PROBLEMATICHE DI CERTIFICAZIONE DEL SOFTWARE SAFETY-CRITICAL:*
I - Generalità, elicitazione e specifica dei requisiti

Fabio A. Schreiber e Angelo Morzenti
Dipartimento di Elettronica e Informazione
Politecnico di Milano
{schreibe, morzenti}@elet.polimi.it

Pubblicato su ***Ingegneria Ferroviaria,*** Anno LIV, n°. 10, ottobre 1999
pp. 709-746

---

* Ricerca parzialmente finanziata dal MURST, progetto "Scienza e tecnica per la qualità della mobilità locale"

SOMMARIO
La problematica affrontata in questo lavoro concerne la certificazione dei sistemi safety-critical che, come loro parte integrante, contengono componenti informatici hardware e software, con particolare riguardo alle applicazioni per i sistemi di controllo e comando in ambito ferroviario.

Sono state analizzate 14 metodologie utilizzate nell'analisi dei rischi (hazard), confrontandole sotto diversi profili:

- pregi e difetti;
- skill richiesti;
- applicabilità;
- collocazione ambientale.

Sono stati quindi analizzati e confrontati 12 metodi formali per la specifica dei requisiti del software, rilevando i loro punti di forza e di debolezza ai fini della certificazione di "nuclei safety-critical" di software nell'ambito di sistemi ferroviari. Parametri utilizzati per il confronto sono stati, tra gli altri:

- stato della standardizzazione;
- livello di rigore;
- tipo di formalismo e tecnica di ragionamento;
- strumenti disponibili;
- scalabilità e modularità;
- supporto di tempo reale;
- capacità di astrazione.

Si è poi analizzato criticamente quanto le norme prescrivono in fatto di architettura del software e di tecniche per garantire la software fault-tolerance, confrontando tali prescrizioni con la prassi industriale e gli orientamenti accademici in merito a:

- progetto e sviluppo di software;
- verifica del software;
- integrazione software/hardware;
- convalida del software;
- valutazione del software.

Sono state infine formulate alcune raccomandazioni rispetto al "modus operandi" di un organismo di certificazione nell'ambito dei sistemi di controllo e comando per sistemi safety-critical.

Una versione più estesa di questo lavoro si trova nel rapporto [113] e può essere reperita on line all'indirizzo:

file:////Www_dei/usr/Users/DEI/Sections/CompEng/Angelo.Morzenti/listpub.html

Ringraziamenti:gli autori desiderano ringraziare vivamente l'Ing.Antonio Puggioni per il contributo da lui dato a questo lavoro.

1. I SISTEMI SAFETY-CRITICAL

Vi sono varie definizioni di Safety; il progetto di normativa europea EN 50126 definisce la Safety "*Assenza di livelli inaccettabili di rischio*"; una definizione, meno tecnica, di più immediata comprensione (e al tempo stesso accettata dalla maggior parte dei mondo accademico) può essere: la Safety *è la proprietà di un sistema di non arrecare danno alla vita umana o all'ambiente.*

Nelle definizioni meno recenti non si tiene conto del danno all'ambiente, ma tale ampliamento (o completamento) è caldeggiato da molti studiosi come Taylor, Storey e Dahl, anche se, a onor del vero, non è stato ancora recepito dagli Standard (p.e IEC 1508, EN 50126) che, nel classificare la severità degli azzardi, si limitano a considerare le conseguenze per le persone e per il servizio.

Alcuni autori Italiani traducono Safety semplicemente con la parola "Sicurezza", ma ciò ci sembra riduttivo dato il significato consolidato di “protezione da violazioni all’integrità del sistema” che il termine sicurezza ha assunto da tempo; perciò, in questo lavoro, continueremo ad usare il termine inglese Safety con l'accezione sopra riportata.

Nessun sistema può essere assolutamente sicuro (Safe), ma può essere sicuro ad un livello adeguato per il suo ruolo. Da questa osservazione nasce il concetto di livelli d'integrità Safety che servono a indicare i metodi di progetto e d'implementazione secondo una vera e propria scala di criticità del sistema.

Sarebbe corretto iniziare tutti i progetti di sistema basati su computer cercando di capire se essi, come molto spesso accade, hanno implicazioni con la Safety. Inoltre sarebbe auspicabile che questa strada, seguita ad esempio dal ministero della difesa inglese per il software militare, fosse seguita in tutti gli standard dei software di pubblico impiego.

Come rimarcato più volte da molti studiosi, si può assicurare la Safety di un sistema soltanto considerando tutti i suoi aspetti sia Hardware sia Software in quanto, come ad esempio ricordato da Leveson [88], il tasso di fallimento del software è tanto più elevato quanto più frequentemente si presentano guasti hardware .

Si è cominciato a comprendere realmente (e a tentare di rimediare) i problemi potenziali dei sistemi Safety-Critical nei primi anni '70 e da allora si sono sviluppati molti standard per regolare l'applicazione di computer che debbano svolgere funzioni di Safety (EWICS-TC7, IEC, ISA, EN50126, EN50128, EN50129, ecc.) [10].

Tale attività di standardizzazione è tanto più necessaria se si pensa che la continua caduta del costo dei sistemi computerizzati, iniziata agli inizi degli anni '70 ed ancora ben lungi dall'esaurirsi, ha portato e porterà ad un incremento oltre ogni previsione dell'uso dei computer.

I risultati principali di oltre vent'anni di ricerca possono essere così sintetizzati:

- la Safety è un risultato culturale;
- la Safety è un risultato di gestione;
- la Safety è un risultato ingegneristico.

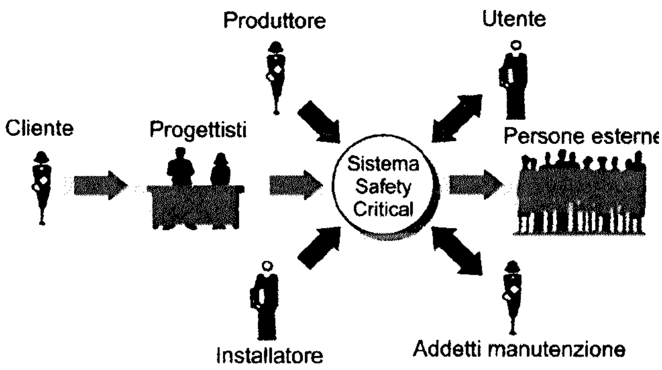

Fig. 1.1 Gli attori della Safety

Nonostante tanti studi sulla Safety, questo argomento ha bisogno di ulteriore approfondimento anche perchè non bisogna sottovalutare il fatto che i continui progressi tecnologici aumentano le potenzialità dei sistemi computerizzati, ma anche la loro complessità e più aumenta la complessità di un sistema più diventa difficile assicurare la sua Safety: "affrontare la complessità nello sviluppo, la costruzione, la manutenzione e l'uso di un sistema è un problema serio " [104].

Inoltre non bisogna dimenticare che la Safety è una caratteristica di primaria importanza (ma forse sarebbe più corretto dire la più importante) del sistema, perchè non solo influenza tutti gli stadi del ciclo di vita del sistema, ma anche ogni persona che entra in contatto con esso sia indirettamente sia direttamente (v. Fig. 1.1).

L'insoddisfacente situazione degli standard attuali e del processo di standardizzazione è ben illustrata in un recente articolo [47] che svolge un'analisi critica a partire dallo standard IEC 1508.

La Safety di un sistema computerizzato può essere suddivisa in:

- *primaria* - indica l'assenza di livelli inaccettabili di rischio di elettrocuzione, incendio ecc. causati direttamente dall'hardware del computer;
- *funzionale* - indica l'assenza di livelli inaccettabili di rischio riguardanti l'impianto sotto controllo e connessi al corretto funzionamento dell'hardware e del software del computer.

Oggi si tende poi a considerare anche una forma di "*Safety indiretta*", relativa ai rischi indotti dalle conseguenze indirette di un malfunzionamento in sistemi informativi relativi alla Safety, come il sistema automatico di smistamento delle telefonate ad un centralino di un ospedale o di un pronto soccorso.

Questa ricerca tuttavia si limiterà ad esaminare aspetti di Safety funzionale, che in settori come quello dei trasporti o quello industriale è di gran lunga la più importante perlomeno dal punto di vista della severità delle potenziali conseguenze degli azzardi.

1 sistemi relativi alla Safety possono essere classificati in:

- *sistemi di controllo*, che di solito svolgono il duplice ruolo di controllare l'equipaggiamento e di svolgere funzioni di Safety;

- *sistemi di protezione*, che usano sensori per rilevare condizioni di guasto e cercano di mitigare gli effetti del malfunzionamento, nella maggior parte dei casi arrestando il funzionamento dell'equipaggiamento sotto controllo, ragion per cui vengono indicati comunemente come "*shutdown systems*".

L'interazione tra i sistemi di controllo e protezione e l'impianto avviene per mezzo di sensori e di attuatori che monitorano e controllano alcuni parametri. Comunque va sempre tenuto ben presente, che nonostante il continuo incremento dell'uso di sistemi elettronici programmabili nello svolgimento di funzioni di Safety, normalmente la Safety di un sistema è assicurata da diversi sistemi di protezione basati su diverse tecnologie (meccaniche, elettriche, elettroniche ecc.), e ci sono molte situazioni in cui l'uso dei sistemi non programmabili è vantaggioso. La figura 1.2 illustra, secondo l'ISA, la ripartizione delle probabilità di guasto nei componenti di una catena di controllo.

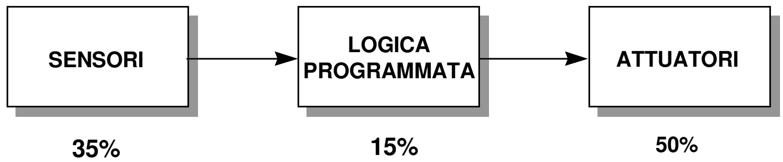

Fig. 1.2 Ripartizione della probabilità di guasto in una catena di controllo

Un discorso a parte, per la sua rilevanza nei sistemi Safety-Critical, merita la notevole importanza della Reliability dei sistemi basati su computer, poiché un fallimento in questi sistemi normalmente porta a condizioni operative fuori del normale, e, queste hanno una probabilità molto più alta di portare ad incidenti [19].

Gli studi più recenti sostengono che **la Reliability è una condizione necessaria (anche se non sufficiente) per ottenere la Safety**, pertanto assume particolare rilevanza il fatto che i sistemi elettronici programmabili abbiano questa qualità.

Lo svantaggio principale dei sistemi computerizzati in applicazioni Safety-Critical è la complessità sia a livello hardware che software, insita nella natura stessa di questi sistemi; inoltre, in certe applicazioni, controllo distribuito, requisiti real-time stringenti, ed esecuzione concorrente tendono ad aumentare ulteriormente la complessità, così come la richiesta da parte dei clienti di sistemi sempre più grandi ed elaborati [104]. I sistemi complessi sono più difficili da progettare, e ciò porta ad una maggiore probabilità di errori di progetto, sono difficili (se non impossibili) da testare col solo testing funzionale (si pensi a programmi software con migliaia di possibili cammini di esecuzione o ai microprocessori il cui numero di possibili modi di fallimento è addirittura incalcolabile [118]) e ciò porta ad una grande probabilità di difetti non rilevati.

Un principio di buona progettazione e sviluppo ormai assodato per i sistemi Safety-Critical è quello della "difesa in profondità e diversità" (defence in depth and diversity), con cui si intende raggiungere il previsto livello di Safety mediante l'uso di sistemi di difesa sovrapposti e ridondanti.

Un sistema Safety-Critical ha un suo ciclo di vita che va dalla specifica dei requisiti fino alla dismissione, ed ogni fase del ciclo deve essere studiata con la massima attenzione, perché una sottovalutazione delle possibili conseguenze dei suoi azzardi può avere esiti drammatici; perfino la dismissione di un impianto Safety-critical può creare seri problemi, basti pensare alle difficoltà e ai pericoli connessi con lo stoccaggio e lo smaltimento dei materiali radioattivi a lungo tempo di vita derivanti dallo smantellamento di una centrale nucleare. Il ciclo di vita di un sistema Safety-Critical secondo la norma EN 50126 è illustrato nella figura 1.3.

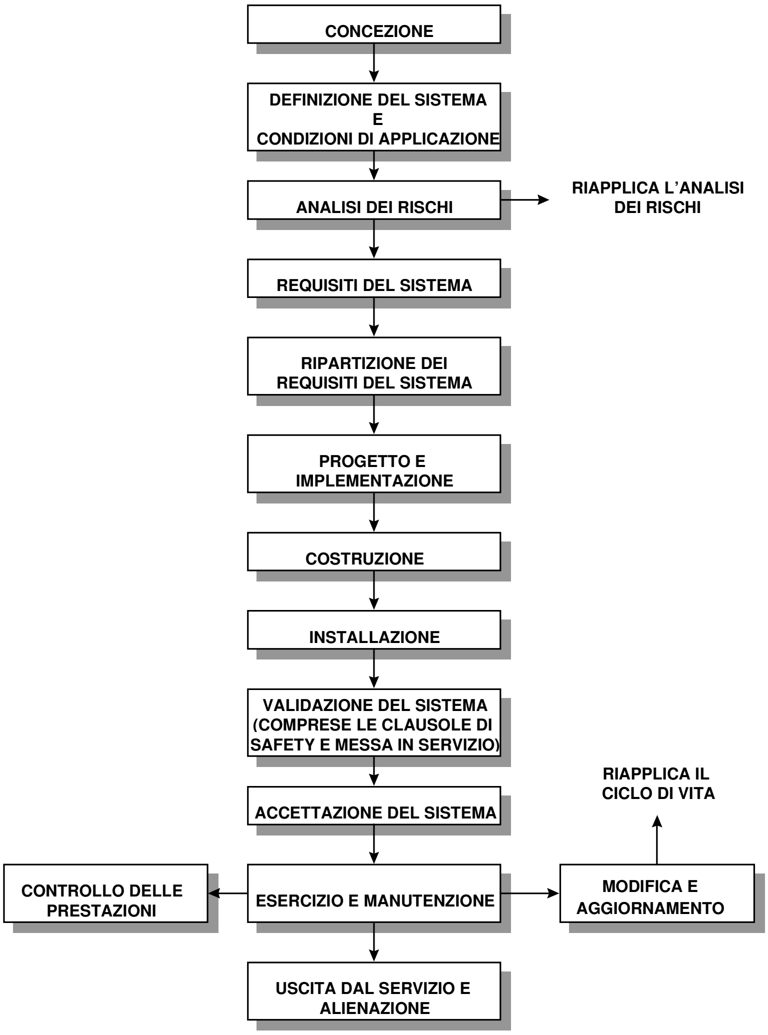

Fig. 1.3 Ciclo di vita del sistema

All'interno di questo ciclo ci si occuperà solo della parte relativa allo sviluppo del sistema Safety-Critical, di cui si danno nelle figure 1.4 e 1.5 due diversi modelli, tratti da EN50128. In particolare si focalizzerà l'attenzione sul ciclo di vita del software poichè, anche se la certificazione viene posta come ultima fase nei modelli, essa coinvolge un'attività di verifica della conformità agli standard qualitativi e quantitativi in tutte le fasi dello sviluppo.

Far coincidere l'attività di certificazione e quella di verifica, affiancando gli organismi di certificazione al team di sviluppo, può portare ad una significativa riduzione dei costi, come ci insegnano molte esperienze di grande successo.

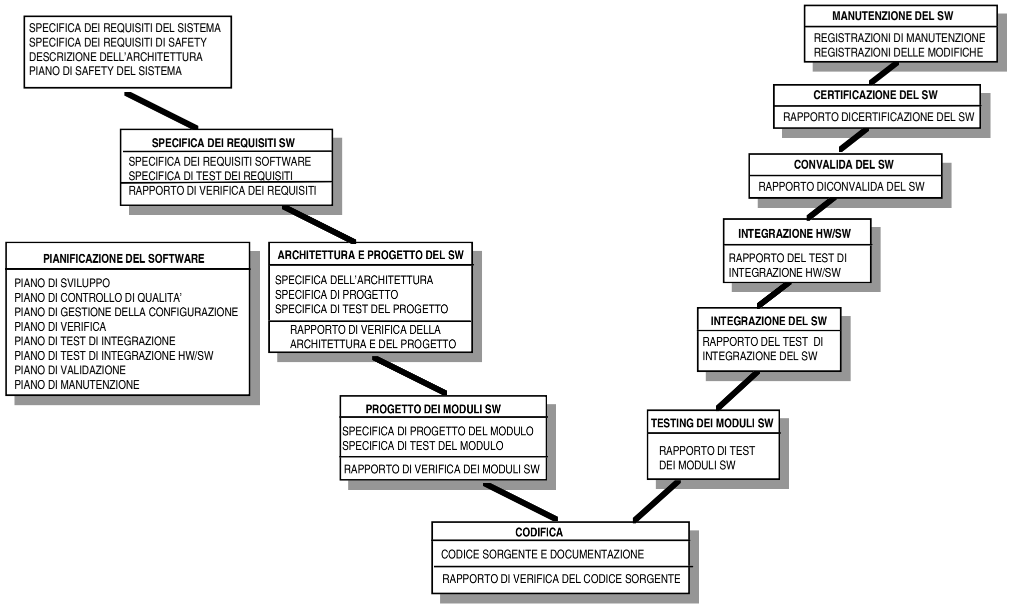

Fig. 1.4 Fasi del ciclo di vita del software (1)

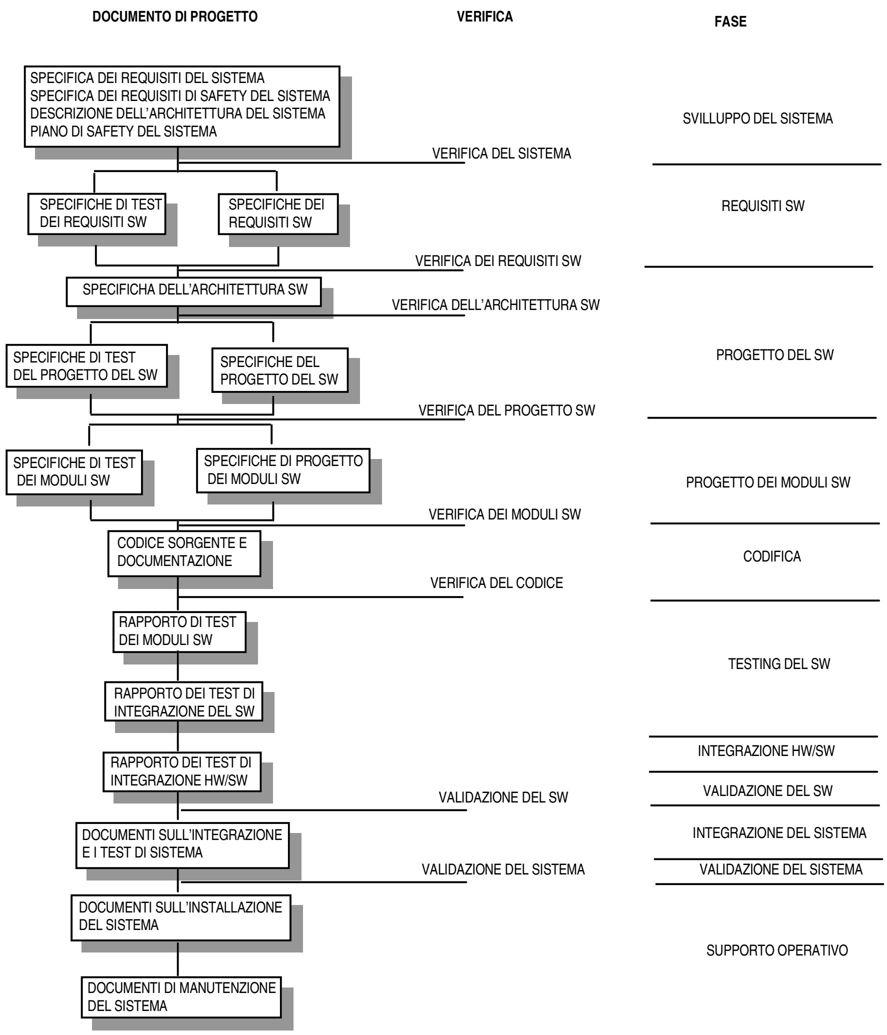

Fig. 1.5 Fasi del ciclo di vita del software (2)

2. ATIPICITÀ DEL SOFTWARE

Il software è, probabilmente, il componente dei sistemi safety critical che dà maggiori preoccupazioni, soprattutto per i problemi particolarmente difficili da risolvere riguardanti la certificazione e gli standard [121]; è infatti molto difficile raggiungere accordi a livello internazionale per l'emanazione di normative valide in paesi con culture, legislazioni e tradizioni industriali diverse (basti pensare a quanti anni ha richiesto la definizione di EN 50128). Una volta emanati gli standard, risulta estremamente arduo riuscire a *dimostrare* il raggiungimento dei requisiti di ultrareliability software che, per molti di essi, sono specificati nell'ordine di $<10^{-7}$ fallimenti/ora, valore ancora molto lontano dallo stato dell'arte nel campo della verificabilità sperimentale.

Il software non è intrinsecamente pericoloso, non può infatti di per se stesso provocare incidenti o danni a persone o cose quali elettrocuzione, incendi ecc., come invece succede con molti dispositivi hardware, ma diventa safety-critical quando lo si usa con funzioni di protezione e controllo in sistemi potenzialmente pericolosi; infatti la safety è una proprietà di sistema e il software può essere valutato sicuro o non sicuro solo se considerato come parte dell'intero sistema.

Per la sua natura intrinseca il software, a differenza dell'hardware:

- non subisce l'usura del tempo;
- non si "rompe" a causa del sovraccarico [99];
- non ha fallimenti casuali, ma solo sistematici e correlati al suo progetto [118];
- non è influenzato negativamente da fenomeni ambientali come campi elettromagnetici o particelle alfa.

I difetti del software sono presenti sin dall'inizio e, se non vengono rimossi, permangono per tutta la vita del sistema [32]. Di contro è universalmente riconosciuto che tutti i programmi contengono inevitabilmente dei "bugs" quali: errori nella specifica software, errori nella codifica, uso di variabili non inizializzate ecc..

Gli errori di progetto del software sono molto difficili sia da evitare, sia da "tollerare" con meccanismi di fault tolerance [91], specie se dipendono da errori di specifica in quanto, in questo caso, non saranno rilevati dal testing, che si limita a verificare che un programma svolga le funzioni specificate, e saranno presenti in tutte le versioni del software "N-version programming", poichè, in questo caso, a causa della specifica comune sbagliata cade l'ipotesi basilare di questa tecnica che "programmi sviluppati indipendentemente falliranno indipendentemente" [83].

In genere è di gran lunga più impegnativo dimostrare che un programma software svolge le sue funzioni correttamente che dimostrare la corretta funzionalità di sistemi tradizionali analogici. I sistemi software operano in ambienti non deterministici, da cui ricevono sequenze di input e ad ogni input producono una risposta corretta o sbagliata, quindi il software produce errori in maniera imprevedibile; la sequenza di errori si comporta come un processo stocastico [32]. I fallimenti software non sono causati da un livello eccessivo di un qualche fattore identificabile (come succede nell'hardware), ma dall'occorrenza di particolari e non ovvie combinazioni di eventi [91].

Il fatto di operare nel discreto è una delle principali (se non la principale) ragioni

della difficoltà di convalidare il software. Mentre i sistemi che operano nel continuo sono stati oggetto di studio per secoli ed hanno una solida base teorica grazie al calcolo differenziale, i sistemi discreti vengono invece studiati da molto meno tempo e non hanno una base teorica così ben definita.

Un altro motivo della maggior difficoltà nel convalidare il software è che in un sistema computerizzato bisogna considerare un numero molto maggiore di stati rispetto ad un sistema "tradizionale". Per i sistemi di controllo hardware analogici, è possibile usare l'analisi continua e l'interpolazione dei risultati dei test, mentre le ipotesi che discendono dalla continuità non possono essere usate nella convalida del software, poiché la sua natura discreta (il cambiamento anche di un solo bit può avere conseguenze notevoli) fa sì che i metodi di testing siano molto meno efficaci [91]. Ad esempio l'hardware di un computer ha una struttura ripetitiva che ne facilita l'analisi, mentre il software ha un numero di possibili stati così grande da rendere impraticabile, nei casi non banali, il testing esaustivo [86].

I potenziali modi di fallimento dello hardware sono generalmente predicibili e limitati in numero (anche se in realtà questa ipotesi ha importanti eccezioni, come il caso dei processori che hanno un comportamento simile al software per quanto riguarda il numero di modi di fallimento), **invece i modi di fallimento software sono potenzialmente impredicibili e di numero quasi illimitato.**

Da quanto detto sembrerebbe discendere (l'ovvia) conseguenza che è più semplice l'analisi di un sistema senza software che quella di uno con il software, ma le cose non sono così semplici. Il maggior problema da gestire nei sistemi (software e non) è la loro complessità [104]. Spesso un sistema di controllo può essere realizzato sia in versione hardware che software, e per un sistema banale con funzioni limitate come un termostato è certamente preferibile la versione "hardware" [118]. Invece un sistema "puramente hardware" con una funzionalità richiesta notevole implica spesso una complessità tale da renderlo difficile da convalidare quanto la sua versione software con analoghe funzioni [91], con in più lo svantaggio di una modificabilità, flessibilità e adattabilità ben più limitate e costose. **Pertanto, nei sistemi safety-critical, sarebbe sempre preferibile la soluzione più semplice a parità di funzionalità,** anche se purtroppo spesso i criteri di scelta sono ben diversi, basti pensare che l'espansione dell'uso del software nei sistemi di controllo e protezione è dovuta anche a considerazioni prettamente economiche [91].

Il metodo forse più tradizionale ed utilizzato nell'hardware, per assicurare la reliability, è il life testing, ma si è dimostrato che esso è inadeguato per l'eccessivo tempo richiesto nel caso di software ultrareliable (tipico dei sistemi safety-critical) [32]. Pertanto l'approccio oggi comunemente usato per quantificare la reliability del software è la combinazione di fault tolerance e modelli statistici.

Le tecniche più mature per la valutazione dell'affidabilità del software sono quelle che modellano la crescita della reliability come risultato della rimozione di difetti [91]: i cosiddetti "reliability growth models", ma anche essi si sono dimostrati inadeguati a dimostrare tassi di fallimento orario $< 10^{-7}$ in tempi ragionevoli.

I valori richiesti dagli standard (IEC 1508, EN 50126) di 10^{-8} fallimenti/ora per livelli di integrità safety 4, che, per i componenti hardware, vengono assicurati prevalentemente con metodi statistici basati sul testing, non sono assolutamente assicurabili per il software, usando tali mezzi, poiché sarebbe richiesta una quantità impraticabile di test [123], ed è stato dimostrato matematicamente che il testing esaustivo (almeno per programmi non banali) è impossibile [61], a causa degli innumerevoli percorsi possibili nel software, dei possibili input e dei possibili modi di fallimento; basti pensare che **un sistema con appena 40 input binari ha uno spazio di input di 2^{40} combinazioni e che, svolgendo un test ogni millisecondo, si impiegherebbero circa 35 anni a testare tutte le combinazioni** [118], quindi è facile immaginare quali possano essere i valori di copertura dei test (= numero di test/spazio di input) per prodotti software anche limitati.

La quantità di testing, oltre a ben evidenti limiti temporali, ha anche notevoli limiti economici, in quanto il fattore di costo dei test (= spesa per test/(spesa per i test + spesa per il progetto)), secondo una regola dettata dall'esperienza, ma che è molto più seguita di tante regole con basi ben più solide, deve variare tra 0,25 e 0,75 a seconda della complessità del sistema e del livello di integrità safety richiesto [39].

Inoltre, non solo è impossibile il testing esaustivo per la maggior parte del software, ma è anche difficile fornire condizioni di test realistiche, poiché spesso le condizioni operative differiscono dalle condizioni prese in considerazione durante la simulazione [88]; infatti c'è una notevole differenza fra condurre il testing su una workstation e testare il software sull'hardware proprio del sistema da costruire. Ciò è evidenziato dal fatto che, come rilevato da Parnas e altri [107], sostanzialmente nessun software di nuova realizzazione ha mai operato bene la prima volta che è stato usato nell'applicazione reale alla quale era destinato. Nel testing svolto su una workstation prima che l'hardware obiettivo sia pronto, si pone infatti l'attenzione quasi esclusivamente sulla funzionalità e gli aspetti non temporali della specifica [41].

A ciò si aggiunga che oltre la metà di tutti gli errori o omissioni scoperti nel software operativo (safety critical e non) può essere attribuito ad errori di specifica, che, ovviamente, nessun test può rivelare [88] e che Villemeur [122] e Bowen e Stavridou [26] sostengono che addirittura i 2/3 dei fallimenti software possono essere attribuiti ad errori di specifica; si capirà quindi chiaramente perché la certificazione del software desta tante preoccupazioni.

Inoltre Jaffe e altri [76] affermano che correggere errori commessi nella fase di specifica dei requisiti può costare oltre 200 volte in più che correggere quelli compiuti nelle fasi successive.

Queste considerazioni impongono un cambiamento significativo nel modo di agire dei convalidatori e certificatori, che dovranno abbandonare, almeno parzialmente, i metodi di testing dinamico, per accettare che la costruzione di sistemi sicuri sia assicurata da tecniche di specifica e verifica formali [99], anche alla luce della constatazione che i sistemi safety critical sono quasi tutti (se non tutti) real-time [67], mentre gli strumenti per il testing dinamico attualmente in uso, non sono assolutamente adeguati all'analisi delle proprietà temporali dei programmi [41] e non sono certificati per i livelli di integrità necessari [118].

Per quanto riguarda la convalida del software, l'analisi dinamica dovrà passare dal ruolo centrale, che molti (soprattutto gli ingegneri dell'hardware) le attribuiscono, ad un ruolo complementare, focalizzandosi sulle cose che non sono coperte adeguatamente dall'analisi statica [88] e dalla modellazione e prototipazione per mezzo dei metodi formali e di altre tecniche a ciò finalizzate.

Le tecniche di analisi statica e di modellazione e prototipazione del software dovranno essere applicate a partire dalla specifica e dalla analisi dei requisiti e non dall'implementazione come accade per il testing. Pertanto, per il software, le attività di verifica devono iniziare dalle primissime fasi e continuare per tutto il ciclo di vita del suo sviluppo (possibilmente considerando tale attività come parte della certificazione al fine di ottenere sensibili risparmi, come nel caso del software della metropolitana parigina [51]).

Bisogna finalizzare tutte le attività di sviluppo del software al raggiungimento della sicurezza, usando i metodi formali nella specifica dei requisiti, nello sviluppo del progetto e nella verifica. Inoltre il software dev'essere progettato per essere verificabile [88], con precise caratteristiche quali:

- determinismo;
- identificazione e separazione di oggetti e funzioni safety critical;
- semplicità.

La testabilità, per quanto possibile, va assicurata con particolari tecniche di codifica [123, 50].

In sostanza, *nel software, la safety è una caratteristica che si costruisce e che si verifica in ogni passo dello sviluppo*. I problemi legati all'atipicità del software, e quindi dei mezzi per verificare la sua reliability e la sua safety, sono amplificati in molti paesi, fra cui l'Italia (dove nessuno standard di emanazione nazionale raccomanda l'uso di metodi formali nello sviluppo del software e dei sistemi safety critical), da una normativa carente che non stabilisce né i requisiti minimi di preparazione dei progettisti del software, né quelli cui debba sottostare il progetto stesso.

3. ORIENTAMENTO LEGISLATIVO E NORMATIVO: VERSO NUOVI STANDARD

In una società, come quella attuale, in cui la tecnologia progredisce con velocità impressionante, soprattutto nel campo dell'hardware e del software dei sistemi computerizzati, non meraviglia che la legislazione, che, come è noto, ha tempi di sviluppo ben diversi, faccia fatica ad adeguarsi tempestivamente. Ciò è particolarmente vero in Italia, dove tutte le leggi devono essere approvate da entrambe le camere e l'attività legislativa è ulteriormente rallentata dall'enorme mole di proposte di legge e di leggi in vigore.

Nel campo del software safety-critical e in particolare di quello destinato ai sistemi di trasporto su rotaia, i riferimenti legislativi, soprattutto in Italia, sono minimi e, di regola, non sono altro che mere estensioni di leggi già in vigore, spesso molto vecchie [100].

La legislazione italiana in vigore, in campo ferroviario, è costituita dal R.D.L. 11-10-1934 n. 1948, dal D.P.R. 11-07-1980 n. 753 e dalla L. 26-02-1992 (che si limita sostanzialmente ad attribuire le responsabilità e i fondi per la realizzazione di " sistemi di trasporto rapido di massa a guida vincolata" agli enti locali).

Si spera di rimediare almeno in parte a questa carenza legislativa, percepibile soprattutto a livello di normativa tecnica avanzata (come ad esempio nel caso della regolamentazione dei sistemi a marcia automatica), recependo le norme tecniche europee EN 50126 e EN 50128, pubblicate nella versione definitiva a fine '98, ed EN 50129 (già definita), che andranno ad adeguare e completare la norma UNI UNIFER 10257 (gestione automatica di sistemi di trasporto di tipo metropolitano). Le date previste per l'accoglimento di questi standard sono, dagli stessi, fissate come segue:

EN 50126	recepimento come standard nazionale: ottobre 1999; abolizione di altri standard nazionali in conflitto: ottobre 1999.
EN 50128	recepimento come standard nazionale: gennaio 2000; abolizione di altri standard nazionali in conflitto: gennaio 2002.

L'adozione di standard tecnici comuni per i sistemi ferroviari europei avrà evidenti ripercussioni positive, ma il cammino di tali norme è stato ed è molto travagliato: esse hanno subito interventi di revisione e di riscrittura che, oltre a ritardarne molto l'emanazione, ne hanno limitato a tal punto la portata da farle definire da esperti inadeguate [100], pertanto già fin d'ora si prevede che sarà necessaria un'ulteriore edizione della normativa che rimedi alle manchevolezze del progetto attuale e che venga prontamente adottata dai paesi CEE.

Un aspetto molto interessante è che le norme tecniche in progetto per il software ferroviario (EN 50128) non sono specifiche per i sistemi di trasporto su rotaia, ma, sono basate sui precedenti del progetto IEC 1508 (recentemente ridenominato IEC 61508) [19]. Per la precisione tutte le sezioni normative della EN 50128, ad eccezione della 6, sono basate su IEC SC65A/WG9 "Software for Computers in the Application of Industrial Safety-Related Systems" del 1991, mentre i capitoli di EN 50126, espressamente richiamati dalla sez. 6 di EN 50128, riguardanti la specifica dei requisiti di sistema e l'analisi dei rischi sono evidentemente derivati da IEC SC65A/WG10 "Functional Safety of Electrical/Electronic/Programmable Electronic Systems; Generic Aspects: Part 1,General Requirements" del 1992. Pertanto le norme imposte da EN 50128 hanno similitudini impressionanti con l'attuale progetto di norma IEC 1508 del 1995, parte 3 "requisiti software", destinato principalmente ai sistemi di controllo industriale correlati con la safety.

Questa notevole "similitudine" fra norme tecniche, destinate al software di settori diversi, è giustificata solo parzialmente dal fatto che IEC 1508 è uno standard generico e non limitato a una specifica area di applicazione industriale; in realtà, EN 50128 dovrebbe essere una norma "tagliata su misura" per il software ferroviario e quindi non quasi identica a una generica normativa industriale., Ciononostante, la nuova norma può avere una funzione positiva, poiché permette di ridurre al minimo i tempi di apprendimento dei progettisti, purché non si trascuri il fatto che i sistemi safety-critical ferroviari sono anche, nella maggior parte (se non in tutti) i casi, sistemi di tipo real-time, mentre i sistemi industriali a volte possono non avere requisiti di tempo reale, e, pertanto, purchè nelle procedure di sviluppo e verifica si facciano appropriate distinzioni tra software real-time e non, cosa che nella maggioranza dei progetti attuali purtroppo non avviene.

Un altro aspetto negativo e molto criticato della nuova normativa europea è l'uso di espressioni come "recommended" (raccomandato) e "highly recommended" (fortemente raccomandato) che risultano per lo più molto ambigue [121, 47].

Inoltre, nel convegno IEE/BCS di Glasgow del '95 ventisette personaggi di primo piano, provenienti dal mondo accademico, industriale, e da organizzazioni tecnico-professionali come BCS e IEE, hanno criticato pesantemente il fatto che "le pressioni sugli organismi di standardizzazione europea sono tali che, in alcuni casi (il riferimento è al progetto IEC 1508, ma si può trasferire tranquillamente al suo omologo EN 50128), gli emergenti standard europei richiedono l'uso di metodi formali in misura inferiore agli standard nazionali" (del mondo anglosassone).

In Italia queste norme europee, così giustamente criticate altrove, sono comunque un grosso passo avanti, in quanto non esistono standard nazionali che obblighino, o almeno raccomandino, l'uso di metodi formali nella produzione del software safety-critical.

L'attuale situazione italiana, in questo senso, è particolarmente preoccupante sotto tre aspetti:

1. **in un contesto europeo la priorità dovrebbe essere quella di rafforzare i metodi per dimostrare la safety e non ignorarli;**
2. **è ormai assodato che lo sviluppo di strumenti e di guide per l'uso dei metodi formali necessita di un supporto governativo [121];**
3. **nella società postindustriale prevarranno i paesi che riusciranno a produrre prodotti ad alta tecnologia, poiché quelli a bassa tecnologia verranno prodotti inevitabilmente nei Paesi emergenti, e i metodi formali si stanno rilevando un validissimo aiuto per produrre prodotti altamente innovativi, e al tempo stesso sicuri, nel settore dell'informazione, che evidentemente è il più importante nella società dell'informazione.**

Infine è doveroso notare che una chiave di lettura del ritardo italiano nell'adozione dei metodi formali per la produzione industriale di software safety critical è la **mancanza di norme cogenti** in tal senso, che *permette di produrre programmi forse meno costosi, ma certamente meno sicuri*. Al contrario, negli altri paesi industrializzati e soprattutto nel mondo anglosassone sono ben presenti (e da diversi anni) standard per il progetto e la produzione software, come ad esempio UK MoD 00-55 (1991), Canada AECB 2.234.1 (1992), e Canada OH 982C-H69002 (1993), nei quali l'uso di metodi formali è obbligatorio, nonché altri standard che ne raccomandano l'uso in maniera non formalmente cogente, ma difficilmente eludibile (come ESA PSS-05-0 "Software Engineering Standards" (1991), RIA "Safety Related Software for Railway Signalling" (1991)).

In questo panorama europeo con luci e ombre una nota di speranza è costituita dal fatto che si sono formati da tempo gruppi di lavoro per produrre standard internazionali specifici per applicazione in molti settori vitali come l'industria petrolchimica, biomedicale, ferroviaria ecc.[28], per cui, entro pochi anni, dovremmo superare l'attuale situazione di normative ancora troppo generiche.

4. La norma europea EN 50128 e l'analisi e la specifica del software

La norma europea EN 50128 "*Applicazioni ferroviarie: software per i sistemi* *ferroviari di controllo e protezione*", emessa dal CENELEC nell'autunno del 1998, fa parte di un più ampio raggruppamento di standard che comprende, tra le altre, le seguenti norme che sono ad essa strettamente correlate:

- EN 50155 "*Applicazioni ferroviarie: equipaggiamento elettronico usato nel materiale rotabile*";
- EN 50126 "*Specifica e dimostrazione dell'affidabilità (reliability, availability, maintanability e safety (RAMS)) per applicazioni ferroviarie*", che è una norma definitoria molto generale dalla quale dipartono le altre norme particolari sopra elencate.
- ENV 50129 "*Applicazioni ferroviarie: sistemi elettronici rilevanti per la safety dei sistemi di segnalamento*";

Questo insieme di norme ha lo scopo di unificare e armonizzare gli standard presenti nei vari paesi della comunità europea, anche se spesso opera dei "compromessi" criticati da molti studiosi [121].

I nuovi standard non rappresentano comunque delle grosse novità, poiché ENV 50129 ricalca sostanzialmente Mu 8004, lo standard tedesco per le applicazioni ferroviarie, mentre EN 50128, come già rilevato, è strettamente imparentato con IEC 1508, in quanto i due standard derivano, per così dire, dagli stessi progenitori.

La norma EN 50128 specifica le procedure e i requisiti tecnici per lo sviluppo di sistemi elettronici programmabili per l'uso in applicazioni ferroviarie di controllo e protezione, che abbiano o meno implicazioni colla safety, e si applica esclusivamente al software (inclusi programmi applicativi, sistemi operativi, strumenti di supporto, firmware) e all'interazione fra esso e il sistema. Essa, naturalmente, non è retroattiva, quindi si applica, nella sua interezza, principalmente allo sviluppo di nuovo software e a sistemi esistenti solo se questi subiscono modifiche rilevanti; in caso di cambiamenti minori si applica, come minimo, la clausola 16 della norma che riguarda la manutenzione del software.

La scelta dei compilatori di allegare una bibliografia con testi usciti anche da oltre vent'anni e comunque antecedenti il 1990, pur forse criticabile, riflette tuttavia la scelta (disapprovata da diversi esperti) di inserire nello standard solo tecniche consolidate (ma sarebbe più appropriato dire vecchie); in alcuni casi, come quello della specifica dei requisiti software, ciò viene fatto a titolo solo esemplificativo, ma in altri, come in quelli della verifica e della convalida del software, in modo cogente nonostante che il mondo accademico ed industriale stiano imboccando strade diverse. Basta confrontare le tecniche di convalida raccomandate da EN 50128 e quelle consigliate da [10] per rendersi conto del solco che separa lo standard dagli orientamenti moderni sul tema della convalida.

La EN 50128 adotta la teoria dei rischi per la valutazione della safety in via non deterministica, così come altre normative in questo settore [100]. Essa impone agli sviluppatori la conformità a requisiti qualitativi (l'adozione di tecniche raccomandate per ogni fase del ciclo di vita dello sviluppo del software e per ogni livello d'integrità) e il raggiungimento di obiettivi quantitativi, consistenti nella realizzazione di sistemi con ben quantificate (e limitate) probabilità di fallimento/ora che variano a seconda del livello di integrità richiesto.

La valutazione del rischio è basata sulla combinazione della frequenza di occorrenza di un azzardo (una prassi molto criticata [88]) e della severità delle sue conseguenze.

Questo tipo di analisi quantitativa in realtà ha poco o nessun senso soprattutto per il software safety-critical del quale, allo stato attuale delle conoscenze, è notoriamente difficile quantificare la reliability e probabilmente impossibile quantificare la safety [90]. D'altra parte la stessa normativa europea, in una versione preliminare [EN 50128, draft febbraio 1994, cap. 5.8], ammetteva che, anche seguendo le sue raccomandazioni, non era detto che si sarebbero raggiunti gli obiettivi integrità software voluti.

Questo progetto di standard propone, per ogni fase del ciclo di vita, diverse tecniche che, a seconda del livello di integrità software, possono essere:

- vietate;
- non raccomandate;
- prive di raccomandazioni (positive o negative);
- raccomandate;
- fortemente raccomandate;
- obbligatorie;

quest'ultimo livello di prescrizione ricorre in casi rari come per l'assicurazione della qualità del software, certamente molto importante, ma meno vitale di altri aspetti quale ad esempio la specifica dei requisiti, visto che i due terzi dei fallimenti software sono attribuibili ad errori di specifica [122]. Si può scegliere con notevole libertà fra i metodi inclusi nelle tabelle della norma; infatti, spesso, per il medesimo livello di integrità software e per la medesima fase dello sviluppo, vi sono tre o più tecniche "altamente raccomandate" e bisogna optare per una o più di esse. **Uno spettro di possibilità così ampio vanifica quasi del tutto l'obbligo di spiegare dettagliatamente nel piano di assicurazione della qualità del software la ragione per cui non si è usato un metodo "altamente raccomandato": basta usarne un altro alternativo con lo stesso livello di raccomandazione.**

Il fatto che la norma non faccia scelte precise e si mantenga su termini molto generici, pur essendoci chi - e non sono pochi - ritiene che sia frutto di compromessi preoccupanti [73], può da un lato essere apprezzabile in quanto frutto di uno sforzo di armonizzazione con altri standard, come il progetto IEC 1508 e Draft Defence Standard 00-56 (un progetto di standard britannico) [118]; d'altra parte è, per certi aspetti, fortemente criticabile, poiché si dovrebbe almeno imporre obbligatoriamente l'uso dei metodi formali nella specifica, nel progetto e nella verifica di funzioni safety-critical.

Ora si passerà ad analizzare le caratteristiche principali della norma EN 50128, mettendole in relazione con i più recenti ed accreditati orientamenti accademici e con la prassi industriale.

4. 1 Livelli di integrità software

Per analizzare compiutamente il concetto dei livelli di integrità software, che è di fondamentale importanza nella norma EN 50128, bisogna mettere bene a fuoco i concetti che stanno alla sua base.

Una definizione, accettata da molti nel mondo accademico, è che *l'integrità safety è la probabilità che un sistema safety-critical svolga le funzioni di safety richieste (non necessariamente tutte le funzioni) nelle condizioni ambientali stabilite in uno specificato periodo di tempo [29] [11].*

C'è un contributo casuale e uno sistematico all'integrità safety [13]:

- *l'integrità hardware* è connessa con i fallimenti casuali dell'hardware;
- *l'integrità sistematica*, è connessa agli errori di specifica di progetto (nell'hardware e nel software) e agli errori d'implementazione del software. L'integrità software è quella parte dell'integrità safety relativa a fallimenti software pericolosi.

Sebbene la safety sia sempre molto importante, la sua importanza relativa in differenti situazioni è chiaramente connessa ai rischi coinvolti. Sebbene l'integrità safety potrebbe essere rappresentata quantitativamente è più comune assegnare al sistema uno dei *livelli* di integrità safety predefiniti.

I livelli di integrità safety sono un concetto comune a tutti gli standard, anche se il loro numero e la loro definizione variano molto da normativa a normativa. La norma IEC 1508 assegna dei tassi di fallimento obiettivo per ognuno dei suoi quattro livelli di integrità safety, cioè impone il massimo numero di fallimenti che, in un dato periodo di tempo (in questo caso l'anno) un sistema con un certo livello di integrità safety dovrebbe potere subire. L'assegnazione di un livello di integrità a un sistema o a un'unità di esso si basa sulla classificazione dei rischi associati.

Nel processo di assegnazione dei livelli di integrità (Fig. 4.1) si parte quindi con la valutazione della *severità delle conseguenze* e della *frequenza di occorrenza* di un evento che crea azzardo, dalle quali si ricava la *classificazione del rischio associato*. Basandosi sulla classificazione del rischio si opera una classificazione dell'integrità, distinguendo in integrità hardware e integrità sistematica. Dalla classificazione dell'integrità sistematica si evince la classificazione dell'integrità software. Un dato livello d'integrità software può essere assegnato ad una parte di un programma in base all'importanza del corretto funzionamento di quel modulo software per la sicurezza del sistema.

Se assegnare i livelli di integrità software, pur con tutte le cautele raccomandate dagli studiosi [88], appare non troppo difficile a patto di avere svolto un'analisi dei rischi di buona qualità, sulla reale significatività del calcolo della *probabilità di occorrenza* di un azzardo, che, insieme alla sua *severità*, è la base primigenia del calcolo dei livelli di integrità software dei vari moduli, ciò che appare veramente molto difficile è non solo e non tanto ottenere i necessari valori dei tassi di fallimento (dell'ordine di meno di 10^{-7} fallimenti/ora), ma anche dimostrarlo, dato che non si conosce nessun metodo di testing in grado di dimostrare questo grado di reliability per un software safety-critical.

Ad onor del vero, gli ingegneri responsabili del sistema Paris Metro hanno dichiarato di aver ottenuto tassi di 10^{-8} fallimenti/anno, ma questo dato, anche se molto confortante, appare difficile da dimostrare alla luce dei più recenti ed autorevoli studi sulla reliability di sistemi safety-critical. Quantificare la reliability del software connesso alla safety è un compito di enorme difficoltà (o addirittura impossibile in molti casi [32]), secondo, per difficoltà, solo a quantificare la safety, cosa con ogni probabilità adirittura impossibile [90]. Cribbens illustra le difficoltà nel quantificare la reliability di un sistema di segnalazione ferroviaria, mostrando che sia le stime della safety sia quelle della reliability dello stesso sistema, svolte da team diversi, differiscono di diversi ordini di grandezza [40].

Gli obiettivi di reliability sono, più o meno, gli stessi per tutto il software safety-critical (ferroviario, aereo o nucleare che sia) [86] ed alcuni autori non si limitano a sostenere che la reliability è una qualità necessaria per la safety, ma asseriscono che la safety non è identificabile come una proprietà separata dalla reliability [107], ma, nonostante che gli standard abbiano, sulla scorta degli studi di Parnas, recepito l'importanza della reliability, imponendo tassi di fallimento orari tra 10^{-9} e 10^{-7}, tali risultati sono indimostrabili affidandosi ai metodi di testing convenzionali, poiché la quantità di test richiesta sarebbe troppo grande per essere realizzabile [66, 123, 50] specie se abbiamo a che fare con software safety-critical real-time [32].

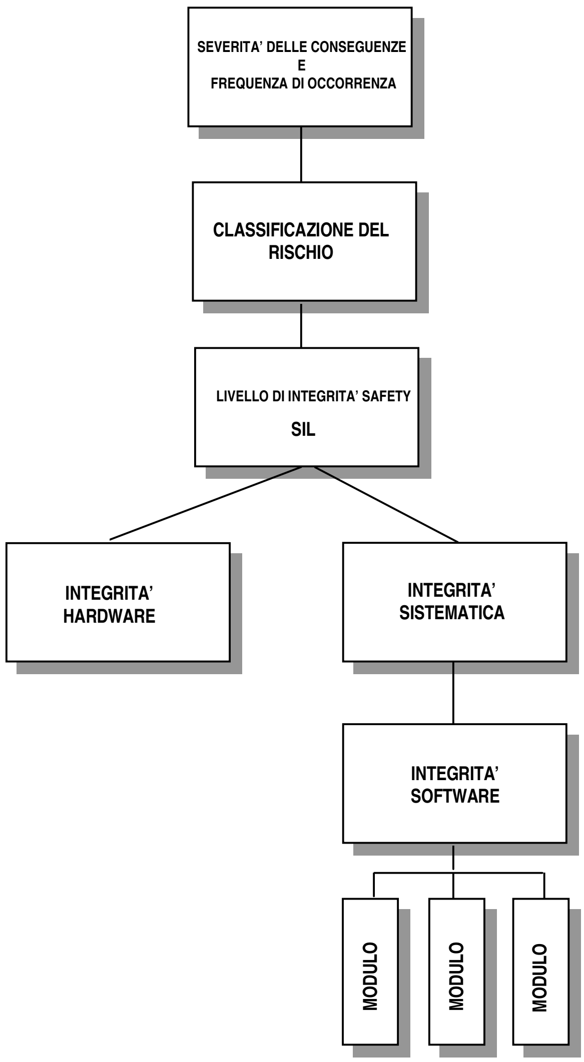

Fig. 4.1 Assegnazione dei livelli di integrità

Pertanto è chiaro che la fiducia nel software safety-critical dovrà basarsi sulla fiducia nei metodi di ingegneria del software utilizzati, sulla fiducia nel personale, sulla fiducia nella gestione e, soprattutto, sulle assicurazioni fornite dall'uso di metodi formali oltre che dal testing.

Anche per la norma EN 50128, i livelli di integrità software sono il concetto chiave. Più è alto il livello di integrità e meno probabile è che un errore di specifica o di progetto del software provochi un fallimento pericoloso. La EN 50128 alloca tecniche di sviluppo ed analisi per 5 livelli di integrità software (da 0 a 4). Il livello 0 (che è assente nel progetto IEC 1508, forse la maggiore differenza fra le due normative) indica software senza implicazioni di safety, mentre i livelli da uno a quattro si riferiscono al software safety-critical.

Una particolarità di questo progetto è che le tecniche richieste per i livelli 1 e 2 sono le stesse, così come quelle per i livelli di 3 e 4**. Questa è certamente una caratteristica negativa, in quanto, così facendo, i livelli di integrità per il software connesso alla safety da quattro si riducono di fatto a due. Un'altra caratteristica negativa è che non si dà una guida per assegnare un livello di integrità software ad ogni rischio.** Per questo secondo aspetto EN 50128 non si discosta da IEC 1508; quest'ultima giustifica tale scelta col fatto di non essere finalizzata a un particolare settore industriale e, pertanto rimanda agli standard specifici il compito di fornire una guida per l'assegnazione dei vari livelli di integrità, pur fornendo, in un allegato informativo, un metodo per determinare i livelli di integrità safety utilizzando un diagramma del rischio [10]. EN 50128 invece giustifica la scelta di non fornire una guida all'assegnazione di un appropriato livello di integrità software asserendo che questa dipenderà da molti fattori quali: la natura dell'applicazione, fattori sociali, economici ecc. (**una giustificazione che appare molto debole in uno standard che dovrebbe essere già destinato ad un'applicazione molto specifica**) e, poi, si limita a imporre che il livello di integrità software, deciso sulla base del livello di rischio associato con l'uso del software nel sistema, sia specificato in base a EN 50126.

Per ognuno dei livelli di integrità safety/software, EN 50126, a sua volta, impone il *tasso di fallimento obiettivo*, cioè il massimo numero di volte che un sistema/software progettato e costruito per un certo livello di integrità, può fallire in un'ora. Nel fare ciò la normativa distingue due tipi di sistemi/software:

- quelli di *controllo*, che operano in modo continuo, per i quali, ad esempio, 10^{-9} < tasso di fallimento orario < 10^{-7} al SIL (Safety Integrity Level) 4;

- quelli di *protezione*, che operano "su domanda", (ad esempio i cosiddetti shutdown systems) per i quali si vogliono porre ben precisi limiti alla probabilità di fallimento - fra 10^{-5} e 10^{-4} per SIL 4 - nei casi in cui devono intervenire per mitigare gli effetti dei guasti, di solito (ma non sempre) bloccando il sistema o comunque portandolo in uno stato safe.

4.2 Elicitazione dei requisiti

4.2.1 Maieutica dei requisiti

Oggi diversi standard e progetti di standard per il software safety-critical, come ad esempio IEC 604-1-4 e il progetto IEC 61508 (che andrà a sostituire IEC 1508) richiedono una valutazione del rischio come parte del processo di convalida, oppure impongono esplicitamente che sia svolta un'analisi degli azzardi come base per i requisiti del software, come nel caso della norma EN 50128. Una buona analisi dei rischi è la base migliore per la specifica dei sistemi safety critical, però, purtroppo, allo stato dell'arte è ancora molto lontana da una copertura - intesa come percentuale di azzardi individuati - totale dei possibili azzardi e delle loro conseguenze. Il successo stesso dei metodi di progettazione e verifica di software safety critical dipende dalla qualità della analisi dei rischi che si è in grado di svolgere [120].

Un aspetto di primaria importanza, di cui tenere conto durante tutta la fase di maieutica dei requisiti, è che **la safety di sistema ha pesanti implicazioni per la safety del software, in quanto proprio l'analisi della safety del sistema, controllato dal software, identifica i modi di fallimento critici del software, cioè gli output che possono portare ad un incidente da soli o in concomitanza con altri eventi.**

Leveson sostiene che *l'analisi degli azzardi è l'elemento centrale nel processo di ricerca della safety* [88]; infatti essa coordina e valuta gli sforzi nelle diverse fasi del progetto, grazie al lavoro di identificazione e valutazione degli azzardi, dei sottosistemi safety-critical e dei difetti, errori o guasti critici per la safety.

A partire dagli anni 60 sono nati e si sono sviluppati moltissimi metodi per svolgere l'analisi degli azzardi basati su logiche e strategie diverse, ma con un unico denominatore comune: l'incompletezza dell'identificazione degli azzardi e la grande difficoltà nell'analizzare sistemi grandi e complessi. Il buon successo del metodo della macchina a stati usata nel progetto TCAS II della Federal Aviation Administration (FAA) e quello dello stesso metodo per modellare i vari scenari di SACEM [51] e [88] ci fanno capire che *possiamo applicare alcuni metodi formali alla analisi degli azzardi, ma non tutti*. Poiché il fine principale della analisi degli azzardi è mettere in comunicazione le conoscenze di un piccolo numero (tra 4 e 8) di esperti di varie discipline e spingerli a discutere per conoscere sempre meglio il sistema ed i suoi rischi**, bisogna usare un formalismo che sia accessibile a tutti gli esperti in gioco senza uno sforzo eccessivo, e permetta inoltre agli analisti di collaborare superando le barriere culturali che li dividono.**

Oltre agli automi a stati finiti, che hanno già dimostrato la loro efficacia sia pure con qualche problema legato all'esplosione degli stati, un altro buon candidato all'analisi degli azzardi potrebbe essere la logica temporale, che ha già dato buona prova di sé nell'analisi degli incidenti [78]; essa può essere usata prima di aver costruito il sistema (ed è assodato che prima s'identificano gli azzardi e meno difficile ed oneroso sarà rimuoverli o approntare delle misure difensive). La logica temporale può essere usata sia con logica deduttiva e strategia top-down - come è stato fatto per analizzare alcuni incidenti e *l'analisi di un incidente è qualcosa di molto simile, per non dire uguale, a un'analisi top-down e backward di un azzardo* **-** sia bottom-up; inoltre essa può essere usata in tutti casi in cui si usa la Fault Tree Analysis (FTA) con una marcia in più, infatti *può considerare anche gli eventi in cui la sequenza è importante e soprattutto può tenere conto del requisito temporale*, cosa che la FTA tenta di fare con l'operatore *AND prioritario* nella versione elaborata da NUREG nel 1981, ma con scarsi risultati poichè tale versione non è usata e Villemeur afferma senza mezze misure che l'implementazione della Fault Tree Analysis è difficile se non del tutto impossibile quando studiamo sistemi elementari che interagiscono tra loro ed hanno un'elevata dipendenza dal tempo [122]. Con un'analisi svolta con i metodi formali (ad esempio logica temporale, logica del primo ordine o meglio ancora TRIO, che coniuga i vantaggi di entrambe le tecniche precedenti) **si può dimostrare formalmente la realizzabilità o l'irrealizzabilità di un azzardo e prendere gli opportuni provvedimenti.**

Come sostenuto da molti autori [40, 88], è più importante identificare il maggior numero possibile di azzardi che dare a essi delle probabilità di occorrenza senza o con poco significato, basti ricordare che **il 35 per cento dei guasti in volo nei lanci della NASA è stato causato da azzardi individuati, ma giudicati non credibili**. Perciò utilizzare un metodo di analisi in più migliorerebbe certamente i risultati, con poca spesa - ad esempio indirizzando a questo scopo una parte delle risorse destinate all'analisi quantitativa probabilistica di criticità - si potrebbero ottenere maggiori garanzie di copertura (p.e. FMEA non individua i fallimenti dovuti a guasti di più componenti contemporaneamente e FTA ha molte zone d'ombra, illustrate nella relativa scheda nel par. 4.2.2). Il formalismo dei modelli basati sulla logica temporale è facilmente comprensibile da ingegneri che lavorano in campo industriale (ad esempio ACTL è stata usata con successo all'Ansaldo), gli ingegneri elettronici conoscono l'algebra booleana come fondamento della loro disciplina e gli altri hanno comunque familiarità con la simbologia matematica.

Esplorare nuovi metodi è un dovere morale ed una necessità del mondo accademico e industriale, poiché non possiamo continuare a basarci su metodi di analisi degli azzardi che hanno una copertura di appena il 55%, risultato ottenibile solamente combinando tre o quattro metodi; usandone uno solo si riesce al massimo a raggiungere il 35% (cfr. gli studi sulla copertura delle tecniche di analisi degli azzardi contenuti in [128]).

Secondo [10] si può riuscire ad ottenere valori di copertura più elevati, anche se non soddisfacenti, ma bisogna considerare che tali stime variano da caso a caso (se ci si riferisce ad un impianto "vecchio", che usa una tecnologia consolidata per un prodotto da molti anni sul mercato, e, soprattutto, se si considerano nel numero degli azzardi identificati quelli giudicati non credibili, per i quali non si sono presi provvedimenti, ma che poi puntualmente si sono verificati si avrà certamente una copertura elevata).

Con le nuove tecnologie avremo sempre nuovi azzardi e costituirà un'eccezione costruire impianti che lavorano con tecnologie consolidate.

L'obiezione che metodi basati sulla logica lavorano meglio su componenti del sistema safety-critical che sull'intero sistema, può essere rivolta a maggior ragione ai metodi oggi in uso, compreso FTA, che, in caso di azzardi generali di sistemi molto grandi come una centrale nucleare, ha fallito miseramente [108] e non riesce a gestire bene sistemi molto complessi se non a prezzo di notevoli semplificazioni, ma forse sarebbe meglio dire di "potature arbitrarie" di alcuni (spesso molti) dei suoi rami. A questo proposito basta vedere esempi didattici relativamente semplici in cui per evitare alberi troppo ramificati e con foglie troppo lontane si effettuano delle potature arbitrarie. Comunque c'è anche chi sostiene che la stima probabilistica degli azzardi sia parte integrante e inscindibile dell'analisi degli azzardi; Taylor, in [120], afferma che un processo completo per ricavare le specifiche di safety è composto dalla seguenti fasi:

1. Identificare i potenziali azzardi del sistema obiettivo;
2. descrivere i potenziali scenari degli incidenti;
3. determinare la frequenza degli eventi che innescano un dato incidente, e il valore dell'indisponibilità - ossia della probabilità che siano disattivate a causa di un guasto - delle barriere di safety. La frequenza di ogni particolare incidente è il prodotto di questi due fattori; naturalmente questo procedimento va ripetuto per ogni possibile incidente;
4. determinare se la frequenza di ogni incidente è "accettabile" o tollerabile, in accordo con i criteri di safety stabiliti per il progetto;
5. se la frequenza di un incidente è troppo elevata, identificare un modo di ridurre la frequenza dell'evento che lo innesca, migliorando la reliability delle barriere di safety, o aggiungendo nuovi allarmi o nuovi dispositivi;
6. elencare le nuove misure di safety, le condizioni sotto le quali lavoreranno e le azioni, svolte da ogni dispositivo di safety. Questo costituisce la nuova specifica di safety. Si noti che molto spesso ci saranno diversi modi di migliorare la safety, quindi bisognerà effettuare una scelta;
7. dopo che si sono elencate tutte le misure di safety, è necessario revisionare l'elenco per garantirne l'uniformità e la consistenza, e per individuare eventuali conflittualità fra le azioni svolte dai vari dispositivi di safety.

L'impostazione di [120] è in gran parte condivisa anche dalla norma EN 50126 nel capitolo 6, nella parte riguardante i requisiti di safety e il calcolo dei livelli di integrità safety, espressamente richiamata dalla clausola 5.2 della norma EN 50128; tuttavia, alla luce dello studio dei difetti e delle limitazioni delle tecniche di analisi degli azzardi e della scarsa affidabilità delle stime probabilistiche ricavate da esse, illustrate nelle schede e nelle tabelle del paragrafo 4.2.2, si ritiene necessario quanto meno integrare questo approccio.

Così come Johnson usa la logica del primo ordine per risalire alle cause di un incidente [78], è possibile partire da un azzardo conosciuto (che non è altro che una condizione che può portare ad un incidente), risalire alle sue cause ed eliminarle**,** se possibile, o prendere misure di fault tolerance. Allo stesso modo possiamo partire dal guasto di un componente e vedere i suoi effetti sull'intero sistema, con la *possibilità di combinare tale guasto con altri guasti e di tenere conto delle requisito temporale*, cose impensabili con FMEA. Un formalismo che si presta particolarmente bene a questi tipi di analisi è TRIO, poiché "le specifiche TRIO sono utilizzabili per la simulazione del sistema specificato, attraverso la generazione di sequenze di eventi corrispondenti ad una possibile evoluzione del sistema a partire da determinate condizioni iniziali" [98].

Per l'analisi degli azzardi "l'attuale stato dell'arte" richiede di combinare diverse tecniche bottom-up e top-down per cercare di identificarne il maggior possibile e individuare le loro cause. Nella seconda parte di "*linee guida per le analisi e verifiche di sicurezza per l'accettazione del sistema*" [100] si propone di usare (oltre a dati storici) almeno FMECA e FTA, una strategia consolidata visto che è stata usata anche per il modulo lunare, ma ancora perfettibile; si potrebbe ad esempio usare anche il metodo di analisi degli azzardi con la macchina a stati finiti che ha già dato buoni risultati ed è utile per scegliere tra vari progetti visto che lo si può usare in una fase molto anteriore rispetto all'utilizzo di FMECA. Inoltre si dovrebbe procedere a **un'analisi degli azzardi, utilizzando lo stesso linguaggio di specifica formale basato sulla logica (o sulla algebra a seconda della scelta) che poi verrà usato per la specifica dei requisiti**, perché ciò aiuterebbe anche a prendere confidenza col metodo di lavoro che, comunque, come dimostrano le esperienze di successo in questo campo, deve essere imparato (perfettamente) da tutte le persone coinvolte nel progetto, nella sua attuazione, nella sua verifica e nella sua convalida e certificazione [51].

Come verrà spiegato più ampiamente in seguito, nei metodi formali esistono due modi di ragionare: *Model Theoretic*, che ragiona su modelli, e *Proof Theoretic*, che sfrutta il ragionamento assiomatico deduttivo. Essi possono essere combinati al meglio sfruttando *il primo per fare un modello del comportamento di parti del sistema e trovare i possibili azzardi e il secondo, che ha una maggiore maturità industriale, per la specifica dei requisiti e ulteriori prove sugli azzardi, le loro cause e i loro effetti*. **È necessario e inderogabile usare i metodi formali, poiché i sistemi sono sempre più complessi e real-time** quindi non si può continuare a combinare semplicemente metodi come FMEA/FMECA (che non considera fallimenti multipli, va ripetuto per ogni Stato operativo e risente pesantemente dell'esperienza del personale), FTA (che ha problemi con le sequenze di eventi e non considera il requisito temporale) e ETA (che in realtà può essere usato solo per un piccolo insieme di eventi ordinati temporalmente e non può considerare eventi iniziali multipli a causa dell'esplosione degli stati). Per di più **il lavoro fatto per l'analisi degli azzardi di un sistema è solo in minima parte riutilizzabile per un sistema simile, mentre con alcuni linguaggi di specifica formale ad oggetti sarebbe possibile creare un modello generico, ad esempio di un sistema ferroviario, riusabile per l'analisi degli azzardi anche di altri sistemi ferroviari.** Un approccio di questo tipo è stato utilizzato da Botting e Johnson, impiegando il linguaggio Z+ per l'analisi degli incidenti ferroviari [23].

Se è vero che le risorse e il tempo sono limitati, a maggior ragione essi non dovrebbero essere totalmente dedicati ad un singolo metodo, ma, al contrario, dovrebbero essere distribuiti tra più metodi. Infine va notato che MORT e altri metodi di analisi degli azzardi sono nati come metodi di analisi degli incidenti e *ciò è una ulteriore dimostrazione indiretta della validità delle reti di Petri e della logica temporale in questo campo*. D'altra parte, come i vecchi metodi di analisi dei compiti umani derivati da FMEA vengono oggi abbandonati a causa dell'aumento della complessità delle funzioni che oggi svolge l'uomo, così dovremmo iniziare a pensare ad un uso integrato di FMEA e simili, perché, in realtà, i sistemi nella loro interezza sono diventati e diventeranno sempre più complessi.

Comunque ogni sistema va considerato molto attentamente, poiché anche nella società moderna, specie in Italia, sopravvivono grosse realtà industriali con un'organizzazione interna non certo moderna dove sopravvivono e prosperano i mansionari per operatori e addetti alla manutenzione e un'organizzazione con coordinamento verticale gerarchico. In queste realtà un metodo di analisi dell'errore umano come *Action Error Analysis*, del tutto inadeguato per organizzazioni moderne con coordinamento orizzontale per obiettivi (quali ad esempio Crysler o Olivetti), si rileva di enorme efficacia. Se questo può sembrare non essere poi un gran male, visto che si possono usare metodi di analisi degli azzardi dovuti all'operatore ampiamente sperimentati, bisogna, però, anche considerare che la realtà cristallizzata di alcuni grandi enti, in particolare nel settore pubblico e delle partecipazioni statali, provoca enormi problemi. Ad esempio Barrett e altri notano l'influenza negativa di questo immobilismo sugli ingegneri delle ferrovie che hanno avuto seri problemi nello specificare requisiti funzionali e di safety, proprio per la novità che rappresentava per loro questa attività, e, a volte, anche per la loro mancanza di buona volontà [19]. Questo aspetto non è affatto da sottovalutare poiché, come sottolineato in [34], molti incidenti si potrebbero evitare (o almeno mitigare), con una più attenta analisi degli azzardi e con la simulazione del sistema durante gli stadi di specifica e di progetto, quindi *con una specifica formale che permetta simulazione, prototipazione (e riutilizzo) come quella ottenuta con TRIO+ si potrebbero evitare molti incidenti con grandi vantaggi sia dal punto di vista dell'incolumità di utenti ed operatori, sia da quello economico.*

4.2.2 Metodi di analisi degli azzardi attualmente in uso

Questo paragrafo presenta, in modo sintetico, tutte le caratteristiche rilevanti dei metodi di analisi degli azzardi più ampiamente usati al momento attuale, in modo da permettere una loro panoramica rapida e al tempo stesso approfondita.

Le tabelle permettono il rapido raffronto tra le caratteristiche di **14 metodi di analisi degli azzardi**. La costruzione di queste tabelle sembra particolarmente interessante, poiché, in un caso concreto, potrebbe rappresentare **un ausilio per una scelta rapida, ma non superficiale, tra i metodi e le loro possibili combinazioni.** Inoltre le tabelle permettono di rendersi conto subito che nessuna combinazione di metodi non formali è esente da difetti e che siamo ben lontani da una copertura totale dei possibili azzardi. Infatti *molti di essi non sono che delle mere ripetizioni di altri, molti difetti sono comuni a più metodi e comunque non vengono eliminati da una sia pur ampia combinazione***.**

I dati che emergono dalle tabelle sono sconsolanti, infatti le tecniche informali attualmente in uso soffrono dei seguenti difetti:

- il fatto di essere state concepite in tempi, tecnologicamente parlando, lontani le rende inadeguate ai sistemi moderni;
- quasi tutte poco si adattano al software;
- esiste una limitata offerta di strumenti automatici di analisi;
- sono legate alla necessità di operare con sistemi già consolidati di cui siano conosciute operazioni e possibili modi di fallimento;

dipendono dall'esperienza e dalla capacità di giudizio degli analisti.

Ancora una volta emerge la necessità *dell'integrazione di queste metodologie con i metodi formali* che permettono simulazione (e anche prototipazione nei casi dove la sequenza degli eventi e i requisiti temporali giocano un ruolo importante) per l'analisi degli azzardi di progetti anche solo in parte innovativi, quali sono certamente i sistemi di controllo nei settori del trasporto aereo e terrestre e della produzione e distribuzione di energia con hardware digitale e software.

4.2.3 Sintesi delle osservazioni più salienti

La safety del sistema ha pesanti implicazioni per la safety del software che lo controlla, in quanto proprio l'analisi della safety del sistema identifica gli output critici del software.

I metodi (informali) attualmente in uso per l'analisi degli azzardi hanno pesanti limitazioni, che non possono essere superate neppure da un utilizzo combinato di essi, quindi bisogna integrarli.

La copertura, già limitata, delle tecniche informali di analisi degli azzardi è destinata a diminuire ulteriormente a causa delle nuove tecnologie e della realizzazione di sistemi sempre più complessi.

E' possibile applicare alcuni metodi formali (quali ad esempio reti di Petri, logica temporale e TRIO) all'analisi degli azzardi, dimostrando formalmente l'irrealizzabilità o la realizzabilità di un azzardo, o simulando guasti (anche combinati) dei componenti del sistema.

Metodi formali, quali TRIO, possono superare le limitazioni delle tecniche attualmente in uso, potendo trattare anche combinazioni di guasti, sequenze e requisiti temporali.

L'analisi degli azzardi, svolta con metodi come TRIO+, è riutilizzabile per sistemi simili, realizzando, ad esempio, modelli generici di sistemi ferroviari.

Per progetti, anche solo in parte innovativi, è necessario integrare le tecniche attualmente in uso con metodi formali che consentano simulazione e prototipazione.

4.3 Specifica dei requisiti software

4.4 4.3.1 Posizione della norma EN 50128 sulla specifica dei requisiti software

La norma EN 50128 tratta la specifica dei requisiti software nel capitolo 8. L'obiettivo di questo capitolo è descrivere un documento che definisca un insieme completo di requisiti per il software, che rispetti tutti i requisiti di sistema. Tale documento dev'essere al tempo stesso comprensibile per qualsiasi ingegnere del software e completo, cioè comprendente tutti i requisiti senza che vi sia mai il bisogno di ricorrere ad altri documenti per avere una panoramica completa dei requisiti software.
Per raggiungere questi scopi avrà in input i seguenti documenti:

- specifica dei requisiti di sistema;
- specifica dei requisiti di safety di sistema;
- descrizione dell'architettura di sistema;
- piano di assicurazione della qualità software.

A sua volta l'attività svolta in questa fase produrrà in output i seguenti documenti:

- specifica dei requisiti software;
- specifica dei test dei requisiti software.

La specifica dei requisiti software esprimerà le proprietà richieste per il software che viene sviluppato, non le procedure per sviluppare tali proprietà. Tra queste proprietà vi dovranno essere: funzionalità, reliability, manutenibilità, disponibilità e safety.

TABELLA 1 - Riassunto della descrizione dei metodi di analisi degli azzardi

Caratteristiche Metodi	Input	output	Strumenti di analisi predefiniti	Strumenti
FMEA (Failure Modes and Effects Analysis)	Dati statistici, storici e studi di reliability	Scheda cartacea con modi di fallimento del componente, sue cause e suoi effetti	Schede standard da compilare per i vari settori	Strumenti per la compilazione interattiva delle schede
FMECA (Failure Modes, Effects and Criticality Analysis)	Stessi dati di FMEA	Scheda con 2 colonne in più rispetto a FMEA: una per la probabilità l'altra per la criticità dell'azzardo	Schede standard da compilare per i vari settori	Strumenti per la compilazione interattiva delle schede
HAZOP (Hazard and Operability studies)	Informazioni sulle operazioni di base del sistema per poterne poi valutare le possibili deviazioni	Tabella con le scoperte e le raccomandazioni del team	Schede con parole chiave che guidano la ricerca delle deviazioni delle operazioni	Assistito al calcolatore
FTA (Fault Tree Analysis)	Azzardi forniti da altri metodi	Albero la cui radice è l'azzardo studiato e le cui foglie sono le sue cause	-	Vari strumenti software, ma con molte limitazioni
ETA (Event Tree Analysis)	Eventi che possono avere come esito un azzardo	Albero la cui radice è l'evento studiato e le cui foglie sono tutti i suoi possibili esiti	nessuno	Vari programmi per valutare la probabilità delle conseguenze
CCTA (Cause Consequence Tree Analysis)	Evento critico tratto da dati statistici o rapporti di incidenti	Albero le cui foglie sono tutti i possibili esiti dell'evento	nessuno	Può essere semiautomatizzato vi sono vari programmi
SMHA (State Machine Hazard Analysis)	Basta anche solo un'idea del sistema che voglio realizzare	Esplorazione di vari scenari tramite modello del sistema	-	RSML (Reqirements State Machine Language), simulatori
Checklists	Standard e codici di buona pratica ingegneristica	Liste di azzardi e di caratteristiche di progetto, nonché liste di domande cui si deve rispondere per verificare la conformità del progetto alla buona pratica	Liste di verifica predefinite per tutti i campi che forniscono l'elenco degli azzardi già identificati, informazioni su essi e condizioni di alto rischio	-
GFCM (Gathered Fault Combination Method)	Dati forniti da FMEA, in particolare le 2 colonne della scheda di FMEA riguardanti i modi di fallimento e i loro effetti	Effetti sul sistema delle combinazioni di guasti o difetti	-	-

TABELLA 1 - (continua)

Caratteristiche Metodi	Input	output	Strumenti di analisi predefiniti	Strumenti
MORT (Management Oversight and Risk Tree Analysis)	Azzardi forniti da altri metodi, nonché, nel nucleare, una checklist di 1500 eventi di base	Albero come FTA con analisi del comportamento umano, fattori ambientali e funzioni gestionali	Checklists predefinite	-
AEA (Action Error Analysis)	Dati di riferimento (stime di esperti, osservazioni di sistemi simili già esistenti), dati sulle cause di errore umano	Scheda cartacea con colonne per i modi di errore nell'attività in esame, le sue cause, i suoi effetti sul sistema, i possibili rimedi	Schede cartacee da compilare	-
Indici di azzardo	Descrizione accurata dell'impianto, diviso in unità, che sono segmenti del processo, divise le une dalle altre da entità fisiche (muri, distanze ecc.), o dall'esistenza di particolari azzardi propri di un'area dell'impianto	Valutazione del massimo danno alla proprietà, del massimo numero di giorni di inagibilità, del livello di azzardo per incendio o esplosione, e oggi anche il livello di azzardo per esposizio- ne a sostanze tossiche	Tabelle con il "fattore materiale" un numero da 1 a 40 calcolato sulla base dell'infiammabilità e reattività di un materiale, cui si applicano delle penalità a seconda delle quantità coinvolte, il tipo di processo i materiali da costruzione del luogo dove si svolge	-
PTA (Procedure or Task Analysis	Descrizione dei compiti, esposizione agli azzardi, criticità di ogni compito e passo di ogni procedura, caratteristiche dell'impianto e requisiti sul fisico e la psiche dell'operatore	Scheda cartacea: sulle colonne compito, pericolo, effetti, cause, misure correttive, in alcuni casi raccomandazioni per cambiamenti o miglioramenti delle procedure, particolare preparazione ed equipaggiamento (ad es. abiti protettivi)	Schede cartacee da compilare	-
WSA (Work Safety Analysis)	Descrizione accurata dei compiti da dividere in passi ele- mentari, di cui esami- nare le deviazioni	Tabella con le scoperte del team, cioè le conseguenze di deviazioni nei passi del lavoro	Scheda con parole guida (le stesse di Hazop, ma applicate ai passi del lavoro) che guidano la ricerca delle deviazioni	-

TABELLA 2 - Confronto tra i metodi di analisi degli azzardi

Caratteristiche **Metodi**	**Principali pregi**	**Principali difetti**	**Team**	**Dipendenza esperienza analisti (1)**
FMEA (Failure Modes and Effects Analysis)	Favorisce il dialogo e la conoscenza del sistema; facilità; ben conosciuto in tutto il mondo	Onerosità; non rileva le combinazioni di guasti dei componenti; che causano un fallimento di sistema	4 – 8 ingegneri esperti	4
FMECA (Failure Modes, Effects and Criticality Analysis)	Stessi di FMEA, in più permette analisi quantitative della probabilità degli azzardi	Stessi di FMEA, le sue stime risentono della precisione dei dati in input	4 – 8 ingegneri esperti	4
HAZOP (Hazard and Operability studies)	Possibilità di essere applicato anche al software, efficace nel dimostrare effetti di cambiamenti dei parametri	Noiosità, onerosità, forte legame dei suoi risultati con l'esperienza degli analisti	4 – 8 ingegneri con competenze assortite e leader esperto nell'applicazione dei metodi di analisi degli azzardi	4
FTA (Fault Tree Analysis)	Elevata leggibilità, utile nel caso di eventi iniziali multipli; trova relazioni causa-effetto	Difficoltà con le sequenze di eventi; non prende in consi- derazione il tempo	Anche un solo analista, ma molto esperto	4
ETA (Event Tree Analysis)	Molto utile nel caso di stati finali multipli; possibilità di analisi probabilistica	Non può considerare eventi iniziali multipli	Variabile nel numero dei componenti, ma il successo del metodo dipende molto dall'esperienza	4
CCTA (Cause Consequence Tree Analysis)	Permette analisi qualitative e quantitative	Non considera le influenze tra vari eventi iniziali	Numero e competenze variabili, ma personale esperto	4
SMHA (State Machine Hazard Analysis)	È' un modello formale, non dipende dai modelli mentali degli analisti; semplicità	Se il sistema non corrisponde al modello sorgono dei problemi; necessità di opportuni accorgimenti per far fronte all'esplosione degli stati in sistemi grandi e complessi	Nelle poche esperienze circa 10 persone con esperienze e retaggi culturali eterogenei	2
Checklists	Inducono a pensare, permettono di fare tesoro delle lezioni del passato, sono facili da usare	Inducono false fiducie, provocano errori se usate per prodotti "vecchi", ma con nuova tecnologia	Variabile nel numero, ma composto da persone molto esperte e competenti	4
GFCM (Gathered Fault Combination Method)	Può identificare gli azzardi derivanti dall'interazione dei componenti del sistema; molto efficace	Richiede tempi di analisi notevoli, permette solo analisi qualitative	Variabile nel numero, ma composto da persone esperte nei meccanismi del metodo	4

TABELLA 2 - (continua)

Caratteristiche Metodi	Principali pregi	Principali difetti	Team	Dipendenza esperienza analisti
MORT (Management Oversight and Risk Tree Analysis)	Considera fattori legati all'organizzazione, e al comportamento umano	Complessità, difetti dei metodi basati sulle checklists più difetti della FTA, difficile reperire esperti	Variabile nel numero, ma con leader molto esperto di questo metodo	4
AEA (Action Error Analysis)	Facilità d'uso; identificazione delle attività umane più critiche	Onerosità; non applicabilità a errori contemporanei di più operatori	4 – 8 ingegneri con competenze ed esperienze nella direzione dei lavori e nella manutenzione	4
Indici di azzardo	Semplicità; fornisce un'indicazione quantitativa per azzardi relativi ai materiali processati	Copre solo i materiali e le strutture con cui vengono in contatto; si basa su fattori empirici	Analista esperto	4
PTA (Procedure or Task Analysis)	Facilità d'uso; analisi particolareggiata dei compiti umani nel sistema	Onerosità; non applicabilità ad errori umani contemporanei in più compiti	4 – 8 ingegneri esperti	4
WSA (Work Safety Analysis)	Trova le conseguenze di deviazioni dalle normali operazioni in ogni passo del lavoro	Non individua gli azzardi correlati alla gestione e quelli non connessi direttamente al compito dell'operatore	4 – 8 ingegneri con un leader molto esperto del metodo	4

(1) Si è utilizzata una scala da 0 a 4 per quantificare il livello di dipendenza del successo del metodo dall'esperienza degli analisti nell'analisi degli azzardi e nel particolare tipo di sistema in esame:

0 = assoluta indipendenza;
1 = dipendenza lieve;
2 = "" media;
3 = "" elevata;
4 = L'esperienza degli analisti è una condizione necessaria per il corretto utilizzo e per il successo del metodo

TABELLA 3 - Applicabilità dei metodi di analisi degli azzardi

Caratteristiche **Metodi**	**Fase ideale di utilizzo**	**Metodi con cui si integra**	**Logica di ricerca**	**Strategia di analisi**
FMEA (Failure Modes and Effects Analysis)	Progetto avanzato	GFCM, FTA	Forward	Bottom-up
FMECA (Failure Modes, Effects and Criticality Analysis)	Progetto avanzato	GFCM, FTA	Forward	Bottom-up
HAZOP (Hazard and Operability studies)	Traccia di progetto, poi di nuovo a progetto completo	FTA	Prima backward poi forward	Bottom-up
FTA (Fault Tree Analysis)	Progetto completo	HAZOP, FMEA, FMECA, ETA	Backward	Top-down
ETA (Event Tree Analysis)	Progetto completo o addirittura impianto esistente per valutarne i rischi d'esercizio	FTA	Forward	Bottom-up
CCTA (Cause Consequence Tree Analysis)	Progetto completo	Tutti eccetto FTA e ETA	Sia backward sia forward	Top-down
SMHA (State Machine Hazard Analysis)	Concezione		Bacward/forward	Top-down/bottom-up
Checklists	Tutte le fasi		Libera	Libera
GFCM (Gathered Fault Combination Method)	Progetto avanzato, comunque dopo FMEA o FMECA	Complementare a FMEA	forward	Bottom-up
MORT (Management Oversight and Risk Tree Analysis)	Progetto e pianificazione completi	FMEA/FMECA, ETA	Backward	Top-down
AEA (Action Error Analysis)	Progetto completo con struttura organizzativa definita	Tutti	Forward	Bottom-up
Indici di azzardo	Fase preliminare di identificazione degli azzardi	Tutti		Top-down
PTA (Procedure or Task Analysis	Progetto completo	AEA più tutti i metodi di analisi degli azzardi non imputabili agli operatori	Forward	Bottom-up
WSA (Work Safety Analysis)	Progetto completo	AEA più tutti i metodi di analisi degli azzardi non imputabili agli operatori	Prima backward poi forward	Bottom-up

TABELLA 4 - Collocazione ambientale dei metodi per l'analisi degli azzardi

Caratteristiche Metodi	Periodo di nascita	n. di standard in cui è presente	standard	Campo di nascita	Campi di utilizzo
FMEA (Failure Modes and Effects Analysis)	1960	>5	IEC 812, 1985	Industria aeronautica	Quasi tutti dal- l'aerospaziale al- l'automobilistico
FMECA (Failure Modes, Effects and Criticality Analysis)	Anni 60	>5	MIL-STD-1629A	Industria aeronautica	Quasi tutti dal- l'aerospaziale al- l'automobilistico
HAZOP (Hazard and Operability studies)	Anni 60	>5	CIA, 1987 Mod 1995	Industria chimica	Oggi sostanzialmente tutti
FTA (Fault Tree Analysis)	1961	>5	IEC 1025, 1990	missilistico	Sostanzialmente tutti
ETA (Event Tree Analysis)	1970	3	-	Centrali nucleari	Prevalentemente nucleare
CCTA (Cause Consequence Tree Analysis)	1970	-	-	Analisi incidenti	Vari
SMHA (State Machine Hazard Analysis)	1991	-	-	software	Software aeronautico ferroviario
Checklists	-	>5	-	-	Tutti
GFCM (Gathered Fault Combination Method)		-	-	-	Sostanzialmente tutti
MORT (Management Oversight and Risk Tree Analysis)	-	-	-	Analisi incidenti in centrali nucleari	Centrali nucleari
AEA (Action Error Analysis)	Anni 60	-	-	Industria aeronautica	Analisi dei compiti operativi
Indici di azzardo	1964	-	-	Industria chimica	Impianti che usano materiali pericolosi
PTA (Procedure or Task Analysis	Anni 60	-	-	industria	Industrie con scarsa automazione
WSA (Work Safety Analysis)	Anni 80	-	-	Industria chimica	Industrie con tecnologia stabile

Legenda:
di standard in cui è presente = numero di standard nazionali e internazionali in cui si è rilevata la presenza del metodo (>5 significa che si è rilevata la presenza del metodo in più di 5 standard fra nazionali e internazionali e si ritiene molto probabile la presenza anche in ulteriori standard non esaminati)

Secondo la clausola 8.4.2, la specifica dei requisiti software dovrà essere:

i) chiara precisa, inequivocabile, verificabile, testabile, manutenibile e realizzabile;

ii) riconducibile a tutti i documenti in input.

Dalla descrizione delle proprietà della specifica dei requisiti sembrerebbe ovvio che essa debba essere compilata usando un linguaggio di specifica formale, anche alla luce di quanto affermato dai paragrafi 8.4.13 e 8.4.15.

Il paragrafo 8.4.13 impone che "una specifica di test dei requisiti software sia sviluppata dalla specifica di requisiti software e che questa specifica di test sia usata per la verifica dei requisiti funzionali, come descritti nella specifica dei requisiti software, e anche come una descrizione dei test che devono essere svolti sul software completo".

Il paragrafo 8.4.15 dice che "La riconducibilità ai requisiti sarà un'importante considerazione nella convalida di un sistema correlato alla safety e saranno forniti dei mezzi per permettere che questa (corrispondenza) sia dimostrata attraverso tutte le fasi del ciclo di vita". Inoltre, anche nel seguito della normativa, si pone grande enfasi sulla necessità di assicurare corrispondenza tra i requisiti ed il progetto e tra il progetto e l'implementazione, ponendo grande attenzione a non introdurre parti non corrispondenti e non riconducibili ai requisiti.

Da questa descrizione sembrerebbe derivare la più che naturale conseguenza che non solo la specifica dei requisiti debba essere svolta con l'uso dei metodi formali, ma anche tutto il progetto e l'implementazione, visto che un procedimento di sviluppo per raffinamenti successivi sembra il modo migliore (se non l'unico) per assicurare una corrispondenza ed una riconducibilità (dimostrabili) tra la specifica e le fasi successive del ciclo di vita dello sviluppo del software.

Purtroppo, ciò che per il mondo accademico è naturale, non lo è altrettanto per la norma europea, che, nella tabella riguardante la specifica dei requisiti contenuta nell'appendice normativa, impone che "la specifica dei requisiti software richiederà sempre una descrizione del problema in linguaggio naturale e qualsiasi notazione matematica necessaria che rifletta l'applicazione". Inoltre saranno scelte/(a) una o più delle seguenti tecniche per definire chiaramente e precisamente la specifica:

- metodi formali;
- metodi semiformali;
- metodologia strutturata includendo per esempio JSD, Mascot, SADT, SDL, SSADM, e Yourdon.

Il fatto che i metodi formali siano solo un'alternativa per la specifica dei requisiti da affiancare, eventualmente, a un documento scritto in un linguaggio naturale è preoccupante per non dire allarmante; infatti è universalmente riconosciuto che i linguaggi naturali sono ambigui per loro stessa natura, quindi possono generare dei fraintendimenti da parte degli sviluppatori che, come vedremo, sono le cause dei maggiori incidenti.

Inoltre, senza l'uso dei metodi formali, non si vede come si possano raggiungere, con un buon margine di sicurezza, gli obiettivi di chiarezza, inequivocabilità, verificabilità, testabilità e riconducibilità ai requisiti di sistema e ai requisiti di safety. Infatti queste sono qualità proprie dei metodi formali, del tutto estranee ai linguaggi naturali e in buona parte estranee alle altre tecniche proposte nello standard. Inoltre non si vede *come si possa dimostrare che il progetto e l'implementazione siano riconducibili ai requisiti software e questi ai requisiti di sistema senza l'uso dei metodi formali.* Con l'uso di altre tecniche queste corrispondenze potrebbero essere soltanto intuite, ma non dimostrate affidandosi al giudizio e all'esperienza di esperti, senza alcuna possibilità di dimostrare la safety del sistema né da un punto di vista qualitativo né quantitativo. Inoltre con l'uso di un linguaggio di specifica formale avanzato come TRIO si potrebbe sia effettuare il testing della specifica, grazie alle sue capacità di simulazione e di prototipazione, sia ricavare casi di test dalla specifica da utilizzare per il testing funzionale del software.

Infine è doveroso notare che la classificazione e la ripartizione dei vari metodi di specifica tra metodi formali e semiformali, operate dalle tabelle normative di EN 50128, divergono da quanto comunemente si intende con tali accezioni negli ambienti accademici che li hanno prodotti; classificare ad esempio le reti di Petri temporizzate o i metodi basati sulle macchine a stati finiti come metodi semiformali è a dir poco fuori luogo.

4.3.2 Criticità della specifica dei requisiti software

La specifica dei requisiti software è la specifica al più alto livello di una componente software [10]. Essa dipende dai requisiti di sistema che sono assegnati al software ed ha un ruolo vitale per la software safety, in quanto una sua formulazione precisa e non ambigua è una condizione necessaria per avere la certezza che il software raggiungerà il suo scopo. La specifica dei requisiti software si 'limita' a specificare "che cosa" deve fare il software non "come" lo deve fare, ma influenza tutto il ciclo di vita (così come eventuali errori ivi commessi). **Il mezzo principale per ottenere software affidabile è avere una specifica dei requisiti completa e priva di errori**; infatti il problema principale nell'introdurre i computer nei sistemi di segnalazione ferroviaria è stato proprio la mancanza di consuetudine nello specificare i requisiti funzionali e di safety [19]. *La specifica dei requisiti è certamente l'aspetto più critico del ciclo di vita dello sviluppo del software.* Tale constatazione, ormai universalmente accettata, nasce da numerosi studi sulla distribuzione degli errori nelle varie fasi dello sviluppo del software. Inoltre qualsiasi difetto o errore nella specifica dei requisiti corromperà gli stadi seguenti dello sviluppo del sistema. L'esperienza acquisita nel campo dei sistemi (e del software) safety-critical ha dimostrato che gli errori in questa fase possono provocare e provocano veri e propri disastri [93].

Villemeur nota che ben i due terzi dei difetti del software sono provocati da specifiche errate e solo un terzo da errori di codifica [122] e tali dati sono riportati anche da altri studi svolti in Francia, come ad esempio quelli compiuti da Albin e Ferreol. Anche Bowen e Stavridou [26] confermano che i due terzi dei difetti del software nascono da errori di specifica. In realtà non sono stati soltanto gli studi statistici più recenti a farci scoprire la consistenza numerica degli errori di specifica; infatti Reifer, già nel 1979, aveva stimato che gli errori di specifica rappresentavano il 62,5% dei difetti software [110] e dati altrettanto allarmanti erano già contenuti in [20, 46]. La stima di Reifer nasce dal rapporto fra tutti i cambiamenti occorsi durante il test, che sono stati causati da requisiti omessi, e gli errori di progetto (cioè le codifiche rivelatisi non adeguate alla specifica). Osservando che il testing trova più facilmente errori dovuti a mancata corrispondenza del progetto alla specifica piuttosto che i requisiti tralasciati, le stime contenute in [122] e [26] risultano molto più attendibili, in quanto nascono da studi statistici più solidi.

Comunque già gli studi di Reifer, molto noti in tutto il mondo, poiché segnano la nascita molto attesa della software FMEA (poi rivelatasi non particolarmente efficace), avrebbero dovuto provocare un cambiamento di rotta nella produzione del software (almeno di quello safety critical), con applicazione estesa dei metodi formali; ciò tuttavia non è avvenuto e stenta ad avvenire a tutt'oggi, nonostante la moltitudine di studi che confermano la consistenza numerica e la severità delle conseguenze degli errori di specifica.

Gli errori nella specifica dei requisiti o i fraintendimenti di essa costituiscono la principale causa dei *fallimenti di modo comune*, quei fallimenti cioè che avvengono in diversi moduli o componenti ridondanti e che sono provocati dai(l) medesimi(o) errori(e)/difetti(o)), indipendentemente dal numero di congegni di safety diversi utilizzati e che, come notano Gargantini e altri in [54], sfuggono alla verifica. Pratiche dai nomi suggestivi come *difesa in profondità* e *diversità* sono perfettamente inutili contro questo tipo di errori, e hanno qualche effetto positivo solo sugli errori di codifica. Inoltre gli errori di specifica, e più specificamente quelli dovuti a incompletezza nel comportamento specificato e implementato per il software, sono quelli che hanno maggiori implicazioni per la safety, mentre gli errori di codifica hanno più effetto sulla reliability e su altre qualità del sistema che non sulla safety [88].

Leveson, Jaffe e altri affermano anche essi che oltre il 50% dei fallimenti software è attribuibile direttamente ad errori di specifica [86, 76, 88] e questa (spiacevole) situazione era già ben nota da molto tempo, almeno dal 1975 [20, 46]. Questo dato però probabilmente va corretto al rialzo, perché basato sullo studio degli incidenti; infatti è molto verosimile che tra gli incidenti di cui non si è trovata una causa, come ad esempio le due serie di problemi verificatisi il 23-05-83 e il 09-12-86 che mandarono in tilt la metropolitana di S. Francisco con identiche modalità e identica durata [104], si celi un errore di specifica. Inoltre spesso è molto comodo (e particolarmente ben accetto dagli sviluppatori [118]) attribuire ad un errore dell'operatore la responsabilità di incidenti che magari provocano molti morti, mentre questa è spesso solo una concausa che si è sovrapposta ad una specifica incompleta o fraintesa. Ad esempio, nel caso dei missili Patriot che mancarono il bersaglio durante la guerra del golfo, provocando l'uccisione di oltre venti soldati da parte del missile Scud non intercettato, certamente ci fu un errore degli operatori che non effettuarono le operazioni di risincronizzazione dovute dopo 14 ore di attività operativa, ma una specifica che, in uno scenario di guerra, consentiva solo 14 ore consecutive di finestra operativa e poi richiedeva la necessità di un'attività di resettamento del sistema era certo molto discutibile, e ciò in concomitanza con altri problemi dovuti a progettazione di basso profilo, che si verificavano anche rispettando il vincolo delle 14 ore [104]. L'errore umano, poi, spesso è dovuto più ad errori nella specifica dell'interfaccia uomo macchina che a vere e proprie "colpe" dell'operatore.

Croll e altri, in [34], analizzando le cause di 21 incidenti avvenuti in sistemi safety-critical di recente costruzione, hanno ottenuto risultati sorprendenti e preoccupanti: ben il 40% dei difetti che hanno contribuito a provocare gli incidenti erano errori di specifica dei requisiti e in 13 dei 21 incidenti esaminati le deficienze nella fase di specifica dei requisiti hanno contribuito direttamente (e in maniera preponderante) agli incidenti. Inoltre i difetti software (di codifica e di progetto) che hanno avuto un qualche contributo negli incidenti erano solo l'11%.

Questi dati ci devono far capire che **l'allocazione delle risorse operata nello sviluppo tradizionale del software** per la realizzazione di sistemi complessi, coinvolgenti l'interazione di software hardware ed esseri umani, **deve cambiare completamente**: è infatti assurdo destinare solo il 16% delle risorse alla specifica, quando gli errori commessi in questa fase contribuiscono per oltre il 50% agli incidenti, e assegnare il 27% delle risorse al progetto e alla codifica e più del 50% al testing (che è utile solo contro gli errori di progetto e di implementazione) quando gli errori commessi in queste fasi contribuiscono per appena l'11% agli incidenti. Sarebbe auspicabile una ridistribuzione delle risorse simile a quella operata nello sviluppo basato su TRIO [54], nel quale ben il 34% delle risorse viene destinato alla fase di specifica.

Leveson chiarisce che i maggiori problemi per la safety sono quelli causati da fraintendimenti dei requisiti software [88]. *I linguaggi di specifica formali potrebbero letteralmente eradicare questo tipo di problemi essendo assolutamente non ambigui,* ciononostante sono stati utilizzati in pochissimi progetti. Questa situazione forse è dovuta al fatto che il software safety-critical, come sostiene Neumann [104] non ha un grosso mercato, quindi il mondo industriale non è interessato a sviluppare un'adeguata metodologia di sviluppo. Una prova indiretta di ciò è che non esiste un linguaggio di programmazione con tutte le caratteristiche necessarie per il software safety-critical. Il mercato è certamente più interessato ai sistemi informativi, ma ciò che vale per essi è del tutto inadeguato per i sistemi safety-critical.

Jaffe e altri sostengono che i costi necessari per correggere un errore di specifica sono oltre duecento volte superiori a quelli per correggere un altro tipo di errore. Inoltre questi errori sono molto difficili da scoprire, perché si traducono in fallimenti ad intervalli temporali indeterminati (e ciò li fa scambiare per errori casuali) e non sono rivelati dal testing [76].

Pressoché tutti gli studi concordano sul fatto che gli errori di specifica sono i più pericolosi tra gli errori sistematici e ciò, evidentemente*, è stato recepito anche dagli standard* quando impongono che la specifica dei requisiti software sia scritta in maniera chiara, inequivocabile, verificabile, facilmente riconducibile ai requisiti di sistema e ai requisiti di safety.

Adottare metodi adatti per evitare gli errori di specifica è necessario quindi poiché essi:

1. **costituiscono almeno i due terzi dei difetti del software;**
2. **provocano oltre il 50% dei fallimenti software;**
3. **sono gli errori che hanno le conseguenze più gravi per la safety;**
4. **costituiscono errori sistematici che restano nel sistema finché non vengono**

trovati ed eliminati;

5. **sono molto difficili da rilevare, poiché con essi il testing non serve;**
6. **sono facilmente confondibili con gli errori casuali, in quanto innescati da sequenze di bit imprevedibili, il che li rende ancora più difficili da individuare;**
7. **ammesso di riuscire ad individuarli, sono costosissimi da rimuovere,** (senza considerare che per individuarli di solito è necessario almeno un incidente e a volte ciò non basta come nel caso degli incidenti alla metropolitana di S. Francisco, dove il ripetersi della stessa dinamica nello stesso intervallo di tempo fa apparire più che plausibile la presenza di qualche errore sistematico, che tuttavia non si riesce ad individuare);
8. **non vengono tollerati con tecniche di software fault tolerance.**

4.3.3 Applicazione dei metodi formali alla specifica dei requisiti

In [105] vengono identificati sette attributi o proprietà dell'informazione contenuta nella specifica dei requisiti software:

- *completezza*: inclusione di tutti i requisiti necessari;
- *non ambiguità*: stessa interpretazione da parte di tutti i lettori;
- *consistenza*: assenza di conflitto fra i requisiti;
- *verificabilità*: esistenza di un metodo pratico per verificare che ogni requisito è soddisfatto;
- *modificabilità*: facilità di modifica dei requisiti;
- *riconducibilità*: i requisiti software devono essere riconducibili ai requisiti di sistema, e il progetto software dev'essere riconducibile ai requisiti software;
- *leggibilità*: facile lettura e comprensione di tutti i requisiti da parte dei lettori.

Anche Bologna e altri concordano pienamente con questa posizione, ma si limitano a riportare le caratteristiche che dovrebbe avere la specifica dei requisiti software, senza fornire nessun suggerimento su come fare in modo che essa abbia realmente queste caratteristiche [10].

Le tecniche di analisi della specifica dei requisiti sono costituite sostanzialmente dall'*analisi degli azzardi* (che serve a ricavare i requisiti) e dai *metodi formali* (che servono a specificare e verificare i requisiti), anche se la distinzione non è così netta, in quanto i metodi formali possono essere usati anche per l'analisi degli azzardi.

Il termine metodi formali descrive *l'uso di tecniche matematiche nella specifica, progetto e analisi dell'hardware e del software dei computer*.

Un metodo formale è un metodo di specifica e produzione del software che comprende:

1. una collezione di notazioni matematiche che affrontano le fasi di specifica, progetto e sviluppo;

2. un sistema logico con solidi fondamenti nel quale possono essere formulate prove di verifica formali e altre proprietà;
3. una struttura metodologica dentro la quale si può sviluppare il software dalla specifica all'implementazione in un modo formalmente verificabile.

Metodi formali sono già stati messi a punto da diversi anni eppure i progetti di una certa importanza in cui sono stati applicati sono ancora poco più di una manciata. Inoltre la loro applicazione è stata inferiore a quella prevista dieci anni addietro [121]. In questo campo, come anche in quello dei linguaggi di programmazione, c'è un'evidente frattura fra mondo accademico e gran parte del mondo industriale, il quale si mostra sensibile alla logica del profitto più che ad ogni altra considerazione. Le cause più frequentemente addotte per la mancanza di un uso più esteso dei metodi formali sono:

- inadeguata preparazione degli ingegneri;
- sindrome del "non inventato qui" (quindi non lo uso) molto presente soprattutto in Inghilterra a giudicare dall'ostinazione con cui si continua ad usare Z e a scrivere sempre nuovi libri che presentano sempre nuovi dialetti Z (Z a oggetti, o Z temporizzato) che però hanno in comune con il loro progenitore la maggior parte degli acciacchi, come scarsa comprensibilità e inadeguatezza per la descrizione di sistemi concorrenti;
- maggiore enfasi sul ridurre i costi piuttosto che sull'aumentare la safety.

Secondo [72] i principali motivi per l'assenza di un uso industriale su larga scala dei metodi formali sono:

1. strumenti inadeguati (ma ciò non è vero per tutti i metodi);
2. esempi inadeguati (e questo purtroppo è in gran parte vero perché, come appare chiaro dalle tabelle, gli esempi di utilizzo sono ancora troppo pochi);
3. un certo atteggiamento di attendismo.

L'uso dei metodi formali è ampiamente auspicabile ed è auspicato da studiosi di primo piano. Neumann [104] ad esempio sostiene che i metodi formali (come le specifiche basate su una logica matematica e il ragionamento basato sul theorem proving) hanno il potenziale per affrontare i requisiti, le specifiche, i programmi, e l'hardware, possibilmente includendo selezionate analisi formali di consistenza o correttezza per proprietà e componenti particolarmente importanti. *I metodi formali vengono applicati soprattutto in paesi come il Canada dove il loro uso è obbligatorio*. Comunque in alcuni settori, come ad esempio quello ferroviario, il mondo industriale si sta muovendo verso la loro applicazione, soprattutto dopo i successi della Gec-Alsthom con il metodo B, che sembra abbia fatto risparmiare centinaia di milioni di dollari [51]. Uno dei motivi che rendono particolarmente utili i metodi formali nella specifica dei requisiti software è la loro capacità reale di verificare la corrispondenza di questi con i requisiti di sistema ed i requisiti di safety. **Vi dev'essere un'equivalenza funzionale fra i requisiti di sistema e di safety e i requisiti software** (che non rappresentano altro che quella parte di funzioni, necessarie al sistema e alla safety di esso, assegnate al software). **Se i requisiti di sistema, quelli di safety e quelli del software fossero tutti scritti con un linguaggio formale opportuno sarebbe possibile *provare***

quest'equivalenza, incrementando notevolmente la nostra fiducia nel fatto che il software svolga le funzioni di safety richieste. Certamente vi sarebbe il problema di scegliere adeguatamente il metodo formale da utilizzare in maniera così estensiva nel progetto, ma questo sarebbe un problema di portata decisamente inferiore a quelli che sorgono quando si adotta una specifica informale; infatti, quando il sistema è definito informalmente, il processo di verifica della congruenza fra le specifiche di sistema e quelle software dovrà fare affidamento in modo sostanziale sul giudizio e sulla perizia del personale di progettazione e sviluppo, e ciò appare veramente anacronistico oggi che sappiamo che almeno il 70 per cento degli incidenti sono provocati dall'errore umano a vari livelli [112].

Se poi si continuasse a usare lo stesso linguaggio formale in tutte le varie fasi del progetto si potrebbe facilmente verificare l'equivalenza fra i requisiti di sistema e di safety, quelli software e i vari livelli di progetto (architettura, progetto esecutivo), riuscendo ad eliminare così certamente buona parte dei due terzi degli errori presenti nel software, dovuti a errori di specifica.

Operando così, certo non si raggiungerebbe l'utopia (per adesso) auspicata da [118] di un linguaggio di specifica dei requisiti software e hardware pienamente eseguibile, ma **si raggiungerebbe comunque una buona efficacia e soprattutto si eliminerebbero definitivamente gli errori più pericolosi per la safety, quelli dovuti a fraintendimenti della specifica dei requisiti, con una spesa centinaia di volte inferiore a quella necessaria per eliminarli in una fase successiva.** A ciò si aggiunga che, adottando un metodo di specifica formale come B o TRIO, anche i difetti di specifica, progetto e implementazione che comunque sarebbero stati scoperti nelle fasi di testing, convalida e manutenzione saranno scoperti nelle prime fasi (cioè specifica, progetto e implementazione) del ciclo di sviluppo, come sottolineato anche con l'ausilio di un grafico molto interessante da Hennebert [68]. Inoltre una specifica formale dei requisiti sarebbe auspicabile anche per l'hardware [99] (un esempio di applicazione all'hardware di sistemi digitali è contenuto in [33]).

La scelta del metodo formale giusto (o almeno di uno dei più appropriati) potrebbe non essere così difficile, in quanto esistono molti studi in proposito e, soprattutto, perché oggi si conoscono sempre meglio le caratteristiche dei sistemi safety critical.

Ad esempio è noto che non si può usare Z (almeno nella forma sotto standardizzazione ISO) per i sistemi real-time né per quelli concorrenti (come ammesso anche dall'allegato B di EN 50128). I sistemi safety-critical sono quasi tutti (se non tutti) real-time [67] e inoltre, secondo le visioni più moderne si deve rappresentare esplicitamente l'ambiente a fianco del sistema, pertanto la specifica dei requisiti dovrebbe modellare esplicitamente la concorrenza [8]. Da questi dati di fatto nasce l'evidente conseguenza che Z non è assolutamente utilizzabile nella specifica dei moderni sistemi safety-critical, anche se tale linguaggio di specifica è ancora ampiamente usato soprattutto nel mondo anglosassone, forse per un fenomeno di inerzia. Il fatto di continuare ad utilizzare linguaggi di specifica formali come Z e VDM, i cui difetti sono ampiamente conosciuti ([48, 58] che spiega le difficoltà degli studenti dell'università di Nantes (Francia) nell'apprendere Z e le motivazioni per cui esso è stato sostituito con B) va a detrimento di tutti i metodi formali più moderni, più comprensibili e con molte possibilità in più (come poter affrontare real-time e concorrenza). **Oggi vi sono metodi**

formali, nati proprio per il tempo reale, con possibilità di automatizzare molte verifiche e simulazioni grazie all'eseguibilità, con grandi doti di astrazione e modularità, e sono questi che dobbiamo impiegare nella specifica dei sistemi e del software safety-critical.

Abbiamo già visto come le specifiche formali siano nettamente superiori a quelle scritte in linguaggio naturale come mezzo di comunicazione fra gruppi di sviluppo, perché dotate di maggiore precisione e di totale assenza di ambiguità. *Il processo stesso di formulare una specifica formale fa aumentare la safety e la reliability*; infatti questo processo contribuisce alla chiarificazione dei requisiti: la formalità costringe gli specificatori a rendere chiare le ipotesi, le definizioni e gli obblighi per chiarire le inconsistenze, risolvere i conflitti e rimuovere le ambiguità. Una specifica formale facilita la comprensione delle precise conseguenze di qualsiasi cambiamento nelle ipotesi, incluse le ipotesi safety-critical riguardanti l'ambiente. Pertanto un metodo formale, anche quando è usato solo per la specifica, fornisce un contributo notevole per la safety e la reliability, aiutando a capire il sistema ed eliminando alla radice gli errori dovuti ad ambiguità della specifica. Ciò è importantissimo, in quanto molti dei 21 incidenti, studiati nel citato lavoro di Croll e altri in [34], erano dovuti al fatto che gli sviluppatori non avevano capito bene il sistema. In questo senso i responsabili del gruppo che commercializza gli strumenti di supporto al metodo B opportunamente suggeriscono ai neofiti di iniziare a usare il metodo dapprima anche solo per la specifica. Questo approccio mette in moto un progressivo trasferimento di cultura "formale", limita l'impatto sull'organizzazione del team di sviluppo e, al contempo, mantiene il vantaggio di una specifica dei requisiti che è chiara, non ambigua e convalidata con prove. Un tale approccio ha già avuto successo in varie compagnie come GEMPLUS o EDF.

Una facile obiezione alle considerazioni precedenti potrebbe essere che, se la specifica dei requisiti di sistema e di safety è scritta in linguaggio non formale, scrivere la specifica dei requisiti software in linguaggio formale perde gran parte della sua utilità, in quanto sarà difficile verificare la consistenza fra requisiti scritti in linguaggio naturale e requisiti formali. Questo è vero solo in piccola parte, poiché, anche se non è possibile provare formalmente che una specifica formale è conforme alla specifica dei requisiti informale, ci sono dei metodi per valutare e mettere a confronto una specifica formale con i requisiti informali. Williams, in [125], suggerisce varie tecniche per portare a termine questo confronto e valutare la completezza della copertura dei requisiti formali rispetto ai requisiti informali:

1. *analisi statica*: ad esempio analisi sintattica, o uso di analisi di raggiungibilità (p.e. con automi a stati finiti) o uso delle reti di Petri come suggerito da Leveson e Stolzy [127];

2. *prova presuntiva di teorema*: prova di proprietà (come ad esempio requisiti di sistema e consistenza interna) che il sistema dovrebbe possedere in base alla specifica formale e in questo modo rilevare, per esempio, tutte le contraddizioni nella specifica e tutte le condizioni in più che dovrebbero essere aggiunte. Prima bisogna enunciare un teorema riguardo alla specifica e poi tentare di dimostrarlo. Di solito questi teoremi prendono la forma: "*se la specifica fosse corretta, saremmo in grado di dimostrare X a partire da essa*";

3. *analisi dinamica*: esecuzione della specifica in qualche modo, come ad esempio, mediante simulazioni o esecuzione diretta della specifica. L'analisi dinamica dà l'opportunità al cliente (che si suppone abbia fornito i requisiti informali) di convalidare la specifica di sistema (o del software).
4. *revisioni*: la specifica così come il processo che porta ad essa (per esempio gli strumenti utilizzati nella specifica, o i risultati generati dagli altri tre metodi) può essere revisionata. Poiché la revisione è svolta da esseri umani può essere viziata da errori, ciononostante ha dimostrato di essere una tecnica di grande efficacia.

Per quanto riguarda il punto 3 la possibilità di esecuzione della specifica, fornita ad esempio da TRIO, dà la possibilità di vedere la reale corrispondenza tra requisiti informali e requisiti formali ed è utile per rilevare eventuali difetti di "traduzione". Una possibilità simile è contemplata anche da B (dall'inizio del '97) grazie ad un animatore delle specifiche, con il quale è possibile studiare il comportamento del sistema software provocato dai requisiti adottati.

Oggi sono disponibili molti strumenti di supporto ai metodi formali ed il loro numero è in continuo aumento. Ciò da un lato fa cadere una delle poche obiezioni sensate all'adozione dei metodi formali, ma d'altra parte pone anche qualche problema non banale; infatti *tutti gli strumenti di supporto al processo di sviluppo, tutto il software sottostante (come ad esempio compilatori e sistemi operativi) così come il software di supporto (ad esempio databases) dovrebbero essere sviluppati formalmente per assicurare la correttezza teorica*. Questo è un problema abbastanza serio: basti pensare che esiste un solo sistema operativo certificato (da TUV) per un livello di integrità 3 [117].

Un punto veramente dolente è che **in molti paesi, come ad esempio l'Italia, si dà scarsa importanza all'insegnamento dei metodi formali** che invece sono una tecnica importante di sviluppo del software. In Francia la situazione è molto diversa: si sta cercando di introdurre il metodo B nei programmi ufficiali dei corsi di laurea di ingegneria informatica delle università; da parecchi anni gli studenti di ingegneria del primo anno dell'università di Nantes, ad esempio, dedicano tre ore settimanali all'apprendimento di un metodo formale, che dal '90 al '94 era Z poi dal '94 in poi è stato sostituito con il metodo B a causa delle difficoltà di insegnare Z, nonché dei suoi molti difetti come il fatto che è un linguaggio di specifica che non fornisce un processo di generazione del codice e che ha una sintassi difficile da apprendere, per quanto grande sia l'impegno del docente [58].

Il fatto di prendere confidenza con i metodi formali fin dai primi anni dell'università è fondamentale per il diffondersi della loro conoscenza e per evitare pregiudizi su di essi o anche eccessive mitizzazioni delle loro possibilità. L'università è evidentemente il luogo più indicato per l'apprendimento dei metodi formali; infatti l'industria ha accettato di farli apprendere ai suoi tecnici, durante l'orario di lavoro, solo in un numero molto limitato di casi e per progetti di grande entità e di lunga durata. D'altra parte, si ritiene che sia molto *dannoso sottovalutare l'investimento e lo sforzo necessari alla comprensione e all'apprendimento dei metodi formali* e diffondere veri e propri miti in favore di questa tecnica [59], come che la matematica discreta si impara in un giorno o che Z si impara in 14 giorni, al solo scopo di sfatare i miti contrari.

I vantaggi dei metodi formali non si esauriscono alla sola specifica (a parte casi particolari come Z e VDM); formalismi come il metodo B e TRIO, più che semplici linguaggi di specifica formale, costituiscono veri propri metodi di sviluppo del software che assistono il progettista dalla specifica dei requisiti fino alla verifica e convalida, senza tralasciare neppure le fasi di progetto e di implementazione, con un certo numero di strumenti per automatizzare le prove, per effettuare simulazioni e animazioni ecc..

Un aspetto che influisce molto nella diffusione di un metodo formale è anche il suo supporto di marketing; il metodo B ad esempio usufruisce di un'ottima strategia di marketing e di grandi finanziamenti (da parte di RATP, SNCF, INRET ecc.) che gli permettono di diffondere gratuitamente una rivista in lingua francese e in lingua inglese, illustrante i suoi progressi e i suoi successi, viene insegnato nelle università francesi dove sono già presenti strumenti che lo utilizzano e che si stanno diffondendo sempre di più. Esso è stato sperimentato in una molteplicità di campi diversi (dal software ferroviario a quello per le telecomunicazioni) anche se recentemente ha mostrato serie lacune nel gestire le proprietà temporali, specie in sistemi concorrenti, al punto da portare all'affermazione che “l'uso industriale del metodo (B) nei servizi delle telecomunicazioni sembrerebbe richiedere un'estensione del metodo e una modificazione degli strumenti “[94]. Un'altra deficienza del metodo B è l'impossibilità di condividere variabili sia in modo lettura che in modo scrittura fra diverse macchine astratte; se ogni macchina rappresenta un diverso servizio di una società di telecomunicazioni questo è un problema [94].

TRIO, come indica chiaramente il significato della sua sigla, non ha nessun problema con i sistemi in tempo reale, ma un supporto e una strategia di marketing praticamente inesistenti che, gli impediscono un’ampia diffusione.

4.3.4 Classificazione dei metodi formali

In questo paragrafo viene fatta una classificazione e vengono esaminate le principali caratteristiche dei metodi formali più conosciuti e di quelli più promettenti mediante una serie di tabelle che mettono a confronto più caratteristiche.

Nei campi rappresentanti le caratteristiche si è tentato di scegliere le qualità più significative e qualificanti per la comprensione delle capacità di ciascun metodo. Alcune caratteristiche esaminate che, a prima vista, potrebbero sembrare poco significative, come ad esempio il *periodo di nascita*, hanno invece una loro ragione di essere, in quanto si è osservato che formalismi nati nello stesso periodo hanno spesso caratteristiche simili (ad esempio Z e VDM) e che più o meno tra la fine degli anni ’80 e inizio degli anni '90 vi è stato un notevole cambiamento nel modo di concepire i metodi formali ed il loro ruolo nel processo di sviluppo del software. Rispetto al famoso articolo di Floyd [49] “Assegnare significati ai programmi”, che rappresenta il fondamento delle tecniche di prova formale usate almeno fino alla fine degli anni 80, negli ultimi dieci anni è avvenuto un notevole mutamento tecnologico (per non dire una vera e propria rivoluzione copernicana) nel campo dei metodi formali.

Una volta, terminato un programma lo si convalidava dandogli un significato formale che veniva confrontato con la specifica. Oggi i metodi più moderni (come B e TRIO) cercano di derivare un programma da un modello formale della sua specifica, per ottenere un software "esatto per costruzione" (o ottenuto per raffinamenti successivi, se vogliamo usare una terminologia più comune, ma meno suggestiva). Tale radicale cambiamento è sottolineato dal sottotitolo di "The B-Book" di Abrial "Assegnare programmi ai significati".

Un'altra caratteristica esaminata è lo *stato di standardizzazione*: soltanto pochi metodi formali sono già stati standardizzati o sono in corso di standardizzazione (ad esempio LOTOS, Z, SDL e VDM). Il fatto che un metodo formale venga standardizzato da un organo internazionale quale l'ISO ha dei pro e dei contro. Da un lato l'uso di uno standard significa inevitabilmente usare una tecnologia "vecchia" e può porre dei limiti all'evoluzione di un metodo in un campo ancora in continuo fermento; d'altro lato i metodi formali standardizzati assicurano una comune comprensione dei metodi usati e possono anche incoraggiare lo sviluppo degli strumenti [121], che sono una delle chiavi del successo di un formalismo [72].

In [73] si propone una posizione di compromesso: richiedere che qualsiasi estensione/deviazione da uno standard (per un metodo formale), per essere accettata da un organismo di certificazione, sia prima usata in un safety-case.

Il campo *tipo di formalismo* propone una classificazione "stilistica" indipendente dal livello di rigore del metodo formale. Una specifica *descrittiva* dà una definizione dell'applicazione in modo astratto, con la definizione delle proprietà che devono essere soddisfatte. In una specifica *operazionale* si considera l'applicazione come una macchina astratta basata sul concetto di stato; la macchina evolve cambiando il suo stato; la specifica descrive come la macchina evolve.

Secondo la classificazione di [98]: "*si classificano come operazionali i metodi di specifica nei quali il modello del sistema specificato viene fornito in termini di uno spazio di stati ammissibili per il sistema, e di una funzione di transizione che permette di descrivere le modalità e le regole di passaggio del sistema da uno stato a quello (o quelli, nel caso di forme di non determinismo) temporalmente successivo.* Lo spazio degli stati nelle descrizioni operazionali spesso modella e riflette la struttura di una macchina astratta. I metodi di specifica *descrittivi* partono da una descrizione dello spazio degli stati più astratta e svincolata dal riferimento a macchine o dispositivi, ma soprattutto danno una *descrizione degli stati ammissibili per il sistema modellato in modo molto più generale e meno restrittivo, mediante l'impiego di concetti matematici quali equazioni o assiomi, vincoli e proprietà espressi mediante formalismi il tipo algebrico o logico*."

Naturalmente nei due tipi di formalismo esistono pro e contro. Le specifiche operazionali sono più vicine all'implementazione pertanto si prestano meglio alla simulazione e prototipazione. Le specifiche descrittive sono più astratte, favoriscono la dimostrazione delle proprietà, ma di solito sono un supporto meno valido per la simulazione e la prototipazione. Inoltre, un linguaggio di specifica descrittivo viene definito assiomatico quando fa un uso ampiamente prevalente del processo deduttivo, usando pre-condizioni (cioè condizioni che sono vere prima dell'esecuzione di un'operazione), post-condizioni (cioè condizioni che sono vere dopo l'esecuzione di un'operazione) e invarianti (cioè condizioni che devono essere sempre vere): tipici esempi di questo tipo di formalismo sono Z, VDM e B (sia pure con molti distinguo visto che Z e VDM sono due linguaggi di specifica “model based”, mentre B è un metodo di specifica che supporta anche l'animazione, la progettazione e l'implementazione in modo quasi automatico).

I linguaggi di specifica descrittivi algebrici danno la specifica algebrica di tipi di dati astratti (come ad esempio la pila): rappresentanti di questa categoria sono ad esempio OBJ e ACT ONE.

Poichè nelle tabelle s'incontra spesso la notazione “... basato sulla logica del primo ordine” oppure “...basato sulla logica di ordine superiore”, si cercherà nel seguito di dare una spiegazione il più possibile sintetica di questi termini.

Il *calcolo proposizionale* è la forma più semplice di logica formale: permette la combinazione dei simboli matematici di implicazione e di negazione per costruire delle formule logico-matematiche.

La *logica del primo ordine* è un'estensione del calcolo proposizionale, con una potenza espressiva molto maggiore grazie all'uso di “variabili” che possono occorrere in diverse proposizioni e di parecchi simboli logico matematici. Nella logica del primo ordine si costruiscono le formule con la combinazione di variabili, funzioni, predicati, e congiunzioni logiche [67]. Questo formalismo ha il pregio di coniugare potenza espressiva, possibilità di notevole astrazione e semplicità notazionale.

La *logica di ordine superiore* è (come dice il nome stesso) una logica più sofisticata che permette di quantificare anche su funzioni e predicati. Questo tipo di formalismo “paga” la maggiore potenza in termini di maggiore complessità notazionale.

Un'altra espressione ricorrente, a volte, nelle tabelle è *algebra dei processi*; essa è un tipo di formalismo operazionale basato su un piccolo insieme di operatori e alcune regole sintattiche utilizzate per costruire un processo complesso da processi più semplici [67].

Un altro campo delle tabelle che riveste grande importanza è quello riguardante la *tecnica di ragionamento*. Come già accennato, nei metodi formali vi sono due tecniche di ragionamento principali: model-theoretic e proof-theoretic. Col ragionamento *model-theoretic* si ragiona su modelli dei sistemi. Con questo approccio è possibile usare degli algoritmi di verifica automatica per verificare le proprietà di un sistema, ma, spesso, se il sistema è grande e/o complesso vi può essere la cosiddetta esplosione degli stati, cioè lo spazio degli stati diventa talmente grande da essere ingestibile. Un problema analogo si ha quando si tenta di applicare questa tecnica di ragionamento a sistemi in tempo reale: si ha un modello (della specifica) troppo grande per poter essere analizzato, come è stato sottolineato da Heitmeyer e Mandrioli in [67]. Questa tecnica ha certamente una minore scalabilità, un minore livello di astrazione ed una minore maturità industriale rispetto all'altra, in compenso spesso è di più immediata comprensione e richiede meno preparazione.

Con la tecnica di ragionamento *proof-theoretic* si sviluppa una teoria sul sistema, basata su una qualche logica, come ad esempio la logica del primo ordine, o una logica di ordine superiore e si adotta un ragionamento assiomatico deduttivo. Le proprietà del sistema sono rappresentate come formule nella logica [67]. Certamente sviluppare delle dimostrazioni deduttive richiede una preparazione ed uno sforzo ben maggiori che applicare un algoritmo di verifica automatica, però ciò aiuta ad approfondire la conoscenza del sistema e, inoltre, oggi alcuni metodi (ad esempio B) permettono la dimostrazione automatica dell'80% delle prove necessarie e questa percentuale è destinata ad aumentare. Esistono anche dei “dimostratori” automatici come PVS che possono portare a termine automaticamente molte prove (anche se quelle non banali necessitano la guida dell'utente) e possono essere usati in combinazione con altri metodi come ad esempio TRIO.

Nella casella *progetto/i di utilizzo* si riassumono le applicazioni industriali dei metodi formali, che, nonostante le affermazioni ottimistiche di Hall [59], sono stati usati in pochi progetti (poco più di una manciata secondo [118]), come è stato ribadito anche nella tavola rotonda sui metodi formali pubblicata su IEEE Computer nell'aprile 1996, soprattutto se si rapporta il loro numero a quello dei sistemi safety-critical realizzati in questi anni.

Come sottolineato da Thomas [121] e testimoniato dal continuo sviluppo di nuovi strumenti a supporto del metodo B “ finalizzati ad una sua più ampia distribuzione ed uso”, *gli strumenti di supporto ai metodi formali sono essenziali per un loro uso effettivo*. Gli strumenti oggi esistenti spesso sono troppo generici mentre gli utenti avrebbero bisogno probabilmente di *strumenti specifici per area di applicazione*, come dimostrato dal fatto che gli strumenti di B che avevano dato ottimi risultati in campo ferroviario si sono rivelati insufficienti per il campo delle telecomunicazioni, al punto che Mermet e Mery hanno auspicato apertamente una loro modifica [94].

Nel campo degli strumenti di supporto ai metodi formali il mercato è ancora in fase embrionale e i potenziali compratori non possono essere sicuri di ottenere un'adeguata manutenzione (che tra l'altro è molto costosa) e continuità di rifornimento nel lungo termine. A questo proposito, in [73] si ribadisce che il mercato degli strumenti per i metodi formali è troppo piccolo per lasciarlo in balia delle forze di mercato: un sostegno governativo è di fondamentale importanza (come rimarcato anche da Holloway e Butler in [72]). In Francia e negli Stati Uniti c'è una politica governativa di sostegno allo sviluppo e alla ricerca nel campo dei metodi formali e dello sviluppo di strumenti di supporto ad essi. Il Regno Unito fornisce un sostegno (economico) inferiore, ma dà un notevole sostegno “indiretto” grazie all'adozione di standard che obbligano o che raccomandano (in maniera difficilmente eludibile) l'uso dei metodi formali in certi settori.

La politica italiana a sostegno dei metodi formali non è citata né in [73] né in nessun altro documento esaminato, forse perché, in Italia non si è ancora sufficientemente presa in considerazione una normativa in questione.

Gli strumenti sono giudicati inadeguati e quindi un serio impedimento all'adozione industriale dei metodi formali da Holloway e Butler; in [72] essi sostengono che la maggior parte degli strumenti esistenti sono difficili da imparare efficacemente, a causa dell'uso di notazioni specialistiche, tratte dalla logica matematica, nei loro linguaggi di input e contengono al loro interno numerosi errori, al punto che i neofiti dei metodi formali molto spesso non riescono a capire se la loro incapacità di completare una prova è dovuta alla loro mancanza di preparazione, alla falsità di ciò che stanno cercando di provare, o a un errore nello strumento. Questa situazione tuttavia non va troppo generalizzata in quanto ad esempio gli strumenti a supporto di B, TRIO e PVS sono in continua evoluzione e migliorano e miglioreranno “generazione” dopo “generazione”.

Il sostegno governativo non può limitarsi ad un mero stanziamento di fondi per la ricerca e lo sviluppo di strumenti, ma deve dare impulso alla diffusione della cultura degli stessi metodi formali; infatti [72] sostiene che, una volta che siano disponibili strumenti adeguati, c'è bisogno di esperti in questo campo e quindi di corsi di preparazione. A questo riguardo la Francia sta facendo una politica veramente lungimirante: sostiene economicamente la ricerca di sviluppo del metodo B, ne saggia l'efficacia in progetti sempre diversi in modo che anche gli insuccessi (come nelle telecomunicazioni) possano servire a potenziare sempre più il metodo e l'ambiente di sviluppo, ne favorisce la diffusione con l'istituzione di corsi monografici sul metodo B in ben 14 università (dati riferiti al 1997). **I risultati invece della politica italiana sia nel campo dei finanziamenti, sia in quello dell'istituzione di corsi ad hoc nelle università, non tarderanno a manifestarsi.**

Nella casella dedicata alla *scalabilità* (cioè la possibilità del metodo di essere usato per sistemi di diversa dimensione) si esamina una proprietà ben più importante di quanto potrebbe sembrare a prima vista; infatti questa qualità e spesso fondamentale nella scelta di un formalismo o di un altro, ad esempio le reti di Petri, pur essendo molto intuitive, molte volte non possono essere usate a causa della loro mancanza di scalabilità.

I sistemi (software) safety-critical reali hanno spesso dimensioni significative; basti pensare ai casi delle centrali nucleari come ad esempio quella di Ontario, ai sistemi di controllo ferroviario come ad esempio SACEM (18.000 linee di codice), e ai sistemi metropolitani a automatismo integrale come ad esempio Meteor (90.000 linee di codice). Alla luce di queste considerazioni la presenza o l'assenza di scalabilità in un metodo formale giocheranno un ruolo ragguardevole nella decisione di adottarlo o meno, in particolar modo per quanto riguarda la specifica del software ferroviario.

Una caratteristica che si coniuga molto bene con la scalabilità è la *modularità*, cioè la possibilità di comporre la specifica grazie all'uso di componenti (moduli) collegati, ma con un buon grado di indipendenza, corrispondenti ai componenti (logici o fisici) del sistema. La modularità è indispensabile per sistemi grandi e complessi “la cui complessità può essere dominata solo scomponendoli in parti più semplici e con interfacce ben definite” [98].

Una qualità di notevole importanza, anche se molto difficile da valutare oggettivamente, è la *comprensibilità* cioè la facilità di comprensione e di apprendimento anche da parte di non specialisti o comunque di persone senza una preparazione tecnica adeguata nel campo dei metodi formali (come normalmente sono i committenti). La specifica dei requisiti è un'attività che richiede la collaborazione e l'interazione tra esperti di diversi settori, pertanto, come in tutte le attività che richiedono stretta interazione, se alcuni partecipanti non riescono a capire bene il linguaggio usato non si otterrà altro che un risultato incline all'errore [88, 34]. Si sono trovati elementi di valutazione oggettiva per sette dei metodi esaminati, prendendo spunto dai giudizi in proposito espressi dalla letteratura (prediligendo quella non eccessivamente di parte), dalle esperienze empiriche di apprendimento in ambito industriale e dalle esperienze di insegnamento e di apprendimento svolte nelle università.

Dalla ricerca è emerso quanto segue:

- *VDM* è universalmente riconosciuto poco comprensibile e di difficile apprendimento a causa dell'uso di simboli proprietari e di non immediata comprensione [118].
- *Z* si è rivelato di apprendimento particolarmente difficile; durante la già citata esperienza di insegnamento svolta all'università di Nantes gli studenti facevano spesso confusione fra i simboli (ad esempio uguaglianza e inclusione), la notazione grafica usata per le precondizioni non facilitava affatto la comprensione, e infine Z non forniva affatto un processo di generazione del codice. Proprio i problemi di comprensione e di apprendimento di Z hanno portato alla sua sostituzione con B. A onor del vero su Z e VDM esistono anche opinioni diverse come quella di Hall [59] che sostiene che si può imparare perfettamente Z o VDM in soli 14 giorni. Tale opinione può essere tuttavia annoverata tra quelle di coloro che Holloway e Butler (in [72]) definiscono "fanatici dei metodi formali" poiché stride in maniera fin troppo evidente con i risultati osservati da Habrias [58] in quattro anni di insegnamento di Z ai futuri ingegneri. Forzature ottenute usando toni eccessivamente encomiastici e opinioni personali troppo ottimistiche favorisce l'instaurarsi di un clima di diffidenza fra mondo industriale e mondo accademico, che, invece, in questo settore devono collaborare strettamente se vogliono avere successo.
- I responsabili della Gec-Alsthom hanno dichiarato che ai loro tecnici è stato necessario circa un anno per imparare perfettamente il *metodo B*, ma un dato ancora più interessante (e certamente più obiettivo) è quello relativo ai risultati dell'insegnamento del metodo B nelle università francesi. Da uno studio di Habrias [58] è emerso che B era più facile di Z da insegnare, specialmente tra gli studenti che conoscevano già le basi della programmazione, e che "the B-Book" rappresentava un ausilio basilare per l'insegnante, che non aveva nessun equivalente in Z. L'ultima affermazione potrebbe sembrare poco credibile o addirittura paradossale, in quanto vi è una vastissima letteratura su Z e autori molto noti si sono cimentati anche recentemente nello scrivere libri destinati all'apprendimento di questo linguaggio di specifica formale; quindi non resta che accettare l'evidenza: *è Z che ha dei limiti di chiarezza e comprensibilità intrinseci e difficilmente superabili*.
- L'intuitività notazionale è, insieme alla capacità di modellare sistemi (anche non deterministici), la dote probabilmente più apprezzata delle *reti di Petri*. La grande comprensibilità di questo formalismo è già riconosciuta da Leveson e Stolzy [127] ed è confermata dai buoni risultati di apprendimento ottenuti dagli studenti del Politecnico di Milano.
- Gli Statecharts, introdotti in [63], sono un'estensione degli automi finiti che tenta di porre rimedio alle principali manchevolezze di questo formalismo. Essi forniscono costrutti per la rappresentazione della modularizzazione e scomposizione gerarchica, la modellizzazione della concorrenza (anche se esclusivamente di tipo sincrono) e della comunicazione tra le parti in cui il sistema specificato può essere decomposto. I meccanismi di comunicazione tra le parti concorrenti fanno sì che il formalismo non sia banalmente riconducibile a una macchina stati/transizioni. Gli Statecharts contengono anche numerose altre estensioni volte a facilitare la descrizione di

aspetti temporali, o a condizionare lo scatto di certe transizioni a combinazioni proposizionali di altri segnali.

- RSML (Requirements State Machine Language) è una modifica di Statecharts, formalismo basato sulle macchine a stati finiti, la cui semplicità e intuitività è largamente riconosciuta. Inoltre ha dato prova di grande facilità di apprendimento durante il progetto TCAS II, nel quale persone con scarsa preparazione nell'ambito della matematica discreta, come ad esempio i piloti, sono riusciti ad apprendere e a utilizzare RSML, al punto che gran parte dell'attività di specifica è stata svolta senza l'aiuto del team capeggiato dalla Leveson che aveva sviluppato il metodo. Inoltre, secondo Leveson [88], i modelli basati sulle macchine a stati finiti rispecchiano i modelli interni che molte persone usano nel cercare di capire i sistemi complessi. Comunque questo formalismo, se così si può chiamare visto che è un qualcosa di intermedio fra metodi semiformali e metodi formali, ha serie limitazioni [67].

- *TRIO* ha un'ottima comprensibilità, specie se raffrontato alla maggior parte degli altri metodi proof-theoretic. Esso unisce i vantaggi di una notevole comprensibilità ampiamente riconosciuti alla logica dei primo ordine con quelli della logica temporale. Osservazioni empiriche, svolte al politecnico di Milano, hanno mostrato che persone che avevano già maturato esperienze culturali nel campo della logica trovavano TRIO del tutto intuitivo, mentre persone con buona preparazione in altri campi (come ad esempio quella della meccanica), ma senza nessun retroterra culturale nel campo della matematica discreta, dell'informatica e della logica avevano seri problemi a raggiungere un apprendimento professionale del metodo, probabilmente perché si trovavano di fronte a qualcosa di completamente nuovo per loro. Gli studenti del corso di ingegneria del software, che affrontano lo studio di TRIO, conseguono buoni risultati nell'analisi delle specifiche TRIO, cioè raggiungono una buona comprensione del linguaggio, ma hanno risultati più scarsi per quanto riguarda la sintesi, cioè l'elaborazione delle specifiche, probabilmente a causa del poco tempo (solo 6 ore) dedicato a questo linguaggio, tempo che è certamente insufficiente se raffrontato con il tempo che, in Francia, si dedica allo studio del metodo B. Comunque va notato che gli studenti che approfondiscono lo studio di TRIO, nella seconda semestralità del corso di informatica teorica ottengono dei risultati di apprendimento eccellenti.

- *PVS* si basa sulla logica di ordine superiore ed è universalmente riconosciuto ed accettato che i metodi basati su questa logica hanno una comprensibilità inferiore a quella dei metodi basati sulla logica del primo ordine.

- Le specifiche scritte con *OBJ* sono eseguibili, ma notoriamente "pagano" la loro eseguibilità con una dimensione molto più grande rispetto ad altri formalismi e con una maggiore complessità che le rende di scarsa immediatezza.

- Per quanto riguarda *Lotos* e *CCS* non si sono trovati dati oggettivi su di essi e riferire le opinioni personali dei loro "sostenitori" sembra poco corretto (visto che non è stato fatto per gli altri metodi) se non fuorviante in quanto, come si è visto per Z, è stato osservato che i dati oggettivi di [58] erano antitetici alle opinioni personali dei sostenitori di Z come ad esempio [59].

Infine il *livello di astrazione* è un concetto di difficile definizione: un metodo formale ha un livello di astrazione tanto più elevato quanto più la descrizione del sistema che fornisce è generale, non restrittiva svincolata dal contingente, quindi adattabile, priva di vincoli eccessivi, stabiliti in una fase troppo precoce.

4.3.5 Valutazione dei risultati

Un'attenta lettura delle tabelle permette un raffronto immediato tra i vari metodi analizzati, e può anche aiutare a scegliere quello più appropriato per ogni situazione.

La cosa che salta immediatamente agli occhi è la situazione di inferiorità rispetto a molti parametri in cui si trovano due linguaggi di specifica formale molto noti come VDM e Z. Tale situazione per VDM è ormai pacifica; infatti ormai non viene quasi più usato per la specifica dei sistemi reali: una delle sue ultime applicazioni è stata probabilmente la specifica formale della semantica del linguaggio Modula-2, una scelta giudicata molto infelice perché specificare formalmente un linguaggio procedurale, apprezzato da gran parte del mondo accademico per le sue doti di chiarezza ed espressività, con un linguaggio di specifica formale riconosciuto universalmente come poco chiaro e scarsamente comprensibile è quasi paradossale e certamente contribuirà non poco all'immeritato affossamento di Modula-2, il quale, invece, sarebbe potuto essere il linguaggio più appropriato (fra quelli attualmente esistenti) per l'implementazione dei sistemi safety-critical.

L'uso di Z deve essere evitato per la specifica di sistemi grandi e complessi a causa della scarsa comprensibilità e difficoltà di apprendimento, che lo rendono uno strumento inidoneo per aumentare la comprensione del sistema, che è uno degli obiettivi della specifica formale, e non troppo secondario visto che Croll e altri, in [34], ci ricordano quanto siano disastrose le conseguenze di una scarsa o errata comprensione di un sistema da parte degli specificatori. Inoltre Z non è in grado di specificare sistemi concorrenti (una qualità oggi ritenuta indispensabile). *L'impossibilità di supportare il tempo reale, poi, rende Z del tutto inadeguato per i sistemi ferroviari (e la maggior parte dei sistemi safety-critical).* Infine Z è orientato esclusivamente al ragionamento deduttivo e questa è una limitazione che lo rende ancora meno versatile.

Una caratteristica molto interessante (e qualificante) è il *livello di rigore*: soltanto TRIO e PVS forniscono linguaggi di specifica pienamente formali con ambienti di supporto completi, che includono la dimostrazione dei teoremi automatizzata. A onor del vero anche il metodo B si sta avvicinando a questo risultato: è già possibile svolgere in modo automatizzato oltre l'80% delle dimostrazioni e si spera di arrivare presto al 90%, inoltre si stanno sviluppando sempre nuovi strumenti: solo il 97 ha visto la nascita di un animatore di specifica e di un navigatore, da usare, in connessione con Netscape, per 'navigare' tra i vari componenti di un progetto grande e complesso.

TABELLA 1

Caratt. Metodi	Periodo di nascita	Stato di standardiz-zazione	Livello di rigore (1)	Tipo di formalismo	Tecnica di ragionamento	Progetto/i di utilizzo	Strumenti
B-method [2] (2)	Primi anni 90	Non standardiz-zato	2	Descrittivo, assiomatico, basato sulla teoria degli insiemi e la logica del I ord.: il sistema è rappresentato come un insieme di stati e operazioni (con pre e postcond.) che modificano lo stato	Proof-theoretic	SACEM (software per la metropoli-tana di Parigi); Software per la metropolitana di Calcutta; software per le nuove linee ferroviarie francesi ad alta velocità	Atelier-B: type-checker, POG (Proof Obligation Generator),Prover, animatore di specifiche, navigatore tra le specifiche
Z [22], [74], [48]		In via di standardiz-zazione	2	Descrittivo assiomatico basato sulla teoria degli insiemi tipizzata e sulla logica del I ord.	Proof-theoretic	IBM CICS; T 800 transputer; sistema di segnalazione ferroviaria del Regno Unito	CADiZ; ZETA
VDM (Vienna Development Method)	ANNI 60 [125]	In via di standardiz-zazione	2	Descrittivo assiomatico	Proof-theoretic	Interlocking DSB; CDIS	VDM
LOTOS (Language Of Temporal Ordering Specifica-tion) [21]	Primi anni 80	Standard ISO 8807, 1988	2	Il linguaggio ha 2 componenti uno è operazionale basato sull'algebra dei processi, l'altro (ACT-ONE) è un linguaggio di specifica descrittivo (tipi di dati astratti)	Model-theoretic	Molto usato nel campo delle tele-comunicazioni e nel contesto ISO-OSI; software del computer di bordo dell'Airbus 330 e 340	LOLA (LOTOS Laboratory), compilatori, animatori simbolici e verificatori di modello
CCS [95]		Non standar-dizzato	2	Operazionale (Algebra dei processi)	Model-theoretic	Interlocking Ansaldo; controllo distribuito treni Siemens	JACK; PVE; Concurrency Workbench
TRIO (Tempo Reale Implicito) [98], [101]	1989	Non standardiz-zato	3	Descrittivo:fonde le caratteristi-che della logica del I° ord. Con alcuni operatori della logica temporale	Supporta sia il ragionamento proof-theoretic, sia quello model-theoretic	Specifica, convalida e verifica di un sistema real-time per equilibrare il carico dei generatori di una centrale elettrica; misuratore digitale di potenza ed energia; sincronizzatore hardware di sequenze di byte	Editor grafico interattivo; editor di test case; traduttore da TRIO+ a TRIO; generatore di test case; history checker un interprete per la verifica a posteriori in grado di testare sistemi non determini-stici

Caratt. Metodi	Periodo di nascita	Stato di standardiz-zazione	Livello di rigore (1)	Tipo di formalismo	Tecnica di ragiona-mento	Progetto/i di utilizzo	Strumenti
OBJ [60]			2	Descrittivo algebrico: specifica algebrica di tipi di dati astratti	Proof-theoretic	Specifica di programmi ADA; progetto formale di programmi ad oggetti	OBJ
PVS (Prototype Verification System) [124]			3	Descrittivo assiomatico basato su logica di ordine superiore	Proof - theoretic	Verifica di correttezza di un con-trollore real-time di passaggio a livello ferroviario;verifica di elementi di un microprocessore per aerei; ampiamente usato alla NASA	Insieme completo di strumenti di supporto, fra cui un Theorem Prover interattivo automatiz-zato
Reti di Petri [127], [98], [50]	Anni 60		2	Operazionale	Model-theoretic	NORAD; VLSI; sistema elettronico di trasferimento fondi	AeSOP composto da: editor, forward analyzer e backward analyzer
RSML (Require-ments State Machine Language) [69]	199; è un'e-voluzione (o meglio un sottoinsieme) di State-charts		2	Operazio-nale	Model - theoretic	TCAS II; ATP system	Simulatore
StateCharts [63]	1987		2	Operazionale	Model - theoretic	avionica e sistemi militari	editor, simulazione, generazione di codice

(1) Per classificare il livello di rigore si è usata la scala adottata da Rushby in "Formal Methods and the Certification of Critical Systems", technical report CSL-93-7, SRI International, Menlo Park, CA, nella quale:
livello 0 = nessun uso di metodi formali;
livello 1 = uso di concetti e notazioni della matematica discreta;
livello 2 = uso di linguaggi di specifica formali con alcuni strumenti;
livello 3 = uso di linguaggi di specifica pienamente formali con ambienti di supporto completi, che includono la dimostrazione dei teoremi automatizzata.

(2) Si è indicata la/(e) fonte/(i) bibliografica/(che) contenente l'informazione più completa ed esauriente sul metodo, ma le informazioni contenute nei vari campi sono frutto di una rielaborazione di queste fonti e del loro confronto critico con gli altri articoli e testi esaminati, perciò, a volte, vi può essere un dissenso tra le notizie nella tabella e quelle contenute in alcune di queste fonti.

TABELLA 2

Caratt. Metodi	Scalabilità	Modularità	Compren-sibilità	Supporto real-time	Capacità di astrazione	Principali pregi	Principali difetti
B-method [2]	Possibilità di uso anche per sistemi grandi	Elevata: possibilità di riusare le macchine astratte definite per un progetto	Necessità di circa un anno di studio per appropriarsi del metodo	Presente	Notevole	Grande maturità industriale, molti strumenti di supporto	Le dimostrazioni non sono automatiche, spesso il codice implementato è stato rilevato non corrispondere alla specifica, il processo di raffinamento può ingenerare errori di codifica, problemi con sistemi real-time concorrenti per i quali gli strumenti sono inadeguati
Z [22], [74], [48]	Possibilità di costruire sistemi grandi soprattutto nella versione ad oggetti; nella ver-sione stan-dard la massima dimensione che è possibile convalidare è valutata in 65 schemi	Non eccelsa in z standard, elevata nella versione ad oggetti; la scarsa leggibilità di grandi specifiche in Z può in parte vanificare questo risultato	Scarsa, no-nostante la presenza di notazioni grafiche che aumentano la leggibilità di Z, il linguaggio mantiene una complessità tale da renderlo di difficile apprendi-mento	Assente in z standard le applica-zioni real-time enfa-tizzano la scarsa comprensibilità dei derivati Z temporiz-zati (è necessaria la presenza di persone molto preparate ed esperte)	Notevole	Molto conosciuto in tutto il mondo, ha a disposizione diversi tool, dà buoni risultati nella specifica di sistemi sequenziali, le sue dimostrazioni sono riusabili	Oggettiva difficoltà di apprendimento e uso; inutilizzabilità in pratica in caso di applicazioni concorrenti
VDM [125]	Scarsa: nella specifica di sistemi grandi emerge ancora di più la sua astrusità	Quasi inesistente, oggi per di più tende ad essere sostituto da suoi derivati	Notevole complessità: usa diversi simboli proprietari e di non immediata comprensione	Assenza del concetto di tempo; può essere esteso per coprire anche requisiti temporali, ma con scarsi risultati	Notevole	Molti suoi concetti, come quello di raffinamenti successivi sono ancora validi e sono stati presi da altri metodi, dà risultati accettabili con i sistemi sequenziali	Incapacità di affrontare la concorrenza presente nei sistemi di controllo complessi; riconosciuta necessità di molto tempo per imparare VDM,e no-tevole complessità

Caratt. Metodi	Scalabilità	Modularità	Compren-sibilità	Supporta il real-time	Capacità di astrazione	Principali pregi	Principali difetti
LOTOS [21]	Scarsa	Scarsa		LOTOS standard non supporta i requisiti temporali, ma vi sono varie versioni temporiz-zate	Maggiore che in CCS grazie ad ACT-ONE media	Dà ottimi risultati nelle applicazioni concorrenti, è molto conosciuto, supporta vari stili di specifica	I progetti in LOTOS sono a volte troppo grandi per essere processati dai strumenti di verifica formale; a volte i programmi eseguibili generati dal progetto LOTOS si sono rilevati troppo lenti
CCS [95]	Scarsa: risente dell'esplosione degli stati per sistemi grandi, pertanto ha bisogno del supporto di altri formalismi	Scarsa	Scarsa: nettamente inferiore a quella dei metodi basati sulle logiche del I ord.	Come LOTOS tratta bene il tempo relativo, ma non quello reale (ACSR aggiunge il requisito temporale a CCS)	Scarsa: si usa insieme ad altri formalismi come ACTL per aumentarne la capacità di astrazione	Dà buoni risultati con sistemi concorrenti possibilità di dimo-strazioni automati-che, disponibilità di strumenti di verifica automatica	Esplosione degli stati, scarsa astrazione, scarsa maturità industriale
TRIO [98], [101]	Notevole grazie alla versione ad oggetti TRIO+	Notevole, è possibile costruire modelli generici riutilizzabili con pochi adattamenti	Notevole leggibilità: le sue notazioni sono già familiari ad utenti non specialisti, ha i vantaggi della logica del I ord. uniti a quelli della logica temporale	Nasce proprio per trattare si-stemi real-time	Notevole	Elevata leggibilità, eseguibilità delle specifiche, possibilità di uso delle specifiche per la simulazione del sistema	prima della nascita di TRIO+ non era adatto per sistemi grandi e complessi, ma ora questo difetto è stato superato; scarsità di applicazioni industriali
OBJ [60]	Scarsa: le specifiche sono molto grandi anche per sistemi piccoli, per sistemi grandi sono quasi ingestibili	Scarsa	Scarsa a causa della ampiezza delle specifiche, che è tale da ren-dere preferibile in molti casi l'uso di linguaggi model-based	Assente	Notevole	Eseguibilità delle specifiche	Notevole ampiezza delle specifiche, che le rende difficili da ve rificare

Caratt. Metodi	Scalabilità	Modularità	Compren-sibilità	supporta il real-time	Capacità di astrazione	Principali pregi	Principali difetti
PVS [124]	Si può applicare anche a sistemi molto grandi, variano solo i tempi necessari	Notevole, grazie alle "teorie", che permettono modularità e riuso	Nettamente inferiore rispetto ai metodi basati sulla logica del I ord.	Si trova perfetta-mente a suo agio con i sistemi real-time	Notevole: ha molti tipi di dati astratti e la possibilità di costruir-ne altri	Ambiente di supporto completo, possibilità di prove automatiche	Necessita di una workstation e di molto spazio in memoria per le applicazioni; ha la li-mitazione di essere esclusiva-mente orientato alla deduzione
Reti di Petri [127], [98], [50]	Scarsa: le reti di Petri sono l'ideale per sistemi piccoli	Scarsa	Elevata: le notazioni grafiche sono di immediata comprensione anche a persone senza grossa preparazione matematica	Nella forma base il tempo è assente, ma esistono varie versioni di reti di Petri temporizzate	Scarsa	Ottime per la simula-zione e la prototipa-zione di sistemi piccoli	Inapplicabilità a sistemi grandi e complessi a causa dell'esplo-sione degli stati
RSML [69]	Nettamente maggiore rispetto a statecharts	Notevole (d'altra parte era una caratteristica già ben presente in statecharts)	Notevolissima: è il maggior punto di forza del metodo, è addirittura maggiore di quella della logica del I ord.	Per questo aspetto non si discosta da statecharts, è in grado di trattare i sistemi real-time	Maggiore che nelle reti di Petri ed in altri formalismi basati sulle macchine a stati finiti, ma inferiore a quella dei linguaggi descrittivi	Grande semplicità e leggibilità, anche persone senza preparazione matema-tica riescono a parte-cipare alla specifica	Pochi strumenti di supporto (non si può usare neppure STATEMATE); la maggior parte dei difetti di statecharts in realtà si pone a mezzastrada fra i metodi semiformali e quelli pienamente formali
StateCharts [63]	limitata	buona	discreta	in misura limitata	intermedia tra metodi basati su stati/transi-zioni e metodi descrittivi	ricchezza della notazione, modularità	semantica assai complessa, non del tutto formalizzata

TRIO e B nascono quasi contemporaneamente, il primo nasce in ambiente accademico ed ha quasi da subito il supporto di strumenti di ottimo livello, il secondo in ambiente industriale ed all'inizio è praticamente privo di strumenti di supporto, però il suo successo in un progetto industriale gli fa guadagnare immediatamente la fiducia di tutto il mondo produttivo francese e non solo, quindi vengono sviluppati gradualmente molti strumenti e vengono commercializzati con profitto. Questi fatti provano che **la presenza di strumenti adeguati e gli esempi di utilizzo di un metodo formale sono entrambi importanti per conquistare la fiducia dell'industria, ma probabilmente il secondo aspetto è prevalente** soprattutto se associato ad una politica di sostegno adeguata.

TRIO, PVS e B hanno tutti ottime capacità di scalabilità, modularità e astrazione quindi si possono applicare perfettamente anche a sistemi grandi e complessi, però *PVS ha una comprensibilità inferiore che lo rende più adatto alla prova di proprietà (precedentemente specificate anche con altri formalismi) che all'analisi e alla specifica dei requisiti che richiedono un linguaggio che possa essere appreso e compreso da esperti di più settori (anche estranei al software).* Craigen e altri, in [35], notano che sostanzialmente in tutti i più importanti progetti industriali nei quali sono stati usati in metodi formali sono stati usati per la specifica formalismi basati sulla logica del primo ordine: *il fatto che nessun progetto industriale rilevante sia stato specificato con una logica di ordine superiore, forse, significa che il mondo produttivo non è ancora pronto a "digerire" tecniche così complesse.*

B e PVS hanno la limitazione, rispetto a TRIO, di essere orientati esclusivamente alla deduzione, inoltre (almeno per adesso) B ha mostrato dei limiti con i sistemi real-time concorrenti, anche se sembra vi sia la volontà di rimediare ad essi in tempi brevi, problemi che TRIO certamente non ha, visto che è nato per operare con sistemi in tempo reale. **Considerando i sistemi ferroviari, essendo essi sistemi real-time concorrenti (vista la loro interazione con l'ambiente), è molto probabile che TRIO si riveli più adatto alla specifica dei requisiti di tali sistemi e del loro software di quanto non lo sia B. TRIO a differenza di B, può essere usato anche per l'analisi degli azzardi** grazie alla sua capacità di simulare il sistema specificato, "attraverso la generazione di sequenze di eventi corrispondenti ad una possibile evoluzione del sistema a partire da determinate condizioni iniziali" [98]. Ciò è impossibile per B che, nei progetti in cui è stato usato (ad es. SACEM), si è dovuto avvalere di un modello operazionale, fondato sulle macchine a stati finiti, per simulare i vari scenari in corrispondenza ad eventi che potevano creare azzardo e di uno strumento, chiamato Grafcet ed estraneo all'ambiente di sviluppo B, per trasformare la prima specifica, scritta in linguaggio naturale, in una rappresentazione tramite macchine a stati finiti e poi effettuarne la simulazione.

Nell'intento di ovviare a questi inconvenienti, dalla metà del 1997 il metodo B ha a disposizione un animatore di specifica, il quale però si limita a simulare, in maniera interattiva con l'utente, l'esecuzione di un progetto a partire dalla specifica, senza dover rispettare nessun vincolo temporale; per tale motivo questo strumento è inadatto per specificare i requisiti temporali dei sistemi real-time, il cui mancato rispetto, come insegna l'esperienza con Lotos nello sviluppo del software di bordo di Airbus 330/340 [118], spesso decreta il fallimento di un progetto. Al contrario TRIO, grazie alla sua eseguibilità delle specifiche, supporta oltre alla simulazione anche la prototipazione, cioè è perfettamente in grado di costruire prototipi del software specificato che svolgono le sue stesse operazioni nell'ordine prestabilito, rispettando i requisiti temporali della specifica.

I metodi formali operazionali hanno certamente una minore maturità industriale anche perché non si prestano ad essere usati (almeno da soli) per sistemi grandi e complessi. Essi hanno minori doti di scalabilità, modularità e astrazione. Quando sono stati usati per specificare sistemi di una certa entità, sono sempre stati affiancati da linguaggi descrittivi per la specifica di tipi di dati astratti e per evitare problemi di esplosione degli stati.

Gli Statecharts sono stati impiegati con un certo successo in applicazioni militari avioniche, anche se con il coinvolgimento degli ideatori del formalismo. Le maggiori difficoltà nel loro utilizzo derivano paradossalmente dalla loro ricchezza: non è facile dare una semantica formale ai più sottili aspetti di comunicazione mediante broadcasting e sincronizzazione tra transizioni. Possiamo perciò concludere che gli Statecharts sono una notazione molto espressiva e intuitiva, di cui però non esiste una semantica formale semplice e completa. Essi possono essere considerati un metodo di specifica semiformale, in quanto ne è data una sintassi precisa, il cui significato si può comprendere in certi casi, e soltanto intuire in certi altri.

RSML è molto elogiato dalla sua principale fautrice [88], ma in realtà più che essere derivato da Statecharts ne costituisce un sottoinsieme semplificato (e con minori capacità), con in più rappresentazioni tabulari dei predicati che descrivono le transizioni di stato. RSML forse aumenta la comprensibilità e la capacità di comunicazione tra esperti di settori diversi già ben presenti in Statecharts, ma mantiene inalterati tutti i difetti del progenitore, con in più l'assenza quasi totale (almeno per ora) di strumenti adeguati. Inoltre *l'eliminazione di alcune delle caratteristiche più interessanti di Statecharts ha nettamente limitato le capacità di RSML,* rispetto al suo antecedente: ad esempio, a causa dell'eliminazione della comunicazione di tipo broadcasting, un segnale emesso da un componente non può più essere ricevuto da un componente parallelo e provocare una sua transizione. RSML si limita a usare l'analisi di raggiungibilità per verificare un piccolo insieme di proprietà e la sua capacità di specificare il comportamento temporale di un sistema è assai limitata, come rilevato da Heitmeyer e Mandrioli [67], nella loro analisi di Statecharts, che certamente si può tranquillamente estendere a un suo sottoinsieme. Inoltre non dimentichiamo che TCAS II è stato certificato solo per un livello di integrità safety 2, che non è un risultato poi così esaltante dal momento che lo si può raggiungere anche senza l'uso dei metodi formali, come prova l'esperienza di [52].

RSML pertanto non può essere consigliato per la specifica del software di controllo e protezione ferroviario, in quanto ha scarse doti di scalabilità e astrazione, non è particolarmente adatto ai sistemi in tempo reale, ha pochi strumenti e non ha un livello di rigore particolarmente elevato.

Le reti di Petri, come già rilevato da Leveson e [127], sono l'ideale per la specifica di sistemi piccoli, ma sono inapplicabili a sistemi grandi, a causa della loro assenza di scalabilità, dovuta all'esplosione degli stati.

Le reti di Petri temporizzate possono essere usate solo per la specifica di alcune *funzioni safety-critical, ma non dell'intero software di protezione e controllo di un sistema ferroviario, in quanto hanno scarse doti di scalabilità, di modularità e di astrazione*. Inoltre secondo [78] le reti di Petri temporizzate perdono gran parte delle doti di comprensibilità e immediatezza proprie delle reti di Petri non temporizzate.

Un'idea interessante sarebbe affiancare le reti di Petri temporizzate ad un metodo di specifica descrittivo come TRIO, sfruttando così al massimo le caratteristiche dei due formalismi e superando le limitazioni delle reti di Petri. Un esempio di applicazione di un tale approccio è fornito in [101], che, però, ammette che *per ottenere pienamente i vantaggi dei due metodi è necessario integrarli anche a livello teorico* e che si stanno compiendo sforzi in tal senso, che sembrano promettenti nel lungo termine.

Per quanto riguarda OBJ vi sono opinioni contrastanti sulle sue reali doti di scalabilità: Storey, in [118], lo giudica apertamente inadatto per sistemi grandi complessi mentre Halang e Kramer, in [60], sostengono che è applicabile a progetti industriali sia grandi che piccoli. Certamente, però, è incontrovertibile che le sue specifiche sono piuttosto "pesanti" e che OBJ è applicabile solo a sistemi sequenziali e non a sistemi concorrenti. Inoltre non è assolutamente adatto alla specifica (e alla verifica) di sistemi in tempo reale. A causa di tali limitazioni *OBJ non sembra applicabile ai sistemi ferroviari che notoriamente sono real-time e, probabilmente, la dipendenza dal tempo delle funzioni ferroviarie safety-critical è anche maggiore di quello che comunemente si crede*, in quanto tutti gli incidenti, oggetto di letteratura, esaminati, sono accaduti non perché non sia stata fatta la cosa giusta, ma perché la cosa giusta è stata fatta al momento sbagliato: si vedano ad esempio [23], dove l'incidente è stato causato da un macchinista che si è fermato troppo presto e [71], dove l'incidente è stato provocato da un errore dello specificatore nel calcolare il tempo necessario per fermare un treno in corsa.

Recentemente, poi, si è affermata ed è stata accettata l'opinione che si debbano usare comunque metodi formali in grado di gestire sistemi concorrenti, in quanto si pensa che l'ambiente vada sempre tenuto in considerazione "a fianco" del sistema in esame [8], ragion per cui sono da preferirsi linguaggi di specifica formale in grado di gestire la concorrenza e da escludere linguaggi orientati esclusivamente ai sistemi sequenziali come OBJ.

CCS *non può essere usato (almeno non da solo) per la specifica di sistemi ferroviari, o di altri sistemi grandi e complessi e/o real-time, a causa delle sue scarse capacità di astrazione, di scalabilità e di modularità, nonché per la sua incapacità di supportare sistemi in tempo reale*. In verità si è tentato di aumentare la capacità di astrazione e la scalabilità di CCS usandolo insieme ad altri formalismi descrittivi come ad esempio ACTL, ma ciò chiaramente ne diminuisce la comprensibilità (già non eccelsa) e ne aumenta la difficoltà di apprendimento, un ostacolo non da poco in un ambiente industriale, già di per sé diffidente verso formalismi, come questo, con scarsa maturità industriale.

LOTOS dev'essere *considerato inadeguato per la specifica di sistemi, come quelli ferroviari, a causa delle sue scarse doti di scalabilità e di modularità, nonché della sua impossibilità di supportare il tempo reale*, ampiamente dimostrata nel progetto del software dei computer di volo di Airbus 330 e 340; in esso, dapprima il progetto in LOTOS era troppo grande per essere processato dagli strumenti di verifica formale, tanto da consentirne la convalida soltanto con simulazione e testing, quindi il programma eseguibile generato dal progetto Lotos era di parecchi ordini di grandezza più lento del necessario.

Il fallimento, appena ricordato, è al tempo stesso una dimostrazione che, per sistemi in tempo reale, sono necessari metodi formali perfettamente in grado di supportare tale caratteristica e che non è sufficiente poter gestire il "tempo relativo", come vorrebbero alcuni autori in riferimento proprio al software dei sistemi di trasporto.

4.3.6 Sintesi delle osservazioni più salienti

La norma EN 50128 è per lo più inadeguata nella parte riguardante la specifica dei requisiti, in quanto non recepisce la necessità di adottare i metodi formali (che tratta e classifica con estrema superficialità) per soddisfare le sue stesse richieste.

La specifica dei requisiti ha un ruolo vitale poiché gli errori di specifica provocano oltre la metà dei fallimenti software ed i maggiori problemi per la safety.

I metodi formali possono:

- eradicare gli errori software dovuti a fraintendimenti dei requisiti (che si sono rivelati i più pericolosi);
- aumentare la comprensione del sistema in esame e delle sue proprietà, aumentando di conseguenza anche la safety e la reliability;
- dimostrare la corrispondenza fra requisiti software e requisiti funzionali e di safety del sistema (in modo estremamente semplice se tutti i requisiti sono scritti col medesimo formalismo);
- verificare la corrispondenza fra i requisiti del sistema di safety, quelli del software e il loro rispetto ai vari livelli di progetto.

Alcuni metodi formali, come ad esempio TRIO, possono verificare anche l'equivalenza fra requisiti informali forniti dal cliente e requisiti formali redatti dal team di sviluppo.

Bisogna sviluppare una cultura dei metodi formali promuovendone lo studio nell'università e rimuovere la diffidenza (dovuta soprattutto a ignoranza) del mondo industriale verso di essi.

I sistemi ferroviari sono sistemi real-time, pertanto è necessario adottare per la loro specifica linguaggi formali che supportino perfettamente il tempo reale.

BIBLIOGRAFIA

1. T. Anderson, P.A. Barrett, D.N. Halliwell, M.R. Moulding: *Software Fault Tolerance: An Evaluation*, IEEE Transactions On Software Engineering, 11(12), Pagg. 1502-1510, Dicembre 1985.
2. J.R. Abrial: *The B-Method*, University Of Cambridge, 1997.
3. T. Anderson, P.A. Lee: *Fault Tolerance: Principles And Practice*, Prentice Hall International, 1981.
4. *Ariane 5 Flight 501 Failure*, Report By the Inquiry Board, Esa, Paris, 19 Luglio 1996, (Http://Www.Esrin.Esa.It/Tidc/Press/Press96/Ariane5rep.Html).
5. A. Avizienis: *The N-Version Approach To Fault Tolerant Software*, IEEE Transactions On Software Engineering, 11(12), Pagg. 1491-1501, Dicembre 1985.
6. A. Avizienis, J.P.J. Kelly: *Fault Tolerance By Design Diversity: Concepts And Experiments*, Computer, 17(8), Pagg. 67-80, Agosto 1984.
7. R. Barnard: *Tool support for an application-specific language*, in F. Redmill e T. Anderson Eds, Directions in safety-critical systems, Springer Verlag, 1993.
8. L.M. Barroca, J.A. Mcdermid: *Formal Methods: And Relevance in Safety-Critical Systems*, The Computer Journal, 35(6), Pagg. 579-599, 1992.
9. F.B. Bastani, C.V. Ramamoorthy: *Software Reliability: Status And Perspectives*, IEEE Transactions On Software Engineering, Pagg. 359-370, Luglio 1982.
10. S. Bologna, G. Dahll, G. Picciolo, R. Taylor: *Applications Of Computer Based Systems For the Process Industry*, Essi Project 21542, Giugno 1997.
11. R. Bell: *Iec Draft International Standard On Functional Safety: Current Position*, Integrity Systems, 1(1), Pagg. 73-77, 1994.
12. A. Bellini, A. Guidi: *Guida Al Linguaggio C*, Mcgraw-Hill Italia, 1996.
13. R. Bell, D. Reinert: *Risk And System Integrity Concepts For Safety-Related Control Systems*, in Safety-Criticalsystems: Current Issues, Techniques And Standards, Pagg. 273-296, Chapman & Hall, 1993.
14. A. Fantechi, C. Bernardeschi: *The Application of Formal Methods in the Development and Validation of Railway Systems*, Safety & Tecnologie Informatiche nei Sistemi di Trasporto, Torino 26 giugno 1996.
15. C. Bernardeschi, A. Fantechi, S. Gnesi: *validazione formale di meccanismi per la tolleranza ai guasti*, Encress Italia-Napoli, 15 aprile 1997.
16. T. E. Bihari, P. Gopinath: *Object-oriented real-time systems: concepts and examples*, IEEE Computer, 25(12), pagg. 25-32, dicembre 1992.
17. S. Brilliant, J.C. Knight, N. Leveson: *The consistent comparison prolem in n-version programming*, IEEE Transactions on software engineering, 15(11), novembre 1989.
18. S. Brilliant, J.C. Knight, N. Leveson: *Analysis of faults in an n-version software experiment*, IEEE Transactions on Software Engineering, 16(2) febbraio 1990.
19. P. Barrett, R. Nissen, M. Rothfelder, F. Saglietti, E. Shoitsch, G. Sonneck: *Software Intensive High Risk Systems for applications in the transportation And Nuclear Industry Sector*, ESSI Project 21142 Draft 4 Aprile 1997.
20. B.W. Boehm, R.L. Mcclean, D.B. Urfig: *Some Experiences With Automated Aids To the Design Of Large-Scale Reliable Software*, IEEE Transactions On Software Engineering, 1(1) marzo 1975.
21. T. Bolognesi, E. Brinksma: *Introduction To the Specification Language Lotos*, Computer Networks ISDN 14, Pagg. 25-59, 1987.
22. A. Boswell: *Specification And Validation Of A Security Policy Model*, Ieee Transactions On Software Engineering, 21(2), Pagg. 63-68, febbraio 1995.
23. R. Botting, C.W. Johnson: *A Generic Model For the Analysis Of Railway Accidents*, Proceedings Of Eurographics Workshop On the Design, Specification And Verification Of Interactive Systems, 1998.

24. J.L. Boulanger, H. Waeselynck: *The Role Of Testing In the B Formal Development Process*, in Proceedings Of the 6th International Symposium On Software Reliability Engineering, Pagg. 58-67, Tolosa, Ottobre 1995.
25. J.P. Bowen, M.G. Hinchey: *Seven More Miths Of Formal Methods*, IEEE Software, 12(4), Pagg 34-41, 1995.
26. J.P. Bowen, V. Stavridou: *Safety-Critical Systems, Formal Methods And Standards*, IEE/BCS Software Engineering Journal, 8(4), Pagg. 189-209, Luglio 1993.
27. R.S. Boyer, J.S. Moore: *A Computational Logic Handbook*, Academic Press, New York, 1988.
28. J. Brazendale, R. Bell: *Safety-Related Control And Protection Systems: Standards Update*, Computing Control Engineering Journal, Pagg. 6-12, 5(1) 1994.
29. J. Brazendale, A.R. Jeffs: *Out Of Control: Failures Involving Control Systems*, High Integrity Systems, 1(1), Pagg. 67-72, 1994.
30. J.R. Brown, M. Lipow: *Testing For Software Reliability*, Proceedings International Conference On Reliable Software, Pagg. 518- 526.
31. A. Burns, A. Wellings: *Real-Time Systems And Their Programming Languages*, Addison Wesley 1989.
32. R.W. Butler, G.B. Finelli: *The Infeasibility Of Quantifying The Reliability Of Life-Critical Real-Time Software*, IEEE Transactions On Software Engineering, 19(1), Pagg. 3-12, Gennaio 1993.
33. R.W. Butler, J.A. Sjogren: *Formal Design Verification Of Digital Circuitry*, in Reliability Engineering And System Safety, Littlewood And Miller Eds, Elsevier Science, 1991.
34. P.R. Croll, C. Chambers, M. Bowell: *A Study Of Incidents Involving Electrical/Electronic/Programmable Electronic Safety-Related Systems*, Workshop On Human Error And System Development, 19-22 Marzo 1997 Universita' di Glasgow.
35. D. Craigen, S Gerhart, T. Ralston: *Formal Methods Reality Check: Industrial Usage*, IEEE Transactions On Software Engineering, 21(2), Pagg. 90-98, Febbraio 1995.
36. D. Coleman, F. Hayes, S. Bear: *Introducing Objectcharts Or How To Use Statecharts in Object-Oriented Design*, Iee Transactions On Software Engineering, 18(1), Pagg. 9-18, Gennaio 1992.
37. W.J. Cullyer, S.J. Goodenough, B.A. Wichmann*: The Choice Of Computer Languages For Use in Safety Critical Systems*, Software Engineering Journal, 6(2), Pagg. 51-58, Febbraio 1991.
38. S.S. Cha, N. Leveson, T.J. Shimeall: *Safety Verification Of Ada Programs Using* Software Fault Trees, IEEE Software, 8(7), Pagg. 48-58, Luglio 1991.
39. Contesse: *Contesse Test Handbook*, Glasgow Baesema, 1995.
40. A.H. Cribbens: Solid State Interlocking (Ssi): *An Integrated Electronic Signalling System For Mainline Railways*, Iee Proceedings, Pagg 148-158, Maggio 1987.
41. W.J. Cullyer, N. Storey: *Tools And Techniques the Testing Of Safety-Critical Software*, Computing Control Engineering Journal, 5, Pagg. 239-244, 1994.
42. B. Dasarathy: *Timing Constraints Of Real-Time Systems: Constructs For Expressing Them*, Methods Of Validating Them, IEEE Transactions On Software Engineering, 11(1), Pagg. 80-86,Gennaio 1985.
43. E.W. Dijkstra: *A Discipline Of Programming*, Prentice-Hall, 1976.
44. D.E. Eckhardt, L. Lee: *A Theoretical Basis For the Analysis Of Multiversion Software Subject To Coincident Errors*, IEEE Transactions On Software Engineering,11(12) Pagg. 1511-1516, Dicembre 1985.
45. D.E. Eckhardt, A.K. Caglayan, J.C. Knight, L. Lee, D.F. Mcallister, M.A. Vouk, J.P.J. Kelly: *An Experimental Evaluation Of Software Redundancy As A Strategy For Improving Software Reliability*, IEEE Transactions On Software Engineering, 17(7), Pagg. 692-702, Luglio 1991.
46. A. Endres: *An Analysis Of Errors And Their Causes In System Programs*, IEEE Transactions On Software Engineering, 6(1), Pagg. 140-149, Giugno 1975.
47. N.E. Fenton, M. Neil: *A Strategy for Improving Safey Related Software Engineering Standards*,

IEEE Transactions On Software Engineering, 24(11), Pagg. 1002-1013, Novembre 1998.

48. M.D. Fraser, K. Kumar, V.K. Vaishnavi: *Strategies For Incorporating Formal Specifications In Software Development*, Communications Of the Acm, 37(10), Pagg. 74-86, Ottobre 1994.
49. R.W. Floyd: *Assigning Meanings To Programs, In Proceedings Symposium In Applied Mathematics*, American Mathematical Society, Pagg. 19-32, 1967.
50. M.A. Friedman, J.M. Voas: *Software Assessment: Reliability, Safety, Testability*, J. Wiley & Sons, 1995.
51. S. Gerhart, D. Craigen, T. Ralston: *Experience With Formal Methods In Critical Systems*, IEEE Software, Gennaio 1994.
52. P. Giannino', M. Ficco: *PLC Safety Critical, Modellizzazione Funzionale E Metodi Top-Down/Bottom-Up Affidabilistici Ai Fini Di Validazione*, Encress Italia, Napoli 15 Aprile 1997.
53. C. Ghezzi, A. Fuggetta, S. Morasca, A. Morzenti, M. Pezze': *Ingegneria Del Software*, Mondadori Informatica, 1991.
54. A. Gargantini, L. Liberati, A. Morzenti, C. Zacchetti: *Specifying, Validating, and Testing a Traffic Management System in the Trio Environment, Safety e Tecnologie Informatiche nei Sistemi di Trasporto*- Encress – Torino 26 Giugno 1996.
55. P.L. Goddard:*Validating the Safety of Embedded Real-Time Control Systems Using Fmea*, Proceedings of the Annual Reliability and Maintainability Symposium, Pagg. 227-230, Atlanta, 1993.
56. A.L. Goel, K. Okumoto: *A Time Dependent Error Detection Rate Model for Software Reliability And Othr Performance Measures*, IEEE Transactions on Reliability, Vol. R-28, Pagg. 206-211, 1979.
57. C.T. Gray: *A Framework for Modeling Software Reliability,* In Software Reliability: State Of the Art Report, A. Bendell, P. Mellor Eds, Pagg. 81-94, Pergamon Infotech, Londra, 1986.
58. H. Habrias: *From Z To B In the Computing Department Of the Iut Of Nantes*, Lettre B, Pagg. 24-26, Novembre 1996.
59. A. Hall: *Seven Miths Of Formal Methods*, IEEE Software, Pagg. 11-19, Settembre 1990.
60. W. A. Halang, B.J. Kramer: *Safety Assurance In Process Control*, IEEE Software, Pagg. 61-67, Gennaio 1994.
61. R. Hamlet: *Testing For Trustworthiness*, In J.P. Jacky And D. Schuler Eds: Directions And Implications Of Advanced Computing, Pagg. 97-104, Ablex Publishing Company, 1989.
62. R. Hamlet, R. Taylor: *Partition Testing Does Not Inspire Confidence*, Proceedings Of 2nd Workshop On Software Testing, Verification And Analysis, IEEE Computer Society Press, Pagg. 206-215, Washington, 1988.
63. David Harel: *Statecharts: a visual formalism for complex systems*, Science of Computer Programming 8, 1987.
64. P.R. Harvey, N. Leveson: *Analyzing Software Safety*, IEEE Transactions On Software Engineering, 9(5), Pagg. 569-579, Settembre 1983.
65. Hazards Forum: *Safety-Related Systems: Guidance For Engineers*, the Hazards Forum, 1995.
66. W.L. Heimerdinger, C.B. Weinstock: *A Conceptual Framework For System Fault Tolerance*, Technical Report Cmu/Sei-92-Tr-33, Esc-Tr-92-033, Software Engineering Institute, Carnegie Mellon University, Pittsburg, Usa, 1992.
67. C. Heitmeyer, D. Mandrioli: *Formal Methods For Real-Time Computing: An Overview*, In Formal Methods For Real-Time Computing, Edited By Heitmeyer And Mandrioli, J, Wiley & Sons, 1996.
68. C. Hennebert: *Grace A' La Methode B*, Pari Tenu Pour Meteor, Lettre B, Pagg. 6-7, Aprile 1998.
69. M.P.E. Heimdahl, H. Hildreth, N Leveson, J.D. Reese: *Requirements Specification For Process-Control Systems*, IEEE Transactions On Software Engineering, 20(9) Settembre 1994.
70. C.A.R. Hoare: *An Axiomatic Basis Of Computer Programming*, Communications Of the Acm, 12(10), Pagg. 576-580, Ottobre 1969.
71. E. Hollnagel: *Human Machine Interaction (Hmi): Performance And Failures In Safety Critical*

Domains, In F.A. Schreiber Ed.: Human Computer Interface / Human Error In Safety Applications, Encress Italia, Venezia 3 Aprile 1998.

72. C.M. Holloway, R.W. Butler: *Impediments To Industrial Use Of Formal Methods*, IEEE Computer, Pagg. 25-26, Aprile 1996.
73. IEE/BCS Workshop Report, Glasgow, 20 Marzo 1995: Formal Methods And Their Role In Developing Safe Systems.
74. *Specifying A Safety-Critical Control System In Z*, IEEE Transactions On Software Engineering, 21(2), Pagg. 99-106, Febbraio 1995.
75. P. Jalote: *Functional Refinement And Nested Objects For Object-Oriented Design*, IEEE Transactions On Software Engineering, 15(3), Pagg. 264-270, Marzo 1989.
76. M.S. Jaffe, N.G. Leveson, M.P.E. Heimdahl, B.E. Melhart: *Software Requirements Analysis For Real-Time Process-Control Systems*, IEEE Transactions On Software Engineering, 17(3), Pagg. 241-258, Marzo 1991.
77. B.W. Johnson: *Design And Analysis Of Fault Tolerant Digital Systems*, Addison Wesley, 1989.
78. C. Johnson, L. Love: *Using Diagrams To Support the Analysis Of System 'Failure' And Operator 'Error'*, Workshop On Human Error And Systems Development, Universita' Di Glasgow, 19-22 Marzo 1997.
79. P.A. Keiller, D.R. Miller: *On the Use And the Performance Of Software Reliability Growth Models*, In Reliability Engineering And System Safety, Littlewood And Miller Eds, Elsevier Science, 1991.
80. R.A. Kemmerer: *Integrating Formal Methods Into the Development Process*, IEEE Software, Pagg. 37- 48, Settembre 1990.
81. J.P.J. Kelly, T.I. Mcvittie, W.J. Yamamoto: *Implementing Design Diversity To Achieve Fault Tolerance*, IEEE Software, Luglio 1991.
82. J.C. Knight, P.E. Amman: *Design Fault Tolerance,* In Reliability Engineering And System Safety, Littlewood And Miller Eds, Elsevier Science, 1991.
83. J.C. Knight, N. Leveson: *An Experimental Evaluation Of the Assumption Of Independence In Multiversion Programming*, IEEE Transactions On Software Engineering, 12(1), Gennaio 1986.
84. J.Z. Lavi et Al.: *Engineering of Computer-Based Systems: a proposed curriculum for a degree program at Bachelor level* and *Enhancement courses: proposed course outline,* Proc. International Conference on Engineering of Computer-Based Systems, IEEE Computer Society, March 1988, pp. 369-381
85. N. Leveson, S.S. Cha, J.C. Knight, T.J. Shimeall*: The Use Of Self Checks And Voting Software Error Detection: An Empirical Study*, IEEE Transactions On Software Engineering, 16(4), Pagg. 432-443, Aprile 1990.
86. N. Leveson: Software Safety: *Why, What, And How*, Computing Surveys, 18(2), Pagg.125-163, Giugno 1986.
87. N. Leveson: *Software Safety In Embedded Computer Systems*, Communications Of the Acm, 34(2), Pagg. 34-46, Febbraio 1991.
88. N. Leveson: Safeware: *System Safety And Computers*, Addison Wesley, 1995.
89. B. Littlewood: *Theories Of Software Reliability: How Good And How Can They Be Improved?,* IEEE Transactions On Software Engineering 6(5), Pagg. 489-500, settembre 1980.
90. B. Littlewood, L. Strigini: *The Risks Of Software,* Scientific American, Pagg. 62-75, Novembre 1992.
91. B. Littlewood, L. Strigini: *Validation Of Ultrahigh Dependability For Software-Based Systems*, Communication Of the Acm, Pagg. 69-80, Novembre 1993.
92. B. Littlewood, J.L. Verrall: *A Bayesian Reliability Growth Model For Computer Software*, Appl. Statist., Vol. 22, Pagg. 332-346, 1973.
93. R. De Lemos, A. Saeed, T. Anderson: *A Train Set As A Case Study For the Requirements Analysis Of Safety-Critical Systems*, The Computer Journal, 35(1), Pagg. 30-40, Gennaio 1992.
94. B. Mermet, D. Mery: *Specification Of Services And Management Of Interactions*, Lettre B, Pagg. 6-

7, Aprile 1997.

95. R. Milner: *Communication And Concurrency*, Prentice Hall International, 1989.

96. D.R. Miller: *The Role Of Statistical Modeling And Inference In Software Quality Assurance, In Software Certification*, B. De Neuman Ed., Pagg. 135-152, Elsevier Applied Science, Londra, 1989.

97. D. Musa, A. Iannino, K. Okumoto: *Software Reliability: Measurement, Prediction, Application*, New York, Mcgraw Hill, 1987.

98. A. Morzenti: *Sistemi In Tempo Reale Proposta Di Un Formalismo Logico Di Specifica E Convalida*, Clup Citta'studi 1991.

99. A. Morzenti, F.A. Schreiber: *Analysis Of Techniques In the Assessment Of Ultra-Reliable Computerized Systems*, Software Engineering Notes, Marzo 1997.

100. C. Mazzola, G.M. Menafoglio, P. Merenda, F. Senesi: *Elementi Per L'adeguamento Della Normativa Italiana Per Metropolitane Ad Automazione Integrale*, I Parte, Trasporti E Trazione Gennaio 1997, II Parte, Trasporti E Trazione Febbraio 1997.

101. D. Mandrioli, A. Morzenti, M. Pezze', P. San Pietro, S. Silva: *A Petri Net And Logic Approach To the Specification And Verification Of Real-Time Systems*, In Formal Methods For Real-Time Computing, Edited By Heitmeyer And Mandrioli, J. Wiley & Sons, 1996.

102. J.D. Musa: *A Theory Of Software Reliability And Its Application*, IEEE Transactions On Software Engineering 1(3), Pagg. 312-326, settembre 1975.

103. P.G. Neumann: *Certifying Professional,* Communications Of the Acm, Febbraio 1991.

104. P.G. Neumann: *Computer Related Risks*, Addison Wesley, 1995.

105. NUREG/CR-6263: *High Integrity Software Nuclear Power Plants*, Prepared For Usnrc From Mitre Corporation.

106. D.L. Parnas, D.K. Peters: *Using Test Oracles Generated From Program Documentation*, IEEE Transactions On Software Engineering, 24(3), Pagg. 161-173, Marzo 1998.

107. D.L. Parnas, J. Van Schouven, P.K. Shu: *Evaluation Of Safety-Critical Software*, Communications Of the Acm, 33(6), Pagg. 636-648, Giugno 1990.

108. N.C. Rasmussen: *The Application Of Probabilistic Risk Assessment Techniques To Energy Technologies*, In T.S. Glickman And M. Gough Eds: Reading In Risk, Pagg. 195-205, Resources For The Future, 1990.

109. A.P. Ravn, H. Rischel, K.M. Hansen: *Specifying And Verifying Requirements Of Real-Time Systems*, IEEE Transactions On Software Engineering, 19(1), Pagg. 41-55, Gennaio 1993.

110. D.J. Reifer*: Software Failure Modes And Effects Analysis*, IEEE Transactions On Reliability, 28(3), Pagg. 247-249, Agosto 1979.

111. J.P. Rosen: *What Orientation Should Ada Objects Take?*, Communication Of the Acm, 35(11), Pagg. 71-76, Novembre 1992.

112. F.A. Schreiber Ed.: *Human Computer Interface / Human Error Analysis In Safety Applications*, Encress Italia, Venezia 3 Aprile 1998.

113. F.A. Schreiber, A. Morzenti: *Problematiche di certificazione del Software Safety-critical*, Rapporto di avanzamento del progetto "Scienza e Tecnica per la qualità della mobilità locale", Politecnico di Milano, Dicembre 1998, pp. VI+150

114. T.J. Shimeall, N.G. Leveson: *An Empirical Comparison Of Software Fault Tolerance And Fault Elimination*, IEEE Transactions On Software Engineering, 17(2), Pagg. 173-182, Febbraio 1991.

115. M.L. Shooman*: Probabilistic Models For Software Reliability Prediction*, In W.B.Freiberger Ed.: Statistical Computer Performance Evaluation, Pagg. 485-502, New York, Academic, 1972.

116. D. P. Siewiorek, R.S. Swartz: *The Teory And Practice Of Reliable System Design*, Digital Press, 1982.

117. E. Skoglund: *Iec 1508 Certification Of A Real-Time Operating System*, Encress Italia, Napoli 15 Aprile 1997.

118. N. Storey: *Safety-Critical Computer Systems*, Addison Wesley, 1996.

119.B. Stroustrup: *The C++ Programming Language*, Addison Wesley, 1991.

120.J.R. Taylor: *An Integrated Approach To Computer System Safety-Hardware, Software And Human Error*, Encress Italia, Napoli 15 Aprile 1997.

121.M. Thomas: *Formal Methods And Their Role In Developing Safe Systems*, High Integrity Systems,Vol 1, N.5, 1996.

122.A. Villemeur*: Reliability, Availability, Maintanability And Safety Assessment*, John Wiley & Sons, 1991.

123.J.M. Voas, K.W. Miller: *Software Testability: The New Verification*, IEEE Software, Maggio 1995.

124.J. Crow, S. Owre, J. Rushby, N. Shankar, M. Srivar: *A Tutorial Introduction To PVS,* Workshop On Industrial-Strength Formal Specification Techniques, Boca Raton, Florida, Aprile 1995.

125.L. G. Williams: *Assessment Of Safety-Critical Specifications*, IEEE Software, Pagg. 51-60, Gennaio 1994.

126.S. Yau, J. Tsai: *A Survey Of Software Design Techniques*, IEEE Transactions On Software Engineering, 12(6), Pagg. 713-721, giugno1986.

127.N. Leveson, J. Stolzy: *Safety Analysis Using Petri Nets*, IEEE Transactions on Software Engineering, 13 (3), Pagg. 386-397, Marzo 1987.

128.J. Suokas, R. Kakko: *On the Problems and Future of Safety and Risk Analysis*, Journal of Hazardous Materials, Vol. 21, Pagg. 105-124, 1989.

129.G. Myers: *The art of software Testing*, J. Wiley & Sons, 1979.

130.D. Mandrioli, S. Morasca, A. Morzenti: *Generating Test Cases for Real Time Systems from Logic Specifications*, ACM Transactions on Computer Systems, vol. 13, n. 4, Pagg. 365-398, 1995