



# Mittuniversitetet

---

MID SWEDEN UNIVERSITY

## **Laboration 1**

**Sebastian Strindlund**

sest1601

Programvaruteknik  
Mittuniversitetet

## Sammanfattning

Denna rapport behandlar relativt ytligt ett antal olika utvecklingsmetoder för programvara, ett par verktyg för testning av programvara, några stödjande verktyg (exempelvis verktyg för versionshantering) samt ett par dokumentationsverktyg för programvara.

Informationen kommer att tas fram genom att själv eftersöka information från olika webbsidor. Informationen sammanställs och i varje enskild kategori kommer metoder/verktyg jämföras baserat på ett antal olika kriterier.

Utvecklingsmetoderna som berörs är de moderna agila metoderna Scrum och Kanban, den klassiska vattenfallsmodellen och Chaos model. Vattenfallsmodellen är väldigt strikt och kräver väldigt mycket planering innan man kör igång eftersom att i denna modell så går man inte vidare till nästa steg i utvecklingen förrän ett steg är helt klart och godkänt. I dem agila metoderna däremot så kör man mer flexibelt och kan gå igenom planeringsfasen fler gånger.

Om två testverktygen som behandlas är både unit testverktyg i olika programmeringsspråk där dokumentation, användarvänlighet och programmeringsspråk tas upp. Båda verktygen hade bra dokumentation men resultatet för min del blev att unittest var lite enklare att förstå, lite enklare att implementera och lite enklare att använda. Båda verktygen fungerade dock bra och efter förväntan med den lilla kunskap jag har om dem.

I stödjande verktyg tas två olika versionshanteringsverktyg upp, Git som är ett distribuerat system och RCS, en version control system. Även ett verktyg för kommunikation tas upp, nämligen Discord, ett VOIP verktyg. Git verkar vara det bättre verktyget eftersom det är mer uppdaterat, har stöd för att förändra ett helt träd filer på och samma gång, stöd för arbete över nätverk, jämfört med RCS som endast kan förändra en fil åt gången och endast stödjer arbete över samma filsystem.

Sist ut har vi ett par verktyg för dokumentation utav programvaran. Doxygen är ett verktyg som kan generera dokumentation utifrån källkod i ett antal olika programmeringsspråk, C++ och Java bl.a. Det andra verktyget heter NDoc och kan endast generera dokumentation för C# och inte direkt genom källkoden utan istället från en assembly fil som skapas vid kompilering av koden.

## Table of Contents

1 Inledning.....	1
1.1 Syfte .....	1
1.2 Problemformulering .....	1
2 Metod .....	1
2.1 Jämförelsekriterier.....	1
2.1.1 Utvecklingsmetoder .....	1
2.1.2 Testverktyg.....	1
2.1.3 Stödjande verktyg.....	2
2.1.4 Dokumentationsverktyg .....	2
3 Resultat.....	2
3.1 Utvecklingsmetoder .....	2
3.1.1 Scrum .....	2
3.1.2 Kanban .....	3
3.1.3 Vattenfallsmodellen .....	4
3.1.4 Chaos model.....	4
3.2 Testverktyg.....	5
3.2.1 Unittest .....	5
3.2.2 Catch.....	6
3.3 Stödjande verktyg.....	6
3.3.1 Git.....	6
3.3.2 RCS .....	7
3.3.4 Discord .....	7
3.4 Dokumentationsverktyg .....	7
3.4.1 Doxygen .....	7
3.4.2 NDoc .....	8
4 Diskussion .....	8
4.4.1 Utvärdering av utvecklingsmetoder .....	8
4.4.2 Utvärdering av testverktyg .....	9
4.4.3 Utvärdering av stödjande verktyg .....	9

4.4.4 Utvärdering av dokumentationsverktyg .....	10
5 Referenser.....	11

# 1 Inledning

## 1.1 Syfte

Syftet med laborationen är att få en överblick utav dem vanligaste metoderna som används för utveckling av mjukvara, vanligaste metoderna för testning av programvara, stödjande verktyg som underlättar utvecklingsprocessen och dokumentationsverktyg för automatisk generering av dokumentation från programkod. Samt övning på att skriva rapport.

## 1.2 Problemformulering

- Ta reda på vad det finns för olika metoder av utveckling av mjukvara och välja ut fyra stycken att ta reda på mer om
- Leta upp metoder för testning av programvara, ladda ned och testa två stycken.
- Hitta verktyg som finns för att underlätta utvecklingsprocessen och samarbete i utvecklingen av projekt.
- Kolla upp verktyg som kan användas för att generera dokumentation utifrån programkod och ta reda på mer information om två stycken.
- Välja ut jämförelsekriterier och jämföra dem olika metoderna och verktygen i varje kategori.

## 2 Metod

Efterforska på källor som Lektion 1 – Introduktion har använt i sitt dokument. Hitta utvecklingsmetoder, testmetoder, stödverktyg och dokumentationsverktyg, sammanställa informationen och sedan göra en jämförelse i alla fyra kategorier.

### 2.1 Jämförelsekriterier

#### 2.1.1 Utvecklingsmetoder

Jämförelse av utvecklingsmetoder sker på följande kriterier:

- Vad skiljer de agila metoderna åt.
- Agila mot sekventiella metoder.
- Agila och sekventiella mot andra metoder.

#### 2.1.2 Testverktyg

Jämförelse av testverktyg sker på följande kriterier:

- Språk verktyget stödjer
- Dokumentation
- Användarvänlighet

### 2.1.3 Stödjande verktyg

Jämförelse av stödjande verktyg sker på följande kriterier:

- Operativsystem som verktygen stödjer.
- Senaste stabila release av verktyget.
- I vilket scope verktyget kan förändra filer.
- Sammarbetsmöjligheter.

### 2.1.4 Dokumentationsverktyg

Jämförelse av dokumentationsverktyg kommer ske på följande kriterier:

- Vilka operativsystem verktygen stödjer.
- När senast aktuella release av verktyget släpptes.
- På vilket sätt tar verktygen in data för att kunna generera dokumentation.
- Vilka programmeringsspråk verktyget stödjer
- Vilket format som dokumentationen genereras

## 3 Resultat

### 3.1 Utvecklingsmetoder

#### 3.1.1 Scrum

[Scrum\[1\]](#) är en agil metod som fokuserar på att hantera arbetsflödet och används ofta till mjukvaruutveckling. Det scrum gör är att den delar upp arbetet i mindre hanterbara bitar som kan utföras av mindre team inom en bestämd tidsram. Metoden är designad för att köras med utvecklare i grupper om 3-9 personer som kör sprinter om 1-4 veckors cykler där man efter varje sprint levererar fungerande programvara. Scrum kräver att minst tre roller är etablerade i projektet (produktägare, scrum master och utvecklingsteam) men det kan finnas ett flertal roller och beståndsdelar i skepnad av obligatoriska möten och dokument.

**Produktägaren** är en fysisk person som representerar intressenterna dvs kundernas ”röst”, hanterar och prioriterar tillägg, ändringar för produkten, håller intressenterna uppdaterade, sköter kommunikationen med resten av företaget, försäkrar att produktens backlog är tydlig och finns tillgänglig.

**Scrum master** agerar bland annat buffer mellan utvecklingslaget och distraktioner, ser till att ramverket för Scrum följs, coachar och håller lagets moral uppe

**Utvecklingsteamet** är ansvariga för att leverera en produkt i slutet av varje sprint. Laget består som sagt av 3-9 personer som kan analysera, designa, utveckla, testa, dokumentera osv. Kort sagt skall teamet tillsammans ha kunskapen som krävs för att kunna leverera en produkt.

Scrum består av flera beståndsdelar, bland annat ett dokument som kallas för **Product backlog** som är en samling utav alla önskade förändringar på produkten detta dokument användes sig av prioritering dvs, högre prioritet, kräver högre specifikation på ändringsönskemålet. Ett annat obligatoriskt dokument är en **Sprint backlog**. Denna backlog innehåller det som utvecklingsteamet åtagit sig att implementera under aktuell sprint, samt planeringen för hur det skall implementeras.

Med **sprint** menas att arbetet delas in i sprintar (iterativa) som är 1-4 veckor långa cykler. Varje sprint startas genom att man planerar vilket arbete som krävs och hur prognosen ser ut för ungefär vad som rimligtvis kan avklaras inom tidsramen för sprinten. En **daily sprint** är ett möte på cirka femton minuter som hålls på samma tid och samma plats varje dag. Vanligtvis så svarar utvecklarna vid varje möte på frågor som; ”Vad tillförde jag till sprintens mål igår?”, ”Vad planerar jag att tillföra till sprintens mål idag?” och ”Ser jag något eventuellt hinder som kan hindra mig eller teamet från att nå sprintmålet?”. Sprinten avslutas med en sprintöversikt där framsteg visas för intressenter och antecknar eventuella nya krav och förbättringar som kan göras i nästa sprint.

### 3.1.2 Kanban

[Kanban\[2\]](#) hanterar arbetsflödet och delar också upp arbetet i mindre hanterbara delar precis som Scrum. Metoden visualiserar arbetets flöde så att flaskhalsar skall upptäckas och efterfrågan balanseras efter kapacitet istället för att mer och mer arbete skjuts in när det kommer nya önskningar. Att hålla sig till ett visst antal pågående arbeten och ett visst antal saker på att-göra-listan så ska arbetsavfall reduceras genom att det blir mindre multitasking och kontextbyten som laget behöver göra. Kanban grundar sig i ett par principer; förändringshantering och leverera bra service, vilket betonar förändring och kundfokus. Metoden har inte specifika steg på samma vis som Scrum utan utgår istället från befintliga sammanhang där man förespråkar kontinuerliga, inkrementella och evolutionära förändringar i produkten. Kanban fokuserar på ett mycket synligt arbetsflöde så att alla i laget hålls väl uppdaterade och som tidigare nämnts ligger också fokus på kunden och att möta dess behov. Kanban har inga roller som krävs att man bestämmer iförväg utan det kommer eftersom i och med vad projektet har för behov.

### 3.1.3 Vattenfallsmodellen

[Vattenfallsmetoden\[3\]](#) är en sekventiell metod där man strikt arbetar i en bestämd ordning. Man arbetar inte alls med lika bra kommunikation som man gör i agila metoder istället kan man sitta instängd själv på sitt kontor och arbeta. Den bestämda ordningen går som ett vattenfall (uppiifrån och ned utan återvändo) där man börjar med att gå igenom krav på system och mjukvara. När det är gjort så går man vidare till analysfasen för att kolla vad som önskas, vad som behövs för att kunna uppnå detta och dokumenterar det. När det steget är klar påbörjas designfasen som involverar problemlösning, algoritmer och mycket mera. Hur kan problemet lösas bäst? När det steget är klart så går vi in i implementeringsfasen. Här gör man verklighet av det man fått fram genom sin analys och designfas. Man producerar och tar fram en körbar version av den planerade applikationen. Nästa steg är testing, här testas applikationen systematiskt för att hitta buggar, kontrolleras att allt fungerar som det ska, hitta saker som kan optimeras med mera. Det är troligt att en bugg kommer hittas och enligt vattenfallsmodellen går man egentligen inte tillbaka ett steg men hittar man en bugg måste man självfallet gå tillbaka ett steg för att åtgärda buggen. Nu är det dags för det sista steget som är installation/underhåll, här installeras/migreras och underhålls färdiga system.

### 3.1.4 Chaos model

[Chaosmodellen\[4\]](#) arbetas fram eftersom att skaparen tyckte att andra modeller såsom spiralmodellen och vattenfalls modellen inte var tillräckligt kompletta. Dessa två metoder är bra på att hantera scheman och planering så saknar dessa bra metoder för att lösa buggar och andra tekniska problem. Skapar tyckte också att metoderna som var bra på att fixa buggar och tekniska problem då istället saknade bra funktioner för att hantera schema, deadlines och kunders förfrågningar. Meningen med Chaos model är att den skall övervinna det som saknas i andra modeller. Generellt sett så kan man se på modellen på samma vis som programmerare arbetar när slutet på ett projekt närmar sig. Man har en lista med buggar att fixa och funktionalitet att lägga till, dessa kvarvarande uppgifter fixas en åt gången

Modellen bygger på att **alltid** lösa det allra viktigaste problemet först. Problem reffererar till en ofärdig programmeringsuppgift som behöver göras klar. Det viktigaste problemet är en kombination utav stora, brådskande och robusta problem. Brådskande problem betyder att dem måste lösas direkt, annars kan arbetet stå stilla för projektet. Stora problem kommer att vara värdefulla för användarnas funktionalitet och robusta problem är inte lika viktiga utan testas när dem blir lösta, här läggs minst vikt. Ett problem är löst först när det är stabilt.

I Chaos model så säger man att livscykeln kan appliceras på alla nivåer i projektet allt ifrån bara en rad kod till hela projektet i stort sett.



Följande måste vara definerat, implementerat och integrerat;

- Hela projektet i sig
- Systemen
- Modulerna
- Funktionerna
- Alla rader kod

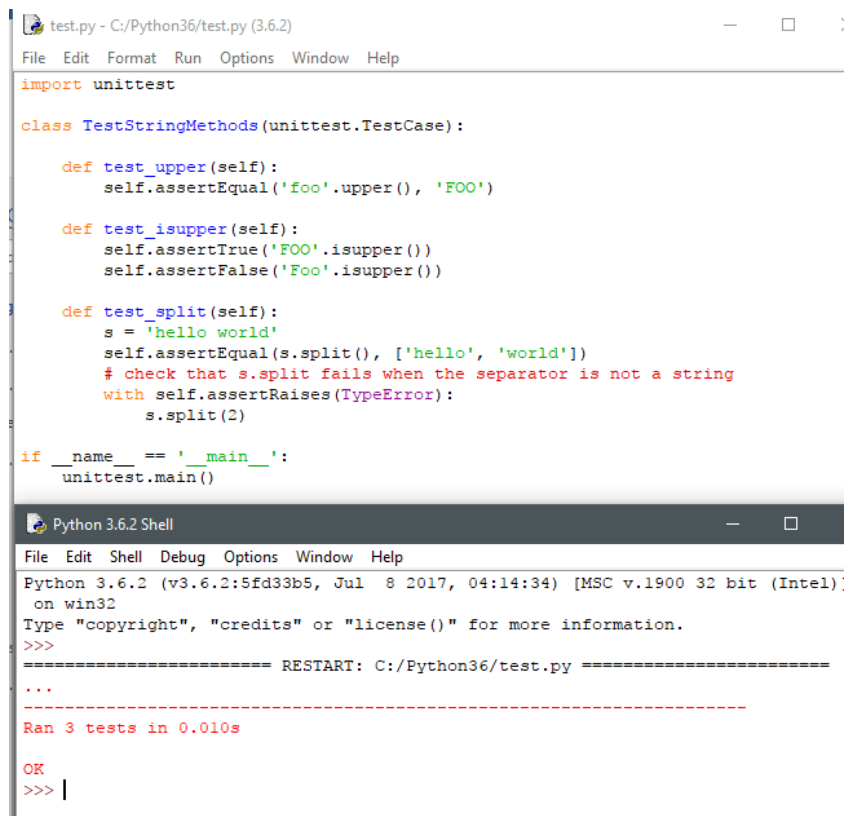
## 3.2 Testverktyg

### 3.2.1 Unittest

[Unittest\[5\]](#) är Pythons unit testing ramverk som är motsvarigheten till Javas version JUnit. Verktöget har stöd för testautomatisering, uppsättning och nerstängning av tester och aggregering utav samlingar av test. Unittest har stöd för flera viktiga testningskoncept på ett objektorienterat sätt.

- En testfixtur som förbereder för att kunna utföra ett eller flera test, uppställning som kan behövas i samband med tester. Exempel på förberedelser kan vara att skapa temporära eller proxy databaser, serverprocesser eller mappar.
- Ett testfall som är själva testenheten. Enheten väntar på ett specifikt resultat som den bör få utifrån en specifik inmatning i testet.
- En test suite vilket är en samling utav testfall, andra samlingar av testfall eller båda. Detta används för att aggregera tester som är menade att utföras tillsammans.
- En testkörningskomponent som styr exekveringen utav testerna och sammanställer resultatet av testerna åt användaren.

Importerade och körde ett testexempel av unittest i python och det fungerade väl, var inte allt för komplicerat och resultatet blev det väntade. Unittest verkar stödja minst både testmoden blackbox och whitebox [\[6\]](#) beroende på hur testerna skrivs i Unittest och vem som kör testerna.



```

test.py - C:/Python36/test.py (3.6.2)
File Edit Format Run Options Window Help

import unittest

class TestStringMethods(unittest.TestCase):

    def test_upper(self):
        self.assertEqual('foo'.upper(), 'FOO')

    def test_isupper(self):
        self.assertTrue('FOO'.isupper())
        self.assertFalse('Foo'.isupper())

    def test_split(self):
        s = 'hello world'
        self.assertEqual(s.split(), ['hello', 'world'])
        # check that s.split fails when the separator is not a string
        with self.assertRaises(TypeError):
            s.split(2)

if __name__ == '__main__':
    unittest.main()

```

```

Python 3.6.2 Shell
File Edit Shell Debug Options Window Help

Python 3.6.2 (v3.6.2:5fd33b5, Jul 8 2017, 04:14:34) [MSC v.1900 32 bit (Intel)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Python36/test.py =====
...
-----
Ran 3 tests in 0.010s

OK
>>> |

```

Figur 1 Skärmbild från test med unittest

### 3.2.2 Catch

På sidan [Catch\[7\]](#) under rubriken "What's the Catch?" så står det att Catch står för "C++ Automated Test Cases in Headers". Detta är ett automatiskt testramverk unit testing inom C++, C och Objective-C. Ramverket implementeras genom att man inkluderar en headerfil. Även Catch verkar stödja minst testmetoden blackbox och whitebox beroende på hur testet är skrivet och vem som kör testet. Laddade ned headerfilen och gjorde ett simpelt test på en multiplikationsfunktion. Resultatet blev det förväntade även om det var lite klurigare att förstå sig på dokumentationen jämfört med unittest.

## 3.3 Stödjande verktyg

### 3.3.1 Git

[Git\[8\]](#) är ett opensource system som används för versionshantering av källkod. Det betyder att systemet håller koll på alla filer i ett projekt och exakt vad som läggs till eller tas bort i varje version av projektet. Git är väldigt bra när man är flera personer som arbetar på ett projekt eftersom Git håller koll på alla ändringar som alla olika personer gör. Går något fel i projektet

kan man enkelt bara gå vidare till en tidigare version och se vad som gått fel alternativt bara börja om från den versionen. Eftersom git är ett distribuerat versionshanteringssystem så är det riktat mot hastighet, dataintegritet och stöd för distribuerad icke-linjära arbetsflöden. Det innebär även att varje Git folder på alla datorer även är ett fullt utrustat repo som innehåller all historik och har koll på allt som hänt i varje version av filerna. Git kan om man vill köras genom nätverk och utöver internet för att möjliggöra samarbete över hela jordklotet.

[https://en.wikipedia.org/wiki/Revision\\_Control\\_System](https://en.wikipedia.org/wiki/Revision_Control_System)

### 3.3.2 RCS

[RCS\[9\]](#) (Revision Control System) Är precis som git ett system för versionshantering. RCS fungerar endast på enskilda filer. Med andra ord så går det inte att bevaka ett helt projekt och stödjar alltså inte så kallade "atomic commits" (påverka flertalet filer).

Systemet bygger på att man använder sig utav revisionsgrupper eller ett set av filer som checkas in via check-in, checkout kommandon. När en fil checkas ut så ersätts den med en .v fil som innehåller huvudfiles metadata. När filen checkas in så återställs revisionerna i en trädstruktur som kan följas och möjlighet att återställa filen till ett tidigare stadi är möjligt.

Systemet är enkelt att använda och revisionerna är inte beroende på av ett centralt repository. Nackdelen med RCS är att endast en användare kan arbeta med varje fil åt gången.

### 3.3.4 Discord

[Discord\[10\]](#) är en freeware Voice-over-IP application som fungerar på alla plattformar (pc, mobiltelefoner och även som en webapp). Applikationen är främst inriktad mot att användas som kommunikationsmedel när man spelar dataspel. Applicationen hade 25 miljoner användare från hela världen i december 2016. Utöver vanlig VOIP så kan man även använda textbaserad chat. Vem som helst kan skapa en Discordserver gratis och innuti denna server kan man skapa ett stort antal "kanaler" som man bestämmer skall handla om olika specifika ämnen, precis som ett forum fast mer interaktivt eftersom det är "live" och du kan se i realtid vilka andra användare som är online och håller på att skriva just nu.

## 3.4 Dokumentationsverktyg

### 3.4.1 Doxygen

[Doxygen\[11\]](#) är alltså ett fritt programvaruverktyg för att kunna referensdokumentera programvara, verktyget kan köras på de flesta operativsystemen (Windows, OSX, Linux, BSD och Unix) och stödjer dem flesta språken för (C, C++, Java, C# m.fl.)[\[12\]](#). Utifrån källfiler så genererar Doxygen automatiskt dokumentation. I och med att dokumentationen skrivs

tillsammans med koden så är den tämligen enkel att hålla väl uppdaterad. Användare kan utan problem läsa dokumentationen och referera till koden eftersom att Doxygen hänvisar till koden i dokumentationen. Doxygen stödjer även dokumentationstagggar och kan generera sin dokumentation i HTML, RTF, PDF, LaTeX, PostScript format och även såkallade "man pages".

### 3.4.2 NDoc

[NDoc\[13\]](#) är likaledes Doxygen, ett fritt programvaruverktyg under GPL licensen som kan generera dokumentation till C# kod. Istället för att generera direkt utifrån källkoden så kan NDoc generera på två andra sätt. Det ena sättet är att generera från en assembly fil som skapas vid kompilering av källkoden. Genom assemblyfilen så får man tag i listor med klasser, metoder med mera. Andra sättet är via en förgenererad dokumentationsfil i XML formatet som vanligtvis skapas genom att parse källkoden och plocka ut speciella kommentarer, dokumentationstext det vill säga. NDoc verkar till skillnad från Doxygen inte stödja lika många operativsystem utan endast Windows fungerar. NDoc kan generera sin dokumentation till HTML och CHM format.

Vid stöd för flera språk så är Doxygen bäst även här med stöd för C, C++, Java, C# med flera till skillnad från NDoc som endast stödjer C#. [\[14\]](#)

## 4 Diskussion

### 4.4.1 Utvärdering av utvecklingsmetoder

Om vi börjar med att titta närmare på Scrum och Kanban som är de agila metoderna jag skrivit om så har Scrum minst tre roller som absolut måste fyllas för att arbetet skall bli effektivt (Produktägare, Scrum master och utvecklingslaget). Utvecklingslaget bör också tillsammans ha alla olika kunskaper som krävs för att leverera produkten som efterfrågas. Kanban kräver inte några specifika roller alls, däremot så är det såklart logiskt att man har någon slags lagledare, chef m.m. Det är något som i Kanban borde växa fram i samband med vad projektet och organisationen behöver.

Som sagt så kör Scrum med bestämda tidsintervaller som kallas för "sprints" där man också bestämt hur mycket man tror kommer hinnas med i en sprint och har då det även som sprintmål. Kanban har inga bestämda intervall eller tidsboxar för olika moment utan metoden är naturligt iterativ och framsteg förväntas allt eftersom arbetet kontinuerligt blir klart.

Agila metoder som scrum och kanban skiljer sig en del från sekventiella metoder som vattenfallsmetoden. I vattenfallsmetoden så jobbas det strikt steg för steg i en bestämd ordning. Laget går inte vidare tills nästa steg innan det aktuella steget är helt slutfört och godkänt. Det första steget är att man analyserar och dokumenterar, planerar allt möjligt oerhört noga eftersom denna plan sedan skall följas punkt för punkt. Dokumenteringen tar upp väldigt mycket tid eftersom den är så viktig att göra rätt från början i vattenfallsmetoden, fördelen blir att man kan estimerar rätt bra hur lång tid arbetet som man har kvar att göra kommer att ta. I agila metoder gör man om alla steg flera gånger (iterativa) och är således öppen för förändringar, risken finns dock för att projektet kan dra ut på tiden eftersom att man inte har en fullständig plan som man har i vattenfallsmodellen.

Både chaos model och kanban är mer självorganiserade än scrum som tidigare nämnt kräver att man har minst tre roller bestämda. En distinkt skillnad från de andra metoderna är att chaos model bygger på att **alltid** lösa det viktigaste problemen först, istället för att följa planen som andra metoder gör.

Alla metoderna har sina för och nackdelar, däremot känns det som att i stort sett så är de agila metoderna scrum och kanban bäst idag eftersom det ofta sker förändringar och då är inte vattenfallsmetoden lika bra. Flexibelt känns som det ultimata även om det kanske inte blir en lika bra tidsoptimering och projektet kan dra ut på tiden om det blir många förändringar med mera.

#### 4.4.2 Utvärdering av testverktyg

Både unittest och catch är av typen unit test vilket gör dem ganska likvärdiga. Unittest är dock i python medan catch är till för C++. Personligen fann jag att pythons unittest var lite enklare att implementera eftersom endast en simpel include behövdes medan man var tvungen att ladda ned och inkludera en fil för att använda catch. Båda verktygen har bra dokumentation men för mig så var unittest dokumentation enklare att ta in.

Slutsatsen är att båda verktygen förmodligen är väldigt bra att kunna använda men jag har inte tillräckligt mycket kunskap ännu för att kunna dra riktig nytta utav verktygen. Det lilla jag testade verktygen fungerade dem precis som förväntat men tyckte att det var enklare att komma igång och förstå hur python's unittest fungerade.

#### 4.4.3 Utvärdering av stödjande verktyg

Discord är ett mycket bra gratisprogram för VOIP, man behöver inte ringa upp någon utan bara gå in i en kanal på sin server så syns det att man är tillgänglig, alternativt bara gå in i en kanal där du ser att en annan person är så är det bara att börja prata direkt. Det är även möjligt att bara chatta via textform. En del open-source projekt använder faktiskt Discord för kommunikation för att få en väldigt bra chatt till projektet. Exempelvis <https://home-assistant.io/> har en

Discordserver som man då gjort ett antal kanaler i där man endast behandlar specifika ämnen. Fungerar mycket bra för mig som själv använder Discord varje gång datorn är igång hemma.

Git stödjer fler operativsystem än vad RCS gör, git stödjer Linux, Windows, macOS samtidigt som RCS bara stödjer Unix-like system. Git är betydligt nyare programvara och senast stabila release kom 08-01-2017 medan RCS senaste stabila release var 01-22-2015.

RCS kan endast påverka en endaste fil åt gången medan Git kan förändra ett helt träd av filer direkt. I RCS så måste alla användare finnas på samma filsystem för att kunna arbeta med filerna. Med Git däremot så kan man om så önskas arbeta över nätverk, genom internet så att man kan sitta på andra sidan jorden och fortfarande sammarbeta med ett projekt.

Overall så känns Git som det allra smidigaste och bästa verktyget för versionshantering och sammarbete då man får en bra översikt och inte ens behöver befinna sig på samma nätverk för att kunna arbeta på projekt.

#### **4.4.4 Utvärdering av dokumentationsverktyg**

Till att börja med så verkar det som att NDoc inte alls är lika uppdaterat som Doxygen i och med att senaste stabila releasen till NDoc kom 01-25-2005 medan senaste stabila release till Doxygen var 12-29-2016. Dessutom så stödjer Doxygen dem flesta operativsystemen (Windows, OSX, Linux, BSD och Unix) medan NDoc endast har stöd för Windows.

Förutom hur uppdaterat och kompitabla verktygen är med olika operativsystem så skiljer sig sättet verktygen läser in informationen som den genererar sin dokumentation utifrån. Doxygen extraherar dokumentation direkt från kommentarer i källkoden till skillnad från NDoc som istället använder sig av en assembly fil som skapas vid kompilering av källkoden, samt så brukar NDoc även förgenererade dokumentationsfiler i XML format som produceras genom specialkommentarer.

NDoc genererar endast dokumentation i HTML och CHM medan Doxygen levererar sin dokumentation till HTML, Java, C#, IDL, Fortan m.fl.

Doxygen känns uppenbarligen som det bättre verktyget med tanke på att den stödjer fler operativsystem och kan generera sin dokumentation i fler format, samt är mer uppdaterat än vad NDoc är. Hoppas på att få mer erfarenhet av Doxygen efter att ha gått igenom det momentet i kursen senare.

## 5 Referenser

- [<sup>1</sup>] Scrum (08-28-2017). I Wikipedia. Hämtad 09-08-2017, från [https://en.wikipedia.org/wiki/Scrum\\_\(software\\_development\)](https://en.wikipedia.org/wiki/Scrum_(software_development))
- [<sup>2</sup>] Kanban (08-17-2017). I Wikipedia. Hämtad 09-08-2017, från [https://en.wikipedia.org/wiki/Kanban\\_\(development\)](https://en.wikipedia.org/wiki/Kanban_(development))
- [<sup>3</sup>] Waterfall model (07-21-2017). I Wikipedia. Hämtad 09-08-2017, från [https://en.wikipedia.org/wiki/Waterfall\\_model](https://en.wikipedia.org/wiki/Waterfall_model)
- [<sup>4</sup>] Chaos model (08-02-2017). I Wikipedia. Hämtad 09-08-2017, från [https://en.wikipedia.org/wiki/Chaos\\_model](https://en.wikipedia.org/wiki/Chaos_model)
- [<sup>5</sup>] unittest – Unit testing framework (09-07-2017). I Python standard library. Hämtad 09-08-2017, från [https://en.wikipedia.org/wiki/Chaos\\_model](https://en.wikipedia.org/wiki/Chaos_model)
- [<sup>6</sup>] Software testing (07-26-2017). I Wikipedia. Hämtad 09-08-2017, från [https://en.wikipedia.org/wiki/Software\\_testing#The\\_box\\_approach](https://en.wikipedia.org/wiki/Software_testing#The_box_approach)
- [<sup>7</sup>] Catch (07-26-2017). Catch Github. Hämtad 09-08-2017, från <https://github.com/philsquared/Catch>
- [<sup>8</sup>] Git (08-24-2017). I Wikipedia. Hämtad 09-08-2017, från <https://en.wikipedia.org/wiki/Git>
- [<sup>9</sup>] RCS (06-25-2017). I Wikipedia. Hämtad 09-08-2017, från <https://en.wikipedia.org/wiki/RCS>
- [<sup>10</sup>] Discord (software) (09-05-2017). I Wikipedia. Hämtad 09-08-2017, från <https://en.wikipedia.org/wiki/RCS>
- [<sup>11</sup>] Doxygen (08-25-2017). I Wikipedia. Hämtad 09-08-2017, från <https://en.wikipedia.org/wiki/Doxygen>
- [<sup>12</sup>] Comparison of documentation generators (09-04-2017). I Wikipedia. Hämtad 09-08-2017, från [https://en.wikipedia.org/wiki/Comparison\\_of\\_documentation\\_generators](https://en.wikipedia.org/wiki/Comparison_of_documentation_generators)
- [<sup>13</sup>] NDoc (09-05-2017). I Wikipedia. Hämtad 09-08-2017, från <https://en.wikipedia.org/wiki/NDoc>
- [<sup>14</sup>] Comparison of documentation generators (09-04-2017). I Wikipedia. Hämtad 09-08-2017, från [https://en.wikipedia.org/wiki/Comparison\\_of\\_documentation\\_generators](https://en.wikipedia.org/wiki/Comparison_of_documentation_generators)