

Analisi statica e verifica dei codici per creare software embedded di qualità elevata

All'aumentare della complessità dei software embedded cresce il numero di utenti che si affidano all'analisi statica del codice, abbinata a un modello di qualità del software per gestirne la qualità

Jay Abraham e Marc Lalo
Product marketing manager per Polyspace
MathWorks



Lo sviluppo del software e le procedure di test si basano oggi su molte best practice e metodologie. Le preferenze e le esperienze personali – sia positive sia negative – influenzano la maggior parte dei flussi di lavoro. Strumenti e script personalizzati vengono spesso abbinati a strumenti di automazione interni ed esterni. Tuttavia, al centro di tutto ciò vi è un insieme di metodi comprovati di sviluppo e test, che consente la realizzazione di software di qualità elevata quasi completamente privi di difetti. Aderenza agli standard di codifica, verifica del software in una fase iniziale del processo di sviluppo, controllo basato su un insieme ben definito di parametri di valutazione della qualità, identificazione delle operazioni software funzionanti e di quelle che contengono codice errato: tutti questi elementi porteranno qualità e coerenza al software embedded.

Il punto di partenza: definire modelli e obiettivi in tema di qualità del software

Il software embedded dei sistemi safety, quelli in cui l'affidabilità è d'obbligo, diventa sempre più complesso. Visto che i sistemi sono via via più sofisticati, le aziende che si occupano di sviluppo software devono raggiungere obiettivi "impegnativi" di qualità, stabiliti internamente oppure imposti dal cliente o dalle normative. Affinché i team addetti allo sviluppo software possano raggiungere questi obiettivi e - idealmente - creare software privi di difetti, è necessario definire un modello di qualità.

Tale modello stabilisce obiettivi specifici in fatto di qualità, con metriche rigorose. Questi obiettivi forniscono un meccanismo mediante il quale i team possono comunicare obiettivi e stato, sia internamente sia con persone esterne all'azienda. Gli strumenti di automazione dell'analisi statica del codice forniscono metriche quantificabili per la qualità del software. Come suggerito dal termine statica, questi strumenti analizzano il codice sorgente senza richiedere l'esecuzione del programma. Usando varie tecniche, tra cui la scansione del codice e l'interpretazione astratta, questi strumenti rilevano gli errori presenti nel codice e misurano la conformità agli standard di codifica. I più avanzati tra gli strumenti di analisi statica dei codici possono dimostrare formalmente che il codice sorgente è privo di specifici errori di runtime.

Esempio di un modello di qualità del software con obiettivi

Un esempio di tale modello di qualità del software con obiettivi ben definiti, sviluppato da vari OEM del settore automobilistico e da MathWorks, è descritto in un documento intitolato "Software Quality Objectives for Source Code" (1). Questo modello si basa su una serie di obiettivi in termini di qualità del software, molti dei quali richiedono implicitamente l'uso dell'analisi statica del codice. La conformità a questi obiettivi viene valutata usando i seguenti criteri:

- esistenza di un piano di qualità;

- descrizione dettagliata del progetto completata;
- metriche ben definite per il codice;
- standard di codifica rispettati (ad esempio, aderenza al MISRA-C);
- nessuna ramificazione non raggiungibile (no dead code);
- nessun costrutto non finito (no loops);
- eliminazione degli errori di runtime o comprensione del potenziale di errori di runtime;
- analisi del flusso di dati completata.

Sulla base di questi criteri, il documento stabilisce sei obiettivi di qualità del software (SQO): l'obiettivo SQO-1 comporta il rispetto solo di alcuni criteri, mentre l'obiettivo SQO-6 richiede il rispetto di tutti i criteri. La scelta dell'SQO appropriato per un certo progetto dipende da:

- criticità del progetto;
- processi di qualità interni all'azienda, che possono prevedere l'applicazione dello standard CMMI (Capability Maturity Model Integration) o dello standard SPICE (Software Process Improvement and Capability Determination);
- necessità di rispettare standard quali IEC 61508, ISO 26262 o DO 178B.

La tabella 1 mostra i criteri applicabili ai vari SQO, alcuni dei quali sono associati a una metrica di soglia. Il superamento della soglia innesca il raggiungimento di uno specifico SQO. Per raggiungere l'obiettivo SQO-1, ad esempio, un'azienda deve avere un piano per la qualità, una descrizione dettagliata del progetto e un insieme di metriche ben definite per il codice. Per raggiungere l'obiettivo SQO-2, l'azienda deve rispettare anche le regole di base degli standard di codifica, eliminare tutti gli errori sistemici di runtime e verificare che nel codice non esistano costrutti non finiti. Per raggiungere l'obiettivo SQO-3, l'azienda deve in più dimostrare che non vi sono ramificazioni non raggiungibili (dead code o codice morto).

Usare un sottoinsieme di linguaggio per conformarsi agli standard di codifica

I linguaggi generici, tra cui C e C++, sono stati progettati per coprire un'ampia gamma di applicazioni, dal software desktop ai sistemi embedded.

Grazie alle estensioni come C99 per il linguaggio C e quelle fornite dai compilatori GNU o Visual C++, questi linguaggi si sono evoluti fino a consentire un maggior numero di pattern di progettazione.

L'inconveniente di questi progressi è il maggior costo della verifica di codici complessi. I linguaggi più complessi sono più difficili da verificare, sia manualmente sia usando uno strumento

automatico. Per semplificare la verifica, la maggior parte degli standard (tra cui IEC 61508 e ISO 26262, Tab. A.3) limita l'uso del linguaggio a un sottoinsieme limitato.

Per conformarsi allo standard, un'azienda deve usare solo quelle funzioni del linguaggio consentite dallo standard stesso.

Il modello di qualità illustrato nella tabella 1, ad esempio, richiede il rispetto dello standard di codifica MISRA-C:2004.

Oltre a semplificare la verifica del codice, gli standard di codifica portano alla creazione di codici più facili da leggere, man-

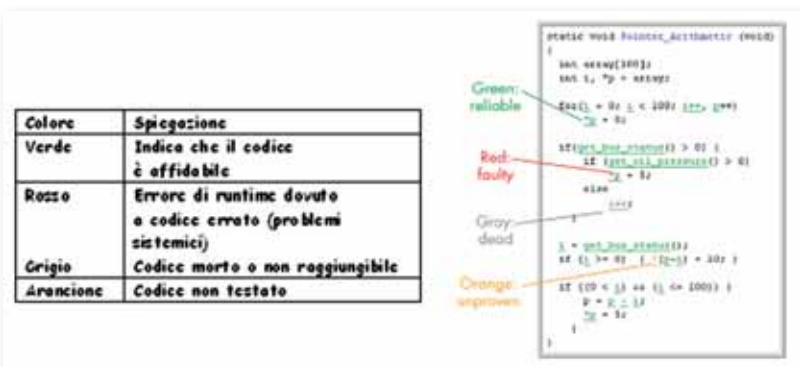


Fig. 1 - Risultati di Polyspace

tenere e trasferire. In generale, l'adozione di uno standard di codifica non significa che è necessario applicare tutte le regole previste dallo standard stesso.

Il modello di qualità degli OEM automobilistici descritto sopra stabilisce due sottoinsiemi di standard di codifica MISRA-C.

Il primo sottoinsieme, richiesto per raggiungere gli obiettivi da SQO-1 a SQO-4, include regole come le seguenti:

- 8.11: lo specificatore di classe di storage statico deve essere usato nelle definizioni e nelle dichiarazioni relative a oggetti e funzioni che hanno un collegamento interno;
- 8.12: quando un array viene dichiarato con collegamento esterno, la sua dimensione deve essere affermata esplicitamente o definita implicitamente mediante inizializzazione;
- 13.3: le espressioni in virgola mobile non devono essere testate in termini di uguaglianza o disuguaglianza;
- 20.2: non deve essere usata l'allocazione di memoria dinamica.

Il secondo sottoinsieme, richiesto per raggiungere gli obiettivi SQO-5 e SQO-6, include regole come le seguenti:

- 8.7: gli oggetti devono essere definiti a livello di blocco se l'accesso ad essi avviene solo dall'interno di una singola funzione;
- 9.2: è necessario usare le graffe per indicare e far corrispondere la struttura nell'inizializzazione non-zero

SOFTWARE

STATIC ANALYSIS

- degli array e delle strutture;
- 13.1: non si dovranno usare gli operatori di assegnazione nelle espressioni che producono un valore booleano;
 - 20.3: è necessario verificare la validità dei valori passati a funzioni di libreria.

Individuare gli errori di esecuzione dinamici nel codice sorgente

Trovare gli errori è il pensiero della maggior parte degli sviluppatori di software quando cercano di migliorare la qualità del loro codice. Una volta salvato e poi compilato il file sorgente, il riflesso immediato è quello di eseguirlo per trovare eventuali errori. Tuttavia, è possibile identificare molti errori di runtime (detti anche errori di esecuzione dinamici) analizzando il codice sorgente in sé.

Prendiamo in esame l'errore illustrato nella tabella 2. In questo esempio, il prototipo della funzione C non corrisponde alla sua implementazione. Di conseguenza, quando questo codice viene compilato ed eseguito, produce risultati imprevisti in un sistema embedded.

Il prototipo della funzione `Arg_f()` specifica che l'argomento è un `int`, ma nell'implementazione della funzione l'argomento è un `float`. È probabile che alcuni compilatori rilevino questo errore, e la maggior parte degli strumenti di analisi statica dei codici lo rileva.

Dimostrare l'assenza di errori di runtime

Una volta individuato e corretto un errore, potrebbe succedere che il codice non soddisfi ancora tutti i requisiti di qualità del software, poiché nel codice stesso potrebbero esserci altri errori non rilevati. Nell'esempio di cui sopra, vi è una potenziale divisione per zero nell'affermazione $r=1/(1-r)$, poiché `r` è dichiarato un volatile `int`.

Dimostrare l'assenza di errori di runtime significa controllare che nel codice non

Tabella 1 - Modello di qualità con criteri per raggiungere obiettivi specifici						
Criteri	Obiettivi di qualità per i software (SQO)					
	SQO-1	SQO-2	SQO-3	SQO-4	SQO-5	SQO-6
Piano di qualità in vigore	X	X	X	X	X	X
Descrizione dettagliata eseguita	X	X	X	X	X	X
Metriche ben definite per il codice	X	X	X	X	X	X
Primo sottoinsieme di regole degli standard di codifica rispettato	X	X	X	X	X	X
Secondo sottoinsieme di regole degli standard di codifica rispettato					X	X
Errori sistemici di runtime eliminati		X	X	X	X	X
Assenza di costrutti non finiti		X	X	X	X	X
Assenza di ramificazioni non raggiungibili			X	X	X	X
Primo sottoinsieme di potenziali errori di runtime eliminato				X	X	X
Secondo sottoinsieme di potenziali errori di runtime eliminato					X	X
Terzo sottoinsieme di potenziali errori di runtime eliminato						X

Tabella 2 - L'implementazione di una funzione (a destra) con la sua prototipazione e il suo utilizzo scorretti (a sinistra)

<pre>void Arg_f(int *y); void WrongArg(void) { volatile int r=0; Arg_f(&r); r = 1/(1-r); }</pre>	<pre>void Arg_f(float *y) { *y=12; }</pre>
---	--

siano presenti divisioni per zero, overflow, accessi ad array al di fuori dei limiti o errori simili. Eliminare tipologie specifiche di errori di runtime è un elemento fondamentale di molti modelli di qualità del software. Nel modello degli OEM automobilistici descritto sopra, l'insieme di potenziali errori di runtime che deve essere eliminato si espande man mano che gli obiettivi SQO diventano più vincolanti.

L'assenza di errori di runtime può essere dimostrata mediante la verifica del codice, usando tecniche matematiche. Basate su metodi formali, queste tecniche vengono applicate per rilevare e dimostrare l'assenza di errori di runtime nel codice sorgente. Polyspace di MathWorks è uno strumento che può essere usato per illustrare l'applicazione di queste tecniche. Polyspace (2) è un verificatore di codice che rileva e dimostra l'assenza di errori di runtime specifici nel codice sorgente C, C++ o Ada usando l'analisi statica: in tal modo, non è necessaria l'esecuzione del programma. Per prima cosa, Polyspace esamina il codice sorgente per stabilire dove si potrebbero verificare dei potenziali errori di runtime. Genera poi un report che usa una codifica basata su colori per indicare lo stato di ciascun elemento del codice (Fig. 1). Il codice in verde è privo di determinati errori di runtime. Il rosso identifica il codice che causerà un errore di runtime ogni volta che viene eseguito.

Il codice in grigio non è raggiungibile (codice morto), mentre l'arancione indica il codice non testato e che potrebbe contenere un errore di runtime. Polyspace verifica inoltre il codice in termini di conformità allo standard MISRA e include strumenti per l'analisi dei dizionari dei dati e del flusso dei dati. Gli sviluppatori e i tester possono usare le informazioni generate dal processo di verifica del codice per correggere gli errori di runtime identificati; inoltre, possono usare le metriche e i risultati che dimostrano l'assenza di errori di runtime per soddisfare i criteri definiti dagli obiettivi di qualità del software per il loro progetto.

Per rispettare i criteri dell'obiettivo SQO-2 nel modello di qualità degli OEM automobilistici, il codice non può contenere

alcun errore di runtime sistemico, e nemmeno costrutti non finiti. Questo significa che i risultati di Polyspace non devono contenere parti di codice in rosso. Per migliorare ulteriormente la qualità del software e raggiungere l'obiettivo SQO-3, nel codice non possono essere presenti ramificazioni non raggiungibili. In altre parole, i risultati di Polyspace non possono contenere codice in grigio.

Poiché il codice non testato (indicato in arancione da Polyspace) non è un problema evidente (il codice potrebbe non funzionare in alcune condizioni), il modello di qualità degli OEM automobilistici stabilisce varie soglie che definiscono quanto di tale codice può esistere per raggiungere gli obiettivi SQO-4, SQO-5 e SQO-6. L'SQO-4, ad esempio, richiede che sia sicuro l'80% delle potenziali

operazioni di divisione per zero. Se Polyspace mostra che è sicuro il 70% delle operazioni, è necessario eseguire una revisione manuale per dimostrare che è sicuro o giustificato anche un altro 10%. Per l'SQO-5, la soglia sale al 90%. Per l'SQO-6, la soglia è pari al 100%. Gli obiettivi di qualità del software consentono che la prima consegna del software avvenga con parte del codice non testato; al momento della consegna del software finale (SQO-6), il codice deve però essere interamente testato.

La definizione di un modello di qualità del software con obiettivi ben definiti è una best practice per le applicazioni embedded critiche dal punto di vista della qualità. L'analisi statica del codice può essere usata per soddisfare in modo efficiente i criteri di tali modelli, verificando la conformità agli standard di codifica, identificando gli errori di runtime e il codice morto, e consentendo ai team di lavoro di quantificare gli errori di runtime potenziali nel codice sorgente. Man mano che la complessità dei software embedded cresce, aumenta il numero di OEM automobilistici che si affidano all'analisi statica del codice abbinata a un modello di qualità del software per gestire la qualità del software stesso. Applicando procedure coerenti e strumenti automatici, i team di progettazione software automobilistici sono maggiormente in grado di identificare il codice che potrebbe funzionare e possono così concentrarsi sul codice che ha maggiori probabilità di non funzionare. Creare software di qualità elevata non è mai facile, ma oggi è possibile: quello che serve sono una procedura e strumenti che consentano di misurare la qualità e migliorarla costantemente.

Bibliografia

1. *Software Quality Objectives for Source Code*. [HYPERLINK "http://www.erts2010.org/Site/0ANDGY78/Fichier/PAPIERS%20ERTS%202010/ERTS2010_0035_final.pdf"](http://www.erts2010.org/Site/0ANDGY78/Fichier/PAPIERS%20ERTS%202010/ERTS2010_0035_final.pdf) Toulouse : s.n., 2010.
2. *Polyspace è un prodotto di The MathWorks*: [HYPERLINK "http://www.mathworks.com/products/polyspace"](http://www.mathworks.com/products/polyspace) www.mathworks.com/products/polyspace