Pilot

Participant:	0				
Date:	11 July 2014				
Start time:	13:27:00				
Finish time:	14:27:00				
Duration:	01:00:00				

Timestamp	Activity
13:27:00	Starts
13:36:00	Reading documentation. Problems understanding target class.
	* Should we interact to guide on the purpose of the target class?
13:39:00	Starts writing code. Starts with exception behaviour.
	* Should we create a list of situations in which we expect verbalization?
13:45:00	Implemented constructors and XXX methods.
13:46:00	Runs EvoSuite.
13:46:00	Checks coverage. Refactors code. Checks coverage.
13:49:00	Changed code. Runs EvoSuite. Decides he'll use tool more often.
13:52:00	Decides to go and see test cases to understand code.
13:55:00	Back to Implementation.
13:58:00	Examines tests.
14:00:00	Receives help on what the class is supposed to do.
14:05:00	Gives up.
14:07:00	Implements more.
14:10:00	Runs EvoSuite.
14:13:00	Still confused with logic of target class.
14:17:00	Implements more.
14:21:00	Runs Evosuite.
14:21:00	Checks Coverage.
14:27:00	Gives up. Believes implementation is complete but EvoSute achieved 100% coverag
14:30:00	Copies tests. Adds more. Checks coverage.
14:40:00	Writes more tests by hand.
14:42:00	Looks a bit frustrated. Coverage gets worse.
14:45:00	Finishes

Participant:	1				
Date:	21 July 2014				
Start time:	15:20:00				
Finish time:	17:07:00				
Duration:	01:47:00				

Timestamp	Activity
15:20:00	Starts. Reads documentation.
15:23:00	Adds private fields.
15:25:00	Finds constructors a bit confusing.
15:28:00	Confused about the initial state of the underlying iterator.
15:30:00	Goes to inspect Predicate class.
15:32:00	Works on target class (method hasNext())
15:35:00	Continues working on target class (method next()). No attempt to generate tests so far.
15:38:00	Thinks of nicer way of writing target class already.
15:40:00	Back to method hasNext() in target class to improve implementation. Adds loop.
15:41:00	Writes tests manually. Starts with exception handling.
15:43:00	Decides to use a list to iterate over in tests.
15:44:00	Runs tests. All pass.
15:45:00	Writes test for non-trivial behaviour.
15:47:00	Runs tests. All pass.
15:49:00	Finds unexpected behaviour with a test. Back to debug code.
15:51:00	Opens debug perspective.
15:53:00	Debugging one of the tests.
15:56:00	Fixes bug.
16:00:00	Moves on to method remove in target class. Reading documentation.
16:03:00	Adds field and writes method remove.
16:05:00	Writes more tests. One fails.
16:06:00	Refactoring target class (method next())
16:07:00	Organizing tests. Writing comments.
16:09:00	Adds more tests.
16:10:00	Plans on writing some more tests and only then checking coverage.
16:13:00	Feels guilty for not needing to use EvoSuite.
16:16:00	Runs EvoSuite.
	* Too concious about the experiment. Executes EvoSuite almost out of guilt.
16:16:00	Plans on comparing test coverage between his tests and EvoSuite tests.
16:17:00	Surprised that EvoSuite only 6 tests.
16:18:00	Inspects generated tests.
16:20:00	Checks coverage for EvoSuite tests.
16:22:00	Surprised that EvoSuite did not reached method remove() at all
16:23:00	Runs EvoSuite. Gets better tests now.
16:24:00	Inspects generated tests.
16:26:00	Fixes own tests.

16:29:00	Debugs one failing test.
16:34:00	Problems understanding method remove().
16:37:00	Still struggling with method remove().
16:40:00	Changes implementation.
16:42:00	Adds new tests.
16:45:00	Thinks about what to do if no predicate was specified.
16:47:00	Decides what to do if no predicate was specified.
16:49:00	Runs tests. Checks coverage.
16:52:00	Adds more tests.
16:53:00	Runs EvoSuite.
16:56:00	EvoSuite is slow.
16:57:00	Checks coverage for EvoSuite tests. 100% Coverage.
16:59:00	Finds redundancy in tests.
17:01:00	Adds tests from EvoSuite. Speaks positively of the tool.
17:03:00	Complains about readability.
17:05:00	Adds more EvoSuite tests. Happy with result.
17:07:00	Finishes.

Participant:	1				
Date:	21 July 2014				
Start time:	15:20:00				
Finish time:	17:10:00				
Duration:	01:50:00				

Timestamp	Activity
15:20:00	Starts. Reads documentation.
15:23:00	Adds private fields.
15:25:00	Finds constructors a bit confusing.
15:27:00	* Prompted to keep verbalizing
15:28:00	Confused about the initial state of the underlying iterator.
15:30:00	Goes to inspect Predicate class *and subclasses*.
15:32:00	*Back to continue working* on target class (method hasNext())
15:35:00	Continues working on target class (method next()). No attempt to generate tests so far.
15:38:00	Thinks of nicer way of writing target class already.
	"Just realized that my first solution doesn't make sense"
15:40:00	Adds loop to hasNext() in target class.
15:41:00	Writes tests manually. Starts with exception handling.
15:43:00	Decides to use a list to iterate over in tests.
15:44:00	Runs tests. All pass.
15:45:00	Writes test for non-trivial behaviour.
15:47:00	Runs tests. All pass.
15:49:00	Finds unexpected behaviour with a test. Back to debug code.
15:51:00	Opens debug perspective.
15:53:00	Debugging one of the tests.
	Test fails because next() calls hasNext() and viceversa, consuming all elements at once.
15:56:00	Fixes bug.
16:00:00	Moves on to method remove in target class. Reading documentation.
16:03:00	Adds field and writes method remove.
16:05:00	Writes more assertions in a previous test. It fails.
16:06:00	Fixing method next() in target class
16:07:00	Organizing tests. Writing comments.
16:09:00	Adds more tests.
16:10:00	Plans on writing some more tests and only then checking coverage.
16:13:00	Feels guilty for not needing to use EvoSuite.
	* Asks whether he must use EvoSuite at some point or not.
	Thinks that it might be more efficient to have EvoSuite generate tests for all possible combinations of calls in his class
	Decides to write one more test manually and then trying EvoSuite
	Testing thoroughly this small class gets tedious. Going to use EvoSuite to generate tests with the idea of comparing coverage and maybe combining test suites
16:16:00	Runs EvoSuite.
	* Too concious about the experiment. Asks whether tests are minimized.

16:16:00	Plans on comparing test coverage between his tests and EvoSuite tests.
16:17:00	Surprised that EvoSuite only generated 6 tests.
16:18:00	Inspects generated tests.
	"Tests are not quite useful" Only simple behaviour is covered, and not cases where actual collections are passed to the iterator.
16:20:00	Checks coverage for EvoSuite tests.
	* EvoSuite failed to generate any collection to pass to the iterator, hence, the body of the loop therein is not exercised and coverage is low.
16:22:00	Surprised that EvoSuite did not reached method remove() at all
	* The non-deterministic nature of EvoSuite can confuse the user by failing to cover methods that seem easy to cover.
16:23:00	Runs EvoSuite. Gets better tests now.
	* New test suire gets much higher coverage (97%)
16:24:00	Inspects generated tests.
	* Checks coverage of his own test suite (85%)
	* Realises he has 2 failing tests.
16:26:00	Fixes own tests. He was expecting wrong exception class
16:29:00	Debugs one failing test.
	* Failing test is related to the remove() method
16:34:00	Problems understanding method remove().
16:37:00	Still struggling with method remove().
	* Assisted to understand the expected behaviour of remove()
16:40:00	Changes implementation.
	* Adds boolean to control if removals are valid or not
16:42:00	Adds new tests.
	* Actually, corrects previous test
16:45:00	Thinks about what to do if no predicate was specified.
	* Some of his tests are failing because of this
16:47:00	Decides what to do if no predicate was specified.
	* The specification is not clear as to how the iterator should behave with no predicate
16:49:00	Runs tests. Checks coverage.
16:52:00	Adds more tests.
	* Adds more tests for uncovered methods
	* Did not consider looking for suitable tests in evosuite
16:53:00	Runs own tests. Checks coverage. 100 % .
	* Mentions again that writing tests manually is tedious.
	* Not sure if implementation is exactly the expected behaviour.
16:56:00	Runs EvoSuite.
	* On inspection of EvoSuite generated tests: "Tests are quite easy to read and understand. If all tests looked like that, I'd rather use EvoSuite than writing tests by hand"
	* EvoSuite is slow.
16:59:00	Checks coverage for EvoSuite tests. 100% Coverage.
	* Did not see the tests. Went directly to run them and check coverage.
17:00:00	On inspection of generated tests, finds some redundancies.
	* More related to replicated code than to coverage

17:02:00	Adds tests from EvoSuite. Speaks positively of the tool.
	* On reflection about all possible combination of invocations to methods, finds one generated case in which remove() is called right after initializing the iterator, which he did not consider in his manual tests: "Those are the tests that are nice to have automatically generated"
17:03:00	Complains about readability.
	"Tests get harder to read once they reach 10-15 lines. I'm not sure if shorter names would make it any better, but it's getting pretty heavy to read"
17:04:00	Adds more EvoSuite tests.
	* Copies EvoSuite test that checks initial state of predicate is null.
17:06:00	* Remembers a useful test from a previous run of EvoSuite and write it by hand.
17:08:00	Adds more EvoSuite tests. Happy with result.
17:10:00	Finishes.

Participant:	2				
Date:	23 July 2014				
Start time:	15:57:00				
Finish time:	17:17:00				
Duration:	01:20:00				

Timestamp	Activity
15:57:00	Starts. Reads documentation.
15:59:00	Implements basic methods.
16:02:00	Implements exceptional behaviour of non-trivial methods.
16:03:00	Chooses HashMap as underlying data structure.
16:03:00	Doubts on what the target object should be. Decides Object.
16:06:00	Coding method compare()
16:08:00	Writes tests. Runs them. All pass.
16:09:00	Writes more tests.
16:11:00	Back to target class to write more methods.
16:11:00	Changes code. Runs tests again to make sure nothing is broken.
16:13:00	Writes tests for yet unimplemented method add()
16:14:00	Writes basic implementation for add()
16:15:00	Writes test for addAsEqual()
16:16:00	Writes addAsEqual()
16:17:00	Runs EvoSuite. Believes implementation is OK.
	* Lets EvoSuite run in background. Goes back to target class.
16:18:00	Inspects EvoSuite tests.
16:20:00	EvoSuite test helps realise about miss-implemented behaviour (null element exception).
16:21:00	Runs EvoSuite.
16:22:00	Complains about readability.
16:22:00	Refactors EvoSuite test which created an array that was not used later.
16:24:00	Copies test from EvoSuite tests.
16:24:00	Decides to stop inspecting EvoSuite tests and to continue coding.
16:25:00	EvoSuite test points to wrong exception being thrown. Fixes it.
16:26:00	Test fails. Fixes it.
16:27:00	Adds more exception handling.
16:28:00	Checks coverage.
16:29:00	Runs EvoSuite. Expects to have previously uncovered behaviour covered.
16:30:00	Wants to cancel EvoSuite job.
16:31:00	Inspects EvoSuite tests.
16:31:00	Copies tests. Refactors them.
16:31:00	Renames variables, removes unnecesary cast. Uses constants from target class.
16:32:00	Uses tests that cover code that he didn't cover.
16:33:00	Checks coverage.
16:34:00	Identifies uncovered code.
16:35:00	Writes tests (reusing an EvoSuite test)

16:36:00	Looks for uncovered behaviour.
16:37:00	Runs EvoSuite. Hopes to have one particular method covered.
16:38:00	EvoSuite covers target method. Copies expected test and refactors it.
16:40:00	Adds more behaviour to target class.
16:41:00	Runs tests. All pass.
16:42:00	Back to target class.
16:43:00	Runs tests. All pass.
16:44:00	Back to target class.
16:47:00	Believes implementation is finished.
16:47:00	Runs EvoSuite.
16:47:00	Checks Coverage.
16:48:00	Looks at code that was not covered.
16:49:00	EvoSuite tests are larger.
16:50:00	Copies test. Complains about readability and useless statements.
16:51:00	Finds a similar test (nicer than the previous one)
16:53:00	Find a test that reveals incorrect behaviour (null entries). Copies. Adds assertions.
16:53:00	Changes code in target class.
16:54:00	Checks coverage.
16:54:00	Looks for shorter tests in EvoSuite tests.
16:54:00	Deletes tests that he knows he already has.
16:55:00	Finds two good tests. Adds them.
16:57:00	Checks coverage.
16:58:00	Wishes for EvoSuite to highlight tests that reach uncovered code (from his test suite)
16:59:00	Copies test that looks easy to modify and cover new behaviour.
17:00:00	Avoids re-executing EvoSuite, in favour of narrowing the search to new tests.
17:01:00	Runs EvoSuite.
17:01:00	Copies all tests. Runs tests.
17:02:00	Checks Coverage. Changes tests. Runs tests.
17:02:00	Checks Coverage. 100%
17:04:00	Revisits test suite, checking which tests actually contribute coverage.
17:04:00	Removes some tests.
17:07:00	Claims generated tests are convoluted, yet not horrendous.
17:09:00	Recognizes useful tests. Renames and revactors them.
17:12:00	Removes spurious assertions.
17:12:00	Runs tests. Checks coverage.
17:14:00	Finishes pass on all tests. Start a new pass to check assertions are correct.
17:17:00	Finishes.
17:17:00	Restrospective phase. Tests are ugly.

Participant:	2				
Date:	23 July 2014				
Start time:	15:57:00				
Finish time:	17:17:00				
Duration:	01:20:00				

Timestamp	Activity
15:57:00	Starts. Reads documentation.
15:59:00	Implements basic methods.
16:02:00	Implements exceptional behaviour of non-trivial methods.
16:03:00	Chooses HashMap as underlying data structure.
16:03:00	Doubts on what the target object should be. Decides Object.
16:06:00	Coding method compare()
16:08:00	Writes tests. Runs them. All pass.
16:10:00	Back to target class to write more methods.
16:11:00	Changes code. Runs tests again to make sure nothing is broken.
16:12:00	Writes tests for yet unimplemented method add()
16:13:00	Writes basic implementation for add()
	* Reruns tests. User has an hybrid approach to testing. First he implemented some behaviour. Then created some tests. And then he created tests for yet unimplemented behaviour, switching to a test-driven approach.
16:14:00	Writes test for addAsEqual()
16:15:00	Writes addAsEqual()
16:16:00	Believes implementation is OK.
16:16:00	Runs EvoSuite.
	* Lets EvoSuite run in background. Goes back to target class.
16:17:00	Inspects EvoSuite tests.
16:19:00	EvoSuite test helps realise about miss-implemented behaviour (null element exception).
16:20:00	Corrects implementation. Runs tests again and some fail.
16:21:00	Runs EvoSuite.
16:22:00	Complains about readability, specially about the test setup.
16:22:00	Refactors EvoSuite test which created an array that was not used later.
	* EvoSuite had created a test where an array a wascreated, then one element put in the array position a[7] and then a[7] passed as argument to a method.
16:24:00	Copies test from EvoSuite tests.
16:24:00	Decides to stop inspecting EvoSuite tests and to continue coding.
16:25:00	EvoSuite test points to wrong exception being thrown. Fixes it.
16:26:00	Test fails. Fixes it.
	* He had used a test believing method adAsEquals() was being tested, and realized later that it was method compare() that was being tested.
16:27:00	Adds more exception handling.
16:28:00	Checks coverage.
16:29:00	Runs EvoSuite. Expects to have previously uncovered behaviour covered.
	* This is a good practice when using EvoSuite, recognizing that there are uncovered branches and looking for tests that excersise them.
	* Realizes he run EvoSuite on his test suite instead of his target class

16:30:00	Wants to cancel EvoSuite job.
	* Feature does not work quite well
16:31:00	Inspects EvoSuite tests.
16:31:00	Copies tests. Refactors them.
	* Spots a test that excercises behaviour he had not covered by hand.
16:31:00	Renames variables, removes unnecesary cast. Uses constants from target class.
16:32:00	Uses tests that cover code that he didn't cover.
16:34:00	Checks coverage.
16:34:00	Identifies uncovered code.
16:35:00	Writes tests (reusing an EvoSuite test)
16:36:00	Checks coverage.
16:35:00	Looks for uncovered behaviour.
16:36:00	Runs EvoSuite. Hopes to have one particular method covered.
16:37:00	EvoSuite covers expected method. Copies expected test and refactors it.
	* Hints on a useful improvement for EvoSuite: generate tests for a single goal: a method
16:38:00	Runs tests. Checks coverage. 92%
16:39:00	Adds more behaviour to target class.
16:40:00	Runs tests. All pass.
16:41:00	Back to target class.
16:42:00	Runs tests. All pass.
16:43:00	Back to target class.
	* Shows an incremental approach to solve the program. Implements little by little, leaving notes about things he is not entirely sure about.
16:46:00	Believes implementation is finished.
16:46:00	Runs EvoSuite.
16:46:00	Checks Coverage.
16:47:00	Looks at code that was not covered.
	* Again, this is a good pattern. Write code. Run own test suite and check coverage. Run EvoSuite and try to find tests that improve own test suite coverage.
16:48:00	EvoSuite tests are larger.
	* Notices the difference from previous runs of the tool. "The EvoSuite tests have grown a little bit since the first time I generated them, obviously, because the behaviour has expanded"
16:49:00	Copies test. Complains about readability and useless statements.
	"I am going to use this test; the actual test looks really ugly, there's a lot of cr*p in there that isn't necessary but I'm going to clean that up in a second" (then he proceeds to give a meaningful name to the test; leaving the refactoring for later) * This is positive. He akwnowledges that a test contains some useless statements, but uses it anyway, because it excercises uncovered behaviour.
16:50:00	Finds a similar test (nicer than the previous one)
	* Replaces previous test with the shorter nicer version.
16:51:00	Find a test that reveals incorrect behaviour (null entries). Copies. Adds assertions.

	* EvoSuite can help write better code: (while inspecting an EvoSuite test) "I've just noticed that EvoSuite has got a test where it is adding null to the list, but null shouldn't be allowed, so I'm going to take that test and change it so it should expect some exception" (then he copies the test, cleans it up, adds the right assertions, and go back to target class to implement the corresponding null-checking behaviour)
16:51:00	Changes code in target class.
16:52:00	Checks coverage.
	"Again, I'm just running the coverage tool just to see which branches I still need to cover"
16:54:00	Looks for shorter tests in EvoSuite tests.
	"I am instinctively looking for shorter tests because they are a lot easier to read"
16:54:00	Deletes tests that he knows he does not need.
16:55:00	Finds two good tests. Adds them.
16:56:00	Finds one more good test. Adds it.
16:57:00	Checks coverage.
16:58:00	Wishes for EvoSuite to highlight tests that reach uncovered code (from his test suite)
	"What would be really useful here is if EvoSuite could tell me which of the test that is generated cover new behaviour that my test suite doesn't, because I am doing that manually at this stage"
	* Uses Find/Replace Eclipse function to find a test that excercises a particular uncovered method (addAsEqual())
16:59:00	Copies test that looks easy to modify and cover new behaviour.
	"This test doesn't actually test what I wanted to but I know I can easily change it, so I am going to take it anyway"
16:59:00	Runs tests. Checks coverage. 93.8%
17:00:00	Avoids re-executing EvoSuite, in favour of narrowing the search to new tests.
17:01:00	Runs EvoSuite.
	* Changes approach: "Start copying lots of test from EvoSuite and then see what tests I can safely delete from my test suite rather than manually copying them from EvoSuite" "It would be really helpful if it was possible for EvoSuite to highlight lines that it's managed to cover that I haven't already covered or at least exclude some of the test cases that my test suite already covers"
17:01:00	Copies all tests. Runs tests.
17:02:00	Checks Coverage. It didn't get to 100%. Changes tests manually. Runs tests.
17:02:00	Checks Coverage. 100%
17:04:00	Revisits test suite, checking which EvoSuite tests actually contribute coverage.
	* A minimization option could be useful in this case.
17:04:00	Removes some tests.
17:07:00	Claims generated tests are convoluted, yet not horrendous.
	(while looking at an EvoSuite test) "This is kind of convoluted; it's not horrendous, but..."
17:09:00	Recognizes useful tests. Renames and revactors them.
17:12:00	Removes spurious assertions.
17:12:00	Runs tests. Checks coverage.
17:14:00	Finishes pass on all tests. Start a new pass to check assertions are correct.
17:17:00	Finishes.

17:17:00	Restrospective phase. Tests are ugly.
	"Having EvoSuite is very nice, specially at the end once I implemented everything it was very helpful to have it generate tests for me" "Tests are ugly; for example Arrays are created in a weird way" "It'd be nice if some of the unnecessary statements were removed" "It was nice being able to have EvoSuite generate tests for me, but it was a pain to try to figure out which tests were actually useful for me" "The manual effort of going through each test and figuring out what it's doing, what it's testing, have I tested this before is quite an arduous task, but that said, it is still much easier, specially once the behaviour is complicated enough"

Participant:	3				
Date:	12 August 2014				
Start time:	10:50:00				
Finish time:	12:50:00				
Duration:	02:00:00				

Timestamp	Activity
10:50:00	Starts. Reads documentation.
10:52:00	Writes tests by hand before starting to code.
10:53:00	Creates inner class in test suite which extends ListPopulation
	* Shows concern about using a subclass to Unit test a super class
10:58:00	Runs tests. Fails as expected.
11:01:00	Writes tests before writing constructors.
11:02:00	Runs tests.
11:05:00	Writes more tests.
11:07:00	Writes more tests.
11:10:00	Continues testing constructors.
11:11:00	Complains about JavaDoc imposing implementation details.
11:14:00	Corrects tests.
11:16:00	Inspects dependencies for Chromosome class.
11:18:00	Creates list of chromosomes for testing.
11:23:00	Continues writing tests.
	* Clear test-driven approach
11:27:00	Writes more code in target class.
11:28:00	Writes tests.
11:34:00	Adds member variables.
11:36:00	Runs tests. All pass. Adds more tests.
11:39:00	Writes code in target class.
11:39:00	Considers using EvoSuite
11:41:00	Believes constructors are implemented and tested.
11:42:00	Runs EvoSuite. Reckons tests might be useless, though.
11:45:00	Inspects generated tests. Complains about readability.
11:47:00	Decides not to use any generated test.
11:48:00	Implements more behaviour in target class.
11:53:00	Writes test assertions.
11:56:00	Writes more code in target class.
11:59:00	Writes more code in target class.
12:00:00	Runs EvoSuite.
12:00:00	Says it'd be nice to only generate tests for a given method.
12:01:00	Inspects generated tests.
12:04:00	Finds interesting tests and edits them.
12:08:00	Runs tests. Adds more tests manually.
12:10:00	Moves on to another method in target class. Starts by writing tests.

12:13:00	Writes behaviour for addChromosomes.
12:14:00	Runs tests. Some fail. Corrects code.
12:20:00	Believes more code is correctly written and tested.
12:21:00	Adds new tests.
	* EvoSuite seems to have little opportunity to help test-driven developers.
12:26:00	Continues writing tests. Runs tests. Some fail.
12:27:00	Back to add behaviour to make last tests pass.
12:29:00	Moves on to another method (getFittestChromosome)
12:34:00	Struggles a bit to understand Chromosomes.
	* Subject prompted to create anonymous objects
12:38:00	Writes tests
12:40:00	Writes tests
12:44:00	Running out of time. Declares he is happy with the result.
12:45:00	Retrospective phase. Too concious during the session.

Participant:	3				
Date:	12 August 2014				
Start time:	10:50:00				
Finish time:	12:50:00				
Duration:	02:00:00				

Timestamp	Activity
10:50:00	Starts. Reads documentation.
10:52:00	Writes tests by hand before starting to code.
10:54:00	Creates inner class in test suite which extends ListPopulation
	* Shows concern about using a subclass to Unit test a super class
10:58:00	Runs tests. Fails as expected.
11:00:00	Splits editor windows to see target class and test suite simultaneously
11:01:00	Writes tests before writing constructors.
11:02:00	Runs tests.
11:05:00	Writes more tests.
11:07:00	Writes more tests.
11:10:00	Continues testing constructors.
11:11:00	Complains about JavaDoc imposing implementation details.
11:14:00	Corrects tests.
11:16:00	Inspects dependencies for Chromosome class.
11:18:00	Creates list of chromosomes for testing.
11:23:00	Continues writing tests.
	* Clear test-driven approach
11:27:00	Writes more code in target class.
	* At this point, he has much more code written in the test suite than in the target class
11:28:00	Writes tests.
11:30:00	Runs tests. All pass.
11:34:00	Adds member variables.
11:36:00	Runs tests. All pass. Adds more tests.
11:38:00	(when trying to write a test, not knowing what kind of exception should be captured) "That seems like the sort of point where I might want to use a test generator" (implements a mockup method in target class to trigger the unknown exception)
11:39:00	Writes code in target class.
11:39:00	Considers using EvoSuite
11:41:00	Believes constructors are implemented and tested reasonably.
	* Tries to move on to setChromosomes method. Seems like he struggles with documentation. Then suddenly he decides to run EvoSuite.
11:42:00	Runs EvoSuite.
	* Prompted to articulate why: "I don't know what EvoSuite is going to generate; so I'm hoping that it will look at the throw's declarations and do something sensible around generating tests for those. However, now that I think about that, I remember that EvoSuite generates observed behaviour, which isn't there yet, as I haven't written any exception throwing yet"
11:44:00	Inspects generated tests. Complains about readability.

	* Although most of his tests are very short (3 lines), he complains about readability: "EvoSuite is generating quite a few tests, but since all of those methods and variables are using generated names, there's no indication of what is being tested other than the code itself. I am finding that makes it hard to read." * Tries to understand a test that checks exceptional behaviour: (while complaining about tests lacking indication of testing intention) "Ok, there is an explanation in here <0 is smaller than the minimum (0)>, which isn't true"
11:47:00	Decides not to use any generated test.
	* Instead, decides to write some exception throwing and then generating tests again
11:48:00	Implements more behaviour in target class.
11:53:00	Writes test assertions.
11:56:00	Writes more code in target class.
11:59:00	Continues coding target class.
12:00:00	"It would be nice if I could generate tests for a particular method, or involving a particular method"
12:00:00	Runs EvoSuite.
12:01:00	Inspects generated tests.
12:03:00	Finds interesting tests.
	* However, he does not copy them, and instead, rewrite them by hand: "I didn't like that generated code, so I'm going to write those manually"
12:04:00	Writes test by hand, comprising the behaviour of the two EvoSuite tests
12:07:00	Runs tests. Adds more tests manually.
12:11:00	Moves on to method addChromosomes in target class. Starts by writing tests.
12:14:00	Writes behaviour for addChromosomes.
12:15:00	Runs tests. Some fail. Back to correct code in target class.
12:21:00	Writes new test for method addChromosome().
12:21:00	Believes more code is correctly written and tested.
12:22:00	Adds new tests.
	* EvoSuite seems to have little opportunity to help test-driven developers.
12:27:00	Continues writing, running and fixing tests.
12:28:00	Back to add behaviour to make last tests pass.
12:30:00	Moves on to another method (getFittestChromosome)
12:35:00	Struggles a bit to understand Chromosomes.
	* Subject is prompted to create anonymous class objects for testing purposes
12:37:00	Creates test chromosome class with variable fitness
12:39:00	Writes tests
12:41:00	Writes tests
12:45:00	Running out of time. Declares he is happy with the result.
12:46:00	Finishes.
12:46:00	Retrospective phase. Too concious during the session.

	* 2 hours. * Subject didn't check code coverage at any moment. * He didn't use any EvoSuite test directly. * He didn't check code coverage either. He is asked to check coverage before leaving. He was expecting to get 100% but got 96.4% instead. Did not cover methods that were not implemented. "I came concious of being testing this class in much finer detail than I would if I was doing it for work and not for an experiment in how I test" "I might have been able to make better use of generated code if I had been more familiar with EvoSuite and known upfront what it was going to generate For example, I am familiar with Eclipse refactoring, and sometimes I tweak the code in a particular way before applying refactoring because I know what the results is going to be" "I was fairly confident in the code that I wrote because I had written the tests upfront" "There was a point where I stopped writting the tests upfront and started implementing just to see what the EvoSuite tool would do, then I ended up not using any generated test and felt quite lost when I came back to the code because I had implemented it and didn't have any test for it" "If I am going to do test first I need to stick at it in order to maintain the confidence in the code that I am writing"

Participant:	4				
Date:	12 August 2014				
Start time:	19:30:00				
Finish time:	21:08:00				
Duration:	01:38:00				

Timestamp	Activity
19:30:00	Starts. Reads documentation.
	* Subject is not familiar with MacOS
19:40:00	Struggles with MacOS keyboard.
19:42:00	Starts writing some code.
	* Still having problems with keyboard.
19:45:00	Does not verbalize as expected. Prompted to do so.
19:47:00	Continues working on coding target class.
19:50:00	Writes code. Moves to another method. Still no tests.
19:53:00	Continues writing code.
19:55:00	Prompted again to verbalize.
19:57:00	Reading more documentation.
20:01:00	* Looks confused about documentation. Does not verbalize enough.
20:03:00	Runs EvoSuite. Believes there is enough code. Curious about EvoSuite output.
20:05:00	Inspects generated tests.
20:09:00	Copies tests from EvoSuite.
20:11:00	Continues inspecting generated tests.
20:15:00	Refactors tests.
20:21:00	Renames tests.
20:22:00	Continues inspecting generated tests.
	* Refactors tests but doesn't seem concerned about coverage yet.
20:24:00	Removes test.
20:26:00	Shows concern about tests excersising trivial (delegated?) behaviour.
20:27:00	Refrains and think about the need for those tests to achieve 100% coverage goal.
20:29:00	Back to write more code in target class
	* Did not copy refactored tests
20:31:00	Creates private method to avoid duplicated code
20:33:00	Moves tests from EvoSuite
20:33:00	Runs EvoSuite.
20:35:00	Splits screen to visualize and compare new EvoSuite results vs previous tests.
20:38:00	Believes a particular test does nothing useful.
	* No test has been executed yet. Seems more worried to discard tests as useless.
20:44:00	Uses Eclipse refactoring function.
20:45:00	Continues editing tests.
	* Approach is not coverage-oriented.
20:47:00	Runs tests. First time. Tests fail.
20:48:00	Checks coverage.

20:49:00	Adds new manual test by copying a similar one.
20:51:00	Another related manual test.
20:53:00	Checks coverage.
20:54:00	Opens debug perspective.
20:57:00	Continues debugging.
20:59:00	Discovers bug. Validation before setting up predicates.
21:00:00	Runs tests. All pass. Checks coverage. 100%.
21:01:00	Feels happy with result.
21:02:00	Retrospective phase.
21:02:00	Complains about readability. Tests are misleading. Hard to tell what they are doing.
21:02:00	Tests need to be refactored.
21:08:00	Finishes

Participant:	4
Date:	12 August 2014
Start time:	19:30:00
Finish time:	21:08:00
Duration:	01:38:00

Timestamp	Activity
19:30:00	Starts. Reads documentation.
	* Subject is not familiar with MacOS
19:40:00	Struggles with MacOS keyboard.
19:42:00	Starts writing some code.
	* Still having problems with keyboard.
19:45:00	Does not verbalize as expected. Prompted to do so.
19:47:00	Continues working on coding target class.
19:50:00	Writes code. Moves to another method. Still no tests.
19:53:00	Continues writing code.
19:54:00	Prompted again to verbalize.
19:56:00	Reading more documentation. Struggling with implementation of method putAll()
20:00:00	* Looks confused about documentation. Does not verbalize enough.
20:02:00	Runs EvoSuite. Believes there is enough code. Curious about EvoSuite output.
	* Subject was prompted to justify why he run EvoSuite
20:04:00	Inspects generated tests.
	* Struggles a bit to understand the auxiliary classes
20:08:00	Copies test from EvoSuite.
20:10:00	Continues inspecting generated tests.
20:14:00	Questions the way assertions are generated by EvoSuite. Reminded that he is allowed to refactor the generated tests in any suitable way.
20:14:00	Refactors tests.
20:16:00	Augments generated test that checks key predicate to check value predicate likewise
20:18:00	Renames tests to more sensible names.
20:19:00	Continues inspecting generated tests.
	* Refactors tests but doesn't seem concerned about coverage yet.
20:21:00	Removes test.
20:23:00	Shows concern about tests excersising trivial delegated behaviour
	* Actually, he is not understanding correctly what the test is doing, mainly because it reuses variables of Predicate type as entries
20:26:00	Refrains and think about the need for those tests to achieve 100% coverage goal.
	* While inspecting a test, he does not understand what the test is actually testing, and states: "I wouldn't normally test such trivialness like that. I wouldn't care if I don't have 100% coverage. But, if my aim is 100% coverage, I guess that test should stay"
20:28:00	Back to improve target class according to specification
	* Did not copy refactored tests

20:30:00	Creates private method to avoid duplicating code
20:33:00	Moves previously refactored tests from EvoSuite to his test suite
20:33:00	Runs EvoSuite.
20:34:00	Inspects generated tests.
20:35:00	Uses split screen editor to visualize and compare new EvoSuite tests vs previous ones
20:38:00	Believes a particular test does nothing useful.
	* No test has been executed yet and he has no coverage information. Seems more worried to discard tests as useless.
20:44:00	Uses Eclipse refactoring function.
20:45:00	Continues editing tests.
	* Approach is not coverage-oriented.
20:48:00	Runs tests. First time. Tests fail.
20:49:00	Checks coverage. There are missing branches.
	* Wonders why EvoSuite didn't cover a branch, and realises that he wrote that piece of code after running EvoSuite the last time
20:50:00	Adds new test manually by copying code from a similar one.
20:52:00	Another related manual test.
20:54:00	Checks coverage. There are failing tests.
20:55:00	Opens debug perspective.
20:58:00	Continues debugging.
20:59:00	Discovers bug. Validation before setting up predicates.
21:00:00	Runs tests. All pass. Checks coverage. 100%.
21:01:00	Feels happy with result.
21:02:00	Retrospective phase.
21:02:00	Complains about readability. Tests are misleading. Hard to tell what they are doing. Tests need to be refactored.
21:02:00	"Automatically generated tests are hard to decipher. For example, the tool uses weird ways of generating null values to use." * Suggests to use static analysis to get rid of useless statements. * Suggests to refine assertions, e.g., assertEquals(false, obj.method()) -> assertFalse(obj.method()) "There are some weird ways of writing tests that mislead you a little bit; made me scratch my head a few times to understand what was it actually trying to test" "I used generated tests as starting point; I changed them to look more to how I would have written them." "I'd probably be a lot stingier in the number of tests that I'd implement. I wouldn't normally in real life be aiming for 100% coverage anyway. I would probably end up with fewer tests without this tool but I couldn't tell you if they would be all the right tests"
21:09:00	Finishes

Participant:	5				
Date:	13 August 2014				
Start time:	12:38:00				
Finish time:	14:18:00				
Duration:	01:40:00				

Timestamp	Activity
12:38:00	Starts. Reads documentation.
12:41:00	Runs EvoSuite.
	* Why?
12:44:00	Goes to target class to write the constructors.
	* Subject not very familiar with MacOS
12:46:00	Studies code.
12:47:00	Starts writing code.
12:48:00	Reads EvoSuite tests. Think about reusing one of them to test the code he just wrote.
12:50:00	Decides to implement methods following order of appearance in EvoSuite tests.
12:52:00	Finishes one method (validate())
12:53:00	Inspects EvoSuite tests to see if any test tests his new method
	* Does not realise he should re-run EvoSuite
12:54:00	Complains about EvoSuite tests being confusing.
12:56:00	Runs EvoSuite. Realises previous tests were generated for an empty class.
12:59:00	Acknowledges that tests have changed.
13:01:00	Realises he should write more code to allow EvoSuite to generate useful tests.
13:02:00	Decides to write his own tests.
13:08:00	Struggles to setup a test by hand.
13:10:00	Still having problems with the test setup.
13:14:00	Finishes test and runs it. Fail.
13:15:00	Realises about missed specification. Back to code to handle exception.
13:18:00	Runs tests again. Pass. Decides to write one more for the same method.
13:21:00	Runs tests. 4 passes.
13:21:00	Checks coverage. Happy with coverage of one method.
13:22:00	Goes back to write more methods.
13:27:00	More behaviour written. Complains about documentation. Makes assumptions.
13:30:00	Continues writing target class.
13:34:00	Believes implementation is finished.
13:35:00	Runs EvoSuite. Background.
13:36:00	Tries to refactor target class a bit.
13:37:00	Inspecting generated tests. Thinks they are too complex. Decides to go manual.
	* Only one minute inspecting tests
13:41:00	Writing manual tests.
13:42:00	Fixing target class.
13:43:00	Back to write tests manually
13:48:00	Runs tests. Fails. Correct and rerun tests.

13:49:00	Checks coverage. Some methods not covered.
	* Does not care to see if EvoSuite generated something useful for those methods
13:53:00	Writes manual tests.
13:55:00	Checks coverage. It has improved.
13:56:00	More manual tests.
14:01:00	Continues writing tests.
14:03:00	Runs tests. 100% coverage.
14:05:00	Runs EvoSuite.
14:07:00	Runs EvoSuite tests.
14:08:00	Inspects generated tests.
	* EvoSuite missed two branches
14:10:00	Revisits implementation. Happy with result.
14:10:00	Retrospective phase.
	Readability: Names should be better. Points out difference between TCG for "utility" classes like Map and more specific purpose classes. Value of tests lie on readability.
14:18:00	Finishes

Participant:	5				
Date:	13 August 2014				
Start time:	12:38:00				
Finish time:	14:18:00				
Duration:	01:40:00				

Timestamp	Activity
12:38:00	Starts. Reads documentation.
12:41:00	Runs EvoSuite.
	* Why? Out of curiosity about what the tool would generate. * Tests are numbered, not named after what they are testing. Says that tests do not excersise the method he wanted (decorate()), but there is actually a test that invokes it. Plans to go and flesh out the class a bit and then write some tests by hand.
12:43:00	Goes to target class to write the constructors and decorate() method.
	* Subject not very familiar with MacOS
12:45:00	Studies code.
12:46:00	Starts writing code.
12:47:00	Reads EvoSuite tests. Think about reusing one of them to test the code he just wrote.
12:49:00	Decides to implement methods following order of appearance.
12:51:00	Finishes one method (validate())
12:52:00	Inspects EvoSuite tests to see if any test tests his new method
	* Does not realise he should re-run EvoSuite. Should we have prompted him to do so?
12:53:00	Complains about EvoSuite tests being hard to read and understand.
	* He was looking at tests generated for the empty class * He believes that one of the reasons the test is confusing is because the target class is a very generic utility class.
12:55:00	* He realizes that the code was generated for the empty class.
12:56:00	Runs EvoSuite.
12:59:00	Acknowledges that tests have changed.
	* Still finds tests confusing, mainly because the type of arguments that are being passed and also because of the type of assertions generated (assertEquals({}, predicateMap.toString())).
13:01:00	Thinks he should write more code to allow EvoSuite to generate useful tests.
13:02:00	Decides to write his own tests.
13:08:00	Struggles to setup a test by hand.
13:10:00	Still having problems with the test setup.
13:14:00	Finishes test, duplicates it and runs both. Test fail expecting exceptions that didn't occur.
13:15:00	Realises about missed specification. Back to code to handle exception.
13:18:00	Runs tests again. Pass. Decides to write two more tests for the same method.
13:21:00	Runs tests. 4 passes.
13:21:00	Checks coverage. Happy with coverage of one method (100%).
13:22:00	Goes back to write more methods.
13:27:00	More behaviour written. Complains about documentation. Makes assumptions.
	* Method put() seems underspecified

13:30:00	Continues writing target class. Moves on to method putAll()
13:34:00	Believes implementation is finished.
13:35:00	Runs EvoSuite. Background.
13:36:00	Tries to refactor target class a bit.
	* Must run EvoSuite again since previous run was on his test suite, not target class.
13:37:00	Inspecting generated tests. Thinks they are too complex. Decides to go manual.
	* Less than two minutes observing EvoSuite generated tests before deciding to proceed manually
13:41:00	Writing manual tests.
13:42:00	Fixing target class.
13:43:00	Back to write tests manually
13:48:00	Runs tests. Fails. Correct and rerun tests.
13:50:00	Checks coverage. Some methods not covered (put() and putAll()).
	* Does not care to see if EvoSuite generated something useful for those methods
13:54:00	Writes manual tests for put()
13:56:00	Checks coverage. It has improved.
13:57:00	Recycles test written for put() and use it to test putAll()
14:00:00	Continues writing tests to comply with specification.
14:02:00	Runs tests. 100% coverage.
	* Plays with instruction, branch, complexity coverage
14:04:00	Runs EvoSuite.
	"To check if it has created any additional test or tests that look a little bit more meaningful."
14:06:00	Runs EvoSuite tests.
14:07:00	Inspects generated tests.
	* EvoSuite missed two branches. 87.5% branch cov.
14:09:00	Revisits implementation.
14:11:00	Happy with result.
14:11:00	Retrospective phase.
	Readability: Names should be better. Points out difference between TCG for "utility" classes like Map and more specific purpose classes. Value of tests lie on readability.
	"EvoSuite seems to be working fine. It would be easier for human beings to have the tests with names so as to know what method is EvoSuite trying to test" "Maps are very generic utility classes. For an automated tool to infer any desired behaviour is much harder than inferring behaviour from a class that has a very specific purpose" "I got scared by the tests generated by EvoSuite." "The value of the tests, I think, lies on the readability" "Test coverage is easy to assess because it's a number, while readability of the tests is a very non-tangible property. What is readable to me may not be readable to you. It's readable to me just because I spent the last hour and a half doing this." "The tool itself looks impressive. It's a nice size to implement this class, but perhaps something less generic would be better for the programmer to see the value of automated test generation."
14:19:00	Finishes