



## Verifica e validazione: analisi dinamica


Anno accademico 2016/17  
Ingegneria del Software

Tullio Vardanega, [tullio.vardanega@math.unipd.it](mailto:tullio.vardanega@math.unipd.it)

IS

Laurea in Informatica, Università di Padova

1/42




Verifica e validazione: analisi dinamica

## Caratterizzazione

- ❑ Parte essenziale del processo di verifica
- ❑ Produce una misura della qualità del sistema
  - Aumenta il valore di qualità del sistema identificandone e rimuovendone difetti
- ❑ Il suo inizio non deve essere differito al termine delle attività di codifica
- ❑ Le sue esigenze devono essere tenute in conto nella progettazione del sistema

Laurea in Informatica, Università di Padova

3/42




Verifica e validazione: analisi dinamica

## Definizione

- ❑ **Analisi dinamica = *test* (prova)**
  - Attività che comporta esecuzione dell'oggetto di verifica
- ❑ **Verifica dinamica del comportamento del programma su un insieme finito di casi**
  - Selezionati nel dominio (in generale, infinito) di tutte le esecuzioni possibili
  - Ciascun caso di prova specifica i valori di ingresso e lo stato iniziale del sistema
  - Ciascun caso di prova deve produrre un esito decidibile (oracolo) verificato rispetto a un comportamento atteso

Laurea in Informatica, Università di Padova

2/42



Verifica e validazione: analisi dinamica

## Classificazione delle problematiche

Prove software

Laurea in Informatica, Università di Padova

4/42

Verifica e validazione: analisi dinamica

## Dentro la classificazione – 1

- ❑ **Terminologia**
  - *Fault* → *Error* → *Failure*
  - Guasto (difetto) → Errore → Malfunzionamento
- ❑ **Fondamenti teorici**
  - Decidibilità, testabilità, criteri
- ❑ **Oggetto delle prove**
  - Unità, aggregati, sistema completo

Laurea in Informatica, Università di Padova 5/42

Verifica e validazione: analisi dinamica

## Visione gerarchica

Laurea in Informatica, Università di Padova 7/42

Verifica e validazione: analisi dinamica

## Terminologia

- ❑ ***A failure occurs when the behavior of a system deviates from what is specified for it***
  - *Failures result from problems internal to the system which eventually manifest in the system's external behavior*
- ❑ ***These problems are called errors and their mechanical or algorithmic or conceptual cause are termed faults***
  - *Errors are states of the system*
  - *Faults are what causes the error to exist*
- ❑ ***Systems are hierarchical compositions of components which are themselves systems***


Laurea in Informatica, Università di Padova 6/42

Verifica e validazione: analisi dinamica

## Dentro la classificazione – 2


- ❑ **Obiettivo delle prove**
  - Accettazione, qualifica, conformità, regressione
  - Installazione nell'ambiente di prova, controllo prestazioni
- ❑ **Vincoli di progetto**
  - Processi, prodotti, risorse, competenze
  - Stima e controllo dei costi, criteri di terminazione
- ❑ **Attività di prova**
  - Pianificazione, specifica e sviluppo dei casi di prova
  - Esecuzione, valutazione, trattamento dei problemi

Laurea in Informatica, Università di Padova 8/42


 Verifica e validazione: analisi dinamica

## Fattori da bilanciare

- ❑ **La strategia di prova richiede di bilanciare**
  - La quantità **minima** di casi di prova sufficienti a fornire certezza adeguate sulla qualità del prodotto
    - Fattore governato da criteri tecnici
  - La quantità **massima** di sforzo, tempo e risorse disponibile per la verifica
    - Fattore governato da criteri gestionali
- ❑ **Legge del rendimento decrescente**
  - *Diminishing returns*




Laurea in Informatica, Università di Padova 9/42


 Verifica e validazione: analisi dinamica

## Criteri guida – 2

- ❑ **Una visione riduttiva ma efficace**
  - Prove: “il processo di eseguire un programma con l’intento di trovarvi difetti”  
*The Art of Software Testing*, G.J. Myers, Wiley-Interscience, 1979
- ❑ **La “provabilità” del SW va assicurata a monte dello sviluppo, non a valle della codifica**
  - Progettazione architettonica e di dettaglio raffinate per assicurare provabilità
  - La complessità è nemica della provabilità: ne riparleremo!



Laurea in Informatica, Università di Padova 11/42

 Verifica e validazione: analisi dinamica

## Criteri guida – 1


- ❑ **Oggetto della prova**
  - Il sistema nel suo complesso (TS)
  - Parti di esso, in relazione funzionale, d’uso, di comportamento, di struttura (TI)
  - Singole unità (TU)
- ❑ **Obiettivo della prova**
  - Specificato per ogni caso di prova
  - In termini precisi e quantitativi
  - Varia al variare dell’oggetto della prova
  - Il PdQ risponde alla domanda: quali e quante prove

Laurea in Informatica, Università di Padova 10/42


 Verifica e validazione: analisi dinamica

## Criteri guida – 3

- ❑ **Una singola prova non basta**
  - I risultati valgono solo per quella esecuzione
    - Non possono essere generalizzati
  - La prova deve essere ripetibile
  - Rileva malfunzionamenti indicando la presenza di guasti
    - In generale, non può provarne l’assenza!
- ❑ **Le prove sono costose**
  - Richiedono molte risorse (tempo, persone, infrastrutture)
  - Necessitano di un processo definito
  - Richiedono attività di ricerca, analisi, correzione



Laurea in Informatica, Università di Padova 12/42




Verifica e validazione: analisi dinamica

## Limiti e problemi

- ❑ **Teorema di Howden (1975)**
  - **Non esiste** un algoritmo che, dato un programma P, generi per esso un *test* finito ideale (definito da criteri affidabili e validi)
- ❑ **Tesi di Dijkstra (1969)**
  - Il *test* di un programma può rilevare la presenza di malfunzionamenti, ma **non può** dimostrarne l'assenza
- ❑ **Teorema di Weyuker (1979)**
  - Dato un programma P, i seguenti problemi sono **indecidibili**:
    - $\exists$  un dato di ingresso che causi l'esecuzione di un particolare comando di P?
    - $\exists$  un dato di ingresso che causi l'esecuzione di una particolare condizione di P?
    - È possibile trovare un dato di ingresso che causi l'esecuzione di ogni comando / condizione / cammino di P?

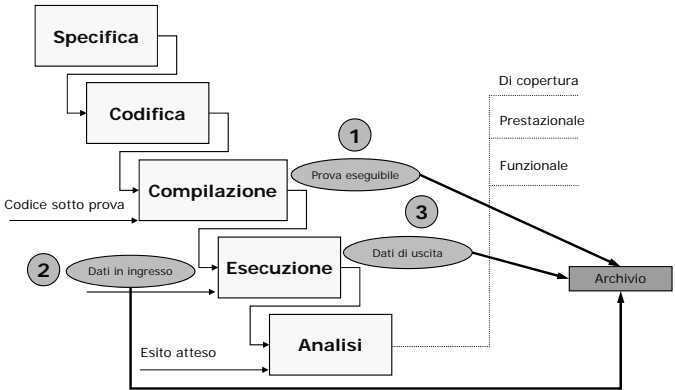
Laurea in Informatica, Università di Padova

**13/42**




Verifica e validazione: analisi dinamica

## Attività di prova



Laurea in Informatica, Università di Padova

**15/42**



Verifica e validazione: analisi dinamica


## Principi del *testing software*

Cf. per approfondire #21

- ❑ **Secondo Bertrand Meyer**
  - *To test a program is to try to make it fail*
  - *Tests are no substitutes for specifications*
  - *Any failed execution must yield a test case, to be permanently included in the project's test suite*
  - *Oracles should be part of the program text, as contracts*
  - *Any testing strategy should include a reproducible testing process and be evaluated objectively with explicit criteria*
  - *A testing strategy's most important quality is the number of faults it uncovers as a function of time*

Laurea in Informatica, Università di Padova

**14/42**




Verifica e validazione: analisi dinamica

## Gli elementi di una prova – 1

- ❑ **Caso di prova (*test case*)**
  - **Tripla <ingresso, uscita, ambiente>**
    - L'ambiente include l'oggetto della prova
- ❑ **Batteria di prove (*test suite*)**
  - **Insieme (sequenza) di casi di prova**
- ❑ **Procedura di prova**
  - **Il procedimento (automatico o manuale) per eseguire, registrare, analizzare e valutare i risultati di prove**
- ❑ **Prova = <Procedura, Batteria di prove>**

Laurea in Informatica, Università di Padova

**16/42**



Verifica e validazione: analisi dinamica

## Gli elementi di una prova – 2

**□ L'oracolo**


- Un metodo per generare a priori i risultati attesi e per convalidare i risultati ottenuti
- Generalmente applicato da agenti automatici
  - Per velocizzare la convalida e renderla "oggettiva"

**□ Come produrre oracoli**

- Sulla base delle specifiche funzionali
- Sulla semplificazione delle prove
- Sull'uso di componenti terze indipendenti

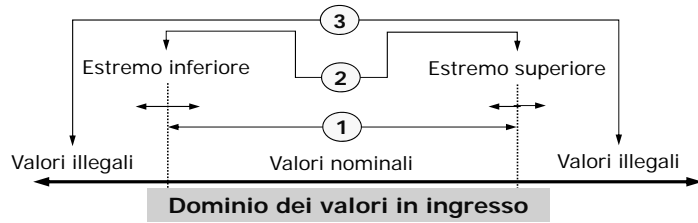
Laurea in Informatica, Università di Padova

17/42



Verifica e validazione: analisi dinamica


## Classi di equivalenza



- 3 classi di equivalenza
  - Valori nominali interni al dominio (1)
  - Valori legali di limite (2)
  - Valori illegali (3)

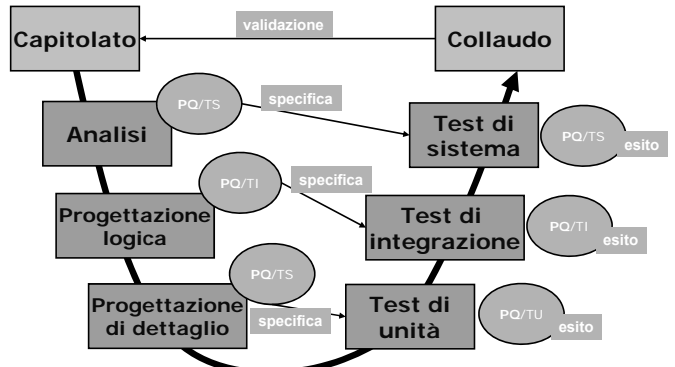
Laurea in Informatica, Università di Padova

19/42




Verifica e validazione: analisi dinamica

## Esecuzione delle attività di prova



Laurea in Informatica, Università di Padova

18/42



Verifica e validazione: analisi dinamica

## Test di unità – 1

**□ Unità software composta da uno o più moduli**

- Modulo = componente elementare di progetto di dettaglio (quindi non necessita di *test* di unità)

**□ Unità e moduli sono determinati in progettazione di dettaglio e quindi anche il piano di TU**

**□ La TU completa quando ha verificato tutte le unità**

**□ ~2/3 dei difetti identificati tramite analisi dinamica vengono rilevati in TU**

- 50% di essi viene identificato da prove strutturali (*white-box*)

Laurea in Informatica, Università di Padova

20/42

Verifica e validazione: analisi dinamica

## Copertura – 1

- ❑ Per ogni test di unità si definiscono
  - Oggetto, strategia, risorse necessarie, piano di esecuzione
- ❑ Si ha **Statement Coverage** al 100%
  - Quando i *test* effettuati sull'unità sono sufficienti a eseguire – complessivamente – almeno una volta tutte le linee di comando di ciascuno dei moduli dell'unità
- ❑ Si ha **Branch Coverage** al 100%
  - Quando ciascun ramo del flusso di controllo viene attraversato – complessivamente – almeno una volta

Laurea in Informatica, Università di Padova

21/42

Verifica e validazione: analisi dinamica

## Copertura – 3

- ❑ **Definizione (DO-178B)**
  - **Condizione**
    - Espressione booleana semplice non contenente al suo interno ulteriori condizioni combinate da operatori booleani
  - **Decisione**
    - Espressione composta contenente condizioni combinate da operatori booleani

Laurea in Informatica, Università di Padova

23/42

Verifica e validazione: analisi dinamica

## Copertura – 2

**Lo *statement coverage* è meno potente del *branch coverage***

```
A1.1;
if (Cond)
{A1.2;
A2};
A3;
```

Poniamo che A1.2 sia stato posto per errore nel ramo condizionale, quando invece dovrebbe essere sempre eseguito incondizionatamente insieme ad A.1.1

La strategia di *statement coverage* può non rilevare questo errore (che può produrre un *failure* funzionale) perché è solo interessata ad attraversare complessivamente quante più linee di comando possibile

La strategia di *branch coverage* (che è funzionale, per cammino) invece lo rileva, eseguendo la prova dove "Cond" vale Falso e A1.2 non viene eseguito

Important Notice

Laurea in Informatica, Università di Padova

22/42

Verifica e validazione: analisi dinamica

## Copertura – 4

- ❑ Per il frammento di codice a lato, massimizzare **statement coverage** e **branch coverage** non assicura il **test** di tutte le combinazioni T/F delle singole condizioni
- ❑ Una batteria di **test** potrebbe per esempio esercitare soltanto i casi {x=2, y=0} e {x=2, y=1}
- ❑ Ma con un programma equivalente (vedi a lato), quella batteria di **test** raggiungerebbe **statement** e **branch coverage** solo dei 2/3
- ❑ I casi di **test** {x=2, y=0}, {x=0, y=0}, {x=2, y=1} permettono invece di garantire **condition coverage** al 100%


```
if (x>1 && y==0)
{comando1}
else
{comando2}
```

```
if (x>1)
if (y==0) {comando1}
else {comando2}
else {comando2}
```

**Il *branch coverage* può non comprendere appieno la logica del programma**

Laurea in Informatica, Università di Padova

24/42




Verifica e validazione: analisi dinamica

**Copertura – 5**


  

- ❑ **Al crescere del numero di condizioni all'interno di una decisione il numero di *test* necessario a massimizzare il *condition coverage* diventa proibitivo**
- ❑ **DO-178B richiede allora di massimizzare il *modified decision condition coverage***
  - MCDC implica *branch coverage*



Laurea in Informatica, Università di Padova

25/42



Verifica e validazione: analisi dinamica

**Copertura – 7**

if A=B and (C or D>3) then ...

Caso	Condizione/Variabile			Esito
	A=B	C	D>3	
1	*	F	F	F
2	T	T	*	T
3	T	*	T	T
4	F	*	*	F

Laurea in Informatica, Università di Padova

27/42



Verifica e validazione: analisi dinamica


**Copertura – 6**

- ❑ **Requisiti DO-178B (*software* di avionica)**
  - Che tutte le decisioni siano soggette a *test* e tutti i loro possibili esiti siano effettivamente prodotti
  - Che ciascuna condizione all'interno di una decisione assuma entrambi gli esiti (vero/falso) almeno una volta
- ❑ **Occorre allora verificare se e come ogni singola condizione possa da sola determinare tutti gli esiti possibili della decisione**
- ❑ **Non basta più provare l'intera decisione come vera o falsa!**

Laurea in Informatica, Università di Padova

26/42



Verifica e validazione: analisi dinamica


**Test di unità – 2**

- ❑ **Test funzionale (*black-box*)**
  - Da solo non può accertare correttezza e completezza della logica interna dell'unità
    - Va necessariamente integrato con *test* strutturale
  - Fa riferimento alla specifica dell'unità e utilizza dati di ingresso capaci di provocare l'esito atteso
  - Ciascun insieme di dati di ingresso che producono un dato comportamento funzionale costituisce un caso di prova
  - Classi di equivalenza invece che infiniti valori di ingresso
    - Valori nella medesima classe producono lo stesso comportamento

Laurea in Informatica, Università di Padova

28/42




Verifica e validazione: analisi dinamica

## Test di unità – 3

- ❑ **Test strutturale (*white-box*)**
  - Verifica la **logica interna** del codice dell'unità cercando massima copertura
  - Ciascuna prova deve essere progettato per attivare **ogni cammino** di esecuzione all'interno del modulo
  - Ciascun insieme di dati di ingresso che **attivano un percorso** costituiscono un caso di prova
  - L'uso di *debugger* può agevolarne l'esecuzione ma non esonera dalla progettazione dei casi di prova

Laurea in Informatica, Università di Padova29/42



Verifica e validazione: analisi dinamica

## Integrazione incrementale

- ❑ **Bottom-up**
  - Si sviluppano e si integrano prima le parti con minore dipendenza funzionale e maggiore utilità
  - Poi si risale l'albero delle dipendenze
  - Questa strategia riduce il numero di *stub* necessari al *test* ma porta più tardi alla disponibilità di funzionalità di alto livello
- ❑ **Top-down**
  - Si sviluppano prima le parti più esterne, quelle poste sulle foglie dell'albero delle dipendenze e poi si scende
  - Questa strategia comporta l'uso di molti *stub* ma integra a partire dalla funzionalità di più alto livello

Laurea in Informatica, Università di Padova31/42



Verifica e validazione: analisi dinamica

## Strategie di integrazione

- ❑ **Assemblare parti in modo incrementale**
  - Aggiungendo solo a insiemi ben verificati, i difetti rilevati in un *test* di integrazione sono più probabilmente da attribuirsi alla parte ultima aggiunta
- ❑ **Assemblare produttori prima dei consumatori**
  - La verifica dei primi fornisce ai secondi flusso di controllo (chiamate) e flusso dei dati corretti
- ❑ **Assemblare in modo che ogni passo di integrazione sia reversibile**
  - Consente di retrocedere verso uno stato noto e sicuro

Laurea in Informatica, Università di Padova30/42




Verifica e validazione: analisi dinamica

## Test di integrazione – 1

- ❑ **Si applica alle componenti specificate nella progettazione architeturale**
  - La loro integrazione costituisce il sistema completo
- ❑ **Logica di integrazione funzionale**
  - Seleziona le funzionalità da integrare
  - Identifica le componenti che svolgono quelle funzionalità
  - Ordina le componenti per numero di dipendenze crescente
    - Dipendenze nel flusso di controllo (chiamata) e nel flusso di dati
  - Esegue l'integrazione in quell'ordine

Laurea in Informatica, Università di Padova32/42






Verifica e validazione: analisi dinamica

## Test di integrazione – 2

- ❑ **Problemi rilevati durante TI**
  - Manifestano difetti di progettazione o insufficiente qualità nei *test* di unità
  
- ❑ **TI ha tanti *test* quanto ne servono per**
  - Accertare che tutti i dati scambiati attraverso ciascun interfaccia siano conformi alla loro specifica
  - Accertare che tutti i flussi di controllo previsti in specifica siano stati effettivamente realizzati e provati


Laurea in Informatica, Università di Padova33/42



Verifica e validazione: analisi dinamica

## Altri tipi di *test*

- ❑ **Test di regressione**
  - Ripetizione selettiva di TI (e poi di TS)
    - Integrando solo parti che abbiano precedentemente superato TU
    - Nel *repository* di progetto devono stare solo unità SW di questo tipo
  - Per accertare che modifiche intervenute per correzione o estensione di parti non comportino errori nel sistema
    - Può essere molto oneroso
  - I contenuti del *test* di regressione vanno decisi nel momento in cui si approvano modifiche al SW
  
- ❑ **Test di accettazione (collaudo)**
  - Accerta il soddisfacimento dei requisiti utente



Laurea in Informatica, Università di Padova35/42



Verifica e validazione: analisi dinamica

## Test di sistema

- ❑ **Verifica il comportamento dinamico del sistema completo rispetto ai requisiti SW**
  
- ❑ **Ha inizio con il completamento del *test* di integrazione**
  
- ❑ **È inerentemente funzionale (*black-box*)**
  - Non dovrebbe richiedere conoscenza della logica interna del *software*

Laurea in Informatica, Università di Padova34/42



Verifica e validazione: analisi dinamica

## Fattore di copertura


- ❑ **Quanto la prova esercita il prodotto**
  - Copertura funzionale
    - Rispetto alla percentuale di funzionalità esercitate come viste dall'esterno
  - Copertura strutturale (*branch, condition*)
    - Rispetto alla percentuale di logica interna del codice esercitata
  
- ❑ **Una misura della bontà di una prova**
  - Copertura del 100% non prova assenza di difetti
  - Il 100% di copertura può essere irraggiungibile
    - Costi eccessivi, codice sorgente non disponibile, codice irraggiungibile non eliminabile, copertura esaustiva dei cicli

Laurea in Informatica, Università di Padova36/42

Verifica e validazione: analisi dinamica

### Maturità di prodotto

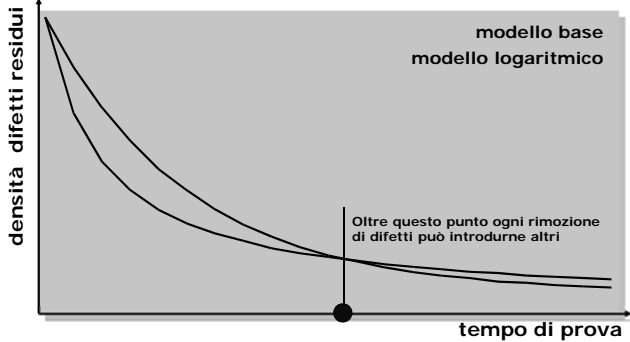
- ❑ **Valutare il grado di evoluzione del prodotto**
  - Quanto il prodotto migliora in seguito alle prove
  - Quanto diminuisce la densità dei difetti
  - Quanto può costare la scoperta del prossimo difetto
- ❑ **Le tecniche correnti sono spesso empiriche**
  - Sotto l'influenza del modello *code-and-fix*
- ❑ **Definire un modello ideale**
  - **Modello base:** il numero di difetti del *SW* è una costante iniziale
  - **Modello logaritmico:** le modifiche introducono difetti



Laurea in Informatica, Università di Padova 37/42

Verifica e validazione: analisi dinamica

### La funzione $\lambda(t)$



densità difetti residui

tempo di prova

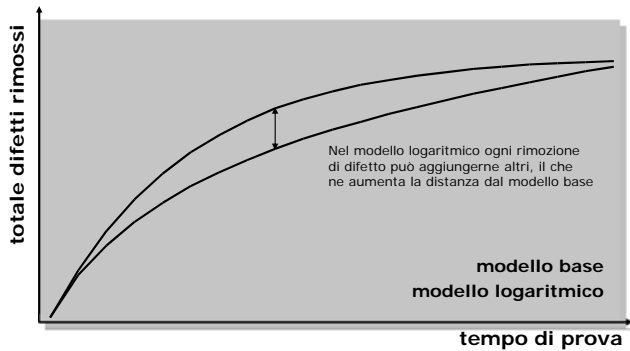
modello base  
modello logaritmico

Oltre questo punto ogni rimozione di difetti può introdurne altri

Laurea in Informatica, Università di Padova 39/42

Verifica e validazione: analisi dinamica

### La funzione $\mu(t)$



totale difetti rimossi

tempo di prova

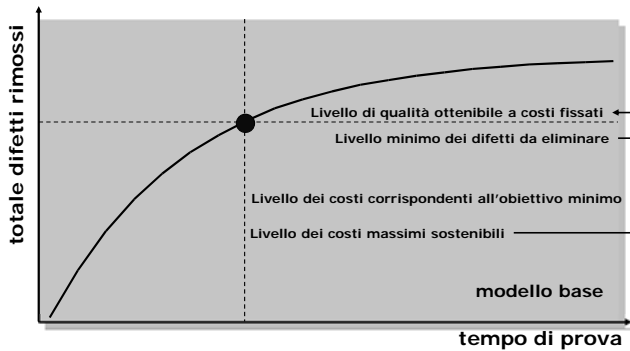
modello base  
modello logaritmico

Nel modello logaritmico ogni rimozione di difetto può aggiungerne altri, il che ne aumenta la distanza dal modello base

Laurea in Informatica, Università di Padova 38/42

Verifica e validazione: analisi dinamica

### Criteri economici e qualitativi



totale difetti rimossi

tempo di prova

modello base


Livello di qualità ottenibile a costi fissati

Livello minimo dei difetti da eliminare

Livello dei costi corrispondenti all'obiettivo minimo

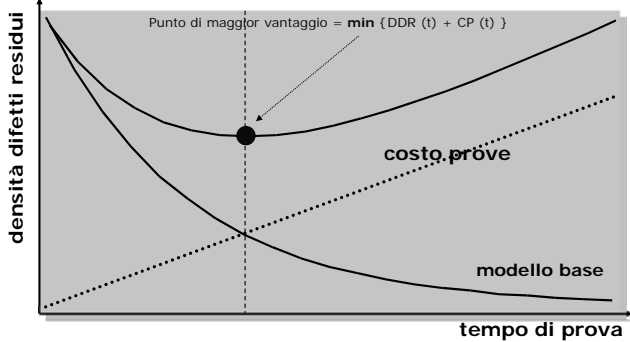
Livello dei costi massimi sostenibili

Laurea in Informatica, Università di Padova 40/42



Verifica e validazione: analisi dinamica

## Criteri analitici



Punto di maggior vantaggio =  $\min \{ DDR (t) + CP (t) \}$


densità difetti residui

tempo di prova

costo prove

modello base

Laurea in Informatica, Università di Padova41/42



Verifica e validazione: analisi dinamica

## Bibliografia

- ❑ **J.D. Musa, A.F. Ackerman**  
**Quantifying software validation: when to stop testing?**  
**IEEE Software, maggio 1989**
  
- ❑ **B. Meyer**  
**Seven Principles of Software Testing**  
**IEEE Computer, agosto 2008**  
**(cf. per approfondire #21)**

Laurea in Informatica, Università di Padova42/42