REQUISITO PERFETTO

Guida pratica + Caso di Studio

Il videocorso ***"REQUISITO PERFETTO"*** rappresenta per te un metodo per migliorare almeno del 50-60% (quantificabile!) 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.**

E non sarà solo teoria: come vedrai nel Caso di Studio, 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.

Guida pratica in 10 passi

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. E' 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'avete letto vi consiglio il libro: "The mithycal man month", di F. Brooks), ovvero 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 maggiori rispetto dei tempi e delle performance e la riduzione dei tuoi potenziali problemi a breve ma soprattutto a lungo termine.

E' un approccio complicato? ***SI***

E' 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 dei primi in assoluto da adottare, qualunque sia il tipo d'industria e clienti che tu abbia.

Per essere chiari: il ***METODO PER UN EFFICIENTE SVILUPPO DEL SOFTWARE (M.E.D.S.)*** raccomanda TUTTI, compreso tu lettore, un'adozione integrale, seppur progressiva, dei 10 punti stabiliti sopra.

Guardati questa pagina intanto, per quanto riguarda il M.E.D.S.:

https://www.softwaresicuro.it/MEDS

Mi ringrazierai poi!

Intanto, se hai pazienza di leggere fino in fondo, o se sei curioso e vuoi leggere direttamente le conclusioni, alla fine... 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****.*

Caso di Studio

Vediamo adesso in pratica come trasformare un requisito scritto male, attraverso varie passate e toccando un po' tutti i punti descritti sopra, in un requisito quasi perfetto.

Questo approccio è proprio esemplificativo di come si potrebbe prima di tutto prendere i requisiti esistenti e riscriverli in maniera più precisa, formale, non ambigua, testabile e così via come descritto nei punti fondamentali.

Soprattutto è un ottimo esercizio per incominciare a pensare fin dall'inizio in un modo che ci garantisca il rispetto dei capisaldi della scrittura dei requisiti, che è la cosa più distante in assoluto dalla scrittura creativa che a volte sembra abbondare nei pasticci di documenti che ti vedi spesso recapitare dal tuo cliente.

A volte è quasi meglio che sia TU a riscrivere i requisiti del tuo cliente, facendo quasi del lavoro per lui, ma quest'attività di revisione potrebbe rivelarsi fondamentale nel capire mancanze, lacune, contraddizioni e imprecisioni che è totalmente nel TUO interesse evidenziare e scoprire il prima possibile.

Vediamo un esempio di requisito scritto MOLTO MALE e che in varie passate andremo a sgrezzare prima con lo scalpello e poi pian piano a levigare:

L'ALGORITMO DI ORDINAMENTO DEVE ORDINARE ALFABETICAMENTE UN VETTORE IN UN TEMPO RAGIONEVOLE OTTIMIZZANDO IN UN SECONDO MOMENTO LE PRESTAZIONI E L'OCCUPAZIONE DI MEMORIA. È IMPORTANTE CHE L'ACCESSO AL VETTORE ORDINATO RISULTANTE SIA SUFFICIENTEMENTE RAPIDO E CONSENTA RAPIDE RICERCHE SIA PER INDICE CHE IN FUTURO PER HASH.

Allora, vediamo uno per uno tutti i punti precedenti e proviamo ad applicarli, senza pretesa di esaurire completamente l'argomento e di essere sempre rigorosi al 100%, perché l'argomento integrale richiederebbe settimane di lavoro, ma mostrando quanto più possibile il metodo corretto in termini pratici.

IDENTIFICABILE

Manca completamente una numerazione breve ed univoca, l'informazione sintetica se il requisito è di basso o alto livello, se riguarda un'implementazione hardware o software e via discorrendo. Riproviamo:

SW_HLR_001: *L'ALGORITMO DI ORDINAMENTO DEVE ORDINARE ALFABETICAMENTE UN VETTORE IN UN TEMPO RAGIONEVOLE OTTIMIZZANDO IN UN SECONDO MOMENTO LE PRESTAZIONI IN MEMORIA. È IMPORTANTE CHE L'ACCESSO AL VETTORE ORDINATO RISULTANTE SIA*

SUFFICIENTEMENTE RAPIDO E CONSENTA RAPIDE RICERCHE SIA PER INDICE CHE IN FUTURO PER HASH.

Ora abbiamo se non altro aggiunto un identificativo univoco che ci dice:

SW: è un requisito software

HLR: è un requisito di alto livello

001: numerazione progressiva

FORMALE

Qua è un totale disastro. Questo requisito è tutt'altro che formale, vogliamo vederne tutti i difetti?

- il soggetto è l'algoritmo: a meno che tutto il software si identifichi con l'algoritmo stesso, cosa improbabile, si deve cambiare l'ordine della frase. Un'ottima consuetudine della scrittura del ***Requisito Perfetto*** è quella di avere ***sempre lo stesso soggetto (il Software, il Sistema, l'Algoritmo, ecc.), una parola chiave "DEVE/DOVRÀ" e un solo punto o caratteristica per requisito***
- è tutt'altro che preciso: termini del tutto ambigui (per cui ***VIETATI***) come *"ragionevole", "efficiente", "ottimizzando", "rapido", "sufficiente"* e ***qualunque altro termine non linguisticamente o scientificamente preciso sono del tutto banditi***
- ci sono due o più requisiti distinti mischiati in uno: si parla di algoritmo di ordinamento e di metodi di accesso al vettore successivi. Questi sono due requisiti distinti.

Sicuramente potremo riscriverlo così, scindendolo in tre requisiti, di cui seguiremo solo uno:

SW_HLR_001A: *L'ALGORITMO* ***DEVE*** *IMPLEMENTARE UN ALGORITMO DI ORDINAMENTO ALFABETICO DI UN VETTORE, DA COMPLETARE NEL CASO PEGGIORE IN UN TEMPO DI 200 MICROSECONDI, OTTIMIZZANDO IN UN SECONDO MOMENTO LE PRESTAZIONI DI ORDINAMENTO DI ALMENO IL 50% E DI MEMORIA DELL'80%.*

SW_HLR_001B: *L'ALGORITMO* ***DEVE*** *GARANTIRE L'ACCESSO AL VETTORE ORDINATO USCENTE DALL'ALGORITMO DI ORDINAMENTO IN UN TEMPO INFERIORE AI 5 MICROSECONDI.*

SW_HLR_001C: *L'ALGORITMO* ***DEVE*** *CONSENTIRE RICERCHE SIA PER INDICE CHE IN FUTURO PER HASH.*

INTERFACCIABILE

Quali sono le interfacce in ingresso e in uscita? Gli stati del sistema? Non se ne parla minimamente, difatti non ho idea se:

- i dati di ingresso sono numerici, alfanumerici, con che codifica?
- gli elementi sono totalmente disordinati o posso fare delle assunzioni per velocizzare l'ordinamento?
- gli elementi di che dimensione sono, di che numero minimo e massimo?
- come sono organizzati i dati: si tratta di un vettore, una matrice, una lista puntata? La memoria è allocata prima?
- la struttura dati in uscita è la stessa, o una nuova? Devo allocare della memoria in uscita?
- cambia qualche stato o sottostato quando il vettore è ordinato e quando non lo è più?

Adesso scendiamo come livello di dettaglio implementativo, è un buon momento per passare dai requisiti di alto a quelli di basso livello.

SW_LLR_001A_001: *L'ALGORITMO* ***DEVE*** *RICEVERE UN VETTORE IN LINGUAGGIO C ALLOCATO GLOBALMENTE COL NOME* ***VECTOR_DATA_CONTENT****, CONTENTENTE STRINGHE ALFANUMERICHE NON ORDINATE, CODIFICATE ISO-85591, NULL-TERMINATED, DI DIMENSIONE MASSIMA 255 CARATTERI PIÙ NULL FINALE. IL NUMERO MINIMO È DI 0 ELEMENTI, IL NUMERO MASSIMO È DI 65.535 ELEMENTI. TUTTO IL VETTORE È COMPLETAMENTE ALLOCATO STATICAMENTE PRIMA DELLA CHIAMATA DELL'ALGORITMO DI ORDINAMENTO.*

SW_LLR_001A_002: *L'ALGORITMO* ***DEVE*** *RESTITUIRE LO STESSO VETTORE RICEVUTO IN INGRESSO ORDINATO ALFABETICAMENTE SENZA CONSIDERARE LA DIFFERENZA FRA MAIUSCOLE E MINUSCOLE*

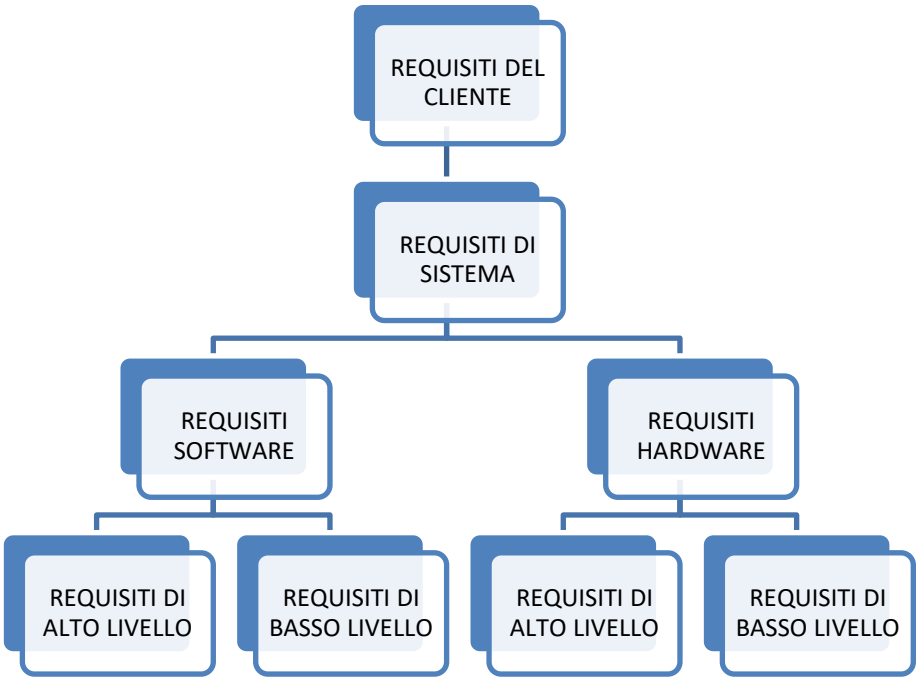
SW_LLR_001A_003: *L'ALGORITMO* ***DEVE*** *SUBITO VERIFICARE LA VARIABILE GLOBALE* ***DIRTY_VECTOR*** *E USCIRE SENZA FARE NULLA SE È SEGNATA COME* ***SORTED,*** *ALTRIMENTI DEVE FLAGGARE A* ***SORTING*** *PER SEGNALARE L'ATTIVITÀ DI ORDINAMENTO IN CORSO E A* ***SORTED*** *QUANDO HA TERMINATO COMPLETAMENTE*

SW_LLR_001A_004: *L'ALGORITMO* ***DEVE*** *COMPLETARE UN ORDINAMENTO COMPLETO DI UN VETTORE CASUALE TOTALMENTE DISORDINATO NEL TEMPO MASSIMO DI 200 MICROSECONDI.*

Ora va meglio, ma ci siamo già ritrovati 4 requisiti! Ma mica è finita...

TRACCIABILE

Qui invece dobbiamo costruire una matrice di tracciabilità. Dobbiamo in teoria seguire quest'ordine e fornire una tracciabilità bi-direzionale, dall'alto verso il basso:



Immaginiamo nel nostro caso che ci siano dei requisiti del cliente e di sistema:

CUS_001: IL SISTEMA DEVE RICEVERE CONTINUAMENTE NUOVI DATI IN INGRESSO RELATIVI ALLE CONNESSIONI APERTE E MANTENERLI ORDINATI ALFABETICAMENTE IN MANIERA TRASPARENTE ALL'OPERATORE, SENZA ALTERARE LE COMPONENTI CRITICHE DEL SISTEMA

Il requisito è ovviamente molto generico, essendo del cliente. Non sappiamo nulla di come arriveranno questi dati, in che formato, con che velocità. Non sappiamo neanche quali sono e cosa vuol dire alterare le componenti critiche del sistema: dovrebbe essere dato un tempo minimo di risposta garantito all'operatore! Perciò dovremmo guardare se ci siano altri requisiti cliente che ce lo spieghino, oppure prendere delle decisioni operative e documentarle (requisiti DERIVATI, lo vedremo dopo).

SYS_001: IL SISTEMA DEVE IMPLEMENTARE UN ALGORITMO DI ORDINAMENTO ALFABETICO DEI DATI ALFANUMERICI IN INGRESSO, RELATIVI ALLE CONNESSIONI APERTE, PROVENIENTI DA UN PROTOCOLLO SERIALE CHE PUÒ INVIARE 100 NUOVI ELEMENTI AL SECONDO DA INSERIRE NEL VETTORE PRE-ORDINATO, SENZA COMPROMETTERE L'USABILITÀ E LE PRESTAZIONI CRITICHE DEL SISTEMA.

Qualcosa di più ora lo sappiamo, o lo abbiamo deciso noi. Sappiamo che tipo di dati è, la frequenza di arrivo, ancora non ci è chiaro quali sono le funzionalità critiche e che tempi di risposta devono avere.

Questo è MOLTO importante da sapere a questo punto, perché noi ci troviamo davanti ad un bivio: DOVE implementare queste funzionalità? Potremmo implementarle:

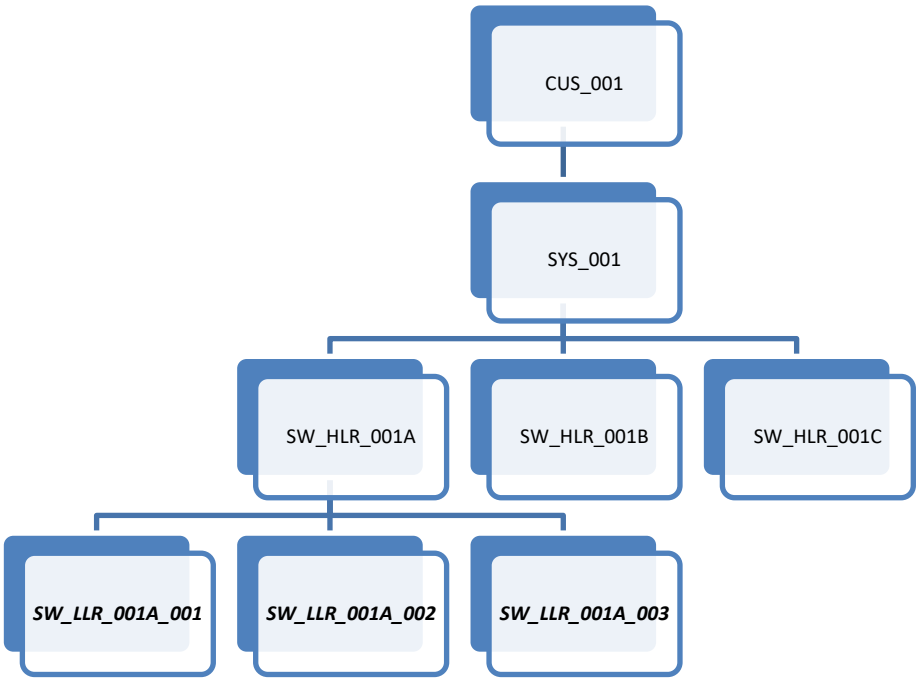
- in HARDWARE: potrebbe essere necessario, visti i tempi stretti, implementarlo in hardware magari usando una FPGA, un ASIC e un linguaggio VHDL

- in SOFTWARE: ovviamente potremmo usare un microcontrollore, una ECU, ma a quel punto dovremmo essere sicuri che le performance della CPU siano adeguate a garantire sia le funzionalità critiche sia quelle di ordinamento, appunto

Noi sceglieremo la seconda opzione, ovviamente dopo che avremo fatto in teoria un'analisi prestazionale. In ogni caso dobbiamo come minimo prevedere un requisito hardware in cui specificheremo le performance del microcontrollore, che poi andremo a riempire con i dati dopo opportuna analisi e simulazione durante la validazione:

HW_HLR_001: L'HARDWARE **DEVE** GARANTIRE UNA PERFORMANCE DI CALCOLO DI ALMENO XXX MFLOPS IN MODO DA GARANTIRE L'ESECUZIONE DI FUNZIONALITÀ CRITICHE CON TEMPI DI RISPOSTA DI YY SECONDI ANCHE NEL CASO PEGGIORE DI ARRIVO CONTINUO DI DATI DA ORDINARE ALLA MASSIMA VELOCITÀ.

Nella sezione successiva daremo una risposta a queste domande, nel frattempo noi ci concentreremo sulla tracciabilità per cui per il momento avremo una cosa di questo tipo, da aggiornare poi di volta in volta che aggiungiamo nuovi requisiti:



DERIVATO

Cos'è un requisito derivato? Come accennato, si tratta di requisiti che... al contrario di quello che farebbe intuire il nome, non sono "derivati" dai requisiti precedenti (questa è la TRACCIABILITÀ) ma sono derivati da scelte ingegneristiche, progettuali, prestazionali non chiaramente specificate e individuabili nei requisiti di più alto livello.

Ad esempio: è chiaro cosa deve fare il nostro algoritmo di ordinamento, i dati in ingresso e uscita, le prestazioni. Nessuno però ci ha detto ad esempio che algoritmo dobbiamo usare: Bubblesort? Quicksort? Qual è la complessità computazionale e a che ritmo cresce: col quadrato degli elementi, linearmente, o come? E quanto influenza le prestazioni? Ovvio che la decisione potrebbe essere presa ai livelli più bassi direttamente dal programmatore, ma che garanzia avremmo a quel punto che il resto dei requisiti sia rispettato? Magari il programmatore ha le sue preferenze, è un team o un'azienda esterna addirittura, per cui va possibilmente guidato. In ogni caso, la fase di validazione che vedremo nel punto seguente è indispensabile per la determinazione dei requisiti derivati.

Non sempre è facile definire se un requisito è derivato o è semplicemente tracciabile: qua non viene specificato il tipo di algoritmo a nessun livello, certo la decisione potrebbe essere tracciabile da elementi come le prestazioni richieste, la velocità della CPU a disposizione, la complessità computazionale, il numero degli elementi ecc. ma in questo caso noi preferiamo fare un nuovo requisito derivato di alto livello per l'hardware, da raffinare eventualmente nelle fasi successive:

HW_DER_001: L'HARDWARE **DEVE** adottare una CPU della famiglia ARM Core preferibilmente M3-M4 con almeno 35 Ghz di clock, 4 ADC + 4 DAC, ... almeno 512Kbytes di RAM 8 bit, 16 MBytes di FLASH.

PRIORITIZZATO

Dobbiamo definire delle priorità d'implementazione del nostro sistema, perché se da un lato vogliamo ottenere la massima stabilità, dall'altro la spinta a essere presto sul mercato, a fornire dei prototipi al cliente, ad avviare delle fasi di test preliminari ecc. richiedono che tu faccia diverse iterazioni prima di arrivare al prodotto finale, così come anche raccomandato da tecniche moderne come AGILE.

Questo non vuol dire di dover fare delle iterazioni a caso, con rilasci decisi di volta in volta in base all'ispirazione e a quello che è pronto in quel momento: vuol dire pianificare correttamente il numero di release intermedie e assegnare esattamente diversi requisiti a ogni fase o comunque versioni diverse dello stesso requisito.

Prendiamo il nostro requisito iniziale, quello scritto sicuramente male, in cui ci sono due frasi sotto accusa:

- *OTTIMIZZANDO **IN UN SECONDO MOMENTO** LE PRESTAZIONI E L'OCCUPAZIONE DI MEMORIA*
- *CONSENTA RAPIDE RICERCHE SIA PER INDICE CHE **IN FUTURO** PER HASH*

Perciò, dobbiamo sicuramente decidere delle iterazioni o release e assegnare la realizzazione di ogni dettaglio nel giusto orizzonte temporale. Chiameremo **R1, R2** e **R3** le nostre release e aggiungeremo un prefisso adeguato ai nostri requisiti che a questo punto saranno diversi per ogni versione.

Questa sarà la versione della prima release in cui la memoria è allocata staticamente e le prestazioni di ordinamento sono lineari con la dimensione attuale dell'array:

SWR1_LLR_001A_001: *L'ALGORITMO* ***DEVE*** *RICEVERE UN VETTORE IN LINGUAGGIO C ALLOCATO GLOBALMENTE COL NOME* ***VECTOR_DATA_CONTENT****, CONTENTENTE STRINGHE ALFANUMERICHE NON ORDINATE, CODIFICATE ISO-85591, NULL-TERMINATED, DI DIMENSIONE MASSIMA 255 CARATTERI PIÙ NULL FINALE. IL NUMERO MINIMO È DI 0 ELEMENTI, IL NUMERO MASSIMO È DI 65.535 ELEMENTI. TUTTO IL VETTORE È COMPLETAMENTE ALLOCATO STATICAMENTE PRIMA DELLA CHIAMATA DELL'ALGORITMO DI ORDINAMENTO. LA COMPLESSITÀ COMPUTAZIONALE PUÒ INIZIALMENTE ESSERE LINEARE.*

Nella seconda iterazione invece si passa ad allocazione dinamica e a un algoritmo le cui prestazioni non crescono linearmente ma in maniera ottimizzata:

SWR2_LLR_001A_001: *L'ALGORITMO* ***DEVE*** *RICEVERE UN VETTORE IN LINGUAGGIO C ALLOCATO GLOBALMENTE COL NOME* ***VECTOR_DATA_CONTENT****, CONTENTENTE STRINGHE ALFANUMERICHE NON ORDINATE, CODIFICATE ISO-85591, NULL-TERMINATED, DI DIMENSIONE MASSIMA 255 CARATTERI PIÙ NULL FINALE. IL NUMERO MINIMO È DI 0 ELEMENTI, IL NUMERO MASSIMO È DI 65.535 ELEMENTI. IL VETTORE È ALLOCATO DINAMICAMENTE DALL'ALGORITMO STESSO, TRAMITE LISTA LINKATA SEMPLICE. LA COMPLESSITÀ COMPUTAZIONALE DEVE ESSERE MIGLIORE RISPETTO A QUELLA LINEARE.*

Nella terza release, andremo a specificare di usare una lista e un meccanismo di ordinamento molto efficiente:

SW_LLR_001A_001: *L'ALGORITMO* ***DEVE*** *RICEVERE UN VETTORE IN LINGUAGGIO C ALLOCATO GLOBALMENTE COL NOME* ***VECTOR_DATA_CONTENT****, CONTENTENTE STRINGHE ALFANUMERICHE NON ORDINATE, CODIFICATE ISO-85591, NULL-TERMINATED, DI DIMENSIONE MASSIMA 255 CARATTERI PIÙ NULL FINALE. IL VETTORE È ALLOCATO DINAMICAMENTE DALL'ALGORITMO STESSO, TRAMITE LISTA LINKATA DOPPIA. LA COMPLESSITÀ COMPUTAZIONALE DEVE ESSERE PARI O MIGLIORE DI QUELLA LOGARITMICA.*

VALIDABILE

I requisiti sono realizzabili, descrivono completamente le funzionalità da realizzare in tutti gli aspetti, senza ripetizioni e lacune? Sono implementabili con le tecnologie attuali?

Sicuramente dovremmo avere sott'occhio tutti gli altri requisiti per sapere meglio se ci sono ripetizioni, lacune o contraddizioni. Quello che possiamo chiederci sulla realizzabilità tecnologica delle performance, guardando questo singolo algoritmo, ad esempio è:

- abbiamo sufficiente memoria per memorizzare tutto l'array da 65.536x256? Non è in teoria compito di chi scrive l'algoritmo perché sono dati già inizializzati staticamente, ma un occhio globalmente alla memoria da qualcuno va dato. Sicuramente nella prima release del

software ci saranno problemi di occupazione di memoria, ma questo può essere inizialmente accettato, se ben pianificato. Poi interverranno i miglioramenti adeguati nelle release successive, l'importante è che sia definita la priorità dei requisiti stessi

- la CPU è sufficiente per effettuare l'ordinamento nel tempo richiesto? In teoria sì, sulla carta dovrebbe essere possibile, ma le domande potrebbero essere: ci sono altri task? Quanta % di CPU è disponibile in media? E se sono in un momento di picco di utilizzo della CPU, di quanto si allungano i tempi? Stesso discorso di sopra, la fase di validazione con analisi e simulazione, nonché la prima release, serviranno come prototipo per capire bene come procedere

Ci vengono sicuramente in aiuto, per la fase di validazione, alcune metodologie piuttosto note e utilizzate:

1) **MODELLAZIONE:** costruire un modello del mio sistema, prodotto, funzionalità è una delle tecniche che di recente si sono affermate e che consentono effettivamente di anticipare tantissimo la fase di revisione e validazione dei requisiti. Sono tanti i linguaggi di modellazione, da quelli standard (tipo ***UML,SysML, MARTE***) a quelli proprietari. Unita alla simulazione, consente di effettuare tantissime verifiche di validità di quanto si sta per progettare e mettere in pratica
2) **SIMULAZIONE:** oramai, vista la complessità di una lunghissima serie di prodotti, di funzionalità, di algoritmi ecc. è diventato praticamente obbligatorio far precedere qualunque fase di sviluppo da una di ricerca approfondita tramite la simulazione (preceduta e accompagnata sempre dalla modellazione). Tramite una nutrita serie di tool oramai sempre più potenti, si possono simulare una serie di situazioni, di scenari, di test che non sono soltanto studiati a tavolino ma spesso sono presi dalla realtà tramite misurazioni e telemetria, per poi essere applicati da un simulatore su computer. Questo taglia in maniera drastica tempi, costi e rischi dei test, soprattutto anticipandoli alle fasi preliminari del progetto
3) **PROTOTIPAZIONE:** anche la classica attività di costruire dei prototipi è un ottimo modo per anticipare l'individuazione e la risoluzione dei problemi, limitando i rischi. Con la consapevolezza che esisteranno delle differenze anche sostanziali tra prototipo e produzione finale, magari in serie, un approccio a più fasi, ***se ben gestito***, è un eccellente metodo di validare i requisiti preliminarmente.

Nel nostro caso, il fatto di far modellare questo algoritmo di ordinamento con un tool es. come Simulink, oppure di farne un prototipo che gira su un'architettura diversa seppur simile, mi consentirebbe di rispondere ad alcune delle domande sulla realizzabilità tecnica e di scoprire tutta una serie di problemi, lacune, incoerenza e sovrapposizione dei requisiti, ancora prima di scrivere una linea di codice.

VERIFICABILE

La considerazione che si deve fare **ogni volta** che si pensa, si scrive o si fa la revisione di un requisito è la seguente:

POSSO SCRIVERE UN TEST CASE, UNA PROVA, UN'ANALISI, UNA VERIFICA DI QUESTO REQUISITO?

In poche parole: *POSSO VERIFICARLO*? E la risposta deve essere sempre: ***SI. Nel 100% delle volte.***

Un requisito che non può essere verificato, semplicemente NON è un requisito. Altrimenti:

- *in che modo posso mostrare al cliente che la funzionalità è effettivamente presente e risponde alle specifiche?*
- *come posso dimostrare di averla implementata correttamente?*
- *come faccio a dimostrare all'eventuale certificatore la copertura e verifica totale dei requisiti?*

Perciò, già prima di scrivere qualunque requisito, devo immaginare che un futuro Verificatore o Tester potrà agevolmente scrivere almeno due tipi di test:

- **CASO NORMALE:** quando tutto funziona alla perfezione. Gli input sono dati nel formato e range corretto, lo stato del sistema è valido, ci sono abbastanza risorse come memoria e CPU e via discorrendo. Sto testando la cosiddetta *"giornata di sole"*: il software fa, bene, correttamente tutto quello per cui è stato scritto.
- **CASO ANORMALE:** qui invece si testano tutte le condizioni non abituali, i cosiddetti *"giorni di pioggia"*. Stress test, verifica performance, casi anomali, ingressi fuori dal range consentito, valori non validi, messaggi fuori sequenza, stati anomali. Non sempre e non per tutti i software è indispensabile gestire questa categoria di test, ma in linea di massima è un'ottima strategia pensare già in quella nella direzione.

Nel nostro caso, abbiamo scritto piuttosto bene nella loro ultima estensione i requisiti in maniera che siano perfettamente testabili in casi normali. Ma in caso di stress, di sovraccarico della CPU, della memoria? Siamo sicuri di riuscire a rispettare i tempi imposti dai requisiti di sistema sempre e comunque?

Facciamo una bella cosa, introduciamo un nuovo requisito **DERIVATO**, perché non ci è stato chiesto prima, in cui decidiamo una strategia da adottare in caso il sistema sia sovraccarico:

SYS_DER_001: IL SISTEMA **DEVE** GESTIRE IL CASO DI CPU CHE RIMANGA OLTRE IL 90% DI UTILIZZO PER PIÙ DI XX SECONDI TRAMITE UN'AZIONE DI DEGRADO DELLE PRESTAZIONI O DEL SISTEMA (SOFT RESET DEL SISTEMA, PERDITA TEMPORANEA FUNZIONALITÀ SECONDARIE)

Non vogliamo andare nel dettaglio perché la gestione integrale del problema richiederebbe pagine e pagine di trattazione, ci limitiamo a livello di sistema di identificare la potenziale problematica evidenziandola a livello di requisito derivato, lasciando a iterazioni successive e/o a implementazioni hardware o software le reazioni a una situazione di sovraccarico o stress.

REVISIONATO

Un'altra fondamentale caratteristica del requisito perfetto: la *revisione*, o ***review***. L'attività di scrittura requisiti è talmente fondamentale, come abbiamo visto ampiamente finora, che non può e non deve essere affidata a una sola persona, per quando geniale possa essere. Per due semplici motivi:

- chi scrive un requisito, deve farlo in maniera che CHIUNQUE sia dotato dello stesso livello di esperienza e cultura deve poterlo capire. Non si può dare nulla per scontato, per cui un'altra persona dovrà andare a verificare come minimo tutti i rimanenti punti del decalogo qui presentato e in generale andrà a riscrivere o modificare il requisito in maniera che sia comprensibile a tutti
- chi ha scritto il requisito... prima o poi se ne andrà. Cambierà team, responsabilità o addirittura azienda. Il 90% abbondante dei bug sopravvive alla carriera aziendale di chi li ha inseriti e anche in un requisito può essere nascosto un "bug". Motivo per cui quando qualcun altro si troverà in futuro a implementare, testare o modificare un requisito, deve essere certo che sia stato soggetto a una revisione ***indipendente e formale***

Cosa vuol dire revisione ***indipendente***? Vuol dire semplicemente che deve essere effettuata da un'altra persona ma con il requisito che sia del tutto simile in termini di capacità, di competenza, di cultura di chi ha scritto il testo originale. Questo è un semplice rimedio che non costa nulla ed è facilmente implementabile da qualunque azienda, tranne proprio nei rari casi in cui si tratta di una sola persona (ma non ci vuole molto comunque a trovare qualcuno che ti dia una mano lo stesso).

Cosa vuol dire revisione ***formale***? Vuol dire che avrò utilizzato delle domande di controllo pre-configurate, chiamate ***checklist*****,** che serviranno a fare in modo che, chiunque si trovasse ad avere in mano il compito di rivedere questi requisiti, si troverebbe a riscontrare gli stessi identici problemi perché risponderebbe alle stesse domande.

In questo kit ***"REQUISITO PERFETTO"***, troverai la guida pratica che al suo interno contiene già una checklist adatta per chiunque sviluppi software ***BUSINESS-CRITICAL*****.** Per la checklist invece di tipo ***SAFETY-CRITICAL****,* a volte è compresa in questo kit se sei stato abbastanza veloce, bravo o fortunato... altrimenti puoi sempre provare a chiedercela a:

checklist@softwaresicuro.it

STABILE

Sembrerà scontato e banale, ma i requisiti a un certo punto del progetto, meglio prima che poi, devono diventare come i X Comandamenti della Bibbia: **scolpiti nella roccia.**

Non ha nessun senso lavorare su un vero e proprio caposaldo del ciclo di vita del software, come i requisiti, che sia instabile:

- *quale architetto, costruttore, progetterebbe una casa su delle fondamenta precarie, ballerine, mutevoli?*

- *che abbiano frane, smottamenti, aggiustamenti continui quasi giornalieri?*
- *quale inquilino abiterebbe in una casa fondata sulla sabbia, sull'argilla, su un terreno franoso?*
- ***voi utilizzereste un software safety- o business-critical che per tutto il suo ciclo di vita è stato continuamente rimaneggiato, modificato, smembrato, ricomposto, rivoltato?***

C'è poco da fare: bisogna stabilire delle milestone, dei cicli, delle iterazioni e regolarmente congelare i requisiti. Certo, richiede fatica: servono committenti che lo facciano, fornitori che si adeguino, team di sviluppatori che li sappiano gestire. Richiede tempo per fare la *review, la validazione*. Comporta un maggior effort quando arriva la fatidica modifica, la *Change Request*. Però ci salva da tanti quei casini, da tanti quei problemi... che la metà basta.

Se non vi basta la mia esperienza personale di oltre 20 anni nel software, è sufficiente leggere decine di statistiche fatte su migliaia di progetti: le modifiche dell'ultimo minuto, i cambi in zona Cesarini, le patch della notte prima hanno la responsabilità della gran parte dei bug software.

Come si risolve questa cosa? In teoria si risolve con un uso estensivo della *Change-Impact Analysis (Analisi d'impatto dei cambiamenti)* e con i *Test di Regressione*... ma come vedremo nei prossimi articoli, il **METODO PER UN EFFICIENTE SVILUPPO DEL SOFTWARE (M.E.D.S.)** prende il meglio di Agile e della certificazione DO-178 per suggerire la ***strategia "perfetta"***.

Nel frattempo... difendiamo a spada tratta la stabilità dei nostri requisiti! E qui ovviamente serve dotarsi di un *Configuration Management System (CMS)* che consenta di dare una versione, un'identificazione e una protezione da modifiche accidentali al mio set di requisiti.

Conclusioni

Bene, tornando al motivo principale per cui ho scritto questo lungo corso:

PERCHÈ DOVRESTI ADOTTARE QUANTO PRIMA IL METODO PRESENTATO IN QUESTO CASO DI STUDIO?

A questo punto ti domanderai: *"Ma Massimo, perché dovrei cambiare il mio approccio aziendale e cominciare a investire tempo, denaro e soldi nel cambiare radicalmente il modo di lavorare della mia azienda per quello che riguarda i requisiti e gli altri punti del M.E.D.S.?"*

Eh già... perché?

- *Perché dovresti far studiare questo corso a tutti i tuoi collaboratori e dipendenti?*
- *Perché dovresti iniziare a pensare a come adottare progressivamente queste 10 regole?*

- *Perché dovresti metterti a misurare le metriche a partire dall'introduzione di queste tecniche e a verificarne i miglioramenti (o peggioramenti) nel tuo progetto?*

La mia risposta magari ti stupirà:

PERCHÈ... NON LO FA (QUASI) NESSUNO!

Esatto! Pur essendo un metodo collaudato, che garantisce al software di tipo certificato come quello avionico una sicurezza quasi assoluta, pur migliorando la qualità, le prestazioni e i costi del ciclo di vita, pur mostrando in migliaia di progetti in tutto il mondo che l'investimento iniziale si ripaga con ritorni pari a 10 o 100 volte tanto in termini di risparmi successivi, quasi nessuno dei tuoi competitor non l'ha ancora adottato.

Fidati di me: aziende ne giro e pochi hanno un approccio rigoroso come quello descritto. A parole sì magari, ma nei fatti è un gran macello.

Perciò... adottare i ***DIECI COMANDAMENTI DEL REQUISITO (QUASI) PERFETTO*** insieme alle altre tecniche del **METODO PER UN EFFICIENTE SVILUPPO DEL SOFTWARE (M.E.D.S.):**

https://www.softwaresicuro.it/MEDS

semplicemente ti darà...

UN VANTAGGIO COMPETITIVO (QUASI) SLEALE NEI CONFRONTI DELLA CONCORRENZA!

È più chiaro adesso? Prima cominci ad abbracciare una filosofia derivata dagli ambienti veramente critici e collaudata in migliaia e migliaia di altri progetti in tutto il mondo, prima comincerai a crescere in qualità, prestazioni e ad abbattere i costi... e prima ti distanzierai in maniera netta dai tuoi competitor, che vedranno migliorare le tue performance, acquisire e tenere i clienti, ampliare il tuo giro d'affari... mentre loro continueranno a vendere software costoso, scadente e instabile.

Stai ancora aspettando? Le prime dieci regole d'oro sono qua sopra, già pronte... comincia ad applicarle, se poi hai dei dubbi contattami:

info@softwaresicuro.it