



Verifica e validazione:
analisi statica



Anno accademico 2014/15
Ingegneria del Software mod. A

Tullio Vardanega, tullio.vardanega@math.unipd.it

Laurea in Informatica, Università di Padova 1/30



Verifica e validazione: analisi statica

Premessa – 2

- **Il SW di tali sistemi deve possedere**
 - Tutte le capacità funzionali specificate nei requisiti, che determinano cosa il sistema debba fare
 - Tutte le caratteristiche non funzionali necessarie per garantire che il sistema lavori sempre come previsto
- **Ciò richiede il possesso dimostrabile (verificabile) di determinate proprietà**
 - Di costruzione (progettazione, architettura, integrazione)
 - D'uso
 - Di funzionamento

Laurea in Informatica, Università di Padova 3/30



Verifica e validazione: analisi statica

Premessa – 1

- **Un numero crescente di sistemi SW incorpora funzionalità critiche**
 - **Rispetto alla sicurezza intesa come *safety***
 - Prevenzione di condizioni di pericolo a persone o cose
 - Sistemi di trasporto pubblico
 - Sistemi per il supporto di transazioni finanziarie
 - Sistemi di controllo di impianti e di produzione
 - **Rispetto alla sicurezza intesa come *security***
 - Prevenzione di intrusioni
 - Sistemi per il trattamento dei dati personali
 - Sistemi per lo scambio di informazioni riservate

Laurea in Informatica, Università di Padova 2/30



Verifica e validazione: analisi statica

Premessa – 3

- **Nessun linguaggio di programmazione garantisce a priori la completa verificabilità di ogni programma scritto in esso**
 - La progettazione di un linguaggio di programmazione richiede bilanciamento tra funzionalità (potere espressivo) e integrità (costo di verifica)
- **Scelto un linguaggio di sviluppo, occorre valutare quali costrutti usarne rispetto al loro impatto su quel bilancio**
 - Potere espressivo vs. costo di verifica

Laurea in Informatica, Università di Padova 4/30

 Verifica e validazione: analisi statica

Tecniche di verifica – 1

❑ Tracciamento /I

- **Dimostrare completezza ed economicità della soluzione**
 - Soddisfacimento di tutti i requisiti
 - Nessuna funzionalità superflua
 - Nessun componente ingiustificato
- **Ha luogo**
 - Tra requisiti *software* e requisiti utente
 - Tra procedure di verifica e requisiti, disegno, codice
 - Tra codice sorgente e codice oggetto (!)
- **Può essere organizzato in modo da automatizzare la manutenzione dei dati e i relativi controlli**



Laurea in Informatica, Università di Padova 5/30

 Verifica e validazione: analisi statica

Tecniche di verifica – 3

❑ Revisioni (*joint review / audit*)

- **Strumento essenziale del processo di verifica**
- **Possono essere condotte su**
 - Analisi, progettazione, codice, procedure e risultati di verifica
- **Non sono automatizzabili**
 - Richiedono la mediazione e l'interazione di individui
- **Possono essere formali (*audit*) o informali (*joint review*)**
 - Richiedono indipendenza tra verificato e verificatore
 - Richiedono semplicità e leggibilità di codice e diagrammi

Laurea in Informatica, Università di Padova 7/30

 Verifica e validazione: analisi statica

Tecniche di verifica – 2

❑ Tracciamento /II

- **Particolari stili di codifica facilitano la verifica mediante tracciamento**
 - Assegnare singoli requisiti elementari singoli moduli del programma così da richiedere una sola procedura di prova e ottenere una più semplice corrispondenza tra essi
 - Maggiore l'astrazione (potenza espressiva) di un costrutto del linguaggio maggiore la quantità di codice oggetto generato per esso e maggiore l'onere di dimostrazione di corrispondenza
 - Esempio di bassa astrazione: addizione tra interi
 - Esempio di alta astrazione: attivazione di un *thread*

Laurea in Informatica, Università di Padova 6/30

 Verifica e validazione: analisi statica

Tipi di analisi statica

1. Flusso di controllo
2. Flusso dei dati
3. Flusso dell'informazione
4. Esecuzione simbolica
5. Verifica formale del codice
6. Verifica di limite
7. Uso dello *stack*
8. Comportamento temporale
9. Interferenza
10. Codice oggetto

Prima di e in aggiunta all'analisi dinamica



Laurea in Informatica, Università di Padova 8/30



Verifica e validazione: analisi statica

1. Analisi di flusso di controllo

- Accertare che il codice esegua nella sequenza specificata**
- Accertare che il codice sia ben strutturato**
- Localizzare codice non raggiungibile**
- Identificare segmenti d'esecuzione che possano non terminare**
 - L'analisi dell'albero delle chiamate (*call-tree analysis*) mostra se l'ordine di chiamata corrisponda alla specifica e rileva la presenza di ricorsione diretta o indiretta
 - Divieto di modifica di variabili di controllo delle iterazioni

Laurea in Informatica, Università di Padova9/30



Verifica e validazione: analisi statica

3. Analisi di flusso d'informazione

- Determinare quali dipendenze tra ingressi e uscite risultino dall'esecuzione di una unità di codice**
- Le sole dipendenze consentite sono quelle previste dalla specifica**
 - Consente l'identificazione di effetti laterali inattesi o indesiderati
- Può limitarsi a un singolo modulo oppure estendere a più moduli correlati oppure anche all'intero sistema**

Laurea in Informatica, Università di Padova11/30



Verifica e validazione: analisi statica

2. Analisi di flusso dei dati

- Accertare che nessun cammino d'esecuzione del programma acceda a variabili prive di valore**
 - Usa i risultati dell'analisi di flusso di controllo insieme alle informazioni sulle modalità di accesso alle variabili (lettura, scrittura)
- Rilevare possibili anomalie**
 - Esempio: più scritture successive senza letture intermedie
- Attività complicata dalla presenza e dall'uso di dati globali raggiungibili da ogni parte del programma**

Laurea in Informatica, Università di Padova10/30



Verifica e validazione: analisi statica

4. Esecuzione simbolica – 1

- Verificare proprietà del programma mediante manipolazione algebrica del codice sorgente**
 - Combinando tecniche di analisi di flusso di controllo, di flusso di dati e di flusso di informazione
- Si esegue effettuando "sostituzioni inverse"**
 - Sostituendo a ogni LHS di un assegnamento il suo RHS
- Trasformando così il flusso sequenziale del programma in un insieme di assegnamenti paralleli le cui uscite sono funzione degli ingressi**

Laurea in Informatica, Università di Padova12/30



Verifica e validazione: analisi statica

4. Esecuzione simbolica – 2

Assumendo assenza di *aliasing* e di effetti laterali di funzioni

```
X = A+B;    -- X dipende da A e B
Y = D-C;    -- Y dipende da C e D
if (X>0)
  Z = Y+1;  -- Z dipende da A, B, C e D
```

```
A+B ≤ 0 ⇒
  X == A+B
  Y == D-C
  Z == Z
A+B > 0 ⇒
  X == A+B
  Y == D-C
  Z == D-C+1
```

Laurea in Informatica, Università di Padova

13/30



Verifica e validazione: analisi statica

6. Analisi di limite

- ❑ **Verificare che i dati del programma restino entro i limiti del loro tipo e della precisione desiderata**
 - Analisi di *overflow* e *underflow*
 - Analisi di errori di arrotondamento
 - Rispetto dei limiti (*range checking*)
 - Analisi di limite di strutture
- ❑ **Linguaggi evoluti assegnano limiti statici a tipi discreti consentendo verifiche automatiche sulle corrispondenti variabili**
 - Più problematico con tipi enumerati e reali

Laurea in Informatica, Università di Padova

15/30



Verifica e validazione: analisi statica

5. Verifica formale del codice

- ❑ **Provare la correttezza del codice sorgente rispetto alla specifica algebrica dei requisiti**
 - Esplorando tutte le esecuzioni possibili
 - Non fattibile tramite analisi dinamica
- ❑ **Correttezza parziale**
 - Le condizioni di verifica sono espresse come teoremi la cui verità implica certe pre-condizioni in ingresso e certe post-condizioni in uscita
 - La prova di correttezza vale sotto l'ipotesi di terminazione del programma
 - La prova di correttezza totale richiede prova di terminazione

Laurea in Informatica, Università di Padova

14/30



Verifica e validazione: analisi statica

7. Analisi d'uso di *stack* – 1

- ❑ **Lo *stack* è l'area di memoria che i sottoprogrammi usano per immagazzinare dati locali, temporanei e indirizzi di ritorno generati dal compilatore**
 - Ogni flusso di controllo (*thread*) ha un suo proprio *stack*
 - L'ampiezza dello *stack* cresce con l'annidamento di chiamate di procedura
- ❑ **L'*heap* invece è l'area di memoria globale**
 - L'ampiezza dell'*heap* è fissata a configurazione e poi consumata con la creazione dinamica di oggetti

Laurea in Informatica, Università di Padova

16/30

Verifica e validazione: analisi statica

Stack & heap

the heap

free blocks blocks in use

Laurea in Informatica, Università di Padova 17/30

Verifica e validazione: analisi statica

8. Analisi temporale

- ❑ Studiare le proprietà temporali richieste ed esibite dalle dipendenze delle uscite dagli ingressi del programma
 - Sapere di produrre il valore giusto al momento giusto
- ❑ Limiti espressivi dei linguaggi e delle tecniche di programmazione complicano questa analisi
 - Iterazioni prive di limite statico (*while*), ricorso sistematico a strutture dati dinamiche (*new*), ...

Laurea in Informatica, Università di Padova 19/30

Verifica e validazione: analisi statica

7. Analisi d'uso di *stack* - 2

- ❑ Determinare la massima domanda di *stack* richiesta da un'esecuzione in relazione con la dimensione dell'area di memoria assegnata al processo
- ❑ Verificare che non vi possa essere collisione tra *stack* e *heap* per qualche esecuzione

Laurea in Informatica, Università di Padova 18/30

Verifica e validazione: analisi statica

9. Analisi d'interferenza

- ❑ Mostrare l'assenza di effetti di interferenza tra parti separate ("partizioni") del sistema
 - Non necessariamente limitate a componenti *software*
- ❑ Veicoli tipici di interferenza
 - Memoria dinamica (*heap*) condivisa, dove parti separate di programma lasciano traccia di dati abbandonati ma non distrutti (*memory leak*)
 - Azzeramento preventivo delle pagine di memoria riutilizzate (p.es. NT v5.x)
 - I/O e altri dispositivi programmabili con effetti a livello sistema (esempio: *watchdog*)

Laurea in Informatica, Università di Padova 20/30

Verifica e validazione: analisi statica

10. Analisi di codice oggetto

- ❑ **Assicurare che il codice oggetto da eseguire sia una traduzione corretta del codice sorgente corrispondente e che nessun errore od omissione siano stati introdotti dal compilatore**
 - Viene ancora effettuata manualmente
 - Viene facilitata dalle informazioni di corrispondenza prodotte dal compilatore

Laurea in Informatica, Università di Padova 21/30

Verifica e validazione: analisi statica

Costo di correzione di errori

Stage	Incremento di costo
Requirements	1
Design	5
Code	10
Unit Test	20
Acceptance Test	50
Maintenance	200

Laurea in Informatica, Università di Padova 23/30

Verifica e validazione: analisi statica

Programmi verificabili – 1

- ❑ **L'adozione di standard di codifica e di sottoinsiemi del linguaggio appropriati deve essere coerente con la scelta dei metodi di verifica adottati**
 - L'uso di costrutti del linguaggio inadatti può compromettere la verificabilità del programma
- ❑ **La verifica solo retrospettiva (a valle dello sviluppo) è spesso inadeguata**
 - Sviluppare tenendo sempre conto delle esigenze di verifica
 - Il costo di rilevazione e correzione di un errore è tanto maggiore quanto più avanzato è lo stadio di sviluppo

Laurea in Informatica, Università di Padova 22/30

Verifica e validazione: analisi statica

Programmi verificabili – 2

- ❑ **Eeguire cicli «revisione – verifica» dopo ogni rilevazione di errore nel codice è inefficace e troppo oneroso**
 - Approccio retrospettivo ⇒ *correctness by correction* 
- ❑ **Molto meglio effettuare analisi statiche durante la codifica**
 - Approccio costruttivo ⇒ *correctness by construction* 

Laurea in Informatica, Università di Padova 24/30



Verifica e validazione: analisi statica

Programmi verificabili – 3

- ❑ **Prediligere o proibire l'uso di particolari costrutti del linguaggio di programmazione**
 - Regole d'uso per assicurare comportamento predicibile
 - Regole d'uso per consentire analisi del sistema
 - Regole d'uso per facilitare le prove
 - Criteri di programmazione
 - Considerazioni pragmatiche

- ❑ **Norme di progetto**

Laurea in Informatica, Università di Padova25/30



Verifica e validazione: analisi statica

Analizzabilità del sistema

- ❑ **L'analisi statica costruisce modelli astratti del SW in esame**
 - Questi modelli rappresentano ogni programma come un grafo diretto e ne studiano i cammini possibili
 - Le transizioni tra stati (archi) hanno etichette che descrivono proprietà sintattiche o semantiche dell'istruzione corrispondente
 - La presenza di flussi di eccezione e di risoluzione dinamica di chiamata (*dispatching*) complica notevolmente la struttura del grafo
 - ❗ ○ Ciascun flusso di controllo (*thread*) viene rappresentato e analizzato separatamente

Laurea in Informatica, Università di Padova27/30



Verifica e validazione: analisi statica

Comportamento predicibile

- ❑ **Codice sorgente senza ambiguità**
 - **Effetti laterali (p.es. di funzioni)**
 - Diverse invocazioni della stessa funzione producono risultati diversi
 - **Ordine di elaborazione e inizializzazione**
 - L'esito di un programma dipende dall'ordine di elaborazione entro e tra unità
 - Per esempio, l'attivazione dei *thread* in Java è fonte di imprevedibilità
 - **Modalità di passaggio dei parametri**
 - La scelta di una modalità di passaggio (per valore, per riferimento) può influenzare l'esito dell'esecuzione

Laurea in Informatica, Università di Padova26/30



Verifica e validazione: analisi statica

Facilità di prova

- ❑ **Strategie di prova**
 - ❗ ○ **Investigativa: informale e senza obiettivi prefissati di copertura**
 - **Formale: regolata da norme e grado di copertura fissato**
- ❑ **Alcuni costrutti di linguaggio sono di ostacolo**
 - **La risoluzione dinamica di chiamata (*dispatching*) complica le prove di copertura**
 - Vedi esempio: Per approfondire #26
 - **La conversione forzata tra tipi (*casting*) complica l'analisi dell'identità dei dati**
 - **Le eccezioni predefinite complicano le prove di copertura**
 - Cammini di esecuzione difficili da raggiungere senza modificare il codice

Laurea in Informatica, Università di Padova28/30



Verifica e validazione: analisi statica

Criteri di programmazione

- ❑ **Riflettere l'architettura nel codice**
 - Programmazione strutturata per esprimere componenti, moduli, unità e facilitare il riuso
- ❑ **Separare le interfacce dall'implementazione**
 - Fissare bene le interfacce già a partire dall'architettura logica
- ❑ **Massimizzare l'incapsulazione (*information hiding*)**
 - Usare membri privati e metodi pubblici per l'accesso
- ❑ **Usare tipi specializzati per specificare dati**
 - La composizione e la specializzazione aumentano il potere espressivo del sistema di tipi del programma

Laurea in Informatica, Università di Padova29/30



Verifica e validazione: analisi statica

Considerazioni pragmatiche

- ❑ **L'efficacia dei metodi di analisi è funzione della qualità di strutturazione del codice**
 - Esempio: un modulo con un solo punto di ingresso e un solo punto di uscita è più facilmente analizzabile per il suo effetto sullo stato
- ❑ **La verifica di un programma relaciona frammenti di codice con frammenti di specifica**
 - La verificabilità è funzione inversa della dimensione del contesto
 - Conviene confinare gli ambiti (*scope*) e la visibilità
 - Una buona architettura facilita la verifica
 - Esempio: incapsulazione dello stato e controllo di accesso

Laurea in Informatica, Università di Padova30/30