REQUISITO PERFETTO:

i 10 Comandamenti

LA PRIMA AZIONE PER METTERE IN SICUREZZA IL TUO SOFTWARE

Se tu dovessi decidere di fare un primo passo, uno solo, per incominciare a mettere in sicurezza il tuo software, da cosa dovresti partire? Quale potrebbe essere la singola azione più redditizia da mettere in pratica immediatamente per avere dei risultati evidenti soprattutto nel medio e lungo periodo e consolidati che portino ad un evidente e misurabile miglioramento della qualità del codice prodotto?

Te lo racconto a partire da una storia vera che mi è successa tanti anni fa, all'inizio della mia carriera lavorativa, ma che somiglia a tantissime altre storie simili che sicuramente saranno successe a te e a tutti coloro che si occupano di software.

Stavo sviluppando un piccolo pezzo di programma per un cliente, che metteva in collegamento due tool diversi, un "bridge" diciamo. Rilasciata una prima versione del software, ho cominciato a ricevere da parte del cliente una serie di email che mi segnavano alcune anomalie. All'inizio analizzavo le segnalazioni ed apportavo prontamente delle correzioni. Benissimo, il codice comunque funzionava bene e faceva già da subito il suo "dovere". Ma non bastava.

Le email hanno cominciato ad accavallarsi e a dimostrarsi strumento poco adatto a gestire la situazione: non era chiaro quali erano i problemi aperti e quelli risolti, in che versione erano stati risolti, per cui si è passati ad un sistema rudimentale di Bug Tracking, basato su Excel, su cui tornerò. Questo accorgimento ha migliorato molto la situazione, anche se non era finita così.

Si procedeva comunque abbastanza bene fino a quando ad un certo punto, si sono cominciate a creare delle situazioni non proprio piacevoli, anche se molto note: il cliente ha cominciato a lamentarsi non solo di funzionalità non presenti o che funzionavano male, ma anche di aspetti tutto sommato estetici o del tutto secondari. Anche qua sembrava risolto, al momento.

Alla fine, sono fioccate richieste che seppur apparentemente logiche, del tipo: *i file di log devono avere il formato YYYY_MM_DD_HH_SS.log invece che HH_SS_DD_MM_YYY.log* e decine simili che erano sicuramente ragionevoli, ma che avevano in comune con tantissime altre una caratteristica:

NESSUNO LE AVEVA CONCORDATE PRIMA!

Stava succedendo una cosa **MOLTO** spiacevole: il cliente mi tempestava di richieste, diligentemente divise per priorità, catalogate e condivise in un sistema di tracciamento errori, ma la verità era una soltanto.

IL CLIENTE SE NE STAVA APPROFITTANDO CHIEDENDO FUNZIONALITA' ED ASPETTI LOGICI ED ESTETICI CHE NON AVEVAMO STABILITO E CONCORDATO IN ALCUN MODO

Ed era una cosa apparentemente senza fondo: sono dovuto andarmi a riguardare le email scambiate all'inizio ma di formale, di scritto a livello di dettagli c'era ben poco. Un capitolato di poche decide di righe con le funzionalità di alto livello e basta. Quasi tutto il dettaglio era stato uno scambio verbale non registrato da nessuna parte. E il cliente ne stava approfittando.

Ma la stessa cosa capita a parti invertite, stai tranquillo: a volte sicuramente eri tu il cliente e hai litigato con i fornitori che ti hanno fornito un prodotto che secondo te era incompleto, malfunzionante, lacunoso ma non avevi a disposizione argomenti solidi per contestare quanto secondo te mancava.

Altre volte mi sono capitate cose simili senza necessariamente cattiva fede: semplicemente, non avendo messo nero su bianco quello che il cliente o committente esterno ma anche interno vogliono, si creano dei profondi disallineamenti tra richiesta da una parte e offerta dall'altra.

Quante volte ti è capitata una cosa del genere? Quanti di queste fraintendimenti, ambiguità, contraddizioni, estensioni non richieste delle funzionalità, capitano sia con aziende esterne che a volte all'interno della stessa azienda, tra team diversi? O addirittura si creano scollamenti tra manager e sviluppatori o tra designer e coloro che vengono dopo?

Tutto questo ha una causa ben precisa, che gli americani riassumono in uno dei loro soliti acronimi brevissimi e fulminanti

GIGO

(Garbage In, Garbage Out)

Tradotto: la Qualità di qualunque processo, qualunque esigenza, qualunque lavorazione, qualunque ricetta dipende sicuramente dal metodo, dalla competenza, dalla professionalità ma in primis dipende dai dati in ingresso, dagli "ingredienti", dall'input.

E questo diventa un principio fondamentale, valido in una qualunque delle fasi del Ciclo di Vita del Software:

se la qualità dei dati in ingresso di un processo è pessima, i risultati in uscita NON possono essere che altrettanto scarsi!

Io posso avere il miglior cuoco del mondo e la cucina più attrezzata: ma se uso uova marce, farina infestata, latte scaduto, ingredienti avariati come posso pensare di ottenere in uscita qualcosa di commestibile?

Se discuto delle caratteristiche di un prodotto, software, attività con una serie di email confuse, riunioni senza un report, scambi al telefono come posso pensare di farmi capire correttamente dal mio cliente o dal mio fornitore?

Ora posso rispondere alla domanda iniziale: qual è la tua prima mossa che DEVI implementare per iniziare un rapido, efficace e duraturo processo di miglioramento della Qualità del Software?

I REQUISITI!

Esatto: senza dei requisiti adatti, non può nemmeno esistere il concetto di un software che sia:

- **corretto:** che si comporti come da richiesta del cliente o come pubblicizzato, in maniera formale e dimostrabile
- **completo:** in cui siano presenti e funzionanti correttamente tutte le funzionalità desiderate o offerte

- **tracciabile**: che contenga SOLO le caratteristiche descritte nei requisiti e che non abbia al suo interno funzionalità obsolete, nascoste o inattive che possano improvvisamente attivarsi creando funzionamenti indesiderati

- **verificabile**: per dimostrare a qualunque cliente, committente o commissione che ogni singola funzionalità, ogni requisito non solo è stato tracciato, ma è anche stato verificato e testato

- **ripetibile**: se hai un ottimo set di requisiti, potresti dare il compito di realizzare lo stesso software a team o aziende diverse e avresti dei risultati funzionalmente del tutto equivalente

Basare tutto il ciclo di vita della produzione del software sui Requisiti è un passo assolutamente fondamentale per la rimozione dell'ambiguità, per la definizione certa dello stato di avanzamento di un progetto, per avere delle interfacce certe e formali con clienti e fornitori, per rendere tutto lo sviluppo di un prodotto contenente software un processo stabile, certo, ripetibile quindi in poche parole **ingegneristico**.

Perciò se devi puntare a *una singola azione volta migliorare il tuo processo di sviluppo, a creare una cultura adeguata del software, a rivoluzionare il tuo modo di lavorare riducendo enormemente i rischi e abbassando i costi, devi capire bene cos'è un* ***REQUISITO*** e come si scrive correttamente (e come NON si scrive).

Ma come si scrivono correttamente i requisiti? Il **METODO PER UN EFFICIENTE SVILUPPO DEL SOFTWARE (M.E.D.S.)** prevede che i requisiti abbiano ben 10 caratteristiche fondamentali per essere utili ed efficaci e non diventino un'arma a doppio taglio. Perché i requisiti siano davvero uno strumento potente che vada ad accrescere la Qualità e al contrario non aumentino la confusione, l'ambiguità e non si trasformino in un'attività che viene percepita come noiosa, inutile e controproducente per cui abbandonata dopo alcuni tentativi iniziali, è indispensabile seguire alcune regole ed evitare degli errori comuni.

I dieci comandamenti del requisito perfetto (o quasi)

Si parla quindi di un metodo per migliorare almeno del 50-60% (misurabile) la qualità, l'efficienza, i costi di un software e a ridurre al minimo il rischio di interventi successivi, modifiche, richiami dal campo, rifiuti da parte del cliente.

Miracolo? No, **DURO LAVORO DI ANALISI E PROGETTAZIONE.**

Come vedrai fra poco, partiremo da un requisito di altissimo livello quasi innocuo, che sembra molto tranquillo e non particolarmente critico. Ma ci accorgeremo subito, esasperando un po' ovviamente la situazione, come si trasformerà in una ramificazione di requisiti e di considerazioni.

LE 10 CARATTERISTICHE FONDAMENTALI DI UN BUON REQUISITO:

1) **IDENTIFICABILE:** un requisito deve essere identificato con precisione da un numero e/o una sigla, che siano del tutto univoci e che non diano adito alla minima confusione. Deve essere

distinto se è un requisito di Sistema, se è Software o Hardware o si applica a entrambi, se è di Alto o Basso Livello e via discorrendo.

2) **FORMALE**: un requisito deve essere scritto in un linguaggio il più possibile formale, senza ambiguità, il soggetto deve essere sempre e solo il software con una sola formula "...DEVE/ SHALL...", senza imprecisioni nel testo, senza assunzioni e preconcetti, senza dare nulla per scontato a priori, in modo che chiunque lo legga con formazione adeguata possa implementarlo, anche e soprattutto se è un committente esterno e non un team aziendale. L'utilizzo di un linguaggio matematico formale o di un linguaggio grafico semi-formale (come UML, SysML, MARTE, ...) aiuta moltissimo la comprensione, soprattutto a livello di team internazionali dove il linguaggio può essere una barriera

3) **INTERFACCIABILE**: di ogni requisito, soprattutto se di basso livello ma non solo, vanno descritti accuratamente i suoi confini, le interazioni con il resto del sistema Hardware e Software, le sue interfacce, le condizioni al contorno, i cambiamenti indotti negli stati del sistema, tutte le variabili in ingresso e in uscita.

4) **TRACCIABILE**: deve esistere un sistema di tracciamento che parta dai requisiti del cliente, quelli di prodotto, del software/hardware/firmware e in alcuni casi come nelle certificazioni scenda fino a livello del codice e dei test e viceversa. È importante avere un tool formale per il tracciamento dei requisiti, perché oltre ad essere indispensabile per tenere sotto controllo lo sviluppo di un software e le conseguenze di un cambiamento, si rivela l'unico strumento di management fondamentale per capire il reale stato di avanzamento di un progetto.

5) **DERIVATO**: si definisce derivato un requisito che non proviene dalle fasi precedenti per cui non è tracciabile verso l'alto, ma è frutto delle decisioni ingegneristiche e di programmazione a livello software. Essendo un requisito nuovo non riconducibile a nessuna scelta precedente, ogni requisito derivato dovrà essere accuratamente valutato per le sue possibili conseguenze a livello di sistema e di safety.

6) **PRIORITIZZATO**: in caso di implementazione in più fasi, prototipazioni, ciclo a spirale o similare, varie fasi di immissione sul mercato ecc. deve essere chiara la priorità di implementazione e realizzazione dei vari requisiti, in modo che tutto sia pronto solo al momento giusto e non prima o peggio dopo

7) **VALIDABILE:** le definizioni formali di Validazione e Verifica si accavallano e contraddicono tra le varie discipline; io adotto quella della DO-178 per cui la fase di Validazione decide se i requisiti sono realmente fattibili, realizzabili, completi, coerenti, non sovrapposti e non lacunosi, implementabili nei tempi e con le tecnologie disponibili e coerenti tra loro

8) **VERIFICABILE**: bisogna sempre avere in mente, fin dall'inizio, un test, una procedura, una verifica finale che il requisito nelle ultime fasi del ciclo di vita sia implementato correttamente e funzioni come previsto. Caratteristiche, effetti nel sistema, elaborazione degli ingressi ed uscite, precisione, performance, aspetto visivo o temporale di ogni requisito devono essere specificati in maniera che esista il modo di verificarli.

9) **REVISIONATO**: è molto semplice, fidarsi è bene, non fidarsi è meglio. Il lavoro di una persona che lavora da sola è quasi sempre pieno di ***assunzioni*** (se non l'hai ancora letto, ti consiglio il libro: "The mithycal man month", di F. Brooks), ossia la pretesa che gli altri abbiano la nostra

stessa visione, cultura e capacità analitica e capiscano da soli le pre-condizioni, i vincoli, i retroscena, le cose "scontate", le assunzioni quindi che ho in testa quando scrivo un requisito o qualunque altra cosa. Ragion per cui è INDISPENSABILE una review indipendente di tutto quello che si fa.

10) **STABILE**: ecco l'ultima caratteristica e una delle più importanti. La modifica di un requisito, una volta approvato e "congelato", da parte del cliente o di qualunque committente, deve essere un evento eccezionale, regolamentato, preceduto da un'accurata fase di revisione, di analisi di impatto, di modifica non solo dei requisiti ma di tutti quelli correlati e da un'adeguata fase di implementazione ma soprattutto di test

Già solo questo approccio, come descritto, garantisce di per sé un formalismo, un rigore, una struttura mentale e pratica nell'approccio al software che ti cambia radicalmente in meglio il modo di lavorare, ti migliora il rapporto con il cliente, ti garantisce un maggiore rispetto dei tempi e delle performance e la riduzione dei tuoi potenziali problemi a breve ma soprattutto a lungo termine.

È un approccio complicato? ***SI***

È un investimento notevole di tempo? ***SI***

Porta via risorse inizialmente nel progetto impedendo di iniziare a lavorare subito sul codice? ***SI***

Ingessa i rapporti con i fornitori che non possono più facilmente fare i capricci cambiando idea un giorno sì e un giorno no? ***SI***

Perfetto: allora è **ESATTAMENTE** quello che ci vuole, quello che **DEVI** fare. Anche se è complicato e dispendioso, è tale il miglioramento ed il risparmio in termini di costi, qualità ed effetti collaterali futuri che il Ritorno dall'Investimento (ROI) è elevatissimo, si parla di un fattore 10-20.

E proprio per questo motivo è una mossa talmente potente e risolutrice che, di tutti gli aspetti importanti per uno sviluppo rigoroso, che inizialmente per un'azienda normale possono essere sovrabbondanti, è una delle prime in assoluto da adottare qualunque sia il tipo d'industria e clienti che tu abbia.

Per informazioni e per acquistare il video-corso completo Requisito perfetto:

https://www.softwaresicuro.it/Mautic/requisitoperfetto

In ogni caso, per essere chiari: il **METODO PER UN EFFICIENTE SVILUPPO DEL SOFTWARE (M.E.D.S.)** raccomanda a TUTTI, compreso a te lettore, un'adozione integrale, seppur progressiva, dei 10 punti stabiliti sopra. Mi ringrazierai poi!

Per informazioni sul metodo M.E.D.S.:

https://www.softwaresicuro.it/Mautic/corsopubblicomeds2019

*Infatti, nel mio libro **"SOFTWARE SICURO"** ti dirò perché DEVI iniziare quanto prima ad adottare questo decalogo. E perché **potrebbe fare la differenza per la sopravvivenza della tua azienda o team**.*

https://www.softwaresicuro.it/software/cultura/software-sicuro-il-libro/