The European Online Magazine for IT Professional
http://www.upgrade-cepis.org
Edizione italiana a cura di ALSI e Tecnoteca
http://upgrade.tecnoteca.it

# Il bisogno di velocità: automatizzare i test di verifica in un ambiente eXtreme Programming

**di**
**Lisa Crispin, Tip House, Carol Wade (come collaboratrice)**
**(Traduzione italiana, a cura di Luigi Caso (ALSI), dell'articolo "The Need for Speed: Automatic Acceptance Testing in an eXtreme Programming Environment", pubblicato sul Vol. III, No. 2, Aprile 2002, della rivista online UPGrade, a cura del CEPIS)**

**Parole chiave:** Test, Test di Verifica, Tester, Test Automatizzati, Script di Test, Strumenti di Test, Test di GUI, Test di Applicazioni Web.

## Introduzione

I tre libri (i titoli nel prossimo paragrafo, *ndr*) su eXtreme Programming (XP) danno spiegazioni dettagliate su molti aspetti della parte legata allo sviluppo in ambiente XP. Il "test engineer" proveniente da un ambiente tradizionale di sviluppo del software può non trovare sufficienti indicazioni su come effettivamente automatizzare i test di verifica e contemporaneamente stare al passo con il ritmo veloce di un progetto XP. In un team XP, è probabile che anche gli sviluppatori si ritrovino ad automatizzare i test di verifica – un'area in cui possono non avere molta esperienza. Automatizzare i test di verifica in un progetto XP può somigliare a guidare un maggiolino Volkswagen per una discesa con il 12% di pendenza avendo un auto-articolato lanciato a tutta velocità visibile nello specchietto retrovisore. Non c'è da preoccuparsi – come tutto in XP, richiede coraggio, ma può – e dovrebbe – essere divertente, non spaventoso.

Le regole XP che noi seguiamo in Tensegrent prevedono:

- "pair programming"
- prima i test, poi il codice
- fare la cosa più semplice che funziona (NON la cosa più fantastica che funziona!)
- 40 ore alla settimana
- "refactoring"
- standard di scrittura del codice
- piccoli rilasci
- pianificazione

Applichiamo queste stesse regole alla fase di test – compreso il "pair testing".

I team XP hanno veramente bisogno di un tester ad essi dedicato? È difficile per chi fa il tester rispondere in modo obiettivo. Per mia esperienza, anche gli sviluppatori senior non hanno molta pratica con i test, a parte quelli di singole unità ("unit test") e di integrazione, e forse quelli "di carico". Essi tendono a scrivere i test di accettazione solo per i cosiddetti "happy paths" e non pensano a quelle spiacevoli situazioni che potrebbero far fallire il sistema. In Tensegrent, avevamo un progetto che si stava concludendo mentre un altro stava per partire, così fu presa la decisione di fare la prima iterazione di due settimane del nuovo progetto con uno sviluppatore che serviva da tester part-time. Per loro stessa ammissione, senza un tester esperto a metterli sotto pressione, gli sviluppatori considerarono terminate il 90% delle cose da fare alla fine dell'iterazione. Al cliente tutto questo

 

sembrò come se non fosse stato fatto assolutamente nulla, ed erano molto scontenti. C'è voluto un pò di lavoro per riguadagnare la fiducia del cliente.

## Come si differenzia il test in XP?

Quanto si distaccano i test di verifica in un ambiente XP dai tradizionali test del software? Prima di tutto, diamo uno sguardo ai test di verifica. I test di verifica dimostrano che l'applicazione funziona secondo i voleri del cliente. Danno la sicurezza, ai clienti, ai manager e agli sviluppatori, che l'intero prodotto stia procedendo nella direzione giusta. Controllano ogni incremento nel ciclo XP per verificare che ci sia "business value". Sono, sotto la responsabilità del tester e del cliente, test "end-to-end" eseguiti dalla prospettiva del cliente, che non provano a testare ogni possibile percorso attraverso il codice (gli "unit test" si occupano di questo), ma dimostrano che l'applicazione abbia un "business value". Possono anche includere test di carico, stress e performance per dimostrare che la stabilità del sistema soddisfa i requisiti del cliente.

*Bisogna metter su un elmetto ed armare gli air bag?*

I test in un ambiente XP all'inizio sembrano una gita in auto su delle strade serpeggianti di montagna. Quando ho letto per la prima volta *eXtreme Programming Explained*, l'idea stessa di fare dei test senza nessun tipo di specifiche formali scritte mi è sembrata un pò TROPPO "extreme". È stato difficile imparare tutti i diversi modi con cui poter contribuire al successo del team. I miei ruoli possono essere svianti e conflittuali – io faccio parte del team di sviluppo, ma ho bisogno di un punto di vista più obiettivo. Io sono un garante per il cliente, e devo assicurare che abbia quello per cui paga. Allo stesso tempo, ho bisogno di proteggere chi sviluppa da un cliente che vuole PIÙ di quello per cui ha pagato.

Mentre XP è senza dubbio un nuovo modo di guidare, la strada non è così poco familiare come si potrebbe pensare. Per esempio, molte persone nuove ad XP pensano che i progetti XP producano pochissima documentazione. Così per noi non è stato. In primo luogo, gli stessi test di verifica diventano la documentazione principale dei requisiti del cliente: possono essere abbastanza ampi e dettagliati. Man mano che un progetto XP va avanti, molti altri docuenti possono essere prodotti: istruzioni per l'installazione, documenti UML (Unified Modeling Language), Javadocs, documenti di setup dello sviluppatore, la lista può continuare. La differenza tra questi ed i documenti di molti progetti tradizionali è che i documenti di un progetto XP sono aggiornati e precisi.

*Domanda*: Come si fa a scrivere dei "test case" di verifica senza documenti?

*Risposta*: Non c'è bisogno di documenti, perchè c'è un cliente che è lì per spiegare cosa sta cercando. Non che questo sia sempre facile. Per esperienza, è abbastanza facile incontrare un cliente che si presenta con dei test per le funzionalità desiderate dal sistema. Ciò che è più difficile, e richiede l'abilità di un tester, è assicurarsi che il cliente si preoccupi di aree come la sicurezza, la gestione degli errori, la stabilità e le performance in condizioni di carico.

Altre differenze tra lo sviluppo tradizionale e quello XP sono più sottili: è veramente una questione di sfumature. I progetti XP procedono veloci anche se confrontati con il ritmo della "Web startup" dove abitualmente lavoro. È la corsia veloce sulla Autobahn (l'autostrada tedesca, *ndr*). Una nuova iterazione del software, che implementa nuove richieste da parte del cliente, viene rilasciata ogni settimana, al più ogni tre. Il mio obiettivo è sempre l'aver definito i "test case" di verifica entro il primo, al massimo il secondo, giorno di un'iterazione, in quanto queste sono le uniche specifiche scritte disponibili. Per i nostri progetti, le definizioni dei test di verifica sono state un lavoro congiunto del team.

Dal punto di vista di un tester, il rapporto tra sviluppatori e tester in XP appare confortevole come guidare una Jeep senza aria condizionata attraverso il deserto. Secondo Kent Beck, ci dovrebbe essere un tester per ogni team di otto sviluppatori; in Tensegrent, il rapporto è anche più alto.

 

*Eeeeh! Siamo SICURI che non sia richiesto equipaggiamento protettivo?*

Niente paura! XP incorpora controlli ed equilibri tali da consentire ad una piccola percentuale di specialisti di test di fare un lavoro adeguato di controllo qualità.

- Poichè gli sviluppatori scrivono così tanti "unit test", e li devono scrivere prima di iniziare la codifica - il tester non ha bisogno di verificare ogni possibile percorso attraverso il codice.
- Gli sviluppatori sono responsabili dei test di integrazione e devono eseguire tutti gli "unit test" ogni volta che fanno il "check-in" del codice. I problemi di integrazione vengono rivelati prima che vengano eseguiti i test di verifica.
- Il cliente dà degli input ai test di verifica e fornisce dati per i test stessi.
- L'intero team di sviluppo, non solo il tester, è responsabile dell'automazione dei test di verifica. Anche gli sviluppatori aiutano il tester a produrre i report sui risultati dei test così che tutti siano confidenti su come il progetto stia procedendo.

Una raccomandazione – se gli sviluppatori non sono precisi nello scrivere gli "unit test" e i test di integrazione, e nell'eseguirli di frequente, ci sarà bisogno di assumere più tester. Dopo un paio di iterazioni nel nostro primo progetto in Tensegrent, dissi al mio capo che pensavo che dovevamo assumere più tester, non era possibile che io potessi stare al passo! Il problema era semplicemente che gli sviluppatori non avevano ancora compreso il concetto di testare prima di codificare. Una volta fatto un accurato lavoro di "unit test" e di test di integrazione, il mio lavoro diventò molto più gestibile.

I ruoli dei partecipanti ad un team XP sono abbastanza confusi rispetto a quelli di un processo di sviluppo tradizionale del software. Così la nostra filosofia XP in Tensegrent è "*la specializzazione è per gli insetti*". Ecco alcuni dei compiti che io svolgo come tester:

- Aiuto al cliente nello scrivere i requisiti
- Aiuto nel dividere i requisiti in singoli compiti e valutare il tempo necessario a completarli
- Aiuto nel chiarire i problemi del progetto
- Scrittura, insieme al cliente, dei test di verifica
- Definizione e sviluppo, insieme agli sviluppatori, degli strumenti di test, degli script per i test automatizzati e/o dei dati per i test.

*Domanda*: Il concetto di "pair programming" suona abbastanza strano. Come può un tester lavorare in coppia con un programmatore?

*Risposta*: Io non sono un programmatore Java e i nostri sviluppatori non conoscono il linguaggio di scripting WebART, ma noi tuttora programmiamo in coppia. Il partner che in quel momento non sta usando la tastiera contribuisce ragionando sulla strategia, evidenziando errori di battitura e cattive abitudini ed anche facendo da "scheda audio" a colui che scrive il codice. Questo è un modo favoloso per sviluppatori e tester per capire e lavorare meglio insieme. Consente anche al tester molta più comprensione del sistema che si sta codificando.

All'inizio io ero riluttante riguardo al "pair testing". Se gli sviluppatori scrivevano gli script di test, sarei stata in grado di comprenderli e manutenerli? Nemmeno gli sviluppatori erano così ansiosi di lavorare in coppia con me per la fase di test. Si sentivano troppo occupati per dedicare tempo ai test di verifica. Poi ci capitò un progetto in cui avevo bisogno di dati di test molto complessi da caricare in un database POET (http://www.poet.com) per testare un modello di sicurezza. Lavorando in coppia con uno sviluppatore, ci misi almeno la metà del tempo che ci avrei messo facendolo da sola, e svolsi un lavoro migliore. Ora gli sviluppatori fanno a turno "test support" per produrre gli script di test ed i dati necessari per l'automazione, talvolta anche per aiutare a definire i "test case" se sto avendo problemi nel comprendere una situazione.

Una volta preso il coraggio a due mani per passare alla corsia veloce di XP, è divertente e sicuro.

*Come ci si autoeduca ad XP?*

Così come nessuno proverebbe a guidare una macchina di Formula Uno senza prepararsi con allenamento ed esercizi, il team XP necessita di un buon allenamento per partire sulla strada giusta e rimanervici.

 

Si comincia leggendo i libri su XP. Il primo libro scritto su XP è *Extreme Programming Explained*, di Kent Beck. Anche gli altri due sono essenziali: *Extreme Programming Installed*, di Ron Jeffries, Ann Anderson e Chet Hendrickson, e *Planning Extreme Programming*, di Kent Beck e Martin Fowler.

Si può ricavare una visione generale ed avere maggiori dettagli su XP, ed altre discipline similari, dai molti siti Web ad esso relativi, inclusi:

http://www.xprogramming.com
http://www.extremeprogramming.org
http://c2.com/cgi/wiki?ExtremeProgrammingRoadmap
http://www.martinfowler.com

Quando in Tensegrent avemmo creato il nostro primo team di otto sviluppatori ed un tester, ci riunimmo ed esaminammo, in gruppo, *Extreme Programming Explained* ed *Extreme Programming Installed*, discutendo ciascun principio XP, prendendo nota delle domande (molte delle quali sulla fase di test) e decidendo come pensavamo di implementare ciascun principio. Questo portò via molte ore ma diede a tutti noi una base comune e ci rese più sicuri della nostra comprensione dei concetti.

Una volta che un nuovo team ha letto e discusso la letteratura su XP, è tempo di ricevere un addestramento professionale. Noi assumemmo Bob Martin di ObjectMentor, un'azienda di consulenza con molta esperienza su XP, per due giornate di addestramento intenso (vedere http://www.objectmentor.com per maggiori informazioni). Dopo che Bob ebbe risposto a tutte le nostre domande, ci sentimmo molto più sicuri di noi su delle aree che erano state precedentemente difficili da comprendere, come la pianificazione, gli "unit test" automatizzati ed i test di verifica.

Non bisogna fermarsi qui. Si deve parlare con esperti XP. Guardare le pagine Wiki (http://c2.com/cgi/wiki?WikiWiki) e registrarsi nei gruppi di discussione online. Se non è stato creato nessuno user group su XP nella propria città, crearne uno.

**Automatizzare i test di verifica**

*Cosa si può automatizzare?*

Secondo Ron Jeffries, l'autore di *Extreme Programming Installed*, i test di verifica conclusi con successo sono, tra le altre cose, di proprietà del cliente ed automatici. Tuttavia, essere di proprietà del cliente non significa necessariamente essere scritti dal cliente. Infatti, come fa notare Kent Beck in *Extreme Programming Explained*, i clienti non possono tipicamente scriversi i test funzionali da sè, il che spiega perchè un team XP ha un tester dedicato: per trasformare le idee del cliente in test automatici.

Anche con un tester dedicato, però, il criterio dei test automatici ci ha dato qualche problema. Noi automatizziamo ogni volta che ha senso, ma, come la maggior parte delle cose, si deve bilanciare. Quando ci si deve arrampicare per una strada ripida ed in terra battuta tutti i giorni, un veicolo a trazione integrale è una necessità, ma è un'esagerazione se si deve solo andare in giro in mezzo ai palazzi.

Per esempio, non abbiamo trovato un modo conveniente di automatizzare i test di Javascript (quindi, abbiamo semplicemente evitato di usare Javascript). E stiamo anche facendo sforzi su come automatizzare test di GUI non-Web in un tempo accettabile.

Costa tempo e denaro automatizzare i test e, una volta che li si ha a disposizione, manutenerli. Di recente abbiamo avuto un contratto per tre iterazioni di due settimane ciascuna per sviluppare alcuni componenti di un sistema per un cliente, con quattro sviluppatori più me stessa. Nonostante il sistema prevedesse un'interfaccia utente, il disegno di quest'ultima era da farsi in seguito, fuori dal nostro progetto. Noi sviluppammo una interfaccia di base semplice per poter testare il sistema. Il sistema prevedeva più server, interfacce, dei monitor ed un database. L'automazione completa dei test sarebbe stata un grosso sforzo. Non aveva senso impegnare le ristrette risorse del cliente su degli script che avevano vita breve. Tuttavia, ho automatizzato le parti più noiose dei test in modo da poterli terminare in tempo. Inoltre, ho avuto bisogno di script per i test di carico. Circa il 40% dei test ha finito per essere automatizzato. Per un progetto più lungo, preferirei automatizzare di più.

 

*Principi di automazione di test funzionali in XP*

Per avere maggiore automazione, bisogna fare in modo che abbia un ritorno in tempi brevi, e questo significa impegnare meno tempo nello sviluppo e nella manutenzione dei test automatizzati. Ecco i principi da noi usati per ottenere quest'obiettivo:

- *Pilotare il progetto di automazione dei test con uno "Smoke Test"*, ossia una verifica ampia ma leggera di tutte le funzionalità critiche.
- *Progettare i test come il software*, in modo che i test automatizzati non contengano codice duplicato ed abbiano il minor numero possibile di moduli.
- *Separare i dati dal codice dei test*, in modo da poter aumentare la copertura dei test solo aggiungendo ulteriori dati.
- *Rendere i moduli di test auto-controllati* in modo che dicano, ovviamente, se sono stati superati o sono falliti, ma anche che incorporino gli "unit test" per il modulo.
- *Verificare soltanto la funzione di interesse per un particolare test*, non ogni funzione che deve essere eseguita per costruire il test.
- *Verificare i criteri minimi di successo*. Minimi non significa insufficienti. Se non fossero buoni a sufficienza, non sarebbero quelli minimi. Bisogna dimostrare il "business value" in modo completo, ma non fare più di quello che necessita al cliente per definire il buon esito del progetto.
- *Comporre e ricomporre continuamente i test automatizzati ("refactoring")*, combinando, spezzando, o aggiungendo moduli, o cambiandone interfacce o comportamento ogni volta che serve ad evitare duplicazioni, o a rendere più semplice l'aggiunta di nuovi "test case".
- *Programmare i test in coppia*, con un altro tester o con un programmatore.
- *Progettare il software in modo che possa essere testato*, lasciando degli "hook" all'interno dell'applicazione per aiutare ad automatizzare i test di verifica. Portare quante più funzionalità è possibile nella parte di "backend", perchè è molto più facile automatizzare i test per un "backend" piuttosto che per una interfaccia utente. Io assisto alla pianificazione dell'iterazione degli sviluppatori e traccio dei rapidi schizzi delle sessioni di progetto. Se mi accorgo che c'è della "business logic" che sta entrando nell'interfaccia, per esempio Javascript, io contesto la saggezza di questo tipo di mossa.

*Un progetto XP di test automatizzato*

L'Appendice A mostra un esempio di un progetto di test leggero che illustra l'applicazione dei principi che stiamo usando con successo in Tensegrent. Sto usando WebART (vedere nel paragrafo Strumenti, di seguito) per creare ed eseguire gli script. In ogni caso, questo approccio di progetto dovrebbe funzionare con qualsiasi metodo di automazione che consenta la modularizzazione degli script. L'Appendice fornisce dettagli su come scaricare sia gli script di esempio che WebART.

*Chi automatizza i test di verifica?*

Alcuni sport sembrano essere individuali, quando in realtà coinvolgono delle squadre. I vincitori del Tour de France ricevono tutti gli onori, ma la loro vittoria rappresenta il lavoro di una squadra. Allo stesso modo, un team XP può avere solo un tester, ma l'intero team contribuisce ad automatizzare i test di verifica. Se c'è bisogno di strumenti per agevolare i test di verifica in un progetto XP, bisogna documentarli ed includerli nella pianificazione insieme a tutto il resto. È probabile che ci sia bisogno di mettere a budget almeno un paio di settimane per definire strumenti di test per un progetto di dimensioni non eccessive.

Nei primi giorni in Tensegrent, iniziammo un progetto con lo scopo specifico di sviluppare strumenti di test automatizzati. Questo ci ha dato diversi vantaggi, oltre al fatto di aver effettivamente prodotto gli strumenti stessi:

- *Fare pratica con XP*, in particolare documentare, pianificare, fare delle stime. Questo ci ha dato confidenza nei nostri skill XP che ci è servita nei progetti successivi.
- *Fare pratica con le tecnologie di sviluppo*. Gli sviluppatori potevano provare diversi approcci e prendere confidenza con nuovi strumenti. Per esempio, hanno individuato in anticipo i vantaggi di usare un parser DOM piuttosto che SAX sui file XML contenenti dati di test del cliente. Il farlo in anticipo ci ha dato più tempo per sperimentare e cercare tecnologie di quello che avremmo avuto in seguito in un progetto per un cliente.

 

- *Reciproca comprensione.* Il team cui era stato assegnato il compito di produrre un test di verifica pilota era composto di solo quattro persone più me stessa, così mi fu chiesto di fare "pair programming". Questo esercizio mi ha dato visibilità su come sia difficile scrivere "unit test", scrivere codice e scomporlo in parti. Gli sviluppatori hanno dato un grosso contributo ai test di verifica ed abbiamo avuto lunghe discussioni su quali sarebbero state le pratiche migliori. Tutto ciò è un'ottima base per qualsiasi team XP.

**Strumenti**

Per mantenere ben funzionante una macchina XP, i tester XP hanno bisogno di una buona "cassetta per gli attrezzi", che contenga strumenti progettati in modo specifico per avere velocità, flessibilità e basso sovraccarico.

Ho posto a diversi "guru" di XP, inclusi Kent Beck, Ward Cunningham e Bob Martin, la seguente domanda: "Quali strumenti commerciali usate per automatizzare i test di verifica?" La loro risposta è stata univoca: "Sviluppateli da voi". Il nostro team ha svolto ampie ricerche in quest'area. Il risultato è stato che possiamo usare uno strumento terze parti per l'automazione dei test di applicazioni Web, ma abbiamo bisogno di strumenti "fatti in casa" per altri scopi.

Per gli "unit test", usiamo un framework chiamato jUnit, che è disponibile gratis su http://www.junit.org. Svolge un lavoro notevole con gli "unit test". Anche se io non sono un programmatore Java, posso eseguire i test con il TestRunner di jUnit e posso anche comprendere il codice del test bene a sufficienza, tanto da aggiungere miei test personali. È possibile fare dei test funzionali con jUnit. Alcuni team XP usano questo strumento per automatizzare i test di verifica, ma non può essere usato per il test dell'interfaccia utente. Non lo abbiamo reputato una buona scelta per test di verifica completi.

*Strumenti per la creazione di test di verifica*

Alcuni professionisti XP, come Ward Cunningham, sostengono l'uso di fogli elettronici per pilotare i test di verifica. Noi vogliamo rendere semplice per il cliente scrivere i test, e la maggior parte di loro è a proprio agio con l'inserimento di dati in un foglio elettronico. I fogli elettronici possono essere esportati in formato testo, così che il team di sviluppo possa scrivere programmi o script per leggere i dati dal foglio e fornirli agli oggetti nell'applicazione. In caso di applicazioni finanziarie, i calcoli e le formule che il cliente mette nel foglio comunicano agli sviluppatori come il codice che producono dovrebbe funzionare.

In Tensegrent, abbiamo un paio di modi per documentare i "test case" di verifica. Di solito, usiamo un foglio elettronico di formato semplice, separando i dati stessi del "test case" dalla descrizione di passi, azioni e risultati attesi. Abbiamo anche sperimentato la scrittura di "test case" in formato XML, per dei test ad uso interno. Stiamo continuando a sperimentare "l'idea XML", ma il foglio elettronico ha funzionato bene. Nell'Appendice B c'è un template di esempio di foglio elettronico per un test di verifica.

L'Appendice C mostra un estratto parziale di un file XML di esempio, usato per dei "test case" di verifica. Il "test case" consiste di una descrizione del test, dati ed output attesi, passi con le azioni da eseguire e risultati attesi.

*Test automatizzati per applicazioni Web*

Automatizzare i test è relativamente semplice per le applicazioni Web. La sfida sta nel creare gli script automatici in modo sufficientemente veloce da stare dietro alle rapide iterazioni di un progetto XP. Questo è sempre più faticoso nelle prime iterazioni. Ci sono delle volte che io mi sento come quelle vecchie automobili lente che bloccano la corsia veloce. Per avere quella maggior iniezione di velocità, io uso WebART (http://www.oclc.org/webart), uno strumento poco costoso, basato su HTTP, con un potente linguaggio di scripting. WebART mi permette di creare script di test modularizzati, in modo da avere molte parti riutilizzabili in un tempo breve a sufficienza per stare dietro al ritmo degli sviluppi. I test di Javascript presentano un ostacolo più grande. Noi lo testiamo manualmente e controlliamo attentamente le nostre librerie Javascript per minimizzare le modifiche e così la necessità di ritestare. Nel frattempo, continuiamo a cercare dei modi per automatizzare i test di Javascript.

I nostri sviluppatori hanno scritto uno strumento per convertire i dati di test forniti dal cliente, come fogli elettronici o in XML, in un formato che può essere letto dagli script di test di WebART, in modo da poter automatizzare i test delle applicazioni Web. Anche dei piccoli sforzi come questo possono aiutare a guadagnare quel margine competitivo nel veloce mondo XP.

*Test automatizzati per applicazioni con una GUI*

 

L'automazione dei test per applicazioni non-HTTP con una GUI è stata molto più che una difficile scalata. Si può viaggiare più veloce in elicottero che su una mountain bike, ma ci vuole molto tempo per imparare a pilotare un elicottero; costano molto più di una bicicletta e si può non trovare un posto dove atterrare. Allo stesso modo, gli strumenti di automazione di test per GUI in commercio che noi abbiamo visto richiedono che molte risorse imparino e implementino. Hanno un effetto dirompente sui budget di piccole realtà come la nostra. Abbiamo cercato in lungo e in largo, ma non c'è riuscito di trovare un equivalente di WebART nel mondo del test di GUI. JDK 1.3 contiene un robot che permette di automatizzare i test di eventi GUI con Java, ma è basato sulla posizione reale dei componenti sullo schermo. Gli script basati sul contenuto dello schermo e sulle posizioni sono rigidi e costosi da manutenere. Noi abbiamo bisogno di test che diano a chi sviluppa sicurezza nel modificare l'applicazione, sapendo che i test troveranno qualsiasi problema essi introducano. I test che necessitano di essere aggiornati dopo ogni modifica di un'applicazione potrebbero farci "perdere il treno".

Noi pensiamo che il criterio più importante per i test di verifica sia che siano ripetibili, poichè devono essere eseguiti per ciascuna integrazione. Abbiamo deciso di iniziare a sviluppare un nostro strumento, *TestFactor-e,* che aiuterà clienti e tester ad eseguire test manuali in maniera coerente; registrerà anche i risultati. Abbiamo intenzione di migliorare questo strumento per inserire dati di test ed azioni direttamente nei "backend" applicativi, in modo da automatizzare i test. Poichè abbiamo sviluppato solo applicazioni Web, questo è un lavoro di "background".

Qualsiasi sistema si stia testando, ci vuole tempo per prendere velocità con l'automazione. Io pianifico di fare test manuali nella prima iterazione. All'inizio della seconda iterazione, si può iniziare ad automatizzare, usando il metodo descritto nell'Appendice A. Ci sono delle volte che incontro un ostacolo che mi fa ritardare di un giorno o due. La soluzione per questo tipo di problemi è trovare qualcuno che lavori in coppia con me. Come tester in un progetto XP, a volte ci si può sentire soli, ma, è bene ricordarlo, non si è mai soli!

*Reports*

Ricevere feedback è uno dei quattro valori di XP. Beck dice che un feedback concreto sullo stato attuale del sistema è un qualcosa di inestimabile. Se si è in un lungo viaggio, si controllano i segnali stradali e le pietre miliari che dicono a che punto del percorso si è arrivati. Se ci si rende conto di essere indietro, si salta la successiva fermata per il caffè o si aumenta un pò la velocità. Se si è in anticipo, si potrebbe deviare su una strada più panoramica. Il team XP ha bisogno di un flusso costante di informazioni per condurre il progetto, apportando correzioni per rimanere sulla strada giusta. I continui, piccoli aggiustamenti da parte del team tengono il progetto sulla giusta rotta, nei tempi e nei costi. Gli "unit test" forniscono ai programmatori un feedback minuto per minuto. I risultati dei test di verifica danno un feedback sul quadro generale per il cliente e per il team di sviluppo.

I report non hanno bisogno di essere elaborati, solo facili da leggere con un'occhiata. Un grafico che mostra il numero di test di verifica scritti, quelli attualmente in esecuzione e quelli attualmente conclusi con successo dovrebbe essere affisso ben in vista su una parete. Si possono trovare esempi di questo tipo sui libri XP. Il nostro team di sviluppo ha scritto degli strumenti per leggere sia i log dei test automatizzati che quelli dei test manuali eseguiti con *TestFactor-e.* Questi strumenti producono dei report di facile lettura, di dettaglio e di riepilogo, in formato HTML e grafico.

Con tutti questi feedback, si può essere confidenti nel consegnare software di alta qualità in tempo per battere la concorrenza. Così si affronteranno le sfide nello sviluppo del software del 21° secolo!

## Appendice A: Progetto di un test leggero

### Progetto di un test automatizzato XP

Gli script di esempio usati per illustrare il progetto dei test sono scritti con uno strumento di test chiamato WebART (http://www.oclc.org/webart). Qualsiasi strumento di test che consente la modularizzazione e la parametrizzazione degli script dovrebbe supportare questo tipo di progettazione. Per scaricare una copia degli script di esempio, si può andare su http://www.oclc.org/webart/samples (questo link, ad oggi, non è più attivo, *ndr*) e cliccare sul link *qwmain Sample Scripts.*

 

*L'applicazione di esempio*

La nostra applicazione di esempio è un sito Web per la consultazione di un elenco telefonico, http://www.qwestdex.com. Questa non è certamente da intendersi come un'operazione di sostegno a Qwest e noi non abbiamo alcuna relazione con loro, era solo un'applicazione pubblica e a portata di mano, con caratteristiche che ci permettevano di illustrare i test.

| *Azione* | *Criterio minimo di superamento* |
|---|---|
| Vai alla pagina di login | La pagina contiene la form di login |
| Login | Login e password validi portano alla pagina del profilo |
| Ricerca di una categoria valida nella città specificata | Una ricerca valida recupera una tabella contenente le aziende trovate |
| Logout | La pagina contiene i link alla pagin di login ed alla homepage |

**Tabella 1**

*Lo "Smoke Test"*

Considereremo come funzionalità critiche le fasi di login al sito e di ricerca di aziende in una certa città e categoria. Simuliamo che questa sia la cosa più importante nella prima iterazione. La Tabella 1 mostra lo scenario di base che vogliamo testare.

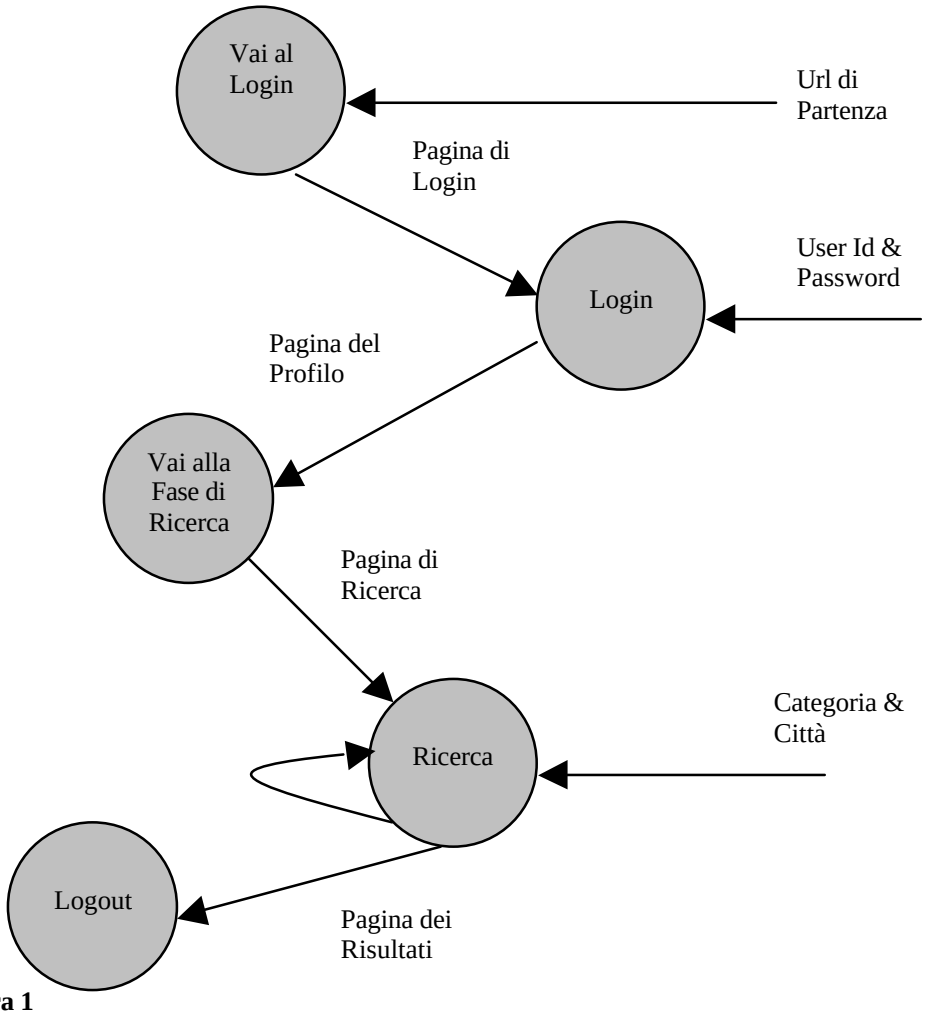


**Figura 1**

*Il progetto dei test*

Sappiamo che ci saranno più funzionalità da testare nelle iterazioni successive, ma useremo il progetto più semplice che si possa pensare per effettuare questi test senza duplicazione. Poi applicheremo il "refactoring" quanto necessario per favorire l'inserimento dei test aggiuntivi.

I moduli saranno Vai al Login, Login, Vai alla Fase di Ricerca, Ricerca e Logout. Nella figura 1, il diagramma mostra come i moduli vengano parametrizzati.

*Separare i dati di test dal codice*

Gli elementi sul lato destro del diagramma rappresentano i dati di test: l'URL della pagina di login, lo user id e la password da usare per il login, e la categoria e la città da ricercare. I dati di test sono isolati in un file del "test case", che viene letto dal test quando va in esecuzione. Nella figura 2 c'è un esempio di contenuto di quel file per eseguire un singolo "test case".

*Verifica*

I moduli principali usano un insieme di moduli base di validazione per controllare le specifiche condizioni richieste da una risposta del sistema e determinare la condizione di superamento o di fallimento. I moduli di validazione a loro volta chiamano delle utility per registrare i risultati.

Questo esempio utilizza i seguenti tre moduli di validazione:

- *vtext* verifica che una risposta contenga un testo specifico, per una stringa di testo.
- *vlink* verifica che una pagina contenga uno specifico link.
- *vform* verifica che una pagina contenga una specifica form HTML.

```
smoketest
 [
  :iter1:
  Url <url=http://qwestdex.com>
  UseridPassword <uid=bob&psw=bob>
  CatCity <cat=banks&city=dallas>
 ]
```

**Figura 2**

*Moduli di utility*

Ci sono anche due utility che vengono usate dai moduli principali:

- *trace* – mostra informazioni di tracing in esecuzione nella finestra di esecuzione di WebART, per fare debugging dei test.
- *log* – registra i risultati di verifica in un file di log.

Il modulo "*zslog*" negli script di esempio scrive i risultati dei test in formato XML. Uno strumento fatto in Tensegrent, chiamato *TestFactor-e*, costruisce una pagina HTML da questo file di log, mostrando i risultati codificati per colore a seconda se il test è superato, non eseguito o fallito. Nell'Appendice B si può trovare un esempio.

## Creare gli script

Crare il primo gruppo di script è il lavoro difficile. Una volta che si ha un "working set" di moduli, è possibile riusarli interamente in qualche caso o trasformarli in template in altri casi. Ecco i passi che io uso (preferibilmente come parte di una coppia) per creare degli script di test:

1. Cattura di una sessione per lo scenario che voglio testare. Vedere "*capqwest*" negli script di esempio.
2. Copia di "*qwmain*", "*zsqwlogin*" e degli altri moduli di supporto già in mio possesso con dei nuovi nomi. Eliminazione del codice specifico per quella applicazione.

 

3. Inserimento del codice specifico per lo scenario che voglio testare, copiandolo dallo script catturato nei nuovi template creati. Qui bisogna usare i principi XP: lavorare per piccoli incrementi, assicurarsi che gli script funzionino prima di proseguire. Per esempio, prima verificare che funzioni la fase di login. Poi aggiungere la ricerca. Poi aggiungere la logica per cambiare in base al fatto che il risultato sia test superato/test fallito. Ricordarsi di fare la cosa più semplice che funziona e di aggiungere complessità solo se ce ne è bisogno.

## Appendice B: Estratto parziale da un template XML per un “test case” di verifica

```
<?xml version=“1.0" encoding=“UTF-8” standalone=“no” ?>

<!DOCTYPE at-test SYSTEM “at-test.dtd” [
<!ELEMENT input ANY >
<!ELEMENT loan-amount ANY >
<!ELEMENT interest-rate ANY >
<!ELEMENT term-of-loan ANY >
<!ELEMENT output ANY >
<!ELEMENT monthly-payment ANY >
]>

<at-test name=“calc-monthly-payment” version=“1.0" severity=“CRITICAL”>

  <at-project>mortgage-calc</at-project>

      <at-description>
          Enter loan amount, interest rate, term of loan (in months)
          to calculate monthly payment.
      </at-description>

      <at-data-sets>
        <at-struct id=“values”>
          <input>
              <loan-amount>1000000000.00</loan-amount>
              <interest-rate>0.5</interest-rate>
              <term-of-loan>1200</term-of-loan>
          </input>
          <output>
              <monthly-payment>A big, fat wad of dough!</monthly-payment>
          </output>
        </at-struct>
      </at-data-sets>

      <at-plan>

        <at-step name=“populate-loan-amount”>
          <at-action>
              <at-text>Enter “{0}” in the “Loan Amount field”.</at-text>
              <at-value dset=“values” select=“/input[2]/loan-amount”/>
          </at-action>
          <at-expect>
              <at-text>Cursor moved to “Interest Rate” field for input.</at-text>
          </at-expect>
        </at-step>

      </at-plan>

</at-test>
```

 

## Appendice C: Esempio di uno spreadsheet per un test di verifica

| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | | **Ref #:** | Template per test di verifica: Timecard/QA/AcceptanceTest.sdc | | |
| 2 | | **Iterazione:** | | | |
| 3 | | **La funzionalità controllata da questo test case *è / non è* critica** | **Cosa fa questo test?** *Esempio: test per essere certi che quando un record con l'ora d'ingresso viene inserito, viene salvato nel database ed il report viene generato correttamente* | | |
| 4 | | **Cosa fare:** | *Gli esempi sono scritti in corsivo* | | |
| 5 | **Step** | **Comando/URL** | **Azione** | **Dati in input** | **Ouput attesi** |
| 6 | 1 | *Accesso a www/timecard/addEntry con il browser* | *Selezionare un progetto, release, iterazione, task ed inserire durata, data e commenti* | *Riga 1, colonne Progetto, Release, Iterazione, Task, Durata, Data, Commenti* | *Dati selezionati mostrati sullo schermo* |
| 7 | 2 | *Ancora su www/timecard/addEntry* | *Cliccare sul bottone Salva* | | *Messaggio che i dati sono stati salvati nel database* |
| 8 | 3 | *Accesso a www/timecard/generateReports con il browser* | *Selezionare un progetto, data di inzio e data di fine* | *Riga 1, colonne Progetto, Data Inizio, Data Fine* | *Report generato - i dati corrispondono alla riga 1, colonne Progetto, Release, Iterazione, Durata, Costo e Costo Totale* |
| 9 | 4 | *Ripetere i passi 1-3 con ciascuna riga del foglio elettronico del test case* | | | |

**Note biografiche sugli autori**

**Lisa Crispin** ha più di 10 anni di esperienza in attività di test e controllo qualità ed è attualmente Senior Consultant presso la BoldTech Systems (http://www.boldtech.com), dove lavora come tester in team che operano in ambienti eXtreme Programming. Il suo articolo "Extreme Rules of the Road: How an XP Tester Can Steer a Project Toward Success" è apparso nel numero di Luglio 2000 della rivista STQE (*Software Testing and Quality Engineering* magazine, http://www.stqemagazine.com). La sua presentazione "The Need for Speed: Automating Acceptance Tests in an eXtreme Programming Environment" ha vinto il premio come miglior presentazione al Quality Week Europe nel 2000. I suoi saggi "Testing in the Fast Lane: Acceptance Test Automation in an eXtreme Programming Environment" e "Is Quality Negotiable?" saranno pubblicati da Addison-Wesley in una raccolta chiamata *Extreme Programming Perspectives*. Sta scrivendo, insieme ad altri autori, un libro, *Testing for Extreme Programming*, che sarà pubblicato da Addison-Wesley nell'ottobre 2002. Le sue presentazioni ed i suoi seminari nel 2001 sono stati "XP Days" a Zurigo, Svizzera, "XP Universe" a Raleigh e STARWest ("Software Testing Analysis and Review", Western Coast) a San Jose. Lisa può essere contattata all'indirizzo lisa.crispin@att.net.
**Tip House** è Chief Systems Analyst presso OCLC, Online Computer Library Centre Inc., un'organizzazione no-profit che si dedica a promuovere l'accesso alle informazioni nel mondo, all'interno della quale sviluppa e supporta strumenti per l'automazione dei test e sistemi di gestione documentale per il Web. Sebbene il suo interesse principale sia sempre stato lo sviluppo del software, egli ha anche un interesse di vecchia data per le attività di test e misurazione del software e di controllo qualità, avendo presentato saggi su questi argomenti in conferenze negli Stati Uniti ed in Europa. Ha ottenuto tre certificazioni, Certified Quality Analyst, Certified Software Quality Engineer, Lead Ticket Auditor, ed ha gestito le operazioni di test in OCLC durante il loro lavoro, durato tre anni e conclusosi positivamente, volto ad ottenere la certificazione ISO9000. Tip può essere contattato all'indirizzo house@oclc.org.
**Carol Wade**, "technical writer" per più di vent'anni, ha lavorato principalmente nel campo del software, scrivendo documentazione per gli utenti finali. Per nove anni ha lavorato per il Los Alamos National Laboratory, svolgendo sia funzioni di autore che di redattore ed avviando un servizio di traduzioni. Attualmente è l'unico "technical writer" che lavora per la Health Language Inc., che ha prodotto il primo "language engine" per il servizio sanitario.

**Luigi Caso**, Laurea nel 1989 all'Università di Salerno in Scienze dell'Informazione. Attualmente lavora presso la Delos S.p.A., società del gruppo Getronics, come SW Engineer. Certificato Microsoft, ha partecipato e partecipa, come Team Leader, alle fasi di analisi, progettazione ed implementazione di progetti SW in diversi ambiti (Gestione Documentale, Banking, Help Desk/CRM) (luigi.caso@getronics.com).

 