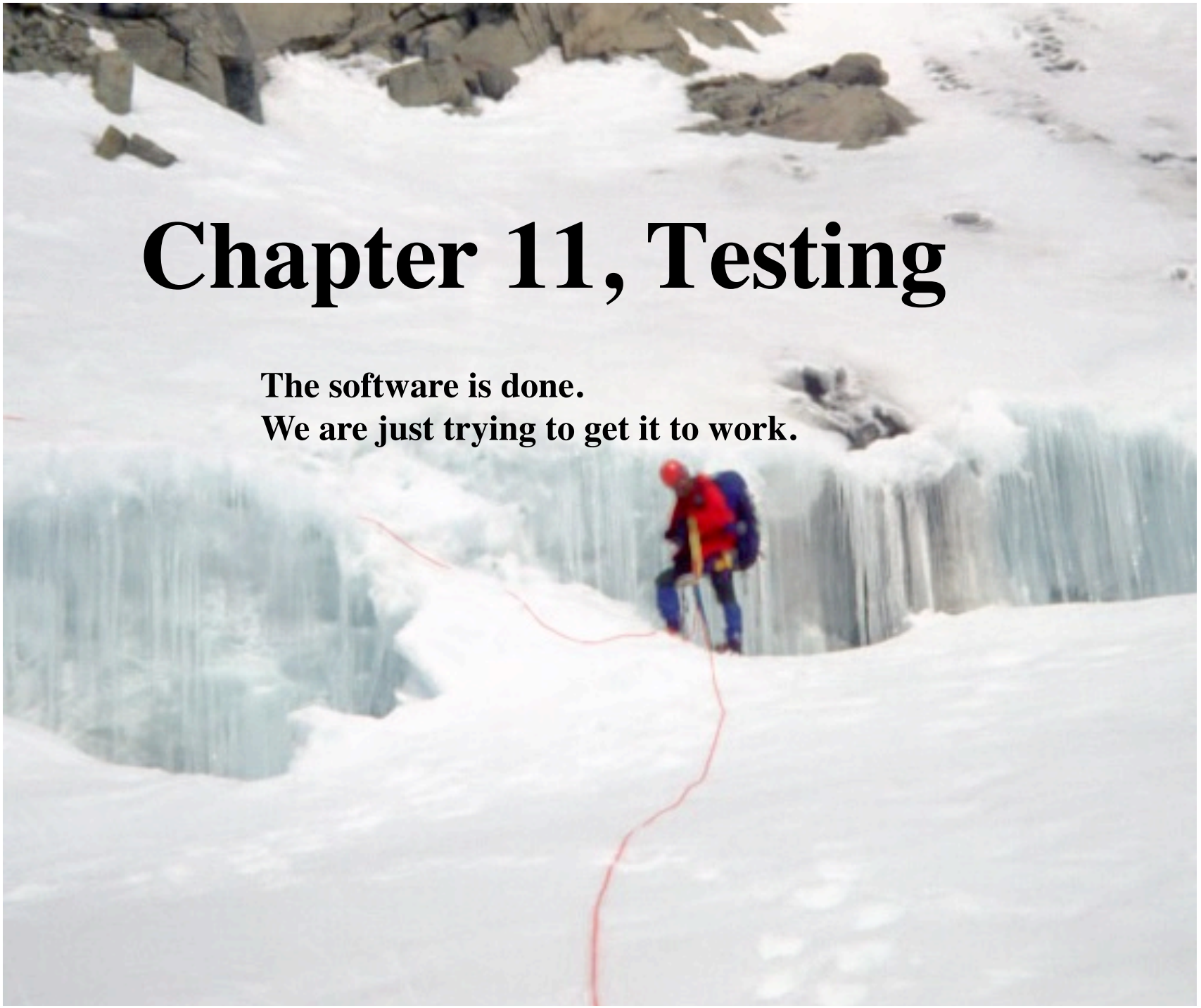
Object-Oriented Software Engineering
Using UML, Patterns, and Java



Chapter 11, Testing

The software is done.
We are just trying to get it to work.

Overview

- Il Testing consiste nel trovare le differenze tra il comportamento atteso specificato attraverso il modello del sistema e il comportamento osservato dal sistema implementato.

- **Unit** testing
  - **Trovare differenze tra object design model e corrispondente componente**
- **Structural** testing
  - **Trovare differenze tra system design model e un sottoinsieme integrato di sottosistemi**
- **Functional** testing
  - **Trovare differenze tra use case model e il sistema**
- **Performance** testing
  - **Trovare differenze tra requisiti non funzionali e le performance del sistema reale**

Overview

- The Goal: progettare test per provare il sistema e rivelare problemi
  - **Massimizzare il numero di errori scoperti che consentirà agli sviluppatori di correggerli**
- Questa attività va in contrasto con le altre attività svolte prima: analisi, design, implementazione sono attività “costruttive”. Il testing invece tenta di “rompere” il sistema
- Il testing dovrebbe essere realizzato da sviluppatori che non sono stati coinvolti nelle attività di costruzione del sistema

Outline

- Terminologia
- Tecniche per gestire i difetti e gli errori
- Le attività di testing
- Component/Unit testing
- Integration testing
- System testing
- Gestione del testing

Quality of today's software….

- I prodotti software messi sul mercato non sono "error free".

A Joke

- How cars would work if they followed a development history similar to that of computers
- At the COMDEX computer exposition, Bill Gates
  - **"If General Motors had kept up with technology like the computer industry has, we would all be driving $25 cars that got 1,000 miles to the gallon."**

General Motors' Reply

- "If GM had developed technology like Microsoft, we would all be driving cars with the following characteristics:
  - **For no reason whatsoever your car would crash twice a day.**
  - **Every time they repainted the lines on the road, you would have to buy a new car.**
  - **Macintosh would make a car that was powered by the sun, reliable, five times as fast, and twice as easy to drive, but would run on only five percent of the roads.**

General Motors' Reply

- New seats would force everyone to have the same size hips.
- The airbag system would say "Are you sure?" before going off.
- Occasionally, for no reason whatsoever, your car would lock you out and refuse to let you in until you simultaneously lifted the door handle, turned the key, and grabbed hold of the radio antenna."

Is quality a real problem with IT projects?

- Most people simply accept poor quality from many IT products.
  - **So what if your computer crashes a couple of times a month? Just make sure you back up your data.**
  - **So what if you cannot log in to the corporate intranet or the Internet right now? Just try a little later when it is less busy.**
  - **So what if the latest version of your word-processing software was shipped with several known bugs? You like the software's new features, and all new software has bugs.**

Is quality a real problem with IT projects?

- Yes, it is! IT is not just a luxury available in some homes, schools, or offices.
- Many aspects of our daily lives depend on high-quality IT products.
  - **Food is produced and distributed with the aid of computers;**
- Many IT projects develop mission-critical systems that are used in life-and-death situations
  - **navigation systems on aircraft, computer components built into medical equipment…**

Is quality a real problem with IT projects?

- Financial institutions and their customers also rely on high-quality information systems.
  - **Customers get very upset when systems provide inaccurate financial data or reveal information to unauthorized users that could lead to identity theft.**
- When one of these systems does not function correctly, it is much more than a slight inconvenience!

WHAT WENT WRONG?

- In 1981, a small timing difference caused by a computer program change created a 1-in-67 chance that the space shuttle's five onboard computers would not synchronize. The error caused a launch abort.
- In 1986, two hospital patients died after receiving fatal doses of radiation from a Therac 25 machine. A software problem caused the machine to ignore calibration Data.
- …

WHAT WENT WRONG?

- One of the biggest software errors in banking history:
  - **Chemical Bank mistakenly deducted about $15 million from more than 100,000 customer accounts. The problem resulted from a single line of code in an updated program that caused the bank to process every withdrawal and transfer at its automated teller machines (ATMs) twice. For example, a person who withdrew $100 from an ATM had $200 deducted from his or her account, though the receipt indicated only a withdrawal of $100. The mistake affected 150,000 transactions**

Software quality management

- Concerned with ensuring that the required level of quality is achieved in a software product.
- Involves defining appropriate quality standards and procedures and ensuring that these are followed.
- Should aim to develop a ‘quality culture’ where quality is seen as everyone’s responsibility.

Perché? Che cosa? Quando?

- GOAL: software con zero difetti …
- MA impossibile da ottenere e garantire
- Necessaria una attenta e continua ***VERIFICA*** *e* ***CONVALIDA*** *(V&V)*
- Tutto deve essere verificato: documenti di specifica, di progetto, dati di collaudo, ….programmi
- Si fa lungo tutto il processo di sviluppo, **NON** solo alla fine!

Verification & Validation (V&V)

- Verifica (*verification):*
- –insieme delle attività volte a stabilire se il sw costruito soddisfa le specifiche (non solo funzionali)
- –did we build the program right?
- si assume che le specifiche esprimano in modo esauriente e corretto i *desiderata del committente*
-
- Convalida (validation):
- –stabilire che il sistema soddisfa le esigenze vere dell'utente
- –did we build the right program ?
- –Può essere svolta sulla specifica (meglio!) e/o sul sistema finale

Prova di correttezza e testing

- Prova di correttezza:
  - **–argomentare sistematicamente (in modo formale o informale) che il programma funziona correttamente per *tutti i possibili dati di ingresso***

- Testing:
  - **particolare tipo di attività sperimentale fatta mediante esecuzione del programma, selezionando alcuni dati di ingresso e valutando risultati**
  - **dà un riscontro *parziale: programma provato solo per quei dati***
  - **Tecnica *dinamica* rispetto alle *verifiche statiche fatte dal compilatore***

Testing

- Program testing can be used to show the *presence of bugs, but never to show their absence. (Dijkstra 1972)*
- Quindi obiettivo del testing è di **trovare "controesempi"**
- si cerca di:

trovare dati di test che massimizzino la probabilità di scoprire errori durante l'esecuzione

Terminology

- **Affidabilità:** La misura di successo con cui il comportamento osservato di un sistema è conforme ad una certa specifica del relativo comportamento.
- **Fallimento (failure):** Qualsiasi deviazione del comportamento osservato dal comportamento specificato.
- **Stato di Errore:** Il sistema è in uno stato tale che ogni ulteriore elaborazione da parte del sistema porta ad un fallimento.
- **Difetto (Bug/fault):** La causa meccanica o algoritmica di un failure.

Fault e Failure

```
int raddoppia(x)
{
    int y;
    y = x*x;
    return (y);
}
```

Per il valore di ingresso x = 3 si ha il valore di uscita y = 9. La causa di questa ***failure*** è il ***fault*** di linea 2, in cui anziché l'operatore + è usato l'operatore *

NB: Se il valore di ingresso è x = 2, il valore di uscita è y = 4 (***nessuna failure***)

Fault, Failure e Difetto

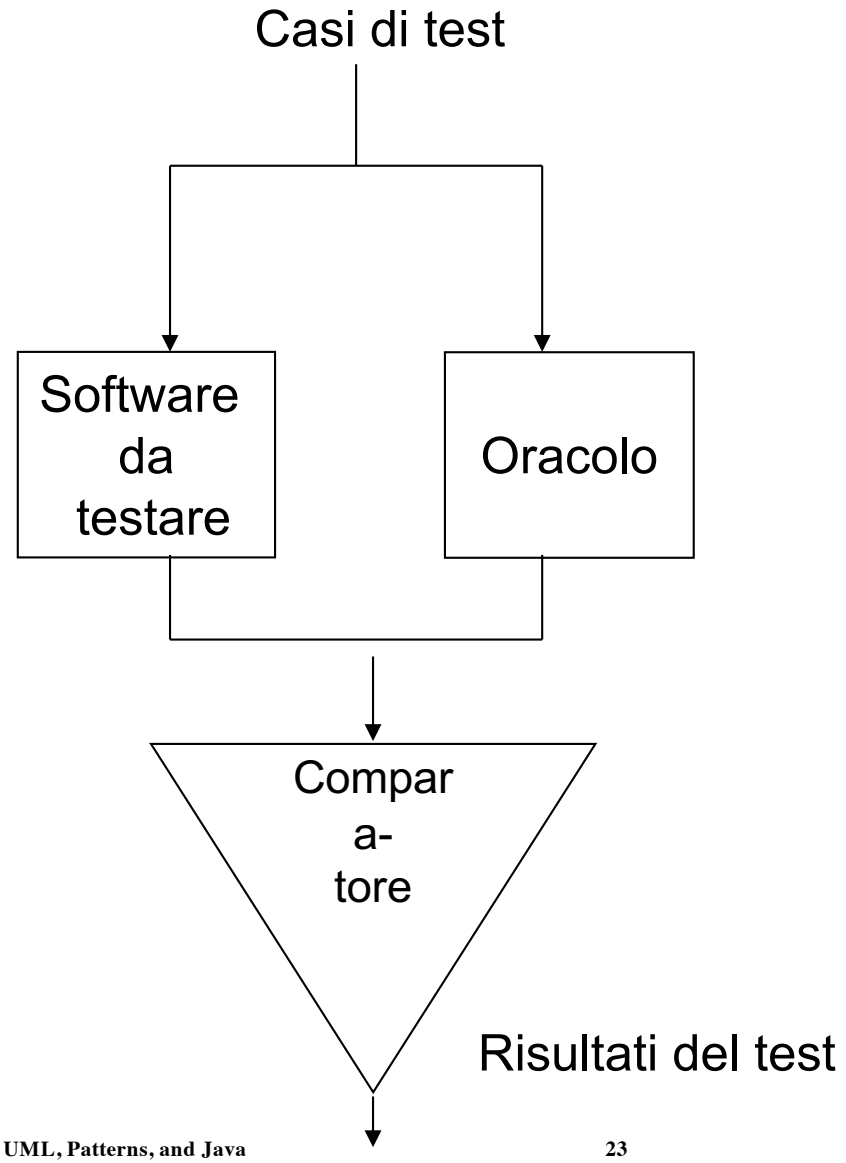
- Non tutti i fault generano failure
- Una failure può essere generata da più fault
- Un fault può generare diverse failure
- ***Defect*** (difetto) quando non è importante distinguere fra fault e failure si può usare il termine defect per riferirsi sia alla causa (fault) che all'effetto (failure)

Test e Casi di test

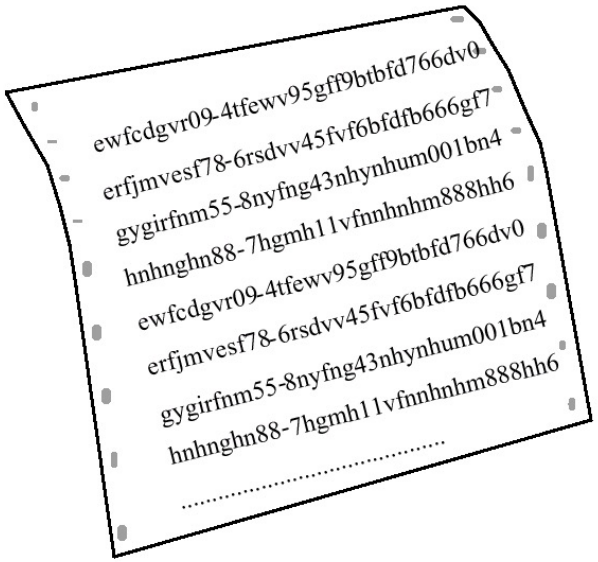
- Un programma è esercitato da un caso di test (insieme di dati di input e…risultato atteso...)
- Un ***test*** (o test suite) è formato da un **insieme di casi di test**
- **L'esecuzione del test** consiste nell'esecuzione del programma per tutti i casi di test
- Un test ha **successo** se rileva uno o più malfunzionamenti (failure) del programma

L'oracolo …

- Condizione necessaria per effettuare un test:
  - **conoscere il comportamento atteso per poterlo confrontare con quello osservato**
- L'oracolo conosce il comportamento atteso per ogni caso di prova
- Oracolo umano
  - **si basa sulle specifiche o sul giudizio**
- Oracolo automatico
  - **generato dalle specifiche (formali)**
  - **stesso software ma sviluppato da altri**
  - **versione precedente (test di regressione)**



Un buon oracolo



- Il test di applicazioni grandi e complesse può richiedere milioni di casi di test
- La dimensione dello spazio di uscita può eccedere le capacità umane
- L'occhio umano è lento e poco affidabile anche per uscite di piccole dimensioni

ORACOLI AUTOMATICI SONO ESSENZIALI !

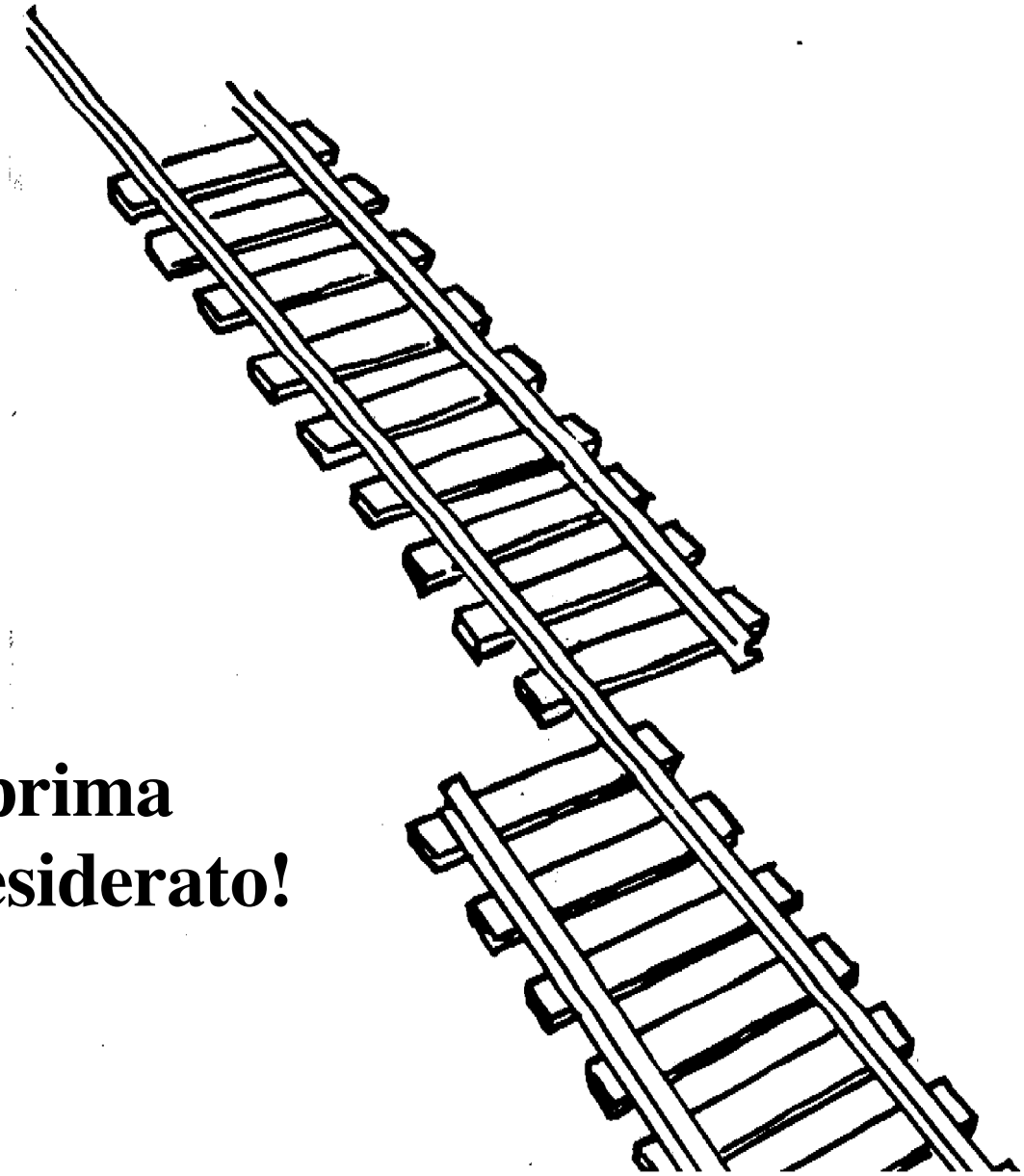
Cos'è?

Un failure?

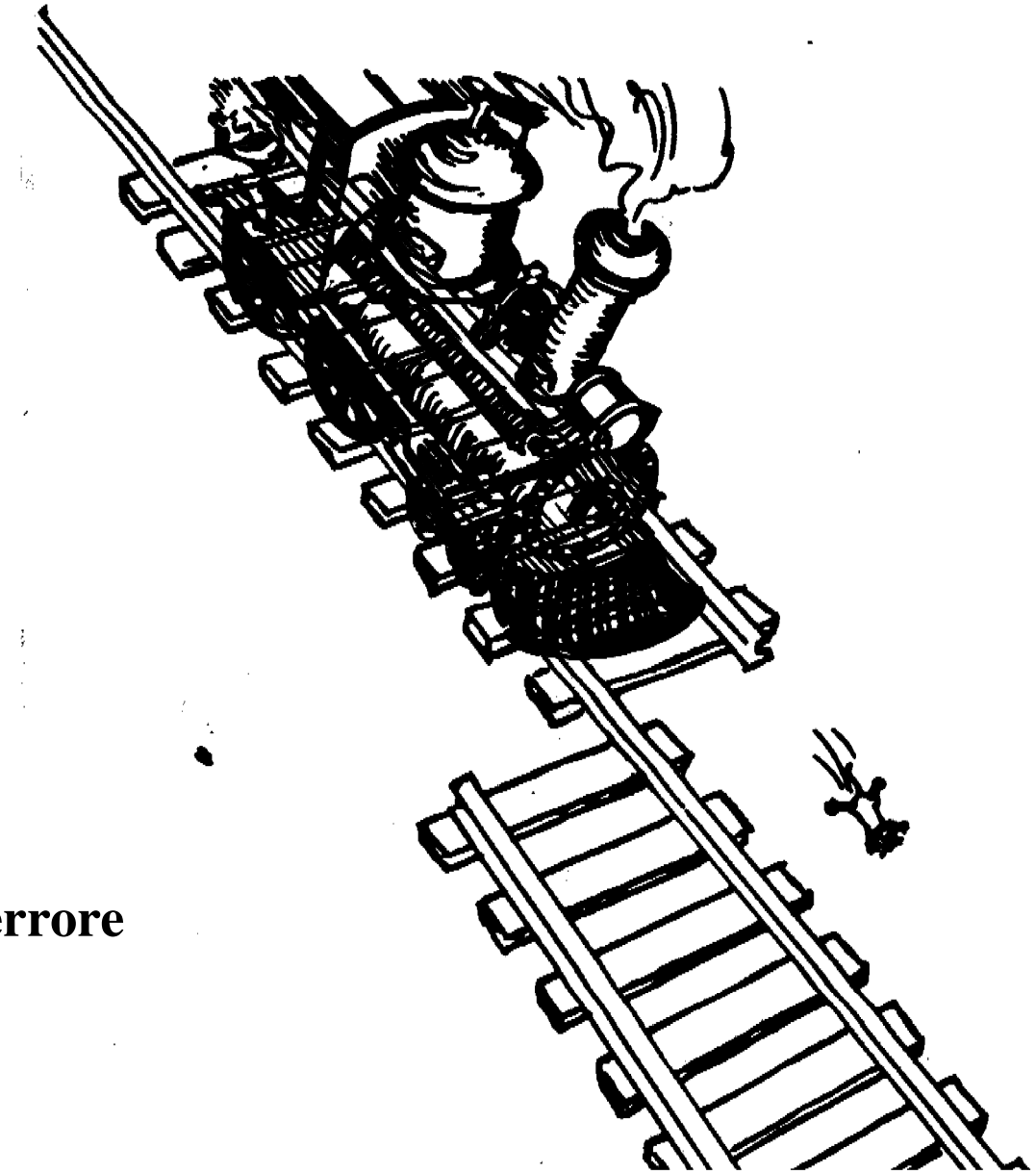
Un errore?

Un fault?

Bisogna specificare prima il comportamento desiderato!



Erroneous State ("Error")

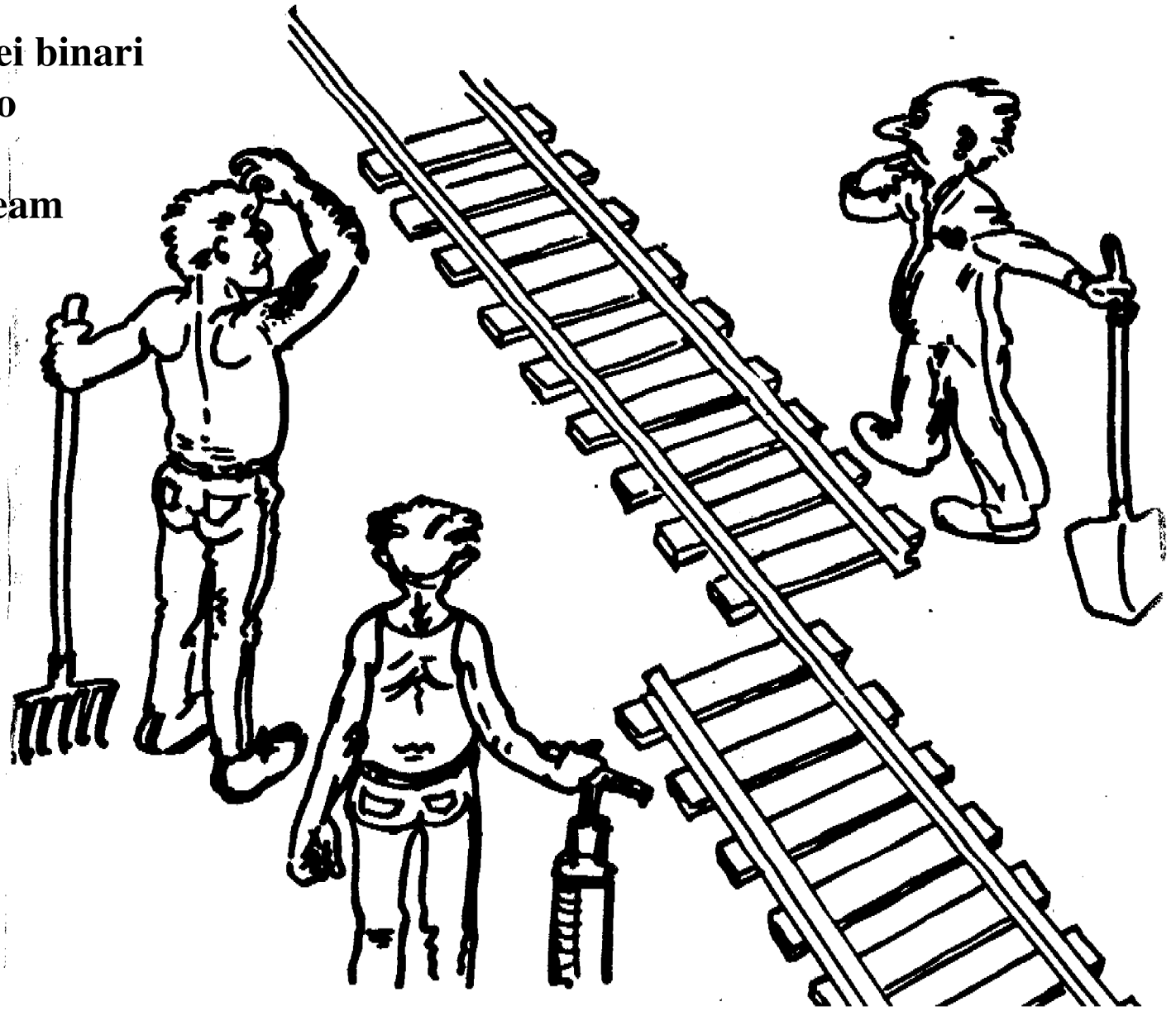


Lo stato corrente mostra un errore
Ma non un fallimento!

Algorithmic Fault

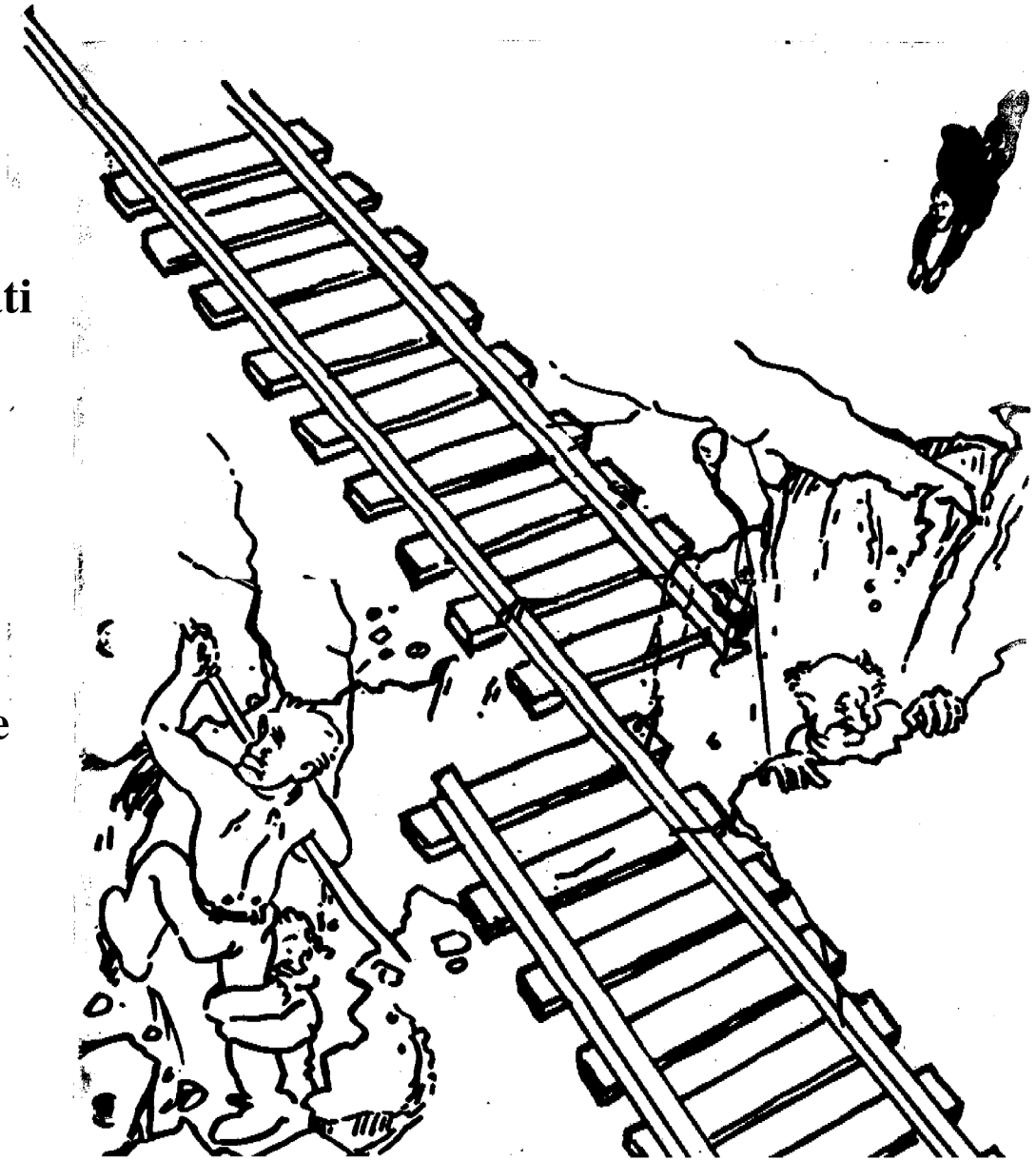
Il non allineamento dei binari può essere un risultato della cattiva comunicazione tra i team

Oppure l'implementazione sbagliata della specifica da parte di un team



Mechanical Fault

Anche se i binari sono implementati in accordo alle specifiche nel RAD potrebbero risultare non allineate nella messa in opera, per esempio, a causa di un terremoto



Un fallimento della virtual machine di un sistema software è un altro esempio di fallimento meccanico: anche se lo sviluppatore ha implementato correttamente, cioè mappato correttamente l'object model nel codice, il comportamento osservato può deviare ancora da quello atteso

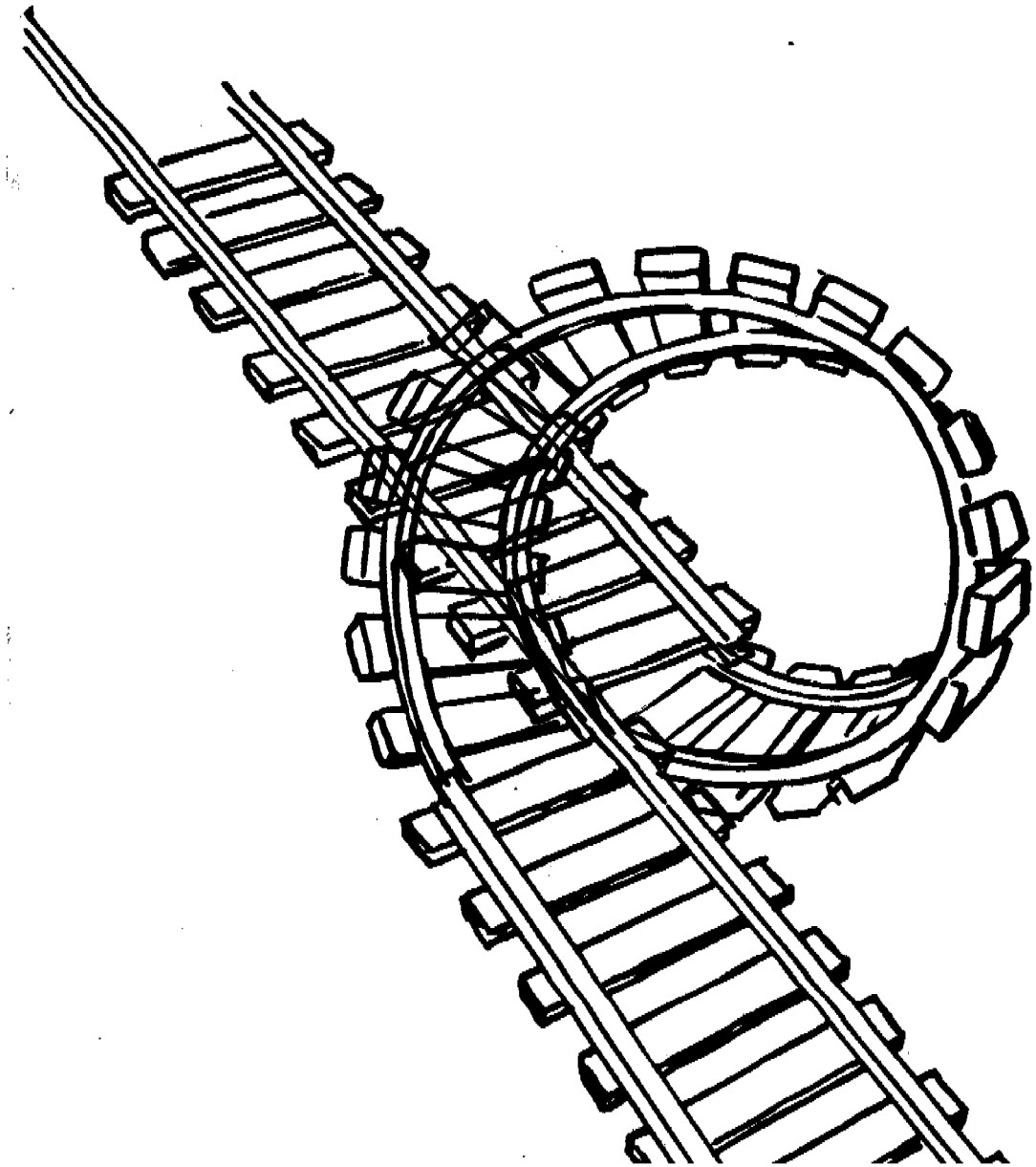
Esempi: Fault e Errori

- Fault nella specifica delle interfacce
  - **Disallineamento tra quello di cui il client ha bisogno e cosa gli viene offerto**
  - **Disallineamento fra la specifica e l'esecuzione**
- Algorithmic Faults
  - **Inizializzazione mancante Errori di ramificazione (uscita dal loop troppo presto, troppo in ritardo)**
  - **Prova mancante per zero**

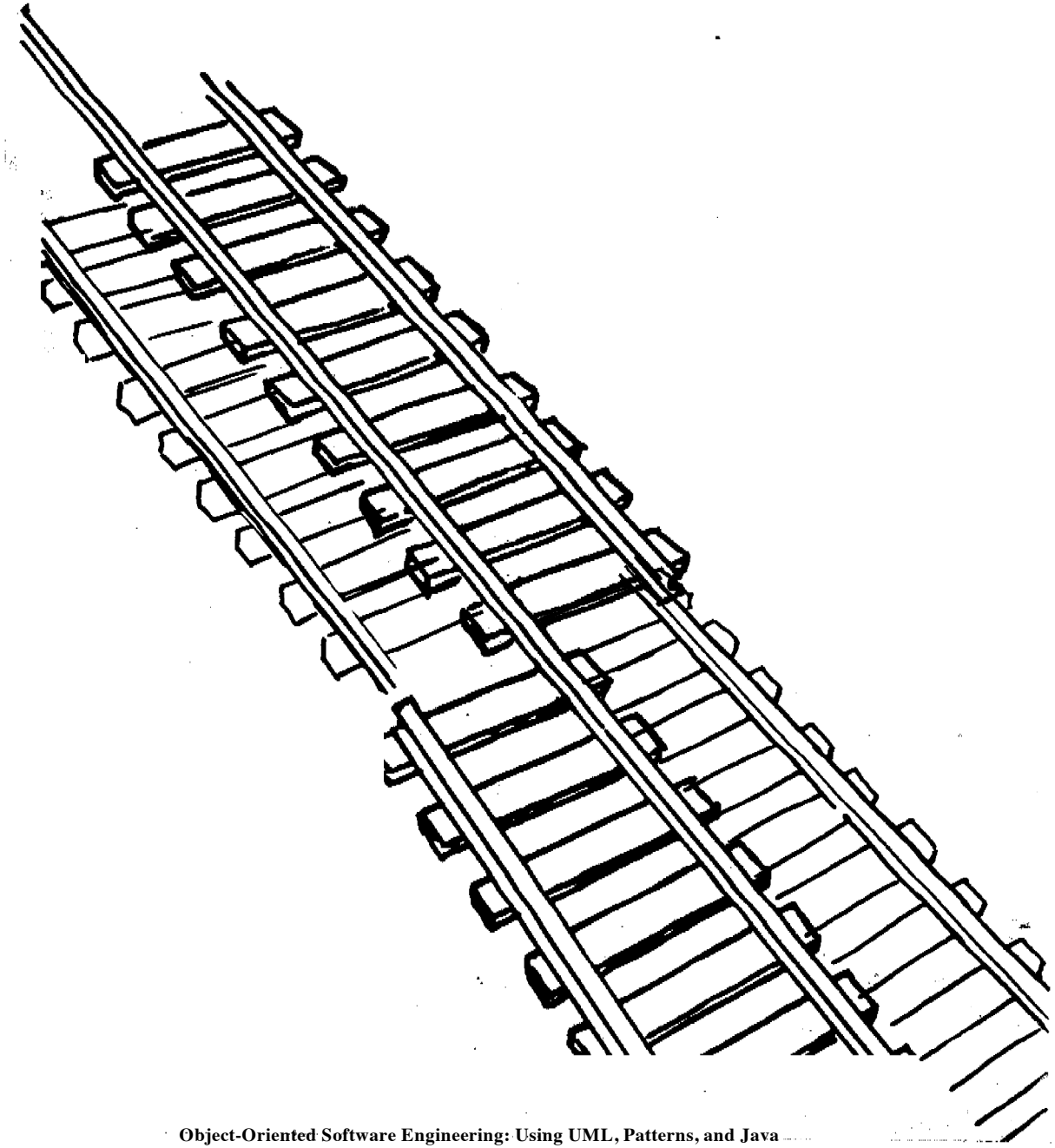
- Mechanical Faults (difficili da trovare)
  - **La documentazione non è aderente alle condizioni reali e alle procedure operative**
- Errori
  - **Errori che scaturiscono da sovraccarico**
  - **Errori limite e di capacità**
  - **Errori legati al throughput o alla performance**

Come trattare gli errori e i difetti?

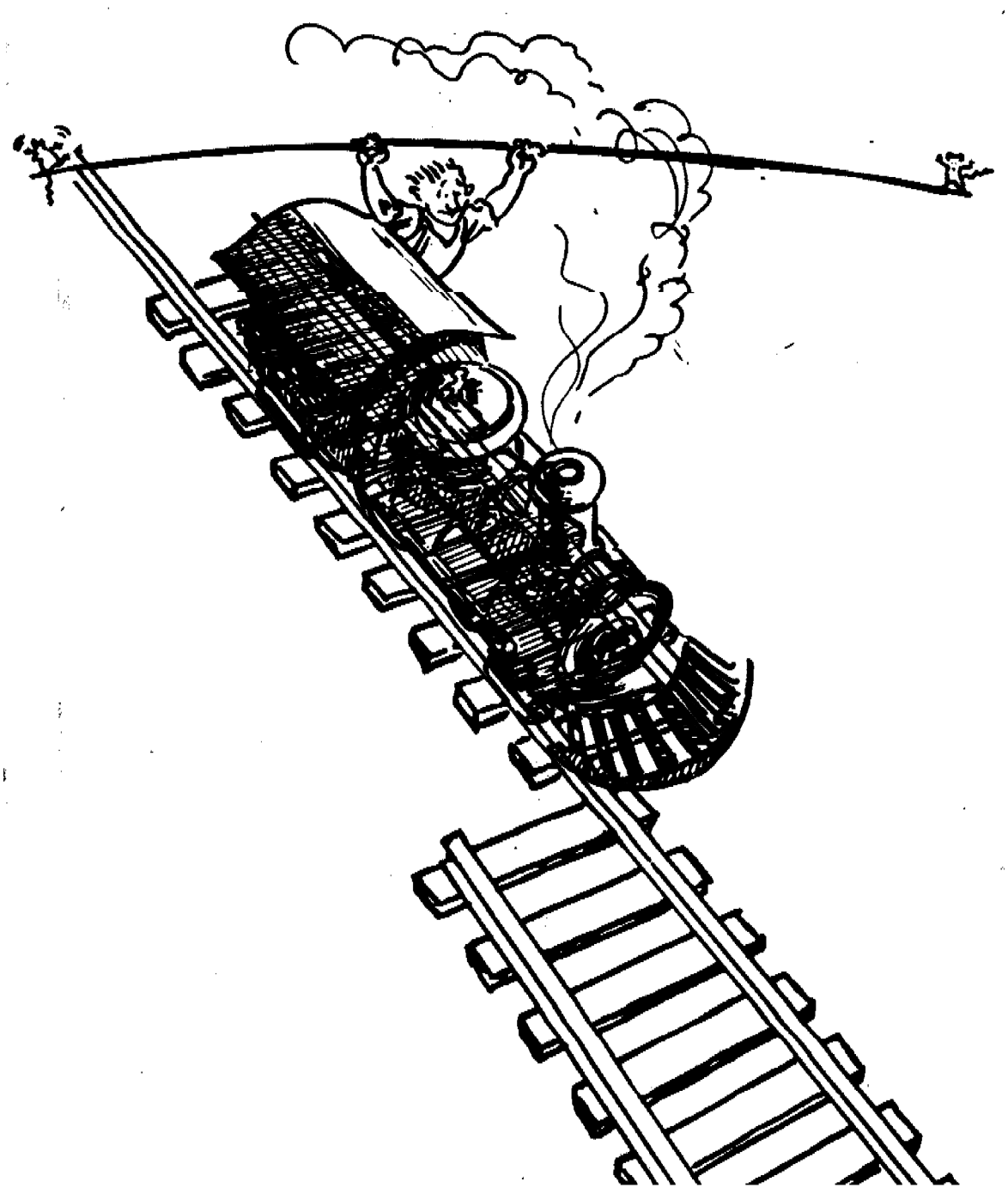
Verification?



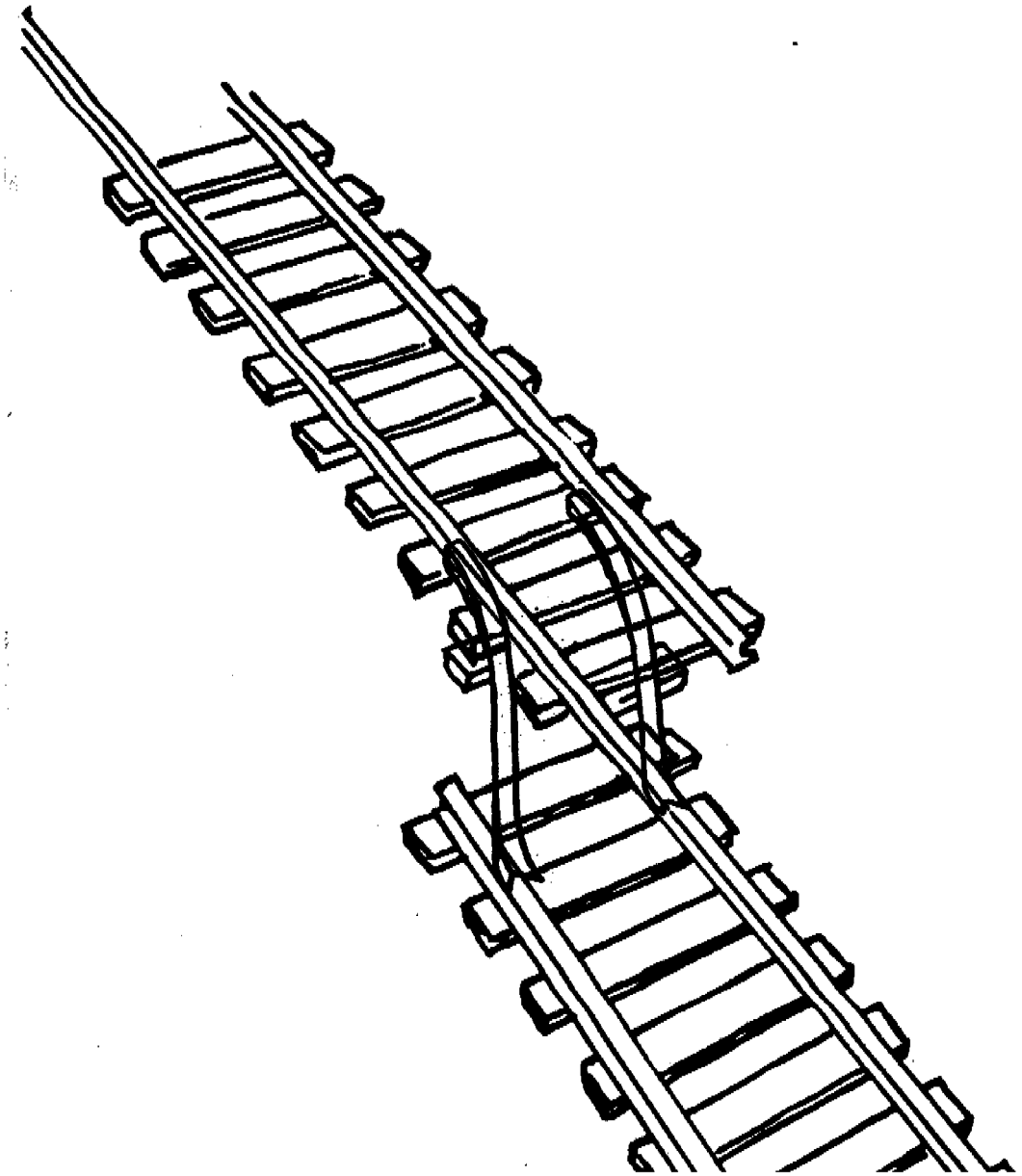
Modular Redundancy?



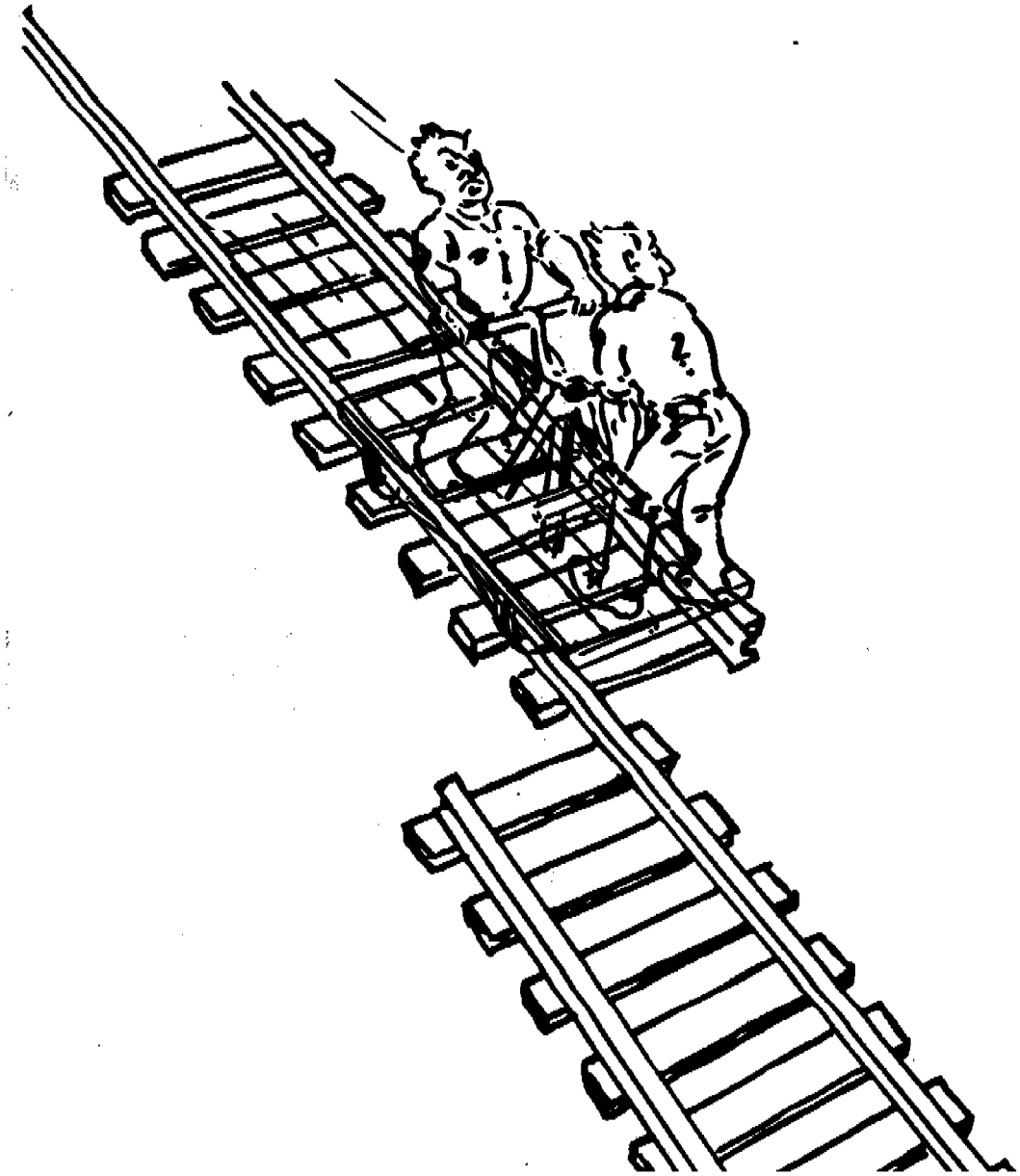
Declaring the Bug as a Feature?



Patching?



Testing?



Dealing with Defects

- **Verification**:
  - Assumes hypothetical environment that does not match real environment
  - Proof might be buggy (omits important constraints; simply wrong)
- **Modular redundancy:**
  - Expensive
- **Declaring a bug to be a "feature"**
  - Bad practice
- **Patching**
  - Slows down performance
- **Testing** (this lecture)
  - Testing is never good enough

Tecniche per aumentare l'Affidabilità di un sistema software

- **Fault Avoidance**. Tecniche che tentano di prevenire l'inserimento di difetti nel sistema prima che sia realizzato
  - **Includono metodologie di sviluppo, gestione delle configurazioni, e verifica**
    - **Use good programming methodology to reduce complexity**
    - **Use version control to prevent inconsistent system**
    - **Apply verification to prevent algorithmic bugs**
- **Fault detection**. Tecniche come debugging e testing, sono rispettivamente esperimenti non controllati e controllati usati durante il processo di sviluppo per identificare stati di errore e trovare il difetto alla base. Includono anche review
  - **Testing: Create failures in a planned way**
  - **Debugging: Start with an unplanned failures**
  - **Review: revisione manuale**
- **Fault tollerance**. Assumono che un sistema possa essere realizzato con bug e che i fallimenti del sistema possano essere gestiti effettuando il recovery a run-time
  - **Data base systems (atomic transactions)**
  - **Modular redundancy**

Tecniche per aumentare l'Affidabilità di un sistema software

- Le tecniche **Fault detection**, includono anche *review*
- **Review**: ispezione manuale di alcuni o tutti gli aspetti del sistema senza eseguire realmente il sistema (molto efficace: 85% dei fault rilevati avvengono grazie a review)
- Due tipi di review: walkthrough e inspection.
  - Walkthrough**.** Lo sviluppatore presenta informalmente le API, il codice, la documentazione associata delle componenti al team di review. Il team commenta sul mapping modelli-codice usando RAD
  - Inspection**.** Simile al walkthrough, ma la presentazione delle unità è formale.
    - Lo sviluppatore non può presentare gli artefatti. Questo è fatto dal team di review che è responsabile di controllare
      - le interfacce e il codice rispetto ai requisiti
      - I commenti rispetto al codice
      - l'efficienza degli algoritmi rispetto alle richieste non funzionali
    - Lo sviluppatore interviene solo se si richiedono chiarimenti

Tecniche per aumentare l'Affidabilità di un sistema software

- **Debugging**. Assume che il bug possa essere trovato partendo da un failure non pianificato
  - Lo sviluppatore attraversa una sequenza di stati senza errore fino ad arrivare ad identificare un stato di errore.
  - A questo punto bisogna determinare il **bug** algoritmico o meccanico che ha causato questo stato.
  - Debugging sia per correttezza sia per performance
- **Testing**. Tecnica che tenta di creare stati di fallimento o errore in modo pianificato.
- Questa definizione implica che per successo del testing si indica la situazione in cui siamo stati in grado di generare un failure e quindi identificare fault.
  - Un buon test contiene test case che identificano fault
  - I test case dovrebbero includere un range ampio di valori di input, incluso input invalidi, e condizioni limite (boundary)
  - questo approccio richiede molto tempo anche per il testing di sistemi di piccole dimensioni

Some Observations

- It is impossible to completely test any nontrivial module or any system
  - **Theoretical limitations: Halting problem**
    - **Il settore del testing è tormentato da problemi indecidibili**
    - **un problema è detto *indecidibile* (irrisolubile) se è possibile dimostrare che non esistono algoritmi che lo risolvono ...**
    - **es. stabilire se l'esecuzione di un programma termina a fronte di un input arbitrario è un problema indecidibile**
  - **Practical limitations: Prohibitive in time and cost**
- Testing can only show the presence of bugs, not their absence (Dijkstra)
- Testing should be integrated with other verification activities, e.g., inspections

Testing takes creativity

- Testing often viewed as dirty work.
- To develop an effective test, one must have:
  - **Detailed understanding of the system**
  - **Knowledge of the testing techniques**
  - **Skill to apply these techniques in an effective and efficient manner**
- Testing is done best by independent testers
  - **We often develop a certain mental attitude that the program should in a certain way when in fact it does not.**
- Programmer often stick to the data set that makes the program work
  - **"Don't mess up my code!"**
- A program often does not work when tried by somebody else.
  - **Don't let this be the end-user.**

Testing esaustivo

- Testing *esaustivo (esecuzione per tutti i possibili ingressi) dimostra la correttezza*
  - **Es.** se programma calcola un valore in base a un valore di ingresso nel range 1..10, testing esaustivo consiste nel provare tutti i valori: per le 10 esecuzioni diverse si verifica se il risultato è quello atteso
- … impossibile da realizzare in generale:
  - **Es.** programma legge 3 input interi nel range 1..10.000 (10^4) e calcola un valore, testing esaustivo richiede 10^{12} ($(10^4)^3$) esecuzioni!

    1 test / millisecondo
  - **Quanti anni? (1 anno = 3,15 . 10^{10} msec)**
- …per programmi banali si arriva a tempi di esecuzione superiori al tempo passato dal big-bang

Terminazione del testing

- **Quando il programma si può ritenere analizzato a sufficienza?**
  - Criterio **temporale**: periodo di tempo predefinito
  - Criterio di **costo**: sforzo allocato predefinito
  - Criterio di **copertura**:
    - percentuale predefinita degli elementi di un modello del sw (si vedano i criteri di accettazione)
    - legato ad un criterio di selezione dei casi di test
  - Criterio **statistico**
    - MTBF (mean time between failures) predefinito (e confronto con un modello di affidabilità esistente)

Basic Testing Definitions

♦ **Fault:** A fault is the result of an error in the software documentation, code, etc.

♦ **Failure:** A failure occurs when a fault executes

♦ **Incident:** Consequences of failures – Failure occurrence may or may not be apparent to the user

♦ **Testing :** Exercise the software with test cases to find faults

♦ **Test case:** Set of input data and expected results (oracle) that exercise a component with the purpose of causing failures

♦ **Test suite (or test):** a set of test cases

Testing Concepts

- Una **Componente** è una parte del sistema che può essere isolata per essere testata (un oggetto, un gruppo di oggetti, uno o più sottosistemi)
- Un **test case** è un insieme di input e di risultati attesi che esercitano una componente con lo scopo di causare fallimenti e rilevare fault.
  - **Ha almeno 5 attributi: name, location, input, oracle, e log.**
    - **Name.** Per distinguere da altri test case (euristica: determinare il name a partire dal nome della componente o il requisito che si sta testando)
    - **Location**. Descrive dove il test case può essere trovato (un path name oppure un URL al programma da eseguire e il suo input)
    - **Input**. Descrive l'insieme di dati in input o comandi che l'attore del test case deve inserire
    - **Oracle**. Il comportamento atteso dal test case, ovvero la sequenza di dati in output o comandi che la corretta esecuzione del test dovrebbe far avere.
    - **Log**. È l'insieme delle correlazioni del comportamento osservato con il comportamento atteso per varie esecuzioni del test

Test case e relazioni

- Una volta che i test sono identificati e descritti si determinano le relazioni tra questi.
  - **Aggregation.** Usata quando un test case può essere decomposto in un insieme di subtest**.**
  - **Precedence.** 2 test case sono caratterizzati da questa relazione quando un test case deve precedere l'altro test case
- **Un buon modello di test deve contenere poche relazioni → velocizzare il processo di testing**
- Test case sono classificati in **blackbox** e **whitebox**.
  - **Blackbox**. Si focalizza sul comportamento I/O. Non si preoccupa della struttura interna della componente
  - **Whitebox**. Si focalizza sulla struttura interna della componente (non sul comportamento I/O): ogni stato nel modello dinamico dell'oggetto e ogni interazione tra gli oggetti viene testata.

Test stub and test driver

- Eseguire test case su una componente o una combinazione di componenti richiede che le componenti testate siano isolate dal resto del sistema
- **Test driver** e **test stub** sono usati per sostituire le parti mancanti del sistema
  - Un **test stub** è una implementazione parziale di componenti da cui la componente testata dipende (componenti che sono chiamate dalla componente testata).
  - Un **test driver** è una implementazione parziale di una componente che dipende dalla componente testata (componente che chiama la componente testata).

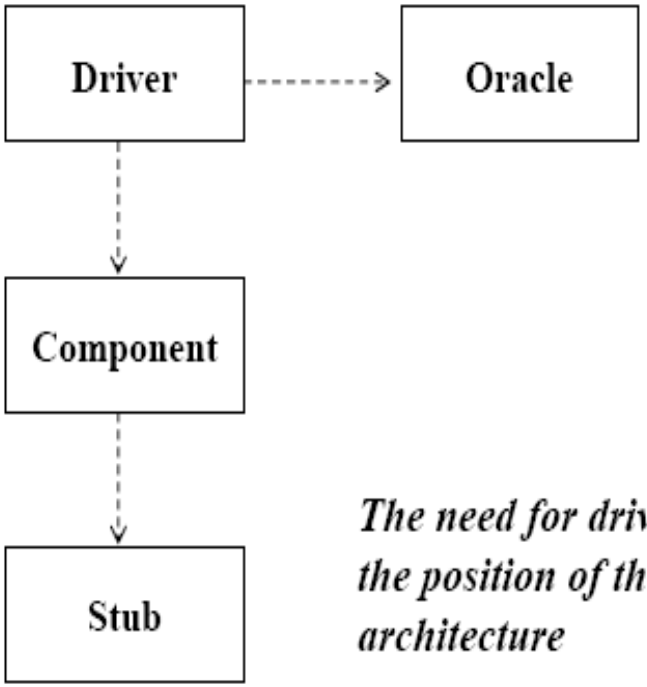
Test stub and test driver

- Un test stub deve fornire la stessa API del metodo della componente simulata e ritornare un valore il cui tipo è conforme con il tipo del valore di ritorno specificato nella signature.
  - **Se l'interfaccia di una componente cambia anche il corrispondente test driver e test stub devono cambiare → gestione delle configurazioni**
- L'implementazione di un test stub non è una cosa semplice.
  - **Non è sufficiente scrivere un test stub che semplicemente stampa un messaggio attestante che il test stub è iniziato**
    - **La componente chiamata deve fare un qualche lavoro, se il test stub non simula il giusto comportamento la componente testata potrebbe avere un failure non perché c'è un fault**
    - **Il test stub non può restituire sempre lo stesso valore**
  - **Compromesso, tra implementare un test stub accurato e sostituire il test stub con la componente reale**
    - **spesso test stub e test driver sono scritti dopo che la componente è stata realizzata (purtroppo se si è in ritardo non vengono scritti…)**

Il problema dello scaffolding

Driver, Stubs, and Scaffolding

The oracle knows the expected output for an input to the component to be compared with the actual output to identify failures



The need for drivers and stubs depends on the position of the component in the system architecture

Correzioni

- Quando i test sono stati eseguiti e i fallimenti sono stati rilevati, gli sviluppatori cambiano la componente per eliminare il fault sospettato
- Una **correzione** è un cambiamento di una componente. Lo scopo è riparare un fault.
- Una correzione potrebbe introdurre nuovi fallimenti. Molte tecniche possono essere usate per gestire tali difetti:
  - **Problem tracking.** Include la documentazione di ogni fallimento, stato di errore, e bug rilevato, la sua correzione, e la revisione delle componenti coinvolte nel cambiamento → consente di restringere la ricerca di nuovi fault
  - **Regression testing**. Riesecuzione dei test precedenti subito dopo il cambiamento. Per assicurare che le funzionalità che lavorano prima della correzione non sono state influenzate → particolarmente importante quando si utilizza un approccio iterativo allo sviluppo come nel caso OO
  - **Rational maintenance**. Include la documentazione delle motivazioni alla base dei cambiamenti, e le relazioni dietro le motivazioni nella revisione della componente → consente agli sviluppatori di evitare di introdurre nuovi fault analizzando le assunzioni utilizzate per la costruzione della componente

Sommario delle definizioni