
MONITORIZACIÓN DE PRuebas

Título proxecto: Monitorización Pruebas VVS

Ref. proxecto: Join Me!

Alumnos: Andrea Ardións, Adrián Leira

Grupo: 1.2

Validación e Verificación de Software

Data de aprobación	Control de versións	Observacións
22/12/2016	1.0	

1. Contexto

Neste documento temos unha suite de probas que aplicamos ao noso proxecto. Esta suite de probas axudounos a atopar erros e bugs potenciais na práctica que non atoparíamos de non empregar as probas.

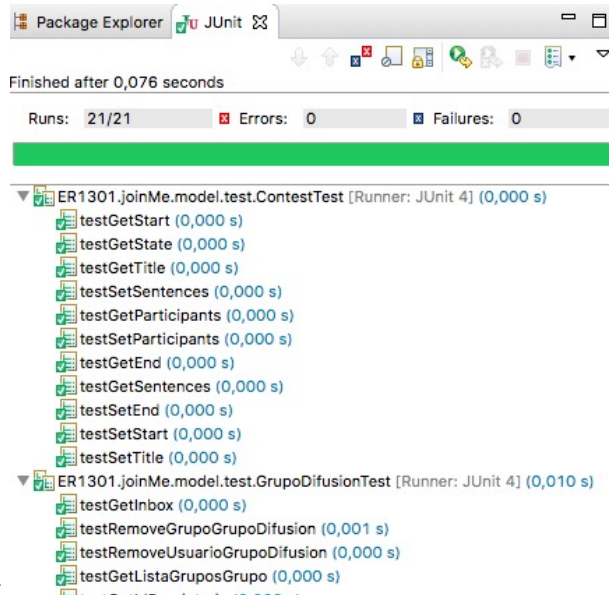
2. Estado actual

Ainda que tivemos problemas o principio coa migración de proxecto a Maven, co tempo puidemos ir axustando o pom.xml e engadindo todas las librerías necesarias pra executar as probas que levamos a cabo.

Con estas probas, queremos mellorar a calidade do proxecto y resolver posibles erros que non hubéramos atopado de no ser polas probas. Probas necesarias en cada un dos seus contextos.

- Ferramentas para automatizar a execución de probas dinámicas:

JUnit: antes de empezar a elaborar esta suite de probas, xa tiñamos test con JUnit, sin embargo, tiñámos algún método sen probar ou que non non cumpría co que tiña que facer. Polo que coa ferramenta JUnit puidemos seguir completando os tests unitarios da práctica. Permítenos elaborar probas unitarias e ver se os métodos se comportan da forma esperada. Mais adiante, veremos mais disto coa axuda de Cobertura.



2.jpg

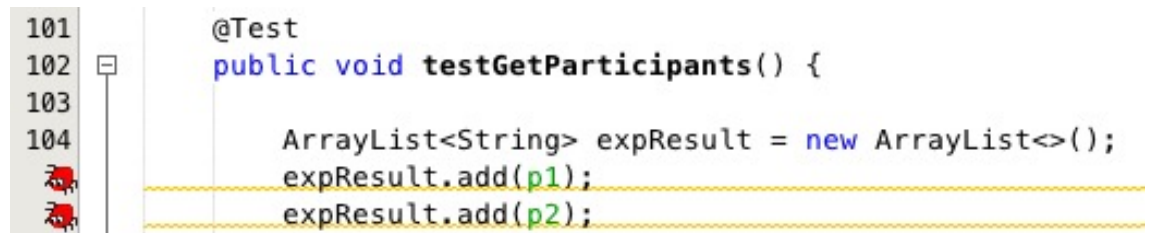
- Ferramentas para facilitar as probas dinámicas de unidade:

Mockito: é unha librería de Java (baseado en EayMock) que nos permite crear Mocks ou obxectos simulados que son moi empregados en probas unitarias en TDD (Test Driven Development). É importante porque consiste na creación de probas primeiro.

Na nosa práctica non puidemos aplicalo por problemas que tiñamos no transcurso de Ant a Maven, e non puidemos implementalo.

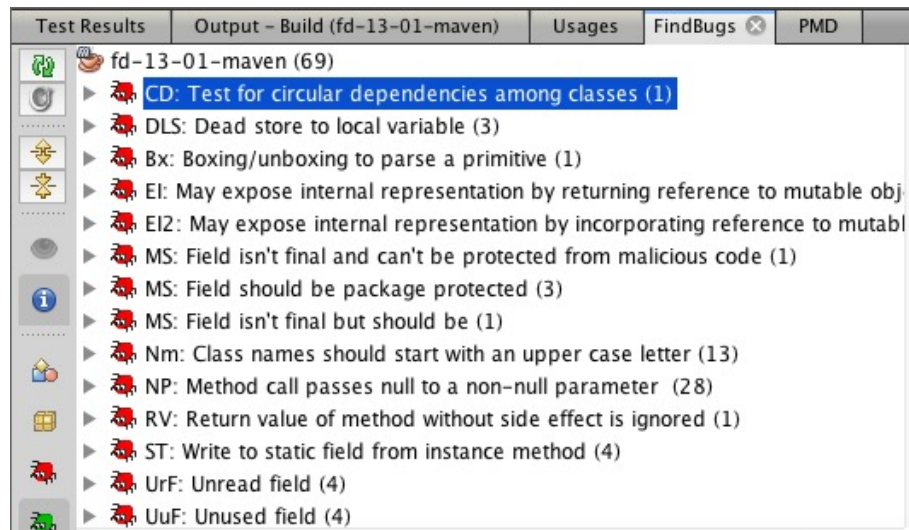
- Análise estática de caixa branca

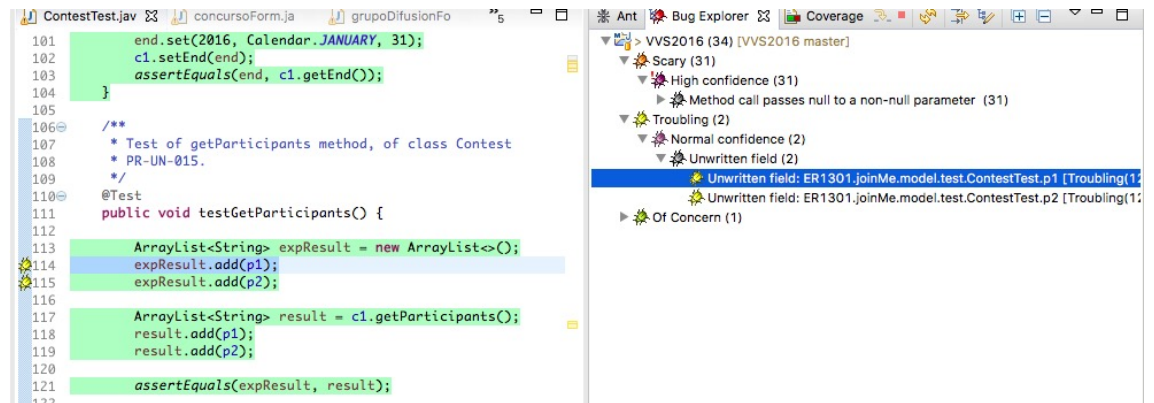
Findbugs: esta ferramenta permítenos probar se temos bugs potenciais no sistema, devolve bugs graves ou leves. Tras executalo o resultado podémolo ver nas seguintes imaxes. No obstante, moitos erros son de encoding, polo que non son relevantes (e normal que aparezan).



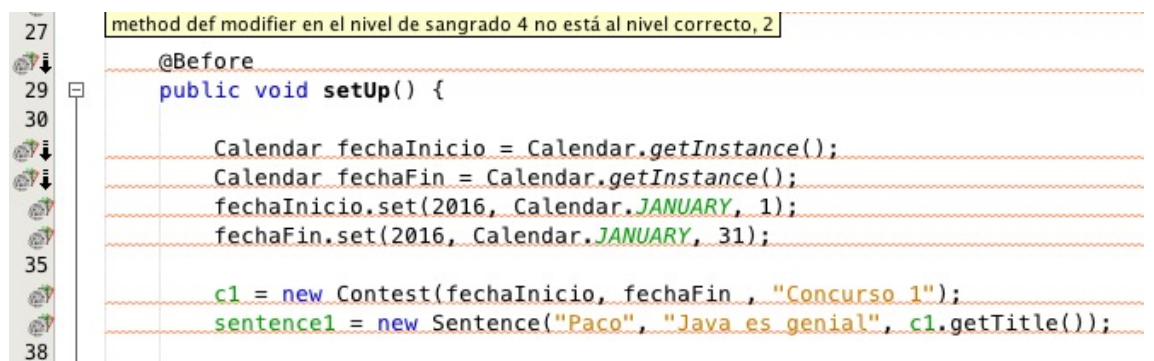
```

101 @Test
102 public void testGetParticipants() {
103
104     ArrayList<String> expResult = new ArrayList<>();
        expResult.add(p1);
        expResult.add(p2);
    
```





CheckStyle: esta ferramenta axuda a que programando cumplamos cos estándares de codificación apropiados e que nos guíemos polas boas prácticas. Aínda que non se pedía, decidimos incluíla porque axuda a detectar problemas de código duplicado, imports que non se usan, comentarios Javadoc, etc.



■ Mutación de código

Pitest: persoalmente esa ferramenta de mutación considéroa imprescindible pra mellorar a calidade das probas e atopar erros que coas probas unitarias xamáis poderíamos atopar. O que fai é darnos pistas pra que melloremos a calidade das probas e atopemos erros dándolle a volta ao código. O feito de resolver os erros atopados por Pitest, fíxonos mellorar os resultados de Cobertura.

Mutations

13	1. Removed assignment to member variable idPropietario → SURVIVED
15	1. Removed assignment to member variable idGrupoDifusion → SURVIVED
17	1. removed call to java/util/ArrayList::<init> → SURVIVED 2. Removed assignment to member variable listaUsuariosGrupo → SURVIVED
19	1. removed call to java/util/ArrayList::<init> → SURVIVED 2. Removed assignment to member variable listaGruposGrupo → SURVIVED
21	1. removed call to java/util/ArrayList::<init> → KILLED 2. Removed assignment to member variable inbox → KILLED
27	1. Removed assignment to member variable idGrupoDifusion → KILLED
28	1. Removed assignment to member variable idPropietario → KILLED
29	1. Removed assignment to member variable listaUsuariosGrupo → KILLED
30	1. Removed assignment to member variable listaGruposGrupo → KILLED
mutaciones.jpg	1. negated conditional → KILLED

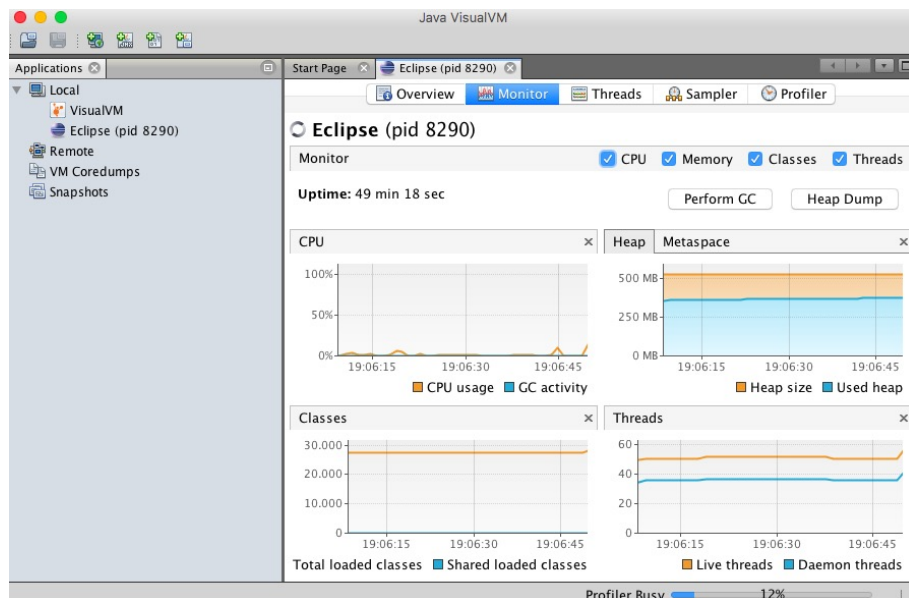
```

13 1 private String idPropietario = null;
14
15 1 private String idGrupoDifusion = null;
16
17 2 private List<String> listaUsuariosGrupo = new ArrayList<String>();
18
19 2 private List<GrupoDifusion> listaGruposGrupo = new ArrayList<GrupoDifusion>();
20
21 2 private List<Mensaje> inbox = new ArrayList<Mensaje>();
22
23 /* crear grupo difusion */
24 public GrupoDifusion (String idGrupoDifusion, String idPropietario, List<String> listaUsuariosGrupo,
25     List<GrupoDifusion> listaGruposGrupo){
26
27 1     this.idGrupoDifusion = idGrupoDifusion;
28 1     this.idPropietario = idPropietario;
29 1     this.listaUsuariosGrupo = listaUsuariosGrupo;
30 1     this.listaGruposGrupo = listaGruposGrupo;

```

■ Estrés:

VisualVM: é unha ferramenta de monitorización. Permite analizar os fíos de execución, uso de CPU e outros factores. Tamén decidimos incorporala o proxecto.



Name	Total Time	Total Time (CPU)
Java2D Queue Flusher	13.663 ms (100%)	206 ms (100%)
AWT-EventQueue-0	6.379 ms (100%)	1.432 ms (100%)
main	460 ms (100%)	284 ms (100%)
AppKit Thread	190 ms (100%)	190 ms (100%)
AWT-Shutdown	181 ms (100%)	0,0 ms (-%)
process reaper	0,0 ms (-%)	0,0 ms (-%)
Finalizer	0,0 ms (-%)	0,0 ms (-%)
Reference Handler	0,0 ms (-%)	0,0 ms (-%)
Java2D Disposer	0,0 ms (-%)	0,0 ms (-%)
TimerQueue	0,0 ms (-%)	0,0 ms (-%)

- Ferramentas para automatizar a xeración de datos en execución de probas dinámicas:

QuickCheck: está ferramenta úsase, como di máis arriba, para a xeración aleatoria de casos de proba. Por problemas coa práctica, tam-pouco pudemos implementalo con éxito. No obstante, sí coñecemos o seu funcionamento de outras prácticas, mesmo a do ano pasado.

- Probas basadas en modelos:

GraphWalker: é unha librería de Java para automatizar a xeración e execución de probas dinámicas. Básicamente consiste no deseño do diagrama de estados do sistema para logo mediante unhas probas verificar o funcionamento contra este diagrama... fixemos o diagrama de estados,

metímolo no proxecto pra xerar a interfaz pero tivemos problemas a hora implementar as probas, numerosos erros que no puidemos arranxar.

■ Ferramentas para a validación de probas - Cobertura:

Cobertura: ter Cobertura no proxecto é crucial pra saber a calidade das probas, pra saber con exactitude a cantidade de código que estamos probando cos test. Moitas veces cremos que probamos pero non estamos pasando por todos os bucles, como resultado poderíamos ter erros no proxecto que non atopamos por facer as probas incompletas. Cobertura mellorou moito dende o comezo do proxecto ata o final.

Coverage Report - ER1301.joinMe.model

Package /	# Classes	Line Coverage	Branch Coverage	Complexity
ER1301.joinMe.model	9	15% <div><div></div></div> 66/437	2% <div><div></div></div> 4/144	1,643
Classes in this Package /	Line Coverage	Branch Coverage	Complexity	
Challenge	0% <div><div></div></div> 0/18	N/A <div><div></div></div> N/A	1	
Comentario	0% <div><div></div></div> 0/12	N/A <div><div></div></div> N/A	1	
Contest	48% <div><div></div></div> 23/47	0% <div><div></div></div> 0/12	1,353	
Contest\$1	0% <div><div></div></div> 0/2	N/A <div><div></div></div> N/A	1,353	
GrupoDifusion	90% <div><div></div></div> 30/33	50% <div><div></div></div> 4/8	1,333	
Mensaje	31% <div><div></div></div> 7/22	N/A <div><div></div></div> N/A	1	
Sentence	33% <div><div></div></div> 6/18	N/A <div><div></div></div> N/A	1	
Sistema	0% <div><div></div></div> 0/103	0% <div><div></div></div> 0/34	2	
Usuario	0% <div><div></div></div> 0/182	0% <div><div></div></div> 0/90	2,382	

Report generated by Cobertura 2.1.1 on 23/12/15 2:11.

Coverage Report - ER1301.joinMe.model

Package /	# Classes	Line Coverage	Branch Coverage	Complexity
ER1301.joinMe.model	9	30% <div><div></div></div> 33/437	4% <div><div></div></div> 7/144	1,643
Classes in this Package /	Line Coverage	Branch Coverage	Complexity	
Challenge	100% <div><div></div></div> 18/18	N/A <div><div></div></div> N/A	1	
Comentario	100% <div><div></div></div> 12/12	N/A <div><div></div></div> N/A	1	
Contest	70% <div><div></div></div> 33/47	25% <div><div></div></div> 3/12	1,353	
Contest\$1	0% <div><div></div></div> 0/2	N/A <div><div></div></div> N/A	1,353	
GrupoDifusion	90% <div><div></div></div> 30/33	50% <div><div></div></div> 4/8	1,333	
Mensaje	100% <div><div></div></div> 22/22	N/A <div><div></div></div> N/A	1	
Sentence	100% <div><div></div></div> 18/18	N/A <div><div></div></div> N/A	1	
Sistema	0% <div><div></div></div> 0/103	0% <div><div></div></div> 0/34	2	
Usuario	0% <div><div></div></div> 0/182	0% <div><div></div></div> 0/90	2,382	

Report generated by Cobertura 2.1.1 on 23/12/15 3:36.

Coverage Report - ER1301.joinMe.model

Package /	# Classes	Line Coverage	Branch Coverage	Complexity
ER1301.joinMe.model	9	74% <div><div></div><div>327/439</div></div>	42% <div><div></div><div>61/144</div></div>	1,638
Classes in this Package /	Line Coverage	Branch Coverage	Complexity	
Challenge	100% <div><div></div><div>18/18</div></div>	N/A <div><div></div><div>N/A</div></div>	1	
Comentario	100% <div><div></div><div>12/12</div></div>	N/A <div><div></div><div>N/A</div></div>	1	
Contest	70% <div><div></div><div>33/47</div></div>	25% <div><div></div><div>3/12</div></div>	1,353	
Contest\$1	0% <div><div></div><div>0/2</div></div>	N/A <div><div></div><div>N/A</div></div>	1,353	
GrupoDifusion	90% <div><div></div><div>30/33</div></div>	50% <div><div></div><div>4/8</div></div>	1,333	
Mensaje	100% <div><div></div><div>22/22</div></div>	N/A <div><div></div><div>N/A</div></div>	1	
Sentence	100% <div><div></div><div>18/18</div></div>	N/A <div><div></div><div>N/A</div></div>	1	
Sistema	74% <div><div></div><div>78/105</div></div>	58% <div><div></div><div>20/34</div></div>	1,944	
Usuario	63% <div><div></div><div>118/187</div></div>	37% <div><div></div><div>24/90</div></div>	2,382	

Report generated by Cobertura 2.1.1 on 24/12/15 0:09.

O último informe de cobertura mellorou moito respecto dos anteriores, pola suite de probas empregada.

No caso de ContestTest.java podémolo ver na seguinte imaxe:

▼ ContestTest	100,0 %	268	0	268
ContestTest()	100,0 %	3	0	3
setUp()	100,0 %	65	0	65
testGetEnd()	100,0 %	20	0	20
testGetParticipants()	100,0 %	32	0	32
testGetSentences()	100,0 %	14	0	14
testGetStart()	100,0 %	20	0	20
testGetState()	100,0 %	1	0	1
testGetTitle()	100,0 %	16	0	16
testSetEnd()	100,0 %	17	0	17
testSetParticipants()	100,0 %	31	0	31
testSetSentences()	100,0 %	19	0	19
testSetStart()	100,0 %	17	0	17
testSetTitle()	100,0 %	13	0	13

3. Rexisto de probas

Tuvemos problemas coa migración do proxecto a Maven. Tíñamolo en Netbeans o tivemos moitos erros no código a hora de pasalo a eclipse como Maven, con pom.xml. Isto fíxonos perder moito tempo amañando problemas que non se correspondían con esta práctica. Incluso pensamos en cambiar o proxecto, pero xa era tarde.

Polo xeneral sí tuvemos problemas de falta de tempo debido a unha mala organización. Sobre todo pola falta de información a hora de preparar o entorno pra executar as probas. Sí e certo que contamos con moita documentación de cada tipo de proba na súa correspondente páxina oficial, pero moitas veces queda un pouco no aire cómo seguir os pasos.

Na maioría das probas, sobre todo nalgunas que creemos que son moi importantes, tivemos problemas pra probalas e a práctica puido quedar un pouco mais coxa nese sentido, porque a calidade das probas non sei probou na sua totalidade. Pero creemos que pode tar bastante completo.

4. Rexistro de erros

Non atopamos nada relevante. Ainda que puidemos haber probado mellor o código, polo xeneral non parece haber erros potenciais. Todo fai o que debe. E como o noso proxecto ten interfaz gráfica, visualmente funciona ben e non hai

erros por ningures.

5. Estatísticas

Deben incluírse como mínimo:

- *Número de erros encontrados diariamente e semanalmente. Nas primeiras semanas uns 2-3 por semana.*
- *Nivel de progreso na execución das probas. Nivel favorable.*
- *Análise do perfil de detección de erros (lugares, compoñentes, tipoloxía). A maioría dos erros malas prácticas de programación, sen bugs potenciais.*
- *Informe de erros abertos e pechados por nivel de criticidade. As issues están en GIT. Un total de 7.*
- *Avaliación global do estado de calidade e estabilidade actuais. O proxecto aínda que non está probado con tódalas ferramentas, pensamos que está medianamente completo.*

6. Outros aspectos de interese

Aínda que parece que sempre se deixan as probas pro final, e moi importante telas en conta porque sempre nos permiten non so atopar erros, senon mellorar a calidade de código e detectar malas prácticas de programación, como pode ser CheckStyle que axuda moito neso.

Como resultado, coa execución de todas estas probas puidemos mellorar a calidade de código e penso que podemos dicir sen equivocarnos que a práctica fai o que debe facer.