



ITAlian Software Testing Qualifications Board

Syllabus Foundation

Versione 2011

DATI IDENTIFICATIVI

| CODICE DOCUMENTO | DATA DI EMISSIONE | STATO | |
|------------------------|-------------------|---|---|
| ITASTQB-FLSY-110501-01 | 01/05/2011 | <input checked="" type="checkbox"/> REDATTA | <input checked="" type="checkbox"/> APPROVATA |

| REDATTORI | DATA |
|-------------|------------|
| M. SOGLIANI | 10/04/2011 |

| APPROVATORI | DATA |
|-------------|------------|
| G. BAZZANA | 01/05/2011 |

Associazione ITA-STQB

Sede Legale e Amministrativa: 25125 Brescia - Via Brozzoni 9
C.F. 03265510176 P.I 03121180982
e-mail info@ita-stqb-org

STORIA DELLE MODIFICHE

| CHI | DATA | VERSIONE | CONTENUTO |
|---------------------------------------|-------------------|----------|---|
| Alessandro Collino, Marco Sogliani | 01 Giugno 2007 | 01 | Non applicabile in quanto si tratta della prima versione del documento |
| Marco Sogliani | 01 Mag 2010 | 02 | Il syllabo è stato allineato alla nuova versione ufficiale emessa da ISTQB® |
| Gualtiero Bazzana | 14 Ott 2010 | 03 | Apportate alcune modifiche minori per migliorare alcuni aspetti di traduzione |
| Marco Sogliani | 01 Mag 2011 | 04 | Il Syllabo è stato allineato alla nuova versione 2011 emessa da ISTQB® |

Nota Copyright © International Software Testing Qualifications Board (in seguito chiamato ISTQB®)

ISTQB® è un marchio registrato dell' International Software Testing Qualifications Board,

Copyright © 2011 gli autori per le modifiche 2011 (Thomas Müller (chair), Debra Friedenber, and the ISTQB® WG Foundation Level).

Copyright © 2010 gli autori per le modifiche 2010 (Thomas Müller (chair), Armin Beer, Martin Klouk, Rahul Verma).

Copyright © 2007 gli autori per le modifiche 2007 (Thomas Müller (chair), Dorothy Graham, Debra Friedenber e Erik van Veendental)

Copyright © 2005, gli autori (Thomas Müller (chair), Rex Black, Sigrid Eldh, Dorothy Graham, Klaus Olsen, Maaret Pyhäjärvi, Geoff Thompson e Erik van Veendental).

Tutti i diritti riservati.

Gli autori stanno trasferendo il copyright alla 'International Software Testing Qualifications Board (ISTQB®)'. Gli autori (come attuali possessori del copyright) e ISTQB® (come futuro possessore del copyright) si sono accordati con riferimento alle seguenti condizioni d'uso:

- 1) Ogni persona o azienda di formazione può usare questo syllabus come base per un corso di formazione se gli autori e ISTQB® sono riconosciuti come la fonte e i detentori del copyright del syllabus e a condizione che ogni annuncio di corso di formazione possa menzionare il syllabus solo dopo la presentazione per l'accreditamento ufficiale del materiale formativo ad una National Board riconosciuta da ISTQB®.
- 2) Ogni persona o gruppo di persone possono usare questo syllabus come base per articoli, libri, o altre pubblicazioni derivate se gli autori e ISTQB® sono riconosciuti come la fonte e i detentori del copyright del syllabus.
- 3) Ogni National Board riconosciuta da ISTQB® può tradurre questo syllabus e può autorizzare l'utilizzo del syllabus (o della traduzione) a terzi.

INDICE DEI CONTENUTI

| | |
|--|-----------|
| Storico delle modifiche del SILLABO Originale in Lingua Inglese | 9 |
| Prefazione | 10 |
| Ringraziamenti..... | 10 |
| Introduzione a questo syllabus | 11 |
| Scopo del presente documento | 11 |
| Il ‘Certified Tester Foundation Level’ nel Software Testing | 11 |
| Obiettivi dell’apprendimento/livello di conoscenza..... | 11 |
| L’esame..... | 11 |
| Accreditamento..... | 12 |
| Livello di dettaglio..... | 12 |
| Come è organizzato questo syllabus..... | 12 |
| 1. Fondamenti del testing (K2) | 13 |
| Obiettivi di apprendimento per i fondamenti del testing..... | 13 |
| 1.1. Perché il testing è necessario (K2)..... | 14 |
| 1.1.1. Termini | 14 |
| 1.1.2. Contesto dei sistemi software (K1)..... | 14 |
| 1.1.3. Cause dei difetti del software (K2)..... | 14 |
| 1.1.4. Ruolo del testing nello sviluppo, manutenzione e funzionamento del software (K2)..... | 14 |
| 1.1.5. Testing e qualità (K2) | 14 |
| 1.1.6. Quando il testing può essere considerato sufficiente? (K2) | 15 |
| 1.2. Che cos’è il testing? (K2) | 15 |
| 1.2.1. Termini | 15 |
| 1.2.2. Contesto | 15 |
| 1.3. Principi generali del testing (K2)..... | 16 |
| 1.3.1. Termini | 16 |
| 1.3.2. Principi..... | 16 |
| 1.4. Fondamenti del processo di test (K1) | 17 |
| 1.4.1. Termini | 17 |
| 1.4.2. Contesto | 17 |
| 1.4.3. Pianificazione e controllo del test (K1) | 18 |
| 1.4.4. Analisi e progettazione del test (K1) | 18 |
| 1.4.5. Implementazione ed esecuzione del test (K1) | 18 |

| | | |
|-----------|---|-----------|
| 1.4.6. | Valutazione dei criteri di uscita e report dei risultati (K1) | 19 |
| 1.4.7. | Attività di chiusura del test (K1) | 19 |
| 1.5. | La psicologia del testing (K2)..... | 19 |
| 1.5.1. | Termini | 19 |
| 1.5.2. | Contesto | 19 |
| 1.6. | Codice Etico (K2) | 21 |
| 1.7. | Riferimenti..... | 21 |
| 2. | Il testing durante il ciclo di vita del software (K2) | 22 |
| | Obiettivi di apprendimento per il testing durante il ciclo di vita del software | 22 |
| 2.1. | Modelli di sviluppo del software (K2)..... | 22 |
| 2.1.1. | Termini | 22 |
| 2.1.2. | Contesto | 22 |
| 2.1.3. | Modello a V (modello di sviluppo sequenziale) (K2) | 22 |
| 2.1.4. | Modelli di sviluppo iterativo-incrementale (K2)..... | 23 |
| 2.1.5. | Il testing in un modello di ciclo di vita (K2) | 23 |
| 2.2. | Livelli di test (K2) | 24 |
| 2.2.1. | Termini | 24 |
| 2.2.2. | Contesto | 24 |
| 2.2.3. | Testing di componente (K2) | 24 |
| 2.2.4. | Testing di integrazione (K2)..... | 25 |
| 2.2.5. | Testing di sistema (K2)..... | 25 |
| 2.2.6. | Testing di accettazione (K2)..... | 26 |
| 2.3. | Tipi di test (K2) | 28 |
| 2.3.1. | Termini | 28 |
| 2.3.2. | Contesto | 28 |
| 2.3.3. | Testing di funzioni (testing funzionale) (K2) | 28 |
| 2.3.4. | Testing di caratteristiche software non funzionali (testing non funzionale) (K2)..... | 28 |
| 2.3.5. | Testing dell'architettura/struttura del software (testing strutturale) (K2)..... | 29 |
| 2.3.6. | Testing legato a modifiche (retesting e regression testing) (K2)..... | 29 |
| 2.4. | Testing di manutenzione (K2) | 29 |
| 2.4.1. | Termini | 29 |
| 2.4.2. | Contesto | 30 |
| | Riferimenti..... | 30 |
| 3. | Tecniche statiche (K2) | 31 |

| | |
|--|-----------|
| Obiettivi di apprendimento per tecniche statiche | 31 |
| 3.1. Tecniche statiche ed il processo di testing (K2) | 31 |
| 3.1.1. Termini | 31 |
| 3.1.2. Contesto | 31 |
| 3.2. Processo di revisione (K2)..... | 32 |
| 3.2.1. Termini | 32 |
| 3.2.2. Contesto | 32 |
| 3.2.3. Fasi di una revisione formale (K1) | 32 |
| 3.2.4. Ruoli e responsabilità (K1)..... | 33 |
| 3.2.5. Tipi di revisioni (K2)..... | 33 |
| 3.2.6. Fattori di successo delle revisioni (K2) | 34 |
| 3.3. Analisi statica con strumenti (K2) | 35 |
| 3.3.1. Termini | 35 |
| 3.3.2. Contesto | 35 |
| Riferimenti..... | 36 |
| 4. Tecniche di test design (K3) | 37 |
| Obiettivi di apprendimento per le tecniche di test design (Progettazione dei Test) | 37 |
| 4.1. Il processo di sviluppo di test (K2)..... | 38 |
| 4.1.1. Termini | 38 |
| 4.1.2. Contesto | 38 |
| 4.2. Categorie delle tecniche di test design (K2)..... | 39 |
| 4.2.1. Termini | 39 |
| 4.2.2. Contesto | 39 |
| 4.3. Tecniche basate sulle specifiche o black-box (K3) | 40 |
| 4.3.1. Termini | 40 |
| 4.3.2. Partizionamento in classi di equivalenza (K3) | 40 |
| 4.3.3. Analisi ai valori limite (K3)..... | 40 |
| 4.3.4. Testing basato su tabelle delle decisioni (K3)..... | 40 |
| 4.3.5. Testing basato su diagrammi di transizioni tra stati (K3)..... | 41 |
| 4.3.6. Testing basato su ‘use cases’ (K2)..... | 41 |
| 4.4. Tecniche basate sulla struttura o white-box (K3)..... | 42 |
| 4.4.1. Termini | 42 |
| 4.4.2. Contesto | 42 |
| 4.4.3. Testing delle istruzioni e copertura delle istruzioni (K3) | 42 |

| | | |
|-----------|--|-----------|
| 4.4.4. | Testing delle decisioni e copertura delle decisioni (K3) | 42 |
| 4.4.5. | Altre tecniche basata sulla struttura (K1) | 43 |
| 4.5. | Tecniche basate sull'esperienza (K2) | 43 |
| 4.5.1. | Termini | 43 |
| 4.5.2. | Contesto | 43 |
| 4.6. | Scelta delle tecniche di testing (K2) | 43 |
| | Riferimenti | 44 |
| 5. | Gestione del testing (K3) | 45 |
| | Obiettivi di apprendimento per la gestione del testing | 45 |
| 5.1. | Organizzazione del testing (K2) | 46 |
| 5.1.1. | Termini | 46 |
| 5.1.2. | Organizzazione ed indipendenza del testing (K2) | 46 |
| 5.1.3. | Compiti del test leader e del tester (K1) | 46 |
| 5.2. | Stima e pianificazione del testing (K2) | 48 |
| 5.2.1. | Termini | 48 |
| 5.2.2. | Pianificazione del testing (K2) | 48 |
| 5.2.3. | Attività di pianificazione del testing (K2) | 48 |
| 5.2.4. | Criteri di ingresso (K2) | 49 |
| 5.2.5. | Criteri di uscita (K2) | 49 |
| 5.2.6. | Stima del testing (K2) | 49 |
| 5.2.7. | Strategie di testing, approcci di testing (K2) | 49 |
| 5.3. | Controllo e monitoraggio nell'avanzamento del testing (K2) | 50 |
| 5.3.1. | Termini | 50 |
| 5.3.2. | Monitoraggio nell'avanzamento del testing (K1) | 50 |
| 5.3.3. | Reportistica del testing (K2) | 51 |
| 5.3.4. | Controllo del testing (K2) | 51 |
| 5.4. | Gestione della configurazione (K2) | 51 |
| 5.4.1. | Termini | 51 |
| 5.4.2. | Contesto | 52 |
| 5.5. | Rischio e testing (K2) | 52 |
| 5.5.1. | Termini | 52 |
| 5.5.2. | Contesto | 52 |
| 5.5.3. | Rischi di progetto (K2) | 52 |
| 5.5.4. | Rischi di prodotto (K2) | 53 |

| | | |
|-----------|---|-----------|
| 5.6. | Gestione degli incidenti (K3)..... | 54 |
| | Riferimenti..... | 55 |
| 6. | Supporto di strumenti per il testing (K2) | 56 |
| | Obiettivi di apprendimento per il supporto di strumenti per il testing | 56 |
| 6.1. | Tipi di strumenti di testing (K2) | 56 |
| 6.1.1. | Termini | 56 |
| 6.1.2. | Capire il significato e lo scopo di uno strumento di aiuto al testing (K2)..... | 56 |
| 6.1.3. | Classificazione degli strumenti di testing (K2) | 57 |
| 6.1.4. | Strumenti di supporto per la gestione del testing e dei test (K1)..... | 57 |
| 6.1.5. | Strumenti di supporto per il testing statico (K1) | 58 |
| 6.1.6. | Strumenti di supporto per la specifica dei test (K1) | 58 |
| 6.1.7. | Strumenti di supporto per l'esecuzione ed il logging dei test (K1)..... | 59 |
| 6.1.8. | Strumenti di supporto per prestazioni e monitoraggio (K1)..... | 59 |
| 6.1.9. | Strumenti di supporto per specifiche esigenze di testing (K1)..... | 60 |
| 6.2. | Uso efficace degli strumenti: benefici e rischi potenziali (K2) | 60 |
| 6.2.1. | Potenziali benefici e rischi di strumenti di supporto per il testing (K2)..... | 60 |
| 6.2.2. | Considerazioni speciali per alcuni tipi di strumenti (K1)..... | 61 |
| 6.3. | Introduzione di uno strumento all'interno di un'organizzazione (K1)..... | 62 |
| 6.3.1. | Termini | 62 |
| 6.3.2. | Contesto | 62 |
| | Riferimenti..... | 63 |
| | Riferimenti..... | 64 |
| | Standard | 64 |
| | Libri | 64 |
| | Appendice A – Informazioni relative al syllabus | 66 |
| | Storia di questo documento | 66 |
| | Obiettivi del Certificato di qualifica 'Foundation' | 66 |
| | Obiettivi della qualifica internazionale..... | 66 |
| | Requisiti di ingresso per la qualifica ISTQB®..... | 67 |
| | Contesto e storia del Certificato 'Foundation' nel Software Testing | 67 |
| | Appendice B – Obiettivi di apprendimento / livello di conoscenza | 68 |
| | Livello 1: Ricordare (K1) | 68 |
| | Livello 2: Comprendere (K2) | 68 |
| | Livello 3: Applicare (K3) | 68 |

| | |
|--|-----------|
| Livello 4: Analizzare (K4)..... | 69 |
| Appendice C – Regole applicate a ISTQB®..... | 70 |
| Syllabus ‘Foundation’ | 70 |
| Regole generali | 70 |
| Contenuto attuale | 70 |
| Obiettivi di apprendimento | 70 |
| Struttura complessiva..... | 70 |
| Appendice D – Note ai fornitori della formazione..... | 72 |
| Appendice E - Release Notes Syllabus 2007 | 73 |
| Appendice F - Release Notes Syllabus 2010 | 74 |
| Appendice G - Release Notes Syllabus 2011..... | 75 |
| Indice Dei termini | 76 |

STORICO DELLE MODIFICHE DEL SILLABO ORIGINALE IN LINGUA INGLESE

| Versione | Data | Note |
|-------------|------------------|---|
| ISTQB® 2011 | 01-Aprile-2011 | Certified Tester Foundation Level Syllabus Maintenance Release – see Appendix E – Release Notes Syllabus 2011 |
| ISTQB® 2010 | 31-Marzo-2010 | Certified Tester Foundation Level Syllabus Maintenance Release – see Appendix E – Release Notes Syllabus 2010 |
| ISTQB® 2007 | 01-Maggio-2007 | Certified Tester Foundation Level Syllabus Maintenance Release – vedi Appendice E - Release Notes Syllabus 2007 |
| ISTQB® 2005 | 01-Luglio-2005 | Certified Tester Foundation Level Syllabus |
| ASQF V2.2 | Luglio-2003 | ASQF Syllabus Foundation Level Version 2.2 “Lehrplan „Grundlagen des Softwaretestens“ |
| ISEB V2.0 | 25-Febbraio-1999 | ISEB Software Testing Foundation Syllabus V2.0 25 Febbraio 1999 |

PREFAZIONE

Nel compilare questo glossario il gruppo di lavoro ha cercato le idee e i commenti di uno spettro di opinioni il più largo possibile nei settori e nelle organizzazioni in campo industriale e del commercio e nelle agenzie governative, con l'obiettivo di produrre uno standard internazionale per il testing che potesse essere accettato nel campo più ampio possibile. Sarà raro, se non impossibile, trovare un accordo unanime nella compilazione di un documento di questa natura. I contributi ricevuti per questo glossario

Definizioni

RINGRAZIAMENTI

International Software Testing Qualifications Board Working Group Foundation Level (Edition 2011): Thomas Müller (chair), Debra Friedenberg. Il core team ringrazia il gruppo di revisori (Dan Almog, Armin Beer, Rex Black, Julie Gardiner, Judy McKay, Tuula Pääkkönen, Eric Riou du Cosquier Hans Schaefer, Stephanie Ulrich, Erik van Veenendaal) e tutti i Board nazionali per i suggerimenti alla versione attuale del syllabus.

International Software Testing Qualifications Board Working Group Foundation Level (Edition 2010): Thomas Müller (chair), Rahul Verma, Martin Klöck and Armin Beer. Il core team ringrazia il gruppo di revisori (Rex Black, Mette Bruhn-Pederson, Debra Friedenberg, Klaus Olsen, Tuula Pääkkönen, Meile Posthuma, Hans Schaefer, Stephanie Ulrich, Pete Williams, Erik van Veenendaal) e tutti Board nazionali per i suggerimenti alla versione attuale del syllabus.

International Software Testing Qualifications Board Working Group Foundation Level (Edition 2007): Thomas Müller (chair), Dorothy Graham, Debra Friedenberg, ed Erik van Veenendaal. Il 'core team' ringrazia il 'review team' (Hans Schaefer, Stephanie Ulrich, Meile Posthuma, Anders Pettersson, and Wonil Kwon) e tutte le boards nazionali per tutti i suggerimenti alla versione attuale del syllabus.

International Software Testing Qualifications Board Working Group Foundation Level (Edition 2005): Thomas Müller (chair), Rex Black, Sigrid Eldh, Dorothy Graham, Klaus Olsen, Maaret Pyhäjärvi, Geoff Thompson and Erik van Veenendaal. Il 'core team' ringrazia il 'review team' e tutte le board nazionali per i suggerimenti all'attuale syllabus.

INTRODUZIONE A QUESTO SYLLABUS

SCOPO DEL PRESENTE DOCUMENTO

Questo syllabus costituisce la base per 'International Software Testing Qualification' al livello 'Foundation'.

'International Software Testing Qualifications Board (ISTQB®)' fornisce questo syllabus agli enti di certificazione nazionali per accreditare i fornitori della formazione e per derivare da esso delle domande d'esame nella lingua locale. I fornitori della formazione produrranno i corsi e determineranno i metodi di insegnamento più appropriati per l'accREDITAMENTO: il syllabus aiuterà i candidati nella loro preparazione agli esami.

Informazioni sulla storia e su informazioni relative al syllabus sono riportate in Appendice A.

IL 'CERTIFIED TESTER FOUNDATION LEVEL' NEL SOFTWARE TESTING

Il livello di qualificazione 'Foundation' è rivolto a chiunque sia coinvolto nel software testing. Questo include persone che ricoprono ruoli di testers, analisti di testing, ingegneri del testing, managers di testing, testers a livello di accettazione utente e sviluppatori software. Questo livello di qualifica denominato appunto 'Foundation' è anche appropriato per chiunque voglia avere una comprensione dei fondamenti del software testing, come managers di progetto, di sviluppo software, analisti e direttori di aziende IT. I possessori del 'Foundation Certificate' saranno in grado di conseguire un livello di qualifica superiore nelle competenze di software testing.

OBIETTIVI DELL'APPRENDIMENTO/LIVELLO DI CONOSCENZA

Per ogni sezione di questo syllabus sono identificati i livelli cognitivi come segue:

- K1: ricordare termini, riconoscere, richiamare definizioni;
- K2: comprendere;
- K3: applicare;
- K4: analizzare

Ulteriori dettagli ed esempi di obiettivi di apprendimento sono descritti nell'Appendice B.

Tutti i termini elencati sotto "Termini" immediatamente sotto il titolo del capitolo devono essere ricordati (K1), anche se non sono menzionati esplicitamente negli obiettivi di apprendimento.

L'ESAME

L'esame per il conseguimento del 'Foundation Certificate' sarà basato su questo syllabus. Le risposte alle domande degli esami possono richiedere l'utilizzo di materiale basato su più di una sezione di questo syllabus. Tutte le sezioni del syllabus sono oggetto d'esame.

Il formato dell'esame è a scelta multipla.

Gli esami possono essere presi come parte di un corso di formazione accreditato o in modo indipendente (ad esempio, presso un centro d'esame).

ACCREDITAMENTO

I fornitori della formazione il cui materiale del corso segue questo syllabus possono essere accreditati da un 'national board' riconosciuto da ISTQB®. Le linee guida per l'accREDITAMENTO devono essere ottenute dal 'board' o dall'ente che effettua l'accREDITAMENTO. Un corso accREDITATO viene riconosciuto come conforme a questo syllabus ed è consentito avere un esame ISTQB® come parte del corso. Ulteriori informazioni sono descritte nell'Appendice D.

LIVELLO DI DETTAGLIO

Il livello di dettaglio di questo syllabus consente una formazione ed un esame coerente a livello internazionale. Per raggiungere questo obiettivo, questo syllabus consiste di:

- Un'indicazione di obiettivi generali che descrivono i propositi del livello 'Foundation'.
- Un elenco di informazioni per l'insegnamento, che ne include una descrizione, e riferimenti a fonti aggiuntive se necessario.
- Un apprendimento di obiettivi per ogni area di conoscenza, che descriva il risultato cognitivo e l'atteggiamento mentale che si intendono ottenere.
- Una lista di termini che lo studente deve ricordare ed avere compreso.
- Una descrizione dei concetti chiave per l'insegnamento, che comprenda fonti come letteratura universalmente accettata o standards.

Il contenuto del syllabus non è una descrizione dell'intera area di conoscenza del software testing; riflette il livello di dettaglio che deve essere coperto dai corsi per il livello 'Foundation'.

COME È ORGANIZZATO QUESTO SYLLABUS

Il syllabus è articolato in sei capitoli principali. L'intestazione di livello più alto mostra i livelli degli obiettivi di apprendimento che sono coperti dal capitolo e specifica il tempo per capitolo. Per esempio:

2. Il testing durante il ciclo di vita del software (K2) 115 minuti

mostra che il capitolo 2 ha gli obiettivi di apprendimento di K1 (assunto sempre in quanto viene associato al capitolo un livello più alto) e K2 (ma non K3), ed è organizzato per occupare 115 minuti nell'insegnamento del materiale di quel capitolo. In ogni singolo capitolo ci sono un certo numero di sezioni. Per ogni sezione sono specificati anche gli obiettivi di apprendimento e la quantità di tempo richiesta. Le sottosezioni che non hanno un tempo assegnato sono incluse nel tempo di quella sezione.

| | |
|---------------------------------------|------------|
| 1. FONDAMENTI DEL TESTING (K2) | 155 minuti |
|---------------------------------------|------------|

OBIETTIVI DI APPRENDIMENTO PER I FONDAMENTI DEL TESTING

Gli obiettivi identificano le competenze che vengono acquisite seguendo il completamento di ogni singolo modulo.

1.1 Perché il testing è necessario? (K2)

- LO-1.1.1 Descrivere, con esempi, il modo nel quale un difetto nel software può causare danni a persone, all'ambiente o ad un'azienda. (K2)
- LO-1.1.2 Distinguere tra la causa scatenante di un difetto ed i suoi effetti. (K2)
- LO-1.1.3 Fornire delle ragioni del perché il testing è necessario attraverso esempi. (K2)
- LO-1.1.4 Descrivere perché il testing è parte del processo di garanzia di qualità e fornire esempi di come il testing contribuisce ad ottenere una più alta qualità. (K2)
- LO-1.1.5 Ricordare i termini errore, difetto, fault (guasto), failure (fallimento) e i corrispondenti termini mistake (sbaglio) e bug (baco), utilizzando esempi. (K2)

1.2 Che cos'è il testing? (K2)

- LO-1.2.1 Ricordare gli obiettivi comuni del testing. (K1)
- LO-1.2.2 Fornire esempi degli obiettivi del testing nelle diverse fasi del Ciclo di Vita del Software (K2)
- LO-1.2.3 Differenziare il testing dal debugging (K2)

1.3 Principi generali del testing (K2)

- LO-1.3.1 Spiegare i fondamentali principi del testing. (K2)

1.4 Fondamenti del processo di test (K1)

- LO-1.4.1 Ricordare le attività di test fondamentali, dalla pianificazione alla chiusura delle attività di test e i principali compiti di ogni singola attività di test. (K1)

1.5 La psicologia del testing (K2)

- LO-1.5.1 Richiamare i fattori psicologici che influenzano il successo del testing (K1)
- LO-1.5.2 Confrontare i punti di vista di un tester e di uno sviluppatore. (K2)

| |
|-----------|
| 20 minuti |
|-----------|

1.1. PERCHÉ IL TESTING È NECESSARIO (K2)

1.1.1. Termini

Bug (baco), difetto, errore, failure (fallimento), fault (guasto), sbaglio, qualità, rischio.

1.1.2. Contesto dei sistemi software (K1)

L'interazione con Sistemi Software occupa una parte sempre crescente della nostra vita, dalle applicazioni orientate al business (ad esempio, i sistemi bancari), ai prodotti di consumo (ad esempio, le automobili). Molte persone hanno avuto a che fare direttamente con esperienze nelle quali il software non si è comportato come esse si attendevano. Software che non opera correttamente può provocare molti problemi, compresi la perdita di denaro, di tempo, di reputazione negli affari e può anche causare infortuni o morte.

1.1.3. Cause dei difetti del software (K2)

Un essere umano può commettere un errore (sbaglio), il quale produce un difetto (fault, bug) nel codice del programma oppure in un documento. Se il codice difettoso viene eseguito, il sistema può fallire nel fare quello che deve fare (o può fare qualcosa che non deve), causando un esito negativo ('failure'). Difetti nel software, nei sistemi, nei documenti possono provocare 'failures', ma non tutti i difetti li possono provocare.

I difetti nascono perché l'essere umano non è infallibile e diventa maggiormente incline agli errori nelle condizioni in cui può essere costretto ad operare: mancanza di tempo, complessità del codice, complessità dell'infrastruttura, cambiamento delle tecnologie e/o a causa delle molte interazioni nel sistema.

Esiti negativi possono essere causati anche da condizioni ambientali: ad esempio radiazioni, magnetismo, campi elettrici e inquinamento possono causare guasti nel firmware o influenzare l'esecuzione del software andando a modificare condizioni hardware.

1.1.4. Ruolo del testing nello sviluppo, manutenzione e funzionamento del software (K2)

Un rigoroso testing dei sistemi e della documentazione di qualità può aiutare a ridurre il rischio di problemi che si possono verificare durante il funzionamento e a contribuire alla qualità del sistema software, se i difetti trovati vengono corretti, prima che il sistema venga rilasciato per il suo utilizzo operativo. Al testing del software può anche essere richiesto di rispettare requisiti legali, contrattuali, oppure standard industriali specifici del settore.

1.1.5. Testing e qualità (K2)

Con l'aiuto del testing, è possibile misurare la qualità del software in termini di difetti trovati, sia per requisiti funzionali che non funzionali e caratteristiche (ad esempio, affidabilità, usabilità, efficienza, manutenibilità e portabilità). Per maggiori informazioni sul testing non funzionale, si veda il Capitolo 2; per maggiori informazioni sulle caratteristiche del software si faccia riferimento a: "Software Engineering – Software Product Quality' (ISO 9126)".

Il testing può dare confidenza sulla qualità del software se trova alcuni difetti o nessuno. Un test progettato correttamente che viene eseguito con successo riduce il livello di rischio complessivo di un

sistema. Quando il testing trova difetti, la qualità del sistema software aumenta a seguito della correzione dei difetti.

Diverse lezioni devono essere imparate da precedenti progetti. Attraverso la comprensione delle cause di difetti trovate in altri progetti, i processi possono essere migliorati, e questo a sua volta deve prevenire il verificarsi di questi difetti e, di conseguenza, migliorare la qualità dei sistemi futuri. Questo è un aspetto di garanzia di qualità.

Il testing dovrebbe essere integrato come una delle attività di garanzia di qualità (quindi accanto allo sviluppo di standard, di formazione e analisi dei difetti).

1.1.6. Quando il testing può essere considerato sufficiente? (K2)

La decisione di quanto testing è necessario, deve tenere conto del livello di rischio, includendo rischi tecnici, di business, di progetto, e vincoli di progetto come tempo e budget (il rischio è discusso successivamente).

Il testing deve fornire sufficienti informazioni agli 'stakeholders' per prendere decisioni consapevoli sul rilascio del software o del sistema sotto testing, per la successiva fase di sviluppo o la consegna ai clienti.

| | |
|--|-----------|
| 1.2. CHE COS'È IL TESTING? (K2) | 30 minuti |
|--|-----------|

1.2.1. Termini

Debugging, requisito, review (revisione), test case (caso di test), testing, obiettivo del test.

1.2.2. Contesto

Una percezione molto diffusa sul testing è che esso consista solo nell'eseguire test e quindi nell'esecuzione di software. Questa è solo una parte del testing; vi sono diverse altre attività di testing.

Le attività di test sono presenti sia prima che dopo l'esecuzione di test: attività come la pianificazione e il controllo, la scelta delle condizioni di test, la progettazione dei 'test cases' e del controllo dei risultati, la valutazione dei criteri di uscita, il report sul processo di testing e sul sistema sotto test e la finalizzazione o le attività di chiusura dopo che una fase di test è stata completata. Il testing include anche la revisione dei documenti (compreso il codice sorgente) e la conduzione di attività di analisi statica.

Sia il testing dinamico sia il testing statico possono essere usati con l'intento di perseguire obiettivi simili e forniranno informazioni con lo scopo di migliorare sia il sistema che deve essere testato, sia i processi di sviluppo e del testing stesso.

Ci possono essere differenti obiettivi dei test:

- trovare difetti;
- acquisire confidenza circa il livello di qualità e ottenere opportune informazioni;
- prevenire i difetti.

L'impegnativo processo di dedicarsi alla progettazione dei test nelle fasi iniziali del ciclo di vita (verificando le fondamenta di tali test attraverso la loro progettazione) può aiutare a prevenire difetti prima che essi vengano introdotti nel codice. Anche revisioni di documenti (ad esempio dei requisiti) aiutano a prevenire difetti che compaiono nel codice.

Differenti punti di vista del testing tengono conto di diversi obiettivi. Per esempio, in diverse fasi di testing (ad esempio, testing di componente, testing di integrazione e testing di sistema), il principale obiettivo può essere quello di causare tanti più esiti negativi (failure) possibili, così che i difetti del software possano essere identificati e quindi corretti. Nel testing di accettazione, l'obiettivo principale può essere di confermare che il sistema si comporti come atteso, per acquisire confidenza sul fatto che esso soddisfi i propri requisiti. In alcuni casi il principale obiettivo del testing può essere di valutare la qualità del software (senza l'intenzione di correggere difetti), per fornire informazioni alle parti interessate del rischio di rilasciare il sistema in un certo momento. Per testing di manutenzione spesso si intende quel testing mirato a verificare che nessun nuovo difetto sia stato introdotto durante lo sviluppo delle correzioni. Durante il testing operativo, il principale obiettivo può essere di valutare caratteristiche del sistema come affidabilità o disponibilità.

Il debugging e il testing sono attività differenti. Il testing può mettere in evidenza degli esiti negativi (failures) causate da difetti. Il debugging è l'attività di sviluppo che identifica la causa di un difetto, corregge il codice e controlla che il difetto sia stato risolto in modo corretto. Il successivo testing confermativo eseguito da un tester assicura che la correzione effettivamente risolva l'esito negativo. La responsabilità per tali attività è molto differente e conseguentemente i testers fanno testing, gli sviluppatori fanno debug.

Il processo di testing e le relative attività sono spiegate nella sezione 1.4.

| | |
|--|-----------|
| 1.3. PRINCIPI GENERALI DEL TESTING (K2) | 35 minuti |
|--|-----------|

1.3.1. Termini

Testing esaustivo

1.3.2. Principi

Un certo numero di principi sul testing sono stati suggeriti negli ultimi 40 anni e offrono linee guida generali per tutto il testing.

Principio 1 – Il testing mostra la presenza di difetti

Il testing può mostrare la presenza di difetti, ma non può provarne l'assenza. Il testing riduce la probabilità di difetti non scoperti ancora presenti nel software, ma anche se nessun difetto viene trovato, questa non è una prova di correttezza, ovvero di assenza di difetti.

Principio 2 – Il testing esaustivo è impossibile

Il testing esaustivo (ovvero di tutte le combinazioni di dati in ingresso e di precondizioni) non è fattibile, tranne che per casi estremamente semplici. Quindi invece di concentrarsi sul testing esaustivo bisognerebbe focalizzare gli sforzi di testing sulle analisi dei rischi e delle priorità.

Principio 3 – Anticipare il testing il prima possibile

Per individuare i difetti anticipatamente, le attività di testing devono partire il prima possibile nel ciclo di vita del software o del sistema e devono essere focalizzate con obiettivi ben definiti.

Principio 4 – “Clustering” dei difetti

Lo sforzo del testing deve essere indirizzato proporzionalmente alla densità dei difetti (attesa o misurata a posteriori) di ogni modulo. Un piccolo numero di moduli contiene normalmente la maggior parte dei difetti scoperti durante il testing di pre-rilascio, o sono responsabili di molti esiti negativi operativi (ovvero i difetti tendono a formare degli agglomerati, detti appunto “cluster”, concentrandosi, soprattutto nelle prime fasi di sviluppo, in alcuni moduli).

Principio 5 – “Paradosso pesticida”

Se gli stessi test sono ripetuti più e più volte, presumibilmente lo stesso insieme di test cases non troverà nessun nuovo difetto. Per superare questo “paradosso pesticida”, i casi di test necessitano di essere riveduti e corretti, e nuovi test necessitano di essere riscritti per stimolare differenti parti del software e del sistema e per poter potenzialmente trovare più difetti.

Principio 6 – Il testing è dipendente dal contesto

Il testing viene eseguito diversamente in contesti differenti. Per esempio, un software safety-critical viene testato in modo differente da un sito di commercio elettronico.

Principio 7 – Il luogo comune dell’ assenza di errori

Trovare e correggere difetti non è sufficiente se il sistema sviluppato è inutilizzabile e non soddisfa le necessità e le aspettative dell’utente. L’assenza di errori è una falsa aspettativa. Un tester può solo dire, al limite, che non può trovare altri errori in un software, ma non che quel software non contenga errori. Questo sempre in base al Principio 1.

| | |
|--|-----------|
| 1.4. FONDAMENTI DEL PROCESSO DI TEST (K1) | 35 minuti |
|--|-----------|

1.4.1. Termini

Testing confermativo, retesting, criterio di uscita, segnalazione, testing di regressione, basi del test, test condition (condizione di test), copertura del test, dati di test esecuzione del test, test log, test plan (piano di test), test procedure (procedura di test), politica di test, strategia di test, test suite (insieme di test), test summary report (report riassuntivo del test), testware.

1.4.2. Contesto

La parte più visibile del testing è quella relativa all’esecuzione dei test. Tuttavia per essere efficaci ed efficienti, i piani di test devono includere anche il tempo che dovrà essere speso nella pianificazione dei test, nella progettazione dei test cases, nella preparazione per l’esecuzione e nella valutazione dei risultati.

I fondamenti del processo di test consistono delle seguenti attività principali:

- pianificazione e controllo;
- analisi e progettazione;
- implementazione ed esecuzione;
- valutazione dei criteri di uscita e report dei risultati;
- attività di conclusione dei test.

Benché queste attività siano logicamente sequenziali, nel processo descritto esse possono sovrapporsi o svolgersi contemporaneamente.

1.4.3. Pianificazione e controllo del test (K1)

La pianificazione del test comprende la definizione degli obiettivi del testing e le specifiche delle attività di test che consentano di raggiungere i suoi obiettivi e quindi soddisfare la sua “mission”.

Il controllo del test è la fase continuativa di confronto tra la pianificazione e lo stato di avanzamento, che consiste nel documentare lo stato attuale evidenziando deviazioni rispetto a quanto pianificato. Questa attività include la messa in atto delle contromisure atte a fare rispettare la “mission” e gli obiettivi del progetto: il controllo implica che il testing debba essere monitorato lungo tutto l’arco del progetto. La pianificazione del test tiene a sua volta conto dei riscontri delle attività di monitoraggio e controllo.

I compiti da svolgere nelle attività di pianificazione e controllo sono definiti nel Capitolo 5 di questo syllabus.

1.4.4. Analisi e progettazione del test (K1)

L’analisi e la progettazione del test è l’attività nella quale gli obiettivi vengono trasformati in opportune ‘test conditions’ e ‘test cases’.

L’analisi e la progettazione consiste dei seguenti compiti principali:

- Revisione delle basi del test (come requisiti, architettura, progettazione, interfacce).
- Valutazione della testabilità delle basi del test e degli oggetti del test.
- Identificazione e assegnazione delle priorità delle ‘test conditions’ basata sull’analisi delle unità di test, delle specifiche, del comportamento e della struttura del software.
- Progettazione e assegnazione delle priorità dei ‘test cases’.
- Identificazione dei dati di test necessari per supportare le ‘test conditions’ e i ‘test cases’.
- Progettazione dell’ambiente di test e identificazione di ogni elemento dell’infrastruttura richiesta e degli strumenti di supporto all’attività.

1.4.5. Implementazione ed esecuzione del test (K1)

L’implementazione e l’esecuzione sono le attività nelle quali le ‘test procedures’ (e gli eventuali scripts) vengono specificate combinando i ‘test cases’ in un particolare ordine ed includendo ogni altra informazione necessaria per l’esecuzione: l’ambiente di test viene allestito e configurato ed i test cases vengono eseguiti.

L’implementazione e l’esecuzione consistono dei seguenti compiti principali:

- Sviluppo, implementazione e definizione delle priorità dei test cases.
- Sviluppo e definizione delle priorità delle ‘test procedures’, creando i dati di test, preparando (eventualmente) i necessari corredi (test harness) e scrivendo degli scripts di test automatizzati.
- Creazione di ‘test suites’ dalle ‘test procedures’ per una efficiente esecuzione.
- Verifica che l’ambiente di test sia stato preparato e configurato correttamente.
- Esecuzione delle ‘test procedures’ sia manualmente sia usando degli strumenti di esecuzione, in accordo con la sequenza pianificata.
- Salvataggio in opportuni log, dell’esito dell’esecuzione dei ‘test cases’ e registrazione degli identificativi e delle versioni dei software sotto test, degli strumenti di test e del ‘testware’.
- Confronto dei risultati effettivi con i risultati attesi.
- Indicazione delle discrepanze sotto forma di incidenti e loro analisi finalizzata a stabilire la loro causa (ad esempio, un difetto nel codice, nei dati di test, nel documento di test, od un errore nel modo in cui il test è stato eseguito ecc).
- Ripetizione delle attività di test come risultato delle azioni adottate per ogni discrepanza. Per esempio, ri-esecuzione di un test precedentemente fallito con lo scopo di convalidare una

correzione (testing confermativo), esecuzione di un test valido e/o (ri-)esecuzione di test con lo scopo di assicurare che non siano stati introdotti difetti in parti di software non modificate o che la correzione di difetti non abbia scoperto altri difetti (testing di regressione).

1.4.6. Valutazione dei criteri di uscita e report dei risultati (K1)

La valutazione dei criteri di uscita è l'attività nella quale l'esecuzione di test è valutata rispetto agli obiettivi definiti. Questa attività deve essere svolta per ogni livello di test.

La valutazione dei criteri di uscita consiste dei seguenti compiti principali:

- Controllo dei test logs rispetto ai criteri di uscita specificati nella fase di pianificazione.
- Valutazione della necessità dell'introduzione di nuovi test o della modifica dei criteri di uscita inizialmente scelti.
- Preparazione di un report riassuntivo dei test per le parti interessate (stakeholders).

1.4.7. Attività di chiusura del test (K1)

Le attività di conclusione dei test raccolgono i dati prodotti dalle attività di test completate per consolidarne l'esperienza, il testware, i risultati e i dati quantitativi. Le attività di chiusura del test sono svolte: quando un sistema software viene rilasciato, un progetto di test viene completato (o cancellato), una milestone è stata raggiunta od un rilascio di manutenzione è stata completato.

Le attività di conclusione dei test consistono dei seguenti compiti principali:

- Controllo di quali consegne pianificate sono state effettuate.
- Chiusura dei report degli incidenti o apertura delle segnalazioni di modifica per ogni incidente che rimane aperto.
- Documentazione di accettazione del sistema.
- Finalizzazione ed archiviazione del 'testware', dell'ambiente di test e dell'infrastruttura di test per un successivo riuso.
- Consegna del 'testware' alla organizzazione di manutenzione.
- Analisi delle esperienze apprese da utilizzare per modifiche a futuri progetti.
- Utilizzo delle informazioni raccolte per il miglioramento della maturità dei processi di testing.

| | |
|--|-----------|
| 1.5. LA PSICOLOGIA DEL TESTING (K2) | 35 minuti |
|--|-----------|

1.5.1. Termini

Error guessing (congetture sugli errori), indipendenza.

1.5.2. Contesto

L'approccio mentale da usare durante il testing e le fasi di revisione è differente da quello usato durante lo sviluppo del software. Col corretto atteggiamento mentale anche gli sviluppatori sono in grado di testare il loro proprio codice, ma l'assegnazione di questa attività ad un tester viene tipicamente fatta per aumentare lo sforzo di focalizzazione e fornire ulteriori benefici, come una visione indipendente (e quindi senza inconsci condizionamenti) da parte di risorse professionali e dotate di una specifica formazione nell'ambito del testing. Il testing indipendente può essere effettuato ad ogni livello di testing .

Un certo grado di indipendenza (che eviti il condizionamento dell'autore) è spesso più efficace nella rilevazioni di difetti e di esiti negativi (failures). L'indipendenza non è, comunque, un sostituto della

dimestichezza col software e anche gli sviluppatori possono trovare in modo efficiente molti difetti nel loro codice. Possono essere definiti diversi livelli di indipendenza:

- Test progettati dalla persona (o dalle persone) che hanno scritto il software sotto test (basso livello di indipendenza).
- Test progettati da una persona (o da delle persone) diversa da chi ha scritto il software (ad esempio, da quella parte del team di sviluppo che non ha scritto quella parte di software).
- Test progettati da una o più persone di un gruppo organizzativo differente (ad esempio, da un team di test indipendente) o da specialisti di testing.
- Test progettati da una o più persone di un ente o di un'azienda diversa (ad esempio, in outsourcing o certificazione da parte di un ente esterno).

Le persone e i progetti sono guidati da obiettivi. Le persone tendono ad allineare i loro piani con gli obiettivi determinati dal management o da altri 'stakeholders', per esempio, per trovare difetti o per confermare che il software funzioni. Per questo è importante definire chiaramente gli obiettivi del testing.

L'identificazione di esiti negativi (failures) durante il testing può essere percepita come una critica nei confronti del prodotto e del suo autore. Il testing viene, per questo motivo, visto spesso come una attività distruttiva, benché sia molto costruttiva nella gestione dei rischi del prodotto. La ricerca di esiti negativi (failures) in un sistema, richiede curiosità, pessimismo costruttivo, occhio critico, attenzione ai dettagli, una buona comunicazione col team di sviluppo ed esperienza sulla quale basare le congetture sugli errori ('error guessing').

Se errori, difetti o esiti negativi (failures) vengono comunicati in un modo costruttivo, si possono evitare cattivi rapporti tra testers e analisti, progettisti e sviluppatori. Ciò è vero sia per le revisioni che per il testing.

Il tester e il test leader hanno bisogno di buone capacità interpersonali per comunicare in modo costruttivo le informazioni riguardanti i difetti, i progressi ed i rischi. Le informazioni sui difetti possono aiutare l'autore del software o di documenti a migliorare le proprie capacità. Difetti trovati e corretti durante il testing faranno risparmiare tempo e denaro in futuro e ridurre i rischi.

Problemi di comunicazione possono particolarmente verificarsi se i testers sono visti solo come messaggeri di notizie indesiderate sui difetti. Comunque, ci sono diversi modi per migliorare la comunicazione e le relazioni tra i testers e le altre persone coinvolte:

- Iniziare con un atteggiamento di collaborazione piuttosto che di disputa, ricordando ad ognuno che l'obiettivo comune da raggiungere è sempre una migliore qualità dei sistemi.
- Comunicare quanto scoperto sul prodotto in un modo neutrale, focalizzandosi sui fatti e sulle evidenze senza nessun atteggiamento di critica nei confronti della persona che ha sviluppato quel prodotto; tipicamente, scrivere il report sull'eventuale problema riscontrato in modo oggettivo e basato sui fatti.
- Provare a capire come le persone percepiscono l'atteggiamento del tester e perché esse reagiscono in un certo modo.
- Essere sicuri che l'altra persona abbia compreso quello che il tester gli ha detto e viceversa, per evitare malintesi e incomprensioni.

| | |
|-------------------------------|-----------|
| 1.6. CODICE ETICO (K2) | 10 minuti |
|-------------------------------|-----------|

La partecipazione ai test del software consente alle persone di apprendere informazioni riservate e privilegiate. Un codice etico è necessario, tra le altre ragioni, per garantire che tali informazioni non siano utilizzate ad uso improprio.

® ISTQB®, riconoscendo il codice etico per gli ingegneri di ACM e IEEE, ha definito il seguente codice etico:

- * PUBBLICO – i software tester certificati devono agire coerentemente con l'interesse pubblico
- * CLIENTE E DATORE DI LAVORO – i software tester certificati devono agire nel miglior interesse del loro cliente e del loro datore di lavoro, in coerenza con l'interesse pubblico
- * PRODOTTO - i software tester certificati devono assicurare che quanto rilasciato (per i prodotti e/o sistemi soggetti a test) soddisfi i più elevati standard professionali
- * GIUDIZIO - i software tester certificati devono esprimere i propri giudizi professionali in modo integro e indipendente
- * GESTIONE - I software manager e leader devono sottoscrivere e promuovere un approccio etico alla gestione del testing del software
- * PROFESSIONE – I software tester certificati devono far prevalere l'integrità e la reputazione della professione, coerentemente con l'interesse pubblico
- * COLLEGHI – I software tester certificati devono essere leali e disponibili ad aiutare i propri colleghi e promuovere la cooperazione con gli sviluppatori di software
- * FORMAZIONE – I software tester certificati devono partecipare a una formazione continua per quanto riguarda la pratica della loro professione e devono mantenere un approccio etico alla pratica di tale professione

1.7. RIFERIMENTI

- 1.1.5 Black, 2001, Kaner, 2002
- 1.2 Beizer, 1990, Black, 2001, Myers, 1979
- 1.3 Beizer, 1990, Hetzel, 1988, Myers, 1979
- 1.4 Hetzel, 1988
- 1.4.5 Black, 2001, Craig, 2002
- 1.5 Black, 2001, Hetzel, 1988

| | |
|--|--------------------------|
| <p>2. IL TESTING DURANTE IL CICLO DI VITA DEL SOFTWARE (K2)</p> | <p>115 minuti</p> |
|--|--------------------------|

OBIETTIVI DI APPRENDIMENTO PER IL TESTING DURANTE IL CICLO DI VITA DEL SOFTWARE

Gli obiettivi identificano quello che si dovrà essere in grado di fare al termine di ogni singolo modulo.

2.1 Modelli di sviluppo del software (K2)

- LO-2.1.1 Comprendere le relazioni tra lo sviluppo, le attività di test e quanto prodotto nel ciclo di vita dello sviluppo software, e fornire esempi basati su contesto e sulle caratteristiche di progetti e prodotti. (K2)
- LO-2.1.2 Riconoscere il fatto che i modelli di sviluppo del software devono essere adattati al contesto del progetto e alle caratteristiche del prodotto. (K1)
- LO-2.1.3 Ricordare i ruoli dei differenti livelli di testing e le caratteristiche di un buon testing in ogni modello di ciclo di vita. (K1)

2.2 Livelli di test (K2)

- LO-2.2.1 Comparare i differenti livelli di testing: obiettivi principali, tipici oggetti del testing, tipici livelli del testing (ad esempio, funzionale o strutturale) e prodotti correlati, persone che effettuano il testing, tipi di difetti e di esiti negativi (failures) da identificare. (K2)

2.3 Tipi di test (K2)

- LO-2.3.1 Comparare quattro tipi di software test (funzionale, non-funzionale, strutturale e legato alle modifiche) attraverso esempi. (K2)
- LO-2.3.2 Riconoscere che test funzionali e strutturali sono possibili ad ogni livello di test. (K1)
- LO-2.3.3 Identificare e descrivere tipi di test non-funzionali basati su requisiti non-funzionali. (K2)
- LO-2.3.4 Identificare e descrivere tipi di test basati sull'analisi dell'architettura e struttura di un sistema software. (K2)
- LO-2.3.5 Descrivere lo scopo del testing confermativo e del testing di regressione. (K2)

2.4 Testing di manutenzione (K2)

- LO-2.4.1 Confrontare il testing di manutenzione (testing su un sistema esistente) al testing di una nuova applicazione, rispetto ai tipi di test, agli inneschi del testing ed alla quantità di testing. (K2)
- LO-2.4.2 Identificare i ruoli del testing di manutenzione (modifica, migrazione e ritiro). (K1)
- LO-2.4.3 Descrivere il ruolo del testing di regressione e l'analisi di impatto .nella manutenzione. (K2)

| | |
|--|-------------------------|
| <p>2.1. MODELLI DI SVILUPPO DEL SOFTWARE (K2)</p> | <p><i>20 minuti</i></p> |
|--|-------------------------|

2.1.1. Termini

Commercial off-the-shelf (COTS), modello di sviluppo iterativo-incrementale, validazione, verifica, modello a V.

2.1.2. Contesto

Il testing non esiste in modo isolato; le attività di testing sono legate a quelle dello sviluppo software. Diversi modelli di cicli di vita dello sviluppo necessitano di differenti approcci al testing.

2.1.3. Modello a V (modello di sviluppo sequenziale) (K2)

Benché esistano diverse varianti del modello a V, un tipo comune di modello a V usa quattro livelli di test, corrispondenti ai quattro livelli di sviluppo.

I quattro livelli usati in questo syllabus sono:

- Testing di componente (o di unità) ;
- Testing di integrazione;
- Testing di sistema;
- Testing di accettazione .

In pratica, un modello a V può avere un numero differente, sia maggiore che minore, di livelli di sviluppo e testing, a seconda del progetto e del prodotto software. Per esempio, ci possono essere il testing di integrazione dei componenti, dopo il testing dei componenti, e il testing di integrazione dei sistemi dopo il testing di sistema.

I prodotti del software (come scenari di business o uses cases, specifiche dei requisiti , documenti di design e codice), ovvero tutti quegli elementi (non solo codice) prodotti durante la fase di sviluppo del software sono spesso le basi del testing, gli inputs in uno o più livelli di test.

Riferimenti per generici cicli di vita includono il Capability Maturity Model Integration (CMMI) o il ‘Software life cycle processes’ (IEEE/IEC 12207). La verifica e la validazione (e le prime fasi di test design) possono essere svolte durante lo sviluppo dei prodotti del ciclo di vita del software.

2.1.4. Modelli di sviluppo iterativo-incrementale (K2)

Lo sviluppo iterativo-incrementale è il processo di definizione dei requisiti, progettazione, costruzione e testing di un sistema, svolto come una serie di cicli di sviluppo più corti. Esempi di questo modello sono: la prototipazione, il ‘Rapid Application Development’ (RAD) , il ‘Rational Unified Process’ (RUP) e i modelli cosiddetti di sviluppo “Agile” .

Il sistema risultante da un tale approccio può essere testato a diversi livelli durante ogni iterazione . Un incremento, aggiunto agli altri sviluppati precedentemente, forma un sistema parziale in crescita, che deve essere anch'esso testato.

In questo contesto il regression testing diventa di importanza crescente durante tutte le iterazioni successive alla prima. La verifica e la validazione possono essere condotte su ogni singolo incremento.

2.1.5. Il testing in un modello di ciclo di vita (K2)

In ogni modello di ciclo di vita, ci sono diverse caratteristiche per condurre un buon testing:

- Per ogni attività di sviluppo c’è una corrispondente attività di testing.
- Ogni singolo livello di test ha degli obiettivi di test specifici di quel livello.
- L’analisi e la progettazione dei test per un dato livello di test deve iniziare durante la corrispondente attività di sviluppo.
- I testers devono essere coinvolti nella revisione dei documenti non appena le loro versioni in bozza siano disponibili nel ciclo di vita di sviluppo.

I livelli di test possono essere combinati o riorganizzati a seconda della natura del progetto o della architettura di sistema. Per esempio, per l’integrazione di un prodotto software COTS (commercial off-the-shelf), l’acquirente può eseguire del testing di integrazione a livello di sistema (ad esempio, integrazione all’infrastruttura e ad altri sistemi, o ‘deployment’ del sistema) e del testing di accettazione (funzionale e /o non funzionale e testing utente e/o operativo).

40 minuti

2.2. LIVELLI DI TEST (K2)

2.2.1. Termini

Alpha testing, beta testing, testing di componente (noto anche come testing di unità, di modulo, o di programma), driver, field testing, requisiti funzionali, integrazione, testing di integrazione, requisito non funzionale, testing di robustezza, stub, testing di sistema, livello di test, sviluppo test-driven (sviluppo guidato dal test), ambiente di test, testing di accettazione utente

2.2.2. Contesto

Per ogni livello di test, i seguenti elementi possono essere identificati: i loro obiettivi generici, gli inputs di riferimento per la derivazione dei test cases (quindi le basi di test), l'oggetto del test (quindi quello che deve essere testato), i tipici difetti e gli esiti negativi (failures) più critici da trovare, i requisiti di test harness e il supporto degli strumenti, nonché gli specifici approcci e responsabilità.

Il testing dei dati di configurazione di sistema deve essere preso in considerazione nella pianificazione del testing, se tali dati fanno parte del sistema.

2.2.3. Testing di componente (K2)

Basi di test:

- Requisiti dei componenti
- Disegno dettagliato
- Codice

Tipici oggetti di test:

- Componenti
- Programmi
- Programmi di conversione dati / migrazione
- Banche Dati

Il testing di componente cerca i difetti e verifica il funzionamento di software (ad esempio, moduli, programmi, oggetti, classi, ecc) che sono testabili separatamente. Esso può essere condotto in isolamento dal resto del sistema, a seconda del contesto del ciclo di vita dello sviluppo e del sistema. In questa attività possono essere usati stubs, drivers e simulatori.

Il testing di componente può includere il testing di funzionalità e di specifiche caratteristiche non funzionali, come il comportamento di alcune risorse (ad esempio, 'memory leaks') o testing di robustezza, nonché il testing strutturale (ad esempio, copertura delle condizioni). I 'test cases' sono derivati da input come una specifica del componente, il design del software o il modello dei dati.

Tipicamente, il testing di componente si realizza con accesso al codice sotto testing e con il supporto dell'ambiente di sviluppo, come un framework di test di unità o uno strumento di debugging e, in pratica, coinvolge normalmente il programmatore che ha scritto il codice. I difetti sono tipicamente corretti non appena sono trovati, senza effettuare delle segnalazioni formali.

Un approccio al testing di componente è di preparare e automatizzare i test cases prima della codifica: questo viene chiamato approccio 'test-first' o sviluppo test-driven). Questo approccio è altamente iterativo ed è basato su cicli di sviluppo di test cases, quindi di compilazione e integrazione di piccoli pezzi di codice, e di esecuzione dei test di componente fino a quando risultino corretti.

2.2.4. Testing di integrazione (K2)

Basi di test:

- Disegno software e di sistema
- Architettura
- Workflows
- Use cases

Tipici oggetti di test:

- Sottosistemi
- Banche Dati dei sottosistemi
- Infrastruttura
- Interfacce
- Configurazioni di sistema e configurazioni dei dati

Il testing di integrazione testa le interfacce tra componenti, le interazioni con differenti parti di un sistema, come il sistema operativo, il file system, l'hardware o le interfacce tra sistemi.

Ci può essere più di un livello di testing di integrazione ed esso può essere applicato su oggetti di test di varie dimensioni. Per esempio:

1. Il testing di integrazione dei componenti si occupa del testing delle interazioni tra componenti software ed è fatto dopo il testing di componente;
2. Il testing di integrazione dei sistemi invece si concentra sulle interazioni tra differenti sistemi e può essere svolto dopo il testing di sistema. In questo caso, l'organizzazione di sviluppo può controllare solo un lato dell'interfaccia e quindi le modifiche possono essere destabilizzanti. Processi di business implementati come 'workflows' possono coinvolgere una serie di sistemi. Problemi legati a piattaforme differenti possono essere molto significativi.

Tanto maggiore è la portata dell'integrazione, quanto più difficile diventa isolare i difetti associandoli ad un componente specifico o addirittura ad un sistema, il che può condurre ad un maggiore rischio.

Le strategie di integrazione sistematica, possono essere basate sull'architettura del sistema (ad esempio, top-down o bottom-up), sulle funzionalità, sulle sequenze di elaborazione delle transazioni o su altri aspetti specifici del sistema o dei componenti. Con l'obiettivo di ridurre il rischio di una tardiva scoperta di difetti, è raccomandata un'integrazione incrementale invece che "big bang".

Il testing di specifiche caratteristiche non funzionali (ad esempio, le prestazioni) possono essere incluse nel testing di integrazione.

Ad ogni stadio di integrazione, i testers devono focalizzare il proprio lavoro esclusivamente sull'integrazione stessa. Per esempio, se essi stanno integrando il modulo A con il modulo B, essi sono interessati solo al testing della comunicazione tra moduli e non delle funzionalità dei singoli moduli. Possono essere utilizzati approcci di tipo sia funzionale sia strutturale.

Idealmente, i testers devono considerare l'architettura e l'impatto che essa comporta sulla pianificazione dell'integrazione. Se i test di integrazione fossero pianificati prima che i componenti o sistemi siano completi, tali test potrebbero essere pronti, completati e messi a disposizione nell'ordine richiesto per ottenere la massima efficacia dal testing di integrazione.

2.2.5. Testing di sistema (K2)

Basi di test:

- Specifiche dei requisiti di software e di sistema

- Use cases
- Specifiche funzionali
- Rapporti analisi rischi

Tipici oggetti di test:

- Manuali operativi di sistema e utente
- Configurazioni di sistema e configurazioni dati

Il testing di sistema verifica il comportamento di un intero sistema/prodotto. L'obiettivo del testing deve essere chiaramente indirizzato nei Piani di Test per quello specifico livello di test.

Nel testing di sistema, l'ambiente di test deve corrispondere il più possibile a quello finale in modo da minimizzare il rischio di esiti negativi (failures) specifici di tale ambiente di test che non siano stati rilevati in precedenza.

Il testing di sistema può includere test basati sui rischi e/o specifiche dei requisiti, processi di business, 'use cases', oltre a descrizioni di alto livello del comportamento del sistema, interazioni col sistema operativo e con le risorse di sistema.

Il testing di sistema deve investigare sia requisiti funzionali che non funzionali del sistema.

I requisiti possono esistere sotto forma testuale e/o di modelli. Inoltre i testers hanno in generale a che fare con requisiti non documentati o incompleti. Il testing di sistema di requisiti funzionali inizia utilizzando le tecniche più appropriate fra quelle basate sulle specifiche (black-box) per gli aspetti del sistema che devono essere testati. Per esempio, si può creare una tabella delle decisioni come combinazione degli effetti descritti in opportune regole. Tecniche basate sulla struttura (white-box) possono quindi essere usate per valutare l'accuratezza del testing rispetto ad elementi strutturali, come struttura dei menu o navigazione di pagine web (si veda il Capitolo 4).

Spesso è un team di test indipendente che svolge l'attività di testing di sistema.

2.2.6. Testing di accettazione (K2)

Basi di test:

- Requisiti utenti
- Requisiti di sistema
- Use cases
- Processi di business
- Rapporti analisi rischi

Tipici oggetti di test:

- Processi di business in sistemi integrati
- Processi operativi e manutentivi
- Procedure utente
- Moduli
- Rapporti
- Configurazioni dati

Il testing di accettazione è spesso di responsabilità dei clienti o degli utenti di un sistema; altri stakeholders possono ovviamente esserne coinvolti.

L'obiettivo nel testing di accettazione è di valutare il grado di confidenza sul funzionamento del sistema, di parti del sistema o di specifiche caratteristiche non funzionali del sistema. La ricerca dei difetti non è il

focus principale nel testing di accettazione. Il testing di accettazione può valutare il grado di maturità del sistema per il rilascio e l'utilizzo, nonostante esso non sia necessariamente il livello finale di testing. Per esempio, un'integrazione del sistema su larga-scala può venire dopo la fase di testing di accettazione del sistema.

Il testing di accettazione può essere svolto in tempi diversi durante il ciclo di vita, per esempio:

- Un prodotto software COTS può essere sottoposto a testing di accettazione quando viene installato o integrato.
- Il testing di accettazione dell'usabilità di un componente può essere fatto durante il testing di componente.
- Il testing di accettazione di un nuovo miglioramento funzionale può venire prima del testing di sistema.

Tipiche forme di testing di accettazione includono le seguenti:

Testing di accettazione utente

Tipicamente verifica l'idoneità di utilizzo del sistema da parte degli utenti finali.

Testing operativo (di accettazione)

L'accettazione del sistema da parte dell'amministratore di sistema, che include:

- il testing di backup/restore;
- il ripristino di situazioni di disastro;
- la gestione degli utenti;
- le attività di manutenzione;
- le attività di carico e migrazione dati
- i controlli periodici di vulnerabilità alla sicurezza.

Testing di accettazione contrattuali e normativi

Il testing di accettazione contrattuale viene eseguito nel rispetto dei criteri contrattuali di accettazione per la produzione di software sviluppato su misura per clienti. Tali criteri devono essere ben definiti quando il contratto viene sottoscritto e accettato da entrambe le parti. Il testing di accettazione normativo viene svolto nel rispetto di regole da rispettare, siano esse di tipo governativo, legale o di sicurezza.

Alpha e beta (o field) testing

Sviluppatori di software COTS vogliono spesso ottenere un riscontro da parte di clienti potenziali od esistenti nel loro mercato prima che il loro prodotto software venga effettivamente commercializzato. Il cosiddetto alpha testing viene svolto presso il sito dell'organizzazione che ha sviluppato il software. Invece il cosiddetto beta testing (o field testing) viene effettuato da altre persone nelle loro proprie sedi. Entrambi sono svolti da clienti potenziali e non da sviluppatori del prodotto.

Le organizzazioni possono usare anche altri termini, come testing di accettazione di 'factory' o testing di accettazione 'on site' per sistemi il cui testing è stato effettuato prima o dopo l'installazione presso i siti dei clienti.

| | |
|-------------------------------|------------------|
| 2.3. TIPI DI TEST (K2) | <i>40 minuti</i> |
|-------------------------------|------------------|

2.3.1. Termini

Testing black-box, copertura del codice, testing funzionale, testing di inter-operabilità, testing di carico, testing di manutenibilità, testing di prestazioni, testing di portabilità testing di affidabilità, testing di sicurezza, testing basati sulle specifiche, testing di stress, testing strutturale, testing di usabilità, testing white-box.

2.3.2. Contesto

Un gruppo di attività di test possono essere mirate alla verifica del sistema software (o una parte di un sistema) sulla base di una specifica ragione o di un obiettivo del testing.

Un tipo di test è focalizzato su un particolare obiettivo di test, che potrebbe essere il testing di una funzione che deve essere eseguita dal software; una caratteristica di qualità non-funzionale, come l'affidabilità o l'usabilità, la struttura o l'architettura del software o del sistema; oppure legato a cambiamenti, e quindi con l'obiettivo di confermare che i difetti siano stati corretti (testing confermativo) o di cercare eventuali effetti indesiderati di tali cambiamenti (testing di regressione).

Un modello del software può essere sviluppato e/o usato nel testing strutturale (modello di controllo del flusso o modello a struttura di menu), nel testing non-funzionale (modello di prestazioni, modello di usabilità, modello di sicurezza), nel testing funzionale (modello di flusso di un processo, modello basato su transizioni tra stati o specifiche in linguaggio naturale).

2.3.3. Testing di funzioni (testing funzionale) (K2)

Le funzioni che un sistema, sottosistema o componente deve eseguire possono essere descritte in varie forme: come vere e proprie specifiche dei requisiti, come 'use cases', come specifiche funzionali o addirittura non essere documentate. Le funzioni rappresentano "cosa" il sistema fa.

I test funzionali sono basati su funzioni, caratteristiche (descritte in documenti e comprese dai testers) e la loro interoperabilità con sistemi specifici, e possono essere eseguite a tutti i livelli di test (ad es. test per componenti possono essere basate su una specifica del componente).

Tecniche basate sulle specifiche possono essere usate per derivare 'test conditions' e 'test cases' dalle funzionalità del software o del sistema (si veda il Capitolo 4). Il testing funzionale considera il comportamento esterno del software (testing black-box) .

Un tipo di testing funzionale, il cosiddetto testing di sicurezza, investiga le funzioni (ad esempio, di firewall) legate alla rilevazione di minacce, come virus, da parte di intrusi potenzialmente pericolosi per la sicurezza del sistema. Un altro tipo di testing funzionale, detto testing di interoperabilità, valuta la capacità del prodotto software di interagire con uno o più componenti e/o sistemi esterni.

2.3.4. Testing di caratteristiche software non funzionali (testing non funzionale) (K2)

Il testing non funzionale include il testing di carico, il testing di stress, il testing di usabilità, il testing di manutenibilità , il testing di affidabilità e il testing di portabilità, ma non è limitato solo ad essi. Esso è il testing di "come" il sistema funziona.

Il testing non funzionale può essere eseguito a tutti i livelli di test. Il termine di testing non funzionale descrive i test richiesti per misurare caratteristiche dei sistemi e del software che possono essere

quantificate su una scala, come i tempi di risposta per il testing di prestazioni. Questi test possono essere ricondotti ad un modello di qualità come quello descritto in: ‘Software Engineering – Software Product Quality’ (ISO 9126). Il testing non funzionale prende in considerazione il comportamento esterno del software e nella maggior parte dei casi usa tecniche di progettazione black-box.

2.3.5. Testing dell’architettura/struttura del software (testing strutturale) (K2)

Il testing strutturale (white-box) può essere eseguito a tutti i livelli di test. Le tecniche strutturali sono più efficaci se usate dopo le tecniche basate sulle specifiche, in quanto aiutano a misurare l’accuratezza del testing attraverso una valutazione di copertura (coverage) di un tipo di struttura.

La copertura è la misura di quanto una struttura sia stata esercitata da una test suite, espressa come una percentuale degli elementi effettivamente coperti. Se la copertura è inferiore al 100%, allora possono essere progettati ulteriori test per quegli elementi che non erano stati coperti e, perciò, incrementare la copertura. Le tecniche di copertura sono discusse nel Capitolo 4.

A tutti i livelli di test, ma specialmente nel testing di componente e nel testing di integrazione, vengono generalmente usati degli strumenti per misurare la copertura del codice degli elementi, che possono ad esempio essere istruzioni o decisioni. Il testing strutturale può essere basato sull’architettura del sistema, come per esempio sulla gerarchia delle chiamate.

Gli approcci al testing strutturale possono anche essere applicati ai livelli di: testing di sistema, testing di integrazione di sistemi o testing di accettazione.

2.3.6. Testing legato a modifiche (retesting e regression testing) (K2)

Dopo che un difetto è stato rilevato e corretto, il software deve essere sottoposto ulteriormente a testing opportuno per avere la conferma che il difetto originale sia stato rimosso con successo: questa attività viene definita di testing confermativo (o retesting). Il debugging (cioè l’identificazione e la correzione dei difetti) è un’attività di sviluppo, non un’attività di testing.

Per testing di regressione si intende invece il testing ripetuto su di un programma già testato, dopo le modifiche apportate per la correzione dei difetti trovati, con l’obiettivo di identificare ogni difetto che possa essere stato introdotto o non coperto, a fronte appunto delle modifiche apportate. Questi difetti possono essere o nel software sotto test, o in un altro componente software legato o non legato ad esso. Esso viene eseguito quando il software oppure il suo ambiente viene modificato. L’entità del testing di regressione è basata sul rischio di non trovare difetti in software che precedentemente funzionava.

I test devono essere ripetibili se essi sono usati per il testing confermativo e per supportare il testing di regressione.

Il testing di regressione può essere eseguito a tutti i livelli di test e si applica al testing funzionale, al testing non-funzionale e al testing strutturale. Le suite di testing di regressione vengono fatte girare molte volte e generalmente evolvono lentamente: da questo punto di vista il regression testing assume un ruolo particolarmente importante anche nell’automazione dei test.

| | |
|--|------------------|
| 2.4. TESTING DI MANUTENZIONE (K2) | <i>15 minuti</i> |
|--|------------------|

2.4.1. Termini

Analisi di impatto, testing di manutenzione

2.4.2. Contesto

Una volta rilasciato ufficialmente, un software rimane spesso in servizio per anni o decenni. Durante questo periodo il sistema, i suoi dati di configurazione e il suo ambiente sono spesso corretti, modificati o estesi. La pianificazione preventiva dei rilasci è cruciale per il successo del testing di manutenzione. Occorre distinguere fra rilasci pianificati e hot fixes. Il testing di manutenzione viene fatto su un sistema esistente e in stato operativo e viene messo in atto a fronte di modifiche, migrazioni o ritiro del software o del sistema.

Le modifiche includono cambiamenti migliorativi pianificati (es. basati su nuove release), cambiamenti correttivi e/o di emergenza, cambiamenti di ambiente come ad esempio upgrades pianificati di database e/o di sistemi operativi, o patches da applicare a fronte di scoperte di punti di vulnerabilità del sistema operativo.

Il testing di manutenzione per migrazioni (ad esempio, da una piattaforma ad un'altra) deve includere test operativi relativi al nuovo ambiente, tanto quanto relativi al software modificato. Il testing di migrazione è necessario anche quando i dati di un'altra applicazione sono migrati nel sistema soggetto a manutenzione.

Il testing di manutenzione per il ritiro di un sistema può includere il testing di migrazione dei dati o di archiviazione, se sono richiesti lunghi periodi di mantenimento dei dati.

In aggiunta al testing di cosa è stato effettivamente modificato, il testing di manutenzione include anche un testing di regressione applicato alle parti del sistema che non sono state cambiate. L'ambito del testing di manutenzione è legato al rischio del cambiamento, della dimensione del sistema esistente e della dimensione delle modifiche.

A seconda delle modifiche, il testing di manutenzione potrebbe essere svolto ad alcuni o addirittura a tutti i livelli di test e per alcuni o per tutti i tipi di test. La stima di quanto un sistema esistente possa essere affetto da modifiche è chiamata "analisi di impatto" ed è utilizzata per aiutare a valutare l'entità del testing di regressione da effettuare.

Il testing di manutenzione può essere estremamente difficoltoso se le specifiche non sono aggiornate o addirittura mancanti o se testers con adeguata conoscenza del dominio non sono disponibili.

RIFERIMENTI

2.1.3 CMMI, Craig, 2002, Hetzel, 1988, IEEE 12207

2.2 Hetzel, 1988

2.2.4 Copeland, 2004, Myers, 1979

2.3.1 Beizer, 1990, Black, 2001, Copeland, 2004

2.3.2 Black, 2001, ISO 9126

2.3.3 Beizer, 1990, Copeland, 2004, Hetzel, 1988

2.3.4 Hetzel, 1988, IEEE 829

2.4 Black, 2001, Craig, 2002, Hetzel, 1988, IEEE 829

| | |
|----------------------------------|-----------|
| 3. TECNICHE STATICHE (K2) | 60 minuti |
|----------------------------------|-----------|

OBIETTIVI DI APPRENDIMENTO PER TECNICHE STATICHE

Gli obiettivi identificano quello si dovrà essere in grado di fare seguendo il completamento di ogni singolo modulo.

3.1 Tecniche statiche ed il processo di testing (K2)

- LO-3.1.1 Riconoscere i prodotti di lavoro del software che possono essere esaminati da differenti tecniche statiche. (K1)
- LO-3.1.2 Descrivere l'importanza ed il valore nel prendere in considerazione le tecniche statiche di valutazione di prodotti di lavoro del software. (K2)
- LO-3.1.3 Spiegare le differenze tra tecniche statiche e dinamiche. (K2)

3.2 Processo di revisione (K2)

- LO-3.2.1 Ricordare le fasi, i ruoli e le responsabilità di una tipica revisione formale. (K1)
- LO-3.2.2 Spiegare le differenze tra differenti tipi di revisioni: revisione informale, revisione tecnica, walkthrough e ispezione. (K2)
- LO-3.2.3 Spiegare i fattori che consentono di effettuare con successo una revisione. (K2)

3.3 Analisi statica con strumenti (K2)

- LO-3.3.1 Ricordare i difetti tipici e gli errori identificati dall'analisi statica e confrontarli a quelli delle revisioni e del testing dinamico. (K1)
- LO-3.3.2 Elencare, usando esempi, i tipici benefici della analisi statica. (K1)
- LO-3.3.3 Elencare i tipici difetti del codice e del design che possono essere identificati dagli strumenti di analisi statica. (K1)

| | |
|--|-----------|
| 3.1. TECNICHE STATICHE ED IL PROCESSO DI TESTING (K2) | 15 minuti |
|--|-----------|

3.1.1. Termini

Testing dinamico, testing statico, tecniche statiche

3.1.2. Contesto

A differenza del testing dinamico, che richiede l'esecuzione del software, le tecniche di testing statico si basano sull'esame manuale (revisione) e sull'analisi automatica (analisi statica) del codice o di altra documentazione di progetto senza l'esecuzione del codice.

Le revisioni (reviews) sono una modalità di testing dei prodotti di lavoro dello sviluppo software (compreso il codice) e possono essere eseguite efficacemente prima dell'esecuzione dinamica dei test. I difetti rilevati durante le revisioni svolte agli inizi del ciclo di vita sono spesso molto più economici da rimuovere di quelli rilevati durante l'esecuzione dei test (ad esempio, difetti trovati nei requisiti).

Una revisione potrebbe essere svolta interamente come attività manuale, ma spesso si utilizzano anche degli strumenti di supporto. La principale attività manuale consiste nell'esaminare un prodotto di lavoro e commentarlo opportunamente. Ogni prodotto di lavoro di sviluppo software può essere sottoposto a review, comprese le specifiche dei requisiti, i documenti di progettazione, il codice, i 'test plans', le specifiche di test, i 'test cases', i 'test scripts', i manuali utente o le pagine web.

I benefici delle revisioni includono una precoce rilevazione e correzione di difetti, miglioramenti nella produttività dello sviluppo e conseguente riduzione dei tempi, riduzione dei costi e tempi del testing, riduzioni dei costi complessivi del ciclo di vita, riduzione dei difetti e miglioramento della comunicazione. Le revisioni possono rilevare mancanze, per esempio nei requisiti, che vengono rarissimamente trovate nel testing dinamico .

Le revisioni, l'analisi statica ed il testing dinamico hanno lo stesso obiettivo – l'identificazione di difetti. Esse sono complementari: le differenti tecniche possono trovare differenti tipi di difetti in modo efficace ed efficiente. Rispetto al testing dinamico, le tecniche statiche trovano le cause di potenziali 'failures' (ovvero difetti) piuttosto che le 'failures' stesse.

Tipici difetti che sono più facili da trovare nelle revisioni rispetto al testing dinamico sono: deviazioni dagli standard, difetti sui requisiti , difetti di progettazione, specifiche errate delle interfacce.

| | |
|--|------------------|
| 3.2. PROCESSO DI REVISIONE (K2) | <i>25 minuti</i> |
|--|------------------|

3.2.1. Termini

Criterio di ingresso, revisione formale, revisione informale, ispezione, metrica, moderatore/leader dell'ispezione, 'peer review', revisore, verbalizzatore, revisione tecnica walkthrough.

3.2.2. Contesto

I diversi tipi di reviews variano da un livello molto informale (ad esempio, senza istruzioni scritte per i revisori) ad un livello molto formale (quindi molto ben strutturate e regolamentate da apposite procedure). Il grado di formalità di un processo di una review è legato a fattori come la maturità del processo di sviluppo, la presenza di requisiti legali o regolamentari o la necessita di un 'audit trail'.

Il modo con cui una review viene svolta dipende dagli obiettivi concordati (ad esempio, ricerca di difetti, miglioramento della conoscenza o discussione e decisioni per consenso).

3.2.3. Fasi di una revisione formale (K1)

Una review formale segue tipicamente le seguenti fasi principali:

1. Pianificazione:
 - definizione dei criteri di review
 - selezione dei partecipanti
 - allocazione dei ruoli
 - definizione dei criteri di entrata e di quelli di uscita
 - scelta di quali parti di documenti siano sottoposti a review
2. Kick-off:
 - verifica dei criteri di ingresso (per le revisioni formali),
 - distribuzione dei documenti;
 - spiegazione degli obiettivi dei processi e dei documenti ai partecipanti
3. Preparazione individuale:
 - Preparazione al review meeting con la revisione dei documenti
4. Identificazione di potenziali difetti e preparazioni di eventuali domande e commenti
5. Esame/valutazione/registrazione dei risultati (Review meeting):

- discussione e/o registrazione, con risultati documentati o minute/verbali (per i tipi di review più formali)
 - annotare difetti, dare delle raccomandazioni per la gestione dei difetti, prendere decisioni circa il trattamento dei difetti
 - esame/valutazione/registrazione durante ogni incontro o tracciatura di ogni comunicazione elettronica
6. Rilavorazione
7. Correzioni dei difetti trovati, tipicamente svolte dall'autore:
- registrazione dello stato aggiornato dei difetti (nelle review formali)
8. Follow-up:
- controllo che i difetti siano stati corretti
 - raccolta di metriche
 - verifica dei criteri di uscita (per le tipologie di review più formali)

3.2.4. Ruoli e responsabilità (K1)

Una review formale include tipicamente i seguenti ruoli:

- Manager: decide sulla necessità di esecuzione di reviews, alloca il loro svolgimento nelle schedalizzazioni di progetto e determina se gli obiettivi della review sono stati soddisfatti.
- Moderatore: la persona che conduce la review del documento o dell'insieme di documenti, compresa la pianificazione della review, lo svolgimento del meeting ed il follow-up successivo al meeting. Se necessario, il moderatore può mediare i vari punti di vista ed è la persona sulla quale si basa il successo o meno della review.
- Autore: è l'autore o la persona con la principale responsabilità del/i documento/i oggetto della review.
- Revisori: persone (chiamate anche verificatori o ispettori) con un background specifico (ad esempio, tecnico) i quali, dopo la necessaria preparazione, identificano e descrivono gli elementi trovati (ad esempio, difetti) nel prodotto sotto review. I revisori devono essere scelti in modo da rappresentare differenti punti di vista e ruoli nel processo di review e devono prendere parte ad ogni review meeting.
- Segretario (o registratore): documenta tutte le questioni, i problemi e i punti aperti che sono stati identificati durante il meeting.

Visionare i documenti da differenti prospettive ed usare delle checklists può rendere le reviews più efficaci ed efficienti: per esempio avere checklists basate sui punti di vista di utenti, testers ecc. o una checklist di tipici problemi nei requisiti.

3.2.5. Tipi di revisioni (K2)

Un singolo documento può essere oggetto di più di una review. Se si usa più di un tipo di review, l'ordine può variare. Per esempio, una review informale può essere svolta prima di una review tecnica, oppure un'ispezione, può essere fatta su una specifica dei requisiti prima di una walkthrough con il cliente. Le principali caratteristiche, opzioni, e scopi dei tipi di review più comuni sono:

Revisione informale

Caratteristiche chiave:

- processo informale;
- svolta come 'pair programming' o tramite un leader tecnico che revisiona design e codice;
- opzionalmente può essere documentata;

- può variare nella sua utilità a seconda del revisore;
- scopo principale: modo economico per ottenere dei benefici.

Walkthrough

Caratteristiche chiave:

- meeting condotto dall'autore;
- scenari, dry-runs (esecuzioni), 'peer group';
- svolgimento a sessioni: aperture e chiusura per sessione;
- opzionalmente un pre-meeting di preparazione dei reviewers, con review report, e verbalizzatore (che non è l'autore);
- può variare in pratica, da un grado abbastanza informale a molto formale;
- scopi principali: comprensione, acquisizioni di informazioni, scoperta di difetti.

Revisione tecnica

Caratteristiche chiave:

- documentata, processo di rilevazione dei difetti ben definito che include colleghi ed esperti tecnici;
- può essere eseguita come una 'peer review' con alcune restrizioni sulle partecipazioni;
- idealmente condotta dal moderatore (non dall'autore);
- preparazione pre-meeting;
- opzionalmente uso di checklists, di review reports;
- può variare da un grado abbastanza informale a molto formale;
- scopi principali: discussioni, presa di decisioni, valutazione di alternative, scoperta di difetti, risoluzione di problemi tecnici e controlli di rispetto di specifiche e di standards.

Ispezione

Caratteristiche chiave:

- condotta dal moderatore (non dall'autore);
- generalmente svolta per 'peer examination';
- ruoli ben definiti;
- inclusione di metriche;
- processo formale basato su regole e checklists con criteri di ingresso ed uscita;
- preparazione pre-meeting;
- report dell'ispezione, lista dei problemi individuati;
- processo 'follow-up' (a seguito) formale;
- opzionalmente, lettura e miglioramento del processo;
- scopo principale: trovare difetti.

I walkthroughs, le reviews tecniche e le ispezioni possono essere eseguite in un 'peer group' – ovvero un gruppo di colleghi allo stesso livello organizzativo. Questo tipo di review viene chiamata una 'peer review'.

3.2.6. Fattori di successo delle revisioni (K2)

Fattori di successo per le revisioni includono:

- Ogni singola review ha un chiaro obiettivo predefinito.
- Devono essere coinvolte le persone giuste per il raggiungimento degli obiettivi della review.
- I difetti trovati sono sempre ben accolti ed espressi oggettivamente.
- I testers sono preziosi revisori che contribuiscono alla review ed approfondiscono la conoscenza del prodotto, il che permette loro di preparare i test in anticipo.

- Sono presi in considerazione i fattori umani e gli aspetti psicologici (ad esempio, rendendo la review un'esperienza positiva per l'autore).
- La review si svolge in un clima di fiducia; i risultati non saranno utilizzati per la valutazione dei partecipanti
- Le tecniche di review applicate sono adattate al raggiungimento degli obiettivi nonché al tipo ed al livello del prodotto di lavoro dello sviluppo software e dei revisori.
- Le checklists o i ruoli sono usati, se appropriati, per incrementare l'efficacia nell'identificazione dei difetti.
- La formazione è necessaria nelle review tecniche, specialmente in quelle più formali come l'ispezione.
- Una buona gestione supporta un buon processo di review (ad esempio, considerando un tempo adeguato per le attività di review nelle pianificazioni di progetto).
- Viene data enfasi sul miglioramento del processo e sull'apprendimento

| | |
|--|------------------|
| 3.3. ANALISI STATICA CON STRUMENTI (K2) | <i>20 minuti</i> |
|--|------------------|

3.3.1. Termini

Compilatore, complessità, flusso di controllo, flusso dati, analisi statica.

3.3.2. Contesto

L'obiettivo dell'analisi statica è di trovare difetti nel codice sorgente del software e nei modelli del software.

L'analisi statica viene condotta per mezzo dello strumento senza l'esecuzione del software che è sotto esame; il testing dinamico esegue il codice del software. L'analisi statica può localizzare difetti che sono difficili da trovare nel testing. Come per le reviews, anche l'analisi statica trova difetti piuttosto che failures. Gli strumenti di analisi statica analizzano il codice del programma (ad esempio, il flusso di controllo e il flusso dati), tanto quanto l'output generato, come HTML e XML.

Il valore aggiunto della analisi statica può comprendere:

- Una precoce rilevazione dei difetti, precedente all'esecuzione dei test.
- Una precoce segnalazione di aspetti potenzialmente critici nel codice o nel design, attraverso il calcolo di opportune metriche, come ad esempio un'elevata misura di complessità .
- Identificazione dei difetti non facilmente rilevabili dal testing dinamico.
- Rilevazione delle dipendenze e delle inconsistenze nei modelli del software, come ad esempio i collegamenti.
- Aumentata manutenibilità del codice e della progettazione.
- Prevenzione dei difetti, se i difetti che l'analisi statica tipicamente evidenzia vengono ricordati e quindi non introdotti erroneamente nella fase di sviluppo .

I tipici difetti scoperti dagli strumenti di analisi statica includono:

- riferimenti a variabili con valore indefinito;
- inconsistenza delle interfacce tra i moduli ed i componenti;
- variabili che non vengono mai usate;
- codice irraggiungibile (spesso detto codice morto);
- logica mancante od erronea (cicli infiniti potenziali);
- complicati costrutti in overlay;

- violazioni degli standard di programmazione;
- punti di vulnerabilità della sicurezza;
- violazioni sintattiche del codice e dei modelli software.

Gli strumenti di analisi statica sono tipicamente usati dagli sviluppatori (per il controllo del rispetto di regole predefinite o di standard di programmazione) prima e durante il testing dei componenti e di integrazione o nella promozione del codice con strumenti di gestione della configurazione, e dai progettisti durante la modellazione del software. Gli strumenti di analisi statica possono produrre un grosso numero di messaggi di allarme/segnalazione, che è importante gestire in modo opportuno per consentire un più efficace utilizzo dello strumento.

I compilatori possono essere di supporto per l'analisi statica, compreso il calcolo di metriche .

RIFERIMENTI

3.2 IEEE 1028

3.2.2 Gilb, 1993, van Veenendaal, 2004

3.2.4 Gilb, 1993, IEEE 1028

3.3 Van Veenendaal, 2004

| | |
|--|-------------------|
| 4. TECNICHE DI TEST DESIGN (K3) | 285 minuti |
|--|-------------------|

OBIETTIVI DI APPRENDIMENTO PER LE TECNICHE DI TEST DESIGN (PROGETTAZIONE DEI TEST)

Gli obiettivi identificano quello che dovrai essere in grado di fare seguendo il completamento di ogni singolo modulo.

4.1 Il processo di sviluppo di test (K2)

- LO-4.1.1 Indicare le differenze tra una specifica di test design, una specifica di 'test case' ed una specifica di 'test procedure'. (K2)
- LO-4.1.2 Confrontare i termini di 'test condition', 'test case' e 'test procedure'. (K2)
- LO-4.1.3 Valutare la qualità dei 'test cases', in termine di una chiara tracciabilità ai requisiti e di risultati attesi.. (K2)
- LO-4.1.4 Tradurre i test cases in una ben strutturata specifica di 'test procedure' ad un livello di dettaglio proporzionato alla conoscenza dei testers. (K3)

4.2 Categorie di tecniche di test design (K2)

- LO-4.2.1 Ricordare le ragioni per le quali sia l'approccio basato sulle specifiche (black-box) che quello basato sulla struttura (white-box), sono utili per la progettazione di 'test cases', ed elencare per ognuno di essi le tecniche più comuni. (K1)
- LO-4.2.2 Spiegare le caratteristiche e le differenze tra il testing basato sulle specifiche, il testing basato sulla struttura e il testing basato sull' esperienza. (K2)

4.3 Tecniche basate sulle specifiche o black-box (K3)

- LO-4.3.1 Scrivere dei 'test cases' per pre-definiti modelli software, usando le seguenti tecniche di test design: (K3)
 - o partizionamento in classi di equivalenza (equivalence partitioning);
 - o analisi ai valori limite (boundary value analysis);
 - o testing basato su tabelle delle decisioni (decision table testing);
 - o testing basato su diagrammi di transizioni tra stati (state transition testing).
- LO-4.3.2 Spiegare il principale obiettivo di ognuna delle quattro tecniche, a quale livello e tipo di testing possono essere usate e come misurarne la copertura. (K2)
- LO-4.3.3 Spiegare il concetto di 'use case' testing ed i suoi benefici. (K2)

4.4 Tecniche basate sulla struttura o tecniche white-box (K3)

- LO-4.4.1 Descrivere il concetto e l'importanza della copertura del codice. (K2)
- LO-4.4.2 Spiegare i concetti di copertura delle istruzioni e delle decisioni, e comprendere che questi concetti possono essere usati anche ad altri livelli rispetto al testing di componenti (ad esempio, su procedure di business a livello di sistema). (K2)
- LO-4.4.3 Scrivere dei 'test cases' da un dato flusso di controllo usando istruzioni e tecniche di test design (K3)
- LO-4.4.4 Valutare la completezza degli "statement coverage" e "decision coverage" rispetto ai predefiniti criteri di uscita. (K3)

4.5 Tecniche basate sull'esperienza (K2)

- LO-4.5.1 Ricordare le motivazioni per la scrittura di test cases basati sull'intuizione, sull'esperienza e sulla conoscenza dei difetti più comuni. (K1)
- LO-4.5.2 Confrontare le tecniche basate sull'esperienza con le tecniche basate sulle specifiche. (K2)

4.6 Scelta delle tecniche di testing (K2)

- LO-4.6.1 Classificare le tecniche di test design in base alla loro idoneità in un particolare contesto, come le basi di test, i modelli applicabili e le caratteristiche del software (K2)

15 minuti

4.1. IL PROCESSO DI SVILUPPO DI TEST (K2)

4.1.1. Termini

Specifica di 'test case', test design, pianificazione della esecuzione dei test, specifica di 'test procedure', test script, tracciabilità

4.1.2. Contesto

Il processo descritto in questa sezione può essere svolto in modi differenti, da un livello molto informale con poca o nessuna documentazione ad un livello molto formale (come descritto di seguito). Il livello di formalità dipende dal contesto del testing, comprendente l'organizzazione, il grado di maturità dei processi di sviluppo e di testing, i vincoli sui tempi, i requisiti di sicurezza od obbligatori e le persone coinvolte.

Durante l'analisi dei test, la documentazione di base di test viene analizzata in modo da determinare che cosa effettivamente sottoporre a test, e quindi identificare le 'test conditions'. Una 'test condition' viene definita come elemento od un evento che potrebbe essere verificato/scatenato da uno o più test cases (ad esempio, una funzione, una transazione, una caratteristica di qualità o un elemento strutturale).

Assicurare la tracciabilità .dalle 'test conditions' verso le specifiche e i requisiti, consente di effettuare sia un'analisi di impatto., quando i requisiti cambiano, sia una copertura dei requisiti che deve essere determinata per un insieme di test. Durante l'analisi dei test viene utilizzato un approccio di test dettagliato per selezionare le tecniche di test design da usare, basate, tra le tante considerazioni, sui rischi identificati (si veda il Capitolo 5 per maggiori informazioni sulla analisi del rischio).

Durante il test design sono creati e specificati i 'test cases' ed i dati di test. Un 'test case' consiste di un insieme di valori di input, di precondizioni di esecuzione, di risultati attesi e di post-condizioni di esecuzione, definiti per coprire una o più 'test conditions'. Lo 'Standard for Software Test Documentation' (IEEE 829) descrive il contenuto delle 'test design specifications' (contenenti 'test conditions') e delle 'test case specifications'.

I risultati attesi devono essere descritti come parte della specifica di un test case e includere outputs, cambiamenti ai dati e agli stati, nonché ogni altra conseguenza del test. Se i risultati attesi non sono stati definiti in modo preciso, come appena detto, allora un risultato plausibile, ma comunque errato, potrebbe essere interpretato come corretto. I risultati attesi devono essere idealmente definiti prima della esecuzione dei test.

Durante l'implementazione dei test, vengono sviluppati i test cases, ne sono definite la priorità e vengono organizzati nella cosiddetta 'test procedure specification'. La 'test procedure' specifica la sequenza di azioni per l'esecuzione di un test. Se i test vengono eseguiti utilizzando uno strumento di esecuzione dei test, la sequenza di azioni viene specificata in un test script , (che è una 'test procedure' automatizzata).

Le varie 'test procedures' ed i test scripts automatici sono successivamente raggruppati in una pianificazione delle esecuzione dei test, che definisce l'ordine nel quale le varie 'test procedures' e gli eventuali test scripts automatici vengono eseguiti, quando essi devono essere lanciati e da chi. La pianificazione dell'esecuzione dei test dovrà tenere conto di fattori come il testing di regressione, la priorità e le dipendenze tecniche e logiche.

15 minutes

4.2. CATEGORIE DELLE TECNICHE DI TEST DESIGN (K2)

4.2.1. Termini

Tecnica di progettazione black-box, test design, tecnica di progettazione basata sull'esperienza, tecnica di progettazione basata sulle specifiche, tecnica di progettazione basata sulla struttura, tecnica di progettazione white-box test.

4.2.2. Contesto

Lo scopo di una tecnica di test design è di identificare le 'test conditions' ed i 'test cases'.

La più classica distinzione è quella di classificare le tecniche di test come black-box o white-box .

Le tecniche black-box (chiamate anche tecniche basate sulle specifiche) sono un modo per derivare e selezionare le 'test conditions', i 'test cases' ed i dati di test basato sull'analisi della documentazione di base del test. Essa include sia il test funzionale che non-funzionale. Il black-box testing, per definizione, non utilizza nessuna informazione riguardante la struttura interna del componente o sistema da testare. Le tecniche white-box (chiamate anche tecniche basate sulla struttura o strutturali) si basano su un'analisi della struttura del componente o del sistema. Il testing black-box e white-box può anche essere abbinato con tecniche basate sull'esperienza per avvalersi dell'esperienza di sviluppatori, testers ed utenti nel definire cosa dover testare.

Alcune tecniche ricadono chiaramente in una singola categoria, mentre altre hanno elementi caratteristici di più di una categoria.

Questo syllabus considera le tecniche di test-design basate sulle specifiche come tecniche black-box e le tecniche di test-design basate sulla struttura come tecniche white-box. Inoltre sono prese in considerazione anche le tecniche di test-design basate sull'esperienza.

Caratteristiche comuni alle tecniche basate sulle specifiche sono:

- Vengono usati modelli, formali o informali, per la specifica del problema che deve essere risolto, del software o dei suoi componenti.
- I 'test cases' possono essere derivati sistematicamente da questi modelli.

Caratteristiche comuni alle tecniche basate sulla struttura sono:

- Vengono usate le informazioni relative a come il software è costruito per derivare i 'test cases', per esempio, codice e design.
- L'entità della copertura del software può essere misurata per i 'test cases' esistenti, ed ulteriori 'test cases' possono essere derivati sistematicamente per aumentare la copertura.

Caratteristiche comuni alle tecniche basate sulla esperienza sono:

- La conoscenza e l'esperienza delle persone sono usate per derivare i 'test cases'
- La conoscenza di testers, sviluppatori, utenti ed altri stakeholders relativa al software, al suo utilizzo ed al suo ambiente è una fonte di informazioni.
- La conoscenza legata a difetti tipici ed alla loro distribuzione è un'altra fonte di informazioni.

150 minuti

4.3. TECNICHE BASATE SULLE SPECIFICHE O BLACK-BOX (K3)

4.3.1. Termini

Analisi ai valori limite (boundary value analysis), testing basato su tabelle delle decisioni (decision table testing), partizionamento in classi di equivalenza (equivalence partitioning), testing basato su diagrammi di transizioni tra stati (state transition testing, use case testing).

4.3.2. Partizionamento in classi di equivalenza (K3)

I dati di ingresso al software o al sistema sono divisi in gruppi per i quali ci si aspetta che esibiscano un comportamento simile, ovvero siano processati allo stesso modo. Le partizioni di equivalenza (o classi di equivalenza) possono essere individuate sia per dati validi che per dati invalidi (ovvero per valori che devono essere rifiutati). Le partizioni possono essere anche identificate sulla base degli outputs, dei valori interni, dei valori legati al tempo (ad esempio, prima o dopo il verificarsi di un evento) e per i parametri delle interfacce (ad esempio, durante integration testing). I test possono così essere progettati per coprire le partizioni. Il partizionamento in classi di equivalenza è applicabile a tutti i livelli di testing.

Il partizionamento in classi di equivalenza può essere usata per realizzare la copertura sia di inputs che di outputs. Inoltre essa può essere applicata all'input umano, all'input attraverso interfacce di un sistema od a parametri di interfaccia nel testing di integrazione.

4.3.3. Analisi ai valori limite (K3)

Il comportamento ai margini di ogni singola partizione di equivalenza è spesso scorretto, e quindi questo fa sì che i cosiddetti 'boundaries' (ovvero i limiti, gli estremi) siano un'area dove il testing riesce a trovare difetti piuttosto frequentemente.

I valori limite massimo e minimo di una partizione sono i suoi valori di boundary. Un valore limite per una partizione valida è un valore di limite valido; un valore di limite per una partizione invalida è un valore di limite invalido: i test possono essere progettati per coprire sia valori di limiti validi che invalidi. Nella progettazione di test cases, viene scelto un test per ogni singolo valore limite.

L'analisi ai valori limite può essere applicata a tutti i livelli di test. Essa è relativamente facile da applicare e la sua capacità di trovare errori è elevata: specifiche dettagliate sono di significativo aiuto per individuare i valori limite interessanti.

Questa tecnica è spesso considerata come una estensione del partizionamento in classi di equivalenza o di altre tecniche di test design. Essa può essere usata su classi di equivalenza per input su schermo utenti, tanto quanto, per esempio, su intervalli temporali (ad esempio, per time out o requisiti di velocità transazionali), o intervalli di tabelle (ad esempio, tabella 256*256).

4.3.4. Testing basato su tabelle delle decisioni (K3)

Le tabelle delle decisioni sono un buon modo per catturare i requisiti di sistema che contengono condizioni logiche e per documentare la progettazione interna del sistema. Esse possono essere usate per registrare delle complesse regole che un sistema deve implementare. La specifica viene analizzata e le condizioni e le azioni del sistema vengono identificate. Le condizioni di input e le azioni sono molto spesso dichiarate in modo tale che esse possano essere o vere o false (booleane). Le tabelle delle decisioni contengono le condizioni di innesco, spesso combinazioni di vero e falso per tutte le condizioni di input, e le azioni risultanti per ogni singola combinazione di condizioni. Ogni singola colonna della tabella

corrisponde alla regola che definisce una combinazione unica di condizioni, la quale risulta nell'esecuzione delle azioni associate con quella regola. La copertura standard comunemente usata con il testing basato su tabelle delle decisioni è di avere almeno un test per colonna, il quale tipicamente coinvolge la copertura di tutte le combinazioni delle condizioni di innesco.

La forza del testing basato su tabelle delle decisioni è che esso crea combinazioni di condizioni che possono non essere state sollecitate in altro modo durante il testing. Inoltre esso può essere applicato a tutte quelle situazioni in cui l'esecuzione del software dipende da diverse decisioni logiche.

4.3.5. Testing basato su diagrammi di transizioni tra stati (K3)

Un sistema può esibire una risposta differente a seconda della condizione attuale o della sua storia precedente (il suo stato). In questo caso, questo aspetto del sistema può essere mostrato attraverso un diagramma di transizioni tra stati. Tale diagramma consente al tester di vedere il software in termini dei suoi stati, delle transizioni tra questi stati, degli inputs o degli eventi che innescano i cambiamenti di stato (le transizioni) e le azioni che possono risultare da queste transizioni. Gli stati del sistema o dell'oggetto sotto test sono separati, identificabili ed in numero finito. Una tabella di stato mostra le relazioni tra gli stati e gli inputs e può evidenziare quali possibili transizioni sono invalide. I test possono essere progettati per coprire una tipica sequenza di stati, per coprire ogni stato, per esercitare ogni transizione, per esercitare specifiche sequenze di transizioni o per testare transizioni invalide.

Il testing basato su diagrammi di transizioni tra stati viene molto usato nell'industria del software embedded e nell'automazione in generale. Comunque, questa tecnica può essere utile anche per modellare un oggetto di business che abbia specifici stati o per il testing di flussi di dialogo sullo schermo (ad esempio, per applicazioni internet).

4.3.6. Testing basato su 'use cases' (K2)

I test possono essere derivati da 'use cases'. Uno 'use case' descrive le interazioni tra gli attori(utenti e sistemi), il che produce un risultato significativo ad un utente o ad un cliente del sistema. Uno 'use case' può essere descritto a livello astratto (use case di business, livello di processi di business, indipendenza dalla tecnologia), oppure a livello di sistema (use case di sistema a livello di funzionalità del sistema). Ogni 'use case' ha delle precondizioni, le quali necessitano di essere soddisfatte affinché lo 'use case' abbia successo. Ogni 'use case' termina con delle post condizioni, le quali sono i risultati osservabili e lo stato finale del sistema dopo che lo 'use case' è stato completato. Uno 'use case' ha generalmente uno scenario principale e scenari alternativi.

Gli 'use cases' descrivono il flusso del processo attraverso un sistema basato sul suo tipico effettivo utilizzo, così che i 'test cases' derivati dagli 'use cases' sono più utili nell'individuare difetti nel flusso del processo durante l'utilizzo effettivo del sistema. Gli 'use cases' sono molto utili per la progettazione dei test di accettazione con la partecipazione di clienti/utenti. Essi possono anche svelare difetti di integrazione causati dalla interazione e dalla interferenza di differenti componenti, che non potevano essere trovati durante il testing di singoli componenti.

60 minuti

4.4. TECNICHE BASATE SULLA STRUTTURA O WHITE-BOX (K3)

4.4.1. Termini

Copertura del codice, decision coverage (copertura delle decisioni), statement coverage (copertura delle istruzioni, testing basato sulla struttura

4.4.2. Contesto

Il testing basato sulla struttura (o testing white-box) è basato su una struttura identificata del software o del sistema, come descritto dai seguenti esempi:

- Livello di componente: la struttura di un componente software, ad esempio istruzioni, decisioni (o branches).
- Livello di Integrazione: la struttura può essere un albero delle chiamate (un diagramma nel quale moduli chiamano altri moduli).
- Livello di sistema: la struttura può essere una struttura a menu, una struttura di una pagina web ecc.

In questa sezione, sono discusse due tecniche basate sulla struttura per la copertura del codice, basate sulle istruzioni (statement coverage) sulle decisioni (decision coverage). Per il testing basato sulla copertura del codice, può essere usato un diagramma di controllo di flusso per visualizzare le alternative di ogni singola decisione.

4.4.3. Testing delle istruzioni e copertura delle istruzioni (K3)

Nel testing dei componenti, la copertura delle istruzioni è la valutazione della percentuale di istruzioni eseguite da una suite di test cases. La tecnica del testing delle istruzioni crea i test cases per eseguire istruzioni specifiche, normalmente per aumentare la loro copertura.

La copertura delle istruzioni è determinata dal numero delle istruzioni eseguibili attivate dai test cases diviso il numero totale delle istruzioni eseguibili del codice soggetto a test.

4.4.4. Testing delle decisioni e copertura delle decisioni (K3)

La copertura delle decisioni, legata al testing delle decisioni, è la valutazione della percentuale di decisioni (ad esempio, le opzioni Vero e Falso di uno statement di IF) che sono state eseguite da un insieme di test cases. Il testing delle decisioni crea dei test cases per eseguire decisioni specifiche, Un branch trae origine da uno statement di decisione nel codice e trasferisce il controllo ad una posizione differente del codice.

La copertura delle decisioni è determinata dal numero di tutte le possibili decisioni attivate dai test cases diviso il numero totale di tutte le possibili decisioni presenti nel codice soggetto a test.

Il testing delle decisioni è una forma di testing del flusso di controllo, dato che esso stimola l'esecuzione di specifici flussi di controllo attraverso i punti di decisione. La copertura delle decisioni è più forte della copertura degli statement: infatti il 100% di copertura delle decisioni garantisce anche il 100% di copertura delle istruzioni, ma non vale il viceversa.

4.4.5. Altre tecniche basata sulla struttura (K1)

Ci sono altri livelli di copertura del codice che sono ancora più forti della copertura delle decisioni, per esempio, la copertura delle condizioni e la copertura delle condizioni multiple.

Il concetto di copertura può essere anche applicato ad altri livelli di test (ad esempio, di integrazione) dove la percentuale di moduli, componenti o classi che sono state eseguite da una suite di test cases può essere espressa rispettivamente come una copertura di modulo, di componente o di classe.

Per il testing strutturale del codice e per la sua copertura è praticamente inevitabile il supporto di strumenti specifici.

| | |
|--|------------------|
| 4.5. TECNICHE BASATE SULL'ESPERIENZA (K2) | <i>30 minuti</i> |
|--|------------------|

4.5.1. Termini

Testing esplorativo, guasto, attacco.

4.5.2. Contesto

Il testing basato sull'esperienza, si ha quando i test sono derivati dalle capacità dei testers, e dalla loro intuizione ed esperienza con applicazioni e tecnologie simili. Queste tecniche, quando usate per rafforzare le tecniche sistematiche, possono essere utili nell'identificazione di test speciali che non possono essere facilmente individuati da tecniche formali, specialmente quando applicate dopo più approcci formali. In ogni caso, questa tecnica può produrre un ampio spettro di gradi di efficacia, a seconda della esperienza del tester.

Una tecnica basata sull'esperienza comunemente usata è la cosiddetta 'error guessing'. Generalmente i testers prevedono i difetti sulla base della loro esperienza. Un approccio strutturato alla tecnica di 'error guessing' consiste nell'enumerare una lista di possibili errori e di progettare i test che scovino questi errori. Questo approccio sistematico viene chiamato 'fault attack'. Queste liste di difetti e di 'failures' possono essere costruite sulla base dell'esperienza, su dati storici di difetti e 'failures', e dalla conoscenza comune delle principali cause di errori nel software.

Il testing esplorativo prevede lo svolgimento contemporaneo di progettazione di test design, esecuzione, logging dei 'test cases' e apprendimento, in base a un documento di test contenente gli obiettivi del test, ed eseguito in periodi temporali di lunghezza prefissata. Si tratta di un approccio che è molto utile dove ci sono poche o inadeguate specifiche e tempistiche molto ristrette, oppure per aumentare in modo complementare, l'efficacia di un altro testing più formale. Esso può servire come un ulteriore controllo sul processo di test, per assicurare che i difetti più seri siano stati trovati.

| | |
|---|------------------|
| 4.6. SCELTA DELLE TECNICHE DI TESTING (K2) | <i>15 minuti</i> |
|---|------------------|

Termini

Nessun termine specifico.

Contesto

La scelta di quali tecniche di test utilizzare dipende da un elevato numero di fattori, che includono il tipo di sistema, gli standards da applicare, i vincoli di tipo contrattuale, il livello di rischio, il tipo di rischio, l'obiettivo del test, la documentazione disponibile, la conoscenza dei testers, il tempo ed il budget, il ciclo di vita dello sviluppo, i modelli di 'use cases' e precedenti esperienze dei tipi di difetti trovati.

Alcune tecniche sono maggiormente applicabili ed efficaci in certe situazioni ed in certi livelli di test, mentre altre sono applicabili a tutti i livelli di test.

I testers, nel creare i test cases, usano normalmente una combinazione di tecniche di test per assicurare un adeguato perseguimento degli obiettivi del test.

RIFERIMENTI

- 4.1 Craig, 2002, Hetzel, 1988, IEEE 829
- 4.2 Beizer, 1990, Copeland, 2004
- 4.3.1 Copeland, 2004, Myers, 1979
- 4.3.2 Copeland, 2004, Myers, 1979
- 4.3.3 Beizer, 1990, Copeland, 2004
- 4.3.4 Beizer, 1990, Copeland, 2004
- 4.3.5 Copeland, 2004
- 4.4.3 Beizer, 1990, Copeland, 2004
- 4.5 Kaner, 2002
- 4.6 Beizer, 1990, Copeland, 2004

| | |
|-------------------------------------|------------|
| 5. GESTIONE DEL TESTING (K3) | 170 minuti |
|-------------------------------------|------------|

OBIETTIVI DI APPRENDIMENTO PER LA GESTIONE DEL TESTING

Gli obiettivi identificano quello che dovrai essere in grado di fare seguendo il completamento di ogni singolo modulo.

5.1 Organizzazione del testing (K2)

- LO-5.1.1 Riconoscere l'importanza del testing indipendente. (K1)
- LO-5.1.2 Elencare i benefici e gli svantaggi del testing indipendente all'interno di un'organizzazione. (K2)
- LO-5.1.3 Riconoscere i differenti membri del team da prendere in considerazione per la creazione di un team di test. (K1)
- LO-5.1.4 Ricordare i compiti specifici di un tipico test leader e di un tipico tester. (K1)

5.2 Stima e pianificazione del testing (K2)

- LO-5.2.1 Riconoscere i differenti livelli ed obiettivi della pianificazione dei test. (K1)
- LO-5.2.2 Riassumere l'ambito ed il contenuto dei documenti di 'test plan', di 'test design specification' e di 'test procedure' in accordo con lo 'Standard for Software Test Documentation' (IEEE 829). (K2)
- LO-5.2.3 Distinguere approcci al test concettualmente differenti, come ad esempio analitici, basati su modelli, metodologici, aderenti a processi/standard, dinamici/euristici, consultivi e contrari alla regressione. (K2)
- LO-5.2.4 Distinguere la pianificazione del testing per un sistema e la programmazione dell'esecuzione dei test. (K2)
- LO-5.2.5 Scrivere un programma di esecuzione di test, per un dato insieme di test cases, considerando le priorità e le dipendenze tecniche e logiche. (K3)
- LO-5.2.6 Elencare le attività di preparazione e di esecuzione dei test che devono essere considerate durante la pianificazione dei test. (K1)
- LO-5.2.7 Ricordare i tipici fattori che influenzano lo sforzo legato al testing. (K1)
- LO-5.2.8 Distinguere i due approcci di stima concettualmente differenti: l'approccio basato sulle metriche e l'approccio basato sull'esperienza. (K2)
- LO-5.2.9 Riconoscere/giustificare dei criteri di ingresso e di uscita adeguati per specifici livelli di test e gruppi di 'test cases' (ad esempio, per il testing di integrazione, per il testing di accettazione o per il testing di usabilità). (K2)

5.3 Controllo e monitoraggio nell'avanzamento del testing (K2)

- LO-5.3.1 Ricordare le metriche più comuni usate per il monitoraggio della preparazione ed esecuzione dei test. (K1)
- LO-5.3.2 Spiegare e comparare le metriche per il controllo e la reportistica dei test (ad esempio, difetti trovati e corretti e test passati e falliti). (K2)
- LO-5.3.3 Riassumere lo scopo ed il contenuto del documento di 'test summary report' in accordo con lo 'Standard for Software Test Documentation' (IEEE 829). (K2)

5.4 Gestione della configurazione (K2)

- LO-5.4.1 Riassumere come la gestione della configurazione supporta il testing. (K2)

5.5 Rischio e testing (K2)

- LO-5.5.1 Descrivere un rischio come un possibile problema che potrebbe compromettere il raggiungimento di uno o più obiettivi di progetto delle parti interessate (stakeholders)'. (K2)
- LO-5.5.2 Ricordare che i rischi sono determinati dalla probabilità di un avvenimento e dal suo impatto (danno risultante dal verificarsi di tale avvenimento). (K1)
- LO-5.5.3 Distinguere tra rischio di progetto e rischio di prodotto. (K2)
- LO-5.5.4 Riconoscere i tipici rischi di progetto e di prodotto. (K1)
- LO-5.5.5 Descrivere, con esempi, come l'analisi del rischio e la gestione del rischio possono essere usate nella pianificazione del testing. (K2)

5.6 Gestione degli incidenti (K3)

- LO-5.6.1 Riconoscere il contenuto di una reportistica di incidente in accordo allo 'Standard for Software Test Documentation' (IEEE 829). (K1)

LO-5.6.2 Scrivere una reportistica di incidente riguardante la rilevazione di un esito negativo (failure) durante il testing.
(K3)

| | |
|---|-----------|
| 5.1. ORGANIZZAZIONE DEL TESTING (K2) | 30 minuti |
|---|-----------|

5.1.1. Termini

Tester, test leader, test manager.

5.1.2. Organizzazione ed indipendenza del testing (K2)

L'efficacia nella ricerca di difetti tramite il testing e le reviews può essere migliorata utilizzando dei testers indipendenti. Esistono vari livelli di indipendenza da questo punto di vista:

- Non utilizzo di testers indipendenti. Gli sviluppatori testano il loro codice.
- Utilizzo di testers indipendenti ma all'interno dei teams di sviluppo.
- Utilizzo di teams di test indipendenti o di gruppi all'interno dell'organizzazione, che si relazionino e facciano gli opportuni reports al project management o all' executive management.
- Utilizzo di testers indipendenti dalla organizzazione o della comunità di utenti.
- Utilizzo di specialisti di test indipendenti per specifiche tipologie di test come usability testers, security testers o certification testers (coloro che certificano un prodotto software rispetto a standard e normative).
- Utilizzo di testers indipendenti esterni (attività in outsourcing) all' organizzazione.

Per progetti di grandi dimensioni, di elevata complessità o safety critical, è generalmente preferibile avere livelli multipli di testing, con alcuni o addirittura tutti i livelli gestiti da testers indipendenti. Lo staff di sviluppo può partecipare al testing, soprattutto ai livelli più bassi, ma la loro mancanza di obiettività spesso limita la loro efficacia. Testers indipendenti possono avere l'autorità di richiedere e definire le regole ed i processi di test, ma essi devono assumere tale ruolo solo in presenza di un chiaro mandato del management.

I benefici dell'indipendenza includono:

- Testers indipendenti vedono difetti ulteriori e di tipo diverso, e sono imparziali.
- Un tester indipendente può verificare la validità delle congetture che le persone fanno durante le specifiche e le implementazioni del sistema.

Gli svantaggi includono:

- L' isolamento dal team di sviluppo (se gestiti come completamente indipendenti).
- Testers indipendenti possono essere considerati un collo di bottiglia o criticati per ritardi nei rilasci.
- Gli sviluppatori possono perdere il senso di responsabilità nella produzione di software di qualità.

Le attività di testing possono essere svolte da persone con uno specifico ruolo di tester, oppure possono essere condotte da qualcun'altro ricoprente un altro ruolo, come un project manager, un quality manager, uno sviluppatore, un esperto di dominio applicativo, di infrastruttura ecc.

5.1.3. Compiti del test leader e del tester (K1)

In questo syllabus vengono descritti due ruoli: quello del test leader e quello del tester. Le attività ed i compiti svolti da persone in questi due ruoli dipendono dal contesto di progetto e di prodotto, dalle personalità dei ruoli e dalla organizzazione.

A volte il test leader viene chiamato test manager o test coordinator. Il ruolo del test leader può essere svolto da un project manager, da un development manager, da un quality assurance manager o dal manager di un gruppo di testing. In progetti di grandi dimensioni possono esistere due posizioni: il test leader ed il test manager. Tipicamente il test leader pianifica, monitora e controlla le attività di testing come definite nella Sezione 1.4.

Compiti tipici del test leader possono includere:

- Coordinare la strategia ed il piano di test col project manager ed altri.
- Scrivere o revisionare una strategia di testing per il progetto e la politica di testing per l'organizzazione.
- Contribuire ad una visione coordinata di altre attività di progetto, come la pianificazione dell'integrazione.
- Pianificare i test – considerando il contesto e comprendendo a fondo gli obiettivi del test ed i rischi connessi – che include la scelta degli approcci di testing, la stima dei tempi, dello sforzo e dei costi del testing, l'acquisizione di risorse, la definizione dei livelli di test, dei cicli e la pianificazione della gestione di incidenti.
- Iniziare con le specifiche, proseguire poi con la preparazione, l'implementazione e l'esecuzione dei test, monitorare i risultati dei test ed infine verificare i criteri di uscita.
- Aggiornare la pianificazione in base ai risultati dei test e allo stato di avanzamento (a volte documentato in uno status reports) e prendere ogni iniziativa necessaria a risolvere eventuali problemi.
- Impostare un'adeguata gestione della configurazione del testware per la sua tracciabilità..
- Introdurre metriche adeguate per la misurazione dell'avanzamento del testing e per la valutazione della qualità del testing e del prodotto.
- Decidere che cosa deve essere automatizzato, con quale grado e in che modo.
- Scegliere gli strumenti per supportare il testing ed organizzare adeguati training nell'uso di tali strumenti, per i testers.
- Decidere sull'implementazione dell'ambiente di testing.
- Scrivere i 'test summary reports' basati sulle informazioni raccolte durante il testing.

Compiti tipici del tester possono includere:

- Revisionare e contribuire attivamente alla stesura dei test plans.
- Analizzare, revisionare e valutare i requisiti utente, le specifiche e modelli per la loro testabilità.
- Creare specifiche di test.
- Configurare l'ambiente di test (spesso in coordinamento con l'amministratore di sistema e l'amministratore di rete).
- Preparare ed acquisire dati di test.
- Implementare test su tutti i livelli di test, eseguire e registrare i test, valutarne i risultati e documentare le eventuali deviazioni dai risultati attesi.
- Usare strumenti di gestione, amministrazione e monitoraggio dei test come richiesto.
- Automatizzare i test (col supporto di uno sviluppatore o di un esperto di test automation).
- Misurare le prestazioni dei componenti e dei sistemi (se significative).
- Revisionare i test sviluppati da altri.

Le persone che lavorano all'analisi dei test, al test design, a tipi di test specifici od alla automazione dei test, possono essere degli specialisti in questi ruoli. A seconda del livello di test e dei rischi legati al prodotto e al progetto, persone diverse possono acquisire il ruolo del tester, mantenendo un certo grado di indipendenza.

Tipicamente i testers ai livelli di testing di componente e di integrazione devono essere sviluppatori, i testers a livello di accettazione devono essere esperti del dominio applicativo ed utenti, e i testers per il testing operativo devono essere operatori.

| | |
|---|-----------|
| 5.2. STIMA E PIANIFICAZIONE DEL TESTING (K2) | 40 minuti |
|---|-----------|

5.2.1. Termini

Approccio e strategia di test.

5.2.2. Pianificazione del testing (K2)

Questa sezione si occupa dello scopo della pianificazione del testing in progetti di implementazione e sviluppo e per le attività di manutenzione. La pianificazione può essere documentata in un 'project test plan') o 'master test plan', ed i piani di testing possono essere separati per i vari livelli di test, come il testing di sistema e il testing di accettazione. Il contenuto di un documento di test plan è descritto dallo 'Standard for Software Test Documentation' (IEEE 829).

La pianificazione è influenzata dalla politica di test dell'organizzazione, dall'ambito del testing, dagli obiettivi, dai rischi, dai vincoli, dalla criticità, dalla testabilità e dalla disponibilità di risorse. Quanto più il progetto avanza, tante più informazioni sono disponibili e maggiori dettagli possono essere inclusi nella pianificazione.

La pianificazione del testing è un'attività continuativa e viene condotta lungo tutte le attività e i processi del ciclo di vita del sistema. I riscontri che si hanno dalle attività di testing vengono utilizzati per individuare modifiche nei rischi di progetto in modo tale che la pianificazione possa essere eventualmente aggiornata di conseguenza.

5.2.3. Attività di pianificazione del testing (K2)

Le attività di pianificazione dei test, per un sistema o una sua componente, possono includere:

- Determinazione della portata e dei rischi, e identificazione degli obiettivi del testing.
- Definizione dell' approccio complessivo al testing (la strategia di test), comprendente la definizione dei livelli di test e dei criteri di ingresso ed uscita.
- Integrazione e coordinamento delle attività di testing all'interno delle attività del ciclo di vita del software: acquisizione, fornitura, sviluppo, operatività e manutenzione.
- Presa di decisioni su che cosa deve fare il testing, quali ruoli eseguiranno le attività di testing, come le attività di testing devono essere svolte e come i risultati verranno valutati.
- Determinazione della programmazione delle attività di analisi e di progettazione dei test.
- Determinazione della programmazione delle attività di implementazione, esecuzione e valutazione dei test.
- Assegnazione delle risorse alle differenti attività definite.
- Definizione della quantità, del livello di dettaglio, della struttura e degli schemi relativi alla documentazione dei test.
- Selezione delle metriche per il monitoraggio, controllo della preparazione ed esecuzione dei test, risoluzione dei difetti e rivalutazione dei rischi.
- Determinazione del livello di dettaglio delle 'test procedures' in modo che esse forniscano sufficienti informazioni a supporto della preparazione e dell'esecuzione di test riproducibili.

5.2.4. Criteri di ingresso (K2)

I criteri di ingresso definiscono quando iniziare il testing, come ad esempio all'inizio di un livello di test o quando un insieme di test è pronto per essere eseguito.

Tipicamente i criteri di ingresso possono consistere in:

- Disponibilità ed adeguatezza dell'ambiente di test.
- Disponibilità di strumenti di test nell'ambiente di test.
- Disponibilità del codice da testare.
- Disponibilità dei dati di test.

5.2.5. Criteri di uscita (K2)

I criteri di uscita definiscono quando fermare il testing, come ad esempio alla fine di un livello di test o quando un insieme di test ha raggiunto uno specifico obiettivo.

Tipicamente i criteri di uscita possono consistere in:

- Misure di accuratezza, come copertura del codice, di funzionalità o stime di rischiosità.
- Stime sulla densità dei difetti o sulla affidabilità delle misurazioni.
- Costi.
- Rischi residui, come difetti non modificati o carenze di copertura di codice in certe aree.
- Vincoli temporali come quelli basati sul 'time to market'.

5.2.6. Stima del testing (K2)

In questo syllabus vengono descritti i due seguenti approcci per la stima dello sforzo del testing:

- L'approccio basato sulle metriche: consiste nello stimare lo sforzo del testing sulla base di metriche di progetti simili o sulla base di valori tipici.
- L'approccio basato sull'esperienza: consiste nella stima delle attività da parte del responsabile di esse o da parte di esperti.

Una volta che lo sforzo di testing è stato stimato, le risorse possono essere identificate ed una pianificazione può essere redatta.

Lo sforzo di testing può dipendere da un elevato numero di fattori, tra cui si citano:

- Le caratteristiche del prodotto: la qualità delle specifiche ed altre informazioni usate per i modelli dei test (ad esempio, le basi di test), la dimensione del prodotto, la complessità del dominio del problema, i requisiti di affidabilità e sicurezza ed i requisiti di documentazione.
- Le caratteristiche del processo di sviluppo: la stabilità dell'organizzazione, gli strumenti usati, il processo di testing, le competenze delle persone coinvolte e le tempistiche (spesso pressanti).
- Il risultato del testing: il numero dei difetti e la quantità di rielaborazione richiesta.

5.2.7. Strategie di testing, approcci di testing (K2)

L'approccio al testing è l'implementazione della strategia di testing per uno specifico progetto. L'approccio al testing è definito e perfezionato nei test plans e nei test design. Esso include tipicamente le decisioni prese in base all'obiettivo del testing del progetto ed al risk assessment. Esso è perciò il punto di partenza per pianificare il processo di test, per selezionare le tecniche di test design e le tipologie di test da utilizzare, nonché per definire i criteri di entrata ed uscita.

Un modo per classificare gli approcci di testing, ovvero le strategie di testing, è basato sull'istante nel quale è iniziato il lavoro di progettazione del testing:

- Strategie preventive (approcci preventivi), nelle quali i test sono progettati il prima possibile.

- Strategie reattive (approcci reattivi), nelle quali il test design viene svolto dopo che il software o il sistema è stato prodotto.

L’approccio selezionato dipende dal contesto di riferimento e prende in considerazione i rischi, i vincoli di sicurezza, le risorse e le competenze disponibili, la tecnologia e la natura del sistema (ad esempio, sviluppato per l’utente o COTS), gli obiettivi del testing e normative applicabili.

Tipici approcci includono:

- Strategie analitiche (approcci analitici), come il testing basato sul rischio, nel quale il testing viene diretto verso le aree a maggior rischio.
- Strategie basate su modelli (approcci basati sui modelli), come il testing stocastico che usa informazioni statistiche come la frequenza di esiti negativi (failures) (ad esempio modelli di crescita di affidabilità) o l’utilizzo (ad esempio profili operazionali).
- Strategie metodiche (approcci metodici), come il testing basato su esiti negativi (compresi ‘error guessing’ e ‘fault-attacks’), il testing basato sull’esperienza, le strategie basate su checklists o strategie basate su caratteristiche di qualità.
- Approcci basati sull’aderenza a standard e/o processi, come quelle specificate da standard industriali specifici o varie metodologie di tipo ‘agile’.
- Approcci dinamici ed euristici, come il testing esplorativo nel quale il testing è più reattivo ad eventi che a pianificazioni, ed in cui l’esecuzione e la valutazione sono compiti che vengono svolti in parallelo.
- Approcci consultivi, come quelli in cui la copertura di codice viene primariamente condotta sulla base dei consigli e dalle linee guida di esperti della tecnologia specifica e/o del dominio applicativo, esterni al team di testing.
- Approcci di regressione inversa, come quelli che includono il riuso di materiale di testing esistente, di ampia automazione, di test di regressione funzionali e di test suite standard.

Differenti approcci possono essere poi combinati, per esempio un approccio dinamico basato sul rischio.

| | |
|--|-----------|
| 5.3. CONTROLLO E MONITORAGGIO NELL’AVANZAMENTO DEL TESTING (K2) | 20 minuti |
|--|-----------|

5.3.1. Termini

Densità dei difetti, frequenza degli esiti negativi (failures), controllo del testing, monitoraggio del testing, test report.

5.3.2. Monitoraggio nell’avanzamento del testing (K1)

Lo scopo del monitoraggio del testing è di dare un riscontro e visibilità sulla attività di testing in corso. Le informazioni che devono essere monitorate possono essere raccolte manualmente o automaticamente e possono essere utilizzate per misurare criteri di uscita, come ad esempio la copertura. Le metriche possono essere usate anche per valutare l’avanzamento rispetto alla programmazione ed al budget pianificati. Le più comuni metriche di testing includono:

- Percentuale di lavoro svolto nella preparazione dei test cases (o percentuale dei test cases preparati rispetto a quelli pianificati).
- Percentuale del lavoro svolto nella preparazione dell’ambiente di testing.

- Esecuzione dei test cases (ad esempio, il numero dei test cases eseguiti/non eseguiti e dei test cases passati/falliti).
- Informazioni sui difetti (ad esempio, densità dei difetti, difetti trovati e corretti, frequenza delle failures, e risultati della ripetizione dei test).
- Test coverage dei requisiti, dei rischi o del codice.
- Grado di confidenza soggettivo dei testers sullo stato del prodotto.
- Date delle milestones di testing.
- Costi del testing, compresi i costi confrontati ai benefici derivanti dalla determinazione di ulteriori difetti o dall'esecuzione di ulteriori test.

5.3.3. Reportistica del testing (K2)

L'attività di reportistica del testing è legata alla produzione di informazioni riassuntive sull'attività di testing in corso, in termini di:

- Quello che è avvenuto durante un determinato periodo di testing, come ad esempio le date nelle quali i vari criteri di uscita sono stati soddisfatti.
- Informazioni e metriche analizzate per supportare segnalazioni e decisioni circa le azioni future, come una valutazione dei difetti rimanenti, del beneficio economico derivante dal proseguire l'attività di testing, dei rischi rilevanti e del livello di confidenza raggiunto nel software testato fino a quel momento.

La struttura di un 'test summary report' è dato nello 'Standard for Software Test Documentation' (IEEE 829).

Le metriche devono essere raccolte durante ed alla fine di un livello di test in modo da stimare:

- L'adeguatezza degli obiettivi del testing per quel livello di test.
- L'adeguatezza delle strategie (ovvero degli approcci di test) che sono state intraprese.
- L'efficacia del testing rispetto ai suoi obiettivi.

5.3.4. Controllo del testing (K2)

L'attività di controllo del testing descrive ogni principio guida o azione correttiva adottati come risultato di informazioni e metriche raccolte e per le quali esiste un opportuno report. Le azioni possono coprire ogni attività di testing e possono influenzare ogni attività del ciclo di vita del software.

Esempi di azioni di controllo del testing sono:

- Prendere decisioni basate su informazioni che provengono dal monitoraggio del testing .
- Effettuare una modifica relativa al cambiamento delle priorità dei test quando si verifica un rischio identificato (ad esempio, software da testare consegnato in ritardo).
- Cambiare la programmazione dei test da eseguire in funzione della disponibilità di un ambiente di test.
- Stabilire un criterio di ingresso che richieda che le modifiche debbano essere ri-testate (testing confermativo) da uno sviluppatore prima di accettarle in una build.

| | |
|--|-----------|
| 5.4. GESTIONE DELLA CONFIGURAZIONE (K2) | 10 minuti |
|--|-----------|

5.4.1. Termini

Gestione della configurazione, controllo di versione.

5.4.2. Contesto

Lo scopo della gestione della configurazione è di stabilire e mantenere l'integrità dei prodotti software (componenti, dati e documentazione) o dei sistemi lungo il progetto ed il ciclo di vita del prodotto.

Per il testing, la gestione della configurazione può cercare di assicurare che:

- Tutti gli elementi del testware siano identificati, controllati nella loro versione, tracciati per le modifiche, legati fra di loro e correlati agli elementi di sviluppo (gli oggetti del test) in modo tale che la tracciabilità possa essere mantenuta lungo tutto il processo di testing.
- Tutti i documenti siano identificati univocamente e i correlati elementi software siano referenziati in modo univoco nella documentazione di testing.

La gestione della configurazione aiuta il tester ad identificare univocamente (e a riprodurre) l'oggetto testato, i documenti di testing, i test e la 'test harness'.

Durante la pianificazione del testing, le procedure di gestione della configurazione e la sua infrastruttura (ovvero gli strumenti) devono essere scelti, documentati ed implementati.

| | |
|------------------------------------|-----------|
| 5.5. RISCHIO E TESTING (K2) | 30 minuti |
|------------------------------------|-----------|

5.5.1. Termini

Rischio di prodotto, rischio di progetto, rischio, testing basato sul rischio

5.5.2. Contesto

Il rischio può essere definito come la casualità di un evento, l'imprevisto, la minaccia o il verificarsi di una situazione e le sue conseguenze indesiderate, portatrici di un potenziale problema. Il livello di rischio sarà determinato dalla probabilità del verificarsi di un evento indesiderato e dal suo impatto (in termini del danno risultante dall'occorrenza di tale evento).

5.5.3. Rischi di progetto (K2)

I rischi di progetto sono i rischi che impattano la capacità del progetto di raggiungere i suoi obiettivi, come ad esempio:

- Fattori organizzativi:
 - carenza di personale, di competenze e di formazione;
 - questioni personali;
 - questioni "politiche", come ad esempio
 - problemi con i testers nella comunicazione delle proprie esigenze e dei risultati dei test;
 - incapacità del team di test nel dar seguito alle informazioni raccolte dal testing o emerse dalle reviews (ad esempio non migliorando lo sviluppo e le modalità di testing).
 - errate aspettative nei confronti del testing (ad esempio, non apprezzando il valore aggiunto dei difetti trovati durante il testing).
- Questioni tecniche:
 - problemi nella definizione di requisiti corretti;
 - fino a che punto i requisiti possono essere soddisfatti con i vincoli esistenti;
 - la qualità dei documenti di progettazione, del codice e dei test.

- Questioni di fornitura:
 - fallimento di una terza parte;
 - aspetti contrattuali.

Il test manager, quando analizza, gestisce e cerca di mitigare questi rischi, deve seguire dei ben impostati principi di gestione del progetto. Lo 'Standard for Software Test Documentation' (IEEE 829) evidenzia che nei test plans i rischi (e la gestione delle loro contingenze) siano identificati e definiti il più accuratamente possibile.

5.5.4. Rischi di prodotto (K2)

Aree di 'failures' potenziali (imprevisti o eventi futuri indesiderati) nel software o nel sistema sono denominati rischi di prodotto, dato che essi mettono a rischio la qualità del prodotto, come ad esempio:

- Software consegnato e tendente a manifestare 'failures'.
- La probabilità, che software/hardware possano causare danni a persone o all'azienda.
- Software carente di caratteristiche di qualità (ad esempio, funzionalità, affidabilità, usabilità, prestazioni).
- Carenza di qualità o di integrità dei dati (problemi di migrazione, di conversione e di trasmissione dati, violazione di standard dati).
- Software che non svolge le funzioni per le quali è stato scritto.

I rischi sono utilizzati per decidere dove far partire il testing e dove renderlo più intenso ed efficace; il testing è usato per ridurre il rischio dell'occorrenza di un effetto indesiderato e per ridurre l'impatto.

I rischi di prodotto sono un particolare tipo di rischio al successo di un progetto. Il testing come attività di controllo di tali rischi fornisce dei riscontri sui rischi residui, andando a misurare l'efficacia della rimozione dei difetti critici e dei piani di contingenza.

Un approccio al testing basato sul rischio fornisce delle fattive e concrete opportunità per ridurre i livelli del rischio di prodotto, partendo dagli stadi iniziali di un progetto. Esso coinvolge l'identificazione dei rischi di prodotto e del loro utilizzo come principi guida nelle fasi di pianificazione e controllo, specifica, preparazione ed esecuzione dei test. In un approccio basato sul rischio i rischi identificati possono essere usati per:

- Determinare le tecniche di test da adottare.
- Determinare l'entità del testing che sarà effettuato.
- Stabilire le priorità del testing nel tentativo di trovare i difetti critici il prima possibile.
- Determinare se qualche altra attività non di testing possa essere adottata per ridurre il rischio (ad esempio, fornire della formazione mirata a progettisti inesperti).

Il testing basato sul rischio trae spunto dalla conoscenza collettiva e da una analisi approfondita da parte degli stakeholders del progetto, per determinarne i rischi ed i livelli di testing richiesti per indirizzarne la soluzione.

Per assicurare che la probabilità di una failure di prodotto sia minimizzata, le attività di gestione del rischio forniscono una strategia disciplinata a:

- Stimare (e a ri-stimare su base regolare) quello che può andare male (rischio).
- Determinare quali sono i rischi veri, ovvero quelli più importanti con cui si ha a che fare.
- Mettere in atto delle azioni per indirizzare questi rischi.

In aggiunta, il testing può supportare l'identificazione di nuovi rischi, può aiutare a determinare quali rischi devono essere ridotti e può diminuire complessivamente l'incertezza sui rischi.

| | |
|---|-----------|
| 5.6. GESTIONE DEGLI INCIDENTI (K3) | 40 minuti |
|---|-----------|

Termini

Registrazione degli incidenti, gestione degli incidenti

Contesto

Dato che uno degli obiettivi del testing è di trovare difetti, le discrepanze tra i risultati attuali e quelli attesi necessitano di essere registrati come incidenti. Un incidente deve essere analizzato e può tramutarsi in difetto. Devono essere definite appropriate azioni per gestire gli incidenti e i difetti. Gli incidenti devono essere tracciati dalla scoperta e classificazione fino alla correzione e conferma della loro soluzione. Un'organizzazione deve definire un processo di gestione degli incidenti e delle regole per la loro classificazione.

Gli incidenti possono emergere durante lo sviluppo, le review, il testing o l'utilizzo di un prodotto software. Essi possono presentarsi per problemi nel codice o nel funzionamento del sistema o in ogni tipo di documentazione, compresa quella sui requisiti, documenti di sviluppo, documenti di testing, e informazioni per l'utente come l' "Help" o le guide di installazione.

Le registrazioni degli incidenti hanno i seguenti obiettivi:

- Fornire agli sviluppatori e ad altri ruoli dei riscontri sui problemi per facilitarne l'identificazione, l'isolamento e le correzioni necessarie.
- Fornire ai responsabili di test uno strumento di tracciamento per valutare la qualità del sistema sotto test e l'avanzamento del testing.
- Fornire idee per il miglioramento del processo di testing.

I dettagli nei report degli incidenti possono includere:

- Date del problema, organizzazione e autore.
- Risultati attuali ed attesi.
- Identificazione dell' elemento di test (elemento sotto configurazione) e dell'ambiente di testing.
- Processo del ciclo di vita del software o del sistema nel quale è stata riportato l'incidente.
- Descrizione dell'incidente per consentirne la riproduzione e la risoluzione, comprendente logs, dumps di database o videate.
- Portata o grado dell'impatto sugli interessi degli stakeholders.
- Severità dell'impatto sul sistema.
- Urgenza/priorità di risoluzione.
- Stato dell' incidente (ad esempio, aperto, differito, duplicato, in attesa di essere risolto, risolto in attesa di essere ritestato, chiuso).
- Conclusioni, raccomandazioni e approvazioni.
- Problemi globali, come altre aree che possano essere affette da una modifica risultante dall' incidente.
- Storico delle modifiche (change history), come la sequenza delle azioni prese dai membri del team di progetto con riferimento all'isolamento dell' incidente, della sua riparazione e della conferma della sua risoluzione.
- Riferimenti, compresi l'identificazione del test case specifico che ha rilevato il problema.

La struttura di report di incidente è descritta anche nello 'Standard for Software Test Documentation' (IEEE 829).

RIFERIMENTI

5.1.1 Black, 2001, Hetzel, 1988

5.1.2 Black, 2001, Hetzel, 1988

5.2.5 Black, 2001, Craig, 2002, IEEE 829, Kaner 2002

5.3.3 Black, 2001, Craig, 2002, Hetzel, 1988, IEEE 829

5.4 Craig, 2002

5.5.2 Black, 2001, IEEE 829

5.6 Black, 2001, IEEE 829

| | |
|---|------------------|
| 6. SUPPORTO DI STRUMENTI PER IL TESTING (K2) | 80 minuti |
|---|------------------|

OBIETTIVI DI APPRENDIMENTO PER IL SUPPORTO DI STRUMENTI PER IL TESTING

Gli obiettivi identificano quello che si dovrà essere in grado di fare seguendo il completamento di ogni singolo modulo.

6.1 Tipi di strumenti di testing (K2)

LO-6.1.1 Classificare i differenti tipi di strumenti di testing rispetto alle attività del processo di testing. (K2)

LO-6.1.2 Spiegare il contenuto e l’obiettivo degli strumenti a supporto del testing. (K1)

6.2 Uso efficace degli strumenti: benefici e rischi potenziali (K2)

LO-6.2.1 Riassumere i potenziali benefici e rischi della automazione del testing e degli strumenti di supporto del testing. (K2)

LO-6.2.2 Fornire utili considerazioni sull’utilizzo degli strumenti di esecuzione dei test, di analisi statica e di gestione del testing. (K1)

6.3 Introduzione di uno strumento all’interno di una organizzazione (K1)

LO-6.3.1 Definire i principi fondamentali per l’introduzione di uno strumento all’interno di una organizzazione. (K1)

LO-6.3.2 Affermare gli obiettivi di una ‘proof-of-concept’ per la valutazione di strumenti e di una fase pilota per il loro utilizzo. (K1)

LO-6.3.3 Riconoscere che per un buon supporto dello strumento sono richiesti altri fattori rispetto al semplice acquisto dello strumento. (K1)

| | |
|---|------------------|
| 6.1. TIPI DI STRUMENTI DI TESTING (K2) | 45 minuti |
|---|------------------|

6.1.1. Termini

Strumenti di gestione della configurazione, Strumenti di copertura, Strumenti di debugging, Strumenti di analisi dinamica, Strumenti di gestione degli incidenti, Strumenti di testing di carico, Strumenti di modellazione, Strumenti di monitoraggio, Strumenti di testing di prestazioni, effetto sonda, Strumenti di gestione dei requisiti, Strumenti di revisione, Strumenti di sicurezza, Strumenti di analisi statica, strumenti di stress testing, comparatori di test, Strumenti di preparazione dei dati di test, Strumenti di test design, test harness, Strumenti di esecuzione di test, Strumenti di gestione di test, framework per unit test.

6.1.2. Capire il significato e lo scopo di uno strumento di aiuto al testing (K2)

Gli strumenti di testing possono essere utilizzati per una o più attività in supporto al testing. Essi comprendono:

1. Strumenti che sono direttamente utilizzati nel testing, come strumenti di esecuzione dei test, di prove generazione di dati di test, di confronto dei risultati dei test
2. Strumenti che supportano la gestione del processo di test, come quelli utilizzati per gestire i direttamente il test, i suoi dati, i requisiti, gli incidenti, i difetti, ecc, e per il reporting ed il monitoraggio della loro esecuzione

3. Gli strumenti che vengono utilizzati nella ricognizione o, in termini più semplici nell'esplorazione (ad esempio, strumenti che monitorizzano le attività di un file per un'applicazione)
4. 4. Qualsiasi altro strumento che supporti il testing (anche un foglio di calcolo è in questo senso uno strumento di testing).

Uno strumento di supporto al testing può avere uno o più dei seguenti obiettivi a seconda del contesto:

- Migliorare l'efficienza delle attività di test, automatizzando le attività ripetitive o supportando le attività manuali, come la pianificazione, la progettazione, il monitorino ed il reporting
- Automatizzare le attività che richiedono un notevole impegno di risorse quando svolte manualmente (ad esempio il testing statico)
- Automatizzare attività di testing che non possono essere eseguite manualmente (ad esempio il testing di prestazioni, su vasta scala di applicazioni client-server)
- Aumentare l'affidabilità del testing (ad esempio automatizzando il confronto e la simulazione del comportamento di banche dati di grandi dimensioni)

Il termine "test framework" è spesso utilizzato nell'IT, con almeno tre significati:

- Librerie di testing riutilizzabili ed estensibili che possono essere utilizzate per costruire strumenti di test (chiamate test harness)
- Un tipo di progettazione di automazione del testing (ad esempio, data-driven, keyword-driven)
- Il processo globale di esecuzione dei test

Ai fini del presente syllabus, il termine "test framework" è utilizzato nei primi due significati, come descritto nella Sezione 6.1.6.

6.1.3. Classificazione degli strumenti di testing (K2)

Ci sono un certo numero di strumenti che supportano differenti aspetti del testing. Gli strumenti possono essere classificati in base a diversi criteri, come obiettivo, commerciali/gratuiti/open-source/shareware, tecnologia ecc. In questo syllabus gli strumenti sono classificati in base alle attività di testing che essi supportano.

Alcuni strumenti supportano chiaramente una sola attività, mentre altri possono supportare più di una attività, ma sono classificati sotto l'attività con la quale sono più strettamente collegati. Gli strumenti di un singolo fornitore, specialmente quando sono stati disegnati per operare insieme, possono essere integrati in un singolo pacchetto.

Alcuni tipi di strumenti di testing possono essere intrusivi, il che significa che possono modificare il risultato effettivo del test. Per esempio, il tempo di esecuzione può essere diverso a causa di istruzioni extra eseguite dallo strumento o si possono ottenere differenti misurazioni di copertura del codice. Le conseguenze degli strumenti intrusivi vengono spesso identificate come 'effetto sonda'.

Alcuni strumenti offrono un supporto più adatto di altri per gli sviluppatori (ad esempio, durante il testing di componenti ed il testing di integrazione). Questi strumenti sono marcati con una "(D)" nella seguente classificazione.

6.1.4. Strumenti di supporto per la gestione del testing e dei test (K1)

Gli strumenti di gestione si applicano a tutte le attività di testing lungo tutto l'intero ciclo di vita del software.

Strumenti di gestione test

Questi strumenti forniscono delle interfacce per l'esecuzione dei test, la registrazione dei difetti, la gestione dei requisiti, in aggiunta al supporto per analisi quantitative ed il reporting degli oggetti del test.

Essi supportano inoltre la tracciabilità. degli oggetti del test verso le specifiche dei requisiti e possono disporre di un'indipendente capacità di controllo delle versioni o un'interfaccia verso funzionalità esterne.

Strumenti di gestione dei requisiti

Questi strumenti memorizzano i requisiti ed i loro attributi (incluse le priorità), forniscono classificazioni univoche e la tracciabilità. dei requisiti verso i singoli test. Essi inoltre possono aiutare nell'identificazione di requisiti indefiniti o mancanti

Strumenti di gestione degli incidenti

Questi strumenti memorizzano e gestiscono i reports degli incidenti (esiti negativi (failures), difetti, problemi percepiti, anomalie, change requests ecc.) e supportano la gestione del ciclo di vita degli incidenti, in alcuni casi col supporto per analisi statistiche.

Strumenti di gestione della configurazione

Benché a rigore non siano degli strumenti di testing, essi sono necessari per la registrazione ed il versioning del testare e del software connesso, specialmente in presenza di diversi ambienti hardware/software (sistemi operativi, compilatori, browsers ecc.).

6.1.5. Strumenti di supporto per il testing statico (K1)

I strumenti di testing statico forniscono una modalità efficace per trovare un maggior numero di difetti in una fase preliminare del processo di sviluppo.

Strumenti di revisione

Questi strumenti forniscono supporto al processo di review con checklists, linee guida ecc. e sono principalmente utilizzati per memorizzare e comunicare i commenti delle review, memorizzare report sui difetti e sullo sforzo. Essi possono anche fornire supporto per delle review online, che sono utili quando il team è geograficamente sparso.

Strumenti di analisi statica (D)

Gli strumenti di analisi statica supportano sviluppatori e testers nella ricerca di difetti prima del testing dinamico, tramite il controllo sull'applicazione degli standard di codifica, l'analisi delle strutture e delle dipendenze (ad esempio, collegamenti di pagine web).

Essi possono anche fornire aiuto nella pianificazione o nell'analisi del rischio, tramite metriche applicate al codice (ad esempio, la complessità).

Strumenti di modellazione (D)

Gli strumenti di modellazione sono utilizzati per validare i modelli del software (per esempio i modelli dati per database relazionali) evidenziando difetti e inconsistenze. Questi strumenti possono spesso aiutare nella generazione di alcuni test cases basati su modelli.

6.1.6. Strumenti di supporto per la specifica dei test (K1)

Strumenti di test design

Questi strumenti sono usati per generare dei test inputs o test eseguibili od oracoli di test a partire dai requisiti, da una GUI (Graphical Use Interface), da modelli di progettazione (a stati, basati sui dati, o ad oggetti) o dal codice.

Strumenti di preparazione dei dati di test

Gli strumenti di preparazione dei dati di test manipolano databases, files o trasmissioni di dati per configurare i dati di test per essere usati durante l'esecuzione dei test, allo scopo di garantire la privacy tramite l'anonimità dei dati

6.1.7. Strumenti di supporto per l'esecuzione ed il logging dei test (K1)

Strumenti di esecuzione di test

Questi strumenti consentono ai test di essere eseguiti automaticamente, o semi-automaticamente, (usando inputs e risultati attesi precedentemente memorizzati) attraverso un apposito linguaggio di scripting.

Essi forniscono di solito un log per ogni test eseguito.

Possono essere anche usati per registrare i test e di solito forniscono un linguaggio di scripting o schermate GUI per la parametrizzazione dei dati ed altre personalizzazioni.

Tool di test harness/ strumenti o framework per unit test (D)

Un insieme di test di unità facilita il collaudo di componenti o parti di un sistema simulando l'ambiente in cui tale oggetto di test verrà eseguito

Una test harness può facilitare il testing dei componenti o di parte di un sistema simulando l'ambiente nel quale tale oggetto di test verrà eseguito, attraverso la fornitura di oggetti fittizi come stub o driver.

Comparatori di test

I comparatori di test determinano differenze tra files, databases o risultati di test. Gli strumenti di esecuzione di test includono normalmente comparatori dinamici, ma il confronto dopo l'esecuzione può essere fatto da uno strumento di confronto separato. Un comparatore di test può utilizzare un oracolo del test, specialmente se esso è automatizzato.

Strumenti di copertura (D)

Questi strumenti, con modalità intrusive o non intrusive, misurano le percentuali di copertura di specifici tipi di strutture di codice (ad esempio, istruzioni, rami o decisioni, moduli o funzioni) che sono state attivate da un insieme di test.

Strumenti di sicurezza

Gli strumenti di testing sono utilizzati per valutare le caratteristiche di sicurezza del software, includendo in ciò la capacità del software di proteggere dati confidenziali, l'integrità, l'autenticazione, l'abilitazione, la disponibilità ed il non-ripudio. Gli strumenti di sicurezza sono specializzati per specifiche piattaforme tecnologiche e/o per predefiniti obiettivi

6.1.8. Strumenti di supporto per prestazioni e monitoraggio (K1)

Strumenti di analisi dinamica (D)

Gli strumenti di analisi dinamica trovano difetti che si manifestano solo quando il software è in esecuzione, come ad esempio dipendenze temporali o falle di memoria ('memory leaks'). Essi sono tipicamente usati nel testing di componenti o nel testing di integrazione, e quando si testano dei middleware.

Strumenti di testing di prestazioni/di testing di carico/di testing di stress

Gli strumenti di testing di prestazioni monitorizzano e riportano il comportamento di un sistema sottoposto a varie condizioni simulate di utilizzo, in termini di numero di utenti contemporanei, di distribuzione temporale di accesso, di frequenza e percentuali relative delle transazioni.

Strumenti di monitoraggio

Gli strumenti di monitoraggio non sono strettamente degli strumenti di testing ma forniscono informazioni che possono essere utilizzate per scopi di testing che non sono funzionali ad altri scopi.

Gli strumenti di monitoraggio analizzano continuamente, verificano e riportano l'utilizzo di specifiche risorse di sistema, e sollevano degli allarmi su possibili problemi di servizio. Essi memorizzano informazioni sulla versione e sulla build del software e del testware, e ne consentono la tracciabilità.

Gli strumenti di monitoraggio analizzano, verificano e riportano con continuità l'utilizzo di specifiche risorse di sistema e segnalano possibili problemi di servizio.

6.1.9. Strumenti di supporto per specifiche esigenze di testing (K1)

Valutazione qualità dati

I dati sono al centro di alcuni progetti, quali la conversione di dati / migrazioni ed applicazioni data warehouse e gli attributi di tali dati possono variare in termini di criticità e di volume. In tali contesti, occorre utilizzare strumenti per la valutazione della qualità dei dati, orientati a verificare e validare le regole di migrazione e conversione dati, per assicurare che i dati trattati siano corretti, completi e conformi a standard pre-definiti e specifici del contesto.

Esistono poi altri strumenti per i test di usabilità .

| | |
|--|------------------|
| <p>6.2. USO EFFICACE DEGLI STRUMENTI: BENEFICI E RISCHI POTENZIALI (K2)</p> | <p>20 minuti</p> |
|--|------------------|

Termini

Data-driven (testing), keyword-driven (testing), linguaggio di scripting

6.2.1. Potenziali benefici e rischi di strumenti di supporto per il testing (K2)

Il semplice acquisto o affitto di uno strumento non garantisce di raggiungere con semplicità gli obiettivi che ci si propone di raggiungere con tale strumento. Ogni tipo di strumento può richiedere uno sforzo aggiuntivo per raggiungere i reali e duraturi benefici attesi. Ci sono sì opportunità e benefici potenziali con l'uso di strumenti di testing, ma ci sono anche dei rischi.

I benefici potenziali nell'uso di strumenti di testing includono:

- Il lavoro ripetitivo viene ridotto (ad esempio, l'esecuzione di test di regressione, il reinserimento degli stessi dati di test ed il controllo del rispetto di standard di codifica).
- Una maggiore consistenza e ripetibilità (ad esempio, i test eseguiti da uno strumento ed i test derivati dai requisiti).
- Una valutazione oggettiva (ad esempio, misure statiche, copertura).
- Una maggiore facilità di accesso alle informazioni relative ai test o al testing (ad esempio, statistiche e grafici sull'avanzamento del testing, prestazioni e frequenza degli incidenti).

I rischi potenziali nell'uso di strumenti di testing includono:

- Aspettative non realistiche relative alle caratteristiche dello strumento (comprese funzionalità e facilità d'uso).
- Sottostime del tempo, del costo e del lavoro per l'introduzione iniziale di uno strumento (comprendente la formazione e l'esperienza esterna).
- Sottostime del tempo e del lavoro necessari per ottenere benefici significativi e continuativi dallo strumento (comprendenti le necessità di cambiamenti nel processo di testing e continui miglioramenti del modo con cui lo strumento viene usato).
- Sottostima del lavoro richiesto per mantenere le configurazioni e gli outputs generati dallo strumento.
- Fiducia eccessiva nello strumento (sostituzione del test design e utilizzo di test automatici laddove il testing manuale sarebbe più efficace e/o efficiente).
- Trascurare il controllo di versione del materiale di test all'interno dello strumento.
- Trascurare le relazioni ed interdipendenze operative fra gli strumenti, come gli strumenti di gestione dei requisiti, di controllo di versione, di gestione degli incidenti, di gestione dei difetti, nonché le problematiche di strumenti di fornitori diversi.
- Rischio che il fornitore dello strumento vada fuori mercato, ritiri lo strumento o lo venda ad un altro fornitore.
- Carente supporto del fornitore nel supporto, nel rilascio di miglioramenti e nella correzione errori.
- Rischi di sospensione di strumenti open-source o gratuiti.
- Imprevisti, come la mancanza di supporto di nuove piattaforme.

6.2.2. Considerazioni speciali per alcuni tipi di strumenti (K1)

Strumenti di esecuzione di test

Gli strumenti di esecuzione di test eseguono scripts progettati per implementare test che sono memorizzati in formato elettronico. Questo tipo di strumenti spesso richiedono un lavoro elevato per raggiungere dei benefici significativi.

Catturare i test (e i relativi outputs) registrando le azioni di un tester manuale può sembrare comodo, ma questo approccio non si applica al caso di un elevato numero di script di test automatici. Uno script catturato è una rappresentazione lineare, con dati ed azioni specifiche facenti parte dello script. Questo tipo di script può essere instabile quando si verificano eventi inattesi.

Un approccio data-driven, separa gli input dei test (i dati), registrandoli generalmente in un foglio elettronico, e usa uno script più generico che può leggere i dati di test ed eseguire lo stesso test con dati differenti. I testers che non hanno consuetudine con i linguaggi di scripting, possono inserire i dati di test in questi scripts predefiniti.

Ci sono altre tecniche data-driven che vengono impiegate, in cui invece di utilizzare combinazioni di dati registrate in fogli elettronici, si generano dati in base a specifici algoritmi, con parametri configurabili al momento dell'esecuzione, che vengono forniti all'applicazione da testare. Per esempio uno strumento può utilizzare un algoritmo per generare delle user.id casuali.

In un approccio keyword-driven, il foglio elettronico contiene delle keywords, che descrivono le azioni da intraprendere (chiamate anche 'action words'), e dati di test. I testers (anche coloro che non hanno familiarità con linguaggi di scripting) possono quindi definire i test usando le keywords, le quali possono essere fatte su misura per l'applicazione che deve essere testata.

Un'esperienza tecnica dei linguaggi di scripting è necessaria per tutti gli approcci (sia dai testers che dagli specialisti in test automation).

Qualunque sia la tecnica di scripting usata, i risultati attesi per ogni test necessitano di essere memorizzati per successivi confronti.

Strumenti di analisi statica

Gli strumenti di analisi statica applicati al codice sorgente possono attuare verifiche sul rispetto di standard di codifica, ma se applicati a codice esistente possono generare molti messaggi. I messaggi di 'warning' non impediscono che il codice venga compilato in un programma eseguibile, ma devono essere comunque presi in considerazione in modo che anche la manutenzione del codice futura sia più semplice. Una implementazione graduale dello strumento di analisi statica, con filtri iniziali per escludere alcuni messaggi, potrebbe essere un approccio efficace.

Strumenti di gestione di test

Strumenti di gestione di test necessitano di interfacciarsi con altri strumenti o con fogli elettronici in modo da produrre informazioni nel formato migliore per le necessità correnti dell'organizzazione.

| | |
|--|------------------|
| <p>6.3. INTRODUZIONE DI UNO STRUMENTO ALL'INTERNO DI UN'ORGANIZZAZIONE (K1)</p> | <p>15 minuti</p> |
|--|------------------|

6.3.1. Termini

Nessun termine specifico.

6.3.2. Contesto

- Le principali considerazioni nella scelta di uno strumento da parte di una organizzazione includono:
- o Valutazione della maturità organizzativa, punti di forza e di debolezza ed identificazione delle opportunità per migliorare il processo di testing supportandolo con opportuni strumenti.
 - o Valutazione basata su requisiti chiari e criteri oggettivi.
 - o Una proof-of-concept, utilizzando lo strumento in una fase valutativa per verificare se esso opera con efficacia col software da testare e con l'infrastruttura disponibile o per identificare quali modifiche infrastrutturali siano necessarie.
 - o Valutazione del fornitore (comprendente formazione, supporto, e aspetti commerciali) o di fornitori di servizi di assistenza in caso di strumenti non commerciali..
 - o Identificazione di requisiti interni per l'istruzione e l'addestramento nell'utilizzo dello strumento.
 - o Valutazione dell'addestramento necessario considerando l'attuale livello di competenza in automazione da parte dei membri del test team.
 - o Stima del rapporto costi/benefici basato su un valido business-case.

L'introduzione dello strumento selezionato all'interno dell'organizzazione parte con un progetto pilota, il quale ha i seguenti obiettivi:

- o Imparare maggiore dettagli sullo strumento.
- o Valutare come lo strumento si adatti ai processi e alle abitudini esistenti, e determinare cosa sarebbe necessario modificare.

- Definire le modalità standard di utilizzo, gestione, memorizzazione e manutenzione dello strumento e gli impatti sul test (ad esempio, decisioni sulle convenzioni dei nomi per il files e i test, sulla creazione delle librerie e sulla definizione della modularità delle test suites).
- Valutare se i benefici saranno raggiunti ad un costo ragionevole.

I fattori di successo per la diffusione dello strumento all'interno di un'organizzazione includono:

- Una diffusione incrementale dello strumento alle varie parti dell'organizzazione.
- L'adattamento e il miglioramento dei processi in modo tale che essi si adattino bene all'utilizzo dello strumento.
- La fornitura di una formazione opportuna in termini di istruzione/addestramento per i nuovi utenti.
- La definizione di linee guida per l'utilizzo dello strumento.
- La definizione modalità per raccogliere utili informazioni dall'utilizzo dello strumento.
- Un monitoraggio sull'utilizzo dello strumento e sugli eventuali benefici che esso fornisce.
- La fornitura di un'assistenza sullo strumento al team di test.
- La raccolta di lesson-learned da tutti i team.

RIFERIMENTI

6.2.2 Buwalda, 2001, Fewster, 1999

6.3 Fewster, 1999

RIFERIMENTI

STANDARD

ISTQB® Glossary of terms used in Software Testing Version 1.0

[CMMI] Chrissis, M.B., Konrad, M. and Shrum, S. (2004) *CMMI, Guidelines for Process Integration and Product Improvement*, Addison Wesley: Reading, MA

Vedi Sezione 2.1

[IEEE 829] IEEE Std 829™ (1998/2007) IEEE Standard for Software Test Documentation (currently under revision)

Vedi Sezioni 2.3, 2.4, 4.1, 5.2, 5.3, 5.5, 5.6

[IEEE 1028] IEEE Std 1028™ (1997) IEEE Standard for Software Reviews

Vedi Sezione 3.2

[IEEE 12207] IEEE 12207/ISO/IEC 12207-1996, Software life cycle processes

Vedi Sezione 2.1

[ISO 9126] ISO/IEC 9126-1:2001, Software Engineering – Software Product Quality

Vedi Sezione 2.3

LIBRI

[Beizer, 1990] Beizer, B. (1990) *Software Testing Techniques* (2nd edition), Van Nostrand Reinhold: Boston

Vedi Sezioni 1.2, 1.3, 2.3, 4.2, 4.3, 4.4, 4.6

[Black, 2001] Black, R. (2001) *Managing the Testing Process* (2nd edition), John Wiley & Sons: New York

Vedi Sezioni 1.1, 1.2, 1.4, 1.5, 2.3, 2.4, 5.1, 5.2, 5.3, 5.5, 5.6

[Buwalda, 2001] Buwalda, H. et al. (2001) *Integrated Test Design and Automation*, Addison Wesley: Reading, MA

Vedi Sezione 6.2

[Copeland, 2004] Copeland, L. (2004) *A Practitioner's Guide to Software Test Design*, Artech House: Norwood, MA

Vedi Sezioni 2.2, 2.3, 4.2, 4.3, 4.4, 4.6

[Craig, 2002] Craig, Rick D. and Jaskiel, Stefan P. (2002) *Systematic Software Testing*, Artech House: Norwood, MA

Vedi Sezioni 1.4.5, 2.1.3, 2.4, 4.1, 5.2.5, 5.3, 5.4

[Fewster, 1999] Fewster, M. and Graham, D. (1999) *Software Test Automation*, Addison Wesley:

Reading, MA

Vedi Sezioni 6.2, 6.3

[Gilb, 1993]: Gilb, Tom and Graham, Dorothy (1993) *Software Inspection*, Addison Wesley: Reading, MA

Vedi Sezioni 3.2.2, 3.2.4

[Hetzel, 1988] Hetzel, W. (1988) *Complete Guide to Software Testing*, QED: Wellesley, MA

Vedi Sezioni 1.3, 1.4, 1.5, 2.1, 2.2, 2.3, 2.4, 4.1, 5.1, 5.3

[Kaner, 2002] Kaner, C., Bach, J. and Pettitcord, B. (2002) *Lessons Learned in Software Testing*, John Wiley & Sons: New York

Vedi Sezioni 1.1, 4.5, 5.2

[Myers 1979] Myers, Glenford J. (1979) *The Art of Software Testing*, John Wiley & Sons: New York

Vedi Sezioni 1.2, 1.3, 2.2, 4.3

[van Veenendaal, 2004] van Veenendaal, E. (ed.) (2004) *The Testing Practitioner* (Chapters 6, 8, 10), UTN Publishers: The Netherlands

Vedi Sezioni 3.2, 3.3

APPENDICE A – INFORMAZIONI RELATIVE AL SYLLABUS

STORIA DI QUESTO DOCUMENTO

Questo documento è stato preparato tra il 2004 ed il 2007 da un gruppo di lavoro comprendente membri nominati dal 'International Software Testing Qualifications Board (ISTQB®)'. Esso è stato inizialmente revisionato da un comitato revisore selezionato, e successivamente da rappresentative delle comunità internazionale di software testing.

Questo documento è il syllabus per il 'International Foundation Certificate' nel Software Testing, il primo livello internazionale di qualificazione internazionale approvato da ISTQB® (www.ISTQB.org).

OBIETTIVI DEL CERTIFICATO DI QUALIFICA 'FOUNDATION'

- Ottenere il riconoscimento del testing come una specializzazione essenziale e professionale nell'ambito dell'ingegneria del software.
- Fornire una struttura di riferimento standard per lo sviluppo delle carriere dei testers.
- Consentire ai testers qualificati professionalmente di essere riconosciuti dai datori di lavoro in ambito IT, dai clienti e dai colleghi, e di fare crescere il profilo professionale dei testers.
- Promuovere delle buone pratiche consistenti e sempre più consolidate in tutte le discipline dell'ingegneria del software.
- Identificare gli argomenti di testing che siano rilevanti, applicabili e di reale valore aggiunto per l'industria.
- Consentire alle aziende di assumere testers certificati e di conseguenza ottenere un vantaggio commerciale sui loro concorrenti pubblicizzando la loro politica di reclutamento di tester.
- Fornire un'opportunità per i testers e per coloro con un interesse nel testing di acquisire una qualifica riconosciuta a livello internazionale.

OBIETTIVI DELLA QUALIFICA INTERNAZIONALE

- Essere in grado di confrontare le capacità di testing tra differenti paesi.
- Consentire ai testers di muoversi più facilmente attraverso i confini dei vari paesi.
- Consentire a progetti multinazionali/internazionali di avere una comprensione comune delle questioni legate al testing.
- Aumentare il numero di testers qualificati a livello mondiale globale.
- Avere un maggiore impatto/valore come iniziativa basata su una collaborazione internazionale e con una vasta diffusione capillare rispetto all'approccio specifico di ogni singolo paese.
- Sviluppare un ente internazionale comune di comprensione e conoscenza del testing attraverso i syllabus e la terminologia, e aumentare il livello di conoscenza sul testing di tutti i partecipanti.
- Promuovere il testing come una professione in più paesi.
- Consentire ai testers di conseguire una qualifica riconosciuta nella sua lingua madre.
- Consentire la condivisione della conoscenza e delle risorse tra i vari paesi.
- Fornire un riconoscimento internazionale dei testers e di questa qualifica, dovuto alla partecipazione da parte di molti paesi.

REQUISITI DI INGRESSO PER LA QUALIFICA ISTQB®

Il criterio di ingresso per partecipare all'esame per il conseguimento del certificato ISTQB® per il Software Testing a livello 'Foundation' è che il candidato abbia interesse nel software testing. Comunque, si raccomanda anche che il candidato:

- Abbia almeno un minimo di preparazione o nello sviluppo di software o nel software testing, come, ad esempio, sei mesi di esperienza come tester al livello di sistema o a livello di accettazione utente o come sviluppatore software.
- Frequenti un corso che sia stato accreditato come ISTQB® standard (da parte di una delle 'boards' nazionali riconosciute da ISTQB®).

CONTESTO E STORIA DEL CERTIFICATO 'FOUNDATION' NEL SOFTWARE TESTING

La certificazione indipendente dei software testers iniziò nel Regno Unito con la 'British Computer Society's Information Systems Examination Board (ISEB)', quando un 'Software Testing Board' fu fondata nel 1998 (www.bcs.org.uk/iseb). Nel 2002, ASQF in Germania iniziò a supportare uno schema di qualificazione per i testers tedeschi. Questo syllabus è basato sui syllabi ISEB e ASQF; rispetto a questi, esso include una riorganizzazione, una modifica e l'aggiunta di nuovi contenuti, e l'enfasi è orientata verso argomenti che forniscano un aiuto più pratico ai testers.

Un certificato a livello 'Foundation' nel Software Testing (ad esempio, ISEB, ASQF o riconosciuto da una 'board' nazionale di ISTQB®) conseguito prima che questo Certificato Internazionale sia stato rilasciato, sarà ritenuto equivalente al Certificato Internazionale. Il Certificato 'Foundation' non scade e non necessita di essere rinnovato. La data nella quale esso è stato conseguito è indicata sul certificato stesso.

In ogni singolo paese partecipante, gli aspetti locali sono controllati da un 'Software Testing Board' nazionale riconosciuto da ISTQB®. I doveri e le responsabilità dei 'boards' nazionali sono specificate da ISTQB®, e sono implementati in ogni singolo paese. Tra questi doveri sono inclusi l'accreditamento di fornitori della formazione e la preparazione degli esami.

APPENDICE B – OBIETTIVI DI APPRENDIMENTO / LIVELLO DI CONOSCENZA

I seguenti obiettivi di apprendimento sono definiti come richiesti da questo syllabus. Ogni argomento nel syllabus sarà esaminato in accordo all'obiettivo di apprendimento associato a quell'argomento.

LIVELLO 1: RICORDARE (K1)

Il candidato è in grado di riconoscere, ricordare e richiamare un termine od un concetto.

Parole chiave

Sapere, conoscere, ricordare, individuare, richiamare alla memoria

Esempio

Può riconoscere la definizione di “failure” come:

- “mancata consegna del servizio ad un utente finale o ad ogni altro stakeholder” oppure
- “deviazione attuale del componente o del sistema dalla sua consegna, dal servizio o dal risultato attesi”.

LIVELLO 2: COMPRENDERE (K2)

Il candidato è in grado di fornire le ragioni o le spiegazioni per le affermazioni legate all'argomento, ed è in grado di riassumere, confrontare, classificare, categorizzare e fornire esempi per i concetti di testing.

Parole chiave

Riassumere, generalizzare, astrarre, classificare, comparare, mappare, esemplificare, interpretare, tradurre, rappresentare, categorizzare, modellare, inferire, concludere.

Esempio

Può spiegare la ragione del perché i test devono essere progettati il prima possibile:

- Per trovare i difetti quando essi sono meno costosi da rimuovere.
- Per trovare prima i difetti più importanti/critici.

Può spiegare le similitudini e le differenze tra il testing di integrazione e il testing di sistema:

- Similarità: testing di più di un componente, e possono testare aspetti non funzionali.
- Differenze: il testing di integrazione si concentra su interfacce ed interazioni, mentre il testing di sistema si concentra su aspetti dell'intero sistema, come l'elaborazione end-to-end.

LIVELLO 3: APPLICARE (K3)

Il candidato è in grado di scegliere la corretta applicazione di un concetto o di una tecnica e applicarli nell'ambito di un dato contesto.

Parole chiave

Implementare, eseguire, utilizzare, seguire una procedura

Esempio

- Può identificare valori limite (boundary values) per partizioni valide ed invalide.
- Può scegliere test cases da un diagramma di stato-transizioni per coprire tutte le transizioni.

LIVELLO 4: ANALIZZARE (K4)

Il candidato è in grado di dettagliare le informazioni riguardanti una procedura od una tecnica .per una loro maggior comprensione ed è in grado di distinguere fra fatti e congetture. Un tipico esempio è l'analizzare un documento, un software od una situazione progettuale e proporre poi le azioni più opportune per risolvere un problema od un compito.

Parole chiave

Analizzare, integrare, organizzare, strutturare, differenziare, distinguere, discriminare, selezionare, focalizzare

Esempio

- Analizzare i rischi di prodotto e proporre azioni preventive e migliorative di riduzione dei rischi.
- Descrivere quali parti di un Report di incidente siano basate sui fatti e quali invece derivanti da congetture.

Riferimento

(For the cognitive levels of learning objectives)

Anderson, L. W. and Krathwohl, D. R. (eds) (2001) *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives*, Allyn & Bacon.

APPENDICE C – REGOLE APPLICATE A ISTQB®

SYLLABUS ‘FOUNDATION’

Le regole qui elencate sono state usate nello sviluppo e nella revisione di questo syllabus (un’etichetta viene indicata al termine di ogni regola come abbreviazione mnemonica della regola)

Regole generali

SG1. Il syllabus deve essere comprensibile ed assorbibile da persone con esperienza da 0 a 6 mesi (o maggiore) nel testing. (6-MESI)

SG2. Il syllabus deve essere pratico e non teorico. (PRATICO)

SG3. Il syllabus deve essere chiaro e non ambiguo per i lettori cui è rivolto. (CHIARO)

SG4. Il syllabus deve essere comprensibile alle persone di differenti paesi e facilmente traducibile in differenti lingue. (TRADUCIBILE)

SG5. Il syllabus deve usare l’Inglese Americano. (AMERICAN-ENGLISH)

Contenuto attuale

SC1. Il syllabus deve includere concetti recenti di testing e deve riflettere la migliore pratica corrente (‘best practice’) e generalmente accettata nel software testing. Il syllabus è soggetto a revisione ogni 3-5 anni. (RECENTE)

SC2. Il syllabus deve minimizzare i problemi legati al tempo, come concetti legati alle attuali situazioni di mercato, per consentirgli di avere una durata dai tre ai cinque anni. (CONSERVAZIONE)

Obiettivi di apprendimento

LO1. Gli obiettivi di apprendimento devono distinguere tra argomenti che devono essere riconosciuti/ricordati (livello cognitivo K1), argomenti che il candidato deve comprendere concettualmente (K2), argomenti che il candidato deve essere in grado di mettere in pratica/usare (K3) ed argomenti relativi alla capacità del candidato di analizzare documenti, software, situazioni progettuali in specifici contesti (K4). (LIVELLO DI CONOSCENZA)

LO2. La descrizione del contenuto deve essere consistente con gli obiettivi di apprendimento. (CONSISTENZA)

LO3. Per illustrare gli obiettivi di apprendimento, delle semplici domande d’esame per ogni sezione principale devono essere emesse con il syllabus. (ESAMI)

Struttura complessiva

ST1. La struttura del syllabus deve essere chiara e consentire dei riferimenti incrociati a e da altre parti, da domande d’esame a da altri documenti di rilievo. (RIFERIMENTI INCROCIATI)

ST2. La sovrapposizione tra le diverse sezioni del syllabus deve essere minimizzata. (SOVRAPPOSIZIONE)

ST3. Ogni singola sezione del syllabus deve avere la stessa struttura (CONSISTENZA STRUTTURALE)

ST4. Il syllabus deve contenere la versione, la date di rilascio e il numero di pagina di ogni pagina (VERSIONE)

ST5. Il syllabus deve includere una linea guida per la quantità di tempo che deve essere speso in ogni singola sezione (per riflettere anche l'importanza relativa di ogni argomento). (CONSUMO DI TEMPO)

Riferimenti

SR1. Le fonti e i riferimenti saranno forniti per alcuni concetti nel syllabus che forniscano un supporto ai fornitori della formazione nel trovare maggiori informazioni sui vari argomenti. (RIFERIMENTI)

SR2. Il syllabus deve fornire maggiore dettaglio laddove non ci siano delle fonti chiare e puntualmente identificate. Per esempio, le definizioni sono nel Glossario, quindi solamente i termini sono elencati nel syllabus. (RIFERIMENTO A DEFINIZIONI)

Fonti di informazione

I termini usati nel syllabus sono definiti nel Glossario ISTQB® dei termini usati nel Software Testing. Il Glossario è reso disponibile da ISTQB®.

Anche un elenco di testi consigliati sul software testing è presente nel syllabus. Tale elenco fa parte della sezione dei Riferimenti.

APPENDICE D – NOTE AI FORNITORI DELLA FORMAZIONE

Ad ogni intestazione di un argomento importante nel syllabus, è assegnato un tempo predefinito in minuti. Il suo scopo è sia di dare una guida sulle relative proporzioni temporali allocate ad ogni sezione di un corso accreditato, sia di dare un tempo minimo indicativo per l'insegnamento di ogni sezione.

I fornitori della formazione possono spendere più tempo di quello indicato ed i candidati possono spendere più tempo ancora nella lettura e nella ricerca. Il programma di un corso non deve necessariamente seguire lo stesso ordine del syllabus.

Il syllabus contiene riferimenti a standard ben prestabiliti, i quali devono essere usati nella preparazione del materiale formativo. Ogni standard usato deve essere quello della versione citata nella versione corrente di questo syllabus. Altre pubblicazioni, modelli o standards non riferiti in questo syllabus possono essere usati e citati, ma non saranno esaminati.

Le aree specifiche del syllabus che richiedono esercizi pratici sono le seguenti:

Tecniche basate sulle specifiche o black-box

Del lavoro pratico (breve esercizi) devono essere inclusi coprendo le quattro tecniche: partizionamento in classi di equivalenza, analisi ai valori limite, testing basato su tabelle delle decisioni, e testing basato su diagrammi di transizioni tra stati. Le lezioni e gli esercizi legati a queste tecniche devono essere basati sui riferimenti forniti per ogni tecnica.

Tecniche basate sulla struttura o white-box

Del lavoro pratico (breve esercizi) devono essere inclusi per valutare se un insieme di test raggiunge, oppure no, il 100% di copertura delle istruzioni e il 100% di copertura delle decisioni, tanto quanto progettare test cases per assegnati flussi di controllo.

Gestione degli incidenti

Del lavoro pratico (breve esercizi) devono essere inclusi per coprire la scrittura e/o la valutazione di un report di incidente.

APPENDICE E - RELEASE NOTES SYLLABUS 2007

1. Learning Objectives (LO) numbered to make it easier to remember them.
2. Wording changed for the following LOs (content and level of LO remains unchanged): 1.1.5, 1.5.1, 2.3.5, 4.1.3, 4.1.3, 4.1.4, 4.3.2, 5.2.2.
3. LO 3.1.4 moved from Chapter 3.3.
4. LO 4.3.1 and 4.4.3 moved from Chapter 4.1. K-Level change in Chapter 4.1
5. LO “Write a test execution schedule for a given set of test cases, considering prioritization, and technical and logical dependencies. (K3)” moved from section 4.1 to LO 5.2.5.
6. LO-5.2.3 “Differentiate between conceptually different test approaches such as analytical, model-based, methodical, process/standard compliant, dynamic/heuristic, consultative and regression averse. (K2)” added because chapter 5.2 specifies the topic and the LO was missing.
7. LO-5.2.6 “List test preparation and execution activities that should be considered during test planning. (K1)” has been added. Before, this list was part of chapter 1.4 and covered by the LO 1.4.1.
8. Section 4.1 changed to “Test Development Process from “Identifying test conditions and designing test cases”.
9. Section 2.1 now discusses the two development models: sequential vs. iterative-incremental.
10. Terms just mentioned in the first chapter of occurrence. Removed in the subsequent chapters.
11. Terms now all in singular
12. New terms: Fault attack (4.5), incident management (5.6), retesting (1.4), error guessing (1.5), independence (1.5), iterative-incremental development model (2.1), static testing (3.1), and static technique (3.1).
13. Removed terms: software, testing, development (of software), test basis, independent testing, contractual acceptance testing (2.2), retirement (2.4), modification (2.4), migration (2.4), kick-off (3.2), review meeting (3.2), review process (3.2).
14. "D" designation has been removed from Section 6.1.5 Test data preparation tools.

APPENDICE F - RELEASE NOTES SYLLABUS 2010

1. Changes to Learning Objectives (LO) include some clarification
 - a. Wording changed for the following LOs (content and level of LO remains unchanged): LO-1.2.2, LO-1.4.1, LO-2.1.1, LO-2.1.3, LO-4.6.1, LO-6.3.2
 - b. K4 has been added. Reason: some requirements (LO-4.4.4 and LO-5.6.2) have already been written in a K4 manner, and LO-4.6.1 questions are easier to write and examine on K4 level.
 - c. LO-1.1.5 has been reworded and upgraded to K2. Because a comparison of terms of defect related terms can be expected.
 - d. LO-1.2.3 Explain the difference between the two activities debugging and testing is a new LO. The content was already covered.
 - e. LO-3.1.3 Comparison issues covered
 - f. LO-3.1.4 removed. Partly redundant with LO-3.1.3.
 - g. LO-3.2.1 Consistency with content.
 - h. LO-3.3.2 upgraded to K2 in order to be consistent with LO-3.1.2
 - i. LO-6.1.2 removed as it is part of LO-6.1.3, which has been reworded due to non appropriate use of a K2 keyword
2. Consistent use for test approach according to the definition in the glossary. The term test strategy will not be required as term to recall.
3. Chapter 1.4 now contains the concept of traceability between test basis and test cases.
4. Chapter 2.x now contains test objects and test basis.
5. Re-testing is now the main term as in the glossary and not confirmation testing.
6. The aspect data quality and testing has been added at several locations in the syllabus: data quality and risk in Chapter 2.2, 5.5, 6.1.8
7. Chapter 5.2.3 Entry Criteria are added as a new subchapter. Reason: Consistency to Exit Criteria (-> entry criteria added to LO-5.2.9).
8. Consistent use of the terms test strategy and test approach with their definition in the glossary.
9. Chapter 6.1 shortened because the tool descriptions were too large for a 45 minute lesson.
10. IEEE Std 829:2008 has been released. This version of the syllabus does not yet consider this new edition. Section 5.2 refers to the document Master Test Plan. The content of the Master Test Plan is covered by the concept that the document "Test Plan" covers different levels of planning: Test plans for the test levels can be created as well as a test plan on the project level covering multiple test levels. Latter is named Master Test Plan in this syllabus and in the ISTQB® Glossary.
11. Code of Ethics has been moved from the CTAL to CTFL.

APPENDICE G - RELEASE NOTES SYLLABUS 2011

1. General: Working Party replaced by Working Group
2. Replaced post-conditions by postconditions in order to be consistent with the ISTQB® Glossary 2.1.
3. First occurrence: ISTQB replaced by ISTQB®
4. Introduction to this Syllabus: Descriptions of Cognitive Levels of Knowledge removed, because this was redundant to Appendix B.
5. Section 1.6: Because the intent was not to define a Learning Objective for the “Code of Ethics”, the cognitive level for the section has been removed.
6. Section 2.2.1, 2.2.2, 2.2.3 and 2.2.4, 3.2.3: Fixed formatting issues in lists.
7. Section 2.2.2 The word failure was not correct for “...isolate failures to a specific component ...”. Therefore replaced with “defect” in that sentence.
8. Section 2.3: Corrected formatting of bullet list of test objectives related to test terms in section Test Types (K2).
9. Section 2.3.4: Updated description of debugging to be consistent with Version 2.1 of the ISTQB® Glossary.
10. Section 2.4 removed word “extensive” from “includes extensive regression testing”, because the “extensive” depends on the change (size, risks, value, etc.) as written in the next sentence.
11. Section 3.2: The word “including” has been removed to clarify the sentence.
12. Section 3.2.1: Because the activities of a formal review had been incorrectly formatted, the review process had 12 main activities instead of six, as intended. It has been changed back to six, which makes this section compliant with the Syllabus 2007 and the ISTQB® Advanced Level Syllabus 2007.
13. Section 4: Word “developed” replaced by “defined” because test cases get defined and not developed.
14. Section 4.2: Text change to clarify how black-box and white-box testing could be used in conjunction with experience-based techniques.
15. Section 4.3.5 text change “..between actors, including users and the system..” to “ ... between actors (users or systems), ... “.
16. Section 4.3.5 alternative path replaced by alternative scenario.
17. Section Structured-Based or White Box Techniques (K4): In order to clarify the term branch testing in the text of Section Structured-Based or White Box Techniques (K4), a sentence to clarify the focus of branch testing has been changed.
18. Section 4.5, Section 5.2.6: The term “experienced-based” testing has been replaced by the correct term “experience-based”.
19. Section 6.1: Heading “6.1.1 Understanding the Meaning and Purpose of Tool Support for Testing (K2)” replaced by “6.1.1 Tool Support for Testing (K2)”.
20. Section 7 / Books: The 3rd edition of [Black,2001] listed, replacing 2nd edition.
21. Appendix D: Chapters requiring exercises have been replaced by the generic requirement that all Learning Objectives K3 and higher require exercises. This is a requirement specified in the ISTQB® Accreditation Process (Version 1.26).
22. Appendix E: The changed learning objectives between Version 2007 and 2010 are now correctly listed.

INDICE DEI TERMINI

| | | | |
|---|--|---|----------------------------|
| affidabilità | 49; 53; 57 | pianificazione del testing | 45; 88 |
| Agile | 23 | principi del testing | 16 |
| alpha testing | 27 | processo di sviluppo di test | 38 |
| ambiente di test | 47; 51 | qualità | 14; 37; 46 |
| analisi ai valori limite | 37; 40 | RAD | 23 |
| analisi di impatto | 22; 30; 38 | reportistica del testing | 51 |
| analisi statica | 15; 31; 32; 35; 58 | reportistica di segnalazione | 45; 46 |
| apprendimento degli obiettivi | 45 | requisiti | 15; 23; 26; 28; 32 |
| approcci di testing | 49 | responsabilità | 31; 33 |
| architettura | 22; 29 | re-testing | 29 |
| attività test di pianificazione del testing | 49 | revisione | 31; 32; 33; 34; 66 |
| beta testing | 27 | revisione formale | 31; 32 |
| big bang | 25 | revisione informale | 31; 33 |
| bottom-up | 25 | revisione tecnica | 31; 34 |
| bug(baco) | 13 | revisioni | 31 |
| causa effetto | 13 | revisore | 34 |
| classificazione degli strumenti di testing | 57 | rischio | 52; 53; 58 |
| CMMI | 23 | rischio | 25; 45; 50; 52 |
| compiti del tester | 47 | rischio di prodotto | 45; 53 |
| complessità | 35 | rischio di progetto | 45; 52 |
| considerazioni per alcuni tipi di strumenti | 61 | risultato atteso | 37 |
| controllo del testing | 45; 50; 51 | ruoli | 31; 33; 34 |
| controllo di versione | 51 | RUP | 23 |
| copertura | 29; 37; 42 | sbaglio | 13 |
| copertura degli statement | 42 | scelta delle tecniche di testing | 43 |
| copertura del codice | 37 | script di test | 18 |
| copertura delle decisioni | 37; 42 | scripting | 59; 61 |
| COTS | 23; 27; 50 | segnalazione | 45; 46; 54 |
| criteri di uscita | 19; 34; 45; 47; 48; 49 | sicurezza | 27; 28; 36; 38; 49; 50; 59 |
| criterio di ingresso | 34; 48; 49; 51 | specification-based technique | 40 |
| data-driven testing | 57; 61 | stakeholders | 19; 45 |
| dati di test | 47 | stima del testing | 49 |
| debugging | 29 | strategie di testing | 49 |
| densità dei difetti | 50 | strumenti di analisi statica | 31; 62 |
| difetto | 13; 14; 15; 22; 31; 34; 45 | strumenti di esecuzione di test | 56 |
| drivers | 24 | strumenti di supporto | 56; 60 |
| effetto sonda | 57 | strumenti di supporto per il testing | 56; 60 |
| errore | 13 | strumenti di supporto per il testing statico | 58 |
| esecuzione di test | 35; 45; 56; 59 | strumenti di supporto per l'esecuzione ed il logging dei test | 57 |
| failure | 13; 22; 50 | strumenti di supporto per la gestione del testing e dei test | 57 |
| fault | 13 | strumenti di supporto per prestazioni e monitoraggio | 59 |
| field testing | 27 | strumenti di supporto per specifica dei test | 58 |
| flusso di controllo | 35; 37; 42 | strumenti di tracciamento dei difetti | 58 |
| framework per unit test | 59 | stub | 24; 59 |
| frequenza delle failures | 50 | sviluppo | 14; 22; 23; 35 |
| gestione del testing | 45 | sviluppo software | 14; 22 |
| gestione della configurazione | 45; 51 | tecniche basate sull'esperienza | 39; 43 |
| gestione delle segnalazioni | 47 | tecniche basate sulla struttura | 39; 42; 43 |
| indipendenza | 19; 20; 46; 47 | tecniche black-box | 37; 40 |
| integrazione | 23; 45; 68 | tecniche di test design | 37 |
| introduzione di uno strumento all'interno di una organizzazione | 56 | tecniche statiche | 31 |
| ISO 9126 | 29; 30; 64 | test case | 18; 37; 39; 45; 69 |
| ispezione | 31; 33; 34; 35 | test case specification | 37 |
| keyword-driven testing | 57; 61 | test design | 37; 38; 39; 43; 47; 49; 58 |
| learning objective | 69 | test design specification | 45 |
| livello di test | 19; 22; 24; 25; 26; 27; 28; 37; 45; 47 | test di accettazione | 26; 27 |
| maturità | 19 | test di sistema | 25 |
| metriche | 33; 34; 35; 36; 45; 49; 50; 51; 58 | test harness | 52 |
| modello a V | 22; 23 | test leader | 20; 45; 46 |
| modello di sviluppo | 22 | test log | 19 |
| modello di sviluppo del software | 22 | test manager | 47; 53 |
| moderatore | 33 | test plan | 31; 45; 48; 49; 53; 73 |
| monitoraggio avanzamento del testing | 50 | test planning | 73 |
| monitoraggio del testing | 50; 51 | test procedure | 37; 38; 45; 48 |
| obiettivi di apprendimento | 13; 22; 31; 37; 56 | test procedure specification | 37 |
| organizzazione del testing | 46 | test report | 50 |
| partizionamento in classi di equivalenza | 37; 40 | test script | 18; 31; 38; 61 |
| peer review | 34 | test summary report | 45 |
| pianificazione dei test | 19 | test-driven | 24 |

| | | | |
|---|----------------|--|------------------------|
| tester | 13; 45; 46; 47 | testing di sistema..... | 23; 27; 68 |
| testing basato su diagrammi di transizioni tra stati..... | 37; 41 | testing di stress..... | 28; 59 |
| testing basato su tabelle delle decisioni..... | 37; 40 | testing di usabilità..... | 27; 28; 45 |
| testing basato sulla struttura..... | 37; 42 | testing dinamico..... | 15; 31; 32; 35; 58 |
| testing basato sulle specifiche..... | 37 | testing e qualità..... | 14 |
| testing black-box..... | 28 | testing esaustivo..... | 16 |
| testing confermativo | 19; 22 | testing esplorativo..... | 43; 50 |
| testing degli statement | 42 | testing funzionale..... | 28 |
| testing delle decisioni | 42 | testing non-funzionale..... | 28 |
| testing di accettazione..... | 23 | testing operativo(di accettazione) | 27 |
| testing di affidabilità..... | 28 | testing statico | 31; 57; 58 |
| testing di carico..... | 28; 59 | testing strutturale | 22; 24; 25; 28; 29; 43 |
| testing di componente | 23 | testing white-box | 29; 39; 42; 72 |
| testing di componenti..... | 25; 27; 37; 42 | testware..... | 18; 19; 47; 52; 60 |
| testing di integrazione..... | 23; 25; 45; 68 | tipi di strumenti di testing | 56 |
| testing di integrazione dei componenti..... | 25 | tipi di test | 28 |
| testing di interoperabilità | 28 | tipo di test | 22 |
| testing di mantenibilità | 28 | top-down..... | 25 |
| testing di manutenzione | 22; 29; 30 | tracciabilità | 37; 38; 47; 52; 58; 60 |
| testing di portabilità | 28 | usabilità..... | 27; 45; 53; 60 |
| testing di prestazioni | 29; 57 | use case testing | 37; 41 |
| testing di regressione | 22 | validazione..... | 23 |
| testing di robustezza | 24 | verifica..... | 23 |
| testing di sicurezza..... | 28 | walkthrough..... | 31; 33; 34 |