Debugging

"Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it."

- *Brian Wilson Kernighan*

Debugging

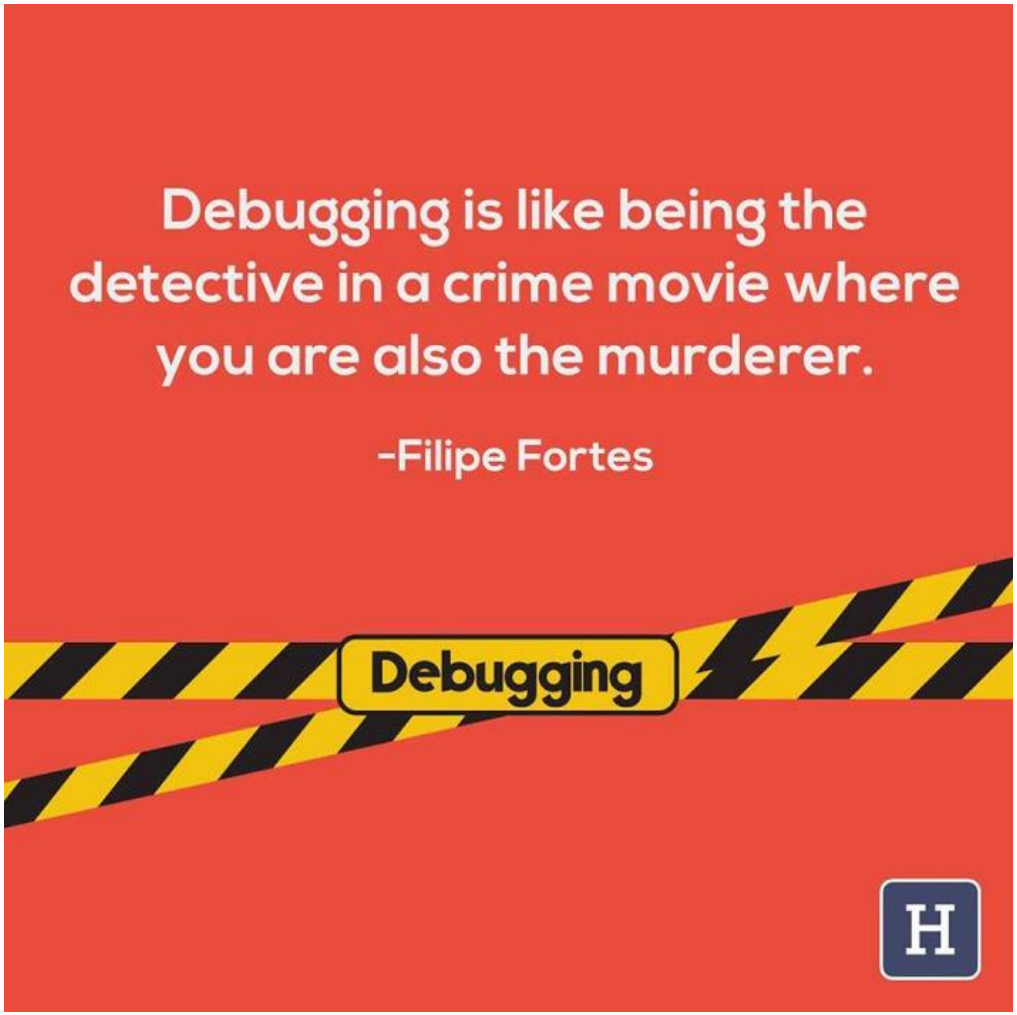
- Attività di ricerca e correzione dei difetti che sono causa di malfunzionamenti.
- É l'attività consequenziale all'esecuzione di un test che ha avuto successo e ha scoperto un malfunzionamento. Comprende due fasi:
  - **Ricerca del difetto**
  - **Correzione del difetto**
- Il debugging è ben lungi dall'essere stato formalizzato
  - Metodologie e tecniche di debugging rappresentano soprattutto un elemento dell'esperienza del programmatore/tester

Difficoltà del debugging

1. Il sintomo e la causa possono essere lontani.
2. Il sintomo può scomparire solo temporaneamente (a seguito di correzione di altro errore)
3. Il sintomo può non essere causato da errori specifici ma intrinseci all'ambiente di esecuzione (es.: errori di arrotondamento)
4. Può dipendere da errori di temporizzazione e non di elaborazione.
5. Può essere difficile riprodurre le condizioni di partenza
6. Il sintomo può essere intermittente.
7. Un solo difetto può causare più malfunzionamenti e un solo malfunzionamento può essere causato dall'azione combinata di più di un difetto

Localizzazione dei difetti

- Ridurre la distanza tra difetto e malfunzionamento
  - Mantenendo un'immagine dello stato del processo in esecuzione in corrispondenza dell'esecuzione di specifiche istruzioni
    - *Watch point e variabili di watch*
      - **Un watch, in generale, è una semplice istruzione che inoltra il valore di una variabile verso un canale di output**
        - » L'inserimento di un watch (sonda) è un'operazione invasiva nel codice: anche nel watch potrebbe annidarsi un difetto
        - » In particolare l'inserimento di sonde potrebbe modificare sensibilmente il comportamento di un software concorrente
    - *Asserzioni, espressioni booleane dipendenti da uno o più valori di variabili legate allo stato dell'esecuzione*


Debugging is like being the detective in a crime movie where you are also the murderer.
-Filipe Fortes
Debugging

Metodologie per il Debugging

- Forza Bruta
- Backtracking
- Eliminazione delle cause

"Forza Bruta"

- Il modo più inefficace per fare debugging. Diversi approcci possibili:
  - Usare storage dump (stampe dello stato della memoria in esadecimale o ottale…)
  - Disseminare il codice di sonde per catturare quante più informazioni possibili e valutarle, in cerca di indizi
  - Usare qualche strumento di debugging automatico (che permette di analizzare l'esecuzione del programma inserendo punti di break, osservazione di variabili, etc..)
    - *Largamente inefficace perché può produrre eccessive informazioni da comprendere*
- Ricorrervi solo quando altre tecniche hanno fallito!

Debugging per BackTracking

- Si cerca di ripercorrere il codice "all'indietro" a partire dal punto dove si è verificato il malfunzionamento (istruzione di output oppure eccezione)
- Analogamente alla tecnica delle Path Condition, diventa via via più difficile procedere all'indietro all'allargarsi del campo di possibilità.

Eliminazione delle cause

- **Prima di tutto, si individua la tipologia dei dati che fanno fallire il programma**

- **Poi si cerca di formulare un'ipotesi sulla possibile causa del difetto, proponendo dati in ingresso in grado di far avvenire il malfunzionamento, poi si cerca di controllare la validità di tale ipotesi**

Automatizzazione del debugging

- Il debugging è un'attività estremamente intuitiva, che però deve essere operata nell'ambito dell'ambiente di sviluppo e di esecuzione del codice
- Strumenti a supporto del debugging sono quindi convenientemente integrati nelle piattaforme di sviluppo (IDE), in modo da poter accedere ai dati del programma, anche durante la sua esecuzione, senza essere invasivi rispetto al codice
  - In assenza di ambienti di sviluppo, l'inserimento di codice di debugging invasivo rimane l'unica alternativa

Funzionalità di debugging

- **Inserimento break point**
  - Tramite Eclipse è possibile accedere a delle "Breakpoint properties"
    - *Breakpoint condizionali:*
      - **Breakpoint che si attivano solo quando si passa in una certa riga e si verifica una certa condizione**
    - *Breakpoint dipendenti dallo Hit Count:*
      - **Breakpoint che si attivano solo dopo un certo numero di passaggi su quella riga di codice**

- **Esecuzione passo passo del codice**
  - Entrando o meno all'interno dei metodi chiamati
  - Uscendo da un metodo verso il chiamante

Funzionalità di debugging

- **Valutazione (watch) del valore delle variabili, mentre l'esecuzione è in stato di interruzione**
  - Nei linguaggi interpretati (tra cui anche Java), è possibile anche fornire la possibilità di modificare in fase di esecuzione il valore di variabili, semplificando notevolmente il problema della ricerca di casi di test in grado di replicare il malfunzionamento.
    - *In questo modo possiamo testare anche cammini infeasible*

- **Inserimento Watchpoint**
  - Un Watchpoint si inserisce similmente ad un breakpoint
    - *Non si riferisce ad una riga di codice eseguibile, ma ad un attributo di una classe*
    - *Con I watchpoint si può eseguire il debugging:*
      - **fino a che quell'attributo non raggiunge un determinato valore ;**
      - **oppure fino a che l'attributo non è stato acceduto in lettura;**
      - **Oppure fino a che l'attributo non è stato modificato**
      - **Oppure fino a che l'attributo non è stato modificato un certo numero di volte**
      - **…**

Funzionalità di debugging

- **Similmente a breakpoint e watchpoint:**
  - Breakpoint per le eccezioni;
  - Breakpoint per il caricamento di classi;
  - Breakpoint per un metodo;

- **Ulteriori approfondimenti ed esempi:**
  - http://www.vogella.com/articles/EclipseDebugging/article.html

The internet will make those bad words go away

Essential

Googling the Error Message

O RLY?

The Practical Developer
@ThePracticalDev