IN3240 - Oblig 1

22. Mars 2019

Gruppemedlemmer:

Nadya Gileva (nadezdag)

Johan Hoi (jchoi)

Marius Haugen (marius5)

Emil H. Unhjem (emilhun)

Oppgave 1

Scriptene for denne oppgaven ligger i mappen “Oppgave 1 - scripts”

Oppgave 2

0.1: Option 1: “Hangman 2017”

1.1: Quality Gate: Passed. Dette forteller oss at produktet er “produksjonsklart”.

1.2:

1.2.1: Bug er et problem som representerer at noe er galt i koden. Selv om den ikke har skapt et problem enda, så er det stor sannsynlighet for at den vil det, og den bør derfor fikses.

1.2.2: Vulnerability er et sikkerhetsrelatert problem som representerer en bakdør for angripere.

1.2.3: Code smells er et vedlikeholdsrelatert problem i koden. Å la code smells være kan føre til at det blir vanskeligere for vedlikeholdere å gjøre endringer i koden, og i verste fall kan de forvirre vedlikeholdere så mye at de kan introdusere nye feil i endringene som blir gjort.

1.2.4: En blocker er en bug som har høy sannsynlighet for å påvirke oppførselen til applikasjonen som utvikles. Eksempler på problemer som dette kan lede til er memory leak og unclosed JDBC connection. Hvis koden inneholder en blocker bør koden fikses umiddelbart.

1.2.5: Graden, gjerne uttrykt i prosent, av hvordan et spesifikt coverage item har blitt utøvd av en test suite. Et coverage item er en enhet eller verdi brukt som en basis for test coverage, f.eks. equivalence partitions eller code statements.
En test suite er et sett av flere test cases for et komponent eller system under testing, hvor postkondisjonen av en test ofte blir brukt som prekondisjonen for den neste testen.

1.3: Mappen med høyest antall LOC i “Hangman 2017” er src/guihangamn med 2,029 LOC.
1.3.1: I denne mappen er det 21 bugs.
1.3.2: I denne mappen er det 23 vulnerabilities.
1.3.3: I denne mappen er det 368 code smells.

2.1:

Issue 1: “Remove this unused method parameter “evt”. ”, Code smell. Analyzer-en har rapportert dette som et problem ettersom at det er en metode her som tar inn et parameter “evt” som ikke brukes til noe i metoden. Selv om parameteren ikke gjør noen skade på koden er det best å fjerne den for å ikke skape forvirring og misforståelser.

Issue 2: “This block of commented-out lines of code should be removed”, Code smell. Analyzer-en har rapportert dette som et problem ettersom at den har funnet et eksempel på kode som har blitt kommentert ut. Selv om det ikke påvirker koden, kan det skape forvirring og dårlig lesbarhet.

Issue 3: “Use “java.util.Random.nextInt()” instead.”, Bug. Analyzer-en har rapportert dette som et problem ettersom at Math.random() ikke nødvendigvis gir en tilfeldig int, men kan for eksempel gi en flyt-verdi. I koden defineres randomWord som int og for å være sikker på at man får en int burde man bruke java.util.Random.nextInt() istedet.

Issue 4: “Change this condition so that it does not always evaluate to “true”.”, Bug. Analyzer-en har rapportert dette som et problem ettersom at if-setningen
if (controlPlay!=null) alltid får verdien “true”, noe som kan lede til “dead code” eller gale resultater. Slik kode er alltid full av bugs og bør aldri brukes i den endelige programvaren.

2.2:

Issue 1: Dette problemet kan enkelt fikses ved å fjerne parameteret fra metoden.

Issue 2: Slike problemer kan fikses ved å slette koden som har blitt kommentert ut. Kode som ikke blir brukt tar unødvendig plass og kan skape forvirring og dårlig lesbarhet i koden.

Issue 3: Dette problemet kan lett fikses ved å bruke java.util.Random.nextInt() i stedet. Da er man garantert en tilfeldig int.

Issue 4: Hvis en if-test alltid har resultatet “true” må man spørre seg om hvorfor man har en if-test der til å begynne med. For å fikse dette kan man enten fjerne if-testen eller gi if-testen et innhold med mening og som kan ta avgjørelser basert på forholdene i programvaren.

3.1:

3.1.1: False-positive (også kjent som false-fail result) er når testene ikke blir godkjent selv om det ikke finnes bugs i koden og at programmet fungerer helt fint funksjonelt.

3.1.2: False-negative (også kjent som false-pass result) i motsetning til false-positive skjer når testene blir godkjent selv om det faktisk er bugs/funksjonelle problemer i programmet.

3.2: Et eksempel på False-positive: False-positive kan oppstå når en testingeniør (under manuell testing) rapporterer en feil til riktig fungerende funksjon på grunn av feil eller uaktsomhet.

Et eksempel på False-negative: Ved automatisert testing kan verktøyet hoppe over en bane til funksjonaliteten som må testes. Det vil føre til at et sårbart system kan bli merket som bestått, eller i noen tilfeller kan det ikke oppdage funksjonaliteten i testen. False-negative er farligere enn false-positive siden det kan føre til problemer etter at programvaren er utgitt, eller i tilfelle webapps kan nettstedet bli hacket eller brukerdata kan bli kompromittert.

Oppgave 3

Parprogrammering

Session id:

- EL5kVB
- l9FaXq

Oppgave 4

Reviews er nødvendig under utviklingsprosessen for å evaluere produkter eller prosjektstatusen for å oppdage avvik fra planlagte resultater og for å foreslå

forbedringer. Fordelene ved reviews er mange, bl.a. kan man tidlig oppdage og fikse defekter, redusere testingstid og -kostnader, og det leder til færre defekter og forbedret kommunikasjon. Et viktig aspekt ved reviews er at man tidlig oppdager defekter i produktet/prosjektet, noe som kan bli dyrt i lengden dersom det forblir uoppdaget. Det er antatt at en defekt kan bli opptil 16 ganger så dyr å fikse, dersom defekten forblir uoppdaget til etter release.
Reviews deles ofte opp i to hovedkategorier: formal reviews og informal reviews. Formal reviews er reviews som karakteriseres av dokumenterte prosedyrer og krav, mens informal reviews er reviews som ikke følger en formell (dokumentert) prosedyre. Det er derfor ingen fundamentale steg i et informal review, i motsetning til et formal review. De fundamentale stegene for å utføre en formal review er planning, kick-off, preparation, review meeting, rework og follow-up.

- Planning – I dette steget velger man ut personell, tildeler roller, definerer «entry criteria» og «exit criteria» for mer formelle review typer (som inspection) og man velger ut hvilke deler av dokumentene som man skal se på.
- Kick-off – I dette steget distribuerer man dokumenter, forklarer objektivene til reviewet og reviewprosessen, forklarer dokumentene til deltagerne, og sjekker og diskuterer entry/exit criteria.
- (Individual) Preparation – I dette steget jobber alle deltagerne individuelt før reviewmøtet og noterer ned potensielle defekter, spørsmål og kommentarer. Hver deltager foreslår en alvorlighetsgrad for defekten og disse er (fra mest alvorlig til minst alvorlig) «critical», «major» og «minor».
- Review meeting – Under reviewmøtet diskuteres det og ting logges, med dokumenterte resultater. Møtets deltagere kan notere defekter, komme med forslag til behandling av defekter eller gjøre avgjørelser angående defekter. Avgjørelsene vil da være basert på prosjektets «exit criteria». Kort oppsummert gjør man undersøkelser, vurderinger og logger hendelser.
- Rework – Her fikser man defekter man har funnet, noe som oftest blir gjort av den som har rollen «author».

- Follow-up – Her sjekker man om defektene har blitt adressert. Man samler inn metrikker som antall defekter funnet, antall defekter funnet per side, tid brukt på å sjekke hver side, den totale review-innsatsen, osv.

Det finnes flere typer for reviews innenfor spekteret av informal/formal reviews. Hovedtypene kalles informal review, walkthrough, technical review og inspection. I et informal review er reviewet ikke basert på en dokumentert prosedyre og kan utføres av så få som to personer. Walkthrough reviews er en steg-for-steg presentasjon av forfatteren («author») av et dokument som gjøres for å samle informasjon og for å få en felles forståelse av dens innhold. Et technical review er en gruppediskusjons-aktivitet som fokuserer på å oppnå enighet på den tekniske tilnærmingen som skal tas. Inspections er en type review som fokuserer på visuell undersøkelse av dokumenter for å oppdage defekter som f.eks. brudd på utviklingsstandarder. Dette er den mest formelle review-teknikken og er derfor alltid basert på en dokumentert prosedyre.

I reviews er det vanlig at team-medlemmene tildeles roller med visse ansvarsområder. Disse rollene er manager, moderator, author, reviewers og scribe:

- Manageren bestemmer hvordan reviewsene skal utføres, tildeler tid i prosjektplaner og bestemmer om review-prosessens objektiver har blitt møtt.
- Moderatoren planlegger og leder reviewene, leder møtene, utfører entry checks og follow-ups etter møtene i rework-prosessen, og lagrer/tar vare på dataene som samles inn. I tillegg er moderatoren den som fastsetter review-typen, tilnærmingen som skal tas og definerer/velger teammedlemmer.
- Forfatteren («author») er den som skriver dokumentene som blir reviewet eller den som er ansvarlig for dokumentene som blir reviewet. Hans grunnleggende mål er å lære så mye som mulig for å forbedre dokumentets kvalitet samt tyde lite forståelige områder og forstå defekter som blir funnet.
- Reviewerne (også kalt inspectors) er individer med spesifikk teknisk bakgrunn/businessbakgrunner som identifiserer og beskriver funnene i

produktet under reviewet, og sjekker vanligvis ethvert materiale og dokumenter for defekter før møtene.

- «Scribe»-en dokumenterer hele review-møtet med et spesifikt fokus på å identifisere problemer, defekter, forslag til forbedringer, o.l.

Reviews har visse kriterier/suksessfaktorer som må være oppfylt for å lykkes. Disse kriteriene er kjent som objectives, roles, approach og training and learning. Disse faktorene sier at reviews må ta stilling til …

- Objektiver: Hvert review må ha klart predefinerte objektiver.
- Roles (roller): De riktige personene for review-objektivene må være involvert.
- Approach (tilnærming): Defekter som blir funnet er velkomne og må uttrykkes objektivt. Bruk passende review-teknikker for typen og nivået av programvare-produkter. Bruk sjekklister eller roller hvis passende for å forbedre effektiviteten til defektidentifiseringen. Ledelsen bør støtte en god review-prosess.
- Trening og læring: Trening er gitt i review-teknikker, spesielt i de mer formelle teknikkene, som inspection. Det er et fokus på læring og prosessforbedring.

Det finnes også andre generelle regler som er lure å følge for en vellykket review-prosess:

1. Review-prosessen er nødt til å ha en person som vil lede prosessen på et organisatorisk nivå.
2. Velg dokumentasjon for reviewene som er viktigst i prosjektet.
3. Velg review-teknikken som passer best for prosjektet
4. Planlegg reviews-aktiviteter
5. Opplæring til deltagerne er en nødvendighet, spesielt i en inspection-prosess
6. Siden et review er en prosess av evalueringer av et dokument som ble skrevet av noen, må psykologiske aspekter av en review håndteres.
7. Følg regler/reglene
8. Kontinuerlig forbedring av prosess og støtteverktøy

9. Rapporter resultater
10. Bruk testere

Når reviews blir brukt i tidlige stadier av utviklingsprosessen blir testingsprinsippet «early testing» brukt. «Early testing» går ut på å finne defekter tidlig ved å iverksette testaktiviteter så tidlig som mulig og har et fokus på definerte objektiver. Reviews har i likhet med dette prinsippet også definerte objektiver og reviews har også som hensikt å finne defekter tidlig slik at man kan komme med forslag til forbedringer og løsninger.
Static analysis er en undersøkelse av kravene, designet og koden som skiller seg fra den mer tradisjonelle dynamiske testingen ved at man ikke kjører programvaren som undersøkes. Ideelt sett er static analysis utført før et formal review. I likhet med reviews er målet til static analysis å finne defekter tidlig, selv om de ikke nødvendigvis skaper problemer. Man kan derfor si at static analysis og reviews er relatert til hverandre ved at de samme målene og kan effektivt brukes kooperativt under en utviklingsprosess.

Som vi vet, finnes det 2 typer testing: dynamisk og statisk. Siden dynamisk testing kun kan brukes til programvarens kode, er det statiske testingsmetoder som sjekker og analyserer om kravspesifikasjonen, dokumentasjonen (testplan, brukermanual) og designet er riktige og om de inneholder alvorlige eller mindre alvorlige defekter. Reviews er en typisk form for statisk testing, så ved hjelp av reviews kan man finne defekter i alle dokumenter og prosesser som ble nevnt tidligere.

Etter at reviews blir gjennomført, kan static analysis brukes for å jevne ut ekstra defekter før koden kjøres. Static analysis er en teknikk for eksaminering av kravspesifikasjon, design og kode ved hjelp av forskjellige verktøy. I likhet med reviews, har static analysis hovedfokus på kravspesifikasjon, design og kode, og ikke bare programvarekode som i dynamisk testing. Både static analysis og static testing eksaminerer dokumenter og prosesser uten eksekvering. Samt begge to har som mål å finne så mange defekter som mulig i tidlige faser av produktutviklingen.

Med tanke på at hensikten til reviews er å tidlig oppdage defekter og komme med forbedringer/løsninger til disse defektene, blir det naturlig å spørre seg om reviews er en effektiv kvalitetssikring. Vi vil argumentere for at reviews er en god kvalitetssikring ettersom at defekter og feil ved systemet blir oppdaget tidlig og kan derfor fikses på et tidlig stadium i utviklingsprosessen. På den måten kan man tidlig ta tak i feil og defekter ved systemet og forhindre at disse defektene skaper større problemer senere i utviklingsprosessen. Som nevnt tidligere kan en defekt bli opptil 16 ganger så dyr dersom den forblir uoppdaget til etter release.

Reviews øker produktets kvalitet, siden de kan brukes i tidlige faser av produktutviklingen, noe som gir oss tidlige tilbakemeldinger på kvalitetsaspekter ved et produkt. Kravspesifikasjonen blir validert tidlig, og ikke bare i løpet av akseptansetesting. Når defektene blir funnet tidlig, reduserer det også rework-kostnader og tid, noe som fører til at utviklingsproduktiviteten økes. I tillegg blir det brukt mindre tid på testing og vedlikehold.