Software Testing & Verification

Project stage 2

Jarno Le Conté 3725154
Bas Meesters 3700569

In dit rapport wordt beschreven hoe wij System Testing hebben uitgevoerd op ons eerder geschreven adresboek-applicatie. Nadat de requirements voor deze fase duidelijk waren hebben wij kleine aanpassingen aan het programma gedaan zodat deze makkelijker te testen is. Eerst wordt beschreven wat voor aanpassingen er zijn doorgevoerd en wat voor effecten dit heeft. Vervolgens wordt uitgelegd hoe Record & Replay is geïmplementeerd en hoe hiermee getest wordt. De coverage criteria waaraan wij voldoen wordt daarna behandeld en er wordt geëindigd met een conclusie over onze applicatie en de correctheid hiervan

System Testing:

Allereerst om sytem testing zo goed mogelijk toe te passen hebben is de structuur van het programma lichtelijk aangepast. Er wordt nu gebruik gemaakt van verschillende views, namelijk die van het laten zien van alle adresboeken, het toevoegen van een adresboek en het laten zien van een specifiek adresboek. Wanneer er in één van deze toestanden via een interactie het programma naar een andere toestand wordt veranderd, wordt eerst gezorgd dat de huidige toestand naar een basis toestand wordt gebracht, namelijk een toestand waarin alles weer onzichtbaar en leeg wordt gemaakt. Hierna wordt pas de nieuwe toestand geladen. Op deze manier wordt gezorgd dat wanneer een toestand geladen wordt deze altijd begint vanuit een zelfde startpositie waardoor de begintoestand elke keer hetzelfde is en het aantal toestanden deterministisch is. Helaas kan door de complexiteit van C# vrijwel niet bewezen worden dat staat van het programma daadwerkelijk elke keer hetzelfde is omdat de hoeveelheid data die wordt bewerkt en gebruikt erg groot is bij het gebruik van de Windows Forms library.

Record & Replay:

Om test cases automatisch te laten verlopen hebben wij gebruik gemaakt van de coded UI tests die te gebruiken zijn in Visual Studio 2010 Ultimate. Met deze tool is het mogelijk om interacties op te nemen zodat deze opnieuw afgespeeld kunnen worden. Via de coverage code tool die ook aanwezig is in Visual Studio hebben wij na een aantal iteraties test cases kunnen opstellen die voldoen aan een 100% code coverage. Voor de oracles om te testen of het programma voldoet aan de te verwachten staat hebben wij gebruikt gemaakt van dezelfde oracles als die gebruikt zijn in stage 1 voor het testen van waardes van data. Daarnaast zijn er extra oracles toegevoegd voor het testen van de GUI elementen. Er wordt gekeken of bij een bepaalde staat van het programma de knoppen, labels en andere controls ook daadwerkelijk zichtbaar, onzichtbaar, leeg zijn of zich in een andere staat bevinden, wat vervolgens wordt vergeleken met wat verwacht wordt. Bij het doorlopen van de interacties wordt na iedere interactie gekeken of het programma zich daadwerkelijk in dezelfde staat begeeft als de staat die verwacht wordt.

Loggen:

Voor het loggen van interacties hebben is het programma opgedeeld in verschillende nodes. De verschillende nodes zijn in dit geval opgesteld aan de hand van de punten in het programma waarin een keuze gemaakt wordt. Hier vallen onder if-then-else statements, loops en functie-aanroepen. Log-statements zijn op de volgende manier geplaatst:

if-then-else	Het 'then'-blok is een node Het 'else'-blok is een node Direct na de if-then-else structuur is een node	if .. then { **log("a")**; } else { **log("b")**; } **log("c")**;
loops	De loop-body is een node Direct na de loop is een node	while .. do { **log("a")**; } **log("b")**;
functieaanroep	Direct na een functieaanroep is een node	call(); **log("a")**;

Note: *Indien een functie-aanroep wordt gelogd dan loggen wij ook de nodes in de body van die functie. Als je dat niet zou doen dan is het mogelijk dat de Replay test niet slaagt.*

Met behulp van de logs kunnen wij onder andere nagaan of wij ons opgestelde coverage-criteria behalen.

Coverage criteria:

Om alle functionaliteiten te testen die wij in stage 1 ontwikkeld hebben hanteren wij minimaal de node coverage criteria. Met de ingebouwde coverage tool van Visual Studio is het mogelijk om na te gaan dat wij test-cases hebben ontwikkeld die aantonen dat er sprake is van 100% code coverage.

Node coverage is uiteraard niet een erg sterke criteria en om de coverage criteria te versterken wordt er ook gebruik gemaakt van prime path coverage voor een complex deel van de code, namelijk de "verwijderen van contacten" methode. Er is voor deze code gekozen vanwege de volgende redenen:

- Het verwijderen van contacten gebeurd op 'System Testing' niveau omdat het van verschillende klassen gebruik maakt waaronder de klassen Adresboek, Contact en View. Het verwijderen hangt bijvoorbeeld af van het geselecteerde adresboek. Ook wordt de view volledig geüpdated nadat het contact verwijderd is.
- De methode bevat een complexe structuur van geneste loops in combinatie met if-then-else statements. Hierdoor zijn er meerdere prime-paden.
- Ondanks de complexe structuur is het een kleine methode waardoor het perfect geschikt is om te demonstreren hoe we de theorie van prime path coverage toepassen.
- We hebben te maken met enkele infeasible test-requirements waardoor we genoodzaakt zijn enkele tests te runnen met sidetrips/detours.

Voor het berekenen van prime-paden hebben wij een eigen programma geschreven volgens het algoritme gegeven in het boek "Introduction to Software Testing". De verkregen prime-paden stellen wij als onze test-requirements. Vervolgens hebben wij test-cases opgesteld die alle prime-paden doorlopen, enkele daarvan met behulp van sidetrips/detours.

Conclusie:

Over het algemeen zijn we redelijk zeker dat ons programma correct is en geen fouten bevat. Zoals aangegeven hebben we 100% code coverage gehaald met Record & Replay functies. In elk test case wordt ook gekeken of de staat van het programma hetzelfde is als de staat die verwacht wordt. In alle test-cases was dit het geval.

Daarnaast hebben we voor een deel van het programma ook de prime paden berekend en met behulp van loggen kunnen aantonen dat er prime path coverage is. Dus ook voor dit complexere onderdeel zijn we vrij zeker dat het programma correct is.

Zoals eerder vermeldt gaan we er van uit dat wanneer gewisseld wordt van views de toestand elke keer eerst in een basis toestand wordt gezet zodat het aantal toestanden deterministisch is. Dit kan alleen niet met zekerheid aangetoond worden en mochten er toch fouten in het programma zitten dan zal dat hoogstwaarschijnlijk met de views te maken te hebben. Ook hebben we slechts prime path coverage voor één methode aangetoond en voor het overgrote deel van het programma alleen code coverage. Voor volledige system testing moet een sterkere coverage criteria gehanteerd worden.

Dus kortom, we zijn er zeker van dat we een groot deel van het programma getest hebben en dat we aan bepaalde coverage criteria hebben voldaan voor het overgrote deel van het programma. Maar er blijft toch een bepaalde onzekerheid doordat de prime path coverage maar op een deel van het programma is toegepast, de code coverage op de rest van het programma niet erg sterk is en er aannames gedaan worden over de staat van het programma die niet bewezen zijn.