Univerza v Ljubljani

Fakulteta za računalništvo in informatiko

Igor Rožanc

Testiranje OO in spletnih tehnologij

Povzetek testiranja

Testna orodja

Praktični vidiki in prihodnost testiranja

Študijsko gradivo za interno uporabo pri predmetu Testiranje in kakovost (TiK)

Ljubljana, 2017/18

Kazalo

Objektno usmerjeni programi

A PIE

Glavne značilnosti za testiranje:

- Konstruktorji
- Dedovanje
- Omejevanje dostopa
- Polimorfizem

Vrste testiranja:

- znotraj iste metode
- med metodami
- znotraj razreda
- med razredi

Objektno usmerjeni programi

Acronym	Fault / Anomaly
ITU	Inconsistent Type Use
SDA	State Definition Anomaly
SDIH	State Definition Inconsistency
SDI	State Defined Incorrectly
IISD	Indirect Inconsistent State Definition
ACB1	Anomalous Construction Behavior (1)
ACB2	Anomalous Construction Behavior (2)
IC	Incomplete Construction
SVA	State Visibility Anomaly

Objektno usmerjeni programi

Uporaba pristopa DEF – USE…

Novi kriteriji za testirenje:

Vsa zaporedja ujemanja - All-Coupling-Sequences (ACS) : Za vsako zaporedje ujemanja S_j iz f() obstaja vsaj en testni primer t, kjer je pot ujemanja povzročena s $S_{j,k}$ ki je del poti sledi izvedbe f(t)

- najmanj ena pot
- ne upošteva dedovanja in polimorfizma

Vsi polimorfizni in razredi - All-Poly-Classes (APC) : Za vsako zaporedje ujemanja $S_{j,k}$ iz metode *f()* in za vsak razred definiran s kontekstom $S_{j,k,}$ obstaja vsaj en testni primer *t*, kjer (ko je f() izveden z t) je pot *p* v množici poti ujemanj $S_{j,k}$, ki je del poti sledi izvedbe *f(t)*

- najmanj en test za vsak tip (razred)
- preveri vsako nadomeščanje tipov

Objektno usmerjeni programi

Vsa Def_Use ujemanja - All-Coupling-Defs-Uses (ACDU) : Za vsako spremenljivko *v* v zaporedju ujemanja $S_{j,k}$ it *t* obstaja pot ujemanja iz $S_{j,k}$ kjer je *p* del poti sledi izvedbe *f(t)* za vsaj en primer *t.*

- vsak zadnji def doseže prvi use
- ne upošteva dedovanja in polimorfizma

Vsi polimorfizni in Def- Use ujemanja - All-Poly-Coupling-Defs-and-Uses (APDU) : Za vsako zaporedje ujemanja $S_{j,k}$ iz *f()*, za vsak razrediz konteksta *Sj,k*, za vsako spremenljivko *v* iz Sj,k, za vsako vozlišče *m* z zadnjo definicijo *v* in vsako vozlišče *n* z prvo uporabo *v*, obstaja vsaj en testni primer *t* kjer (ko je *f()* izveden za *t)* obstaja pot *p* med potmi ujemanja *Sj,k* ki je del poti pri sledi izvedbe *f()*

- vsak zadnji def doseže prvi use za vsak razred
- upošteva dedovanje in polimorfizem

Objektno usmerjeni programi

6

Hierarhija:

APDU

APC **ACDU**

ACS

Spletni programi

7

Značilnosti:

- Program objavljen na svetovnemu spletu (dosegljivost)
- Uporaba HTML-ja (CSS, JavaScript,)
- HTTP: vsak par zahteva-odgovor je neodvisen (stateless protocol)
- Spletne aplikacije: statične ali dinamične
- Spletni servisi: programi (brez UI), ki uporabljajo SOAP (XML)

Tehnične značilnosti:

- Neodvisne, šibko povezane (loosely coupled) komponente
- Delujejo na principu konkurence
- Majhne komponente, a veliko število
- Številne (nove) tehnologije

Spletni programi

8

Spletne aplikacije so heterogene, dinamične in porazdeljene

Zaradi razširjenosti visoke zahteve po kakovosti

Težave:

- Klasični grafi niso uporabni
- Obnašanje je težko opisljivo
- Vsi vhodi preko UI – slab nadzor
- Omejen dostop do strežnikov – slaba opazovalnost
- Ni jasno, kako uporabiti logične predikate
- Mutacijsko testiranje ?

Spletni programi

9

Nove vrste težav:

1) Porazdeljenost in izrazito ohlapna povezanost

- Internet, raznolika strojna in programska oprema, strežniki …
- Tight: Loose: Extremely Loose Coupling

2) Dinamična tvorba HTML form

- Spremenljiv uporabniški vmesnik glede na uporabnika
- Različni podatki glede na uporabnika

3) Spremenljiv tok izvajanja

- Gumb Back, Forward, Refresh
- Zlonamerno spreminjanje URL-ja

4) Dinamično sestavljanje komponent med izvajanjem

- Vstavljena javanska zrna (EJB)
- Spletni servisi se dinamično povezujejo z drugimi servisi

Spletni programi

A) Testiranje statičnih spletnih aplikacij (mest)

- Preverjamo dosegljivost statičnih povezav
- Preverjamo izvedljivost (obremenitveno testiranje)
- Pomembne so tudi performanse
- Možno je preverjati pravice do dostopa

Uporabimo predstavitev z grafom:

- Vozlišče je spletna stran
- Povezava je klic druge HTML strani (link)
- Uporabimo enostavne grafne kriterije (NC, EC)

Spletni programi

B) Testiranje dinamičnih spletnih aplikacij

- Uporabniški vmesnik in del programa (JS) je na odjemalcu
- Večji (ključni) del programa je na strežniku
- Testiramo ločeno uporabniški (client) in strežniški (server) del:
  - **B1) Client-side testing: ni dostopa do strežnika**
  - **B2) Server-side testing: vidimo le strežniški del**

B1) Testiranje na strani odjemalca:

- Testiramo uporabniški vmesnik - dejansko vhode vanj
- Cilj je poiskati vse zaslone, ko je spletna aplikacija v nedovoljenem stanju
- Vhodi definirani z elementi HTML form
- Semantic Domain Problem: tri tehnike za generiranje vhodov
  - Posebej pripravljeni s strani testerja (izvedljivo, a drago)
  - Avtomatsko generirani (random ali vse vrednosti - težko)
  - Pripravljeni na podlagi zgodovine (nepopolno, zahteva razumevanje)
- **Bypass testing**: Za lažjo izvedbo (avtomatizacijo) – spremenimo vir, izpustimo ali zaobidemo kontrole
- Podoben pristop uporablja SQL injection

Primer bypass testiranja:

```
<FORM >
  <INPUT Type=“text” Name=“username” Size=20>
  <INPUT Type=“text” Name=“age” Size=3 Maxlength=3>
  <P> Version to purchase:
    …
  <INPUT Type=“radio” Name=“version” Value=“150” Checked>
  <INPUT Type=“radio” Name=“version” Value=“250 25” Checked>
  <INPUT Type=“radio” Name=“version” Value=“500”>
  <INPUT Type="submit" onClick="return checkInfo(this.form)" >
  <INPUT Type=“hidden”  isLoggedIn=“noyes”>
</FORM>
```

Spletni programi

Primer za SQL injection:

Uporabnik: up1´ OR ´1´=´1 **Geslo: abc ´ OR ´1´=´1**

Pravi SQL:

```
SELECT user
FROM adminuser
WHERE username='up1'
AND
password ='abc'
```

Spremenjen SQL:

```
SELECT user
FROM adminuser
WHERE username='up1' OR '1' ='1'
AND
password ='abc'OR '1' ='1'
```


Spletni programi

B2) Testiranje na strani strežnika:

- Potreben je dostop do strežnika!
- Testiramo kodo, a običajni pristopi niso primerni
- Določimo uporabne dele in sestavimo model dinamične spletne aplikacije
- Samo za predstavitveni nivo aplikacije, na dveh ravneh abstrakcije:
  1. **Component Interaction Model (CIM)**
     - **Za posamezne komponente**
     - **Kombinira elementarne dele (atomic sections)**
  2. **Application Transition Graph (ATG)**
     - **Vsako vozlišče jeen CIM**
     - **Povezave so prehodi med CIM-i**

Elementarni deli 16

	ID = request.getParameter ("Id");
	passWord = request.getParameter ("Password");
	retry = request.getParameter ("Retry");

P1 =	out.println (“<HTML> <HEAD><TITLE>" + title + "</TITLE></HEAD><BODY>)"
	if ((Validate (ID, passWord)) {
P2 =	out.println (“ <B> Grade Report </B>");
	for (int I=0; I < numberOfCourse; I++)
P3 =	out.println(“ " + courseName (I) + "“ + courseGrade (I) + “ ”);
	} else if (retry < 3) {
P4 =	retry++; out.println ("Wrong ID or wrong password"); out.println ("<FORM Method=\“get\" Action=\"gradeServlet\">); out.println ("<INPUT Type=\“text\" Name=\"Id\" Size=10>"); out.println ("<INPUT Type=\“password\" Name=\"Password\" Width=20>"); out.println ("<INPUT Type=\“hidden\" Name=\"Retry\" Value=" + (retry) + ">");
	} else if (retry >= 3) {
P5 =	out.println (“ Wrong ID or password, retry limit reached. Good bye.") }
P6 =	out.println(“</BODY></HTML>");

CIM 17

S = login.html

$A = \{p_1, p_2, p_3, p_4, p_5, p_6\}$

CE = gradeServlet = $p_1 \bullet ((p_2 \bullet p_3^*) \mid p_4 \mid p_5) \bullet p_6$

$T = \{$*login.html* $\longrightarrow$ *gradeServlet [get, (Id, Password, Retry)]*,

gradeServlet.p_4 $\longrightarrow$ *sendMail [get, ()]*,

gradeServlet.p_4 $\longrightarrow\!\!>$ *gradeServlet [get, (Retry)]* $\}$

ATG 18

Γ = { login.html, gradeServlet, sendMail, syllabus.html }

Θ = {*login.html* $\longrightarrow$ *syllabus.html [get, ()]*,

login.html $\longrightarrow\!\!>$ *gradeServlet [get, (Id, Password, Retry)]*,

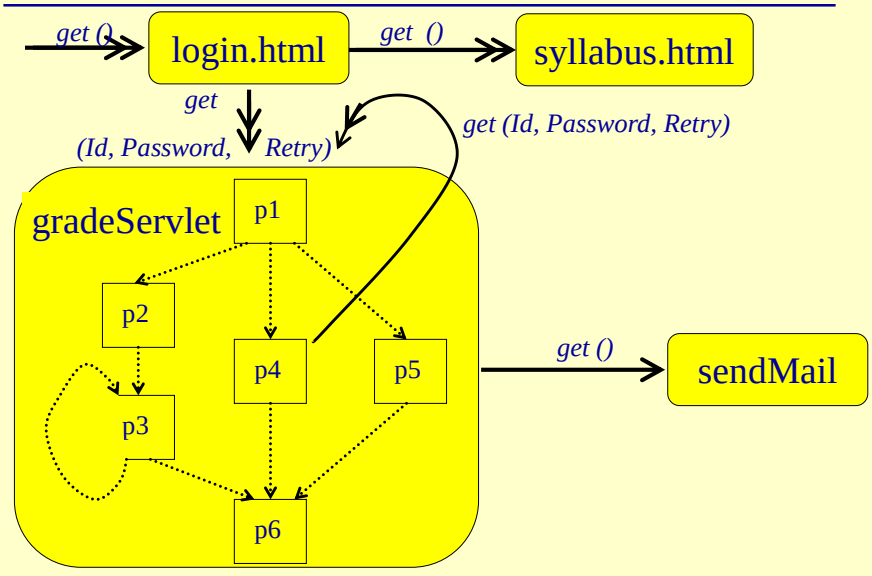
gradeServlet.p_4 $\longrightarrow$ *sendMail [get, ()]*,

gradeServlet.p_4 $\longrightarrow\!\!>$ *gradeServlet [get, (Retry)]* }

Σ = { *Id, Password, Retry* }

α = { login.html }

ATG (grafično) 19



Spletni programi

C) Spletne storitve:

- Dodatne težave:
  - ni uporabniškega vmesnika
  - ni dostopa do opisa strukture, virov,
  - strukturiran prenos podatkov (XML)
- Cilj je testirati sporočila (prenose)
- Uporabimo lahko pristope za testiranje sintakse

Povzetek pokritij

Štirje abstraktni modeli pokritij:

1. pokritje z grafi :

- **Koda, specifikacije, načrt, use cases**
- **11 kriterijev, 5 važnih**

2. pokritje z logični izrazi :

- **Koda, specifikacije, FSM, DNF**
- **13 kriterijev, 3 važni**

3. delitev prostora vhodnih podatkov :

- **6 kriterijev, 3 važni**

4. sintaksno testiranje :

- **7 kriterijev, 3 važni**

Pokritje grafov

22

1.Kriterij pokritja vozlišč (NC)

2.Kriterij pokritja povezav (EC)

3. Kriterij pokritja parov povezav (EPC)

4. Kriterij pokritja primitivnih poti (prime path coverage - PPC)

5. Kriterij pokritja enostavnih krožnih poti (simple round trip coverage - SRTC)

6. Kriterij pokritja vseh krožnih poti (complete round trip coverage - CRTC)

Pokritje grafov

23

7. Kriterij pokritja vseh poti (complete path coverage - CPC)

8. Kriterij pokritja izbranih poti (specified path coverage - SPC)

9. Kriterij All-Defs pokritja (ADC)

10. Kriterij All-Uses pokritja (AUC)

11. Kriterij pokritja vseh DU-poti (ADUPC)

Pokritje logičnih izrazov

1.Kriterij pokritja predikatov (PC)

2. Kriterij pokritja trditev (CC)

3.Kriterij pokritja kombinacij (CoC)

4. Kriterij pokritja aktivnih trditev (ACC)

5. Kriterij splošnega pokritja aktivnih trditev – General Active Clause Coverage (GACC)

6. Kriterij omejenega pokritja aktivnih trditev – Restricted Active Clause Coverage (RACC)

7. Kriterij povezanega pokritja aktivnih trditev – Correlated Active Clause Coverage (CACC)

Pokritje logičnih izrazov

8. Kriterij pokritja neaktivnih trditev – Inactive Clause Coverage (ICC)

9. Kriterij splošnega pokritja neaktivnih trditev – General Inactive Clause Coverage (GICC)

10. Kriterij omejenega pokritja neaktivnih trditev – Restricted Inactive Clause Coverage (GICC)

11. Kriterij pokritja izrazov DNF – Inplicant Coverage (IC)

12. Kriterij pokritja enoličnih TRUE točk – Unique True Points Coverage (UTPC)

13. Kriterij pokritja parov enolične TRUE točke in bližnjih FALSE točk – Unique True Point and Near False Point Pairs Coverage (CUTPNFP)

Pokritje prostora vhodnih podatkov

1.Kriterij pokritja vseh kombinacij – All Combinations (ACoC)

2. Kriterij pokritja vseh izbir – Each Choice Criteria (EC)

3. Kriterij pokritja parov – Pair-Wise Criteria (PW)

4.Kriterij pokritja t-kombinacij – t-Wise (TW)

5. Kriterij osnovnega pokritja – Base Choice (BC)

6. Kriterij večkratnega osnovnega pokritja – Multiple Base Choice (MBC)

Pokritje sintakse

1. Kriterij pokritja terminalnih simbolov – Terminal Symbol Coverage (TSC)

2. Kriterij pokritja produkcij – Production Coverage (PDC)

3. Kriterij pokritja izpeljav – Derivation Coverage (DC)

4. Kriterij pokritja mutacij – Mutation Coverage (MC)

5. Kriterij šibkega pokritja mutacij – Weak Mutation Coverage (WMC)

6. Kriterij pokritja operatorjev mutacij – Mutation Operator Coverage (MOC)

7. Kriterij pokritja produkcij mutacij – Mutation Production Coverage (MPC)

Orodja

Tri vrste orodij:

- **Komercialna**
  - Upravljanje testiranja
  - Enostavna pokritja
  - Metrike
- **Privatna, ne komercialna**
  - Tehnično bolj dovršena, bolj popolna
  - Nekaj za kodo, pogosto specifična za določen namen
- **Razvojna (znanstvena)**
  - Napredne tehnike, generatorji
  - Zelo močna, a nepopolna
  - Pogosto za kodo, splošnonamenska

Orodja

Tipična orodja:

- Analizatorji pokritij
- Generatorji grafov
- Metrike
- Podpora za instrumentiranje
- Avtomatizacija izvajanja testov
- Snemanje in predvajanje vhoda
- Vnos STUB in MOCK objektov
- Generiranje testnih primerov
- Generiranje mutacij

Orodja

Instrumentiranje kode:

- **Namen: vstavimo posebne dele v kodo za analizo pokritosti kode**
- Tvorimo več kopij
- Ne sme spreminjati funkcionalnosti
- Lahko ima vpliv na čas izvajanja
- Najzahtevnejši je ustrezen izpis rezultatov
- Tipično: tabela int vrednosti kot globalna spremenljivka
- Primeri: NC, EC, CACC, DU

Praktični vidiki testiranja

Pokritja so zgolj orodje, "pravo" testiranje zahteva učinkovito vgraditev in uporabo

Smisel: doseči 3. in 4. nivo po Bezieru (kakovost)

Ključni poudarki:

- Regresijsko testiranje
- Integracija testiranja
- Testni proces
- Testni plan
- Prepoznavanje pravih rezultatov

Praktični vidiki testiranja

1. Regresijsko testiranje

- Dejansko razvijamo zelo malo nove PO - najbolj pogosto
- Ni časa - avtomatizirano (veliko orodij – replay, version, skripte)
- Veliko testov – kako jih obvladovati
  - Vključitev - kriteriji pokritja, en test za vsak problem
  - Izključitev – izven kriterija, ne najde napake (!), obstaja podoben (!)
- Veljaven/neveljaven neuspeh reg. Testiranja
- Nujna analiza vpliva sprememb (CIA):
  - Katere teste izvajati: konzervativen / poceni / realističen pristop
  - Algoritmi za prikaz sprememb: vključujoči / natančni / učinkoviti /splošni

Praktični vidiki testiranja

2. Integracija testiranja

- Ključnega pomena za uspešno testiranje
- Več pristopov:
  - Big Bang (vse naenkrat)
  - Postopno (dva dela, del za delom)
- Uporaba nadomestnih objektov:
  - gonilnik (avtomatsko)
  - nadomestek (stub, mock): konstante / random / tabela / vnosi ...
- Vrstni red integracije in testiranja razredov (CITO):
  - hierarhija / OO
  - zahteva ogrodje za pripravo vhodov/tolmačenje rezultatov

Praktični vidiki testiranja

3. **Testni proces**
- **Kakovost** > učinkovitost
- Določene testne aktivnosti v fazah razvoja:
  - Zahteve (cilji, celotni plan testiranja)
  - Načrtovanje (tvori sistemske in uporabniške teste / plani za int. in testiranje enot, orodja / specifikacije testiranja)
  - Izvedba (tvori in izvajaj teste enot)
  - Integracija (izvajaj int. teste)
  - Vgradnja (izvajaj sistemske in uporabniške teste)
  - Uporaba in vzdrževanje (reg. testiranje, uporabnikove težave)
- Hramba rezultatov testiranja
- Etika: pošteno,vzrok:posledica, kar vgradiš testiraj, pravočasno, brez bližnjic

Praktični vidiki testiranja

4. **Testni plan**
- **Vsebina** > struktura
- Standard ANSI/IEEE 829-1983: opisuje namen, pristop, vire in razpored aktivnosti, določa dele za testiranje, značilnosti, testna opravila, izvajalce in tveganja ob tem.
- Vrste:
  - Mission plan: why, eden, krovni
  - Strategic plan: what/when, en za vrsto projekta, splošen
  - Tactic plan: how/who, za vsek projekt, podroben (zahteve, orodja, izvajalci, vrstni red, rezultati)

5. Prepoznavanje pravilnih rezultatov

- Problem: ali je rezultat pravilen:
  - Neposredno preverjanje (človeška napaka - postopek)
  - Dodatno računanje
  - Preverjanje konistentnosti
  - Dodatni podatki

- V 80 – 90 letih testiranje enostavno ni bilo tako pomembno: enostavnejše tehnologije, majhni programi, nižja pričakovanja
- Sprememba po 2000: večji programi, različne tehnologije (mreža, vgrajeni sistemi), uporaba povsod, večja tekmovalnost, pričakovanja glede varnosti
- Ključen problem – računalniška varnost:
  - 80 – kriptografija,
  - 90 – zaščita DB
  - 2000 – omrežna varnost
  - sedaj – napake v programih >> testiranje !!!
- Razvoj testiranja:
  - 80 - kriteriji+unit testi
  - 90 - kriteriji za druge tehnologije+nivoji
  - 2000 - avtomatizacija, več tehnologij
  - sedaj – celovit in enoten pogled, a veliko še manjka

Prihodnost

Testabilnost (testability):

- **Opisuje zmožnost programa za testiranje**
- Razmerje testiranje : testabilnost

Splošno:

Testabilnost = | # vhodov, ki povzročajo napake| / | # vseh vhodov| * 100%

Tehnično:

Za vsako vrstico X v programu P:

R = % vhodov, ki dosežejo x

I = % vhodov, ki sprožijo napako v x

P = % vhodov, ki prikažejo napako v x

Testabilnost = funkcija (R * I * P) za vse X iz P

Prihodnost

Odprti problemi v testiranju:

- Testiranje novih tehnologij
- Avtomatsko generiranje testov
- Določanje testabilnosti, zanesljivosti
- Raziskovanje regresijskega testiranje
- Uporaba testov v industriji

Prihodnost testiranja:

- Specializacija testiranja – bolj učinkovito testiranje
- Več tehničnih znanj za testne ekipe in zagotavljanje kakovosti
- Več testnih znanj in motivacije tudi pri razvijalcih
- Agilni pristopi zahtevajo določanje testov pred razvojem
- Zlivanje področij testiranja in varnosti

Literatura

1. Paul Ammann, Jeff Offutt: Introduction to software testing, Cambridge University Press, 2008.
2. Wikipedia