Piano di Testing

Rev 1.1 (25/02/2017)

Progetto Crimson

Ingegneria del Software (A.A. 2016/17)

Gruppo MALG
Cavallo Mattia (845900@stud.unive.it)
Ceolin Andrea (846559@stud.unive.it)
Naccari Laura (846196@stud.unive.it)
Patti Giulio (847665@stud.unive.it)

Sommario

1. Introduzione..... 1
2. Glossario..... 2
3. Organizzazione e modalità di test..... 3
4. Tracciabilità dei requisiti..... 3
5. Elementi sottoposti al test..... 4
6. Pianificazione temporale (*schedule*)..... 5
7. Raccolta ed interpretazione dei risultati..... 5
8. Requisiti hardware e software..... 6

1. Introduzione

Il gruppo MALG ha avanzato come proposta una “app” che permetterà all’utente finale di ottenere informazioni circa la sicurezza di determinate sostanze additive presenti negli alimenti e nei prodotti cosmetici, in particolare quelli acquistati in (o provenienti da) paesi extra-UE, facendo riferimento ai data-set pubblicati dall’Unione Europea relativi all’approvazione o all’applicazione di un divieto sull’utilizzo di questi ultimi.

Dal momento che la scelta del nome del progetto avverrà presumibilmente durante una fase successiva a quella dello sviluppo, ma comunque precedente a quella della pubblicazione, tutte le repository, la documentazione interna e il codice faranno uso del codename *Crimson* per riferirsi al progetto stesso.

Il fine del presente documento è quello di descrivere dettagliatamente le modalità di test dell’applicazione proposta. A tal proposito, verranno analizzati:

- L’organizzazione e le modalità di svolgimento dei test,
- La tracciabilità dei requisiti,
- La definizione degli elementi sottoposti a test,
- La pianificazione temporale del processo di test,
- Le modalità di raccoglimento e di interpretazione degli esiti,
- I requisiti posti sui dispositivi dei tester,
- I vincoli da rispettare al fine di assicurare l’attendibilità dei test.

2. Glossario

- **Ambiente di sviluppo**: raccolta di strumenti destinati agli sviluppatori nell'ambito della produzione di progetti software.
- **Android**: sistema operativo per dispositivi mobili, in particolare smartphone e tablet.
- **App**: neologismo che sta per "applicazione (software)". Solitamente, con questo termine, siriferisce ad applicazioni aventi per target dispositivi mobili.
- **Beta-tester**: *tester* selezionati che si occupano della terza fase di test (ved. paragrafo 6)
- **Black-box testing**: "test a scatola chiusa", ovvero test delle funzionalità di un prodotto indipendente dall'implementazione di quest'ultimo.
- **Bottom-up**: letteralmente "dal basso verso l'alto", strategia che prevede lo sviluppo (o in questo caso il test) di un prodotto partendo dalle sue componenti elementari (particolare) al prodotto finale (generale).
- **Bug**: problema semantico nel *software* che può dare origine ad anomalie.
- **Build**: versione eseguibile (compilata) di un programma, originata dal *codice sorgente*.
- **Classi**: componenti fondamentali dell'app.
- **Code-name**: denominazione interna riferita ad un prodotto in fase prototipale.
- **Codice sorgente**: rappresentazione testuale di un programma, da cui si generano le *build*.
- **Dataset**: insieme di dati statici (o aggiornati periodicamente in toto), organizzati in maniera analoga ad un *database*.
- **Diagrammi delle classi**: consentono di descrivere tipi di entità, con le loro caratteristiche e le eventuali relazioni fra questi tipi.
- **Dispositivo**: meccanismo, congegno o elemento (fisico) che, da solo o inserito in un meccanismo più complesso serve per compiere una determinata funzione.
- **Emulatore**: ricostruzione *software* dell'architettura *hardware* di un *dispositivo* (fisico).
- **Hardware**: insieme delle componenti elettroniche di un *dispositivo*.
- **ID**: abbreviazione, tipica della lingua inglese, per "identificatore".
- **Input**: dati di partenza di un'elaborazione.
- **Interfaccia utente**: elementi del programma con cui l'utente può interagire.
- **Memoria fisica**: memoria *hardware* necessitata da un *dispositivo* per il suo funzionamento.
- **Output**: indica il risultato di una elaborazione ed in senso più ampio il risultato o l'insieme dei risultati prodotti.
- **Programma**: è un insieme di istruzioni che produce soluzioni per una data classe di problemi automatizzati.
- **Repository**: "deposito" per codice sorgente e documentazione progettuale, che mantiene inoltre traccia delle modifiche apportate al contenuto, al fine di permetterne il ripristino in qualunque momento.
- **Runtime**: tempo in cui un *programma* è in esecuzione.
- **Software**: termine generico che definisce programmi e procedure utilizzati per far eseguire al dispositivo un determinato compito.
- **Tester**: persona incaricata ad una o più attività di test di un prodotto.
- **UML (Unified Modeling Language)**: Linguaggio universale di modellazione e specifica.
- **UI**: abbreviazione, tipica della lingua inglese, per *interfaccia utente*.

3. Organizzazione e modalità di test

Analogamente al processo di sviluppo, si segue per il test il modello **bottom-up**; ovvero si svilupperanno e si testeranno prima le componenti elementari dell'app, per poi verificare la corretta integrazione delle stesse nel prodotto finale.

Ciò sarà complementato dall'utilizzo della strategia ***black-box testing*** al fine di accertare la massima correttezza possibile nella generazione delle risposte da fornire all'utente, indipendentemente dall'implementazione; dapprima utilizzando un dataset artificiale creato ad-hoc, ed un set di interrogazioni predefinite (avendo così la certezza dell'immutabilità degli esiti, e dunque la possibilità di automatizzare questi controlli), ed infine utilizzando il dataset reale ed incaricando i *beta-tester* con il compito di effettuare interrogazioni e riportare eventuali anomalie o sospette anomalie.

4. Tracciabilità dei requisiti

Tutti i requisiti, funzionali e non, saranno verificati (automaticamente laddove possibile, o manualmente negli altri casi) secondo le disposizioni specificate nel documento di *Analisi dei Requisiti*.

Segue l'elenco dei test automatizzati che dovranno essere lanciati prima del rilascio ed al completamento di ognuna della fasi implementative illustrate al par 6:

Unità	Test	Descrizione
t01_dataset	**#1**	URL dataset remoto pre-impostato all'avvio dell'app.
	#2	Download del dataset e verifica riempimento DB locale.
	#3	Verifica post-download che il DB locale risulti pronto.
	#4	Verifica post-download che la versione locale sia non nulla.
	#5	Un successivo tentativo di download dev'essere riconosciuto come non necessario, dunque saltato automaticamente.
t10_additive		Si utilizza l'ingrediente "Carotenes" come riferimento.
	#1	Verifica che il nome corrisponda a "Carotenes".
	#2	Verifica che l'E-number sia "160a"
	#3	Verifica che il nome visualizzato sia "Carotenes (E160a)"
	#4	Verifica che lo stato di approvazione sia positivo.
	#5	Verifica che il timestamp di approvazione sia non nullo.
	#5	Verifica che la documentazione allegata sia fornita.
t20_request		Si genera una richiesta {"Carotenes", "Dihydrogen Monoxide"}
	#1	Verifica che il timestamp di scansione sia non nullo.
	#2	Dopo ricalcolo forzato, si verifica che il timestamp rimanga inalterato.
t30_responses		Si utilizza una richiesta analoga a quella usata in t20
	#1.1	Verifica che la prima query testuale corrisponda a "Carotenes"
	#1.2	Verifica che la prima risposta sia generata con successo

t30_responses	**#1.3**	Verifica che l'esito per la prima richiesta sia positivo.
	#1.4	Verifica che l'esito originale per la prima richiesta sia positivo.
	#1.5	Verifica che l'ingrediente trovato per la prima risposta corrisponde effettivamente all'ingrediente "Carotenes".
	#1.6	Verifica che timestamp e timestamp originale coincidano a seguito di un solo tentativo di determinazione della risposta.
	#1.7	Verifica che timestamp e timestamp originale non corrispondano più (con timestamp > timestampOriginale) a seguito di ricalcolo della risposta.
	#2.1	Verifica che la seconda query testuale corrisponda a "Dihydrogen Monoxide".
	#2.2	Verifica che la seconda risposta fallisca per ingrediente non trovato nel dataset locale
	#2.3	Verifica che non si verifichino eccezioni al tentativo di recuperare l'ingrediente collegato alla risposta, e che venga ritornato invece il valore speciale null.
	#2.4, #2.5	Verifiche su timestamp e timestamp originale analoghe a quelle effettuate per #1.6 e #1.7.

5. Elementi sottoposti al test

Saranno sottoposti a test automatico i **metodi pubblici delle classi** del modello dell'app (elencati nel diagramma UML presente al paragrafo 5.3 del *Documento di Progettazione*), mano a mano che vengono implementati, partendo dapprima da quelli che non prevedono l'integrazione con le altre classi, per poi spostarsi a quelli che coinvolgono due o più componenti.

Per assicurare la coerenza dei risultati, i test faranno uso di un dataset artificiale preparato opportunamente allo scopo.

Verrà inoltre posta particolare attenzione al corretto **avvio e arresto** dell'app, alla corretta visualizzazione di tutti gli elementi della **UI** sullo schermo del dispositivo utente (su emulatori e dispositivi reali con diverse dimensioni, orientamento e proporzioni dello schermo), e alla capacità dell'app di operare su insiemi di dati (artificiali) molto estesi (**test di carico** con dataset artificiale dalle 2 alle 4 volte più grande del dataset reale - prendendo come misura di riferimento il numero di righe per ogni tabella al momento della progettazione).

6. Pianificazione temporale (*schedule*)

La pianificazione temporale dell'attività di test seguirà il presente modello:

- **Fase implementativa**: Vengono effettuati solamente i test automatici, limitatamente alle parti sviluppate dell'applicazione. Gli esiti di questi test, molto probabilmente fallimentari, guideranno gli sviluppatori nella corretta implementazione delle classi di base.
  - **Build interne parziali**: I test automatici saranno avviati automaticamente alla compilazione. Se tutti questi riportano uno stato di successo, lo sviluppatore avrà la possibilità di accertare manualmente l'assenza di problemi legati alla UI e/o di situazioni di anomalia non rilevate dal test automatico. Nel caso ve ne fossero, potrà intervenire di conseguenza, o segnalare l'esistenza di un bug nell'apposita sezione "*Issues*" della repository contenente il codice sorgente.
- **Fase prototipale (*alpha-testing*)**: Sarà distribuita una versione embrionale (ma funzionalmente completa) dell'app a tutti gli sviluppatori, che farà uso di un dataset artificiale creato per lo scopo, i quali dovranno provare ad utilizzarla, sia in situazioni controllate (input di sequenze di ingredienti il cui esito è noto a prescindere), sia in situazioni reali (input di sequenze arbitrarie); verificando inoltre che i requisiti del progetto siano soddisfatti (come illustrato al paragrafo 4), e che l'app si comporti normalmente anche in presenza di un dataset particolarmente grande (test di carico, nelle modalità illustrate al paragrafo 5). Tutti i problemi riscontrati andranno segnalati nell'apposita sezione "*Issues*" della repository, e verrà rilasciata una nuova *alpha-build* (identificata da un numero progressivo) fino a quando non saranno stati corretti tutti i bug segnalati.
- **Fase di pre-rilascio (*beta-testing*)**: Sarà distribuita una versione sperimentale dell'app a tutti gli sviluppatori e ad un insieme ristretto di utenti selezionati, marcata come *candidata al rilascio*, che farà uso del dataset reale e potrà essere utilizzata arbitrariamente dall'utente. Eventuali problematiche o situazioni di anomalia andranno segnalate nell'apposita sezione "*Issues"* della repository (da parte degli sviluppatori), o notificate per e-mail alla casella del gruppo: malg.unive@gmail.com (nel caso dei beta-tester esterni al team di sviluppo). Verrà rilasciata una nuova *beta-build* (identificata anch'essa da un numero progressivo) fino a quando non vi saranno ulteriori segnalazioni. La prima versione esente da bug noti sarà considerata valida per il rilascio.

7. Raccolta ed interpretazione dei risultati

I test automatici verteranno nell'invocare determinati metodi delle classi dell'app con degli input specifici, e nel verificare la coerenza nella restituzione dell'output e/o il verificarsi di determinate condizioni (ad esempio, che non vengano generate eccezioni, o che lo stato di altri oggetti venga mutato di conseguenza). Il lancio e la raccolta/visualizzazione dei risultati di questi test avverranno internamente all'ambiente di sviluppo.

A *runtime*, l'applicazione genererà una serie di registri diagnostici (*log-file*), i quali andranno allegati alle eventuali segnalazioni ed esaminati dagli sviluppatori, al fine di identificare anomalie occulte e/o scoprire le cause di altri problemi precedentemente segnalati.

La corretta visualizzazione dell'interfaccia grafica, e il rispetto dei requisiti progettuali, andranno invece accertati manualmente da parte degli sviluppatori (e limitatamente alla UI, da parte dei tester), facendo riferimento laddove necessario alla documentazione progettuale redatta.

8. Requisiti hardware e software

Al fine di effettuare i test, i dispositivi degli utenti *tester*, cosi come gli eventuali emulatori, dovranno soddisfare i seguenti requisiti:

- **Android 4.2** (***Jelly Bean***) o superiore,
- Almeno **512MiB** di **memoria fisica** totale (**RAM**),
- Connessione ad **Internet** attiva **al primo avvio** dell'app,
- Almeno **200MiB di spazio libero** per l'archiviazione del dataset e dei registri diagnostici,
- Possibilità di **inviare e-mail** per la segnalazione di eventuali problematiche.

9. Vincoli per l'attività di test

Al momento non sono stati posti vincoli per l'attività di test, se non quello di non protrarsi oltre la scadenza limite prevista per il completamento del progetto (28 febbraio 2017).

Appendice A: *Elenco dei cambiamenti*

Segue l'elenco delle modifiche apportate al presente documento, revisione per revisione.

Rev 1.0 - 20/12/2016

Stesura iniziale del piano.

Rev 1.1 - 25/02/2017

Aggiunta tabella riassuntiva dei test automatizzabili alla sezione 4 (tracciabilità dei requisiti).