Verifica e Validazione del Software

Testing White Box

Riferimenti

- Ian Sommerville, Ingegneria del Software, capitoli 22-23-24 (più dettagliato sui processi)
- Pressman, Principi di Ingegneria del Software, 5° edizione, Capitoli 15-16
- Ghezzi, Jazazeri, Mandrioli, Ingegneria del Software, 2° edizione, Capitolo 6 (più dettagliato sulle tecniche)

Testing Strutturale (White Box)

- Il Testing White Box è un testing strutturale, poichè utilizza la struttura interna del programma per ricavare i dati di test.

- **Tramite il testing White Box si possono formulare criteri di copertura più precisi di quelli formulabili con testing Black Box**

  - Test White Box che hanno successo possono fornire maggiori indicazioni al debugger sulla posizione dell'errore

Criteri di Copertura

- Fondate sull'adozione di metodi di Copertura degli oggetti che compongono la struttura dei programmi:
- *istruzioni – strutture controllo – flusso di controllo -…*
- definizione di un insieme di casi di test (input data) in modo tale che gli oggetti di una definita classe (es. istruzioni, archi del CFG, predicati, strutture di controllo, etc.) siano attivati (coperti) almeno una volta nell'esecuzione dei casi di test

Criteri di Copertura e relative Misure di Test Effectiveness

- Criteri di selezione
- **Copertura dei comandi (statement test)**
- **Copertura delle decisioni (branch test)**
- **Copertura delle condizioni (condition test)**
- **Copertura delle decisioni e delle condizioni**
- **Copertura dei cammini (path test)**
- **Copertura dei cammini indipendenti**

- Criteri di adeguatezza
- **n.ro comandi eseguiti/ n.ro comandi eseguibili**
- **n.ro archi percorsi/ n.ro archi percorribili**
- **n.ro cammini percorsi/ n.ro cammini percorribili**
- **n.ro cammini indip. percorsi/n.ro ciclomatico**

UN MODELLO DI RAPPRESENTAZIONE DEI PROGRAMMI: il Control-Flow Graph

- Il grafo del flusso di controllo (Control-Flow Graph) di un programma P:
  - **CFG (P) = <N, AC, nI, nF>**

dove:

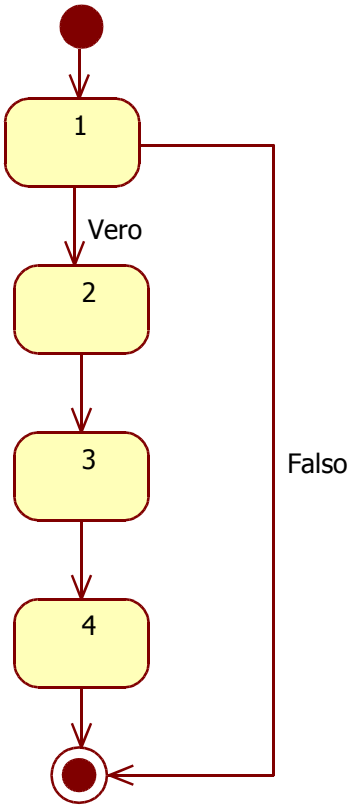
- <N, AC> **è un grafo diretto con archi etichettati,**
- $\{nI, nF\} \subseteq N$, $N - \{nI, nF\} = Ns \cup Np$
- Ns e Np **sono insiemi disgiunti di *nodi istruzione* e *nodi predicato*;**
- $AC \subseteq N-\{nF\} \times N-\{nI\} \times \{vero, falso, incond\}$ **rappresenta la relazione flusso di controllo;**
- nI ed nF **sono detti rispettivamente *nodo iniziale* e *nodo finale*.**

•Un nodo $n \in Ns \cup \{nI\}$ **ha un solo successore immediato e il suo arco uscente è etichettato con incond.**

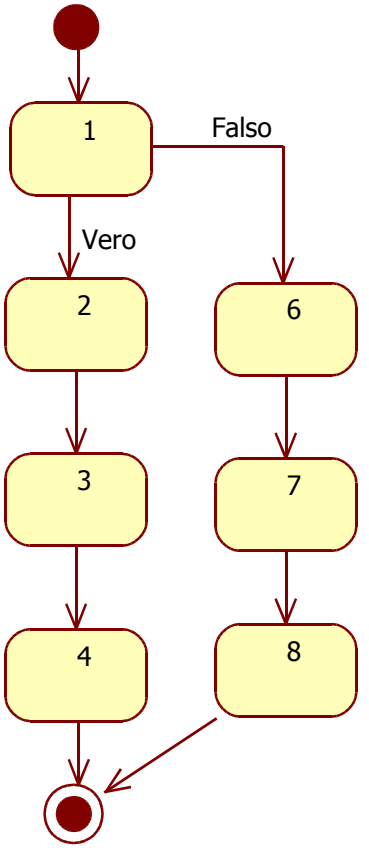
•Un nodo $n \in Np$ **ha due successori immediati e i suoi archi uscenti sono etichettati rispettivamente con vero e falso.**

Strutture di controllo e CFG

```
1.   if
     (decisione)
2.   {
3.     Blocco
4.   }
```



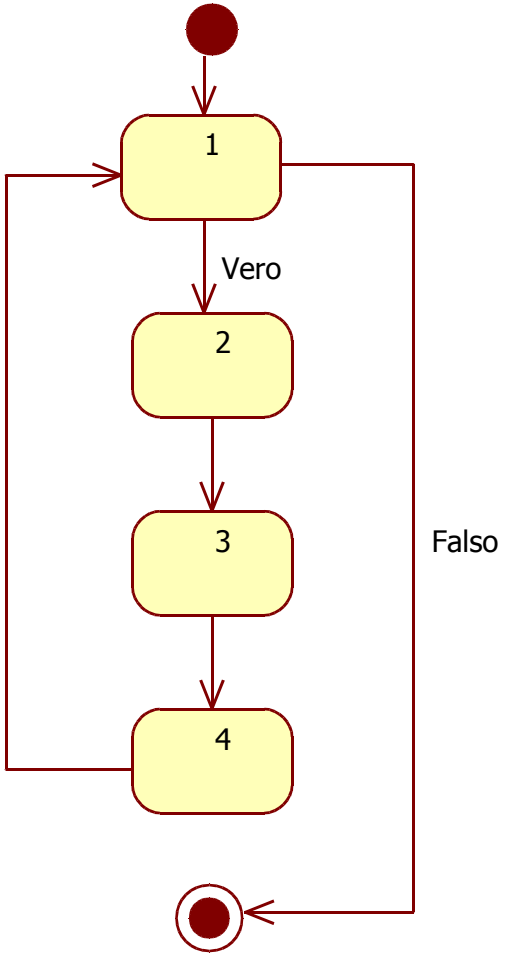

```
1.   if
     (decisione
     )
2.   {
3.     Blocco1
4.   }
5.   else
6.   {
7.     Blocco2
8.   }
```



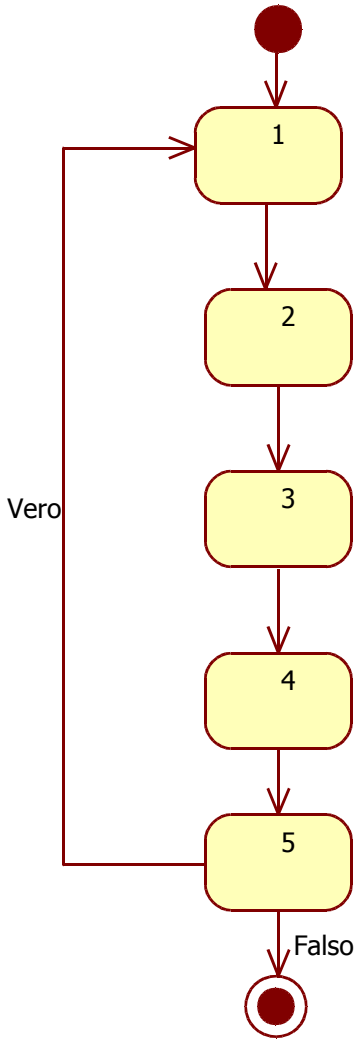
I nodi 2, 4, 6, 8 potevano anche essere omessi, dato che ad essi non corrisponde alcuna istruzione esecutiva, nel codice eseguibile del programma.
Uno switch può essere risolto trasformandolo (come fa il compilatore) in una serie di if else in cascata.

Strutture di controllo e CFG

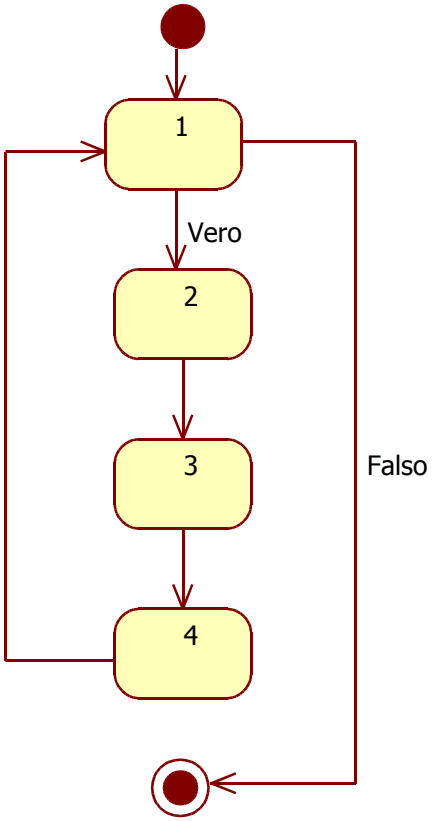
```
1.   while
     (decisione)
2.   {
3.     Blocco
4.   }
```



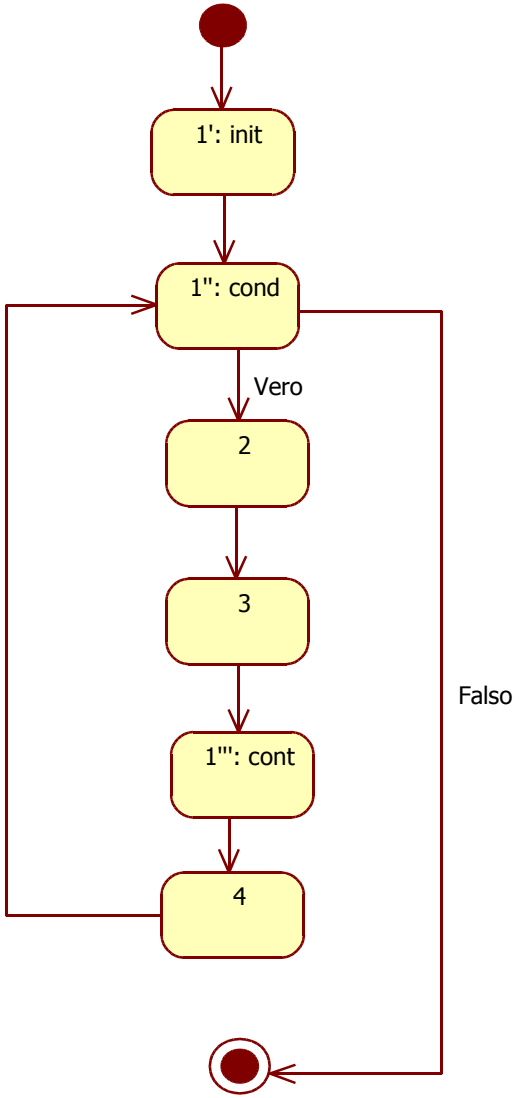

```
1.   do
2.   {
3.     Blocco
4.   }
5.   While
     (decisione
     )
```



Ciclo For e CFG



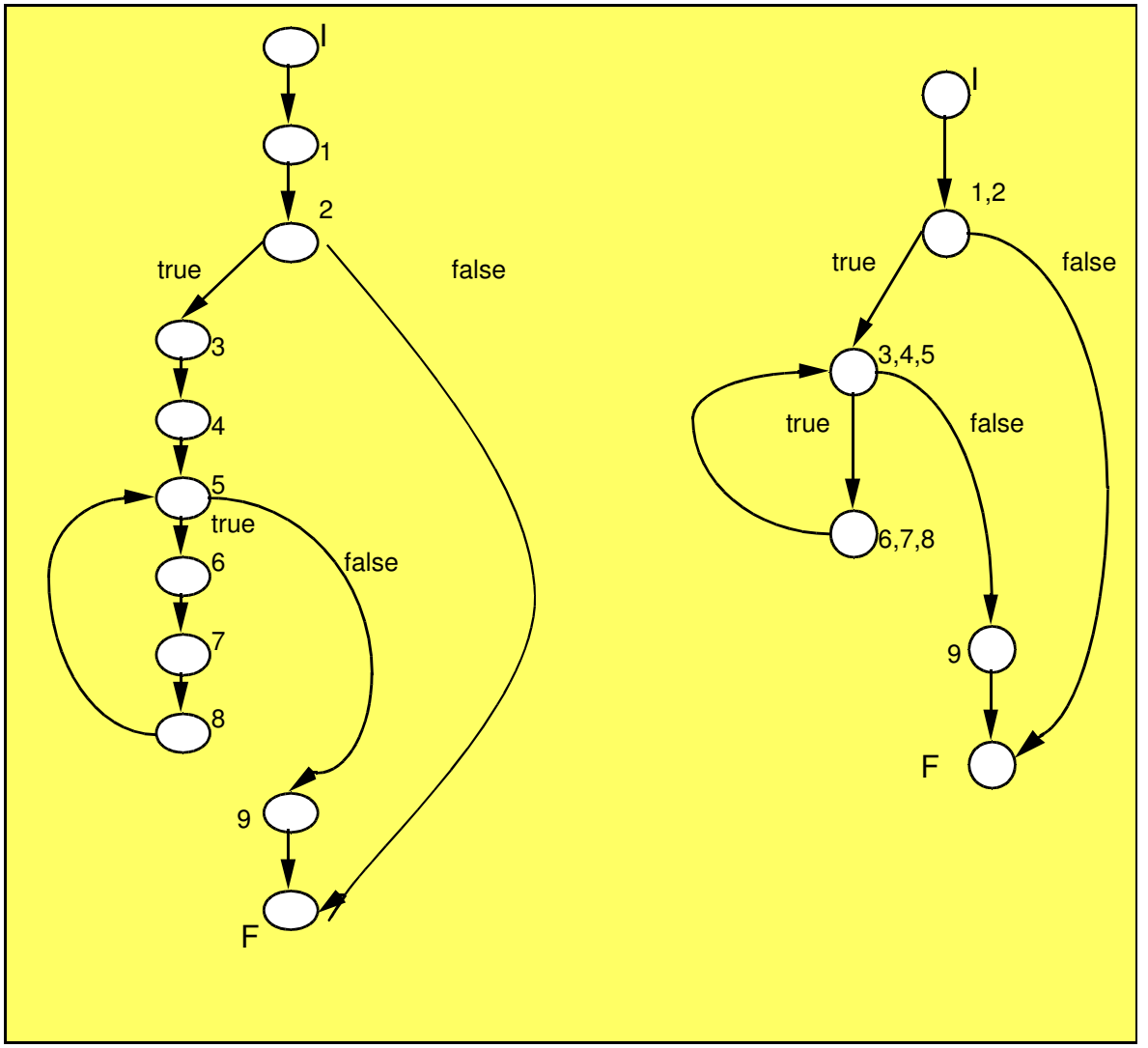

```
1.   for
     (init;decisione;cont)
2.   {
3.     Blocco
4.   }
```



Il CFG a destra esprime più precisamente la semantica del ciclo for, così come viene trasformato in codice oggetto da un compilatore C. Infatti ad ogni ciclo for corrisponde un init preventivo, una decisione all'inizio di ogni ciclo (come in un while) e un codice di continuazione cont (ad esempio i++) da ripetere prima di ricominciare il ciclo

Un esempio

```
procedure Quadrato;
  var x, y, n: integer;
  begin
1.   read(x);
2.   if x > 0
     then begin
3.          n := 1;
4.          y := 1;
5.          while x > 1 do
             begin
6.             n := n + 2;
7.             y := y + n;
8.             x := x - 1;
            end;
9.          write(y);
        end;
  end;
```



Criteri di copertura

- Copertura dei comandi (statement test)
  - Richiede che ogni nodo del CFG venga eseguito almeno una volta durante il testing;
  - è un criterio di copertura debole, che non assicura la copertura sia del ramo true che false di una decisione.
  - Es. nella Procedura quadrato, la test suite TS={x=2} garantisce la copertura di tutti i nodi, ma non dell'arco (1,2 → Fine)

Criteri di copertura

- Copertura delle decisioni (branch test)
  - Richiede che ciascun arco del CFG sia attraversato almeno una volta;
    - *In questo caso ogni decisione è stata sia vera che falsa in almeno un test case*
    - *Nella procedura Quadrato, la TS={x=2, x=-1} garantisce la copertura delle decisioni*
  - un limite è legato alle decisioni in cui più condizioni (legate da operatori logici AND ed OR) sono valutate

Copertura delle condizioni (condition test)

- Ciascuna condizione nei nodi decisione di un CFG deve essere valutata sia per valori true che false.
- Esempio:

```
int check (x);// controlla se un intero è fra 0 e 100
int x;
{    if ((x>=0) && (x<= 200))
      check= true;
     else check = false;
     }
```

TS={x=5, x=-5 } valuta la decisione sia per valori True che False, ma non le condizioni

TS1={ x= 3, x=-1, x=199, x=210} è una Test suite che copre tutte le condizioni

Copertura delle condizioni e decisioni

- Occorre combinare la copertura delle condizioni in modo da coprire anche tutte le decisioni.

- Es. If (x>0 && y>0) …
  - TS1={(x=2, y=-1), (x=-1, y=5)} copre le condizioni ma non le decisioni!
  - TS2={(x=2, y=1), (x=-1, y=-55)} copre sia le condizioni che le decisioni!
  - TS3={(x=2, y=1), (x=2, y=-1), (x=-2, y=1)(x=-1, y=-55)} copre tutte le combinazioni di tutte le condizioni (quindi anche tutte le condizioni e tutte le decisioni

Nota a margine

- Gli esempi precedenti riguardavano codice sorgente di linguaggi di alto livello
- In realtà, il CFG reale e le relative metriche di copertura dovrebbero essere valutate, per massimizzare la precisione, sul codice macchina generato dal codice sorgente
  - Fa eccezione il caso in cui vogliamo effettuare analisi statica, senza testing, direttamente di algoritmi, anziché di programmi
  - Per conoscere l'esatto codice macchina dobbiamo, però, conoscere le caratteristiche della macchina target e del compilatore
    - *Ad esempio, l'adozione di tecniche di ottimizzazione dei compilatori possono portare a variazioni dei CFG*
- In conclusione, la misura di metriche di copertura sul codice di alto livello rappresenta una approssimazione, non sempre affidabile, delle analoghe misure sul codice eseguibile

Cammini linearmente indipendenti (McCabe)

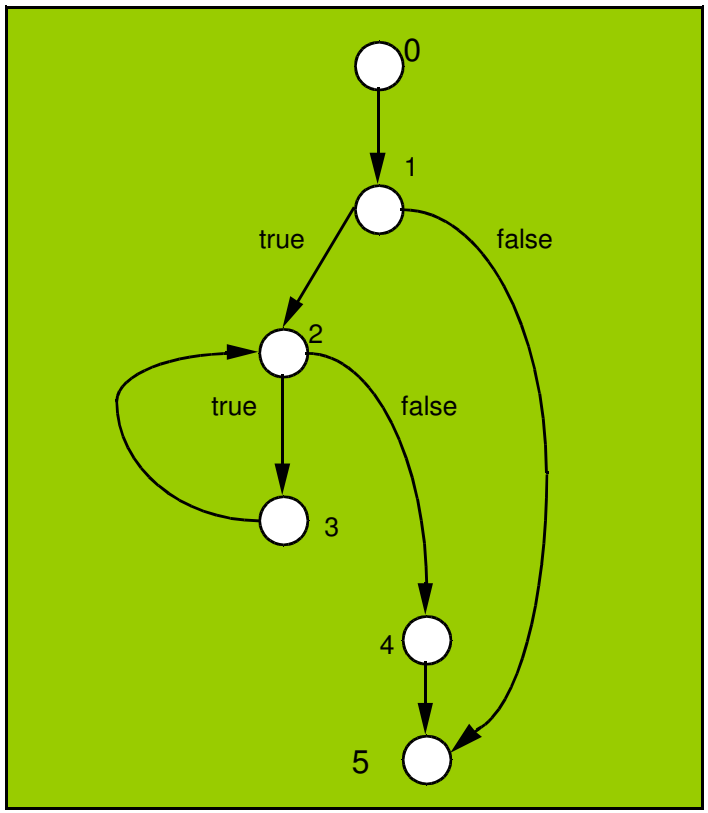
- Un cammino è un'esecuzione del modulo dal nodo iniziale del CFG al nodo finale
- Un cammino si dice indipendente (rispetto ad un insieme di cammini) se introduce almeno un nuovo insieme di istruzioni o una nuova condizione
  - in un CFG un cammino è indipendente se attraversa almeno un arco non ancora percorso
- L'insieme di tutti i cammini linearmente indipendenti di un programma forma i cammini di base; tutti gli altri cammini sono generati da una combinazione lineare di quelli di base.
- Dato un programma, l'insieme dei cammini di base non è unico.

Numero di cammini linearmente indipendenti

- Il numero dei cammini linearmente indipendenti di un programma è pari al numero ciclomatico di McCabe:
  - V(G) = E - N +2
    - *Dove E: n.ro di archi in G - N: n.ro di nodi in G*
  - V(G) = P + 1
    - *Dove P: n.ro di predicati in G*
  - V(G) = n.ro di regioni chiuse in G + 1

- Test case esercitanti i cammini di base garantiscono l'esecuzione di ciascuna istruzione almeno una volta
- La copertura dei cammini linearmente indipendenti garantisce la copertura di tutti i cammini se non consideriamo il numero di volte in cui ogni ciclo può essere eseguito

Esempio



Complessità ciclomatica del programma è 3

V(G)= 3 =>3 cammini indipendenti

c1= 0-1-2-4-5
c2= 0-1-**2-3-2**-4-5
c3= 0-**1-5**

Ogni nuovo cammino è ottenuto negando una decisione di un cammino precedente e coprendo, quindi, almeno un nuovo arco

Criteri di copertura dei cammini

- Copertura dei cammini (path test)
  - spesso gli errori si verificano eseguendo cammini che includono particolari sequenze di nodi decisione
  - non tutti i cammini eseguibili in un CFG possono essere eseguiti durante il test (un CFG con loop può avere infiniti cammini eseguibili)
- Copertura dei cammini indipendenti
  - ci si limita ad eseguire un *insieme di cammini indipendenti* di un CFG, ossia un insieme di cammini in cui nessun cammino è completamente contenuto in un altro dell'insieme, nè è la combinazione di altri cammini dell'insieme
  - ciascun cammino dell'insieme presenterà almeno un arco non presente in qualche altro cammino
  - il numero di cammini indipendenti coincide con la complessità ciclomatica del programma

Relazioni tra i criteri di copertura

- La copertura delle decisioni implica la copertura dei nodi
- La copertura delle condizioni *non sempre* implica la copertura delle decisioni
- La copertura dei cammini linearmente indipendenti implica la copertura dei nodi e la copertura delle decisioni
- La copertura dei cammini è un test ideale ed implica tutti gli altri

Problemi

- E' possibile riconoscere automaticamente quale cammino linearmente indipendente viene coperto dall'esecuzione di un dato test case

- E' indecidibile il problema di trovare un test case che va a coprire un dato cammino
  - Alcuni cammini possono risultare non percorribili (*infeasible),* ma non è, in generale, possibile sapere se un cammino è percorribile

- La copertura dei cammini linearmente indipendenti non garantisce da errori dovuti, ad esempio, al numero di cicli eseguiti, per i quali sarebbe necessaria la copertura di tutti i cammini, che però rappresenta il testing esaustivo!
  - La copertura dei cammini linearmente indipendenti coincide con la copertura dei cammini in un programma senza cicli o in un programma in cui ogni ciclo è percorso sempre lo stesso numero di volte

CodeCover

- CodeCover è un plug-in open source per Eclipse realizzato dall'università di Stoccarda e reperibile a: http://codecover.org/index.html
- **Consente di valutare metriche di copertura**
  - CodeCover measures statement, branch, loop, term coverage (subsumes MC/DC), question mark operator coverage, and synchronized coverage.
- Scritto per Java e Cobol
- Si può utilizzare sia in modalità standalone che sotto Eclipse

CodeCover per Eclipse

- Può essere installato semplicemente aggiungendolo come plug-in aggiuntivo reperibile all'indirizzo:
  http://update.codecover.org/

- Un tutorial d'esempio è reperibile all'indirizzo:

  http://codecover.org/documentation/tutorials/how_to_complete.html

CodeCover Features

- Può misurare tanto la copertura per esecuzioni di un'applicazione per la quale sia stato attivato preventivamente il monitoraggio, sia per esecuzioni di test case scritti con Junit

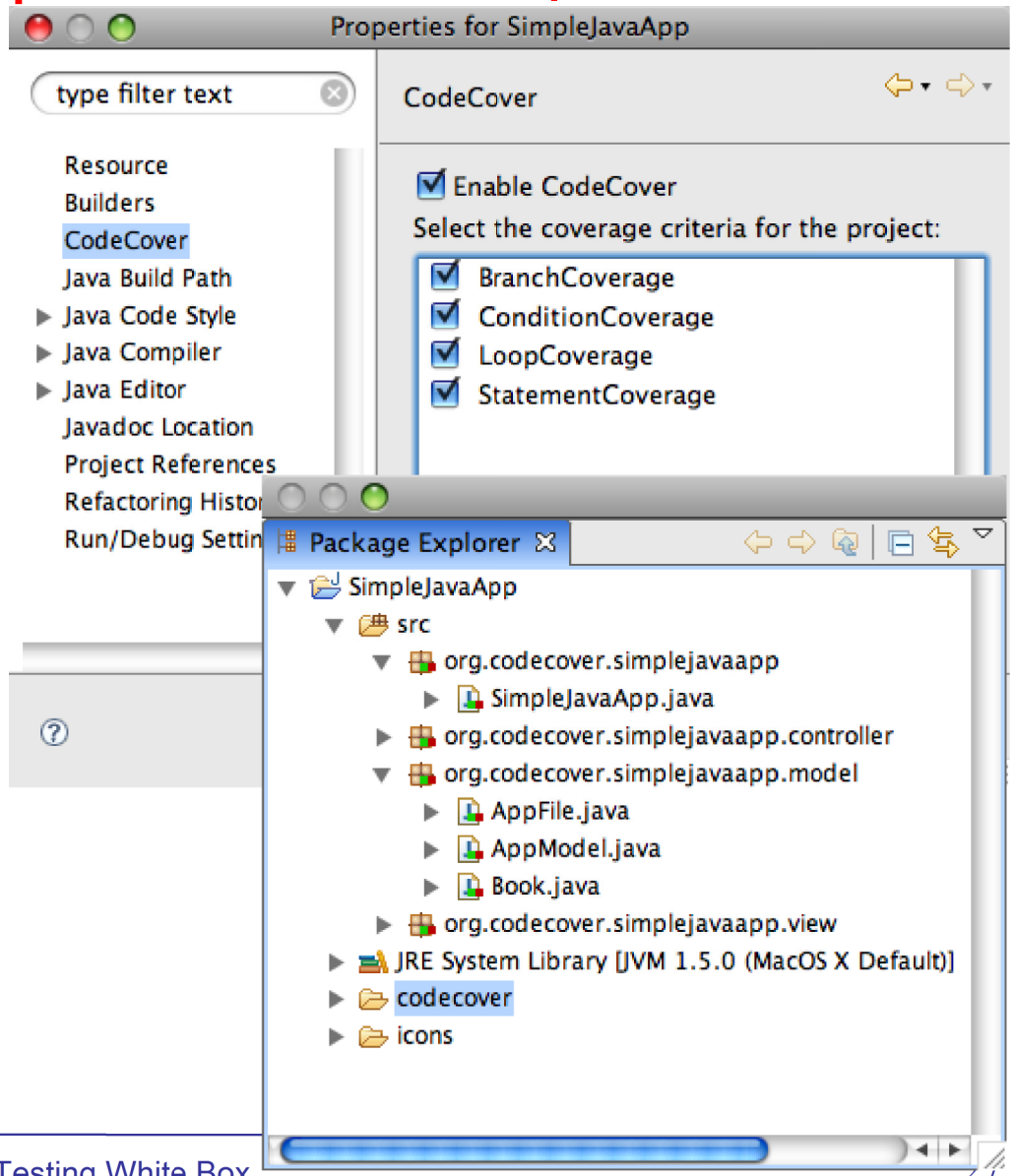
- Mostra molte statistiche riguardanti l'esito dei casi di test e consente di generare report HTML con tali risultati

Eclipse Control Flow Graph Generator

- http://eclipsefcg.sourceforge.net/
- E' un'estensione open source di Eclipse che consente di disegnare CFG di metodi di applicazioni Java
- Può essere installato semplicemente aggiungendolo come plug-in aggiuntivo reperibile all'indirizzo: **http://eclipsefcg.sourceforge.net/**
- Per utilizzarlo basta selezionare l'opzione CFG Generator dal menu contestuale di un qualsiasi metodo
- Si integra anche con CodeCover

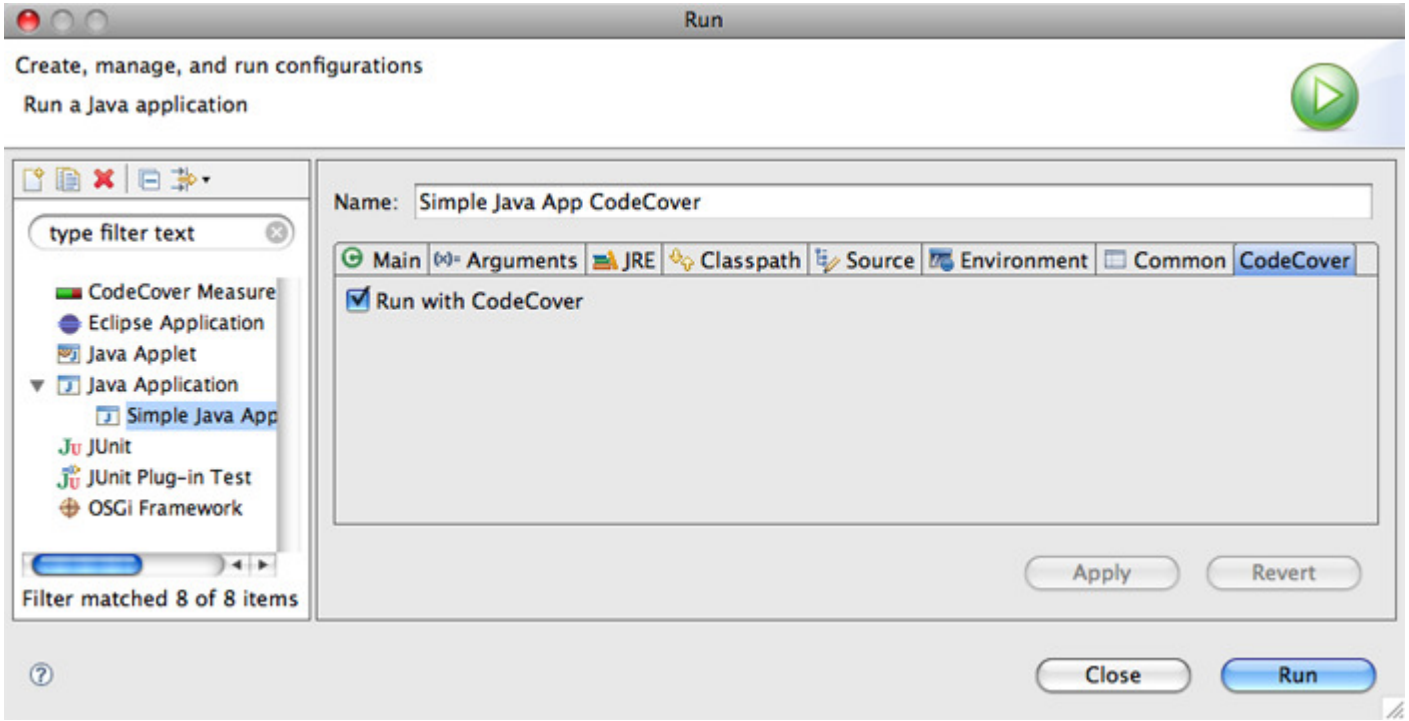
Mini Tutorial per CodeCover 1/5

- Abilitare CodeCover tra le proprietà del progetto
  - Causa l'instrumentazione del bytecode
- Nel menu contestuale relativamente ad ogni package/file, abilitare la misura



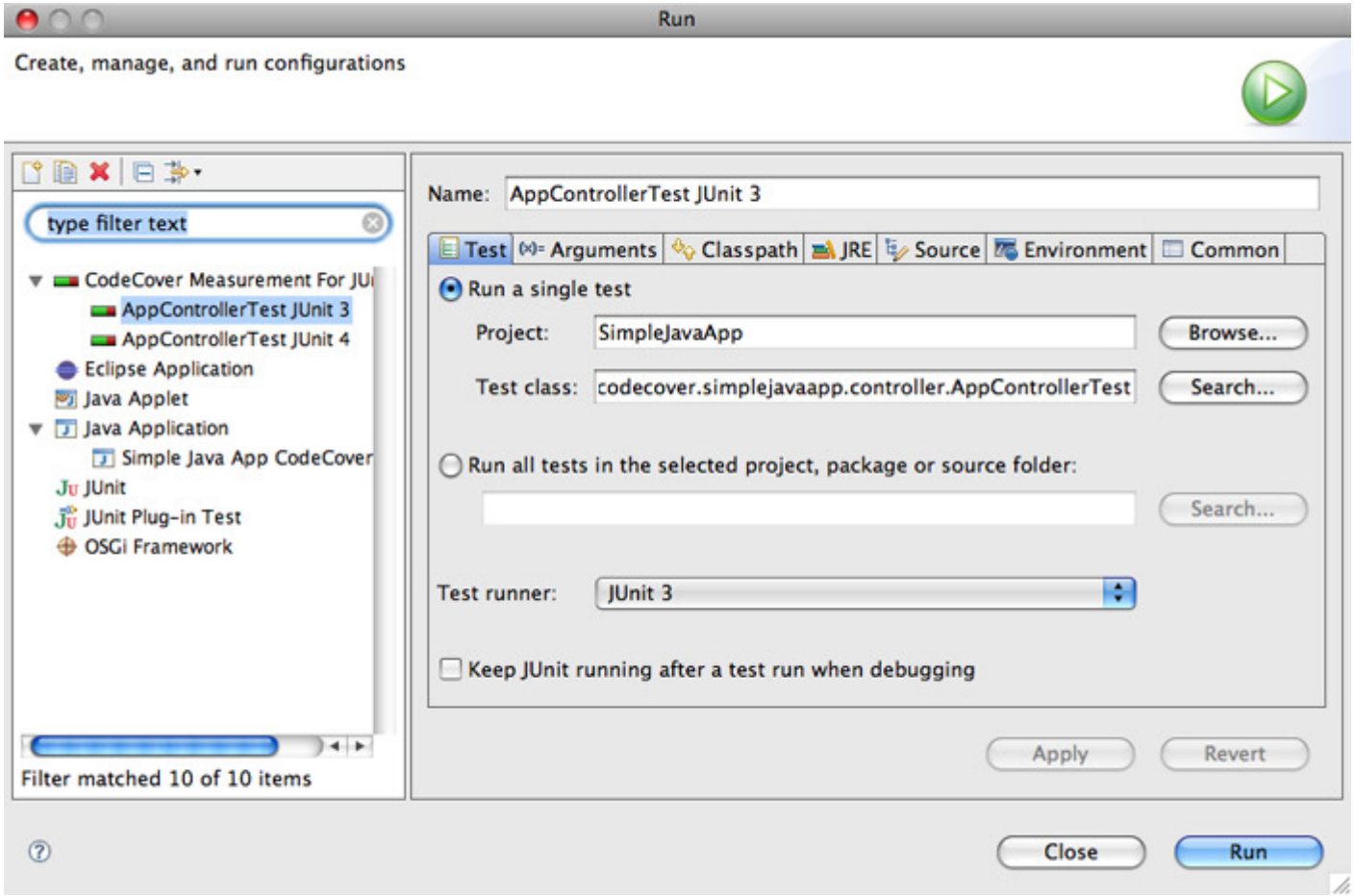
Mini Tutorial per CodeCover 2/5

- Nelle proprietà del profilo di esecuzione abilitare CodeCover
- Abilitare da Windows/Show View/Other le varie view di Codecover:
  - Test Sessions
  - Coverage
  - Correlation
  - …



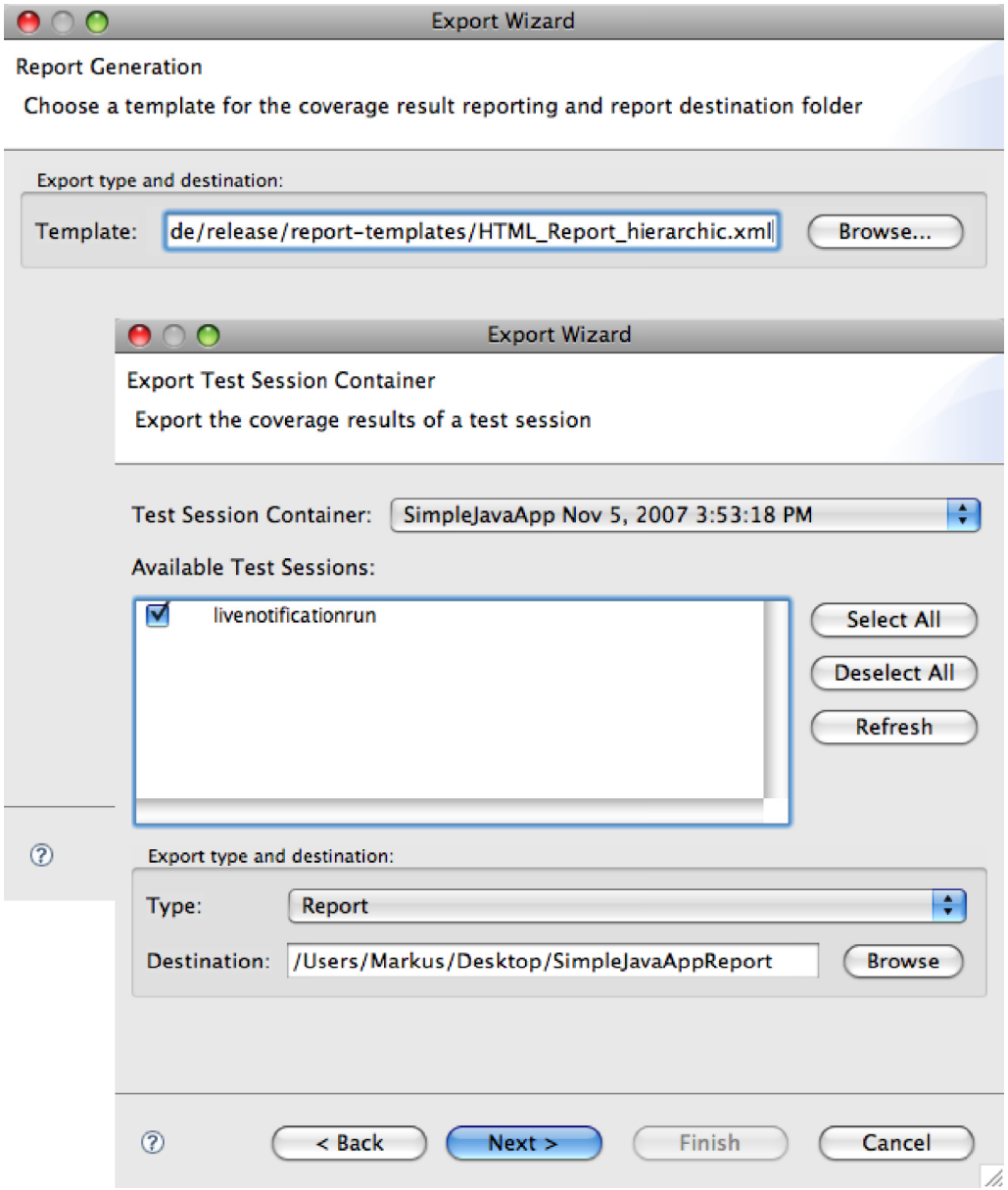
Mini Tutorial per CodeCover 3/5

- Per conoscere la copertura raggiunta con test case Junit è sufficiente creare un profilo di esecuzione dei test sotto CodeCover Measurement for JUnit



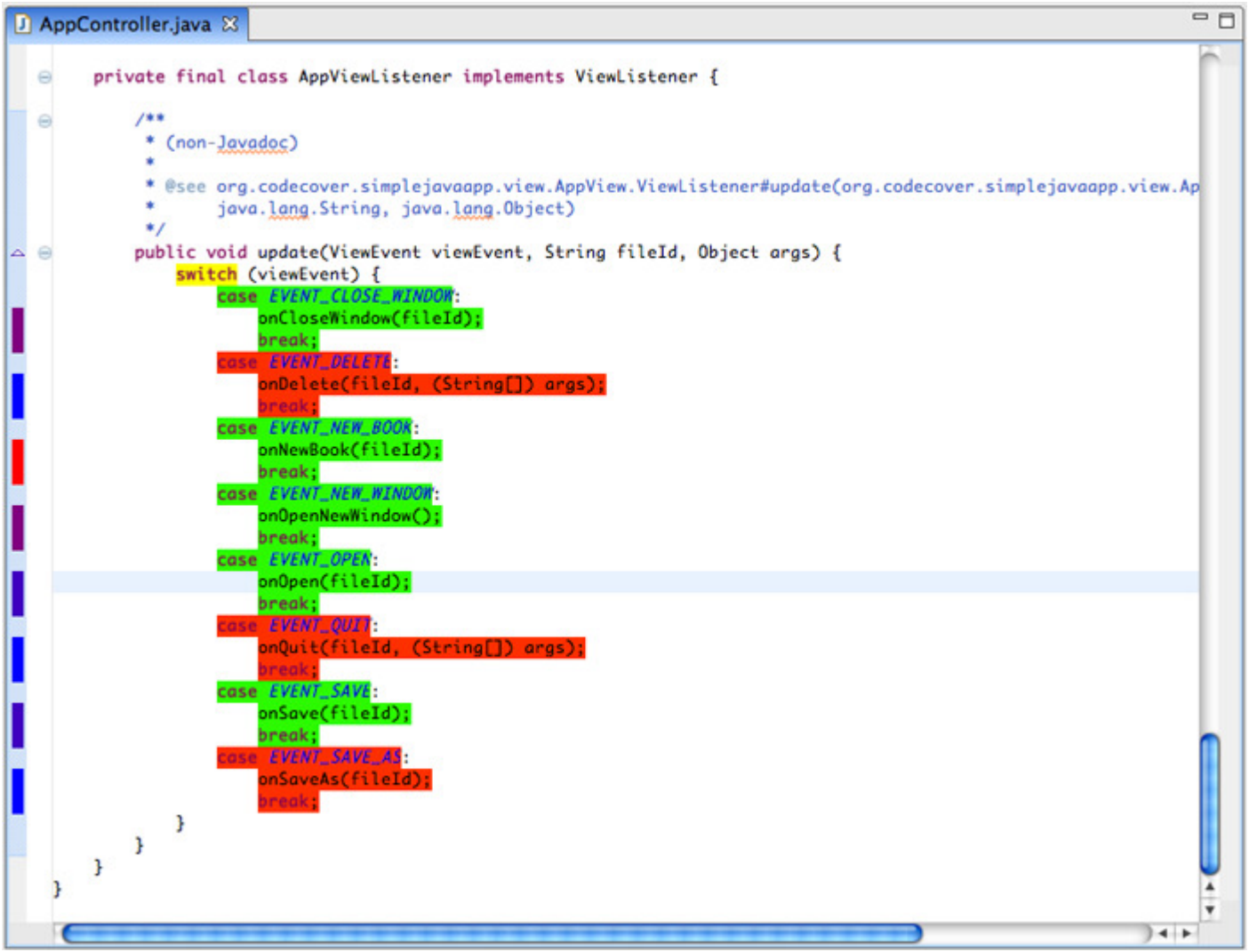
Mini Tutorial per CodeCover 4/5

- Per esportare i risultati in HTML è sufficiente utilizzare il wizard di esportazione in File/Export



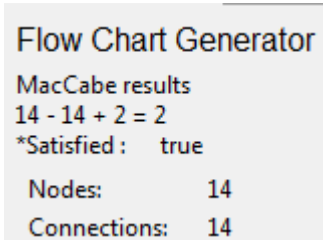
Mini Tutorial per CodeCover 5/5

- La copertura delle righe del codice può essere vista anche direttamente sul codice
- Rosso: non coperta
- Giallo: coperta parzialmente (decisione)
- Verde: coperta

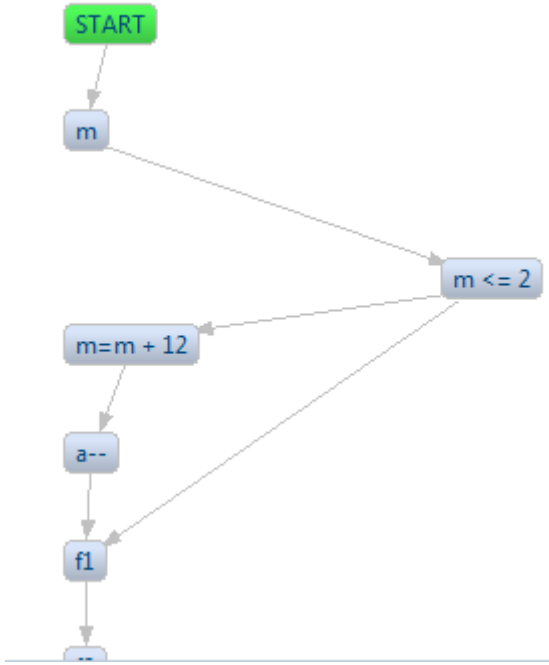


Mini Tutorial per CFG Generator

- E’ sufficiente scegliere CFG Generator/ Build nel menu contestuale di un metodo



- In alternativa, è anche possibile generare il CFG da una sessione di test la cui copertura è stata salvata con CodeCover



Esercizio

- Per la classe calendar, visualizzare il livello di copertura ottenuto con i test JUnit progettati in precedenza
- Scrivere ulteriori casi di test in grado di coprire totalmente, secondo i diversi criteri di copertura studiati, gli elementi del metodo calend
- Sfruttare, eventualmente, le informazioni di copertura per trovare i difetti relativi agli eventuali malfunzionamenti trovati