Model-based testing

7 December 2016

De unde pot proveni modelele?

- din explorarea sistemului
- din specificație
- din cod

De la modele la teste

În toate cazurile, trebuie o mapare de la acțiunile/răspunsurile din model la intrările/răspunsurile sistemului supus testării (SUT)

Exemplu: Web Application Abstract Language [Büchler et al., KIT/TU München]
1) *Acțiuni abstracte* în browser: *FollowLink, ClickButton, SelectItems, ClickImage, gotoURL, InputText, MoveMouse*, etc.

2) Mapare în acțiuni *specifice aplicației testate*:

```
login(user, pwd) =
  selectItem(employeeList, user);
  inputText(passwordField, pwd);
  clickButton(login);
```

3) Mapare în acțiuni ale cadrului de testare (ex. Selenium):
`HtmlUnit.findElement()`, `WebElement.click()`

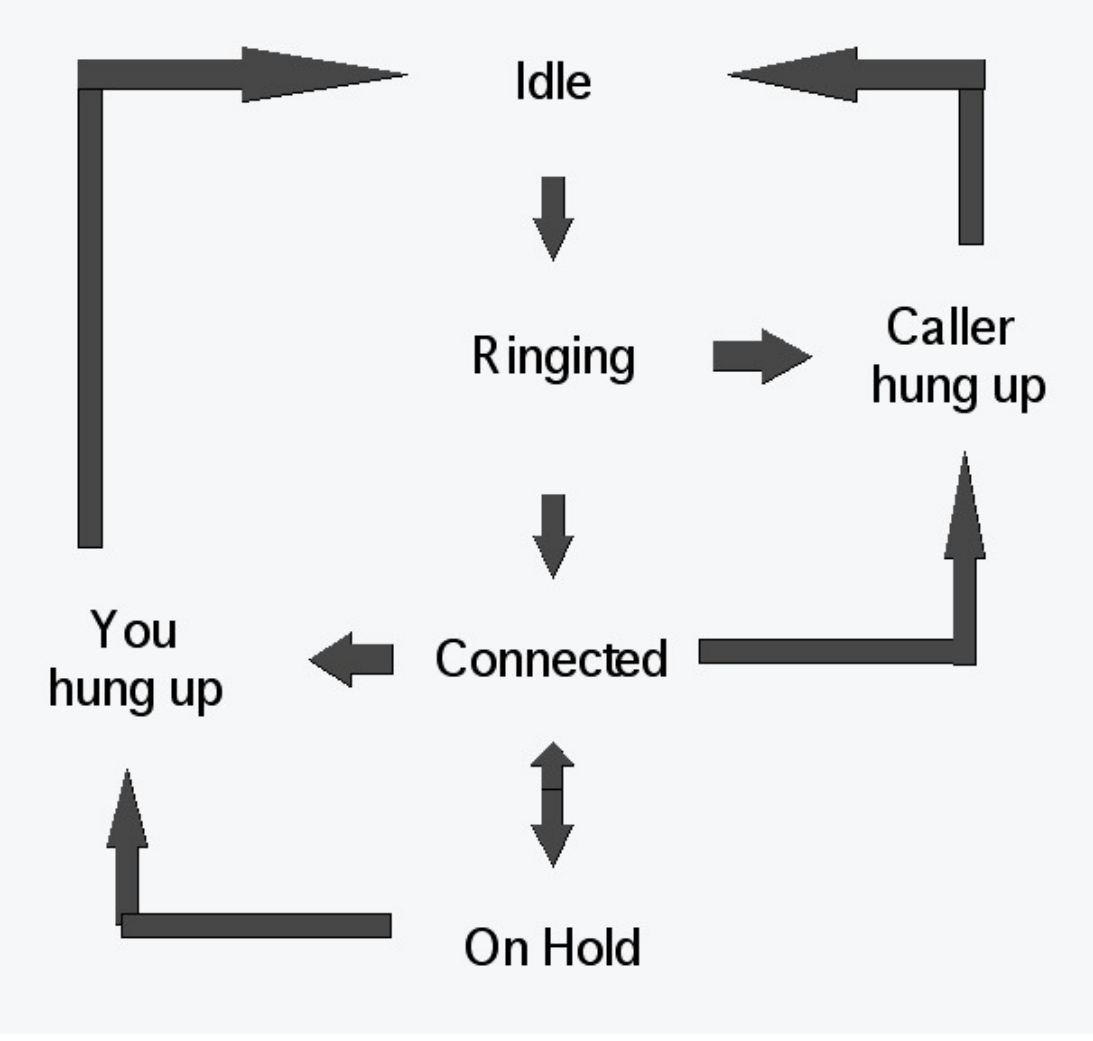
Modele obținute prin explorarea sistemului

Informal: testare prin explorare
  ex.: ceasul (Robinson), simulator transfer date (Bach)
Generarea modelului: manual
Testarea pentru confirmarea/infirmarea modelului: automat

Formal: învățarea unui automat (*active learning*, alg. Angluin)
  se generează secvențe de intrări, observând ieșirile
Dacă secvențe de intrări i_1, i_2 cu același sufix: $i_1 s_i$ și $i_2 s_i$
produc ieșiri cu același sufix $o_1 s_o$, $o_2 s_o$, pentru toate sufixele s_i
până la o limită, atunci i_1 și i_2 duc în aceeași stare
Rafinare succesivă la fiecare infirmare

Modele obținute din specificație

Exemplu: centrală telefonică [Kaner]



Scrise de regulă manual

Modele obținute din specificație: bytecode Java

Discuție:

Instrucțiunile bytecode Java lucrează cu stiva
pentru operanzi și rezultat.

De exemplu, `iadd` transformă stiva `x:y:rest` în `(x+y):rest`

Pentru a verifica corectitudinea de tip, e suficient să modelăm *tipurile* valorilor din stivă, și nu valorile în sine

$\Rightarrow$ avem $\texttt{int:int:rest} \xrightarrow{\texttt{iadd}} \texttt{int:rest}$

$\Rightarrow$ obținem un model cu efectul fiecărei instrucțiuni

Modele obținute din cod

```
do {    // Fragment de device driver [Ball & Rajamani '01]
  KeAcquireSpinLock(&devExt->writeListLock);
  nPacketsOld = nPackets;
  request = devExt->WriteListHeadVa;
  if(request && request->status) {
    devExt->WriteListHeadVa = request->Next;
    KeReleaseSpinLock(&devExt->writeListLock);
    irp = request->irp;
    if (request->status > 0) {
      irp->IoStatus.Status = STATUS_SUCCESS;
      irp->IoStatus.Information = request->Status;
    } else {
      irp->IoStatus.Status = STATUS_UNSUCCESSFUL;
      irp->IoStatus.Information = request->Status;
    }
    SmartDevFreeBlock(request);
    IoCompleteRequest(irp, IO_NO_INCREMENT);
    nPackets++;
  }
} while (nPackets != nPacketsOld);
KeReleaseSpinLock(&devExt->writeListLock);
```

Abstractizarea codului

```
do {
A: KeAcquireSpinLock();
  b = T;      /* b == (nPackets == nPacketsOld) */
  if(*) {
B:   KeReleaseSpinLock();
    if (*) {
      skip;
    } else {
      skip;
    }
    b := choose(F, b);      /* choose(p1, p2) == p1 ?  T :
p2 ?  F : nondet */
  }
} while (!b);
C: KeReleaseSpinLock();
```

Abstractizarea se face folosind reguli Hoare / precondiții Dijkstra.

Abstractizare din cod: JML model fields

Câmpuri "fictive", reprezintă relații între câmpuri reale din cod

Fiecare metodă e anotată cu precondiții / postcondiții / invarianți, exprimate *relativ la* (câmpurile din) *model*

`http://kindsoftware.com/products/opensource/ESCJava2/ESCTools/slides/ETAPSTutorial/5_more_jml.pdf` (p. 35-45)