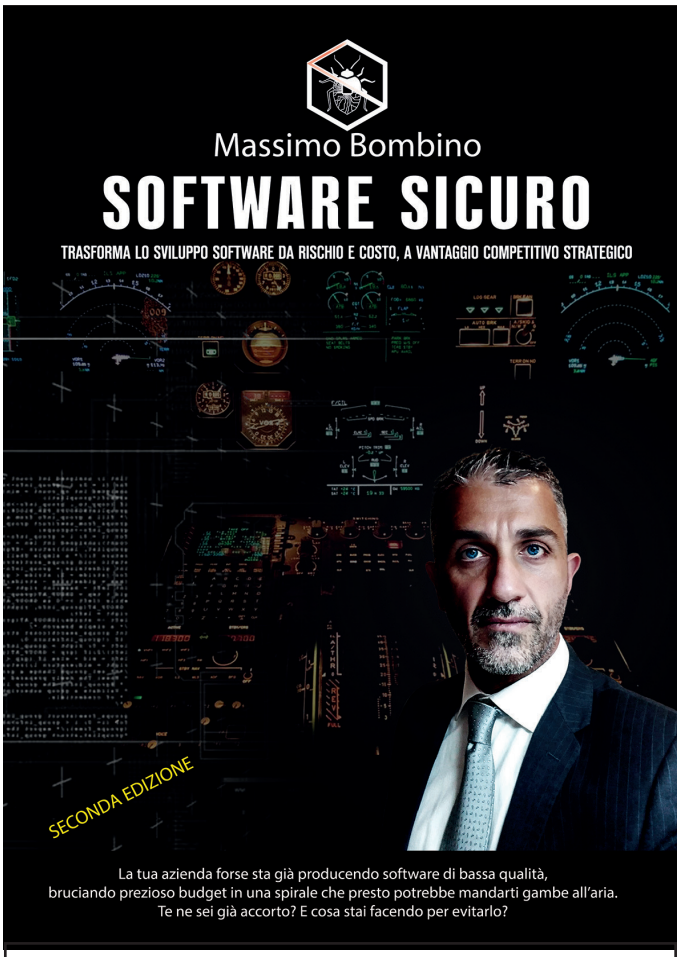
SOFTWARE SICURO

COME SVILUPPARE SOFTWARE SICURO E STABILE IN MODO EFFICIENTE, RIPETIBILE ED ECONOMICO

Newsletter N°2 - Febbraio 2019

E' USCITO IL MIO NUOVO LIBRO

SOFTWARE SICURO



L'unico libro che ti fa muovere i primi passi nel mondo del **software critico** ad altissima qualità, senza che tu sia già un esperto di certificazione!

Impara come le migliori tecniche derivate dal **software critico aerospaziale**, nonché dalla programmazione **AGILE**, in realtà combinate insieme possono essere la tua migliore arma nella battaglia del software.

CHIEDI LA TUA COPIA GRATUITA A:

LIBRO@SOFTWARESICURO.IT

CHRISTINE, LA MACCHINA (A GUIDA AUTONOMA) INFERNALE

Come forse avrai letto, a Marzo 2018 ***un'auto a guida autonoma di Uber ha investito una donna di 49 anni in Arizona, uccidendola.***

Esatto: come nei migliori (e peggiori) film di fantascienza se non horror, una macchina dotata di intelligenza artificiale apparentemente è impazzita e ha ucciso un essere umano, uno dei peggiori incubi dell'evoluzione tecnologica dell'uomo. La cosa come immagini ha creato grande scalpore e polemiche, provocando la sospensione di qualunque test su strada.

In questo articolo, cerco di spiegarti in maniera semplice cosa è successo, di chi è precisamente la colpa (e non è quello che raccontano i giornali, neanche la stampa specializzata!), qual è l'errore di fondo e cosa c'entra un vecchio col cappello.

Capirai perché tutto questo ha a che fare con i progetti software, con il gioco del poker e soprattutto comprenderai che, molto probabilmente, c'entri anche tu. Sì, proprio **TU**.

Ma andiamo con ordine.

La corsa alla guida autonoma

Da qualche anno, una lunga serie di startup tecnologiche, di grandi colossi dell'automobile, di aziende di servizi di trasporto alternativi come Uber hanno deciso che era ora di superare i limiti della guida ad opera degli uomini

(imprecisi, incapaci, stanchi se non addirittura spesso ubriachi) e che bisognava cominciare a sperimentare una guida autonoma operata da intelligenza artificiale: in poche parole dei computer molto potenti che prendono delle decisioni automatiche, analizzando la strada, gli ostacoli, gli altri veicoli e i pedoni e decidendo di accelerare, frenare e sterzare al posto del guidatore.

Ti ricorda qualcosa? Certo... macchine a guida autonoma, robot, sistemi informatici che prendono delle decisioni autonome e, come nei migliori film di fantascienza, diventano il peggior incubo dell›uomo. Da ***"Spazio 1999"*** alla saga di ***Terminator***, da ***"Christine, la macchina infernale"*** a ***"Il tagliaerbe"***, per passare attraverso una lunghissima serie di film, alcuni di serie B, l›uomo da sempre ha immaginato con orrore un futuro distopico in cui le macchine si ribellano e arrivano a uccidere se non addirittura a cercare di sterminare l›umanità.

Ebbene: sembra purtroppo che questo momento sia veramente arrivato.

Ma come funzionano questi sistemi di guida autonoma? Cosa ha provocato veramente l›incidente? Ma soprattutto, come mai tutti i giornali, anche specializzati, un sacco di esperti - se non addirittura le stesse aziende che sviluppano questi sistemi - stanno commettendo degli errori di fondo talmente grossolani per cui tutte le teorie e le applicazioni pratiche delle macchine a guida autonoma su strada in realtà sono un grosso inganno se non addirittura una truffa?

Come funzionano le auto che guidano da sole

L›idea di base di per sé è abbastanza semplice: si prende una macchina di quelle moderne, la si riempie di telecamere, sensori, radar e altri sistemi di sorveglianza e di analisi della strada, la si dota di attuatori molto rapidi che azionano lo sterzo, l›acceleratore e il freno e soprattutto si crea un software molto complesso che analizza tutti questi ingressi e decide come si deve comportare la macchina, cercando di essere uguale se non addirittura meglio di un umano alla guida. Il famoso robot che svolge un compito noioso, pericoloso o comunque obsoleto che l›uomo non vuole più fare.

Ma stiamo parlando di fantascienza... o di realtà?

Non sono poche le aziende in tutto il mondo che hanno investito anche grandissimi capitali umani e tecnologici per creare queste auto a guida autonoma… le maggiori case automobilistiche sono quasi tutte coinvolte con i loro esperimenti, ma ci sono anche grandi nomi della tecnologia come Google e Apple, oltre ad aziende come Uber che realizzano servizi trasporto tipo noleggio con conducente, tutti impegnati in questa gara contro il tempo per riuscire a realizzare per primi il sogno di ogni automobilista stanco, assonnato o che ha bevuto un po› troppo: quello di dire alla propria macchina "Kitt, riportami a casa" come faceva Michael Knight con la sua famosa auto super-intelligente (e pure un po' antipatica) nel telefilm degli anni 80 "Supercar".

Cosa è andato storto nell'auto di UBER (e NON è un guasto)

Allora, il primo problema (e ci tornerò sopra) è che il pedone, che conduceva a mano la bicicletta, stava attraversando in un punto vietato, senza strisce pedonali, senza luci dove le auto andavano a grande velocità. Per cui, già solo questo fatto rende le cause dell'incidente sbilanciate verso quello che ti voglio far capire, ma non è finita qui.

L'auto di Uber in realtà aveva a bordo un sistema di sicurezza, basato su radar, che aveva già identificato la povera vittima ed era già pronto ad entrare in funzione, ma che… era ***DISABILITATO.***

Esatto: il dispositivo che identifica potenziali ostacoli aveva già identificato l'ostacolo decine di metri e alcuni secondi prima dell'impatto ed era pronto a frenare in tempo per salvarla, ma non era abilitato! E come mai questa follia?

Semplice: per testare agevolmente i propri sistemi ed evitare falsi allarmi, l'azienda aveva deciso di disabilitare la frenata di emergenza e di affidare questo compito al guidatore a bordo, che con i tradizionali comandi sarebbe potuto (e dovuto!) intervenire in tempo per evitare la collisione. Anche se sembra assurdo, in realtà è perfettamente legittimo e normale… si lascia al sorvegliante umano la supervisione finale della sicurezza.

E allora perché la persona alla guida

non ha frenato in tempo?

Semplice:

perchè stava guardando… uno show al telefonino

Da quasi un'ora, la guidatrice, che ***era pagata 24$ all'ora per stare attenta e operativa al volante***, stava seguendo una trasmissione sul telefonino, guardando solo occasionalmente la strada. E s***i è accorta dell'imminente impatto meno di un secondo prima, senza avere il tempo di frenare.***

Quindi, tutta colpa sua? Della guidatrice di backup distratta? Andrà lei in galera? Non è la macchina posseduta, il robot impazzito, il software-killer?

Forse, ma in realtà non è neanche quello il punto. Ci sono altri fattori da considerare che adesso ti mostro, perché sembra un po› tutto basato su aziende molto potenti e tecnologicamente avanzatissime, che mettono a disposizione i migliori esperti al mondo e le tecnologie più incredibili per raggiungere questo scopo, non riuscendo però a gestire l'imprevedibile.

Ma in realtà c›è un grandissimo errore di fondo che quasi nessuno incredibilmente nota, se non gente come me che viene da settori ancora più delicati e critici come quello aeronautico.

Perché dei sistemi del genere non potranno mai funzionare, nelle condizioni al contorno attuali, a dispetto di tutti gli sforzi che stanno mettendo in piedi dei giganti dei trasporti e dell›informatica?

Ma soprattutto, esiste già qualcosa a livello di altri mezzi di trasporto autonomi nel mondo in altri settori?

Aerei che atterrano da soli

Certo, esistono eccome dei sistemi di trasporto gestiti completamente da intelligenze artificiali e computer a guida autonoma che funzionano bene, anzi molto bene, e sono in giro da vari anni. Li analizzo brevemente per farti capire che cosa hanno in comune ma soprattutto che cosa hanno di diverso rispetto alle automobili che dovrebbero guidare da sole sulle nostre strade di tutti i giorni e perché un incidente simile non potrà mai succedere.

Il motivo, come vedremo, è che questi altri sistemi funzionano con delle condizioni al contorno molto più restrittive di quello che giornali, esperti e grandissime aziende a livello mondiale e vorrebbero farci credere possibile per le auto.

Parliamo brevemente di uno dei primi sistemi autonomi di trasporto passeggeri che esiste da alcuni decenni e che funziona in maniera praticamente perfetta: ***atterraggio strumentale degli aerei o ILS (Instrument Landing System)***. Ti sarà capitato, soprattutto se vivi in una città del Nord Europa, che nel periodo autunnale o invernale, atterrando in una situazione di forte nebbia, il capitano abbia chiesto a tutti i passeggeri di spegnere qualunque dispositivo elettronico perché era in corso un atterraggio strumentale e non ci dovevano essere interferenze di nessun tipo con i sistemi di bordo.

Te la faccio semplice senza andare troppo in dettaglio di aspetti tecnologici che ti puoi andare a cercare tu anche solo su Wikipedia, ma questi sistemi si basano su un continuo scambio di informazioni tra tutta una serie di sensori e di sistemi informatici che sono a bordo dell›aereo, sui satelliti visibili in quel momento dall›aereo stesso nella torre di controllo e in tutti gli altri sistemi di terra che stanno intorno all›aeroporto e in varie stazioni su tutto il territorio. In poche parole, tutto l›atterraggio strumentale si basa non soltanto sulle apparecchiature per quanto sofisticate, sul software avanzato di bordo, sui sensori e radar che sono montati sull›aereo… ma si basano su una molteplicità di apparecchiature, di computer e di apparati di comunicazione che stanno sia sull'aereo che a terra e sui satelliti. E soprattutto una serie di sistemi di controllo tramite radar e altri strumenti per essere sicuri che tutto il percorso di atterraggio, i dintorni, la pista siano liberi da interferenze, altri aerei o oggetti volanti (vedi la questione dei droni), insomma che non ci siano elementi di disturbo esterni. Questo sarà un punto fondamentale su cui torneremo più avanti.

Questi sistemi esistono dagli anni '60 e hanno permesso di effettuare milioni e milioni di atterraggi strumentali senza nessun incidente grave. Infatti, considera che nel 2017 non c›è stato un singolo incidente mortale di aerei da trasporto passeggeri del mondo.

Ripetilo con me:

37 milioni di voli nel 2017, nessun incidente mortale

Con sistemi che si basano fortemente sul software e sulle comunicazioni radio per svolgere tutta una serie di compiti automatici. Evidentemente, se la probabilità di un incidente automobilistico in generale è 17.000 volte più alta rispetto a quella di un incidente aereo (esatto: l'automobile è diciassettemila volte più pericolosa dell'aereo!), qualcosa da imparare dal mondo avionico ce l›abbiamo, no? E alla fine della lettura di questo articolo ti sarà molto più chiaro perché questo è molto utile anche per te e per la tua azienda che fa software, anche se non avionico.

Treni che frenano in autonomia

Vediamo adesso un altro sistema di trasporto passeggeri altamente critico che si basa integralmente su sistemi di guida autonoma basati su computer: i treni ad Alta velocità, come l›italiano Frecciarossa che ha uno dei sistemi più avanzati al mondo per quanto riguarda la sicurezza, si basano pesantemente sulla guida strumentale, operata anche in questo caso da un mix dei sistemi di bordo e di quelli che stanno dislocati lungo la ferrovia e nelle sale di controllo delle stazioni intermedie e di quelle di partenza e arrivo.

Per avere un›idea, un treno come il ***Frecciarossa*** a pieno carico che viaggia a 300 km orari in una situazione di emergenza totale richiede almeno 4 ***km per fermarsi con un freno di emergenza***. Hai capito bene, ***QUATTRO CHILOMETRI!***

Se il macchinista aziona la frenata rapida, il treno continua ad andare avanti e a colpire qualunque tipo di ostacolo o problema davanti a lui per 4 km prima di fermarsi. Se invece si ferma con tranquillità senza emergenza, un treno di quel tipo ci mette dai 6 agli 8 km per fermarsi completamente.

Cosa significa questo?

Ovviamente, che ***il macchinista non ha nessun tipo di responsabilità e di azione per quanto riguarda le frenate di emergenza*** perché è talmente alta la velocità, e sono talmente lunghi gli spazi di arresto, che se anche lui vedesse ad occhio nudo un ostacolo qualche centinaio di metri più avanti, o anche un chilometro più avanti, non avrebbe nessun tipo di possibilità di fermarsi per tempo provocando quindi potenzialmente un grosso disastro ferroviario. Per questo motivo, anche qui ormai da tanti anni esistono dei sistemi di sicurezza molto avanzati tra cui il nostro ***ERTMS/ETCS (European Rail Traffic Management System/European Train Control System)*** che è un sistema di gestione, controllo e protezione del traffico ferroviario e relativo segnalamento che si basa ancora una volta su sistemi di bordo computerizzati, sensori, radar e altre cose montate sul treno, tutta una serie di transponder e altri sistemi che stando lungo la ferrovia, apparati di segnalazione e di comunicazione del treno con le stazioni più vicine nonché con quelle di partenza e arrivo.

In poche parole, per tutto il tratto in cui un Frecciarossa tra una stazione e l›altra viaggia ad alta velocità, l›azione dei macchinisti è totalmente inutile e il treno è completamente guidato dal computer, tant›è che loro stessi mi hanno detto che potrebbero viaggiare con il vetro completamente oscurato perché tanto non cambierebbe nulla per loro.

Ma come si fa a proteggere la ferrovia ad alta velocità da problemi di interferenze esterne?

È molto semplice: prima di tutto non esistono cose tipo passaggi a livello che rappresenterebbero un rischio troppo alto e non gestibile in nessun modo. Allo stesso modo, le rotaie dei treni ad Alta Velocità sono protette da centinaia di km di reti difficili da scavalcare, sistemi di videosorveglianza e allarme di prossimità nel caso scavalchi qualcuno o qualche grosso animale, e sistemi avanzati che controllano che le rotaie siano perfettamente integre senza nessun tipo di interruzione o problema. Inoltre, come dicevamo, il treno continua a comunicare la propria posizione e lo stato di sicurezza con dei transponder lungo le rotaie e le stazioni per assicurarsi che non ci sia nessun tipo di pericolo.

Il vero problema dell'incidente dell'auto di UBER?

Adesso, ti è forse più chiaro perché c›è un enorme errore di fondo, non tanto quando in cui si progettano questi sistemi a guida autonoma di automobili ma nel momento in cui si ha la pretesa utopistica di farli circolare su strade normale di tutti i giorni?

Strade con buche, senza segnaletica orizzontale o verticale, che si allargano o si stringono senza nessun criterio, con segnalazioni che non hanno sempre un senso o uno standard uniforme in tutto il paese, figuriamoci in paesi diversi.

Senza poi parlare ***degli altri guidatori****… persone di tutti i tipi, di tutte le età, con stili di guida completamente diversi che vanno dal giovane spericolato, al famoso vecchio col cappello di cui ho parlato all›inizio dell›articolo, alla signora che torna a casa con la spesa e via discorrendo.*

Ognuno con esperienza, abilità di guida, comportamento completamente diverso e del tutto imprevedibile.

Per finire poi coi pedoni, che attraversano (sempre?) sulle strisce ma anche improvvisamente, altri tipi di pericoli, carichi che vengono persi dai camion o da altre automobili e via così.

Un macello totale, completamente imprevedibile, che ***nessun computer al mondo potrà mai gestire in nessun modo in maniera sicura come quella di un ambiente iper-controllato, sicuro, standard come quello intorno a un aereo o un sistema ferroviario.***

Il vero problema dell'auto di Uber è stato quello… di permettere di farla circolare nelle strade di tutti i giorni.

La soluzione per le auto a guida autonoma?

L›unico modo per avere delle macchine a guida autonoma che possono portare passeggeri senza nessun rischio per la loro incolumità quella degli altri è quella di:

creare delle nuove strade e autostrade, con degli standard uguali in tutto il mondo o almeno in tutto il paese o insieme di Stati confinanti, perfettamente regolari e manutenute, con segnaletica orizzontale e verticale sempre perfetta, piene di sensori e transponder inseriti nell› asfalto, nei segnali stradali, nei guardrails che comunicano in continuazione con le auto, che individuano pericoli come perdita di carichi, incidenti, rallentamenti e via discorrendo.

Con auto che si parlano tra di loro e si trasmettono in continuazione le informazioni raccolte dai propri sensori e quelli della rete autostradale, che comunicano con la centrale di controllo del traffico più vicina. E soprattutto ***barriere lungo tutto il percorso e lungo gli accessi principali che blocchino completamente l›accesso a pedoni, estranei, animali e ovviamente macchine non a guida autonoma****, o che montano a bordo sistemi malfunzionanti o non aggiornati, macchine che imboccano l›autostrada nel senso sbagliato e via discorrendo. In poche parole, a chiunque non sia aggiornato e autorizzato.*

Adesso, immaginiamo che in tutto questo mondo così perfetto e simile all›ambiente sicuro delle ferrovie ad alta velocità e a quello comunque simile dell›atterraggio strumentale degli aerei, in questa autostrada sicura dove ci sono solo macchine a guida autonoma, improvvisamente irrompa quello che è l›incubo delle strade di tutti i giorni... ma che in un'autostrada del genere potrebbe diventare la causa di una vera e propria tragedia: ***IL VECCHIO CON IL CAPPELLO***.

Ossia quella persona, quell'auto, quel sistema, quel software che non è in regola con tutta l›architettura, lo standard, la conformità di tutte le altre macchine e dell'infrastruttura autostradale e in qualche modo violando gli accessi riesce ad accedere all›autostrada intelligente e si mette in mezzo alla strada a 40 all›ora con le altre macchine a guida autonoma che gli sfrecciano intorno impazzite fino a quando il comportamento è talmente imprevedibile da fare impazzire anche questi computer e provocare incidenti mortali.

Per cui, a maggior ragione, bisogna evitare in tutti i modi che il vecchio con il cappello possa in alcun modo individuare delle falle del sistema di accesso e irrompere violando tutti i requisiti e le caratteristiche di accesso.

Software e vecchi col cappello

E ora vediamo che cosa c›entrano i progetti software, il poker e soprattutto cosa c›entri **TU**.

I progetti software sono molto simili a quello di cui hai sentito parlare finora, in quanto sono sistemi molto complessi in cui interagiscono fra di loro tutta una serie di entità, di sottosistemi, di computer che comunicano fra di loro in maniera molto complessa. Esattamente come una rete autostradale intelligente, il sistema ferroviario ad alta velocità o un aereo che fa un atterraggio strumentale.

La stessa cura destinata alle regole, ai requisiti, all›infrastruttura, alla compatibilità, alla sicurezza degli accessi e via discorrendo deve essere messa anche in un progetto software di tutti i giorni, come quello in cui magari lavori o gestisci tu, altrimenti il rischio è che del codice non compatibile, una modifica non adeguatamente considerata nei suoi impatti, un aggiornamento di un sottosistema, il rilascio di un nuovo componente non previsto nell›infrastruttura iniziale potrebbe avere degli impatti devastanti e provocare conseguenze disastrose nel tuo software, nel tuo business e nei tuoi clienti.

In poche parole:

DEVI PROTEGGERE IL TUO PROGETTO

DAL VECCHIO COL CAPPELLO!

Cosa c'entra a questo punto... il POKER?

Nel famoso gioco di carte, c›è una regola famosa anche se non scritta... che dice che n*el momento in cui ti siedi, al tavolo c›è sempre una persona che non conosce bene le*

regole e che quindi può essere spennata. Ma se nel primo quarto d'ora di gioco non hai individuato chi è il pollo da spennare, c'è la grossa possibilità che sia TU.

Ok, e come mettiamo questo insieme con tutto il discorso precedente?

È molto semplice… *nell'ultimo quarto d'ora io ti ho parlato di come la progettazione di un sistema, di un'infrastruttura complessa sia piena di regole, di requisiti, di intelligenze autonome che si devono parlare tra di loro in un protocollo standard, di attori che interagiscono con il tuo sistema secondo protocolli ben precisi, di modifiche che vengono introdotte in maniera assolutamente accurata e di accessi che vengono protetti da tutte le persone, entità e processi che non sono autorizzati.*

In poche parole, nell'ultimo quarto d'ora ti ho parlato del ***potenziale vecchio con il cappello del tuo progetto software.***

Ecco, allora:

- *Se il mio discorso non ti ha fatto accendere i campanelli di allarme...*
- *Se non ti è venuto il sospetto di quale potrebbe essere il prossimo disastro nel tuo progetto...*
- *Se non immagini quale potrebbe essere la modifica non autorizzata che verrà introdotta a breve...*
- *Se non sai chi sta violando i requisiti per poter interagire con il tuo sistema...*
- *Se c'è una porta aperta dalla quale potrebbe entrare il vecchio col cappello e mandare all'aria il tuo sistema...*

Se queste domande non ti sono balzate in mente o non riesci a darti una risposta, bene, c'è il rischio fondato che il vecchio con il che sta mettendo in pericolo il tuo progetto software… possa essere proprio **TU**

Per cui forse è il caso che fai un bel respiro, rileggi con calma questo e gli altri articoli del blog:

https://softwaresicuro.it/

e cerchi di capire come evitare questi problemi nel TUO futuro, adottando un metodo di lavoro come il ***M.E.D.S (Metodo per lo Sviluppo Efficiente del Software)*** e far atterrare il tuo progetto in totale sicurezza… come un atterraggio strumentale di un aereo.

COME SI SCRIVE UN REQUISITO PERFETTO?

Eravamo rimasti in un altro articolo a 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***

Partire da requisiti chiari e ben definiti non solo è fondamentale per stabilire effettivamente quello che vuole il cliente ma, soprattutto se ci si deve adeguare ad una qualunque Certificazione Safety-Critical, diventa obbligatorio ed inderogabile.

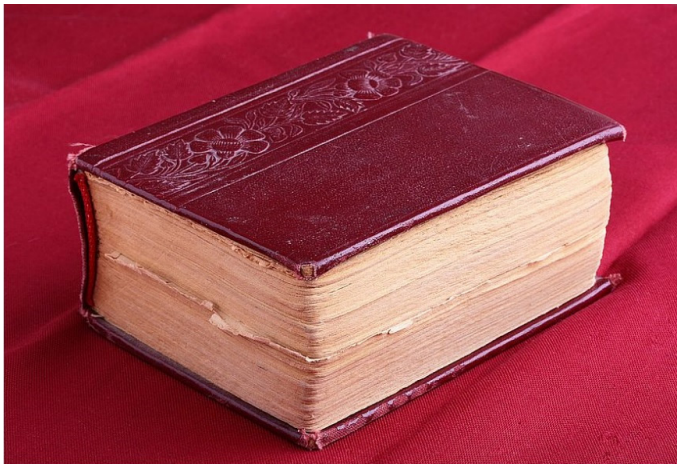
Avere requisiti approssimativi significherà ottenere un risultato non efficiente che avrà bisogno di continue modifiche per poter soddisfare le specifiche non chiarite all'inizio.

REQUISITI APPROSSIMATIVI =

risultato di pessima qualità +

spreco di tempo e risorse

La stesura dei requisiti NON è un'arte ma è un vero e proprio investimento, occorre quindi imparare al meglio a formulare requisiti funzionali e precisi, che non diano adito ad ambiguità o ad interpretazioni soggettive.



I "10 Comandamenti" del Requisito Perfetto

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 ricorso 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 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. Mi ringrazierai poi!

Infatti, ora ti spiegherò perché DEVI iniziare quanto prima ad adottare questo decalogo. E perché potrebbe fare la differenza per la sopravvivenza della tua azienda o team.

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 i***l 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 articolo:

PERCHE' DOVRESTI ADOTTARE QUANTO PRIMA QUESTO DECALOGO?

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 leggere questo articolo 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:

PERCHE'... 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.)*** semplicemente ti darà...

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

E' 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, gratis... ma se vuoi un vero e proprio corso, dopo aver organizzato di recente un webinar su questo argomento, ho organizzato qualcosa di innovativo, in Italia:

Un corso in DVD+Manuale

«COME SCRIVERE IL

REQUISITO SOFTWARE PEFETTO».

Questo corso, fatto appunto da un DVD/CD, un manuale e una guida pratica con un esempio svolto completamente, ti guiderà in dettaglio sui 10 punti precedenti, con esempi e linee guida per capire se e quali di questi punti sono adatti alla tua azienda e ai tuoi progetti, in base alla loro criticità.

Per informazioni:

https://www.softwaresicuro.it/requisitoperfetto

AGILE E DO-178: UN MATRIMONIO (IM) PERFETTO?

Quali sono gli aspetti delle migliori metodologie di sviluppo che si sono affermate negli ultimi anni sul mercato e che vogliamo selezionare accuratamente in modo da prenderne il meglio?

Quali sono i due mondi, migliori nel loro ambito ma imperfetti o troppo specializzati per essere applicati ovunque, da cui vogliamo cannibalizzare e parassitare le idee e le tecniche migliori, per piegarle a nostro piacimento?

Come possiamo sfrondare di orpelli, attività burocratiche e inutili, o inefficienti e costose, due processi altamente diffusi nel mondo del software in modo da ottenere una nuova sommatoria che è (quasi) perfetta?

Ecco, questo lungo articolo andrà a esaminare due metodologie molto conosciute da (quasi) tutti: il ***metodo AGILE*** e la ***Certificazione Avionica DO-178***.

Due tecnologie mature, complete, ricche di un sacco di aspetti interessanti ma a volte ***burocratiche, ridondanti, inefficienti***. O al ***contrario sbrigative, superficiali, incomplete.***

Per cui andiamo a fare un po' di cosiddetto *cherry-picking* e selezioneremo uno per uno gli aspetti, i principi, gli obiettivi principali dei due mondi e decideremo se portarli con noi e in che modo, se integralmente o con giudizio.

E ne tireremo fuori una metodologia nuova e a prova di bomba, da adottare in qualunque tipo di sviluppo, normato e certificato (Safety Critical), ma soprattutto in quello normale, quotidiano del mondo ***Business-Critical***.

Metodo AGILE

Lo conoscono tutti, dai. È diventato il metodo più celebre di produzione del software, almeno a livello di nome, superando di popolarità il vecchissimo Waterfall o il moderno Ciclo a V.

Ma... è veramente conosciuto nei dettagli o solo di nome o peggio ancora di fama?

E che fama ha... ne hai sentito parlare in termini polarizzati, quindi o troppo entusiastici o troppo drammatici?

La verità come sempre sta nel mezzo:

Il metodo AGILE si propone come approccio... appunto agile al mondo complesso e mutante dello sviluppo software, in contrapposizione a quello ingessato e pesante del Waterfall ma tende a essere poco adatto ad approcci rigorosi e critici.

Questo vuol dire che dobbiamo buttare via il bambino e l'acqua sporca? Non sia mai: cerchiamo di capire quali sono gli approcci interessanti di AGILE e quali invece dobbiamo buttare se vogliamo veramente essere ingegneristici e precisi, come richiesto dal software Safety- e Business-Critical pur rimanendo efficienti.

Partiamo dai 4 principi di base e cerchiamo di capire come nel gioco della torre quali buttare giù.

Questi principi sanciscono che bisogna dare "valore" (sigh, un giorno poi ti spiegherò il perché di questo lamento) a:

1) ***Individui e interazioni piuttosto che processi e tool:*** questa affermazione può anche sembrare apparentemente intelligente, pur pensando che ovviamente i PROCESSI sono fondamentali nel mondo della certificazione e in generale della Qualità, ma è facilmente smontabile da un'evidenza sotto gli occhi di tutti, gli individui si licenziano. Prima o poi se ne vanno, cambiano ruolo, mutano responsabilità per cui basarsi solo sulle persone e non sui processi è semplicemente folle. Teniamo buona però la parte sulle interazioni che svilupperemo dopo. I tool invece sono funzionali a quello che bisogna ottenere.

2) ***Software funzionante invece che documentazione completa***: anche qua, le due cose assolutamente devono coesistere, in un approccio che sia ingegneristico e sicuro. Il software funzionante è fondamentale: per questo motivo, come vedremo, l'importanza di AGILE al TEST deve essere assolutamente capitalizzata. Ma la documentazione... è talmente importante che questo punto suona quasi come eretico a chi deve certificare o produrre qualità e farebbe bene a ritenere l'affermazione molto contestabile.

3) ***Collaborazione col cliente invece che negoziazione di contratti***: questa è forse la cosa da smontare più radicalmente... o meglio, come sempre la parte iniziale di queste alternative è sempre golosa, peccato per quel "invece che". No: possiamo e dobbiamo coinvolgere il cliente, farlo partecipe del ciclo di vita, rendere più snelli i contratti, la stesura dei requisiti ma deve essere fatto mettendo anche dei paletti, delle iterazioni e prototipazioni concordate e congelate, dei report frequenti e interattivi.

4) ***Rispondere al Cambiamento invece che seguire un piano***: stesso difetto... prima parte sacrosanta. Il Cambiamento è una costante dei progetti software e ci deve essere in modo di gestire il cambiamento. Infatti, la tecnica della Continuous Integration e Continuous Testing le cannibalizzeremo a piene mani. Ma i Piani... dovranno assolutamente essere fatti bene, precisi, rigorosi. Certo, ci sarà lo spazio per la flessibilità... per prototipi, per iterazioni, per rilasci intermedi. Lasciando però la necessità anzi l'obbligo di rispettare certi standard, certe milestone, certe rigidità da introdurre solo dove necessario.

E delle varie tecniche e pratiche suggerite, cosa andremo a salvare e a far confluire nel nostro ?

a) ***Automazione***: questo è poco ma sicuro, automatizzare tutto, (quasi) ad ogni costo. Soprattutto la cosa più facilmente automatizzabile, da ripetere molto frequentemente se non ogni giorno, che rappresenta la migliore se non l'unica misura di quanto il software risponda ai requisiti e si comporti correttamente: il TEST. Automatizzare i test dovrebbe diventare una vera e propria religione.

b) ***Interazione***: questa non può che fare bene. Interazione con il cliente, nel gruppo di lavoro, tra team di sviluppo e di test, eccetera non possono che far crescere la consapevolezza, la cultura e quindi la qualità del lavoro fatto, così come aiutano a ridurre la catena di feedback e a capire subito se qualcosa non va, invece che all'ultimo.

c) ***Conoscenza e cultura***: come raccontavo del articolo sulla Cultura del software, un lavoro continuo e integrato del team e basato su leadership tecnica, workshop, Hackathon, task force, Gap Analysis ecc. è assolutamente salvifico e fondamentale per aumentare la qualità del lavoro fatto insieme.

d) ***Iterazioni***: qua bisogna fare attenzione... si parla in AGILE tanto di consegne MOLTO frequenti, forse fin troppo. Ok con le iterazioni... ma pianificate, con un piano preciso di rilascio, con la corretta priorità di funzionalità

e requisiti consegnati a ogni step.

e) ***Modellazione***: assolutamente un caposaldo del software moderno. Il Model-Driven Development basato su approcci standard come UML, SysML, MARTE oppure utilizzando tool proprietari consente risparmi di tempo pazzeschi rispetto a fare tutto da zero, pur conservando la qualità e maneggevolezza del software prodotto.

f) ***Programmazione in coppia***: altro principio sanissimo e da applicare a tutto il ciclo di vita. "Two is megl che uan", diceva una pubblicità. La review continua e integrata di ogni lavoro fatto da parte di un altro componente del team è molto positiva, a patto che sia fatta con un minimo di rigore e di pianificazione, magari tramite Checklist.

g) ***Test (Driven Development)***: test, test e ancora test. La chiave del successo di ogni iniziativa di sviluppo software. Riuscireste a pensare a un best-seller che non passi attraverso vari correttori di bozze e editor? O a una casa, un ponte, un palazzo che non sia testato tramite sollecitazioni statiche e dinamiche? Prendiamo il famoso TDD ossia Test-Driven Development: scrivere i test basandosi sui requisiti, prima che il codice sia pronto, è... semplicemente fantastico, perfetto. Così come il concetto di Continuous Testing and Integration... queste ce li prendiamo e ce li portiamo a casa tutti così come sono.

h) ***Versionamento***: questo è un altro principio che adottiamo senza remore. L'uso di un CMS (Configuration Management System) è tra i primissimi acquisti e priorità in assoluto di un processo di sviluppo software moderno, ma si deve estendere a TUTTI i manufatti di qualunque tipo: requisiti, documenti, codice, test, procedure ecc.

Quello che invece NON salviamo e che dobbiamo rapidamente dimenticare di AGILE, l'abbiamo in pratica menzionato...

- *NO a eccessiva prototipazione, iterazione, rilasci troppo brevi e frequenti*

- *NO all'eccessiva personalizzazione dei ruoli e delle attività*

- *NO a documentazione scarna e rilasciata all'ultimo*

- *NO al reverse-engineering o comunque grossa limitazione*

- *NO alle modifiche all'ultimo minuto che non passino attraverso tutte le fasi d'impatto e verifica necessarie*

Perciò dai... direi che non siamo messi così male, no?

Buona parte dei principi sono molto validi, di sicuro hanno un fondamento molto utile ma magari si perdono in contraddizioni che vogliamo evitare, oppure danno troppa poca importanza a certi aspetti fondamentali quando si vuole usare un approccio scientifico e ingegneristico al software, che sia nello stesso tempo agile come il metodo stesso vorrebbe. Perciò, con cautela e qualche eccezione... possiamo abbastanza applicarli.

Mentre delle tecniche... direi che quasi tutte sono MOLTO interessanti! Sono tecniche che permettono in effetti di rendere tutto il ciclo di vita del software molto più automatico, ripetibile, controllato, rapido e soprattutto sicuro il software che ne risulta.

E questo ci fa molta ma molta gola... perché ci consentirà di ridurre in maniera sensibile i costi e i tempi, ma nello stesso modo magicamente ci aiuterà a incrementare la qualità di quello che produciamo, che andrà a incidere non tanto sui costi immediati ma su quelli futuri, di manutenzione e di supporto sul campo del tuo software.

Vediamo ora invece la parte della certificazione avionica, che fa molta più paura (da fuori) per la sua rigidità e per

i suoi costi, ma che come vedremo ci darà in cambio una serie di enormi vantaggi anche per la tua azienda... anche se sviluppi software del tutto innocuo (per le persone) ma non per il business altrui (e quindi di conseguenza anche per il tuo!).

Certificazione Avionica DO-178

La Certificazione Avionica è in assoluto la più temuta: considerata alla stregua delle forche caudine, è considerata terribile e dispendiosa, non a torto direi. Il budget di un progetto "normale" certificato con standard diversi o addirittura non certificato, può lievitare anche del 200 o del 300% se l'azienda volesse procedere lungo il percorso della DO-178, senza tra l'altro nessuna garanzia di riuscita e soprattutto con una difficile gestione e previsione dei tempi.

Qual è la contro-partita?

Bene: nel 2017, su 37 MILIONI DI VOLI non c'è stato ***NESSUN INCIDENTE MORTALE*** *con aerei di linea.* ***NESSUNO****.*

È sufficientemente chiaro?

Pensate a queste cose:

- *ad un telefonino, che per un intero anno non ha app che falliscono rovinosamente*

- *ad un computer, che per 12 mesi non ha nessun crash e perdita di dati*

- *ad un sito Internet, che trovate sempre attivo 24 ore su 24 senza interruzioni, ritardi, pagine perse*

Sarebbe bello, eh?

Peccato che è stato calcolato che un ipotetico telefonino DO-178 compliant potrebbe costare fino a 20-30.000€ se non di più (sì, TRENTAMILA EURO). E ogni singola applicazione migliaia di euro.

Ok, va bene, ti faccio una domanda: *Ti accontenteresti di produrre un telefonino, un'app, un programma per computer, un sistema embedded, molto più affidabile, performante, sicuro al prezzo di un contenuto e ragionevole incremento dei costi? Ma risparmiando poi molto di più in termini di manutenzione, supporto, costi di assistenza?"*

Se la risposta è NO: hai sbagliato articolo, continua pure a spendere poco, pochissimo nello sviluppo e a bruciare capitali ingenti, non prevedibili e fuori controllo nell'assistenza e supporto post-vendita e a gestire le cattive referenze di clienti incavolati neri. AUGURI.

Se invece vuoi scoprire come fare a:

- *sfruttare a tuo personale vantaggio i punti salienti della certificazione avionica DO-178*

- *applicare i principi fondamentali in un ambiente safety-critical non avionico o anche solo business-critical*

- *lasciar perdere tutte la burocrazia, attività ridondanti e inutili per la tua azienda*

- *ottenere un incremento incredibile della qualità del tuo software a fronte di un piccolo investimento nel processo di sviluppo*

- *risparmiare alla fine un sacco di soldi considerando tutto il ciclo di vita compreso supporto e assistenza sul campo*

allora è il caso che continui a leggere, perché i principi più sani ed efficienti della certificazione DO-178 che hanno ispirato il ***METODO PER UN EFFICIENTE SVILUPPO DEL SOFTWARE (M.E.D.S.)*** molto probabilmente fanno al caso tuo.

Ok, ora rispondo alla domanda fondamentale che ti frulla in testa:

"Quali sono i principi fondamentali della certificazione avionica secondo lo standard DO-178 che eventualmente vale la pena di tenere e adottare, e quali invece butto dalla torre?"

Vediamoli uno per uno:

1) ***LIVELLI DI SICUREZZA:*** uno degli aspetti fondamentali della Certificazione Avionica DO-178 è quello del livello di sicurezza, chiamato ***Design Assurance Level (DAL).*** Per farla breve, si divide in 5 livelli dove la percentuale di errore del software, calcolata in maniera del tutto teorica e non misurata realmente, va da 10E-09 ore di voli per il Level A, fino a 10E-03 del Level E. Esatto, un errore ogni miliardo di ore di volo. Lasciando a un altro articolo una spiegazione delle motivazioni e conseguenze reali di questo aspetto, facciamo un discorso più pragmatico. Capiamo da soli che un software di livello A è molto più sicuro ma anche più caro di un software di livello E, che potremmo considerare simile ad un normale software applicativo che gira sul nostro computer o telefonino. Per cui, cosa succederebbe se lasciassimo libera iniziativa alle aziende aeronautiche senza imporre nessun vincolo? Ci sarebbero aerei con solo software di livello E che si schianterebbero un giorno

sì e uno no. E se invece fossero le autorità certificative statali o comunitarie a fissare da sole i paletti? Avremmo tutti aerei di livello A, super-costosi e super-sicuri ma dove il biglietto costerebbe decine o centinaia di migliaia di euro. È quello che vogliamo? Assolutamente no: per quello che si sono fissati i livelli! Perciò, un'attività ben precisa di Safety Assessment, effettuata in più riprese e modalità, assegnerà ad ogni sistema, ad ogni componente, ad ogni software un livello di criticità apposito. Per cui, questa classificazione dei miei sistemi di bordo a seconda dei livelli di criticità rappresenta il miglior compromesso in assoluto tra costi e sicurezza. Ha senso tutto questo al di fuori dell'avionica? Ha senso sì nella certificazione ***Safety-Critical*** in generale: anche nel mondo ferroviario, automobilistico, medicale ecc. esiste una classificazione in livelli assolutamente analoga. Al di fuori di questi mondi certificati e normati, ha senso? No, non molto... sicuramente però posso avere delle linee guida simili, più morbide, per le quali decidere le attività che devono essere svolte per un certo tipo di software, e quelle che invece non debbo necessariamente fare.

2) ***DETERMINISMO***: questo di fatto è uno dei principi più importanti e più ingombranti della certificazione avionica. Per darvi un'idea immediata e comprensibile: nel mondo della certificazione, da pochissimi anni (3-4 al massimo) si sono affacciate tecnologie considerate... distruttive, come l'Object Oriented, il Model-Based, i Metodi Formali e linguaggi "avveniristici" come il... C++. Ti rendi conto di quanto siano conservatori in questo mondo? Ok, va bene essere attenti, prudenti, attenti alle conseguenze di qualunque scelta ma come sappiamo benissimo il mondo non avionico è andato avanti anni-luce e ha prodotto una quantità notevole di innovazioni tecnologiche, di tool, di linguaggi. Cosa vuol dire, che va bene tutto? No: per niente. Non è il caso di fare da sperimentatori per gli ultimi capricci delle case produttrici di tool, non è bello da fare da beta-tester di tecnologie immature e via discorrendo. Forse è il caso di avere un atteggiamento di giusta prudenza e adottare tecnologie che siano abbastanza mature, pur fornendo dei vantaggi misurabili.

3) ***PIANIFICAZIONE***: ok, la parte dei "famigerati" piani della DO-178 è una delle più temute in assoluto, perché come chi è passato dalla tortura... ehm dalla certificazione, sa che l'approvazione dei piani tipo il famoso PSAC (Plan of Software Aspects of Certification) è in assoluto il primo e il più difficile scoglio. Bene, sgombriamo il campo da equivoci: la pianificazione va bene, ma i famosi 5+5 piani della certificazione (5 software, 5 hardware) sono veramente fin TROPPO per noi. Sì, adotteremo alcuni dei principi sani di pianificazione, ma qua c'è veramente tanto da sfrondare

4) ***QUALITA***': nel mondo dell'avionica, esiste una figura tanto delicata e importante, quanto temuta. Quella del ***Quality Assurance Manager (QAM).*** Un vero e proprio direttore d'orchestra: non deve saper necessariamente programmare, non deve sapere effettuare dei test ma è quello che dirige il lavoro di tutti, verificando modi, tempi, conformità e garantendo all'autorità il rispetto dei piani. Sovrabbondante: un Responsabile della Qualità è importante in tutti i settori, anche non avionici, ma non necessariamente al livello di parossismo richiesto dalla DO-178, a meno che tu sia obbligato dalla Certificazione, si intende. Per cui Qualità sì, ma con Equilibrio.

5) ***PROCESSO***: beh certo, il processo è talmente importante nel mondo avionico che è l'essenza stessa della certificazione. L'adesione incondizionata, totale e senza eccezioni ai 71 obiettivi della certificazione DO-178 è semplicemente obbligatoria. Senza eccezioni. Noi cercheremo di sviluppare in questo aricolo un nuovo metodo e approccio, più volte citato e chiamato ***METODO PER UN EFFICIENTE SVILUPPO DEL SOFTWARE (M.E.D.S.)***, ma lasceremo un certo grado di flessibilità e di progressione nell'applicarlo. Ma attenzione: non perché non serva, o sia ridondante, applicarlo tutto. Anzi. Semplicemente per dare a te, lettore, la possibilità di applicarlo secondo i tuoi tempi, perché a ogni passo saranno talmente tanti i vantaggi che troverai nel breve e lungo periodo, che troverai un tale ritorno dell'investimento che vorrai applicarlo integralmente il prima possibile, ma per TUA personale scelta di risparmio ed efficienza.

6) ***REQUISITI***: ecco, i requisiti sono fondamentali. ***FONDAMENTALI***. È sufficientemente chiaro? Sono ***FONDAMENTALI!*** Rinunciare a dei requisiti fatti bene è una tale zappa sui piedi, una tale disgrazia, un tale atto di auto-lesionismo che se non hai intenzione di cambiare radicalmente il modo di gestire i requisiti del tuo software quanto prima come cosa prioritaria, ti consiglio vivamente di cambiare metodo, approccio. Di interrompere qui la lettura. Oppure continuare e capire perché questo è un tassello fondamentale della tua strategia alla fine per spendere MENO per il tuo software. Poi la certificazione avionica ti suggerisce di avere una differenziazione tra requisiti di alto e di basso livello, ma noi non adotteremo questo duplice livello se non ne saremo costretti. Ma tutto il resto... è oggetto di un altro articolo dedicato che ti invito ad andare a leggere con cura. Ti aspetto qui.

7) ***CRITERI DI TRANSIZIONE***: quando ho completato un'attività di scrittura dei requisiti? Quando posso iniziare a testare? Che documenti mi servono per partire con lo sviluppo? Quali sono i risultati di un'attività di pianificazione dei test? Queste sono alcune delle tante domande a cui corrispondono i famosi Transition Criteria della certificazione avionica. In maniera quasi parossistica, lo standard DO-178 definisce decine e decine di criteri di transizione: praticamente per ognuna della attività, vengono definiti tutti i documenti e i dati in ingresso, il momento esatto in cui deve partire il lavoro, quali sono i risultati attesi e qual è il momento

in cui l'attività deve ritenersi conclusa. Cosa salviamo di tutto questo lavoro? Di certo, possiamo semplificare il numero totale di attività che sono fin troppo segmentate nell'ambito avionico e possiamo essere meno rigorosi nel definire tutti gli ingressi ed uscite, ma rendere più rigoroso il metodo tramite una chiara idea delle attività da svolgere, quando vanno iniziate e quando sono da considerarsi terminate e cosa producono come risultato, è in ogni caso un valore aggiunto che responsabilizza i vari attori coinvolti. Da usare con moderazione, ma salviamo anche questo.

8) ***CHECKLIST***: un aspetto meno noto e fondamentale della certificazione avionica, anche se profondamente legato ai criteri di transizione, è sicuramente quello delle checklist. Cosa vuol dire? Normalmente, qualunque attività nel mondo del software (e in mille altri ambiti) si ispira a dei principi fondamentali che vengono seguiti in maniera più o meno fedele, ma di fatto molto difficile da controllare. L'iniziativa personale, il contributo dell'esperienza del singolo, l'aggiunta di ispirazione e creatività dello sviluppatore e del professionista rendono la maggior parte delle attività, della loro pianificazione e della loro verifica molto differenziate e poco ripetibili. Quindi poco ingegnerizzate, cosa che il metodo M.E.D.S. aborre... Per cui, cosa servono le Checklist? Servono a fare in modo che qualunque attività e sua transizione, tipo Pianificazione, Sviluppo, Test, Review ecc. segua dei principi ispiratori ben precisi e codificati in una serie di domande e risposte che bisogna OBBLIGATORIAMENTE seguire, almeno per quanto riguarda la certificazione avionica classica. Perfetto. Cosa facciamo, adottiamo queste checklist anche noi? Certo, perché no? Magari meno rigorose, meno dettagliate, meno invadenti... ma come spiegheremo in dettaglio, delle checklist aiuteranno a ridurre il contributo personale, creativo per cui non ripetibile, ingegnerizzabile dei tuoi collaboratori. E questo è SOLO UN ASPETTO POSITIVO. Perciò... parliamone: delle checklist fatte bene, snelle ed efficienti, aiuteranno soprattutto a raggiungere un livello di review sistematico e ripetibile, e le troveremo indispensabili in caso di avvicendamento e rotazione del personale.

9) ***VERIFICA (REVISIONE, TEST,ANALISI)***: vabbè, gli avionici in quella che chiamano Verification, comprendono anche le attività di Review che pesano molto e quelle di Analisys che coprono i casi limite in cui il Test non arriva, ma alcuni dati parlano molto chiaro: di 71 obiettivi della DO-178, la bellezza di 41 obiettivi sono relativi alle attività di Verifica. Esatto: il 60% degli obiettivi e indicativamente anche dei costi e del tempo sono dedicate ad attività che comprendono appunto la magica triade. Andiamo ad esplodere i tre contributi nella loro importanza al di fuori del mondo avionico, non senza qualche sorpresa.

10) ***REVISIONE***: nel mondo avionico, qualunque tipo di documento, requisito, codice, dato ecc. è sempre e comunque soggetto a un'attività di Review. Non si scappa: anzi la procedura di rilettura, revisione di qualunque tipo di documento o dato è rinforzata dall'imposizione del criterio di Indipendenza, per cui colui che fa la Review di qualunque attività non è la persona che l'ha effettuata. Molto diverso da come si intende normalmente l'indipendenza nel mondo del software: di solito si intende ad esempio che chi fa i requisiti deve essere diverso da chi fa il codice, la stessa cosa chi fa i test rispetto a chi fa sviluppo Bene: la DO-178 invece impone la sola indipendenza del revisore, la cosiddetta Peer Review. Concetto che ci piace, non ci costa niente tanto comunque la revisione dobbiamo farla lo stesso, tanto vale farla fare da qualcun'altro che troverà sicuramente più errori di noi stessi. Se poi vogliamo aggiungere l'indipendenza anche tra le attività, perché no?

11) ***TEST***: c'è bisogno di aggiungere altro? Il Test è la vera e unica attività insostituibile per verificare, misurare e per migliorare la qualità di un software. È un'attività che ti porterà come minimo a raddoppiare inizialmente il tuo budget di sviluppo. Esatto, hai capito bene: a RADDOPPIARE IL TUO BUDGET. Chi al momento ad esempio di acquisire un tool di test si lamenta del costo di poche migliaia di euro, non ha idea che il budget di test sarà enormemente maggiore ed il costo è prevalentemente quello umano di sviluppo dei test. Del resto, è l'unico modo che abbiamo per sapere se abbiamo costruito esattamente quello che ci ha chiesto il cliente. Quello che dovevamo fare e non quello che ci siamo messi in testa di fare. Qua ci sarebbe da scrivere per mesi, per anni anzi e infatti dedicheremo più di un articolo a questo argomento, se non addirittura un intero libro! Qua ricapitoliamo velocemente che non basta solo parlare di test: si parlerà di Test Plan e strategia, di Procedure di Test, di Test Normali e di Robustezza quindi di test nel normale range operativo e in quello eccezionale; si parlerà di vari livelli di test, dai Test di Basso Livello, passando dall'Integrazione Software-Software fino all'Integrazione Hardware-Software. Tutto soggetto a revisione indipendente. Perciò non stiamo ora a scendere in dettaglio di cosa salviamo per i nostri scopi e di cosa no, ma sicuramente gran parte della strategia avionica di test è nostra assoluta ed indissolubile amica.

12) ***ANALISI***: questo è un aspetto molto particolare della DO-178 e non è certo di nostro particolare interesse, se non in casi molto particolari. Si sta dicendo questo: quando non sei in grado di validare qualcosa, non riesci a testarlo, insomma hai difficoltà a verificarlo, allora puoi dimostrare la tua affermazione, requisito, test con un'attività di analisi ingegneristica, quindi umana. Esempio: non riesci a misurare sul campo reale che un certo algoritmo ci metta al massimo 10 millisecondi in tutte le condizioni? E allora fai un'analisi manuale, verificando a mano dall'assembler e contando il numero dei cicli macchina. E via discorrendo. Quindi

è un concetto molto generico e da utilizzare solo in casi veramente specifici, quando è richiesta una prova per validare una certa teoria. Molto raramente si usa altrove, ma teniamone almeno conto.

13) ***COPERTURA***: questo è un altro concetto molto importante, che ha una priorità abbastanza alta nella scaletta di adozione del METODO PER UN EFFICIENTE SVILUPPO DEL SOFTWARE (M.E.D.S.), in modo da incrementare molto velocemente la qualità dei prodotti che vendi e che contengono software. Di cosa si tratta? Si tratta di misurare in tempo reale, durante i tuoi test soprattutto funzionali e di sistema, quindi in ambienti reali o realistici, la copertura del tuo codice in termini di linee eseguite (Statement), di scelte tipo IF-THEN-ELSE eseguiti sia nel ramo True che False (Branch) e di coppie di variabili indipendenti (MC/DC). Qual è l'obiettivo da raggiungere per la DO-178? Semplice: a seconda del livello di criticità, viene richiesto di aderire alla copertura di Statement, Branch o MC/DC a livello del 100%. Esatto: CENTO PER CENTO. Per cui tutto il software che vola deve essere completamente coperto durante i test. Ma qual è il motivo, hai mai riflettuto? Ok, sicuramente è anche una questione di esecuzione accidentale di software non previsto. Ma il motivo principale per cui anche tu non dovresti vendere software che non ha raggiunto percentuali di coverage molto alte è ancora più semplice: il software non coperto, è quello non testato. E il software non testato ha un sacco di bug. E indovina chi andrà a testare e scoprire tutti questi errori? Esatto, il tuo tester più bravo, efficiente ma costoso: il tuo cliente finale. E lo sai benissimo come funziona: i tuoi clienti sono fantastici a creare situazioni anomale, non previste, a schiacciare il bottone sbagliato al momento sbagliato, a fare tutto quello che OVVIAMENTE tu che hai creato il prodotto non faresti mai, ma lui sì. E il cliente se ha un problema, un crash, un danno o ancora peggio se si ferisce e addirittura muore per colpa dei tuoi bug, farà una cosa molto semplice (o gli eredi per lui): ti chiederà un sacco di soldi per ogni errore che trova o smetterà di pagarti e ti manderà in rovina. Ti serve altro per accogliere il principio del Coverage nella tua strategia di sviluppo?

Conclusioni

Ecco, abbiamo quindi visto come da questi due metodi, normalmente applicati in uno specifico contesto non sempre applicabile, possiamo tranquillamente estrarre alcuni principi ispiratori, alcune tecniche e metodologie che sono applicabili in qualunque ambito, anche non certificativo

conservando la flessibilità e scalabilità del metodo ma nello stesso tempo un necessario rigore

A breve organizzerò un webinar su questo argomento, da cui ne ricaverò molto probabilmente un INFOPRODOTTO.

Se vuoi rimanere aggiornato, guarda al link:

https://www.softwaresicuro.it/AgileVsDO178

LA CULTURA, QUESTA SCONOSCIUTA (SECONDA PARTE)

Riprendiamo la seconda parte dell'articolo comparso sul numero 1 di questa Newsletter, dedicato al delicato sforzo di stabilire una ***Cultura Aziendale del Software*** condivisa a tutti i livelli.

Bug Review Meeting

Il ***Bug Review Meeting*** o sua denominazione simile è un meeting periodico (settimanale o più frequente) in cui si analizzano tutti i bug aperti di un software in ordine di priorità, per essere certi che vengano gestiti con tempestività e progrediscano correttamente. Vengono invitati a partecipare tutti i responsabili di un team di sviluppo, direttamente o in maniera remota tramite webinar ed altri metodi di accesso remoto.

A partire dai problemi più urgenti e bloccanti, si analizzano uno per uno lo stato del precedente incontro, i passi avanti effettuati e quanto manca per raggiungere una soluzione o un workaround. Ad ogni bug viene associato un responsabile, che si occuperà direttamente della sua risoluzione o di coordinare altre persone e gruppi, assegnando eventualmente altre azioni (Action Item).

La cosa importante è che nessun bug possa sopravvivere in maniera indisturbata e non controllata, ma che ci sia

un continuo monitoraggio e aggiornamento delle statistiche e delle metriche.

L'utilizzo di tool di Bug Tracking Management e di Dashboard grafiche aiuta fortemente a tenere sotto controllo la situazione.

Il ***METODO DELLO SVILUPPO EFFICIENTE DEL CODICE*** raccomanda fortemente l'implementazione di questi Bug Review Meeting periodici in modo da tenere sempre sotto controllo un progetto, suggerendo tra le altre cose periodicità e ruoli in maniera differenziata a seconda del progetto e della sua dimensione, criticità e vicinanza ad una Release principale.

Le caratteristiche quindi sono:

- *gruppo di 5-15 persone (a seconda di quanti*

gruppi software ci sono)

- *guida: da parte del Manager dei Test e Qualità*
- *stessa stanza se possibile, altrimenti online*
- *periodico (settimanale o più/meno frequente a seconda delle esigenze)*
- *ordine di azione in base alle priorità*
- *risultato: identificazione di un responsabile e delle strategie per ogni bug aperto*

Code/Document Review Meeting

Le Revisioni o Review sono una componente fondamentale di qualunque strategia improntata sia alla Qualità che alla Sicurezza, intesa sia come Sicurezza delle Persone (Safety) che degli Accessi (Security).

Considerate che le Review, quando si ha a che fare con il mondo Safety Critical, devono avere le seguenti caratteristiche:

- *devono essere FORMALI quindi avere un Review Protocol con tutta la storia dei commenti, delle discussioni e delle modifiche*
- *devono essere GUIDATE attraverso delle Checklist che definiscano esattamente quali domande, quali argomenti, quali particolari devono essere oggetto di revisione*
- *devono essere INDIPENDENTI, ossia realizzate da una persona diversa rispetto a quella che ha scritto il documento, il codice, il test*
- *devono essere SUPERVISIONATE da un Quality Manager tramite degli Audit di Qualità*

Ovviamente non tutti dobbiamo scrivere software altamente critico per Safety/Security, ma siccome la Qualità come non mi stancherò mai di dire vuol dire fondamentalmente Risparmio di Costi e Tempi, adottare una strategia di review consistente e formale è importante per chiunque faccia software.

All'interno del ***METODO DELLO SVILUPPO EFFICIENTE DEL CODICE***, fornisco delle linee guida per decidere periodicità, formalità e altri aspetti delle Review secondo le caratteristiche del progetto, in modo da non appesantirlo con inutile burocrazia ma nello stesso tempo da consentire di curare tutto quello che è generato da un progetto.

Le Review possono essere individuali, fatte anche in remoto tramite semplice invio dei documenti via mail o altro metodo di condivisione, oppure di gruppo. Le caratteristiche quindi sono:

- *individuali: o di gruppo tramite meeting*
- *formalizzate: tramite Checklist, indipendenza e Audit*
- *periodicità: ad ogni versione/revisione importante del documento*
- *risultato: Review Protocol che indichi la storia delle modifiche e la versione nuova*