningar som bistår organisationer att uppnå affärsmål. Dessa lösningar består av system som underlättar beslutsfattning och strukturerar upp handel. Ninetech levererar både egna system och hjälper till med anpassning av andra existerande system.

4. **Servicetjänster**

När en organisation har investerat i ett nytt IT-system så måste det också underhållas och förvaltas. Denna typ av tjänster kan Ninetech också bistå med. Ninetech kan bistå med hjälp på den tekniska biten såsom drift och service men också med den mer icke-tekniska biten

såsom webbanalys och hjälp med publicering av information. Om Ninetech exempelvis har hjälpt en kund att skapa en blogg, kan de också hjälpa till och ge tips om vad man kan skriva på den.

2.2 Projektet

Det Ninetech önskar är två nivåer av affärsnytta med hjälp utav en ny applikation. Först och främst önskas en presentation av Ninetechs olika projekt. Denna ska på ett tilltalande och lättillgängligt sätt visa deras status för personal.

Detta ger personalen mer inblick i hur det går för de olika projekten. Det uppmanar även till att få projektinnehavarna att hålla sitt projekts status på en bra nivå. Genom att även visa status på projektens tester, önskar Ninetech att detta ska leda till att deras personal ska arbeta mer med tester och testdriven utveckling (TDD).

Den andra nivån av affärsnytta ligger i att även kunna visa upp projektens status för kunder och därmed visa att de arbetar hårt med tester och distribuerar projektens status öppet inom organisationen.

Tillsammans med denna applikation önskar Ninetech också lansera en manual för att hjälpa personalen komma igång med automatiska byggen. I denna manual bör även en beskrivning på vilka inställningar som behöver göras för att ett projekt ska visas i applikationen.

2.2.1 Applikation

Med hjälp utav en applikation önskar Ninetech visa:

1. Projektnamn
2. Projektets ägare
3. Information om senaste bygge
    - Status
    - Datum
    - Relaterade tester (totalt antal, antal lyckade/misslyckade)
4. Andel av koden som täcks av tester (code coverage)
5. Namnet på den person som senast checkade in kod till projektet (+ ev. bild)
6. Eventuell annan information som är intressant och går att hämta från servern

Då alla på Ninetech förstår svenska önskas applikationen att visa information på detta språk.

Applikationen är tänkt att köras på en skärm på Ninetechs huvudkontor i Karlstad. Den är tänkt att vara igång varje dag under kontorstid. För att inte visa samma värden hela dagen uppdateras applikationen förslagsvis med ny data från Team Foundation Servern med ett visst tidsintervall.

2.2.2 Lathund

Eftersom en del anpassningar troligtvis kan behöva göras för att projekten ska visas i applikationen är det bra om lathunden beskriver dessa. För en utvecklare som inte arbetat med tester och automatiska byggen tidigare, ska manualen kunna användas som ett läromedel.

2.3 Testdriven Utveckling

Testdriven utveckling (TDD - Test-Driven Development) är ett sätt att skapa applikationer genom att först skriva tester och sedan skriva kod som går igenom dem. Genom att skriva testerna först så tänker man igenom kraven och designen på programmet innan man börjar skriva funktionell kod. Skrivs testerna först är det också lättare att skapa effektiv kod med en bra struktur vilket i sig öppnar upp för enkla och billiga eventuella förändringar i framtiden.

För många erfarna utvecklare kan det vara svårt att ta till sig TDD eftersom det ibland kan ses som enklare att direkt skriva funktionell kod. Parprogrammering kan vid detta fall hjälpa till mycket eftersom utvecklarna då kan hjälpa varandra att hålla sig till rätt arbetssätt. En van TDD-utvecklare skriver inte en rad kod utan att först ha skrivit tillhörande test.

Det första steget som tas vid användning av TDD är att skapa ett enkelt test. När testet är skapat så körs det för att se att det misslyckas. Efter att testet har misslyckats är det upp till utvecklaren att skriva den kod som krävs för att testet ska gå igenom. Lyckas utvecklaren med detta skrivs ett nytt test och cirkeln börjar om igen [6]. När ett test har gått igenom körs alltid alla test för att försäkra att den nya funktionella koden inte har påverkat tidigare kod och testresultat.

TDD kan delas upp i två olika nivåer Acceptans TDD och Utvecklar TDD [6].

1. **Acceptans TDD**

Med Acceptans TDD skriver man ett acceptans test som definierar kraven på projektet. I acceptanstestet finns det samlingar med instruktioner kopplade med de förväntade resultat man är ute efter [7]. Verktyg som används vid ATDD är exempelvis Fitnesse eller Rspec.

2. **Utvecklar TDD**

Denna nivå är den som man normalt refererar till när man nämner TDD. Här skrivs ett enda utvecklartest och sedan så effektiv kod som möjligt för att klara av det specifika testet.

Ofta används enhetstester i denna nivå storskaligt, att tänka på är dock att enhetstester inte är samma sak som Utvecklar TDD.

Det går att arbeta med Utvecklar TDD utan att använda sig utav Acceptans TDD men det är svårt att arbeta med Acceptans TDD utan att använda någon typ utav Utvecklar TDD.

Ett exempel på hur utveckling med TDD kan se ut är följande:

En person vill skapa ett program som kan lista en samling böcker. Programmet ska tillåta användaren att lägga till och ta bort böcker ur listan.

Det första utvecklaren gör är att skapa en lista med tester som måste göras. Den kan se ut som följande:

- Skapa en ny lista och kontrollera att den är tom
- Lägg till en bok i listan och kontrollera att listan inte är tom.
- Ta bort alla böcker ur listan och kontrollera att den är tom.
- Skriv ut listan när den är tom, kontrollera att programmet inte kraschar.
- Lägg till två böcker och ta bort en, kontrollera att rätt bok togs bort genom att skriva ut listan

När man skapat föregående lista är det dags att välja vilket test man ska börja med. När man väljer det första testet kan man välja att gå efter två olika egenskaper. Den första är att man väljer det lättaste testet. I listan ovan kan det första testet ses som det lättaste. Den andra egenskapen man kan välja att gå efter är vilket test som bäst symboliserar det vi vill åstadkomma med programmet. I detta fall kan det sista testet passa bäst.[8]

Väljer utvecklaren att börja med det första testet kommer testkoden ungefär se ut som i figur 3.

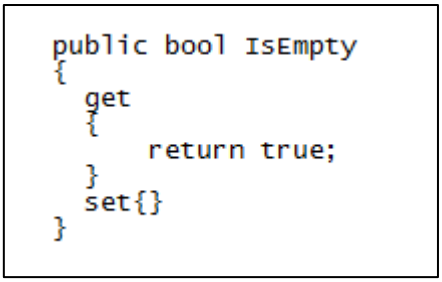
```
[Test]

public void ListaSkapadOchTom()
{
  List bokLista = new List();
  Assert.IsTrue(bokLista.IsEmpty);
}
```

Figur 3: Ett exempel på hur ett test kan se ut

I testkoden kan man se användandet av ramverket Nunit, där används s.k "assertions" som används för att testa förväntningarna på koden.

När den första testmetoden är skriven är det upp till utvecklaren att få testet att gå igenom med så lite funktionell kod som möjligt. Detta test kräver endast några rader kod i List-klassen för att gå igenom, se figur 4.



```
public bool IsEmpty
{
  get
  {
      return true;
  }
  set{}
}
```

Figur 4: Ett exempel på hur lösningen av föregående test kan se ut

När utvecklaren är klar med denna procedur börjar den om från början igen och väljer ett nytt test från listan att arbeta med.

2.3.1 Fördelar med testdriven utveckling

Forskare från Canada NRC har studerat hur effektivt det är att använda TDD som arbetssätt [9]. Enligt forskarna kan man se på TDD utifrån fyra olika positiva synvinklar:

- **Feedback**
  Testerna ger utvecklaren direkt feedback och det går att se direkt om den nya skrivna koden passar bra ihop med den äldre.

- **Uppgiftsorientering**
  Genom att skapa tester är det lätt för utvecklaren att se vad som behöver göras och vad som är gjort. Testerna ger utvecklaren ett slags schema som hjälper utvecklaren att behålla fokus.

- **Kvalitetssäkring**
  Testerna hjälper till att säkerställa projektets kvalitet genom att testa det regelbundet.

- **Lågnivå design**

  Vid skapandet så kan testerna ses som en låg nivå av design. Detta menar att när testerna skrivs bestäms också vilka metoder och klasser som behöver skapas, hur de ska användas och vad de ska heta.

Forskarna nämner också de nackdelarna som skeptikerna påstår. Ett argument är att denna typ av arbetssätt är kontraproduktivt eftersom man kommer att skriva test som inte kommer att vara till någon nytta. Det är lätt att se att tester ökar kvaliteten men svårt att se att det ökar produktiviteten. Ett annat argument mot TDD är att arbetssättet är svårt att ta till sig. Många skeptiker tycker också att tester inte ska skrivas utav utvecklarna själva utan av speciella kvalitetssäkrare.

För att testa sin hypotes om att tester både ökar applikationens kvalitet och utvecklingens produktivitet lät forskarna göra ett experiment. Experimentet gjordes i en datorsal där 40 studenter närvarade. Varje person fick tillgång till en dator med mjukvara för utveckling och testning samt en webbläsare. Efter att ha delat upp studenterna i två olika grupper fick en grupp lära sig hur man skriver tester innan programkoden skrivs (TDD) och den andra gruppen fick lära sig att skriva tester när den funktionella koden var skriven. Därefter var studenterna tilldelade ett antal uppgifter. Uppgifterna var frivilliga att lösa och delta i. När studenterna gjort färdigt en uppgift fick de publicera den på en viss plattform.

Kvaliteten på de olika lösningarna räknades ut med hjälp utav antalet defekter de hade. Antalet defekter fick forskarna fram genom att utföra ett antal acceptanstest.

Studentens produktivitet mättes med hjälp utav programmets storlek (bl.a. antalet rader kod) och med hjälp utav resultatet på acceptanstesterna.

När experimentet var över hade hälften av alla studenter deltagit och forskarna kunde inte se någon skillnad i varken ansträngning eller kompetens mellan deltagarna. Hursomhelst visade resultatet att de som skrev testen först hade ett högre nummer av tester och högre produktivitet. De som skrev testen först uppnådde dock inte högre kvalitet i genomsnitt men dock ett mer konsistent resultat.

Att lägga märke till är dock att denna studie endast hade en provgrupp på 40 studenter där endast hälften av dem deltog fullt ut. Generellt sett är det också väldigt svårt att mäta produktivitet då bakomliggande skillnader hos personer spelar stor roll.

2.3.2 Unit test

Eftersom att applikationen ska kunna användas som ett exempel på hur man arbetar med tester så måste den använda sig av enhetstester eller "unit-tests". En del av den information som ska presenteras i applikationen är resultatet på enhetstesterna i de olika projekten. Under utvecklingsfasen kommer presentationen bl.a. att använda sig av sig själv som testprojekt och presentera statusen av de egna testerna.

Enhetstester används för att försäkra sig om att delar av koden fungerar som man vill under varierande förutsättningar. Testerna tillåter utvecklarna att testa kod som användaren inte utsätts direkt för t.ex. genom det grafiska användargränssnittet. Detta möjliggör testning av komponenter som är svåra att testa med vanlig testning där testare får köra programmet i sin helhet. Konventionell testning sker vanligtvis efter att programmet är utvecklat och hittar man ett större fel så kan det ta lång tid att åtgärda och det får nästan alltid leveransen att dra ut på tiden. Med hjälp utav enhetstester kan utvecklaren få snabb respons vilket öppnar upp för tidiga förbättringar [10].

Även om enhetstester oftast utförs automatiskt så kan man även utföra dem manuellt. Ett manuellt sätt att göra enhetstester på är att t.ex. skapa ett dokument som man kan gå igenom punkt för punkt genom olika instruktioner. Dock, om man inte är försiktig när man skapar dokumentet så kan enhetstestet bli mer som ett integrationstest och testa för många komponenter och man missar därmed fördelarna med enhetstest.

Det mest vanliga tillvägagångssättet vid automatisk enhetstestning kräver att testmetoder blir skrivna. Tiden det tar att skriva dessa metoder gör ibland att utvecklaren ger det lägre prioritet, vilket nästan alltid är ett misstag. Även om metoderna tar tid att skriva och inte känns så kostnadseffektiva så ger dem ändå starka fördelar. Använder man enhetstester på rätt sätt kan man enkelt lokalisera fel genom att isolera olika enheter, testa dem och sedan integrera dem och testa dem tillsammans.

Microsofts nätverk för utvecklare (MSDN) [11] förklarar tillvägagångssättet i en artikel om enhetstester i fem enkla steg:

1. Beror felet på en felaktighet i enhet 1?
2. Beror felet på en felaktighet i enhet 2?
3. Beror felet på felaktigheter i båda enheterna?
4. Beror felet på kommunikationen mellan de båda enheterna?
5. Beror felet på testet i sig?

Många av de testmetoder man har skrivit kan sedan återanvändas på andra komponenter eller projekt. Detta medför att om en organisation inför användning av enhetstester så blir introduktionskostnaden hög och användningskostnaden efter det låg.

Ett vanligt ramverk man använder vid enhetstest är xUnit. Ramverket som egentligen är en samling av många Open-Source ramverk kan användas till att testa metoder och klasser. Fördelen med att använda xUnit är att det tillhandahåller en automatiserad lösning där man inte behöver skriva samma test flera gånger och man behöver inte heller komma ihåg vad resultatet ska bli av varje test. För testning i .Net-miljö används xUnit-varianten Nunit.
Det går att skapa enhetstest utan att använda sig utav något speciellt ramverk. Man tar då hjälp utav programspråkets prejudikat ("assertions") och felhantering för att signalera fel.

2.4 Team Foundation Server

2.4.1 Användning

Visual Studio Team Foundation Server är ett verktyg skapat av Microsoft som används för att underlätta samarbete mellan de personer som är involverade i ett projekt. Med hjälp av detta program kan en person arbeta med en del av ett projekt och ladda upp det på servern. Väl på servern kopplas denna del tillsammans med resten av projektet och bildar ett program. Detta medför att två personer som inte arbetar på samma plats ändå kan arbeta med samma projekt. Utöver detta hjälper TFS till att uppfylla en hel del andra funktioner [12]:

1. Hålla ordning på ”work items”.
2. Hålla ordning på vilken version som är den senaste uppladdade på servern.
3. Hantera de olika ”test case” som kan finnas i ett projekt.
4. Bygga programmet automatiskt, detta går att göra varje gång kod checkas in, men det går också att ställa in så att detta sker vid en speciell tidpunkt varje dag.
5. Möjlighet att skapa olika rapporter, bl. a. bygg-rapporter, testresultat och code coverage.

2.4.2 Arkitektur

Team Foundation Server är uppbyggt med en arkitektur som innehåller tre skikt [13]. Dessa är applikationsskiktet (Application Tier), dataskiktet (Data Tier) och klienter (Client Tier). Serverdelen av TFS består av applikationsskiktet och dataskiktet. Genom webbtjänster kommunicerar klienterna med applikationsskiktet, som i sin tur via databaskopplingar ansluter till den data som ligger lagrad i dataskiktet.

Varje TFS innehåller också minst ett Team Project som samlas i ett Team Project Collection (TPC) [14]. Med hjälp av denna TPC går det kontrollera och definiera en grupp Team Projects i en TFS. Ett Team Project i sig är en samling av kod, ”work items”, tester m.m. och används av alla de som arbetar med varje projekt.

2.4.2.1 Applikationsskiktet

När Team Foundation Servern startar sker den största delen av arbetet i applikationsskiktet. Här sker arbetet för att komma åt spårningen av både ”work items” och information till de olika rapporter som TFS kan skapa. Det är även med hjälp av detta skikt som det går att se om versionen på koden som användaren försöker ladda upp är senare än den som redan ligger på servern. Utan detta skulle en användare som ej har den senaste versionen kunna ladda upp en tidigare version av koden och därav förstöra en del av projektet.

2.4.2.2 Dataskiktet

I dataskiktet ligger data som ej kommer att ändras lagrad. Exempel på detta är de funktionella förråd som Visual Studios egna Team System verktyg tillhandahåller. Bland dessa finns ”work item”-databasen, ”team build”-databasen och ”version control”-förrådet. Klienterna kan inte komma åt detta direkt, utan all begäran av denna data måste gå genom applikationsskiktet. Detta skikt innehåller också Team Project Collections.

2.4.2.3 Klient

Klienterna kommunicerar genom webbtjänster med applikationsskiktet, som i sin tur pratar med dataskiktet. Klienterna består av integration från Microsoft Office, komponenter från VSIP (Visual Studio Industry Partners) samt ett framework för check-in policyer.

2.4.3 TFS Reporting

Team Foundation Server skapar per automatik ett antal rapporter när ett nytt Team Project skapas [15]. Rapporterna är till för att enkelt hjälpa användaren avgöra projektets status. Alla som arbetar med projektet kan själva gå in på sin egen dator och titta på dessa.

I de skapade rapporterna kan man bland annat se hur många av projektets tester som misslyckas och hur många som avklaras, projektets code coverage och även projektets byggresultat. Om det är intressant att se projektets framgång under ett visst tidsintervall går det också att göra detta och då se allt från hur många buggar som har uppstått och lösts under detta intervall till vilken del av koden som inte fungerat.

2.4.4 Begrepp

- Work items [16]

Det finns flera olika sorters work items som kan skapas. Dessa beskriver olika delar och är följande:

  - Scenario: En beskrivning av vad användaren förväntar sig att programmet ska göra.
  - Bug: Uppstår en bugg, en avvikelse från vad programmet förväntas göra, kan man beskriva denna här.
  - Quality of Service Requirement: Vad programmet förväntas göra när det är klart.
  - Task: En uppgift som någon i arbetslaget måste utföra.
  - Risk: Något som kan hända i programmet som har en möjlighet att skapa problem senare i utvecklingen.

De olika work items som finns i ett projekt får unika ID nummer. Detta gör det möjligt att hålla reda på dem.

- Code coverage

Code coverage anger hur stor del av projektets kod som är täckt av test cases. Den anger dock inte om dessa tester är relevanta för programmet, och inte heller att de fungerar. Skulle 100 % av koden vara täckt av fungerande tester går det därför ej direkt anta att programmet kommer att fungera som det ska.

- Projektbygge

När någon medlem i ett projekt begär att servern ska göra ett projektbygge bygger servern projektet. Detta innebär att servern försöker köra den kod som finns uppladdad. Genom att göra detta kan man med säkerhet veta att programmet fungerar som det ska. Mer om byggen nämns i kapitel 2.7.

2.4.5 Team Foundation Server Software Development Kit

Team Foundation Server Software Development Kit (TFS SDK) är en samling med funktioner som levereras tillsammans med TFS. Dessa funktioner kan sedan användas för att kommunicera med servern från ett tredjeparts program. Exempel på funktioner är t.ex. att hämta och skicka data eller att skapa en anslutning.

2.5 Konkurrerande lösningar

Det finns sedan tidigare olika lösningar att hämta projektinformation från TFS. Dessa lösningar är dock inte helt optimala för att uppnå projektets mål.

2.5.1 TFS Reporting

TFS Reporting som nämnts i tidigare kapitel kan ses som en konkurrent till vår applikation. När ett nytt Team Projekt skapas så genereras också en samling med standardrapporter. Dessa rapporter kan ge utvecklaren snabb åtkomst till statusen på projektet, kvaliteten på programkoden och vilka framsteg som görs i projektet. Rapporterna summerar data från bl.a. work items, programkod, testresultat, och byggen. Rapporterna hittas i Team Explorer (Visual Studio) under rubriken "Reports". Förutom standardrapporterna går det också att skapa egna rapporter [17].

Team Foundation Server använder SQL server (databas) för att lagra all information om work items, tester, byggen och kvalitet. Team Foundation Server använder sedan inbyggda analystjänster för att analysera data och skapa rapporterna.

För att skapa en egen rapport kan projektmedlemmarna använda Microsoft Excel eller SQL Server Report Designer.

I Microsoft Excel finns inbyggd funktionalitet för att arbeta mot databaser och eftersom TFS lagrar all information i databaser är det enkelt att hämta relevant data.

Med hjälp av dessa verktyg går det att skapa avancerade diagram och tabeller för användning vid analys och till exempel när en kund vill få en inblick i hur det går med projektet.

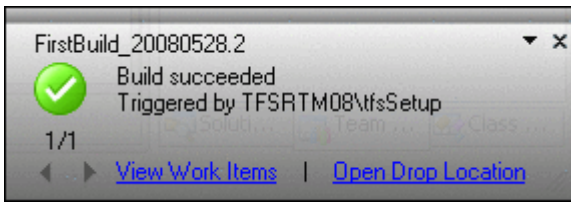
TFS Reporting behöver nödvändigtvis inte ses som en konkurrerande lösning till vår applikation utan kan också ses som ett komplement.

2.5.2 TFS Build Notification Tool

Notification tool är ett enkelt verktyg inkluderat i Visual Studio Power Tools. Detta verktyg körs i Windows aktivitetsfält och kan användas till att övervaka byggen på servern [18]. Applikationen kan ställas in på att reagera på olika händelser såsom köande och färdiga byggen eller om någon checkar in kod till projektet . När byggservern har färdigställt bygget kan man också välja att visa hur det gick.

Notification tool har den fördel att det är väldigt lätt att använda. Har man anslutit korrekt till servern från Visual Studio så fungerar verktyget direkt programmet startar eftersom att inställningarna importeras automatiskt [19].

Figur 5 visar ett exempel på hur en notifikation från programmet kan se ut.

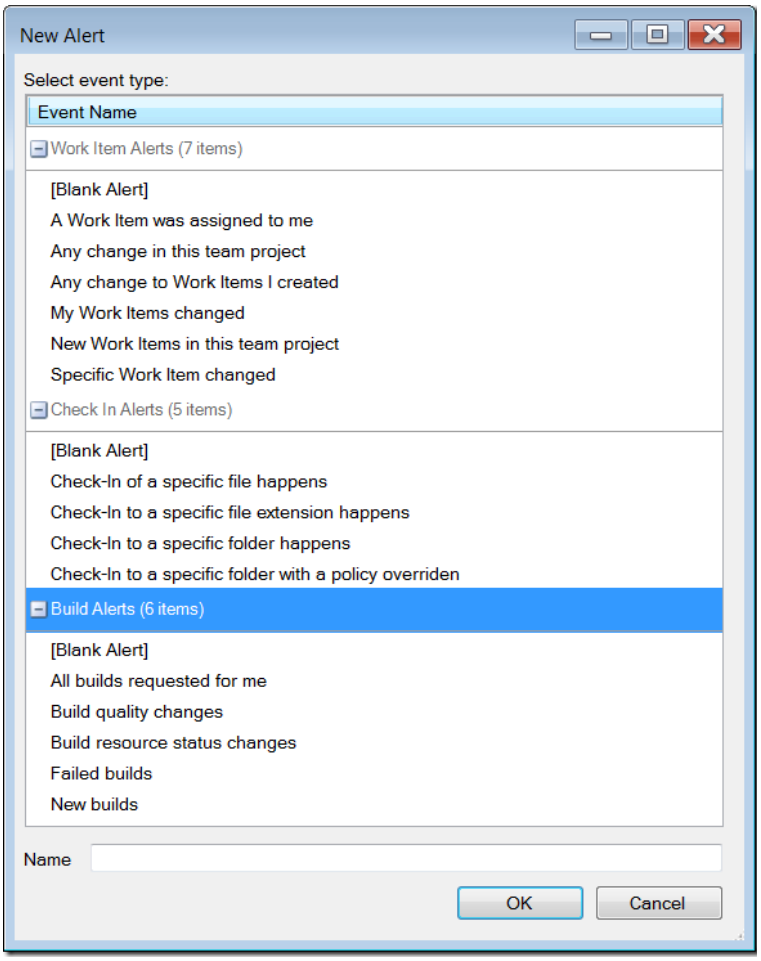


Figur 5: En notifikation från programmet Notification Tool

2.5.3 TFS Alerts

Ett annat hjälpmedel för att få information om de olika projekten på servern är att använda sig av Alerts Explorer. Precis som Notification Tool är Alerts Explorer också inkluderat i Visual Studio Power Tools. Alerts Explorer går att komma åt inifrån Visual Studio genom att man väljer Team -> Alerts Explorer i toppmenyn. Jämfört med Notification Tool ger möjlighet att hantera mer specifika händelser. Förutom de händelser som Notification Tool kan reagera på så kan även TFS Alerts reagera på ändringar inom Work Items [20].

När användaren väljer att hantera en ny händelse visas fönstret i figur 6.



Figur 6: Detta fönster visas när man vill hantera en ny händelse

När någon av de specificerade händelserna sker så kan servern skicka ett meddelande till en angiven e-mail adress. I meddelandet kan sedan användaren få information om vad som hänt.

2.6 Automatiska byggen

Efter versionshantering så är automatiska byggen det näst viktigaste utvecklare gör när de skapar ett program. Med hjälp utav automatiska byggen kan man låta en server kompilera källkod och köra tester automatiskt.

En utvecklare är vanligtvis kapabel till att göra detta lokalt på sin dator med hjälp av Visual Studio. Detta gör det möjligt för utvecklaren att testa sin del av kod, men vad händer

om koden inte fungerar ihop med annan kod i projektet? Med hjälp utav automatiska byggen kan all programkod för projektet testas ihop för att försäkra funktionaliteten.

Automatiska byggen är så viktigt för mjukvaruutveckling att Microsoft valt att integrera tjänsten som standard i Team Foundation Server. Så fort en utvecklare anslutit till en server via Visual Studio kan den se statusen för det senaste bygget som t.ex. om det kompilerades korrekt eller om enhetstesterna gick igenom.

Den data som går att få ut från ett bygge läggs direkt in i TFS data warehouse efter att bygget är färdigt. TFS data warehouse är en databas i TFS och är den komponent som innehåller data om de olika projekten på servern. Det är oftast från denna databas som de applikationer som nämns i 'Konkurrerande lösningar' från sin information ifrån. Det är också därifrån som vår applikation får sin data.

Något annat som introducerades med TFS 2010 är Workflow 4.0 vilket är en motor med tillhörande api som används för att hantera hur processer arbetar i applikationer. Ett flöde kan här ses som en mängd med olika programmeringssteg. Workflow 4.0 tillåter utvecklaren att dela upp applikationen i olika aktiviteter och därefter köra dem var för sig enligt ett visst flöde.

Innan Team Foundation Server 2010 så kördes byggen på en enda server kallad "build agent". För varje bygge kunde man då specificera en "build agent" som standard. En "build agent" kör bygget i den ordning som är skapade med hjälp utav Workflow. Den laddar sedan upp resultatet till en s.k. "drop location".

När 2010 versionen av mjukvaran kom introducerades istället komponenten "build controller". Denna komponent möjliggör att flera "build agents" kan arbeta med samma bygge.

Det finns flera händelser som kan användas för att trigga ett nytt bygge. Ofta vill man att ett bygge triggas så fort ny programkod laddas upp till servern men händer det för ofta kan det belasta byggservern i onödan. För att inte framkalla onödig belastning kan man då istället schemalägga byggen på natten då ingen arbetar med koden. Det går givetvis att använda båda alternativen samtidigt och man kan också trigga nya byggen manuellt eller genom en rad andra händelser.

Teamet som arbetar med ett program bör hela tiden sträva efter att programmet går att köra. Är man ett stort team kan detta dock bli ett problem. Även den bästa utvecklaren gör ibland fel och laddar upp kod som inte kan kompilera. Är det då hundratals utvecklare så kan man snabbt räkna ut att dessa fel sker väldigt ofta.

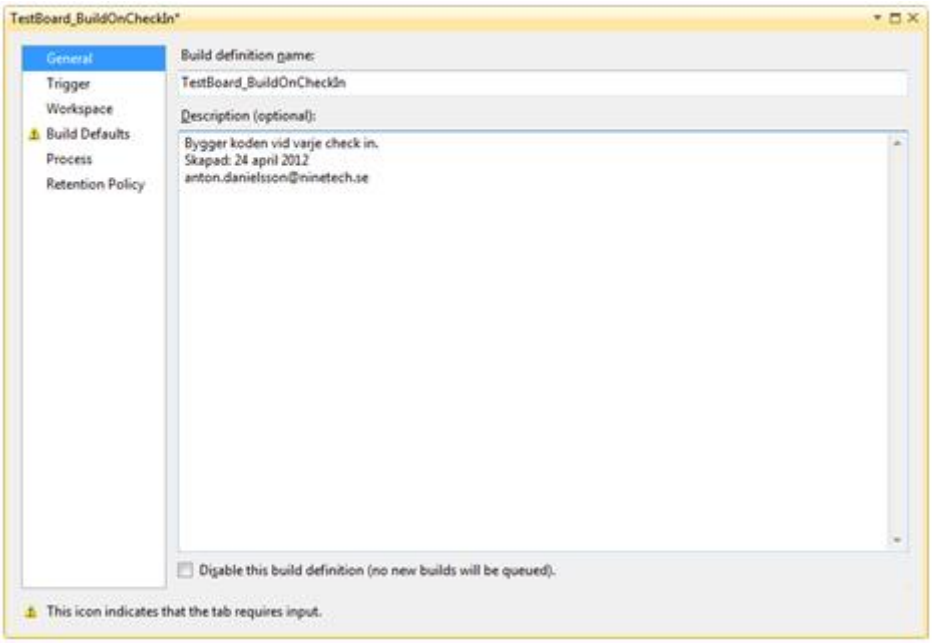
För att lösa detta används s.k. "gated checkins". Med detta menas att den kod som laddas upp först testas tillsammans med den andra koden i en annan miljö. Fungerar den bra så laddas koden upp till byggservern [21].

2.6.1 Build Definition

En "build definition" eller en byggdefinition som översättningen blir måste först göras för att en server automatiskt ska bygga en applikation och köra tester. Definitionen specificerar när bygget ska ske och hur det ska gå till. Det går att ha flera definitioner till samma projekt. Som exempel kan man skapa en definition som gör att servern kör tester och bygger applikationen vid en viss tidpunkt t.ex. på natten och en definition som bygger applikationen varje gång kod checkas in (laddas upp).

Eftersom att vår applikation arbetar med data hämtad från det senaste bygget av varje projekt så var vi tvungna att skapa en "build definition".

En build definition går att skapa direkt inifrån Visual Studio i Team Explorer genom att högerklicka på "builds" och välja "New Build Definition". Man får då upp ett nytt fönster med sex stycken sektioner där inställningar kan göras [21], se figur 7.



Figur 7: Fönstret som visas vid skapandet av en ny "build definition"

De sex olika sektionerna är:

- General

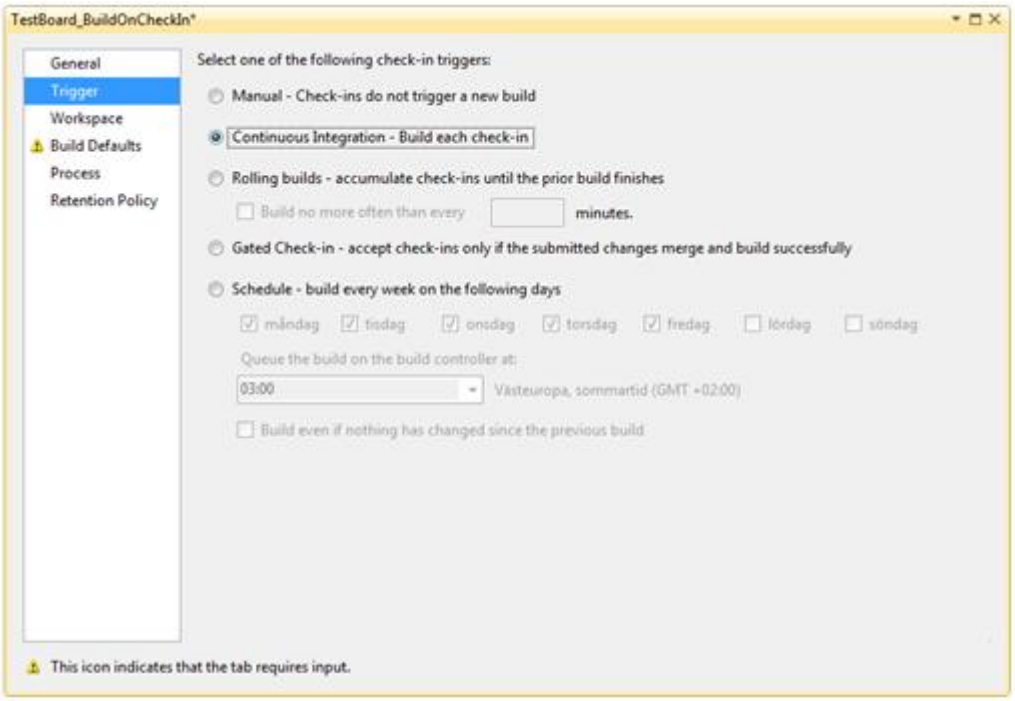
- Trigger
- Workspace
- Build Defaults
- Process
- Retention Policy

I sektionen "General" skriver man in det namn som definitionen ska ha. Här är det bra att följa någon typ utav benämningsstruktur om man har flera definitioner för samma projekt. Det är också i denna sektion som beskrivningen utav definitionen ges, t.ex. skapare, syfte, datum eller versionsnummer.

Efter att ha angett den namn och beskrivning är det upp till utvecklaren att bestämma vad som ska sätta igång ett nytt bygge. Detta görs under sektionen "Trigger", se figur 8. Här ges fem val:

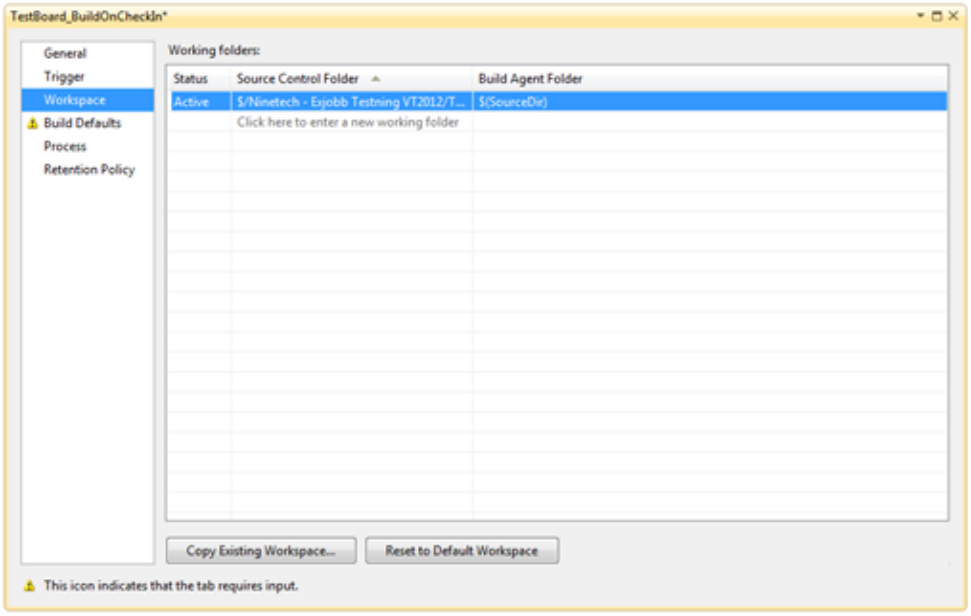
- Manual – Utvecklaren startar ett nytt bygge manuellt när det behövs.
- Continous Integration – Ett nytt bygge görs varje gång någon checkar in kod till servern.
- Rolling builds – Detta val liknar det föregående men här läggs flera incheckningar ihop innan ett bygge görs.
- Gated Check-in - Som nämnt innan. Kod kontrolleras innan den laddas upp till byggservern.
- Schedule – Ett nytt bygge görs vid en viss tidpunkt t.ex. varje onsdag kl.03.00

Det går inte att välja flera alternativ här. Då får utvecklaren istället skapa flera definitioner.



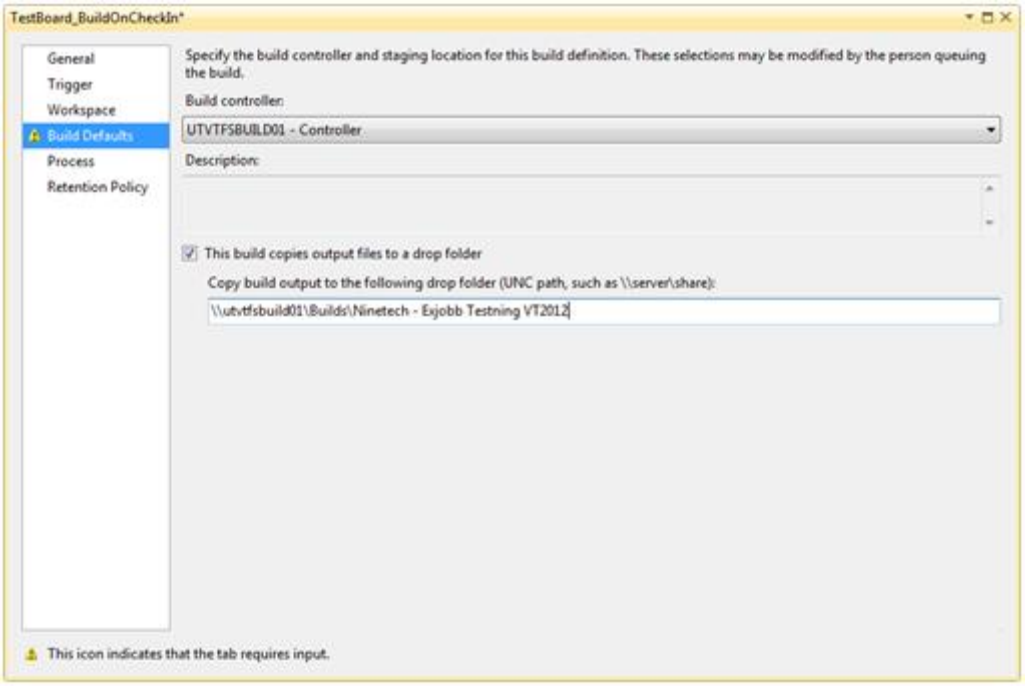
Figur 8: Sektionen "Trigger" som visas när en ny "build definition" skapas

Nästa sektion är "Workspace", se figur 9. Här anger utvecklaren vilken mapp som ska byggas. Det fungerar att sätta projektets rotmapp som arbetsyta men detta är dock inte rekommenderat i större projekt. Att ange en för brett omfamnande rotmapp kan leda till att servern gör ett bygge trots att incheckade filer inte påverkar utgången utav det.



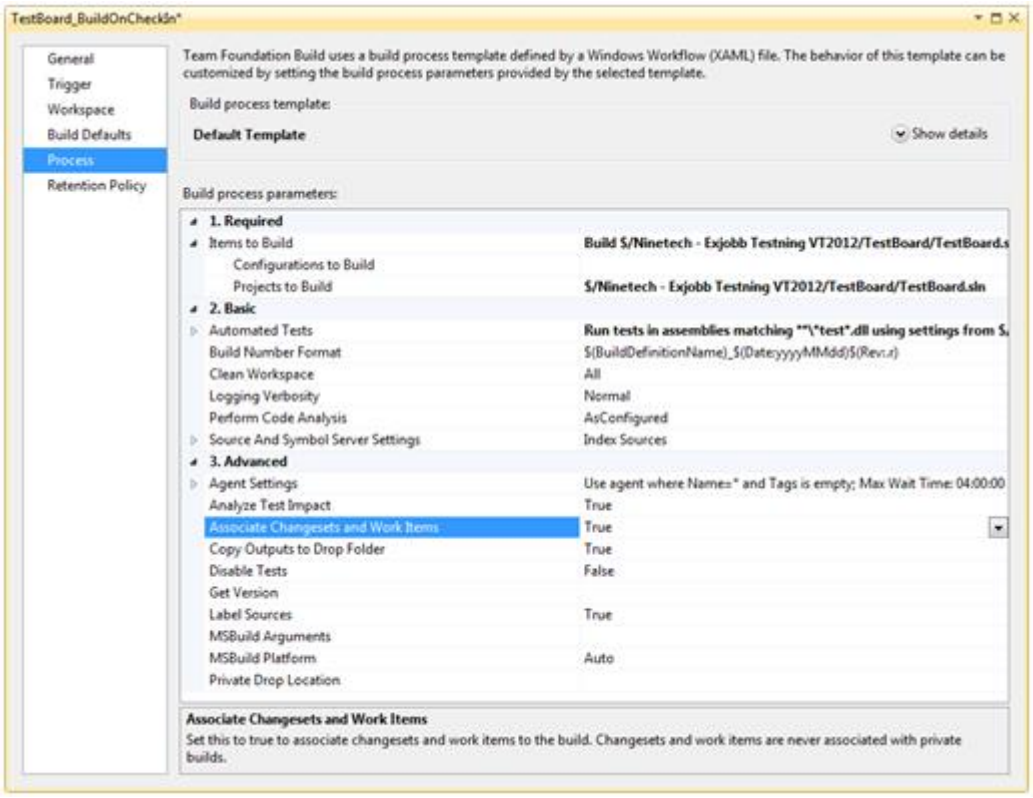
Figur 9: Sektionen "Workspace" som visas när en ny "build definition" skapas

I nästa sektion "Build defaults" (se figur 10) är det dags att ange vilken "build controller" som ska användas för att genomföra bygget. Det är upp till serveradministratören att installera en sådan. Här anges också i vilken mapp bygget ska läggas när det är klart.



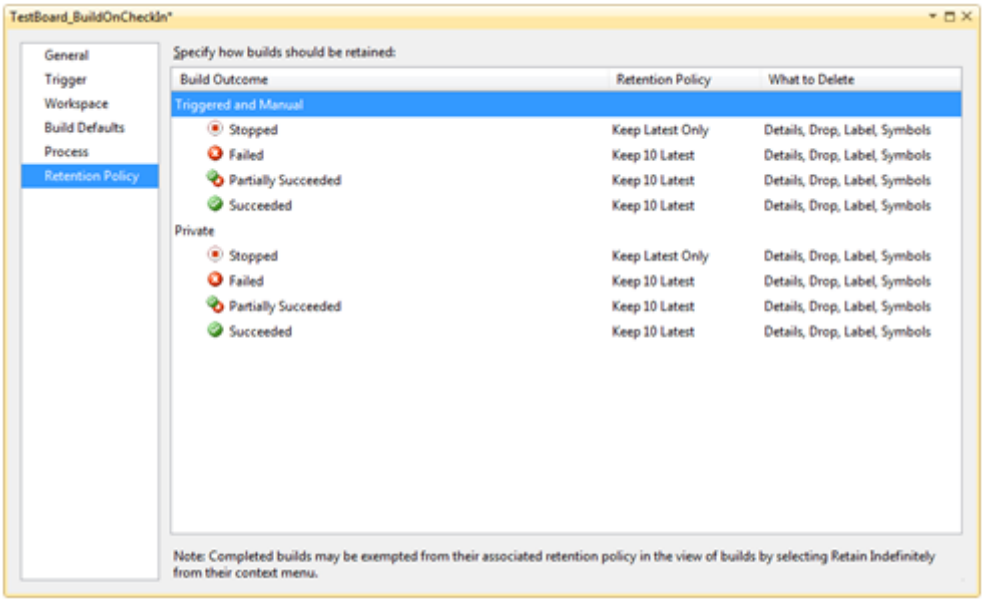
Figur 10: Sektionen "Build Defaults" som visas när en ny "build definition" skapas

I nästa sektion (se figur 11) specificerar utvecklaren vilken process som ska användas vid bygget. Processerna är av typen Workflow 4.0. Det går att skapa egna men det finns också en standardprocess som går att använda. I processen kan man specificera om tester och analyser ska göras och om loggning ska ske bl.a.



Figur 11: Sektionen "Process" som visas när en ny "build definition" skapas

Under tiden man arbetar med ett projekt är det stor risk att det görs väldigt många byggen. I den sista sektionen "Retention policy" (se figur 12) går det att specificera hur länge olika byggen sparas innan de raderas.



Figur 12: Sektionen "Retention Policy" som visas när en ny "build definition" skapas

3 Ninetech TestBoard – Implementation och design

Detta kapitel beskriver vårt tillvägagångssätt för att skapa applikationen Ninetech-Testboard. Här finns också en beskrivning utav applikationens funktionalitet och utseende.

3.1 Utredning

Innan utvecklingen av applikationen påbörjades var vi tvungna att reda ut vilken struktur vi skulle använda. Det första av våra två val var att välja om vår applikation skulle vara en lokal applikation eller om den skulle vara webbaserad. Eftersom det TFS API vi hittat är bäst anpassat för C# programmering, stod valet mellan ASP.NET (C#) och C# med WPF eller Windows Forms.

3.1.1 C# med annat presentationsspråk (Lokal applikation)

Lokala applikationer är betydligt äldre än de webbaserade.

C# (C-Sharp) är inriktat på att vara ett enkelt objektorienterat programspråk. Språket tillhör .NET-ramverket och körs främst på Microsoft Windows. Applikationer skrivna i C# körs i en miljö kallad "Common Language Runtime" (CLR). CLR fungerar som en virtuell maskin som tillhandahåller funktioner såsom fel- och minneshantering och även funktioner som kontrollerar koden efter typfel m.m. [22]. Ett exempel på hur ett C#-program kan se ut visas i figur 13.

```
using System;
class Hello
{
   static void Main() {
      Console.WriteLine("Hello, World");
   }
}
```

Figur 13: Det klassiska HelloWorld programmet, skrivet i C#

- **Fördelar:**

En lokal applikations största fördel är att den kan använda datorns resurser fullt ut. Detta gör att man kan skapa t.ex. avancerade grafiska element eller utföra tunga beräkningar. I vår applikation har vi inte planerat att göra någon av dessa men det kan vara bra att ha möjligheten.

Oftast är en lokal applikation inte lika avancerad som en webbaserad. Data man har i applikationen går enkelt att visa och man slipper att ta hänsyn till en internetförbindelse. Eftersom vår uppgift är att visa en presentation på en bildskärm kopplad till en dator, stående på Ninetechs huvudkontor i Karlstad, finns det ingen anledning till att skicka information över nätverket.

- **Nackdelar**

Har man en lokal applikation kommer man åt informationen den tillhandahåller endast där datorn befinner sig. Skulle en av konsulterna på Ninetech vara på någon annan plats att arbeta är det omöjligt att komma åt informationen.

En applikation skapad i C# är plattformsberoende. Den kräver att rätt mjuk och hårdvara används, annars fungerar den inte. En webbaserad applikation går att nå från vilken webbläsare som helst.

Applikationen kräver också att man har den installerad på det system man vill använda och uppdateras applikationen måste man se till så att alla användare får den.

3.1.2 ASP.NET

ASP.NET är ett ramverk som används för att skapa dynamiska webbsidor och webbapplikationer med hjälp utav HTML, CSS samt JavaScript. ASP.NET är utvecklat av Microsoft och är efterföljaren till ASP (Active Server Pages). Gentemot sin föregångare så är ASP.NET mer inriktat på att separera kod för presentation och innehåll. När man skriver en webbapplikation i ASP.NET använder man något av .NET-språken C# eller Visual Basic. Den kod och de kontroller man skapat konverteras sedan till HTML, CSS samt JavaScript i webbservern, innan den skickas ut till användarens webbläsare.

- **Fördelar:**

Den största fördelen med att göra applikationen webbaserad är att den blir lättillgänglig. I vårt fall skulle kunder eller konsulter som inte befinner sig på företaget för stunden, fortfarande kunna komma åt information vår applikation tillhandahåller.

I detta fall skulle det finnas en klient-del och en server-del. Klientsidan skulle i detta fall vara helt oberoende mjukvara och det enda som krävs är en webbläsare. Problem med uppdateringar är inte heller något problem eftersom applikationen hanteras direkt på servern [23].

En annan fördel med denna typ av lösning är att kraven på användarens hårdvara är lågt eftersom alla beräkningar sker på serversidan.

- **Nackdelar:**

För att uppdatera en webbapplikations innehåll krävs en s.k. "postback" då applikationen efterfrågar en uppdatering från servern. Eftersom vi vill att vårt gränssnitt ska uppdateras ganska ofta, krävs många "postbacks". Ibland kan det gå fel i anslutningen vilket leder till att informationen inte uppdateras.

Med en webbapplikation kan det även vara svårt att komma åt datorns resurser. Kommer vi på under utvecklingen att applikationen behöver reagera på andra processer kan vi stöta på problem.

Distribuerar man någonting på webben bör man även ha i åtanke att det finns risk för att information hamnar i fel händer. Vår applikation visar hur det går för Ninetechs projekt, information som kan vara intressant för konkurrerande organisationer eller potentiella kunder.

3.1.3 Lokal applikation eller Webbapplikation: Slutsats

Vi beslutade oss för att skapa en Lokal applikation. De fördelar som en webbaserad applikation tillför var inte relevanta till vår uppgift. Eftersom vårt program endast ska köras på en enda dator så finns det ingen anledning att blanda in client-server arkitekturen.

När man skapar en lokal applikation i C# så används antingen Windows Forms eller WPF. Vi stod nu inför valet att utse det presentationsspråk som passade vår uppgift bäst.

3.1.4 Windows Forms

Windows Forms (WinForms) är en del utav .NET-ramverket och används till att skapa grafiska gränssnitt för program i Windows-miljö. Ett fönster ("form") är en yta som man som utvecklare kan fylla med diverse kontroller t.ex. text, bilder eller knappar [24]. Använder man Visual Studio så kan man enkelt göra detta genom att endast dra och klicka.

Fördelar:

Eftersom Windows Forms har funnits länge finns en bred användarbas och det finns mycket hjälp att få, både av böcker och på internet.

Nackdelar:

Kan ibland vara krångligt att skapa vissa element med kod, och gör man det grafiskt med Visual Studio så får man med mycket kod man inte behöver.

3.1.5 Windows Presentation Foundation

Windows Presentation Foundation (WPF) är också en del utav .NET-ramverket utvecklat av Microsoft. WPF lanserades senare än WinForms men ska inte enligt Microsoft ses som en

ersättare, utan snarare ett alternativ till WinForms. När man arbetar med WPF kan man också skapa element grafiskt med hjälp utav Visual Studio. Skillnaden är att de här skapas i XAML-kod istället. Ett exempel på denna typ av kod visas i figur 14.

```
<Window x:Class="WpfApplication1.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="MainWindow" Height="350" Width="525">
<Grid>
    <Image Height="150" HorizontalAlignment="Left" VerticalAlignment="Top" Width="200" />
</Grid>
</Window>
```

Figur 14: Exempel på XAML-kod

Fördelar:

Enklare separering av innehåll och presentation. Det är lätt att skapa grafiska element som representerar data på ett bra sätt.

Nackdelar:

Eftersom det är ett relativt nytt koncept så kan det ibland vara svårt att få tag i vissa beskrivningar. Många guider är ofta anpassade för Windows Forms.

3.1.6 Presentationsspråk: Slutsats

Efter att ha provat på dem båda bestämde vi oss för att använda WPF. Det finns egentligen inga stora skillnader utan beror mer på tycke och smak. Vi valde WPF därför att det kändes som att den koden kändes lättare att förstå. Vi läste också att många tyckte det var lättare att skapa grafiska element med WPF.

3.2 Beskrivning av implementation

Ninetech – TestBoard är utvecklad i det objektorienterade programspråket C#. För utveckling användes programmet Visual Studio skapat av Microsoft. Applikationen har ett grafiskt gränssnitt som är skapat med hjälp utav designstandarden WPF. För att köra programmet krävs en dator med Microsoft Windows XP, Vista eller 7 installerat. För att programmet ska kunna köras korrekt krävs även en installation av ramverket Microsoft .Net version 4.

För att applikationen ska kunna hämta information om olika projekt krävs också en anslutning till Team Foundation Server. Det är rekommenderat att servern befinner sig inom

samma nätverk som den dator som kör applikationen. Detta eftersom anslutningen blir snabbare och det blir lättare för serveradministratören att ange de nödvändiga rättigheterna.

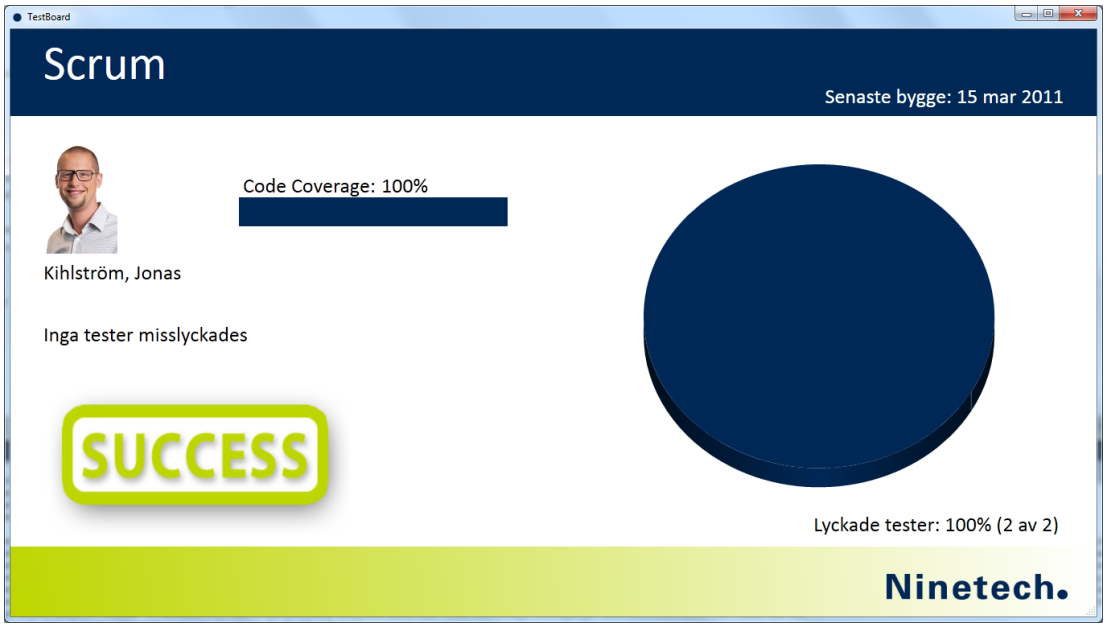
3.2.1 Utseende och användning

När programmet startar möts användaren av en s.k. ”splashscreen” (se figur 15) alltså en bild som visas när programmet laddas. Under visningen at denna bild skapas en anslutning till TF-servern. Alla projekt som applikationen har åtkomst till placeras i en lista. Detta gör att programmet slipper att visa tomma etiketter och diagram som ännu inte blivit tilldelade något värde, vilket i sig ger programmet ett professionellt och bättre utseende. När alla projekt är hämtade från servern döljs bilden och det första projektet i listan visas. Varje projekt visas i 30 sekunder, eller den tid som krävs för att alla tester ska hinna visas.

Figur 15: Den bild som visas när programmet startas

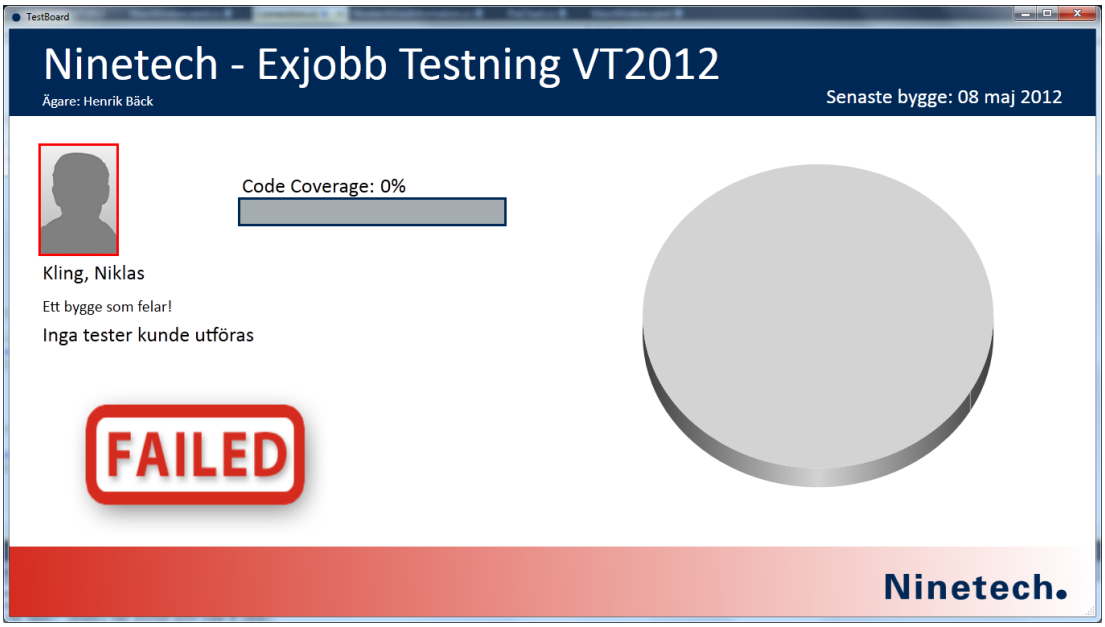
Det finns fem olika statustyper som ett projekt kan visa:

- Success – Denna status visas när ett projekt kompilerades korrekt och alla tester lyckades, se figur 16. Detta är den status som projektarbetarna bör eftersträva. För att indikera att projektet har denna status visas en grön nyans på den nedre delen av skärmen. Eftersom att inga tester misslyckades så finns det inte heller några att visa. Istället visas en grön stämpel med texten ”Success”. Ett pajdiagram visar också att 100 % av testerna har gått igenom. Under pajdiagrammet visas en kompletterande text som visar hur många tester det finns totalt.



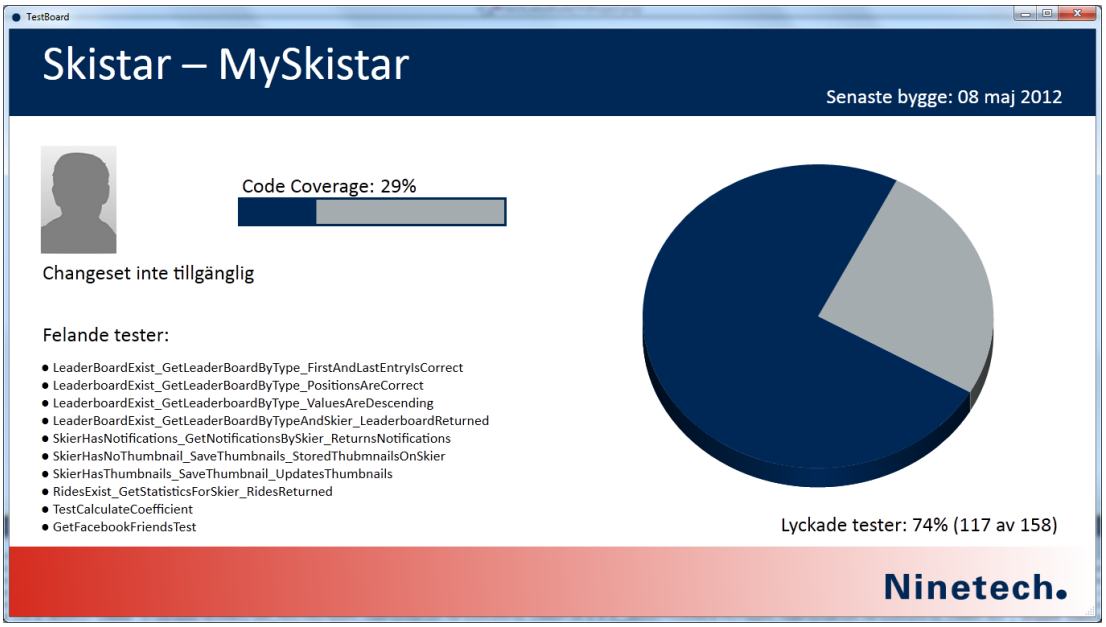
Figur 16: Skärmens utseende när ett projekt bygger korrekt och alla tester lyckas

- Failed – Denna status är den som utvecklarna ska undvika, se figur 17. Den uppstår när bygget inte kunde kompilera. Detta är oftast ett resultat av fel i programkoden. Detta är en kritisk status och kräver snabba åtgärder. För att göra det extra tydligt finns även en röd stämpel med texten ”Failed”. Bilden på personen som gjorde den senaste ändringen har en röd ram om detta är det första misslyckade bygget i en rad av byggen. Var föregående bygge också misslyckat visas inte denna ram. Ramen kan därför indikera om det är personen på bilden som har orsakat felet. Detta kan vara bra att veta när felet ska lösas. Eftersom att inte bygget kompilerade kunde inte några tester köras. Detta resulterar i att pajdiagrammet är tomt.



Figur 17: Skärmens utseende när ett projekt inte kompilerar

- Partially Succeeded – Denna status är ett slags mellanläge mellan Success och Failed. Denna status visas då det senaste bygget kompilerade korrekt men ett eller flera tester misslyckades, se figur 18. Toningen i nedre kant är även här röd. Skillnaden är dock att stämpeln ”Failed” inte visas. Istället visas där en lista med de misslyckade testerna. Denna lista rullar sakta och visar de test som inte lyckades. Rullningseffekten består av att ett nytt test visas varje sekund samtidigt som det översta testet döljs. Pajdiagrammet är vid denna status mest informativt. Det visar hur stor andel av testen som lyckats.



Figur 18: Skärmens utseende när minst ett test misslyckas

- In Progress – Denna status visas då ett projekt är under byggnadsfasen. En bild indikerar att projektet håller på att byggas. Den enda informationen som visas är projektnamnet.
- Inget bygge tillgängligt – När inget bygge är kört på projektet eller när inget bygge går att komma åt, visas denna status. Precis som vid ”In Progress” visas endast projektnamnet.

När samtliga projekt i listan har visats sker en ny anslutning till servern. Här hämtas eventuella förändringar och sparas i den lokala listan. Därefter visas det första projektet återigen.

3.2.2 Vad presenteras?

- Datum för bygge – Detta datum visar när det senaste bygget gjordes. Detta är viktigt då det går att avgöra hur aktuell informationen som visas är. Datumet visas längst upp i det högra hörnet.
- Byggstatus – Fönstrets färg och eventuella bilder indikerar statusen på det senaste bygget.
- Testinformation – Eftersom att syftet med projektet var att försöka få Ninetech att arbeta mer testinriktat, kanske detta är den viktigaste informationen som visas. Applikationen visar vilka och hur många tester som misslyckades. För att lättare kunna identifiera hur stor del av testerna som lyckas finns ett pajdiagram till hjälp.

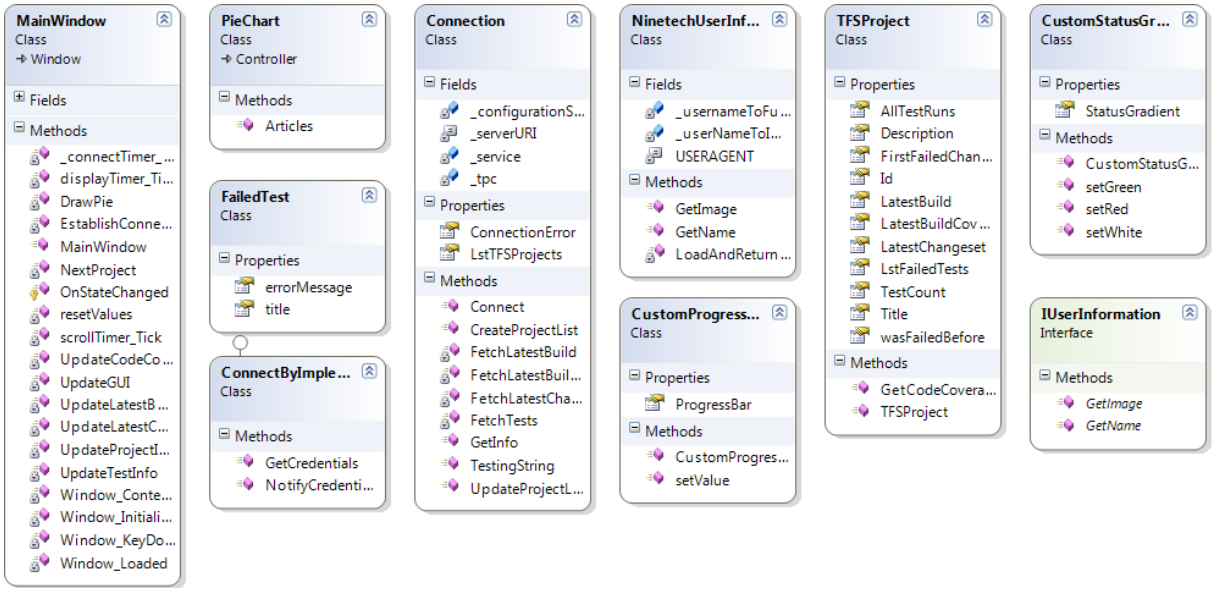
- ”Code coverage” – Att visa hur stor del av koden som är täckt med tester är viktigt. Detta lockar personalen till att skriva fler tester. Skulle inte denna information visas skulle det finnas en risk att personalen hellre skulle skriva färre tester för att lättare nå statusen ”Success”. Som hjälp för att visualisera täckningsgraden används en grafisk förloppsmätare.
- Senaste incheckning – För att enkelt kunna se vem som gjorde den senaste ändringen i projektet visas i applikationen både namn och bild på personen. Under personens namn syns även den eventuella kommentar som lämnades när ändringen laddades upp till servern.

3.2.3 Anslutning till Team Foundation Server

För att ansluta till servern används det utvecklar-kit (TFS SDK) som tillhör TFS. I detta kit finns funktioner som går att använda vid skapandet av en anslutning.

3.2.4 Applikationens uppbyggnad

Applikationen består utav ett XAML-dokument, ett interface och nio stycken klasser, se figur 19.



Figur 19: Klassdiagram över projektet.

- MainWindow.xaml – Det är XAML-dokumentet som bestämmer hur applikationen ser ut. Detta dokument specificerar bl.a. vilka element som ska visas, färger och positioner. Detta dokument används eftersom vi valde att följa standarden WPF. Kopplat till detta dokument finns också en klass.

- MainWindow.xaml.cs – Denna klass har samma namn som dokumentet men dock en annan filtyp. Klassen bestämmer vad som ska ske vid utvalda händelser. Denna klass används också för att räkna ut vad som ska visas.
- Connection.cs – Detta är den klass som används vid anslutning och hämtning av data till vår applikation. Klassen är uppbyggd av ett antal metoder som var och en hämtar olika typer av data, såsom senaste bygge, changeset eller testinformation. Denna klass sparar varje hämtat projekt som ett objekt av typen ”TFSProject” i en lista.
- TFSProject.cs – Denna klass definierar ett objekt. Detta objekt lagrar all information om ett visst projekt. Objektet tillhandahåller även funktioner för åtkomsten av denna data.
- ConnectByImplementingCredentialsProvider.cs – Applikationen använder nu den inloggade Windows-användaren som autentisering vid anslutning till servern. Denna klass möjliggör att en annan användares rättigheter kan brukas. Klassen används för närvarande inte, men finns implementerad om detta önskas i framtiden.
- FailedTest.cs – Denna klass definierar också ett objekt, denna gång innehållandes information om ett misslyckat test. Informationen används sedan i en lista över de misslyckade testerna.
- NinetechUserInformation.cs – Eftersom att det ej går att hämta personalens fullständiga namn från servern måste denna klass konvertera inloggningsnamn till personernas för- och efternamn. Denna klass kan även returnera en bild på den efterfrågade personen. Namn och bild hämtas från Ninetechs hemsida. Klassen använder sig av interfacet ”IUserInformation.cs”.
- PieChart.cs – Denna klass används för att rita upp pajdiagrammet i en bild som returneras till huvudfönstret.
- CustomProgressBar.cs – För att visualisera code coverage används denna klass. Den skapar en toning mellan två färger, där den ena visar täckningsgraden i Ninetechs blåa färg.
- CustomStatusGradient.cs – En typ utav toning skapas även i denna klass. Denna toning är den som används i den nedre delen av skärmen för att indikera ett projekts status. Klassen innehåller tre metoder för att bestämma toningens färg, grön, vit eller röd.

3.2.5 Felhantering

Det är mycket som kan gå fel vid användandet av en applikation. Som utvecklare är det viktigt att tänka på att fånga upp dessa fel för att förhindra att applikationen kraschar eller fungerar på fel sätt. De fel som är mest förekommande i Ninetech – Testboard beror på att applikationen inte kan komma åt önskad information.

Ett utav dessa fel kan uppstå när applikationen inte kan skapa en anslutning till servern. Detta kan bero på t.ex. fel i nätverket eller fel på servern. Applikationen skapar en anslutning till servern när den startas och sedan varje gång den har gått runt ett varv i projektvisningen.

Misslyckas anslutningen vid uppstart av applikationen informeras användaren genom ett felmeddelande, se figur 20. Servern kommer vid detta tillfälle att försöka ansluta igen efter ett visst tidsintervall.

Figur 20: Skärmens utseende om anslutning till server misslyckas vid uppstart.

Anslutningen kan också misslyckas när ny information hämtas om projekten. Vid denna tidpunkt finns dock information från föregående anslutning sparad. Inträffar detta kommer applikationen att visa dessa värden. Samtidigt kommer även en text visas som informerar användaren att informationen inte är aktuell.

Förutom problem med anslutningen händer det ibland att applikationen inte kommer åt information om den senaste förändringen. Går inte denna information att komma åt så visas detta i programmet. Orsaken är att byggdefinitionen inte är korrekt konfigurerad. För att förhindra denna typ av fel så beskrivs skapandet av en byggdefinition i applikationens tillhörande lathund (se bilaga).

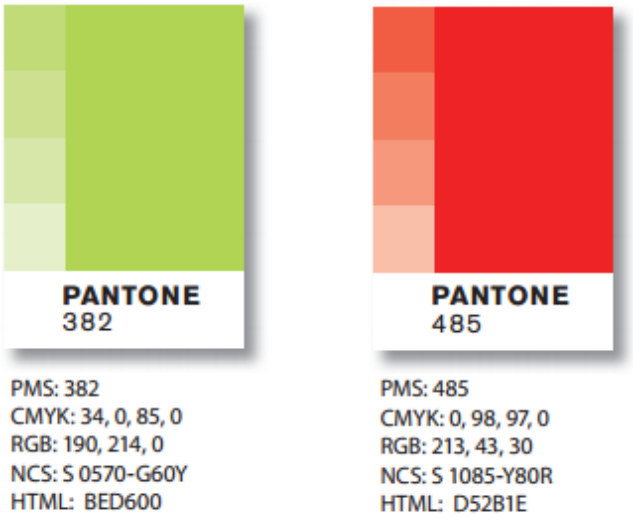
3.2.6 Grafisk profil

Under utvecklandet av applikationen blev vi tillfrågade att följa Ninetechs grafiska profil. En grafisk profil är en samling utav designregler som man bör följa för att få sin design att följa den design en organisation har på andra projekt.

I Ninetechs grafiska profil fann vi regler för hur färg, texter samt hur deras logotyp skulle användas. Dessa regler tillämpades senare på vår applikation. När vår handledare gav den grafiska profilen till oss hade vi redan hunnit skapa en stor del av designen, dock inget som var svårt att ändra på.

Först bestämde vi oss för att ändra på den text vi hade i applikationen. Texten var bl.a. projekttiteln, senaste checkin och information om de misslyckade testerna. I den grafiska profilen kunde vi läsa att det rekommenderade teckensnittet var Myriad men eftersom vi inte hade det tillgängligt stod det att man kunde använda teckensnittet Calibri.

När ändringen utav all text var färdig fortsatte vi med ändring utav färger. Listen högt upp i applikationen fick Ninetechs marinblåa färg. Som statusindikator för toningen i den nedre delen hade vi tidigare använt en röd och en grön färg. Vi såg nu att den grafiska profilen innehåll både en grön och en röd färg (se figur 21) så vi bytte ut de färgerna också.



Figur 21: Dessa är två av de färger som finns att tillgå i Ninetechs grafiska profil

Regler för hur Ninetechs logotyp skulle användas fanns också i profilen, se figur 22. Det fanns några olika typer av positioner den kunde placeras på. Vi valde att placera den i det nedre högra hörnet. Vi valde samtidigt också att ge logotypen en transparent bakgrund. Den transparenta bakgrunden tillåter statusindikatorns toning att synas lite i bakgrunden av logotypen.

Figur 22: Logotypen förklaras som grundstenen i den grafiska profilen

Något som också tillhörde den grafiska profilen var en dokumentmall. Denna mall användes vid skapandet av lathunden. I mallen fann vi information om bl.a. placering utav text, textformatering, ett enkelt sidhuvud och sidfot.

3.2.7 Arbetssätt

Vårt projekt startade med ett möte med handledare från både Karlstads Universitet samt Ninetech. På mötet bestämde vi att vi skulle vara på Ninetechs kontor i början av varje vecka från måndag till onsdag.

På Ninetech fick vi tillgång till en arbetsplats och en dator. För att kunna logga in fick vi även ett varsitt användarkonto. Tillsammans med användarkontot fick vi även en varsin Ninetech-mejladress. Denna mejladress var bra att använda vid kommunikation med vår handledare på Ninetech. Eftersom att vi ibland skulle få viktig information som t.ex. lösenord via mejl så var det rekommenderat att vi använde Ninetechs istället för vår egna.

På vår dator var Microsoft Visual Studio installerat. Där fanns också viktiga komponenter som kom till användning när vi arbetade mot Team Foundation Server. Vår handledare hade också gett oss rättigheter så att vi kunde skapa ett eget Team Projekt på servern. Under projektets gång fick vi även läsrättigheter på andra Team Projekt. Läsrättigheterna gav oss möjlighet att hämta information om de andra projekten till vår applikation.

Vid det första mötet hade vi även bestämt att vi skulle ha ett kort avstämningsmöte med vår handledare på Ninetech varje måndag. Under dessa möten berättade vi vad vi gjort veckan innan och vad vi planerade att göra den kommande. Vi fick även hjälp med funderingar och

problem vi hade under dessa möten. När ungefär hälften av projektets tid hade gått höll vi även ett möte där vi fick hjälp att strukturera upp vår programkod.

När vi satt och arbetade på Ninetech hade vi också en person bredvid oss. Han var praktikant och det passade oss jättebra eftersom att han var bättre än oss på grafisk design och kunde därmed hjälpa oss lite med det när vi fick problem. Det är också han som har skapat de två bilderna "Success" och "Failed" åt oss.

Denna rapport har vi skrivit under den andra hälften av veckan, när vi inte suttit på Ninetech. Rapporten är det vår handledare på Karlstads Universitet som har hjälpt oss med. Ungefär varannan vecka höll vi ett möte på universitetet där vi gick igenom det vi skrivit och fick tips på vad som kunde förbättras och läggas till.

3.2.8 Skapande av ”Unit-test”

Eftersom att applikationen också skulle kunna användas till viss del som ett exempel på hur man använder tester inom projekt så var det ett krav att den skulle inkludera enhetstester.

När man skapar ett enhetstest så kan man välja om man vill skriva testet innan man skriver funktionen eller efter. Använder man testdriven utveckling så skriver man testet innan. Hursomhelst, under utvecklingen av vår applikation skrevs de flesta av testerna efter att funktionen var skapad.

3.2.9 Problem

Piska istället för Morot

Vår applikation är tänkt att uppmana utvecklarna på Ninetech att arbeta mer med tester och att få dem att känna att användandet av tester faktiskt kan gynna dem i framtiden. Applikationen ska därför fungera som en slags morot. Om en utvecklare gör bra ifrån sig så ska det alltså visas på skärmen som sitter synligt för alla runtomkring. Med detta vill vi få utvecklarna att sträva after att skärmen ska visa en bra status hela tiden.

En bra status innebär att alla tester lyckas och testtäckningen (code coverage) är hög. För att uppnå denna status krävs det alltså att utvecklarna skriver tester som täcker koden bra och att programkod skrivs som får testerna att lyckas.

I slutet av vårt arbete hade vi planerat in en liten demonstration av vårt program. Det var många på kontoret som undrade vad den svarta skärmen på väggen skulle visa. Under vår presentation fick vi frågan om vår presentation kanske skulle fungera som en piska istället för en morot. Den röda statusen vid "Partially Succeeded" kan faktiskt få en utvecklare att inte vilja skriva några tester . Ett test som inte går igenom kan få statusen att bli röd. Detta var en fråga som vi inte hade tänkt på innan.

Lösning:

När problemet diskuterades med vår handledare sa han att det inte gjorde någonting om applikationen fungerade som den nu gjorde. Att projektet visar en dålig status är inte bra och visas en för intetsägande bild finns risken för att utvecklarna inte lägger ner den tid som behövs för att förbättra testresultaten.

Tanken om att färre test ökar chansen för att få statusen "Succeeded" stämmer visserligen. Men vi tror inte att detta kommer att motverka att utvecklarna skriver nya tester. Detta eftersom att presentationen även framhäver testens täckningsgrad (code coverage) vilket utvecklarna också bör sträva efter att ha en hög andel av.

4 Skapandet av en Lathund

Detta kapitel beskriver vårt arbete med lathunden. Lathundens målgrupp samt hur den publicerades går även att läsa om här.

4.1 Skapandet

Uppgiften var att skapa en slags manual som beskriver hur personalen ska kunna få sina projekt att visas i applikationen. Förutom detta så skulle även en enklare beskrivning utav automatiska tester göras.

Innan lathunden skapades så samlades information in. Kunskap insamlad under projektets gång kom väl till nytta.

För att försäkra kvaliteten på lathunden gjordes tester som följde instruktionerna i den. Innan publicering korrekturlästes även lathunden utav handledaren på Ninetech.

När applikationen lanserats och flera projekt lagts till, upptäcktes även en del felaktigheter. För att lösa dessa fel var vissa projekt tvungna att ändras. Lathunden uppdaterades efter detta för att informera läsaren om dessa problem tillsammans med föreslagna lösningar.

4.2 Målgrupp

Målgruppen som denna lathund är inriktad åt är de personer som arbetar med Ninetechs olika projekt. Lathunden är främst inriktad på utvecklare som sitter och arbetar på samma våning som applikationen visas på.

Det är en fördel om läsaren har arbetat med Visual Studio innan. En annan fördel är om personen har grundläggande kunskap om Team Foundation Server. Hursomhelst går det att fullfölja beskrivningen utan denna kunskap.

4.3 Publicering

Lathunden publicerades på Ninetechs intranät. Detta intranät går att nås av alla som arbetar inom Ninetech. Tillsammans med lathunden publicerades även information om applikationens syfte.

5 Resultat

Detta kapitel kommer att ta upp det resultat vi upplevde när program och lathund lanserats. Eftersom att programmet kommer att utvärderas i några månader framöver är det i skrivande stund svårt att se ett tydligt resultat.

5.1 Applikation

Vid den första körningen på skärmen i kontoret upplevde vi många intresserade blickar. När applikationen lanserades på intranätet gick det att läsa flera nöjda kommentarer. Många verkade tycka att det var ett steg i helt rätt riktning. Vissa personer hade saknat någon lättillgänglig statuspresentering av projekten.

Vid lansering tillkom flera projektet som inte användes vid testningen av applikationen. I takt med detta upptäcktes även en del nya buggar.

Denna applikation planeras till en början att köras i några månader för att utvärdera dess användbarhet. Under denna period kommer presentationen att visas på en skärm på en av Ninetechs tre våningar. Skulle användandet efter denna period visa sig positiv, kan fler skärmar sättas upp på andra våningar. Applikationen kan även komma att utvecklas ytterligare.

5.2 Lathund

Lathunden lanserades även den på intranätet. Tyvärr hade vi inte möjlighet att på ett lätt sätt mäta hur många som har läst lathunden. Då flera projekt tillkommit till programmet kan vi dock anta att den har kommit till användning. Lathunden finns som bilaga i denna rapport.

6 Slutsats

Detta kapitel summerar våra tankar och åsikter på projektet. Vi ger några förslag på framtida förbättringar och föreslår andra användningsområden.

6.1 Framtida förbättringar

Under projektets gång har vi stött på många problem. De flesta av dessa har vi kunnat lösa men det finns dock några felaktigheter kvar. Dessa felaktigheter är dock inga som kommer att synas under normalt användande.

Ett av dessa fel är om två byggen utlösta av olika användare hinner göras mellan programmets uppdateringar. Är fallet som så att båda byggen misslyckas kommer applikationen visa att felet beror på den användare som utlöste det senaste bygget, istället för användaren innan. Detta går att förbättra genom att från servern hämta information om bygget som skedde innan det misslyckade bygget.

Vi kan även se att den struktur vi har på koden kan förbättras. När vi påbörjade utvecklingen av applikationen hade vi inte den kunskap om Team Foundation Server som vi har nu. Nu kan vi se att metoder och klasser skulle kunna se annorlunda ut och därmed bättre följa en viss standard.

Anslutningen till servern ligger nu nästlad på olika ställen i programkoden. Genom att bryta ut denna kod och placera den på en central plats skulle det vara lättare att anpassa programmet för fler olika serverversioner.

Team Foundation Server har ett brett stöd för s.k. workitems. De innehåller bl.a. buggar och uppgifter som behöver utföras (se kap.2.5.4). Vår applikation kan relativt enkelt anpassas för att även visa information om dessa. I nuläget verkar dock inte Ninetech använda sig utav workitems i så bred utsträckning.

I slutet av vår utveckling kan vi också se att det kanske hade varit ett bättre val att göra applikationen som en webbapplikation trots vår tidigare utredning (se kap.3.2). Att skriva om vår kod bör däremot inte vara alltför tidskrävande, då stora delar av vår kod går att återanvända.

6.2 Annan användning

De funktioner vår applikation har är inte direkt anpassade efter Ninetechs servrar. Med endast en ändring i anslutningslänken går det att få programmet att hämta data från en annan

server. Detta skulle även kunna vara till nytta för andra företag. Applikationen innehåller dock Ninetechs logotyp och följer deras grafiska profil.

6.3 Summering

Att arbeta med detta projekt har varit väldigt intressant. Att få använda de kunskaper vi har fått från Karlstads Universitet ute i arbetslivet har varit roligt. Projektet har ökat vårt kunnande inom Team Foundation Server och dess struktur mycket. Vi har också fått en inblick i hur arbetet sker i ett företag som Ninetech, och hur man arbetar inom projekt.

När vi blev tilldelade detta projekt var vi tveksamma om vi skulle klara av det. Det lät som en svår uppgift att hämta ut all information, speciellt då vi knappt hade hört talats om Team Foundation Server tidigare.

Efter att ha spenderat några dagar på Ninetech med att söka information, förstod vi att det fanns ett brett kunnande om detta på internet. Efter lite sökande hittades lösningar på några av de problem som oroade oss.

När vi väl var färdiga med projektet kunde vi känna oss stolta över ett väl utfört arbete. Även vår uppdragsgivare Ninetech var nöjd.

Referenser

[1] IEEE, *Standard Glossary of Software Engineering Terminology, Std. 610.12-1990*, 1990

[2] CIO Sweden, *Liv Marcks von Würtemberg*, (Besökt 15 maj 2012) http://cio.idg.se/2.1782/1.326833/darfor-floppade-projektentre-svenska-it-fiaskon-under-lupp

[3] Eric J. Braude, Michael E. Bernstein, *Software Engineering – Modern Approaches*, 2011 2nd edition

[4] Wikipedia, *Waterfall model*, (Besökt 15 maj 2012), http://en.wikipedia.org/wiki/Waterfall_model

[5] Ninetech AB, *Ninetech Gasellvinnare 2011*, (Besökt 10 maj 2012), http://www.ninetech.se/Om-Ninetech/Aktuellt/Nyheter/Ninetech-Gasellvinnare-20111/

[6] Agile Data, *Introduction to Test Driven Development (TDD),* (Besökt 25 april 2012), http://www.agiledata.org/essays/tdd.html

[7] Agile Modeling, *Introduction to Acceptance Test* (Besökt 30 april 2012), http://www.agilemodeling.com/artifacts/acceptanceTests.htm

[8] James W. Newkirk & Alexei A. Voronstov, *Test-Driven Development in Microsoft .NET*, 2004

[9] Hakan Erdogmus, National Research Council Canada, *On the Effectiveness of Test-first Approach to Programming*, Mars 2005

[10] Saravanan Subramanian, *Unit Testing 101 For Non-Programmers* (Besökt 20 mars 2012) http://www.saravanansubramanian.com/Saravanan/Articles_On_Software/Entries/2010/1/19_Unit_Testing_101_For_Non-Programmers.html

[11] Microsoft Developer Network, *Unit Testing*, (Besökt 20 mars 2012) http://msdn.microsoft.com/en-us/library/aa292197(v=vs.71).aspx

[12] Microsoft Developer Network, *Team Foundation Server Fundamentals: A Look at the Capabilities and Architecture*, (Besökt 2 maj 2012) http://msdn.microsoft.com/en-us/library/ms364062(v=vs.80).aspx

[13] Microsoft Developer Network, *Team Foundation Server Architecture*, (Besökt 18 mars 2012) http://msdn.microsoft.com/en-us/library/ms252473.aspx

[14] Brian Harry MS, *Team Foundation Server 2010 Key Concepts*, (Besökt 6 mars 2012) http://blogs.msdn.com/b/bharry/archive/2009/04/19/team-foundation-server-2010-key-concepts.aspx

[15] Microsoft Developer Network, *Team Foundation Server Reporting*, (Besökt 7 mars 2012) http://msdn.microsoft.com/en-us/library/ms194922(v=vs.80).aspx

[16] Microsoft Developer Network, *Walkthrough: Tracking work Items*, (Besökt 4 maj 2012) http://msdn.microsoft.com/en-us/library/ms181269(v=vs.80).aspx

[17] Microsoft Developer Network, *Team Foundation Server Reporting*, (Besökt 2 maj 2012) http://msdn.microsoft.com/en-us/library/ms194922(v=vs.80).aspx

[18] Brian Randell, MSDN Magazine, *Essential Power Tools*¸ (Besökt 10 mars 2012) http://msdn.microsoft.com/en-us/magazine/cc721612.aspx

[19] Aniruddha Chakrabarti, Aniruddah's WebSpace, *TFS Build Notification in TFS Power Tool*, (Besökt 20 februari 2012) http://caniruddha.wordpress.com/2010/02/08/tfs-build-notification-in-tfs-power-tool/

[20] Mark Michaelis, IntelliTect, *Subscribing to TFS Alerts with TFS Power Tools' Alert Explorer*, (Besökt 23 februari 2012) http://intellitechture.com/subscribing-to-tfs-alerts-with-tfs-power-tools-alerts-explorer/

[21] Ed Blankenship, Martin Woodward, Grant Holliday & Brian Keller, *Team Foundation Server 2010*, 2011

[22] Microsoft, *C# Language Specification*, *version 4.0*, 2010

[23] Magnus Nilsson, Patrik Johansson, Anders Petersson & Rikard Thunberg, Linköpings universitet, *Appliktioner på webben*, 1999

[24] Microsoft Developer Network, *Windows Forms*, (Besökt 20 februari 2012) http://msdn.microsoft.com/en-us/library/dd30h2yb.aspx

A Lathund för Ninetech-Testboard

Innehåll

Hur man får sitt projekt att visas i Ninetech Testboard

Automatiska Byggen

Med hjälp utav automatiska byggen kan man låta en server kompilera källkod och köra tester på en applikation automatiskt. Vanligtvis kan utvecklaren göra detta lokalt på sin dator, men vad händer om denna kod inte fungerar tillsammans med annan kod i projektet? Automatiska byggen kan försäkra funktionaliteten hos ett projekt genom att testa all programkod samtidigt.

Microsoft ser automatiska byggen som en betydelsefull del vid mjukvaruutveckling och har valt att inkludera funktionalitet för detta i Team Foundation Server. Så fort en utvecklare har anslutit till ett projekt i Visual Studio kan den se projektets senaste byggstatus i Team Explorer.

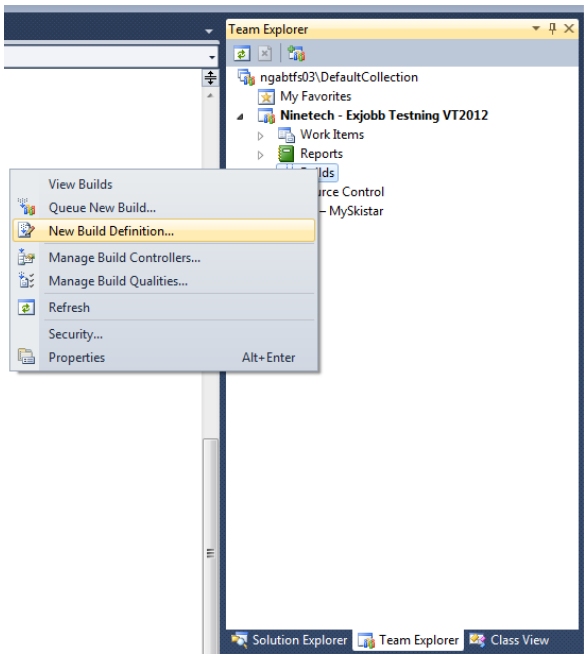
Innan Team Foundation Server 2010 introducerades kördes byggen på en enda server, kallad Build Agent. När den nya versionen kom introducerades begreppet Build Controller. Denna komponent möjliggör att flera Build Agents kan arbeta tillsammans och fördelar arbetsbördan mellan dessa.

Det finns flera händelser som kan användas för att trigga ett nytt bygge. Ofta vill man att ett bygge triggas så fort ny programkod laddas upp till servern, men händer det för ofta kan det belasta byggservern i onödan. För att inte framkalla onödig belastning kan man då istället schemalägga byggen på natten då ingen arbetar med koden. Det går givetvis att använda båda alternativen samtidigt och man kan också trigga nya byggen manuellt eller genom en rad andra händelser.

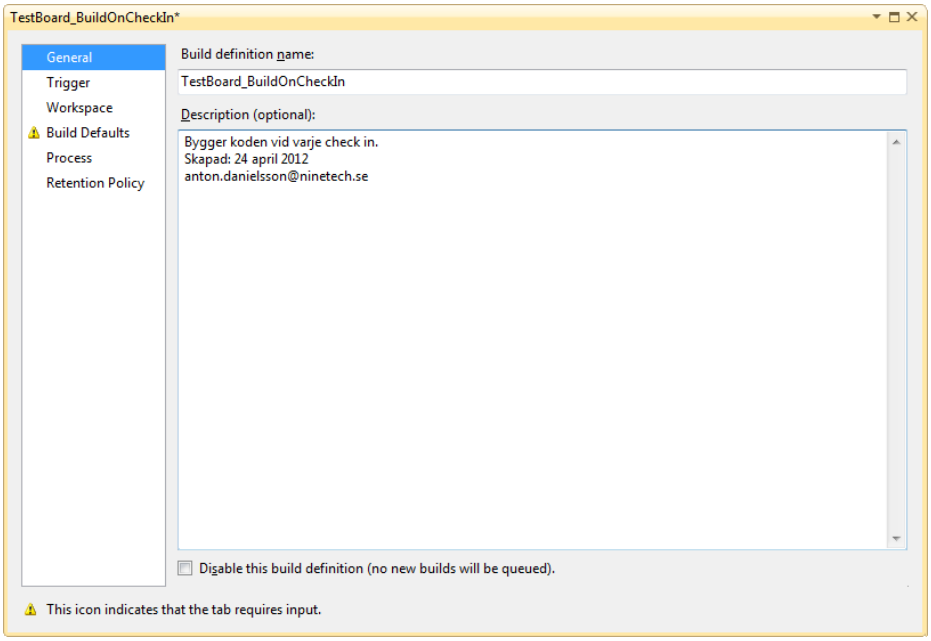
För att en server automatiskt ska bygga en applikation och köra tester måste en Build Definition skapas. Definitionen specificerar när ett bygge ska ske och hur det ska gå till. Det går att ha flera definitioner till samma projekt.

Att skapa en Build Definition

Informationen som visas i Ninetech Testboard hämtas från projektens senaste byggen. Därför måste en Build Definition vara skapad för varje projekt som ska visas i applikationen. En sådan skapas genom att högerklicka på ”Builds” under ”Team Explorer” och välja ”New Build Definition”.



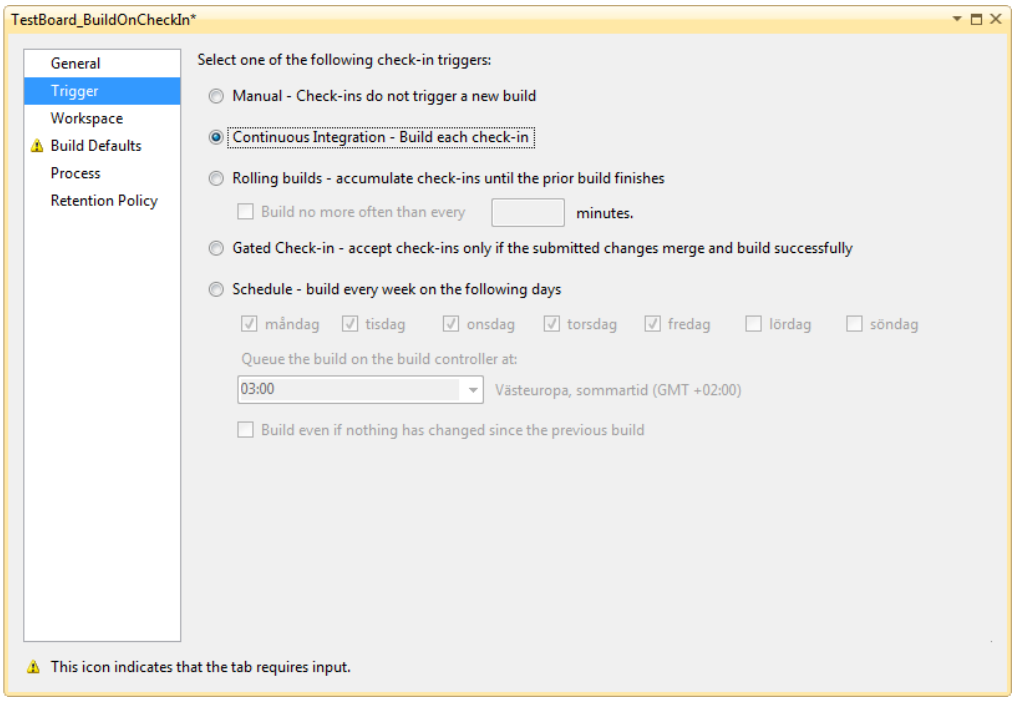
Under ”General” tilldelas Definitionen ett namn. Eventuellt anges även en beskrivning av den.



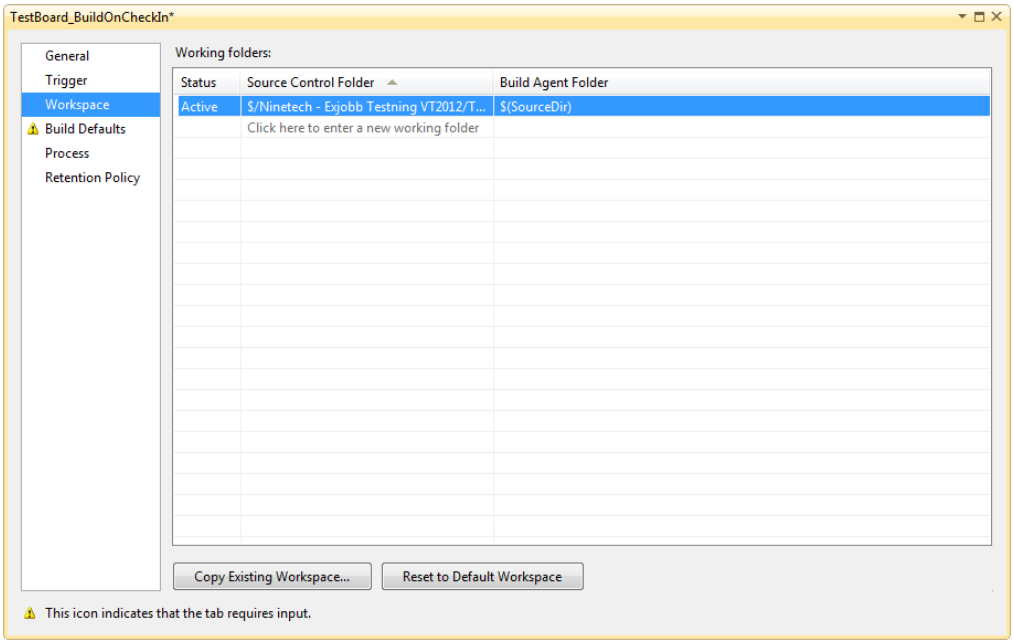
Under ”Trigger” bestäms vad som ska trigga ett nytt bygge. Det finns 5 olika triggers att välja mellan:

- Manual – Servern bygger inte programmet när kod checkas in.
- Continuous Integration – Programmet byggs varje gång kod checkas in.

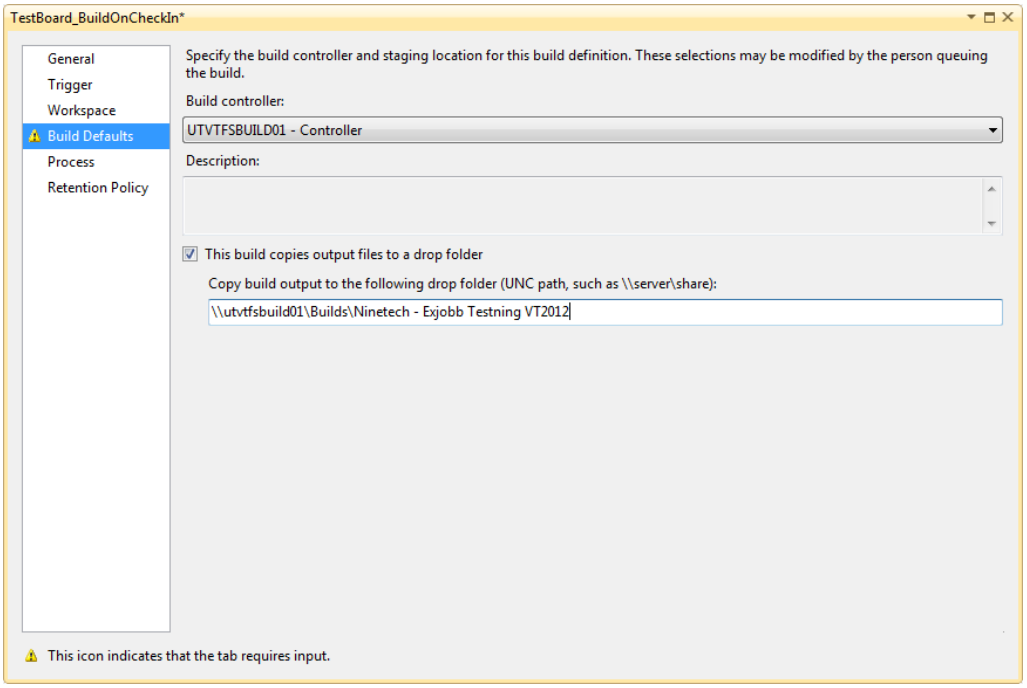
- Rolling builds – Servern bygger programmet när kod checkas in, men här går det välja en minimum tid mellan varje bygge.
- Gated Check-in – Servern försöker bygga programmet, fungerar detta accepteras den incheckade koden, annars förkastas den.
- Schedule – Här finns möjligheten att välja vid vilken tidpunkt servern ska bygga programmet.



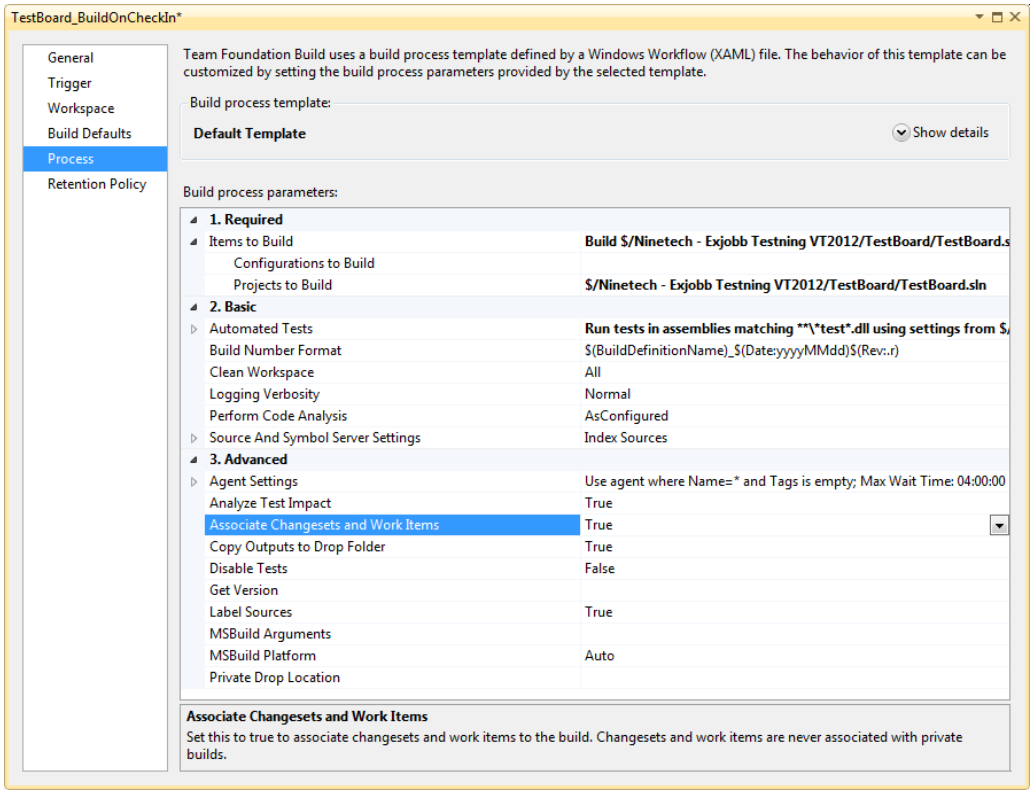
Under rubriken ”Workspace” bestäms vilken mapp som ska byggas. Ofta fungerar det bra att använda den förinställda, som är projektets rot-mapp. I ett större projekt kan detta dock vara bra att välja en lite mer specifik mapp. Att ange en för brett omfamnande mapp kan leda till att servern gör ett bygge trots att incheckade filer inte påverkar utgången utav det.



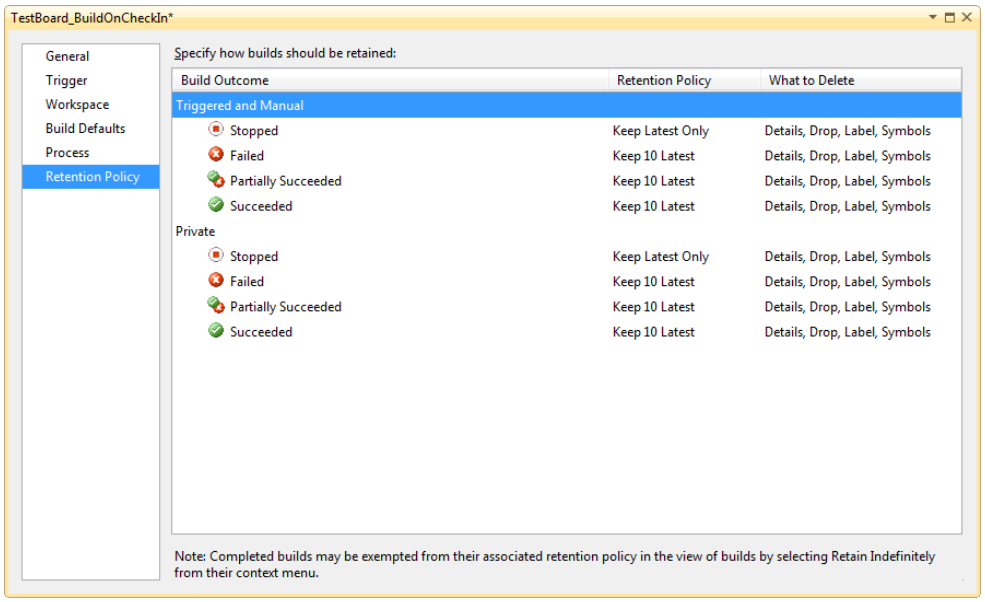
Under ”Build Defaults” anges vilken ”build controller” som ska utföra bygget. Denna komponent bestämmer vilken server som ska användas för att bygga projektet. Här bestäms också vilken mapp de byggda filerna ska hamna i. Förslagsvis ” \\utvtfsbuild01\Builds\{Projektets namn}” (se bild nedan).



Under ”Process” sektionen specificeras vilken process som ska användas vid bygget. Det går att skapa egna men det finns också en standardprocess som går att använda. Här går det specificera om tester och analyser ska göras och om loggning ska ske bl.a. För att Ninetech-Testboard ska kunna hämta all information om den senaste ändringen krävs att ”Associate Changesets and Work Items” är satt till ”True”.



Under ”Retention Policy” anges hur många av projektets byggen som ska sparas. För Ninetech-Testboard fungerar det bra att välja standarinställningarna.

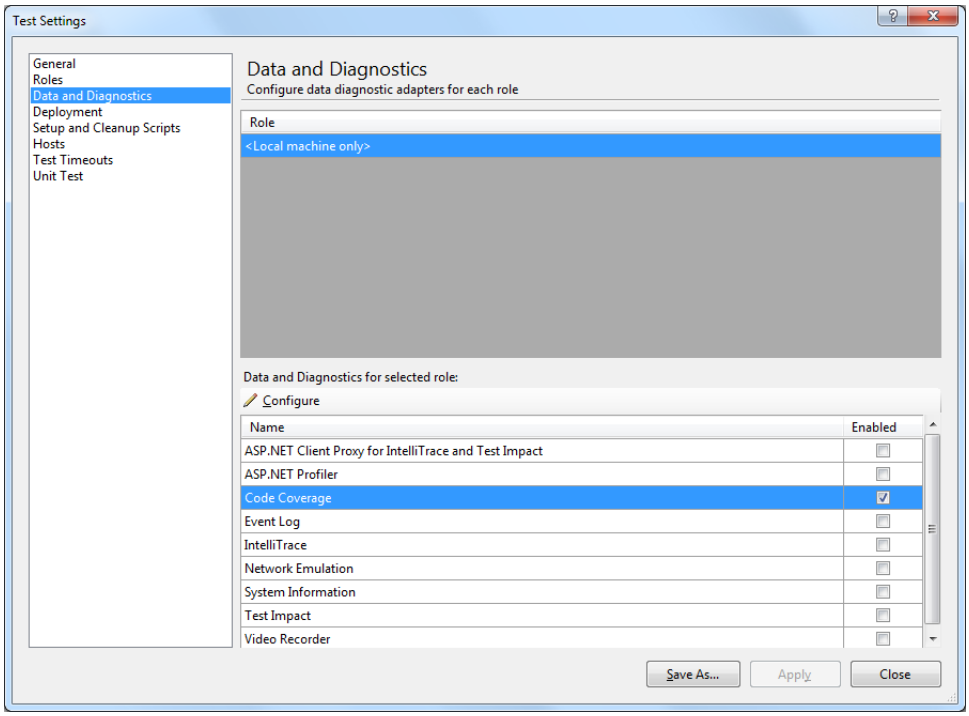


OBS!

Ibland kan Ninetech – Testboard visa att inget changeset finns tillgängligt. En orsak till detta kan vara att den skapade byggdefinitionen måste ha triggat ett lyckat bygge (successful) innan associering kan ske. Detta eftersom att nya changesets jämförs med det senaste lyckade bygget. Finns inget lyckat bygge sedan tidigare sker ingen associering.

Aktivera Code Coverage

För att Code Coverage ska visas korrekt I Ninetech-Testboard krävs att TFS analyserar koden efter det. För att aktivera detta, öppna ditt projekt i Visual Studio. Gå till ”Solution Explorer” -> Expandera ”Solution Items” -> Dubbelklicka på ”Local.testsettings”. Följande fönster visas:

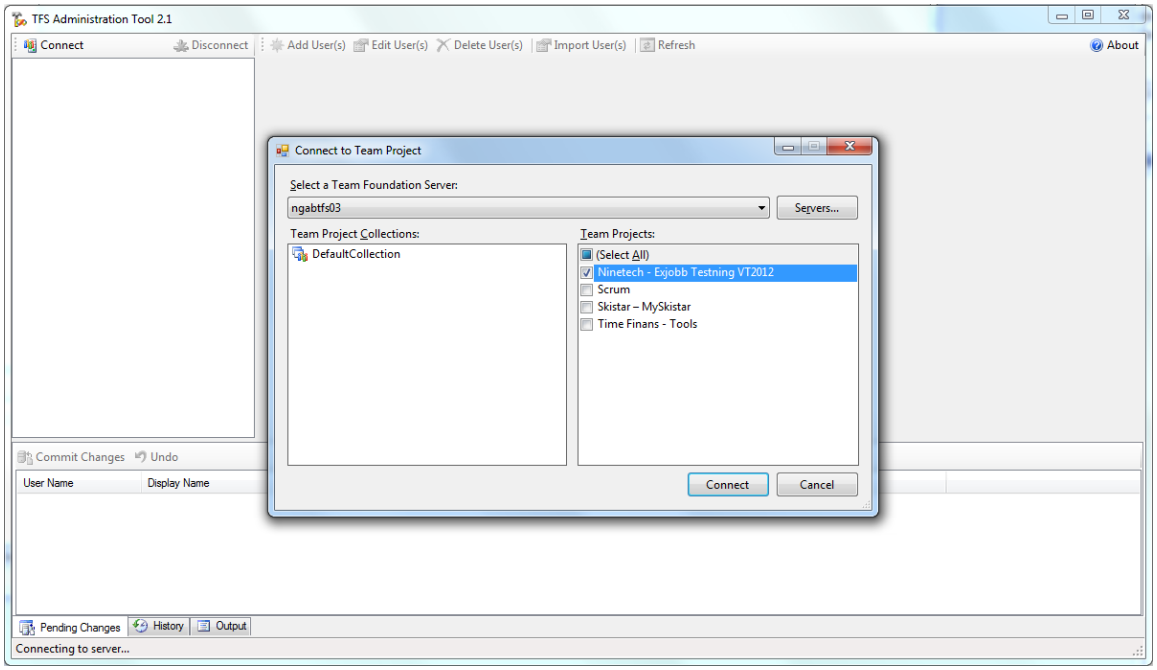


Klicka sedan på “Data and Diagnostics” och aktivera Code Coverage.

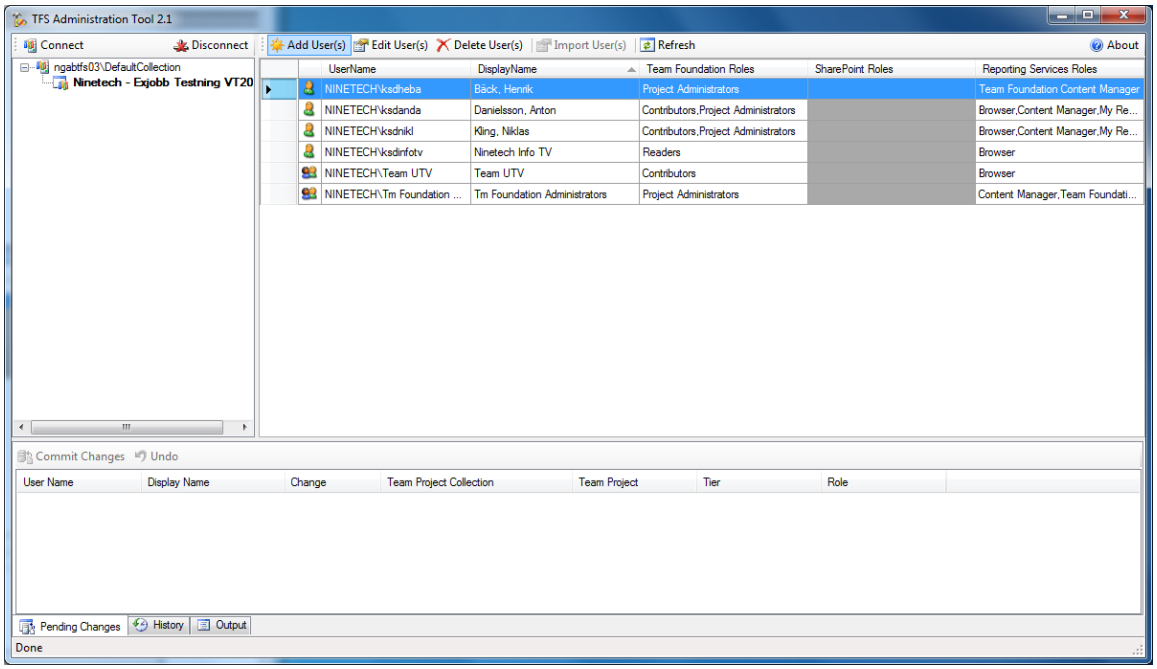
Ge läsrättigheter åt applikationen

För att info-Tv:n ska kunna komma åt projektets information och visa den i Ninetech-Testboard måste läsrättigheter först tilldelas. Detta görs i programmet ”TFS – Administration Tool”. Är programmet inte redan installerat finns det att hämta på http://tfsadmin.codeplex.com/. Endast projektets administratör kan ge dessa rättigheter.

När programmet startat, klicka på ”Connect” uppe i det vänstra hörnet. Om ingen server visas i listen har ingen server ännu lagts till (Servern hämtas automatiskt från Visual Studio). Klicka då på ”Servers…” och i det nya fönstret ”Add…”. Här skrivs Team Foundation Serverns namn eller adress. Adressen till TFS 2010 är ”http://ngabtfs03”, portnumret är ”8080” och path sätts till ”tfs”. Välj det berörda projektet och klicka på Connect.



Här visas de användare som är associerade med projektet. Klicka på ”Add User(s)” för att lägga till eller ändra rättigheter för en användare.



Skriv in ”ksdinfotv” i textrutan och klicka på ”Add”. Markera användaren och kryssa i rutan ”Readers” och klicka på OK. Glöm ej att klicka på ”Commit Changes”. Projektet går nu att se i info-Tv:n.