



Verifica e validazione:
analisi statica

Anno accademico 2010/11
Ingegneria del Software mod. A

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

Corso di Laurea in Informatica, Università di Padova 1/29



Verifica e validazione: analisi statica

Premessa – 1

❑ Molti sistemi SW incorporano 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

Corso di Laurea in Informatica, Università di Padova 2/29



Verifica e validazione: analisi statica

Premessa – 2

❑ Il SW di tali sistemi deve esibire

- Le capacità funzionali specificate nei requisiti
 - Che determinano cosa il sistema deve fare
- Le caratteristiche non funzionali necessarie
 - Per garantire che il sistema lavora come previsto
- Proprietà e requisiti dimostrabili
 - Di costruzione
 - D'uso
 - Di funzionamento

Corso di Laurea in Informatica, Università di Padova 3/29



Verifica e validazione: analisi statica

Premessa – 3

❑ Nessun linguaggio di programmazione garantisce a priori che ogni programma scritto in esso sia verificabile

- La progettazione di un linguaggio di programmazione richiede di bilanciare tra funzionalità e integrità
 - Cioè tra potenza espressiva e onere di verifica
- ❑ Scelto un linguaggio occorre valutare a quali costrutti usare rispetto al loro impatto
 - Sulle caratteristiche di qualità e i costi di verifica

Corso di Laurea in Informatica, Università di Padova 4/29



Verifica e validazione: analisi statica

Tecniche di verifica – 1

❑ Tracciamento

- Dimostrare completezza ed economicità della soluzione
 - Soddisfacimento di tutti i requisiti
 - Nessuna funzionalità superflua o componente ingiustificato
- Ha luogo
 - Tra requisiti *software* e requisiti utente
 - Tra procedure di verifica e requisiti, disegno, codice
 - Tra codice sorgente e codice oggetto (!)

Corso di Laurea in Informatica, Università di Padova 5/29



Verifica e validazione: analisi statica

Tecniche di verifica – 2

❑ Tracciamento (segue)

- Particolari stili di codifica facilitano la verifica mediante tracciamento
 - Assegnare **singoli** requisiti di basso livello a **singoli** moduli del programma così da richiedere **una sola** procedura di prova e ottenere una più semplice corrispondenza tra essi
 - Maggiore l'astrazione di un costrutto del linguaggio maggiore la quantità di codice oggetto generato per esso e maggiore l'onere di dimostrazione di corrispondenza

Corso di Laurea in Informatica, Università di Padova 6/29

Verifica e validazione: analisi statica

Tecniche di verifica – 3

□ Revisioni

- **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 o informali**
 - Richiedono indipendenza tra verificato e verificatore
 - Richiedono semplicità e leggibilità di codice e diagrammi

Corso di Laurea in Informatica, Università di Padova 7/29

Verifica e validazione: analisi statica

Tecniche di verifica – 4

□ Analisi

- **Statica**
 - Applica a requisiti, progettazione, codice
- **Dinamica**
 - Applica a componenti del sistema o al sistema nella sua interezza

□ Le applicazioni critiche usano fino a 10 tipi diversi di analisi statica

Corso di Laurea in Informatica, Università di Padova 8/29

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



Corso di Laurea in Informatica, Università di Padova 9/29

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

Corso di Laurea in Informatica, Università di Padova 10/29

Verifica e validazione: analisi statica

2. Analisi di flusso dei dati

□ Accerta 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)

□ Rileva possibili anomalie

- Esempio: più scritture successive senza letture intermedie

□ È complicata dalla presenza e dall'uso di dati globali raggiungibili da ogni parte del programma

Corso di Laurea in Informatica, Università di Padova 11/29

Verifica e validazione: analisi statica

3. Analisi di flusso d'informazione

□ Determina 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

Corso di Laurea in Informatica, Università di Padova 12/29

Verifica e validazione: analisi statica

4. Esecuzione simbolica – 1

- **Verifica proprietà del programma mediante manipolazione algebrica del codice sorgente**
 - Combina tecniche di analisi di flusso di controllo, di flusso di dati e di flusso di informazione
- **Si esegue effettuando “sostituzioni inverse”**
 - A ogni LHS di un assegnamento sostituisce il suo RHS
 - LHS = parte sinistra
 - RHS = parte destra
- **Trasforma il flusso sequenziale del programma in un insieme di assegnamenti paralleli nei quali le uscite sono espresse come funzione degli ingressi**

Corso di Laurea in Informatica, Università di Padova

13/29

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
```

Corso di Laurea in Informatica, Università di Padova

14/29

Verifica e validazione: analisi statica

5. Verifica formale del codice

- **Prova la correttezza del codice sorgente rispetto alla specifica formale dei requisiti**
 - Esplora tutte le esecuzioni possibili
 - Non è fattibile mediante prove dinamiche
- **Correttezza parziale**
 - Elaborazione e prova di teoremi (condizioni di verifica) la cui verità implica che il verificarsi di certe pre-condizioni assicura il verificarsi di determinate post-condizioni
 - Sotto l'ipotesi di terminazione del programma
- **Correttezza totale**
 - Richiede prova di terminazione

Corso di Laurea in Informatica, Università di Padova

15/29

Verifica e validazione: analisi statica

6. Analisi di limite

- **Verifica che i dati del programma restino entro i limiti del loro tipo e della precisione desiderata**
 - Analisi di *overflow* nella rappresentazione di grandezze
 - Analisi di errori di arrotondamento
 - Verifica di valori di limite (*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**

Corso di Laurea in Informatica, Università di Padova

16/29

Verifica e validazione: analisi statica

7. Analisi d'uso di *stack*

- **Lo *stack* è l'area di memoria che i sottoprogrammi usano per immagazzinare dati locali, temporanei e indirizzi di ritorno generati dal compilatore**
 - Concettualmente uno *stack* per ogni flusso di controllo (*thread*)
- **Determina la massima domanda di *stack* richiesta da un'esecuzione e la relazione con la dimensione dell'area di memoria effettivamente assegnata**
- **Verifica che non vi possa essere collisione tra *stack* e *heap* per qualche esecuzione**
 - L'ampiezza dello *stack* cresce con l'annidamento ricorsivo di chiamate
 - L'ampiezza dell'*heap* è fissata a configurazione e poi consumata con la creazione dinamica di oggetti

Corso di Laurea in Informatica, Università di Padova

17/29

Verifica e validazione: analisi statica

8. Analisi temporale

- **Concerne le proprietà temporali richieste ed esibite dalle dipendenze delle uscite dagli ingressi del programma**
 - Produrre il valore giusto al momento giusto
- **Carenze o eccessi del linguaggio di programmazione e delle tecniche di codifica possono complicare questa analisi**
 - Esempi: iterazioni prive di limite statico, ricorso sistematico a strutture dati dinamiche

Corso di Laurea in Informatica, Università di Padova

18/29

Verifica e validazione: analisi statica

9. Analisi d'interferenza

- Mostra 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
 - Dispositivi programmabili con effetti a livello sistema (esempio: *watchdog*)

Corso di Laurea in Informatica, Università di Padova 19/29

Verifica e validazione: analisi statica

10. Analisi di codice oggetto

- Assicura 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

Corso di Laurea in Informatica, Università di Padova 20/29

Verifica e validazione: analisi statica

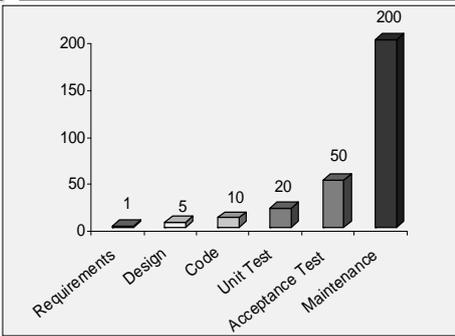
Programmi verificabili – 1

- L'adozione di standard di codifica e di sottoinsiemi del linguaggio appropriati discende dalla scelta dei metodi di verifica richiesti
 - L'uso di costrutti del linguaggio inadatti può compromettere la verificabilità del programma
- La verifica solo retrospettiva (a valle dello sviluppo) è spesso inadeguata
 - Non sviluppare senza tenere 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

Corso di Laurea in Informatica, Università di Padova 21/29

Verifica e validazione: analisi statica

Costo di correzione di errori



Stadio di Sviluppo	Costo di Correzione
Requirements	1
Design	5
Code	10
Unit Test	20
Acceptance Test	50
Maintenance	200

Corso di Laurea in Informatica, Università di Padova 22/29

Verifica e validazione: analisi statica

Programmi verificabili – 2

- Eseguire cicli revisione – verifica dopo ogni rilevazione di errore nel codice è inefficace e troppo oneroso
 - Approccio retrospettivo (*correct by correction*)
- Convienne piuttosto effettuare analisi statiche durante la fase di codifica
 - Approccio costruttivo (*correct by construction*)
 - Produce miglior rapporto costo/qualità a fronte di maggior costo progettuale

Corso di Laurea in Informatica, Università di Padova 23/29

Verifica e validazione: analisi statica

Programmi verificabili – 3

- Richiedere 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

Corso di Laurea in Informatica, Università di Padova 24/29

 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
 - **Modalità di passaggio dei parametri**
 - La scelta di una modalità di passaggio (per copia, per riferimento) può influenzare l'esito dell'esecuzione

Corso di Laurea in Informatica, Università di Padova 25/29

 Verifica e validazione: analisi statica

Analizzabilità del sistema

- ❑ **La verifica statica richiede la costruzione di modelli del sistema da analizzare a partire dal codice del programma**
- ❑ **Rappresentano il programma come un grafo diretto e ne studiano i cammini possibili**
 - **Storie di esecuzione**
 - **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**

Corso di Laurea in Informatica, Università di Padova 26/29

 Verifica e validazione: analisi statica

Facilità di prova

- ❑ **Strategie di prova**
 - **Investigativa: informale e senza obiettivi prefissati di copertura**
 - **Formale: regolata da norme specifiche e con 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 #27
 - **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

Corso di Laurea in Informatica, Università di Padova 27/29

 Verifica e validazione: analisi statica

Criteri di programmazione

- ❑ **Riflettere nel codice l'architettura del sistema (*enforce intentions*)**
 - Usare programmazione strutturata per rappresentare componenti, moduli e unità
 - Usare costrutti di programmazione che facilitano il riuso
- ❑ **Mantenere le interfacce separate dall'implementazione**
 - L'interfaccia si definisce già nell'architettura logica
- ❑ **Massimizzare l'incapsulazione (*information hiding*)**
 - Usare membri privati e metodi esterni di accesso
- ❑ **Usare tipi ben definiti per specificare dati**
 - La composizione e la specializzazione aumentano la potenza espressiva del sistema di tipi del programma

Corso di Laurea in Informatica, Università di Padova 28/29

 Verifica e validazione: analisi statica

Considerazioni pragmatiche

- ❑ **L'efficacia dei metodi di analisi è funzione della qualità di strutturazione del codice**
 - **Esempio: ogni modulo deve avere un solo punto di ingresso e un solo punto di uscita**
- ❑ **La verifica di un programma relaziona frammenti di codice con frammenti di specifica**
 - **La verificabilità è funzione della dimensione del contesto**
 - Controllare e limitare gli ambiti (*scope*) e la visibilità
 - **Una buona architettura facilita la verifica**
 - Esempio: incapsulazione dello stato e controllo di accesso

Corso di Laurea in Informatica, Università di Padova 29/29