



Università degli Studi di Udine

Verifica e Validazione

prof. Maurizio Pighin

Dipartimento di Matematica e Informatica



Verification vs validation

Ingegneria del Software
Progettazione Laboratorio
Verifica e Validazione
Maurizio Pighin

- Verification and Validation: Assuring that a software system meets a user's needs
- Verification:
 - "Are we building the product right"
 - *The software should conform to its specification*
- Validation:
 - "Are we building the right product"
 - *The software should do what the user really requires*

Slide 2





Verifica e Validazione - Risultati Negativi

Ingegneria del Software
Progettazione Laboratorio
Verifica e Validazione
Maurizio Pighin

- Teor. equivalenza programmi
 - *Non è possibile stabilire se due programmi calcolino la stessa funzione o meno*
- Teor. equivalenza cammini
 - *Non esiste un algoritmo in grado di stabilire se due generici cammini del grafo di flusso di due programmi calcolino o meno la stessa funzione*
- Teor. di Weyuker
 - *Dato un generico programma P i seguenti problemi risultano indecidibili*
 - esiste almeno un dato di ingresso che causa l'esecuzione di un particolare comando
 - esiste almeno un dato di ingresso che causa l'esecuzione di un particolare cammino (branch)

Slide 3



Risultati Negativi

Ingegneria del Software
Progettazione Laboratorio
Verifica e Validazione
Maurizio Pighin

- è possibile trovare almeno un dato di ingresso che causa l'esecuzione di ogni comando di P
- è possibile trovare almeno un dato di ingresso che causa l'esecuzione di ogni condizione (branch) di P
- Corollari teor. Rice
 - *Non è decidibile se l'esecuzione di un dato programma termina*
 - *Non è decidibile verificare che un dato programma, con un certo ingresso, fornisca l'uscita desiderata*
- Tesi di Dijkstra
 - *Il test di un programma può rivelare la presenza di malfunzionamenti, ma mai dimostrarne l'assenza*

Slide 4





Defect testing

Ingegneria del Software
Progettazione Laboratorio
Verifica e Validazione
Maurizio Pighin

- Testing programs to establish the presence of system defects
- The goal of defect testing is to discover defects in programs
- A successful defect test is a test which causes a program to behave in an anomalous way
- Tests show the presence not the absence of defects

Slide 5



Tipologie di Approccio

Ingegneria del Software
Progettazione Laboratorio
Verifica e Validazione
Maurizio Pighin

- Test Dinamico
 - *Approccio Generale*
 - Black-Box (funzionale)
 - White-Box (strutturale)
 - Interfacce - richiami
 - Stress - richiami
 - Altri (sicurezza, robustezza, ecc.)
 - *Metodologie di lavoro (catene di test, la regressione)*
 - *Automazione test e test mutazionali*
 - *Risk-Based test*
- Test Statico
 - *Ispezione*
 - *Strumenti automatici*
 - *Metodi formali*
 - *Metodi Statistici*

Slide 6





Interface testing

Ingegneria del Software
Progettazione Laboratorio
Verifica e Validazione
Maurizio Pighin

- Takes place when modules or sub-systems are integrated to create larger systems
- Objectives are to detect faults due to interface errors or invalid assumptions about interfaces
- Particularly important for object-oriented development as objects are defined by their interfaces

Slide 7



Interfaces types

Ingegneria del Software
Progettazione Laboratorio
Verifica e Validazione
Maurizio Pighin

- Parameter interfaces
 - *Data passed from one procedure to another*
- Shared memory interfaces
 - *Block of memory is shared between procedures*
- Procedural interfaces
 - *Sub-system encapsulates a set of procedures to be called by other sub-systems*
- Message passing interfaces
 - *Sub-systems request services from other sub-systems*

Slide 8





Interface errors

Ingegneria del Software
Progettazione Laboratorio
Verifica e Validazione
Maurizio Pighin

- Interface misuse
 - *A calling component calls another component and makes an error in its use of its interface e.g. parameters in the wrong order*
- Interface misunderstanding
 - *A calling component embeds assumptions about the behaviour of the called component which are incorrect*
- Timing errors
 - *The called and the calling component operate at different speeds and out-of-date information is accessed*

Slide 9



Interface testing guidelines

Ingegneria del Software
Progettazione Laboratorio
Verifica e Validazione
Maurizio Pighin

- Design tests so that parameters to a called procedure are at the extreme ends of their ranges
- Always test pointer parameters with null pointers
- Design tests which cause the component to fail
- Use stress testing in message passing systems
- In shared memory systems, vary the order in which components are activated

Slide 10





Stress testing

Ingegneria del Software
Progettazione Laboratorio
Verifica e Validazione
Maurizio Pighin

- Exercises the system beyond its maximum design load. Stressing the system often causes defects to come to light
- Stressing the system test failure behaviour. Systems should not fail catastrophically. Stress testing checks for unacceptable loss of service or data
- Particularly relevant to distributed systems which can exhibit severe degradation as a network becomes overloaded

Slide 11



Altri test di Sistema

Ingegneria del Software
Progettazione Laboratorio
Verifica e Validazione
Maurizio Pighin

- Di sicurezza
 - *ad esempio controllo del numero di accessi*
- Di robustezza
 - *comportamento di fronte a situazioni impreviste generiche*
 - ad esempio mancanza di corrente
- Di configurazione
 - *verifica rispetto a tutte le configurazioni previste*
 - ad esempio su più piattaforme

Slide 12





Catene di test

Ingegneria del Software
Progettazione Laboratorio
Verifica e Validazione
Maurizio Pighin

- Metodologia organizzativa che permette ad ogni livello di avere visibilità della sequenza di test attuati
- Indispensabili per attuare correttamente test e per sviluppare il meccanismo di regressione
- Fasi organizzative
 - *Definizione test set*
 - N.catena e N. test
 - Modulo
 - Meccanismi di attivazione
 - Dati Input
 - Output atteso
 - Eventuali risorse necessarie

Slide 13



Catene di test

Ingegneria del Software
Progettazione Laboratorio
Verifica e Validazione
Maurizio Pighin

- *Pianificazione*
 - Chi fa il test
 - Quali test vengono eseguiti (quali catene/sotto-catene)
 - Quando i test vengono eseguiti
- *Esecuzione*
 - Il test viene eseguito
 - Viene consuntivato
 - *Chi esegue*
 - *Quando esegue*
 - *Tempo impiegato (singolo test o almeno catena)*
 - *Comportamento riscontrato*
 - » *corretto*
 - » *tipo di errore riscontrato*
 - *Eventuali note esplicative*
- NON si fanno ipotesi sul “perché”

Slide 14





Test di regressione

Ingegneria del Software
Progettazione Laboratorio
Verifica e Validazione
Maurizio Pighin

- Permettono di verificare che le azioni correttive non abbiano introdotto altri errori (side-effect indesiderati)
- In caso di errore le azioni sono
 - *Sospendere attività al livello raggiunto*
 - *Riprendere attività ai livelli precedenti*
 - *Localizzare errori*
 - *Far correggere*
 - *Ripetere i test ai livelli precedenti*
 - *Riprendere l'attività al livello raggiunto*

Slide 15



Automazione del testing

Ingegneria del Software
Progettazione Laboratorio
Verifica e Validazione
Maurizio Pighin

- Motivazioni ed obbiettivi
 - *assicurare un n° elevato di test*
 - *contenere i costi*
 - diminuire il tempo
 - diminuire le persone impegnate
- Ambiti d'applicazione
 - *pianificazione automatica (50%)*
 - *analisi dinamica del codice (99%)*
 - *scelta automatica dei dati (50%)*
 - *esecuzione/riesecuzione automatica dei test (95%)*

Slide 16





Automazione del testing

Ingegneria del Software
Progettazione Laboratorio
Verifica e Validazione
Maurizio Pighin

- Pianificazione
 - *raccolta di informazioni*
 - *relazioni fra richieste e test-case*
 - *registrazione test-case*
- Analisi dinamica
 - *analisi copertura, misura della porzione di codice sottoposta a test*
 - *codici autovalidanti: verifica asserzioni eseguibili*
- Scelta automatica dei dati
 - *estremi*
 - *intermezzo*

Slide 17



Automazione del testing Mutazione

Ingegneria del Software
Progettazione Laboratorio
Verifica e Validazione
Maurizio Pighin

- Esecuzione/riesecuzione
 - *definizione di test driver*
 - *Definizione di monitor che registrano l'input ed il relativo output*
 - *come effetto collaterale i test di mutazione*
- Test di mutazione
 - *verificano la qualità dei test effettuati*
 - *viene modificato P in P^* (mutazione)*
 - *vengono verificate le catene di test che danno esito positivo per P ma non per P^**

Slide 18





Risk-Based testing

Ingegneria del Software
Progettazione Laboratorio
Verifica e Validazione
Maurizio Pighin

- Risk Base Testing
 - ICSTEST – 2005 (Dusseldorf)
 - Hans Schaefer

Slide 19



Risk-Based testing

Ingegneria del Software
Progettazione Laboratorio
Verifica e Validazione
Maurizio Pighin

- Compito del Testing
 - *Compito dei manager è di prendere decisioni e compito degli ingegneri informarli*
- Problematiche
 - *Il test è sempre sotto pressione*
 - *E' l'ultima cosa che si fa in un progetto*
 - *Bisogna saper tagliare le cose meno importanti*


Slide 20





Risk-Based testing

Ingegneria del Software
Progettazione Laboratorio
Verifica e Validazione
Maurizio Pighin

- Strategia
 - *Obiettivo*
 - Trovare i difetti più importanti il prima possibile ed al minor costo
 - *Nessun rischio*  *Nessun Test*
 - *Decisione basata su*
 - Business
 - Utente
 - Cliente

Slide 21



Risk-Based testing

Ingegneria del Software
Progettazione Laboratorio
Verifica e Validazione
Maurizio Pighin

- Cosa è “rischio”
 - *Probabilità che qualche cosa di dannoso accada moltiplicata per il costo (danno) delle conseguenze*
- Probabilità di failure
 - *Qualità*
 - Failure rate
 - Defect number
 - *Volume funzionale*
 - FP
- Probabilità di failure = failure/volume

Slide 22





Risk-Based testing

Ingegneria del Software
Progettazione Laboratorio
Verifica e Validazione
Maurizio Pighin

- Danno
 - *Catastrofico (perdita vite, perdita licenza)*
 - *Finanziario (perdita di clienti, perdita di fiducia dei clienti, danno all'identità dell'azienda)*
 - *Impatto su altre funzioni o sistemi*
 - *Tempo per scoprire e riparare i problemi*
- $\text{Rischio} = \text{Danno} * \text{Probabilità_failure}$

Slide 23



Risk-Based testing

Ingegneria del Software
Progettazione Laboratorio
Verifica e Validazione
Maurizio Pighin

- Gestione del rischio
 - *Minimizzare le perdite*
 - Prevenire le perdite
 - *Applicare Metodologie di Quality Assurance e Quality Management*
 - Combattere le perdite
 - *Fare Revisioni, Test, Validazioni*
 - *Ignorare le perdite e pagarle*
 - Trattenere le perdite
 - *Reagire tardi, pagare il danno*
 - Trasferire le perdite
 - *Assicurarsi*

Slide 24

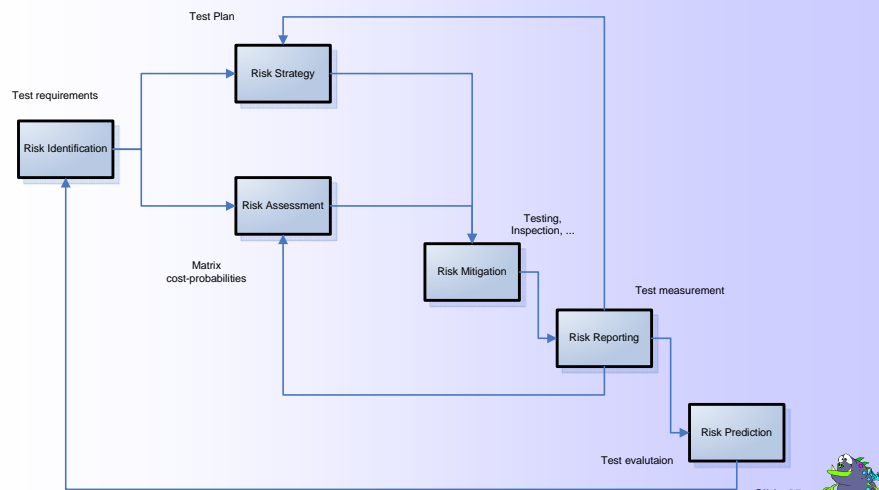




Risk-Based testing

Ingegneria del Software
Progettazione Laboratorio
Verifica e Validazione
Maurizio Pighin

- Risk Analysis



Slide 25



Risk-Based testing

Ingegneria del Software
Progettazione Laboratorio
Verifica e Validazione
Maurizio Pighin

- Risk Analysis

- *Applicabile a livello di*
 - Sistema
 - Sottosistema
 - Singole funzioni
- *Problematiche connesse*
 - Difficile da misurare
 - Failure da accettare per compensare il rischio

Slide 26





Risk-Based testing

Ingegneria del Software
Progettazione Laboratorio
Verifica e Validazione
Maurizio Pighin

- Risk Analysis
 - *L'analisi del rischio dovrebbe limitarsi ad un numero ristretto di classi (3-5) di approssimativamente uguale rischio*
 - *Per capire le tipologie di failure, si può usare ad esempio lo Standard IEEE 9126 come check-list*
 - Difetti funzionali
 - Cattiva performance
 - Cattiva usabilità
 - Bassa manutenibilità
 -

Slide 27



Risk-Based testing

Ingegneria del Software
Progettazione Laboratorio
Verifica e Validazione
Maurizio Pighin

- Test Risk-Based – Aspetti pratici
 - *Prima del test*
 - Identificare che cosa è critico
 - *Il test evidenzia aree con molti difetti*
 - *Attivare extra testing*
 - Extra test fatto da specialisti
 - Test di regressione
 -
- Priorizzare per il primo test

Slide 28





Risk-Based testing

Ingegneria del Software
Progettazione Laboratorio
Verifica e Validazione
Maurizio Pighin

- Priorizzare in base ai fattori di danno
 - Quali funzioni o attributi sono “critici”?
 - Business risk
 - Quanto è visibile un problema in una funzione o in un attributo?
 - Customer, User
 - Quanto spesso la funzione è usata?
 - Si può farne a meno?
 - Quali sono le conseguenze legali?

Slide 29



Risk-Based testing

Ingegneria del Software
Progettazione Laboratorio
Verifica e Validazione
Maurizio Pighin

- Priorizzare in base alla probabilità di fault
 - Aree complesse
 - Aree cambiate
 - Numero di persone coinvolte
 - Turnover
 - Nuove tecnologie, metodologie
 - Nuovi strumenti
 - Time pressure
 - Aree che richiedono ottimizzazioni
 - Aree con molti difetti prima
 - Distribuzione geografica
 - Storia sull'utilizzo precedente
 - Fattori locali

Slide 30





Risk-Based testing

Ingegneria del Software
Progettazione Laboratorio
Verifica e Validazione
Maurizio Pighin

- Non dimenticare
 - Possiamo testare solo una parte del prodotto?
 - Possiamo rimandare altre aree, versioni a dopo?
 - Dobbiamo combattere il “time pressure”
- Come calcolare la priorità?
 - Definire gli elementi che si vogliono analizzare
 - Assegnare i pesi alle aree di Danno e di Probabilità prescelte (1-3-10)
 - Assegnare punti ad ogni area-elemento (1-2-3-4-5)
 - Valutare la probabilità di “defect detection” nella fase di QA
 - Calcolare la somma pesata danno*probabilità
 - [Esempio](#)

Slide 31



Risk-Based testing

Ingegneria del Software
Progettazione Laboratorio
Verifica e Validazione
Maurizio Pighin

- Alcune riflessioni
 - Utilizzo risultati
 - Rischio Alto → Test intensivo
 - Rischio Medio → Test ordinario
 - Rischio Basso → Test blando
 - Valutare attentamente eventuali risultati inattesi

Slide 32





Risk-Based testing

Ingegneria del Software
Progettazione Laboratorio
Verifica e Validazione
Maurizio Pighin

- Cosa fare se non si conosce nulla del prodotto?
 - Lanciare un test generale leggero su tutto
 - Priorizzare in base alle aree di rischio evidenziate nel primo test
 - Analizzare fault su copertura test
 - Analizzare le densità di fault segnalate
 - Lanciare un secondo ciclo di test

Slide 33



Risk-Based testing

Ingegneria del Software
Progettazione Laboratorio
Verifica e Validazione
Maurizio Pighin

- Analisi della copertura dei test
 - Tutte le funzioni importanti?
 - Exception Handling?
 - Possibili stati e transizioni?
 - Requisiti non funzionali importanti?
- Fare Extra-test se la copertura effettiva è diversa dall'attesa
 - Se una area è sovra-coperta significa che il codice è sovrautilizzato. Possibili problemi di performance o colli di bottiglia
 - Se un'area è sotto-coperta significa che il codice è sottoutilizzato. Possibile area superflua o specifiche troppo leggere.

Slide 34





Risk-Based testing

Ingegneria del Software
Progettazione Laboratorio
Verifica e Validazione
Maurizio Pighin

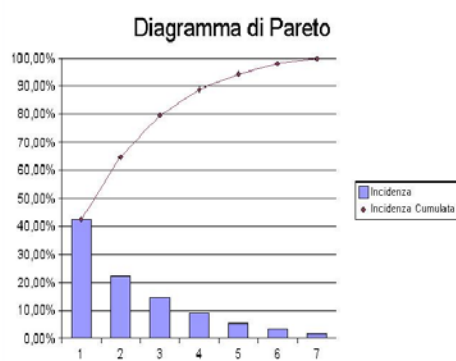
- Analisi della densità di fault
 - *Alcuni fatti*
 - Il test non trova tutti i fault
 - Più ne trova più ne restano
 - I test post-rilascio sono correlati con i fault trovati nel test
 - *Alcuni dati*
 - 90% fault più critici sono nel 2,5% delle unità (NSA)
 - Usualmente 80% fault arrivano dal 20% delle unità
- I fault sono creature “sociali”: tendono a vivere insieme

Slide 35



Risk-Based testing

Ingegneria del Software
Progettazione Laboratorio
Verifica e Validazione
Maurizio Pighin



- Uno strumento utile a definire questo comportamento è il **diagramma di Pareto** che consiste in un istogramma della distribuzione percentuale di un fenomeno, ordinato in senso decrescente, affiancato al grafico delle frequenze cumulate (curva di Lorenz). Il grafico può aiutare a stabilire quali sono i maggiori fattori che hanno influenza su un dato fenomeno
- Secondo la "legge 80/20" (i valori 80% e 20% sono ottenuti mediante osservazioni empiriche di numerosi fenomeni e sono solo indicativi), in genere l'80% dei risultati dipende dal 20% delle cause.

Slide 36





Risk-Based testing

Ingegneria del Software
Progettazione Laboratorio
Verifica e Validazione
Maurizio Pighin

- Come usare la densità di fault
 - *Misurare il numero di fault / dimensione*
 - *Comparare con la media dell'organizzazione*
 - *Spendere extra-analisi se*
 - Unità sotto test molto sopra la media (probabili problemi nell'unità)
 - Unità sotto test molto sotto la media (probabili problemi nel test)

Slide 37



Risk-Based testing

Ingegneria del Software
Progettazione Laboratorio
Verifica e Validazione
Maurizio Pighin

- Analisi delle cause
 - *Se si rileva un accumulo di fault dovuto alla stessa categoria di problemi, va valutato se cambiare l'organizzazione del lavoro*
 - *Categorie tipiche cause*
 - Logiche
 - Computazionali
 - Di interfaccia
 - Problemi nei dati di ingresso
 - Documentazione
 - Cambiamenti

Slide 38





Risk-Based testing

Ingegneria del Software
Progettazione Laboratorio
Verifica e Validazione
Maurizio Pighin

- Rischi di progetto per il Testatore
 - *Rischi PRIMA del test*
 - Cattiva qualità (troppi errori, errori bloccanti, troppe versioni):
 - *Gestire correttamente i processi di qualità prima del test*
 - Ritardi
 - *Piani di lavoro alternativi*
 - Mancanza di conoscenza
 - *Testare le versioni precedenti*
 - *Rischi DOPO il test*
 - Non dovrebbero presentarsi
 - Utente utilizza il sistema in maniera diversa
 - Utente trova errori

Slide 39



Risk-Based testing

Ingegneria del Software
Progettazione Laboratorio
Verifica e Validazione
Maurizio Pighin

- Rischi DURANTE il test
 - *Cattiva gestione*
 - *Mancanza di skill specifici*
 - *Troppo poche persone, o sbagliate, o troppo tardi*
 - *Cattiva cooperazione*
 - *Cattiva coordinazione*
 - *Problemi con strumenti di lavoro*

Slide 40





Risk-Based testing

Ingegneria del Software
Progettazione Laboratorio
Verifica e Validazione
Maurizio Pighin

- Come gestire il test nella maniera più economica?
 - *Persone capaci*
 - *Buona priorizzazione*
 - *Liberarsi di una parte del lavoro*
 - Trovare qualcuno che paghi il lavoro o eliminare alcune fasi
 - Meno documentazione più test esplorativo
- Come tagliare i costi di installazione
 - *Definire quando correggere e quando no*
 - Riparare solo i difetti che causano problemi importanti
 - Fare una richiesta di modifica per la prossima release per gli altri
 - Installare le correzioni a gruppi

Slide 41



Automated static analysis

Ingegneria del Software
Progettazione Laboratorio
Verifica e Validazione
Maurizio Pighin

- Static analysers are software tools for source text processing
- They parse the program text and try to discover potentially erroneous conditions and bring these to the attention of the V & V team
- Very effective as an aid to inspections. A supplement to but not a replacement for inspections

Slide 42





Static analysis checks

Ingegneria del Software
Progettazione Laboratorio
Verifica e Validazione
Maurizio Pighin

Fault class	Static analysis check
Data faults	Variables used before initialisation Variables declared but never used Variables assigned twice but never used between assignments Possible array bound violations Undeclared variables
Control faults	Unreachable code Unconditional branches into loops
Input/output faults	Variables output twice with no intervening assignment
Interface faults	Parameter type mismatches Parameter number mismatches Non-usage of the results of functions Uncalled functions and procedures
Storage management faults	Unassigned pointers Pointer arithmetic

Slide 43



Stages of static analysis

Ingegneria del Software
Progettazione Laboratorio
Verifica e Validazione
Maurizio Pighin

- Control flow analysis. Checks for loops with multiple exit or entry points, finds unreachable code, etc.
- Data use analysis. Detects uninitialised variables, variables written twice without an intervening assignment, variables which are declared but never used, etc.
- Interface analysis. Checks the consistency of routine and procedure declarations and their use

Slide 44





Stages of static analysis

Ingegneria del Software
Progettazione Laboratorio
Verifica e Validazione
Maurizio Pighin

- Information flow analysis. Identifies the dependencies of output variables. Does not detect anomalies itself but highlights information for code inspection or review
- Path analysis. Identifies paths through the program and sets out the statements executed in that path. Again, potentially useful in the review process
- Both these stages generate vast amounts of information. Must be used with care.

Slide 45



138% more lint_ex.c

```
#include <stdio.h>
printarray (Anarray)
{
    printf("%d",Anarray);
}
main ()
{
    int Anarray[5]; int i; char c;
    printarray (Anarray, i, c);
    printarray (Anarray) ;
}
```

139% cc lint_ex.c

140% lint lint_ex.c

```
lint_ex.c(10): warning: c may be used before set
lint_ex.c(10): warning: i may be used before set
printarray: variable # of args. lint_ex.c(4) :: lint_ex.c(10)
printarray, arg. 1 used inconsistently lint_ex.c(4) :: lint_ex.c(10)
printarray, arg. 1 used inconsistently lint_ex.c(4) :: lint_ex.c(11)
printf returns value which is always ignored
```

LINT static analysis

Ingegneria del Software
Progettazione Laboratorio
Verifica e Validazione
Maurizio Pighin

Slide 46





Use of static analysis

Ingegneria del Software
Progettazione Laboratorio
Verifica e Validazione
Maurizio Pighin

- Particularly valuable when a language such as C is used which has weak typing and hence many errors are undetected by the compiler
- Less cost-effective for languages like Java that have strong type checking and can therefore detect many errors during compilation

Slide 47



Formal methods and critical systems

Ingegneria del Software
Progettazione Laboratorio
Verifica e Validazione
Maurizio Pighin

- The development of critical systems is one of the 'success' stories for formal methods
- Formal methods are mandated in Britain for the development of some types of safety-critical software for defence applications
- There is not currently general agreement on the value of formal methods in critical systems development

Slide 48





Formal methods and validation

Ingegneria del Software
Progettazione Laboratorio
Verifica e Validazione
Maurizio Pighin

- Specification validation
 - *Developing a formal model of a system requirements specification forces a detailed analysis of that specification and this usually reveals errors and omissions*
 - *Mathematical analysis of the formal specification is possible and this also discovers specification problems*
- Formal verification
 - *Mathematical arguments (at varying degrees of rigour) are used to demonstrate that a program or a design is consistent with its formal specification*

Slide 49



Problems with formal validation

Ingegneria del Software
Progettazione Laboratorio
Verifica e Validazione
Maurizio Pighin

- The formal model of the specification is not understandable by domain experts
 - *It is difficult or impossible to check if the formal model is an accurate representation of the specification for most systems*
 - *A consistently wrong specification is not useful!*
- Verification does not scale-up
 - *Verification is complex, error-prone and requires the use of systems such as theorem provers. The cost of verification increases exponentially as the system size increases.*

Slide 50





Formal methods conclusion

Ingegneria del Software
Progettazione Laboratorio
Verifica e Validazione
Maurizio Pighin

- Formal specification and checking of critical system components is, in my view (Sommerville), useful
 - *While formality does not provide any guarantees, it helps to increase confidence in the system by demonstrating that some classes of error are not present*
- Formal verification is only likely to be used for very small, critical, system components
 - *About 5-6000 lines of code seems to be the upper limit for practical verification*

Slide 51



Statistical testing

Ingegneria del Software
Progettazione Laboratorio
Verifica e Validazione
Maurizio Pighin

- Testing software for reliability rather than fault detection
- Measuring the number of errors allows the reliability of the software to be predicted. Note that, for statistical reasons, more errors than are allowed for in the reliability specification must be induced
- An acceptable level of reliability should be specified and the software tested and amended until that level of reliability is reached

Slide 52

