



UNIVERSITA' DEGLI STUDI DI
NAPOLI FEDERICO II

Scuola Politecnica e delle Scienze di Base
Corso di Laurea in Ingegneria Informatica

Elaborato Finale in **Ingegneria del software**

Testing delle GUI

Anno Accademico 2014/2015

Candidato:

Luca Ioviero

matr. N46/001278

[Dedica]

Indice

| | |
|--|-----|
| Indice..... | III |
| Introduzione..... | 4 |
| Capitolo 1: Testing delle GUI..... | 5 |
| 1.1 Black Box o White Box..... | 5 |
| 1.2 Generazione dei Casi di Test..... | 6 |
| 1.2.1 Modello macchina a stati finiti..... | 6 |
| 1.2.2 Modello EIG..... | 7 |
| 1.3 P.A.T.H.S..... | 8 |
| 1.4 Implementazione casi di test..... | 8 |
| 1.4.1 Capture and Replay..... | 9 |
| 1.3.2 Cattura degli Eventi..... | 9 |
| Capitolo 2: Tools per il Testing..... | 10 |
| 2.1 Jubula..... | 11 |
| 2.2 QF-Test..... | 17 |
| 2.3 Un confronto..... | 21 |
| Conclusioni..... | 24 |
| Bibliografia..... | 25 |

Introduzione

Il testing delle GUI (Graphical User Interface) è un processo tramite il quale si verifica il corretto funzionamento dell'interfaccia grafica di un programma. Come vedremo in seguito vi sono diversi metodi adottati sia per la generazione dei casi di test, sia per la loro esecuzione, con i suoi pro e contro. Verranno inoltre analizzati i tools usati a supporto del processo di testing, lo scopo dei quali è sia quello di automatizzare le operazioni di testing, che come risaputo sono onerose in termini di ore/uomo e cruciali dal punto di vista del successo di un qualunque progetto. In particolare si cerca di automatizzare sia la generazione di test case tramite l'utilizzo di modelli dell'interfaccia grafica, sia automatizzare l'esecuzione stessa dei test.

Si mostreranno i problemi legati al testing delle GUI, e di come anche leggere modifiche all'applicazione portino a dover effettuare numerosi test di regressione. Basti pensare che la modifica di un singolo widget può portare in alcuni casi al fallimento di innumerevoli test case.

Infine si mostrerà il funzionamento di due software per il Testing di GUI : Jubula e QF-Test. Si tenterà di mostrare le potenzialità dell'uno e dell'altro tramite la creazione di alcuni casi di test su un programma semplice e loro successiva esecuzione. Infine si procederà a un confronto in cui si mostreranno le differenze tra i due evidenziandone pregi e difetti.

Capitolo 1: Testing delle GUI

La GUI è la parte di un applicativo che negli ultimi anni ha ricevuto sempre più attenzione da parte degli sviluppatori dato il suo ruolo cruciale di interazione con l'utente. Negli ultimi anni le GUI stanno diventando sempre più complesse e ricche il che introduce problemi di scalabilità. Abbiamo ovvero la necessità di trovare un approccio per la generazione ed esecuzione dei casi di test che riesca a gestire in maniera efficace interfacce molto complesse.

Inoltre l'eterogeneità degli utenti e la complessità del software porta gli sviluppatori a dover scrivere diversi casi di test che vadano a considerare tutti i possibili scenari. Un utente esperto ad esempio saprà già quale sequenza di passi seguire per eseguire una data funzionalità. Questo caso difficilmente genererà un errore in quanto segue un percorso ben analizzato e testato dai programmatori ; nel caso di un utente neofita invece un programma potrebbe essere sottoposto a una serie di input o sequenze di passi tali da attraversare aree di codice non coperte dai casi di test e quindi soggette a possibili malfunzionamenti.

1.1 Black Box o White Box

Una GUI è un sistema ad eventi ovvero evolve quando riceve input esterni che possono essere sia l'interazione con l'utente sia un evento interno. Alla ricezione di questi input la macchina evolve di stato in stato. Questo ci fa capire che è difficile scrivere codice che riesca a emulare il comportamento del software originale[1] dato che si tratta di software che è +in continua evoluzione e che genera componenti a tempo di esecuzione . Per questo l'approccio per il testing comunemente utilizzato è quello Black Box. Noto il comportamento previsto del software se ne testano le funzionalità andando a porre degli ingressi e verificando che gli output restituiti siano quelli previsti. Nel caso del testing delle GUI gli ingressi, oltre a essere dati, sono anche eventi. Ogni interazione dell'utente con l'interfaccia genera un'evento che viene catturato e trasformato in una chiamata di

funzione. Un evento può essere anche interno come il risultato dell'elaborazione del processo o un segnale ricevuto dall'esterno.

1.2 Generazione dei Casi di Test

Come già preannunciato la scrittura dei casi di test deve tenere conto sia dell'eterogeneità dell'utenza dell'applicazione, e che quindi è necessario testare scenari alternativi in cui si tiene conto dell'utilizzo non corretto dell'applicativo, sia del fatto che anche la applicazione più semplice possa avere un numero enorme di operazioni da eseguire e che queste siano legate fra loro tramite un principio di sequenzialità. Infatti seguendo un diverso ordine di esecuzione delle operazioni, un'applicazione potrebbe funzionare o meno. Se poi consideriamo che nel ciclo di vita di un software l'interfaccia grafica potrebbe sia cambiare leggermente che venir totalmente riscritta, dobbiamo anche tener conto che bisognerà riscrivere almeno in parte i casi di test, questo perché un caso di test è fortemente legato alle componenti istanziate della gui. Se modifico anche solo il nome di una componente della GUI o la sua posizione potrei ritrovarmi con uno o più Test Case non funzionanti. Fra i metodi di generazione dei test case si omette la generazione manuale dei test in quanto porta a diversi svantaggio fra i quali :

- Non copertura di alcuni scenari critici
- Tempo richiesto eccessivo

per questo motivo si è soliti delegare questo compito a un programma che esegue il compito in automatico.

1.2.1 Modello macchina a stati finiti

Un primo approccio per la creazione dei casi di test è quello di generare un'automa a stati finiti della GUI. Come già detto questa si sposta di stato in stato in relazione agli eventi che si verificano; se riuscissimo a modellare l'applicazione in tutti i suoi stati sarebbe facile generare dei casi di test in quanto esistono già dei criteri di copertura (stati , eventi, path) che ci permettono di creare dei test case efficaci. Questo modello può essere generato durante la scrittura dell'applicazione ma potrà non aderire con la versione finale del software oppure lo si potrà generare a GUI ultimata ma specialmente nel caso di

applicazioni dinamicamente configurabili si avrà bisogno di tools per effettuare il reverse engineering e generare quindi in maniera semi automatica il modello. [1]

Il problema di questo approccio è che nel caso delle GUI il numero di stati cresce in maniera esponenziale all'aumentare dei widget e delle funzionalità quindi questo metodo non è scalabile

1.2.2 Modello EIG

Un modello simile al precedente è quello basato sugli eventi. EIG (Event Interaction Graph) è un grafo i cui nodi rappresentano gli eventi scatenati durante l'interazione col programma mentre le transizioni non indicano altro che l'ordine con cui si verificano questi eventi. Una sequenza di eventi rappresenta un Test Case; Questo modello è potenzialmente illimitato per cui si cerca di generare dei test case che eseguano un numero di passi limitato e non troppo grande.

Il grande vantaggio di usare due modelli appena illustrati è che possibile generare in maniera automatica dei test case che coprano ogni volta un percorso diverso. Questo fa risparmiare tempo ai Test designer ed inoltre si evita di tralasciare qualche cammino non percorso che potrebbe sfuggire a un operatore manuale. Un problema che però si introduce è quello dei test non fattibili. Essendo questi modelli delle approssimazioni del modello reale spesso capita che un evento che sulla carta dovrebbe esser disponibile per l'esecuzione in realtà non lo è; questo causa il blocco del test oppure il crash del programma. Per risolvere questo problema sono stati adottati diversi approcci. Uno di questi è quello di sostituire il test case incriminato con uno simile ottenuto da una piccola modifica di quello partenza. L'algoritmo che si occupa di generare questi test case fattibili e che aumentano la copertura è detto algoritmo genetico.

1.3 P.A.T.H.S

Un altro metodo proposto per la generazione di test case è quello chiamato “Planning Assisted Tester for graphHical interface System” che ci permette di generare in maniera semi automatica dei test case sempre diversi e che coprono per buona misura tutti i percorsi possibili. Un enorme vantaggio che invece ha rispetto ai metodi discussi in precedenza è la scalabilità, ovvero possibilità di riuscire a gestire la creazione dei casi di test al crescere della dimensione dell'applicazione.

Si vanno a generare degli operatori gerarchici partendo dall'analisi della struttura della GUI. Se ne analizzano le componenti e le si classifica in base alle proprietà strutturali in 4 classi. In seguito il Test Designer dovrà specificare per ogni operatore le precondizioni e le postcondizioni per ogni operatore generato. Una volta avuta la lista completa degli operatori si dovrà andare a definire un task, ovvero un compito che il nostro programma deve svolgere. Si definiscono gli stati iniziali e finali e si pone il tutto in ingresso all'algoritmo PATHS il quale andrà a calcolare tutti i percorsi possibili.

Un altro vantaggio di utilizzare questo approccio è legato agli operatori gerarchici: una modifica dell'interfaccia andrà a impattare solo su alcuni operatori e i relativi figli. Questo ci permette di modificare in maniera semplice e veloce l'albero per ottenere di nuovo un test case funzionante. Inoltre la divisione in classi dei componenti permette di ridurre la complessità della GUI e quindi rendere questo metodo scalabile.

1.4 Implementazione casi di test

Abbiamo visto sinora come vengono pensati e progettati i casi di test. Come abbiamo detto la generazione automatica dei casi di test è conveniente per evitare errori umani ma soprattutto perché con strumenti automatici si risparmia molto tempo.

Il ruolo degli strumenti automatici nell'implementazione ed esecuzione dei test assume un carattere fondamentale in quanto una test suite ha al suo interno innumerevoli test case che talvolta differiscono solamente per qualche valore in ingresso o per qualche evento. L'esecuzione manuale quindi è scartata apriori. Sono state create due tipologie di programmi che testano in maniera indipendente il programma una volta che gli sono forniti i test case.

1.4.1 Capture and Replay

Tra i metodi più immediati e semplici per l'implementazione di un caso di test c'è sicuramente quello di capture and replay o detto anche capture and playback. Il suo funzionamento è molto semplice: l'utente simula il comportamento da testare mentre nel frattempo il programma di testing registra ogni interazione col programma testato. Una volta terminate le operazioni il programma di testing è in grado di riprodurre il comportamento registrato andando anche a controllare delle asserzioni per verificare la buona riuscita del test. Questo porta alla creazione di diversi test case in maniera semplice e veloce.

Il software di testing cattura durante la registrazione degli screenshots dell'applicazione a ogni interazione; Quando poi si esegue il test il software va a fare un matching fra lo screenshot registrato e quello che osserva. Questo è un po' il limite di questa tipologia di software in quanto basta una piccola modifica della GUI è tale da portare al fallimento del test case in quanto il software di testing considererà diverse le due schermate. C'è la possibilità di impostare un valore di soglia grazie al quale si possono ignorare delle minime modifiche. Inoltre nel tempo sono stati apportati altri accorgimenti per evitare che modifiche alla disposizione dei widgets all'interno dell'interfaccia utente portassero al fallimento di tutti i test case. A ogni widget è assegnato un nome il quale viene collegato all'azione che viene effettuata dall'utente; in questa maniera il sistema automatico conoscerà continuamente la sua posizione.

1.3.2 Cattura degli Eventi

I problemi legati al capture and replay sono per la maggior parte legati a come questo lavora: esso ha bisogno che posizione e forma di pulsanti rimangano gli stessi. Per ovviare a questo problema si è pensato di andare a registrare le interazioni fra il programma e il server grafico. Una volta registrate in un log tutte le interazioni si provvede a una forte scrematura in quanto la maggior parte delle voci non saranno di nostro interesse. Queste voci rappresentano le interazioni fra l'utente e l'applicazione. Un metodo alternativo è quello di inserire dei driver all'interno della nostra GUI il quale non dovrà far altro che ricevere comandi da un programma esterno che piloterà il testing[4].

Capitolo 2: Tools per il Testing

I tools automatici che permettono la creazione e l'esecuzione di Test Case diversi e diversa natura. Ognuno appartiene a una categoria diversa ognuna con i suoi punti di forza e di debolezza. Fra gli strumenti più diffusi abbiamo quelli di “capture and replay”, i keyword driven, quelli a cattura degli eventi. La tecnica del capture and replay è sicuramente la più immediata in quanto fornisce a chiunque conosca le funzionalità attese del programma di scrivere dei casi di test in maniera semplice, senza dover conoscere aspetti dell'implementazione e senza dovere scrivere una riga di codice. Questo metodo però ha una serie di svantaggi notevoli: innanzitutto dovendo “scrivere” il caso di test a partire dalla GUI già realizzata implicherà che il testing inizierà molto in ritardo rispetto al ciclo di sviluppo; inoltre la differenza anche minima fra lo screenshot fatto al momento della registrazione e di quello al momento della verifica porterà a far fallire il test. Il problema può essere aggirato modificando un dato parametro di soglia ma comunque non risolve il problema. Una piccola modifica dell'applicazione potrebbe comunque portare a dover riprogettare il test case registrato andando ad aggiungere a mano le azioni mancanti. Per finire il codice che viene generato in automatico non rispetterà gli attributi di stabilità, manutenibilità e portabilità.

Per risolvere definitivamente il problema della non somiglianza fra gli screenshot si è pensato di agire più a basso livello. Una prima idea è quella di usare una classe driver all'interno dell'applicazione stessa in maniera tale da scrivere dei casi di test tramite codice e quindi pilotare l'applicazione dall'esterno. Per quanto questa soluzione risulti molto efficiente comunque richiede un apporto sostanziale da parte dello sviluppatore per cui risulta molto costosa come soluzione. Inoltre dato che il testing della GUI viene spesso usato come testing per effettuare le validazioni di un software è auspicabile che il test case sia leggibile sia dal committente che dai progettisti. In questo senso è più conveniente usare un linguaggio di più alto livello che sia facilmente interpretabile e analizzabile. La soluzione a questo problema è data dai software di testing che creano test case a partire

dalla combinazione di microoperazioni, ognuna di queste è descritta da una parola chiave. È possibile in questa maniera creare dei test case prima di aver completato la scrittura della GUI. Inoltre chiunque conosca i requisiti del software e la struttura della GUI è in grado di scrivere dei casi di test e chiunque è in grado di capirli leggendo la sequenza di microoperazioni. Questo però richiede un passaggio aggiuntivo rispetto ai precedenti metodi: i test case generati sono indipendenti da qualunque GUI già implementata per cui ci sarà bisogno di effettuare un'operazione detta di “mapping” tramite la quale si collega ogni componente dell'applicazione “virtuale”, ovvero di cui noi conosciamo specifiche e struttura, con una componente della applicazione reale. Questa operazione può essere dispendiosa e non priva problemi. Mentre nei software di capture and replay il riconoscimento dei componenti è fatto a tempo d'esecuzione in maniera automatica, qui c'è bisogno di un apporto umano che colleghi tutti gli elementi. Questa operazione nel caso di software di grandi dimensioni è dispendiosa. Inoltre gli elementi generati a run time risultano ogni volta diversi e quindi quando si manderà in esecuzione il test questo fallirà dato che non troverà la corrispondenza tra i componenti.

2.1 Jubula

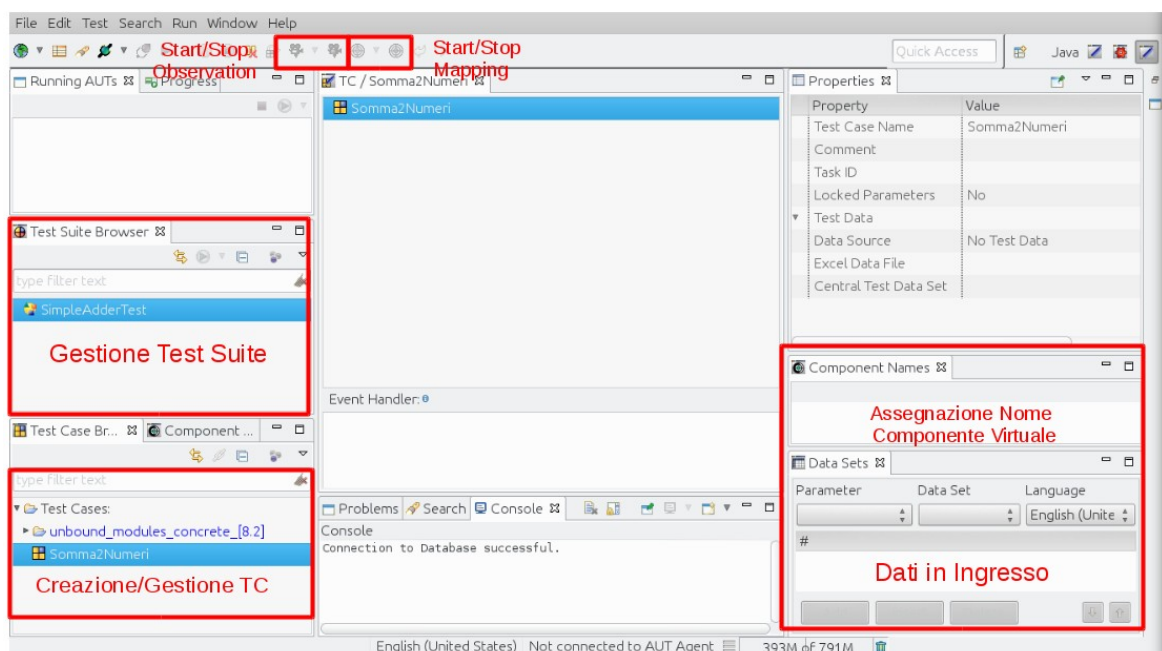
Jubula è un software usato per effettuare testing funzionale delle GUI e fa parte sia della categoria dei keyword based che dei capture and replay. L'ambiente supporta diversi Toolkit fra cui Swing, JavaFx, SWT, RPC e può effettuare testing anche per applicazioni web. Il programma è liberamente scaricabile e utilizzabile. Esiste una versione stand alone del programma e una integrata all'interno di Eclipse ma entrambe forniscono le stesse funzionalità.

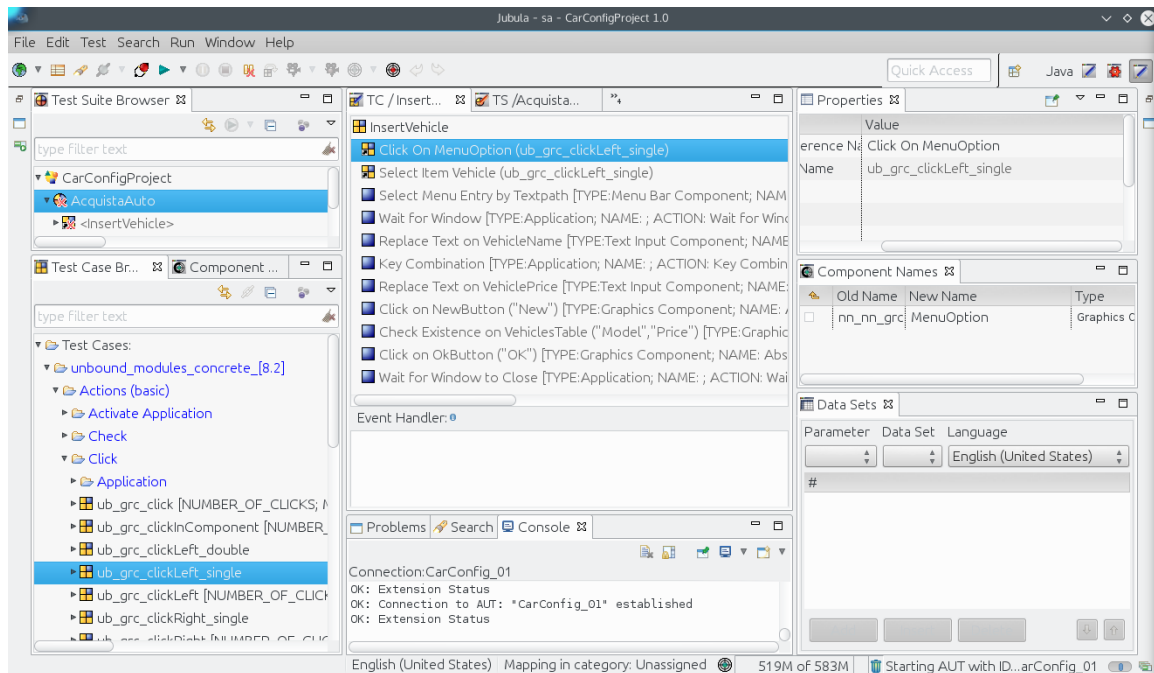
Jubula è uno strumento completo formato da un ITE (Integrated testing enviroment) e un AUT Agent ovvero un componente che permette la comunicazione fra l'AUT (Application under test) e l'ITE. Jubula inoltre mette a disposizione un database integrato in cui memorizzare tutte le test suite. È possibile anche configurare il programma per utilizzare un database esterno in maniera tale da permettere a più persone di collaborare sulla stessa test suite da remoto. Come detto precedentemente Jubula mette a disposizione dell'utente

un software per fare il mapping fra le componenti logiche e quelle della GUI già implementata. Oltre alla modalità di creazione sopra descritta che utilizza microoperazioni per comporre dei test case, Jubula mette a disposizione dell'utente una modalità "Osservatore" che nient'altro è che una funzione di capture and replay. Questa gode dei pregi e difetti sopra elencati ma in più ora è possibile utilizzare le operazioni atomiche per estendere e completare il comportamento di un test case registrato.

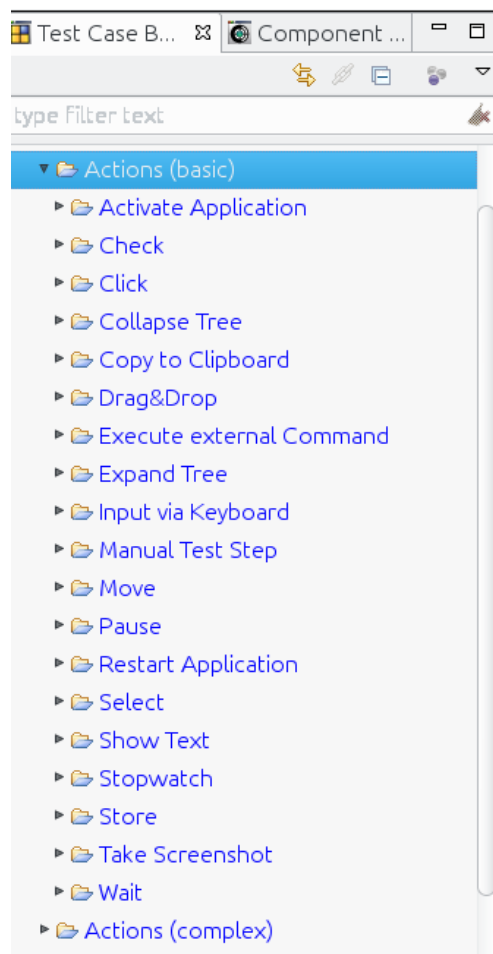
Jubula permette di creare insiemi di test case delle operazioni più comuni che insieme formano dei blocchi che possono essere utili per testare nuove funzionalità. Ogni volta che viene aggiunta una nuova funzionalità vengono riutilizzati questi blocchi e creata una nuova sequenza di operazioni che va a testare la nuova funzionalità.

Presenterò ora un esempio di testing funzionale su un'applicazione Swing distribuita insieme al programma. Per effettuare questi test è stata utilizzato Kubuntu 15.10 con la versione stand alone di Jubula. Ho provato diverse configurazioni prima di riuscire a far comunicare l'ITE con l'AUT agent, sia su Windows 7 che sulla stessa macchina ma con la versione di Jubula integrata in Eclipse ho riscontrato diversi problemi. Un altro malfunzionamento riscontrato con le suddette configurazioni è che l'AUT non venisse avviata correttamente non permettendo quindi l'esecuzione del test. La versione di Java utilizzata è Openjdk 1.8 presente nei repository *Ubuntu.





Una volta creato un progetto e configurato l'AUT seguendo il procedimento guidato (Test



→ New) , specificando con precisione la libreria con cui è stata scritta la GUI, sarà possibile iniziare la creazione della test suite. Come detto prima non c'è bisogno di avere un'applicazione funzionante per scrivere i casi di test per cui non è necessario configurare immediatamente l'AUT.

Una volta pronto l'ITE sarà possibile cliccando nel riquadro in basso a sinistra iniziare la creazione di un test case. Questa è solo una parte delle operazioni effettuabili con Jubula per la simulazione di una sequenza di operazioni. Come si può vedere le azioni sono divise per categorie.

A titolo di esempio creo una test suite chiamata “Acquista Auto”, il cui scopo è quello di creare una nuova auto, effettuare un controllo che venga aggiunta la relativa voce, creare una nuova

versione “speciale” per tale auto, e poi selezionarla per acquistarla. Verrà infine controllato

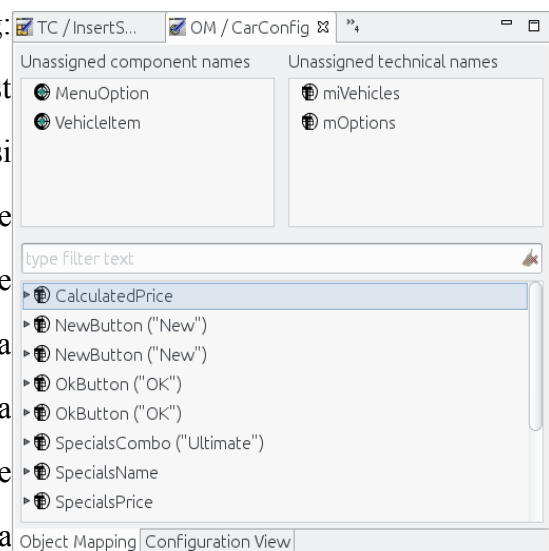
che il prezzo prodotto sia pari a quello preventivato. La test Suite è suddivisa in tre test case diversi per gestire al meglio i compiti. Per mostrare al meglio le potenzialità di Jubula andrò a creare i test case con entrambe le modalità di creazione messe a disposizione da Jubula. Per quanto riguarda la creazione manuale questa si fa trascinando nella parte centrale dello schermo l'operazione che si vuole compiere. In questo caso abbiamo bisogno di elementi della categoria: “Click”.

Una volta aggiunta una operazione vanno specificati diversi parametri tra i quali: Descrizione dell'operazione, parametro legato alla tipologia di elemento (Valore da inserire, operazione di confronto da effettuare etc), e nome della componente virtuale su cui agisce. Il nome da dare a questa componente deve essere scelta in maniera adeguata in modo tale da effettuare in maniera veloce e indolore il mapping.

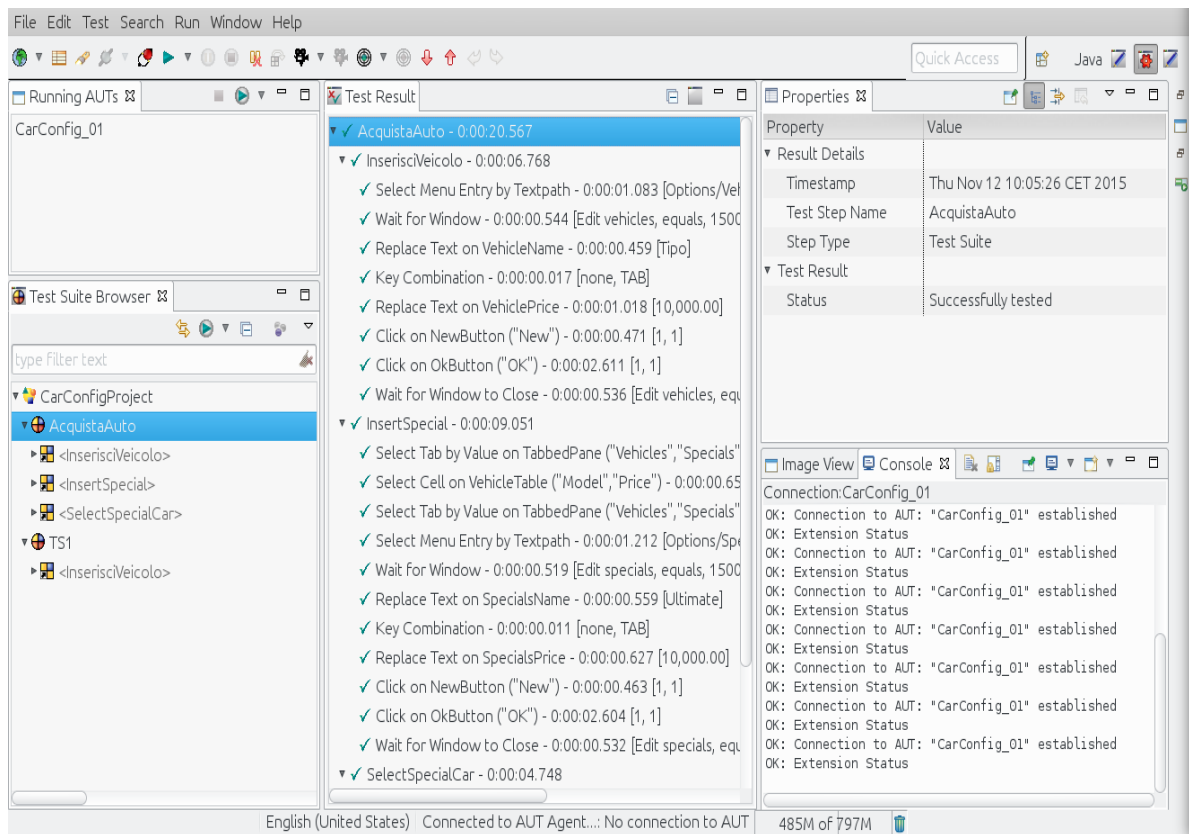
Questo è il risultato della creazione di un test case; è anche possibile combinare più test case insieme. Una volta fatto ciò è necessario creare una test suite tramite l'apposito riquadro e aggiungervi il test case appena creato.

Spesso mentre si è in modalità osservatore si avrà la necessità di effettuare +dei controlli per verificare la bontà della sequenza: per passare in modalità “check” bisognerà usare la combinazione di tasti “ctrl”+”shift”+”f11” e in seguito, dopo aver posto il mouse sopra la componente da controllare usare la combinazione “ctrl”+”shift”+”f12” per far apparire un dialogo dal quale scegliere quale operazione di controllo effettuare. Terminata la fase di creazione bisognerà provvedere al mapping:

innanzitutto cliccando col tasto destro sulla test suite (Open with→ Object Mapping Editor) si aprirà l'editor per effettuare mapping. Per fare ciò bisogna collegare l'ITE con l'AUT agent e quindi all'applicazione. L'AUT agent si avvia cliccando sul pulsante verde presente sulla toolbar e avviamo con il pulsante adiacente l'applicazione che prima abbiamo provveduto a



configurare. A questo punto sarà necessario selezionare quali componenti sono di nostro



interesse per l'esecuzione del test: per avviare il mapping bisognerà cliccare sul relativo pulsante mostrato nel primo screenshot. Per selezionare una componente bisognerà posizionarsi sopra col mouse, attendere che appaia un contorno verde attorno alla componente desiderata, cliccare e poi usare la combinazione di tasti “ctrl”+“shift”+“q” per registrarla. In questo modo verrà memorizzata fra le componenti tecniche non assegnate. Una volta registrate tutte le componenti d'interesse bisognerà collegare le componenti virtuali con quelle tecniche. Questa operazione come detto è sì dispendiosa ma rende il progetto di testing indipendente dalla GUI e quindi permette la creazione dei test case già quando sono noti i requisiti di sistema.

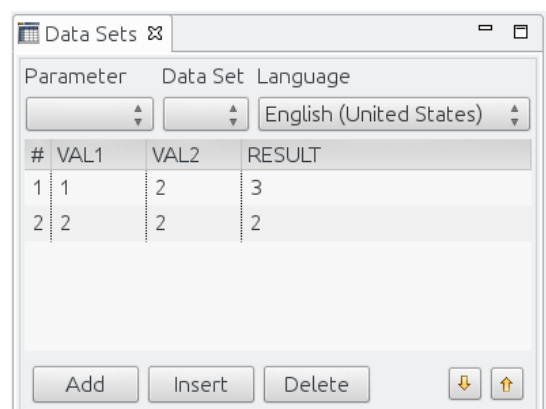
Una volta terminata la fase di mapping è possibile mandare in esecuzione il nostro test e verificare che tutte i controlli che abbiamo impostato vengano superati. Una volta salvato progetto e mandata in esecuzione l'applicazione sarà possibile selezionare una Test Suite dal riquadro a sinistra e mandarlo in esecuzione col pulsante verde soprastante.

Jubula inizierà a riprodurre il nostro comportamento e controllerà di volta in volta che le condizioni da noi poste siano vere o no; nel caso non riesca a verificarne una il test si interromperà immediatamente e verrà aperta la prospettiva relativa al “test report”. Se tutto

è andato a buon fine e il test non ha riportato errori troveremo delle spunte verdi affianco a ogni singolo step eseguito. Nel caso opposto invece l'errore verrà selezionato in automatico e verrà mostrata la tipologia di problema incontrato e in aggiunta verrà anche inserito uno screenshot dell'errore riscontrato.

Nel nostro caso la dipendenza dei dati di ingresso è pressochè nulla. Ma nei programmi in cui la correttezza dell'output dipende anche dai valori degli ingressi oltre che dalla sequenza di ingressi considerata è necessario trovare un modo per controllare in maniera automatica il risultato delle operazioni a valle di diversi ingressi: è possibile specificare fra le proprietà dell'operazione “ub_cti_replacetext” nel campo “TEXT[string]” il valore da inserire in un dato form. Invece di mettere un valore intero è possibile passargli una variabile “VAL1” antepoendo il nome della variabile con il simbolo di uguale.

L'esempio nell'immagine è preso da un semplice programma che fa l'addizione fra due numeri. Nel riquadro “Data sets” sarà ora possibile specificare l'insieme degli input e output attesi. Sarà addirittura possibile specificare più valore di ingresso e uscita in modo da testare il programma per più combinazioni di ingressi. Quello che rimane da fare è mandare in esecuzione il test e aspettare la sua terminazione. Un problema abbastanza serio riscontrato con Jubula è l'impossibilità di effettuare la registrazione delle checkbox all'interno di una tabella. Quando si tenta di fare il riconoscimento viene selezionata l'intera tabella invece del campo interessato. C'è un workaround per effettuare la selezione ma è abbastanza laborioso. Va selezionata la tabella, poi in un secondo momento va specificato a mano l'indice della riga e della colonna da considerare. Per questo motivo si è scelto di effettuare un test leggermente diverso da quello riportato col prossimo software di testing. Una delle potenzialità di Jubula è la



| # | VAL1 | VAL2 | RESULT |
|---|------|------|--------|
| 1 | 1 | 2 | 3 |
| 2 | 2 | 2 | 2 |

capacità di effettuare regressioni testing durante tutto il ciclo di sviluppo dell'applicazione. A ogni modifica che viene effettuata da parte degli sviluppatori può essere rieseguito i test precedentemente creati e controllare la presenza di nuovi malfunzionamenti nati dal

cambiamento del codice. Talvolta può capitare che una modifica renda il caso di test non eseguibile per il mancato riconoscimento di una componente; grazie alla modalità di creazione dei test case estremamente flessibile è possibile, tramite una modifica delle voci riguardante la parte di GUI modificata, ripristinare l'eseguibilità di un dato caso di test.

2.2 QF-Test

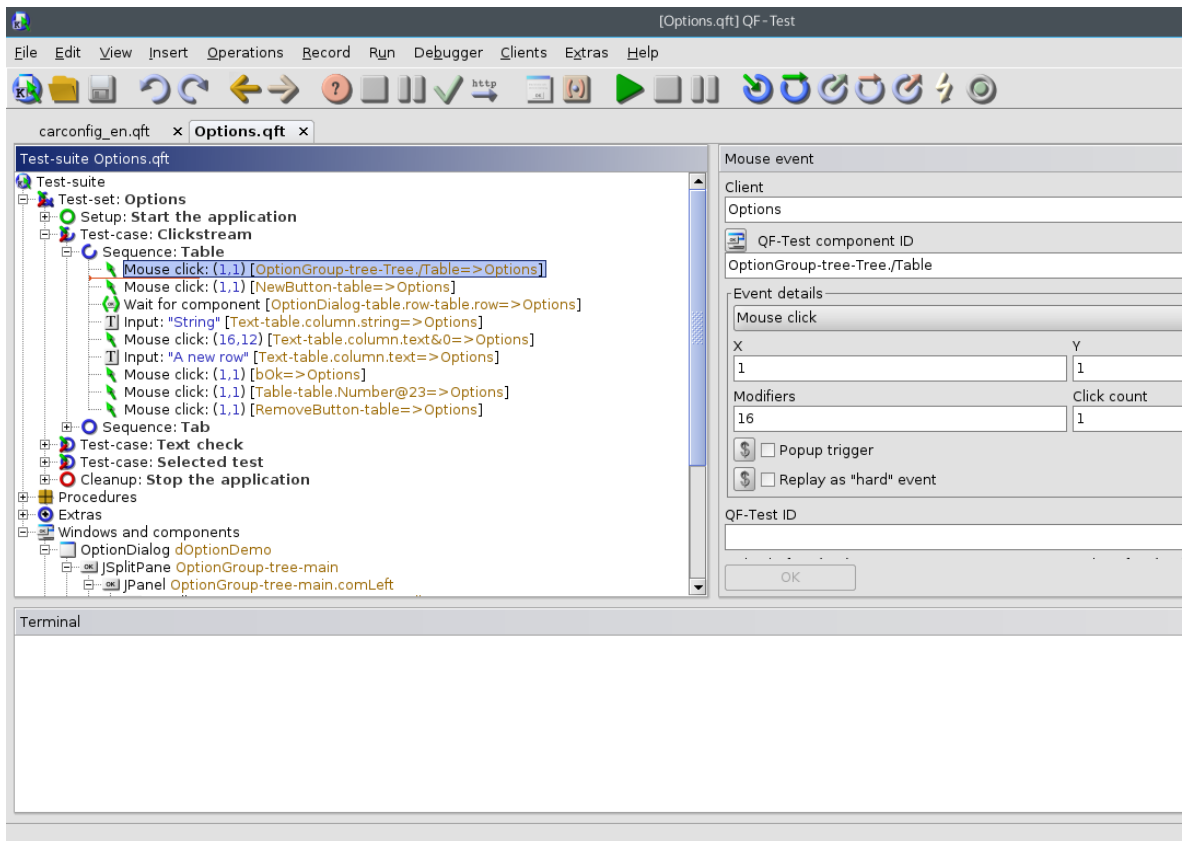
Uno fra i concorrenti più famosi di Jubula fra i tools di capture and Replay è QF-Test, un software proprietario che a differenza del precedente è a pagamento ma comunque liberamente scaricabile per effettuare una prova seppure con funzionalità limitate. Per provare al meglio tutte le caratteristiche di QF-Test è necessario chiedere una licenza di valutazione oppure accontentarsi di testarlo senza però poter salvare nuovi progetti di testing. La suite include tools per effettuare il testing di diverse interfacce grafiche, Swing, SWT, RCP, JavaFx nonché interfacce web.

Uno dei punti di forza di questo software è la presenza di una documentazione ricca e ben strutturata; ogni aspetto viene spiegato o nel manuale scaricabile e consultabile gratuitamente dal sito oppure è possibile avere una breve panoramica tramite dei video realizzati e caricati su Youtube dalla società che sviluppa il software.

Utilizzerò ora uno dei programmi rilasciati insieme all'applicazione per mostrare alcune delle peculiarità di QF-Test e in particolare come si procede alla creazione di una test suite. Possiamo notare nel lato sinistro una test suite che in questo caso ha un solo test-set ma che in realtà potrebbe contenerne più di uno. Un test-set è un insieme di casi di test che eseguono una specifica funzione; al suo interno troviamo un test case il quale contiene le sequenze di azioni. Questa ripartizione gerarchica è molto utile al test designer per gestire la complessità. È possibile in questa maniera organizzare il caso di test come un'insieme di attività elementari che messe assieme possono creare un caso più complesso. Il vantaggio di tale approccio bottom-up, oltre alla creazione di unità funzionali semplici, è che posso riutilizzare queste unità all'interno di altri casi di test.

Come è possibile notare le singole azioni elementari hanno, dopo il nome che descrive

l'azione, una serie di informazioni che vanno a descrivere su quale oggetto verrà eseguita l'operazione. Per capire meglio il funzionamento di QF-Test passiamo a analizzare il riquadro a destra che fornisce alcune informazioni circa l'azione selezionata, in questo caso il click, e il componente su cui opera. Il campo client è semplicemente l'identificatore



del SUT (system under test); il campo che ci interessa analizzare invece è il campo “QF-Test Component id”. Questo identificatore viene usato da QF-Test per effettuare i collegamenti fra azioni e componenti. Un componente è un widget o parte di esso ed è identificato mediante questo id, per questo motivo esso deve essere univoco. Questo garantisce che il programma riesca a ritrovare in maniera automatica il componente a cui si riferisce. Un altro campo di fondamentale importanza è il campo “name”: questo serve invece a identificare una specifica componente della SUT.

I campi “name” e “qf-test id” anche se spesso coincidono rappresentano due concetti diversi. Come già detto il primo serve a identificare una componente in modo tale da permettere al programma di testing di capire su quale componente virtuale agire allo scatenarsi di un evento; il secondo invece è legato all'implementazione della GUI stessa,

serve infatti a identificare un particolare componente sul SUT. Se viene cambiato il nome di una componente sarà necessario aggiornare anche il campo “name” in maniera da farli corrispondere. Per la precisione QF-Test utilizza dei namespace in maniera tale da limitare la necessità degli attributi univoci alle sole componenti di una classe di componenti, un textfield e un pulsante potrebbero benissimo avere lo stesso nome. È preferibile scegliere all'inizio del progetto i nomi delle diverse componenti in maniera da ridurre gli errori legati all'impossibilità di ritrovare un componente e quindi anche il successivo tempo perso nell'aggiornare i campi “name”.

Il nome a ogni componente viene assegnato durante la registrazione di una sequenza in maniera automatica andando a copiare il nome del tipo di componente selezionata e andando ad aggiungere un numero progressivo in maniera tale da distinguere i vari elementi della stessa classe. Il nome viene poi utilizzato per generare automaticamente il “qf-test id”. Per tale motivo è conveniente assegnare in maniera manuale un id in maniera tale che sia adatto all'oggetto a cui si riferisce anziché lasciare il nome automaticamente assegnato. Questo comportamento è facilitato da un tool all'interno di QF-Test che permette l'aggiornamento di tutti i riferimenti a una data componente una volta che il nome o l'id viene modificato.

Dopo questa breve descrizione dei meccanismi fondamentali che sono alla base del funzionamento di QF-Test torniamo ad analizzare il caso di test precedente.

Nel test-case “ClickStream” vi sono due sequenze “Table” e “Tab”. Le operazioni che eseguono sono molto semplici: “Mouse Click”, “Wait for Component”, “Input”. I comandi sono autoesplicativi. Il primo clicca sulla componente identificata dal “qf-test id” mentre il secondo attende per un tempo indicato nel campo timeout, il terzo inserisce all'interno di un campo di tipo textfield un dato valore in di input. “Wait for component” è utilizzata per creare un ritardo nell'esecuzione dell'operazione successiva. Se il click su una componente come un bottone porta alla creazione di una nuova componente e l'azione successiva deve lavorare proprio su quest'ultima che sta venendo generata, potrebbe succedere che l'operazione successiva non trovando la componente pronta all'uso generi un'eccezione (ComponentNotFoundException). Questa viene lanciata quando viene apportata una

modifica alla struttura della GUI o ai relativi identificativi.

Accanto ai Test-case troviamo due voci: Setup e CleanUp. Il primo è una sequenza di operazioni che vengono effettuate per preparare l'applicazione all'esecuzione. Viene generata in automatico tramite un wizard eseguibile mediante il menù “extras→ quickstart guide”. Questo processo guidato aiuta l'utente a configurare il progetto di test e definire le operazioni da compiere per far partire i test; in particolare serve a:

- Scegliere l'applicazione da testare
- Definire la directory di lavoro
- Versione di Java da utilizzare

In più verranno create delle operazioni che inizializzano l'applicazione e la mettono in esecuzione attendendo che l'applicazione e QF-Test abbiano stabilito una connessione.

In particolare le operazioni di Setup e di Cleanup non sono presenti solo all'inizio e alla fine del test case principale ma possiamo trovarle all'interno di sequenze e sotto sequenze. Sostanzialmente il loro scopo è quello di portare la GUI in uno stato stabile e fare in modo che i test che li succederanno possano eseguire senza alcun problema dovuto alle modifiche apportate dal test case attuale.

Vi sono inoltre i nodi “dependency” i quali servono a richiamare delle procedure già definite che riportano l'applicazione in uno stato stabile.

Un'altra funzionalità offerta da QF-Test è quella di supporto al data driven testing. Questa tipologia di testing si effettua fornendo al programma una serie di input e output previsti e controllando la veridicità di tali condizioni al termine dell'esecuzione del test. Per fare in ciò QF-Test basta creare un nodo 'Data Driver' il quale conterrà al suo interno una data sequenza da eseguire. All'interno del “Data Driver” è possibile specificare i diversi ingressi da inserire e i risultati da controllare. La sequenza verrà iterata per ogni riga presente all'interno del set di dati. Ovviamente sarà necessario definire delle sequenze di setup e cleanup per portare l'applicazione in uno stato stabile e quindi rieseguire il tutto senza problemi. I dati possono essere memorizzati in diversi formati: sia in una tabella integrata in QF-Test, oppure in un file esterno come un documento excel, un cvs o un database.

Si possono addirittura avere contemporaneamente più data table o comunque più sorgenti, in maniera tale da combinare diversi input testando in maniera incrociata i valori di una tabella con quelli di un'altra.

L'ultima funzionalità base da mostrare è quella di “check” ovvero di controllo. Per determinare la positività o meno di un test c'è la necessità di controllare se una data condizione è verificata o no. Questo lo possiamo fare tramite i diversi tipi di “check” che QF-Test ci mette a disposizione. Mentre stiamo registrando il nostro caso di test possiamo interrompere la registrazione premendo la spunta verde sulla toolbar per poi cliccare col tasto destro sulla componente da controllare. A seconda della tipologia di componente su cui abbiamo cliccato appariranno una lista di possibili controlli da effettuare. Un esempio banale è quello di una label che riporta in maniera corretta il valore di una data elaborazione: in questo caso ad esempio si avrà dovrà scegliere il CheckText.

Anche con QF-Test è possibile effettuare regression testing. Periodicamente è possibile lanciare in esecuzione i test precedentemente creati e controllarne l'esito.

Invece di avviare l'intera applicazione è possibile eseguire i test in batch mode: da riga di comando è possibile mandare in esecuzione un'intera test suite.

Il comando è “qftest -batch -run /home/tester/suiteA.qft” . L'opzione -batch indica che non verrà lanciata l'interfaccia grafica, il secondo manderà in esecuzione la test suite specificata nel percorso riportato. È possibile anche eseguire test da remoto grazie alla daemon mode: QF-Test resterà in attesa su una porta a scelta in attesa di richiesta dei vari tester che inoltreranno le richieste all'istanza sulla macchina server.

2.3 Un confronto

Entrambi i software analizzati sono usati in ambito professionale in quanto ormai hanno raggiunto uno stato di maturità elevato ma soprattutto perché entrambi vengono mantenuti e aggiornati in maniera tale da supportare le nuove versioni di Java. I due software sono proprietari con una differenza legata ai costi: Jubula è liberamente scaricabile e utilizzabile mentre QF-Test è a pagamento con una licenza singola il cui costo si aggira attorno ai 1000€. Nonostante questo Jubula viene mantenuto da una azienda che fornisce supporto a

360 gradi. Una differenza sostanziale fra i due tools finora analizzati è che appartengono alla categoria di capture and playback ma Jubula permette anche un'altra modalità di creazione dei test case per cui può venire considerato un software ibrido. Questo di per sé è un vantaggio per i motivi già riportati sopra: è molto più adatto rispetto a QF-Test ad uno sviluppo software con un modello di sviluppo testing driven. Inoltre è molto più resistente rispetto quest'ultimo alla modifica dei requisiti e quindi della struttura della GUI rispetto al suo concorrente. Il meccanismo del mapping infatti serve proprio a staccare il test dall'applicazione reale. Se verrà modificata la posizione di un widget basterà rieffettuare il mapping di quella singola componente.

QF-Test in realtà permette l'aggiunta di qualche nodo all'interno di un test case già creato ma questo di certo non basta per la creazione di un test case da zero; al più può essere utile se bisogna modificarne uno già esistente come nel caso del regression testing in cui si ha la necessità di adattare il test case per aderire alla modifica fatta al programma.

Per il resto entrambi hanno un'interfaccia grafica semplice e minimale, entrambi i software sono relativamente facili da studiare in quanto, come già accennato, posseggono una documentazione ricca e completa contornata di esempi svolti e spiegati. L'apprendimento veloce è facilitato dal fatto che nessuno dei due richiede conoscenze tecniche, una volta capito il funzionamento del programma e conoscendo il comportamento atteso del programma da testare, sarà semplice creare Test Case senza l'aiuto né di programmatori né di ingegneri del software.

Essendo l'attività di testing un'attività da effettuare in team è necessario disporre di un software che supporti più piattaforme in maniera tale da non dover comprare macchine ad hoc o adattare sistemi già esistenti. Entrambi i software girano tranquillamente su Linux e su Windows, in più Jubula funziona anche su MacOS. Entrambi i software mettono a disposizione tecniche per lavorare da remoto alla stesura dei casi di test lavorando su database e AUT agent posti su qualche macchina remota. Non è possibile però prendere un test case scritto con un programma e importarlo nell'altro.

Entrambi i software vengono supportati da tools esterni per la gestione dei test case e la memorizzazione dei relativi risultati. Ad esempio Qmetry è in grado di elaborare il test

case e inviarlo in esecuzione su QF-Test.

Un vantaggio notevole che ha Jubula rispetto a QF-Tools è l'integrazione completa in Eclipse. Per questioni di licenza Jubula non può esser distribuito assieme all'IDE ma è comunque installabile tramite repository esterni. In questa maniera è possibile sfruttare tutti i tools presenti per Eclipse andando a sopperire quelle mancanze presenti in Jubula come la generazione di documentazione per i casi di test; al contrario QF-Test può generare in automatico della documentazione in formato HTML inserendo le dovute informazioni nel campo commenti di ogni elemento presente in una test suite.

Conclusioni

Nel mondo del testing delle GUI le alternative ci sono e sono varie, tutte ovviamente orientate a un target aziendale. I due programmi analizzati differiscono in primo luogo per il costo della licenza che però potrebbe essere un fattore irrilevante nel caso di una grande azienda. Per il resto i due software sono completi e hanno raggiunto un buon grado di maturità. Un vantaggio di Jubula sul suo concorrente è la presenza di una vasta community che ne supporta lo sviluppo. La possibilità di scaricarlo e la possibilità di modificarlo è un incentivo per chiunque a contribuire apportando modifiche e aggiungendo funzionalità. Sotto questo punto di vista una community del genere, comunque immersa nell'ecosistema Eclipse, ha un notevole vantaggio rispetto a una singola azienda.

L'unica considerazione che si potrebbe fare nella scelta di uno dei due è sul modello di sviluppo che si intende adottare nello sviluppo del progetto. Jubula permette di creare dei test case prima ancora di avere iniziato a scrivere una sola linea di codice; può essere preferibile per questo motivo nel caso si voglia adottare un modello di sviluppo test driven. Per quanto riguarda il campo mobile Jubula è in grado di testare anche apps IOS mentre per ora non è ancora stato programmato un supporto ad applicazioni Android; QF-Test invece non è in grado di testare né l'uno né l'altro.

Un'altra prova della bontà dei due progetti viene dimostrata dal fatto che Jubula è riuscito a resistere per tutti questi anni alla disgregazione di diversi programmi di testing dovuta alla difficoltà delle diverse community che li mantenevano a stare dietro allo sviluppo tecnologico(di Java fra tutti); così come QF-Test che dal 2008 è ancora in attività e pienamente supportato.

Bibliografia

- [1] Porfirio Tramontana, Slides Ingegneria Del software II : User Interface Testing
- [2] Atif M. Memon, Martha E. Pollack, Mary Lou Soffa, Using a Goal-driven Approach to Generate Test Cases for GUIs , University of Pittsburgh
- [3] Kanchan Gautam , Madhuri Sharma, An Approach to Generate the Test Cases for GUI Testing.
- [4] Wikipedia, https://en.wikipedia.org/wiki/Graphical_user_interface_testing
- [5] Izzat Alsmadi, Using genetic algorithms for test case generation and selection optimization.