Kvalitetssäkring

Redaktör: Ruben Das

Version 0.1

Status

Granskad	Ruben Das	-
Godkänd	Andreas Runfalk	-

Innehåll

1 Inledning

Kvalitetssäkring evaluerar och modifierar en organisations metoder för att se till att en slutprodukt håller rätt nivå. Vad som anses vara rätt nivå kan skilja sig mycket beroende på kundens behov.

Att lyckas uppnå en god kvalitetsstandard genom alla steg vid utvecklandet av en produkt eller tjänst kräver struktur, en struktur som sätts upp av en enskild individ eller en grupp av personer med relevant kunskap. I denna individuella rapport tänker jag ta upp vad som krävs för att uppnå en sådan struktur som sedan kan hjälpa till att generera en produkt eller tjänst av tillräckligt hög kvalité till kundens behov. Jämförelser kommer dras mot hur kvalitetssäkringen har gått tillväga i ett kandidatprojekt som jag har varit delaktig i.

1.1 Syfte

Syftet med denna individuella rapport är att fördjupa mig i ett område som har varit relevant till min roll som kvalitetssamordnare i kandidatprojektet ”Prediktionsreglering”.

Genom att fördjupa mig inom kvalitetssäkring kommer jag kunna lära mig mer om hur kvalitetssäkring fungerar och vad som skulle ha kunna gjorts bättre för projektet jag har varit delaktig i.

1.2 Frågeställning

1. Vad är rätt nivå gällande kvalité?
2. Vad krävs för att en slutprodukt eller tjänst ska vara av tillräckligt god kvalité?
3. Har slutprodukten i kandidatarbetet tillräckligt god kvalité?

1.3 Avgränsningar

Då huvudsyftet i detta kandidatprojekt har varit att implementera en optimeringsalgoritm kommer denna rapport främst fokusera på hur kvalitetssäkringen har gått tillväga för det, men även vilka förbättringar som skulle ha kunna gjorts. Detta för att rapporten inte ska bli onödigt bred och för att tiden är begränsad.

2 Bakgrund

Bakgrunden med denna bilaga är att framföra ett individuellt arbete i kursen kandidatprojekt i programvaruutveckling (TDDD77) som ges på Linköpings universitet som kandidatprojekt för studenter som läser till civilingenjör inom datateknik.

Denna bilaga skrivs inom området kvalitetssäkring, då min roll i kandidatprojektet har varit kvalitetssamordnare. Som kvalitetssamordnare har mitt arbete bestått av att se till att kod och dokument skrivs efter en viss standard som kommer förse att slutprodukten håller tillräcklig hög kvalité samt uppfyller de krav som har satts upp i form av en kravspecifikation.

2.1 Prediktionsreglering

Prediktionsreglering är namnet på kandidatprojektet som har beställts av Saab där Daniel Simon, en industridoktorand är kandidatgruppens kontaktperson. Syftet med projektet var att skapa en produkt som kan lösa konvexa kvadratiska optimeringsproblem utvecklad i programmeringsspråket C samt en parsingmjukvara och GUI utvecklad i Python. Lösaren som skapades fick namnet QuadOpt.

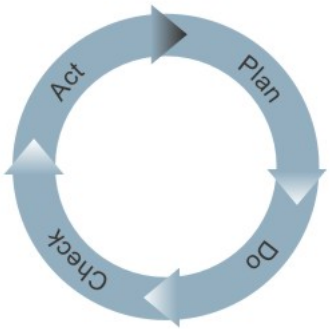
3 Teori

I boken ”Software Product Quality Control” Wagner (2013) nämns ett antal definitioner som förtydligar vad kvalitetssäkring innebär, dessa syns nedan.

- Quality assurance: a planned and systematic pattern of all actions necessary to provide adequate confidence that an item or product conforms to established technical requirements.
- Constructive quality assurance: All means to be used in constructing a product in a way so it meets its quality requirements.
- Analyctical quality assurance: All means of analysing the state of the quality of a product.

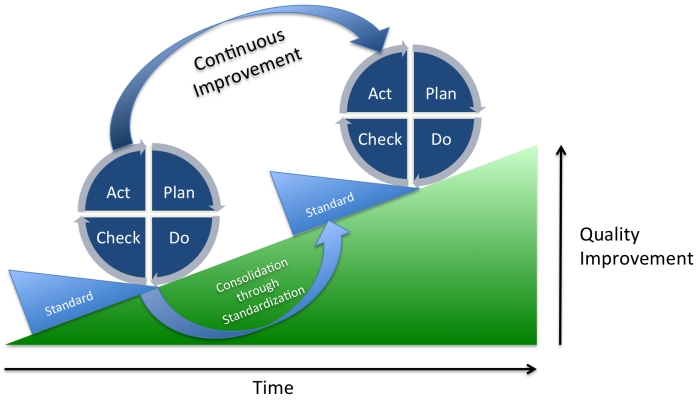
Kvalitetssäkring innebär följaktligen att man som leverantör av en produkt eller tjänst ska se till att de uppfyller de krav som har satts upp i en eventuell kravspecifikation. Att man under arbetsgången analyserar om man är på väg att uppfylla kraven eller inte, i sådana fall måste detta åtgärdas omedelbart.

Med detta sagt finns det flera steg i ett projekt att kvalitetssäkra. Ett effektivt sätt att göra detta på är genom att följa Shewhart cykeln, det vill säga planera, göra, studera och agera (PG-SA) (Johnson, 2002). Shewhart cykeln är en iterativ fyra stegs metod, se figur 1 och metoden är utvecklad av William Edwards Deming. Namnet ”Shewhart” kommer från en av Demings kollegor (Deming, 2000, p. 88).



Figur 1 – Shewhart cykeln (Mindtools)

1. Planera. I detta skede ska målet fastställas, det vill säga sätta upp de krav som behövs för att kunden skall bli nöjd. Genom att göra detta är det tydligt om vad som skall göras och en överenskommelse finns mellan kund och leverantör.
2. Göra. När man väl har fått för sig om vad som behövs göras för att kunden skall bli nöjd är det dags att implementera ett sätt att jobba och fullfölja processen.
3. Studera. Efter att ha följt processen under en viss period är det dags att utvärdera om processen man har följt kommer leda till att man uppfyller de målen man har fastställt i ”planera”-skedet.
4. Agera. Om man under ”studera”-skedet upptäcker att processen man följer inte kommer leda till att man uppfyller de krav som kund och leverantör var överens om måste detta åtgärdas omedelbart genom att planera om arbetsprocessen eller i viss mån diskutera kraven med kunden. Om processen som följs kommer uppfylla de krav som har satts upp kan man fortsätta med iterationerna av Shewhart cykeln precis som innan.



Figur 2 – PDCA process (Vietze, 2015)

Genom att följa Shewhart cykeln under ett projekt kan man iterativt förbättra sin arbetsprocess och genom detta öka kvaliteten av den produkt man utvecklar, se figur 2.

När det väl är dags för överlämning av en produkt eller tjänst är det bra att göra en kravinspektion innan. En kravinspektion enligt IEEE standarden för ”Software reviews” syns nedan.

> **”**A process or meeting during which a software product is examined by a project personnel, managers, users, customers, user representatives, or other interested parties for comment or approval**”** (SFS, 1997)

Det man vill få gjort med en kravinspektion är alltså att produkten eller tjänsten som har tagits fram ska utvärderas av berörda parter så att den uppfyller de krav som har fastställts i ett tidigare skede för ett godkännande. En kravinspektion kan se ut som sådan (Sandahl, 2015):

1. Tillträde.
2. Planering och översikt. Under detta steg sker en planering över hur inspektionen ska gå till och en översikt av produkten ges.
3. Individuell granskning. De berörda parterna granskar produkten eller tjänsten.
4. Möte. Efter granskningen görs en lista av alla de defekter som kan ha hittats.
5. Ändringar och uppföljning. I detta skede åtgärdar man de defekter som stöttes på och normalt brukar verifiera detta.
6. Utträde.

3.1 Sammanfattning

Sammanfattningsvis kan man konstatera att ett återkommande tema för att ta fram en högkvalitativ produkt eller tjänst är att planera, utföra och granska, för att sedan upprepa proceduren under projektets gång. Detta är dock bara en teoretisk tillämpning och verkligheten kan se annorlunda ut.

4 Metod

Kandidatprojektet ”Prediktionsreglering” är något av ett ovanligt projekt då huvudsyftet är att skapa en algoritm och inget övergripande system av komponenter. Detta medföljer att huvudsyftet

är att kvalitetssäkra en algoritm.

Innan kandidatgruppen gick in i första iteration av projektet skrevs en kvalitetsplan enligt IEEE-STD-730-2002 (Software Quality Assurance Plan) av kvalitetssamordnaren Ruben Das, dvs författaren av detta dokument. Detta dokument gav riktlinjer samt procedurer som bör följas av kandidatgruppen för att leverera en produkt av tillräckligt hög kvalitet till kunden. Daniel Simon som är kontaktpersonen för Saab specificerade tydligt att all kod gällande algoritmen ska vara väldokumenterad och skriven i C, samtidigt uppfylla de krav som finns med i kravspecfikationen. Med detta sagt visste kandidatgruppen vad som krävdes för att leverera en algoritm till rätt kvalitativ nivå. Kvalitetsplanen specificerar en kodstandard som innehåller de riktlinjer som skall följas när man skriver C-kod. Detta var första steget med att få in ett sorts kvalitetsmoment i projektet. Kodstandarden kan hittas under bilaga A och är skriven av Ruben Das samt Alexander Yngve som har varit utvecklingsansvarig under kandidatprojektet. Kodstandarden har baserats på vad som anses vara ”snygg” och läsbar kod. All C-kod som skrivs ska också kompileras med diverse olika flaggor, kompilatoranropet kan hittas nedan.

```
gcc -Wall -Werror -Wextra -pedantic -std=c99 -o namn namn.c
```

Flaggorna benämner att ta hänsyn till alla varningar, behandla alla varningar som fel, ta hänsyn till extra varningsfall och att rätta koden ”strängare” än vanligt. Det ska också noteras att koden kompilerar efter c99 standarden. Med hjälp av dessa flaggor ställdes höga krav på hur kod skulle skrivas.

När kandidatprojektet väl gick in i iterationerna och kod hade skrivits togs ett beslut om att centrala funktioner i koden skall ha minst ett testfall som de klarar. Testerna kördes varje gång man kompilerade koden, detta genom ett byggsystem som hade utvecklats i språket Make. Byggsystemet kompilerade koden med kompilatorsträngen som har nämnts tidigare och samtidigt körde den alla testfiler som fanns i biblioteket. Om ett testfall inte lyckats kommer kompilatorn ge ett felmeddelande i form av en ”assertion failed”, dvs resultatet blev inte som väntat. Även en byggserver användes för att se till att all kod var körbar och klarade alla tester varje gång någon hade skrivit ny kod och lagt upp det på kandidatgruppens versionshanteringssystem. En byggserver är precis som det låter, en server som bygger upp kod och kör den. Servern som användes i detta kandidatprojekt var ”Travis CI”, där ”Travis” bara är ett namn och ”CI” översatt till svenska står för kontinuerlig integration. Om någon individ i kandidatgruppen skrev kod som inte var körbar eller inte klara något testfall skrev ”Travis” ett e-mail till denna individ att koden inte är accepterbar och uppdaterade statusen på koden från ”build passing” till ”build failing’ se figur 3, dessa bilder kunde ses på kandidatgruppens versionshanteringssystems hemsida.

Figur 3 – Travis status

Såhär långt in i projektet kvalitetssäkras algoritmen med hjälp av att kod skrivs efter viss standard och att koden går igenom ett antal moment för att se till att den är körbar, varningsfri och gör det den ska. Nästa steg som togs för att se till att koden höll god kvalité var ett moment som kallas refaktorering. Refaktorering innebär att man förbättrar kodens läsbarhet samt reducerar kodens komplexitet. Detta gjordes av ett antal individer ett flertal gånger med jämna mellanrum. Under refaktorering utvärderades också koden om den följde kodstandarden eller inte, om den inte följdes fördes en dialog med personen som skrev koden.

Det sista steget av kvalitetssäkringen för algoritmen var en kravinspektion. Den kravinspektionen som genomfördes var en nedskalad version av den som nämns under teoridelen i kapitel tre. Kravinspektionen som genomfördes var att bocka av den kravspecifikation som skapades under förstudien av kandidatprojektet och ett möte med kunden bokades in för att se om kunden hade

några synpunkter eller problem med vad som hade skapats. Kandidatgruppen visste att ett krav skulle bli svårt att uppfylla och kunden hade inga problem med att ta bort kravet.

När all kod väl var färdigskriven och kravinspekterad behövdes en teknisk dokumentation framställas. Som tidigare nämnt var ett av kundens huvudkrav att koden var väldokumenterad. Kodstandarden kräver att alla funktioner samt filer kommenteras om varför de behövs. Med hjälp av att allting har blivit kommenterad i koden kunde en teknisk dokumentation genereras av ett program kallat "Doxygen". De programmet gör är att skapa en PDF (Portable Document Format) som ger en överblick på de filer och funktioner som finns samt vad de gör med hjälp av kommentarerna som redan finns i koden.

Sammanfattningsvis kan man säga att det följdes fem steg för att algoritmen som har skapats ska ha hög nivå gällande kvalité. Dessa steg är:

1. Utformning av en kvalitetsplan.
2. Skapande av ett byggsystem och tester.
3. Refaktorering av kod.
4. Kravinspektion med kund.
5. Generering av teknisk dokumentation.

5 Resultat

Inledningsvis i detta dokument ställdes frågorna:

1. Vad är rätt nivå gällande kvalité?
2. Vad krävs för att en slutprodukt eller tjänst ska vara av tillräckligt god kvalité?
3. Har slutprodukten i kandidatarbetet tillräckligt god kvalité?

Efter att ha fördjupat i teorin inom kvalitetssäkring och utvärderat arbetet som har genomförts så är svaren:

1. Rätt nivå gällande kvalité är det som har kommit överens om med kunden. Om kund och leverantör har godkänt en kravspecifikation mellan varandra, så räcker det för leverantören att uppfylla de krav som finns i en eventuell kravspecifikation för att uppnå rätt nivå gällande kvalité.
2. Det som krävs för att uppnå en slutprodukt eller tjänst av tillräckligt god kvalité är att man har en gedigen utvecklingsprocess. Genom att följa Shewhart cykeln kan man få struktur och agera om något går fel.
3. Ja, slutprodukten håller tillräckligt god kvalité. Detta på grund av att kandidatgruppen uppfyllde alla de kraven som hade specificerats. Dock så bör det noteras att ett krav omförhandlades bort.

6 Diskussion

Under denna del kommer en diskussion angående om resultaten ske samt metoderna som användes.

6.1 Resultat

Resultaten för alla tre punkter måste anses vara rimliga. Den första punkten angående vad som är rätt nivå är en väldigt diffus fråga och kan bara svaras genom att båda parter, dvs leverantör och kund har ett enhetligt svar. Eftersom det man kommer överens om i en eventuell kravspecifikation måste räcka att fullfölja för att det ska klassas som rätt nivå gällande kvalité. Sen om kunden hade velat haft några andra funktioner med produkten eller tjänsten så borde kunden ha specificerat detta. Leverantörer har också skyldighet att försöka uppfylla alla krav som har specificerats på ett snyggt sätt och inte bara slänga ihop något som fungerar, små detaljer kan utmärka ett arbete.

Den andra punkten finns det inte mycket att diskutera om, som allt annat i livet krävs det struktur för att man ska uppnå något. Under teoridelen i kapitel tre kan man konstatera att planera, utföra och granska antagligen kommer generera en produkt eller tjänst av tillräckligt god kvalité.

Angående den sista punkten kan man byta svaret från ett ja till ett nej, då ett krav förhandlades bort. I detta projekt hade kandidatgruppen endast ett mätbart krav, detta krav specificerade att optimeringsalgoritmen skulle vara likvärdig i hastighet i jämförelse med en kommersiell mjukvara kallad ”Gurobi”. ”Gurobi” specialiserar sig i att lösa olika sorters optimeringsproblem. Under projektets slutskede diskuterade kandidatgruppen med Daniel Simon om att det kommer bli svårt att vara jämbördig med ”Gurboi”, detta var inget problem för Daniel Simon och kravet kunde tas bort. Kontaktpersonen tryckte på att det viktigaste var att koden funkar och att den är väldokumenterad, vilket den är. Däremot kan jag tycka att om man tar bort ett krav så har man misslyckats och därför anser jag att man kan ändra svaret till nej, slutprodukten håller inte tillräckligt god kvalité, men då är jag väldigt hård mot mig själv och kandidatgruppen. Däremot säger teorin bakom det hela att produkten håller tillräckligt god nivå då den uppfyller alla krav, det ska inte spela någon roll om man har förhandlat bort något krav eller inte.

6.2 Metod

Efter slutförandet av projektet och analys av hur det hela gick finns det många saker man hade kunnat gjort annorlunda. Ett återkommande problem som stöttes på under projektets gång var att inte många respekterade kodstandarden som hade satts från början, inte förens den sista iterationen började folk ta den på allvar. Detta resulterade i att mycket tid lades ner på att refaktorera kod. Kodstandarden innehöll inte heller allt som bör ta hänsyns till, men som bör vara uppenbara för en programmerare. Ett exempel är att frigöra minne, det gjordes knappt och det resultera i att vissa var tvungna att hitta minnesläckorna och åtgärda dem. Kandidatgruppen hade ett krav på att skriva pålitlig kod och minnesläckor i koden är inte pålitligt.

Jag tror att slutprodukten skulle ha sett annorlunda ut om kodstandarden täckte fler områden i hur man ska skriva kod samt om man tryckte på hur viktigt det är att följa kodstandarden. Eftersom kandidatgruppen är en grupp av klasskamrater är det inga seriösa arbetsförhållanden mellan individerna. Detta leder till att kod bara slängs ihop och man tar inte kodstandarden seriös, jag som kvalitetssamordnare hade kunnat trycka mer på att kodstandarden skall följas för att en produkt av bättre kvalité skulle skapats.

I teoridelen visade forskningen att om man ska uppnå en högkvalitativ produkt kan man t.ex. använda sig av Shewhart cykeln, planera, göra, studera och agera. Kandidatgruppen planera hur man skulle skriva kod, man följde inte kodstandarden, kandidatgruppen var väl medveten om det. Vid detta stadiet borde man agera, visserligen sas det åt folk att börja använda kodstandarden, men det hade nog varit bättre att utvärdera om kodstandarden och föra dialog om varför den inte följs för att skriva en ny som möjligtvis alla känner sig bekväma med.

7 Slutsatser

Syftet med detta bidrag var att fördjupa sig inom kvalitetssäkring, för att sedan kunna reflektera om vad som kunde ha gjorts bättre för det projektet jag har varit delaktig i.

För det första har jag lyckats att fördjupa mig inom kvalitetssäkring, under kapitel ett hade jag ställt ett par frågor som skulle få svar genom att läsa på om vad kvalitetssäkring innebär. Under kapitel tre har jag tagit lärdomar om vad kvalitetssäkring innebär och hur man kan uppnå en produkt av god kvalité. Sammanfattningsvis kan man säga att kvalitetssäkring innebär att man som leverantör uppfyller de krav en kund har gett en och att man under arbetsgången analyser om man är på väg att uppfylla kraven eller inte. Metoder som att använda Shewhart cykeln finns tillhands för att göra detta. Avslutningsvis kan man göra en kravinspektion för att verkligen fastställa med alla berörda parter att slutprodukten gör det den ska.

Slutligen så anser jag inte att optimeringsalgoritmen som skapades är av god kvalité då vi inte lyckades uppfylla alla de krav som vi hade satt upp från början. Optimeringsalgoritmen skulle på förhand ha en exekveringstid likvärdig mot mjukvaran ”Gurobi” som är en kommersiell mjukvara enbart för optimeringsproblem. Detta var inget vi lyckades med och anledningen till att vi inte lyckades med detta är brist på tid och kunskap skulle jag påstå. En bättre struktur på kvalitetssäkringen under projektet skulle kunna ha gett oss mer tid, då som tidigare nämnt var en svårighet under projektet att kodstandarden inte följdes helt och en analys på om vi var på väg att möta kraven eller inte gjordes alldeles för få gånger. Jag vet att svaret om optimeringsalgoritmen är av god kvalité är tvetydigt genom detta dokument, men för att göra det enklare för läsaren. Optimeringsalgoritmen håller tillräckligt god kvalité enligt teorin, men optimeringsalgoritmen håller inte tillräckligt god kvalité enligt mig.

Litteraturförteckning

IEEE Standard for Software Reviews, 1997.

W. Edwards Deming. *Out of the Crisis*. The MIT Press, 2000. ISBN 0262541157.

Corinne N Johnson. The benefits of pdca. *Quality Progress*, 35(5):120, 2002.

Mindtools. URL `http://www.mindtools.com/media/Diagrams/PDCA.jpg`. [2015-05-02].

K Sandahl. Software reviews, 2015.

Johannes Vietze, 2015. URL `http://en.wikipedia.org/wiki/PDCA#/media/File:PDCA_Process.png`. [2015-05-01].

S Wagner. *Software Product Quality Control*, chapter 1. Springer, 2013.

Bilaga A Kodstandard

- Indentering sker med två blanksteg.
- Funktionsnamn och variabelnamn skrivs med små bokstäver med understreck som separerator mellan ord. Får heta vad författaren önskar så länge det är relevant.
- Pekare skrivs med asterisken direkt efter datatypen.
- Filer inkluderas där de behövs, antingen i c- eller h-filen.
- Den öppnande måsvingeparentesen skrivs på samma rad som funktionsnamnet och returtypen. Den stängande parentesen skrivs ensam på raden efter den avslutande satsen i funktionen. If-satser och liknande skrivs på samma sätt.
- Typedef sker separat från struct-deklarationer.
- Kommentarer skrivs som i exemplet nedan.
- Koden skall vara skriven på engelska.

```c
/*
  Author: Alexander Yngve
  Date: 2015-02-11
  Description: Short description of why this file is needed.
*/

#include <stdio.h>

/* Short explanation of why this type is needed. */
struct struct_t{
  int* pointer;
  int number;
};

typedef struct struct_t struct_t;

/* Short explanation of why this function is needed. */
int main(){
  printf("Hej\n");

  struct struct_t data_1;
  data_1.number = 4;
  printf("%i\n", data_1.number);

  struct_t data_2 = {NULL, 5};
  printf("%i\n", data_2.number);

  return 0;
}
```