Funktionstestning av programvara med grafiskt användargränssnitt utan att använda grafiken

Ett arbete av:

Nima Davoudi-Kia
&
Alijan Momeni

Maj 2001

Med tanke på att examenarbetet i stort sett handlar om programvarutestning, verkar det rimligt at börja med en generell definition av begreppet (programvaru)testning.

Testning är att analysera en programvara för att upptäcka skillnaden mellan dess befintliga och önskade tillstånd och utvärdera programvarans kännetecken.

Testningen, som utgör 40% till 80% av den totala kostnaden för programvaru-utvecklingen, kan klassificeras i följande steg: **Unit testning** innebär att man testar varje komponent i en kontrollerad miljö för att försäkra sig om att komponenten fungerar på rätt sätt. I **Integration testning** testar man däremot gränssnittet mellan komponenterna för att kolla om de fungerar tillsammans. Innan produkten levereras till kunden, gör man **System testning** för att se om den uppfyller de krav som ställdes i kravspecifikationen. För att ta reda på om produkten uppfyller även kundens krav, gör man ytterligare en test som kallas **Acceptance testning**.

Att upptäcka och avhjälpa en defekt kan ibland störa delar av programvaran som fungerade bra tidigare. För att ta reda på sådana defekter gör man **Regression testning**, som är en ofta förekommande test och kan utföras i alla ovannämnda steg.

Man vill helst automatisera testningen för att bland annat: köra testet snabbare och oftare, ha bättre användning av tillgängliga resurser och leverera produkten tidigare till marknaden. För automtiserad testning behöver man förutom en programvara som **testobjekt**, även ett bra **testfall** och ett lämpligt **testverktyg**.

Testfallet är en uppsättning av olika tester utförda i en följd som är relaterade till testobjektet. Varje tangentnedtryckning och musrörelse vid utförandet av testerna, som görs manuellt för första gången, kommer att spelas in, t.ex. av testverktyget, för att ska vid ett senare tillfälle kunna användas som indata för automatisk körning av testet.

Vissa programvaror har grafiska användargränssnitt, som för enkelhetens skull kallas även GUI. **GUI** är ett gränssnitt där man ger kommandon till en dator genom att använda sig av ett pekdon, tex en mus, som manipulerar och aktiverar grafiska bilden i monitorn.

Inom automatiserad programvarutestning är GUI av stort intresse eftersom de har en rad möjligheter för användar- interaktion och mängder av kontrolelement (buttons, pulldown menus, toolbars, osv.).

På företaget Telelogic har man försökt att automatisera testningen av GUI, där man använde sig av GUI. Men nackdelen med den här metoden är att det, bland annat, krävs mycket underhållsarbete av testfallet varje gång programvaran ändras.

Tanken bakom vårt examenarbete är att undersöka eventuella möjligtheter för reducering av det här underhållsarbetet genom automatiskt testning av GUI utan att använda GUI.

Iden var att hitta ett gränssnitt mellan GUI och programkoden så att man kunde fånga in alla önskade signaler för att senare kunna spela upp dem igen. Man ville testa menyer, dialoger och ritareor från samma gränssnitt.
För att kunna testa produktens grafiska användargränssnitt behövde man ha goda kunskaper om produktens bibliotekstrukture. Bibliotekstrukturen kunde delas i fyra under bibliotek;
1)Bases, som är en förvaringsplats för all C kod.
2)Application, som innehåller programkod i applikationsnivå.
3)Framework, som innehåller allt som är gemensamt för alla editorer.
4)Editor, som innhåller specifika koden för en önskad editor.

Efter att ha studerat bibliotekstrukturen noga, så insåg man att det önskade gränssnittet mellan GUI och programkoden saknades och GUI och programkoden överlappade varandra på något sätt. Därför bestämde man sig för att börja testa menyerna först.

När användaren gör ett menyval så anropas en operation i Application-bibliotek, som i sin tur anropar en operation i Framwork-biblioteket som tar reda på den aktuella editorn och anropar operationen **MenuChoice** i Editor-biblioteket. MenuChoice tar reda på aktuella fönstern och aktiverar ett menynummer som är just det utvalda menyalternativet.
Man skrev inspelnings koder i Editorbiblioteket som spelade in menynumret och dess aktuella fönster. Dessa data sparades i en fil som blev en indata till en uppspelnings funktion. Genom att spela upp filen så hade man gjort automatiskt testning av menyvalet. Eftersom alla menyer finns numrerade i en fil så blir det svårt att underhålla gamla testdata. Anledningen är att om man lägger till eller tar bort en meny så måste man spela in menyvalet igen för att få nya indata.

När det gäller dialoger så finns det tre olika typers av dialoger, därför blir det tre olika test metoder för dialoger. Brist på ett gemensamt gränssnitt gör att man blir tvungen att testa varje dialog klass för sig själv, vilket är tidskrävande och underhållningen blir kostsam. Dessa problem finns även för testning av ritarean, eftersom man har svårt att identifiera var de olika objekten skapas. T.ex skapandet av en linje med pil kan vara så att linjen skapas först i en nivå och pilen senare i en annan nivå. Man har även problem med storleken av ritarean som kan ändras varje gång programmet startars på nytt eller när man har flera filer uppe samtidigt.

Som summering kan man säga att testning metoden är möjlig men inte rekomenderbar, eftersom man måste göra massor med ändringar i programmet, vilket är mycket tidskrävande. Det blir även kostsamt och problematiskt att underhålla gamla testdata. Problemmet kan möjligtvist lösas om man försöker skapa den önskade gränssnittet så att man kan spela in alla signaler från ett och samma gränssnitt.