Responsibility Driven Design with Mock Objects

Tell, don't ask!

Rob Westgeest - rob@qwan.it
Marc Evers - marc@qwan.it
Willem van den Ende - willem@qwan.it

XP Days Benelux, 20 November 2008

Agenda

- About us
- Responsibility driven design
- Mocking & test driven development
- Demo
- Guidelines & Issues
- Summary

About us

Independent
(Agile) software development coaches
Trainers
Facilitators
Consultants
Developers

QWAN
Quality Without A Name
www.qwan.it

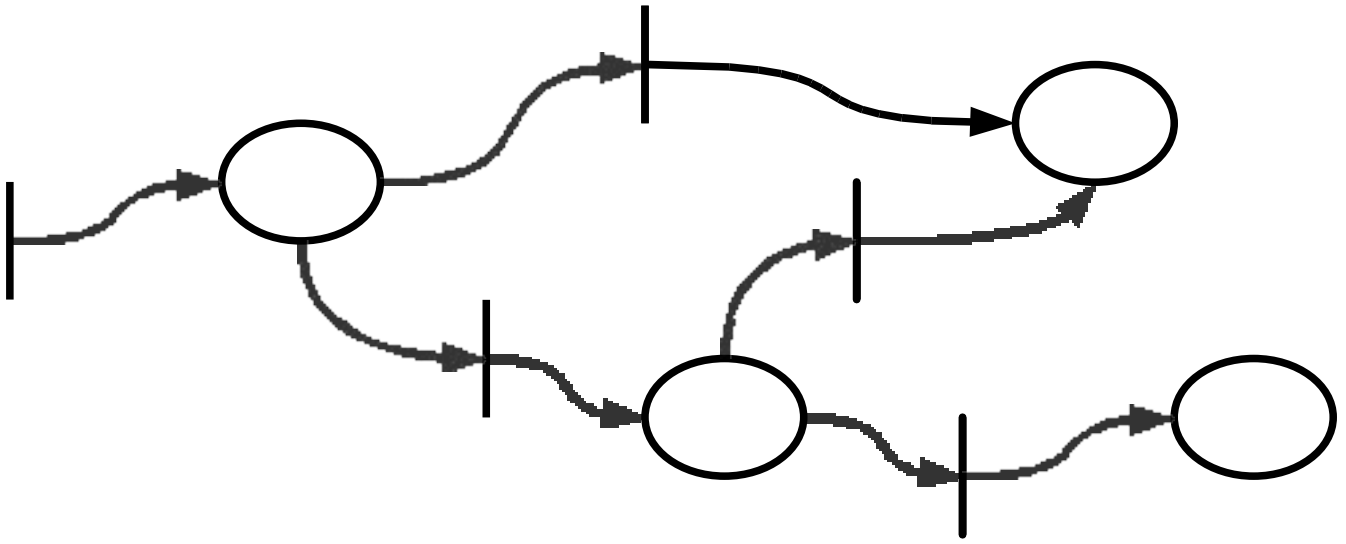
www.agileopen.net

AGILE HOLLAND
www.agileholland.com

XP DAYS 2008 BENELUX
www.xpday.net

Responsibility Driven Design

object has *responsibilities* and *collaborates* with others to fulfill its responsibilities



objects do stuff they can....

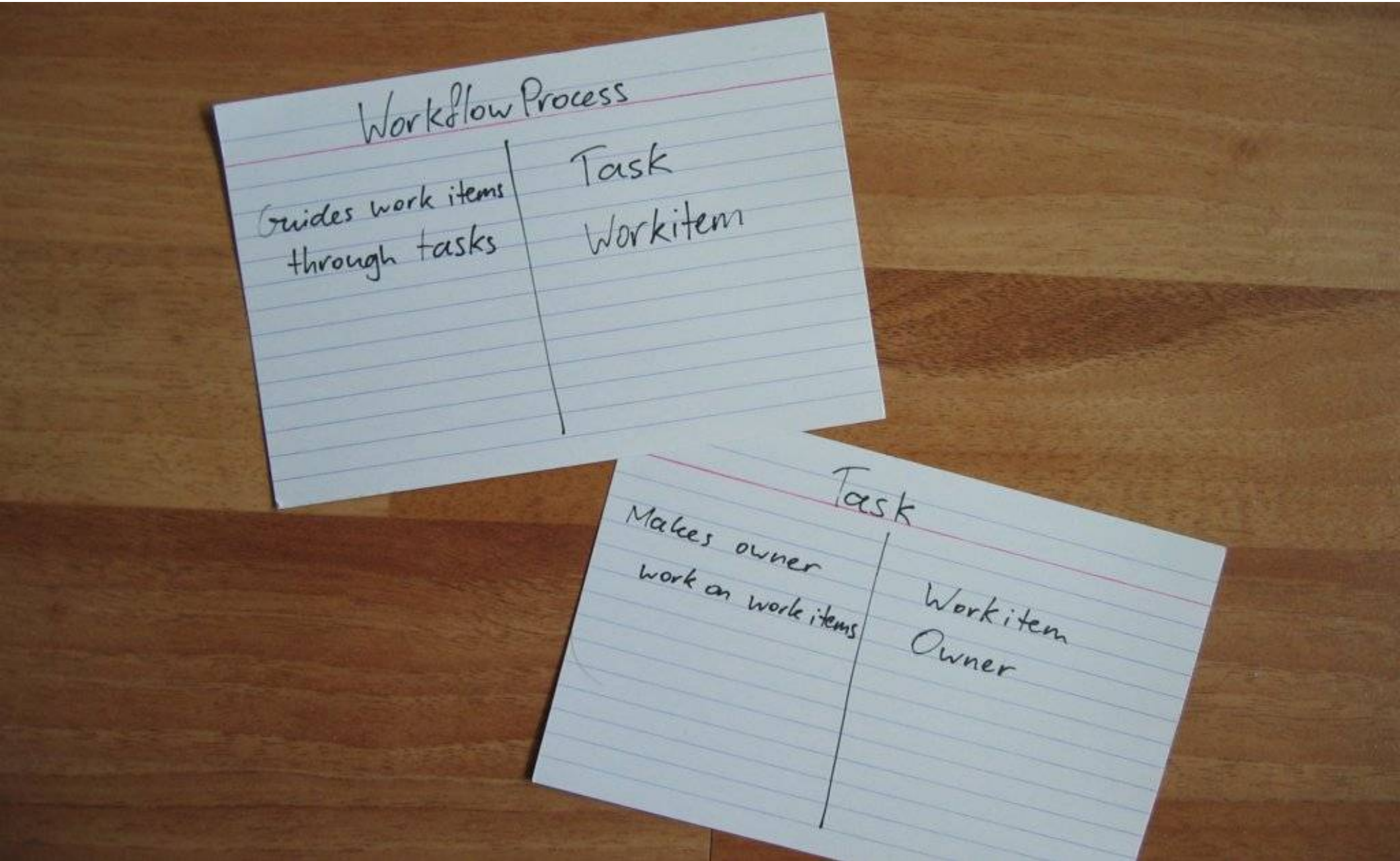
objects do stuff they can....

... and delegate the rest

Responsibility Driven Design

- Describe object in terms of
  - services it provides
  - services it requires from collaborators
- Focus on behaviour, roles & interactions
  - not data and state

CRC cards


Workflow Process
Guides work items through tasks
Task
Workitem
Task
Makes owner work on work items
Work item
Owner

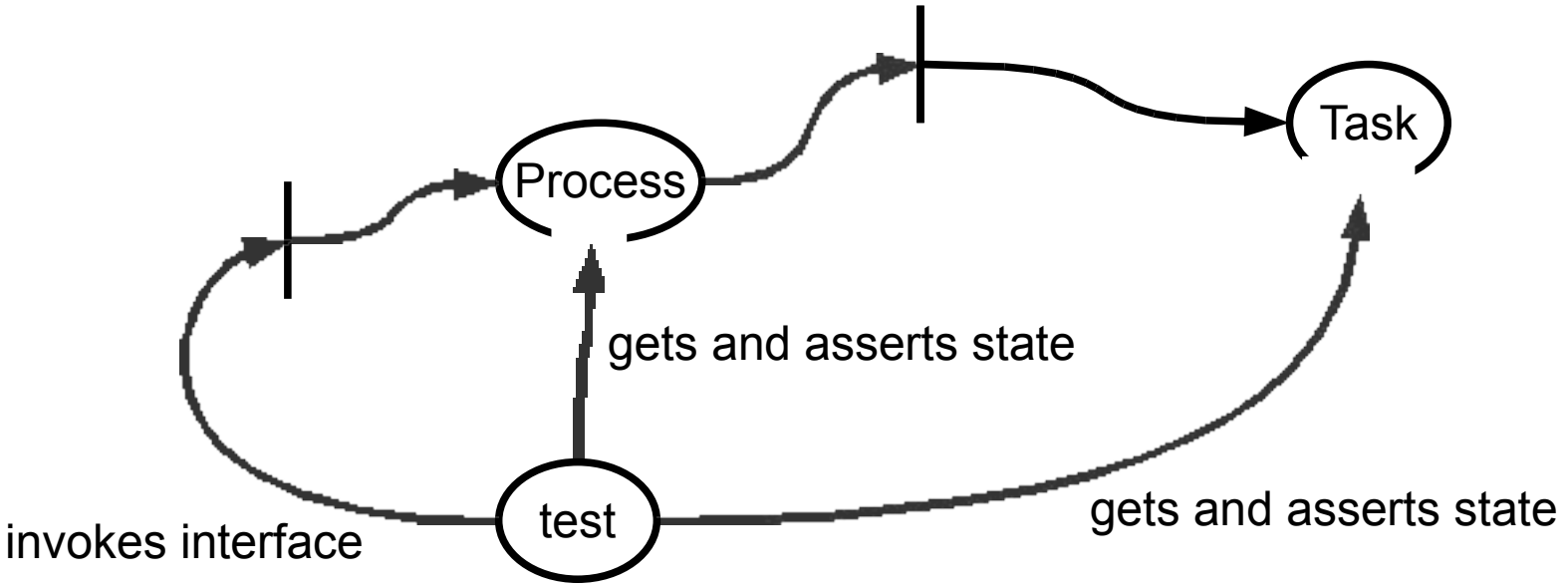
Tell, don't ask

Test driven development

- Tests drive design
- Design activity
- ...and you get a lot of tests too!

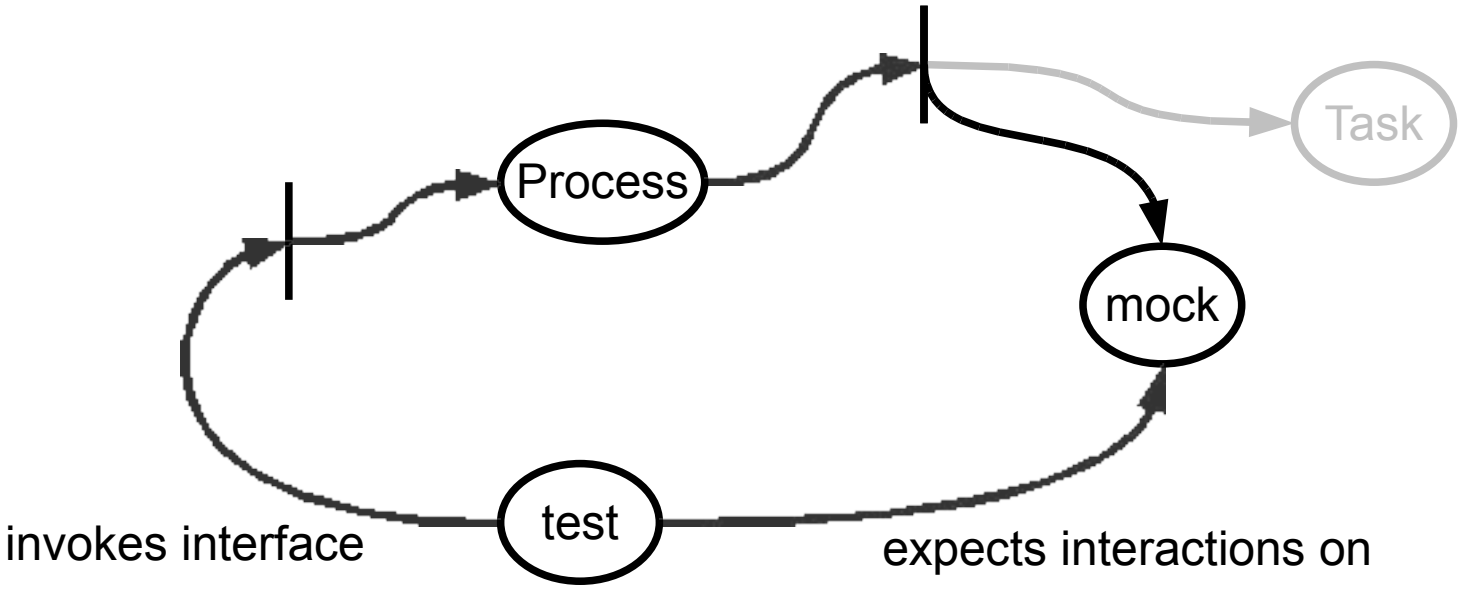
State based testing

```
testStartWorkflowProcess {
   task = new SimpleTask();
   process = new WorkflowProcess(task);
   process.start(workItem);
   assertTrue(task.isStarted());
   assertSame(workItem, task.getWorkItem());
}
```



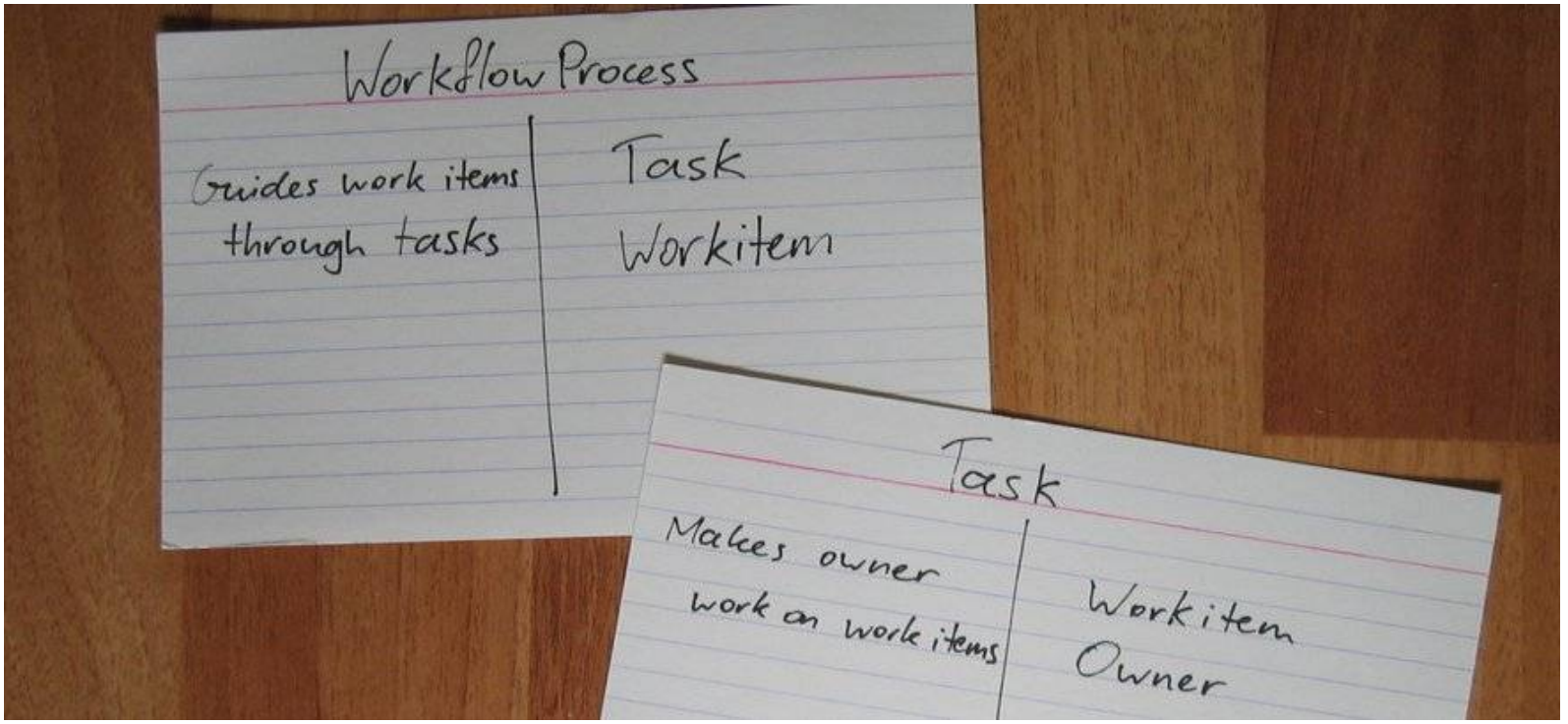
Interaction based testing

```
startingWorkflowInitiatesTask {
   task = new mock();
   process = new WorkflowProcess(task);
   task expects start with workItem
   process.start(workItem);
}
```



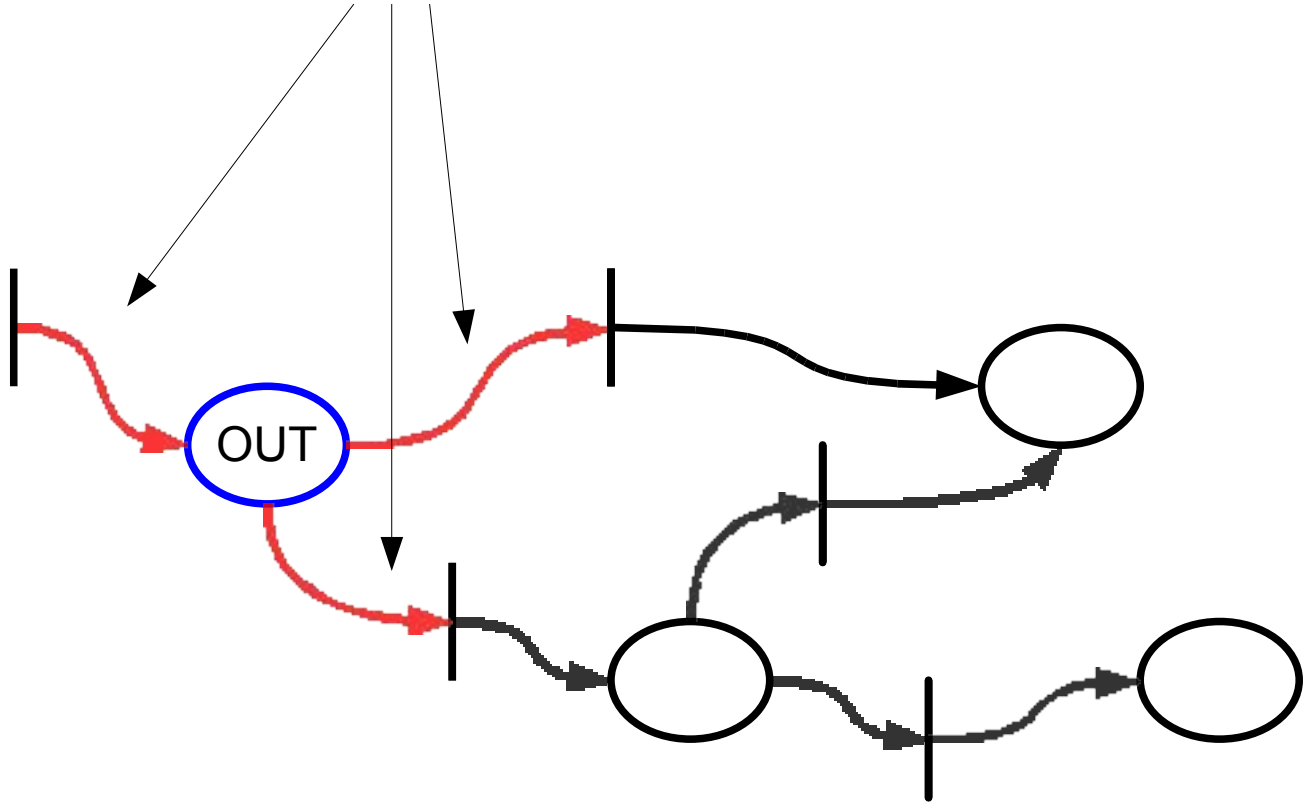
Responsibility driven testing

```
startingWorkflowInitiatesTask {
    task = new mock();
    process = new WorkflowProcess(task);
    task expects start with workItem
    process.start(workItem);
}
```

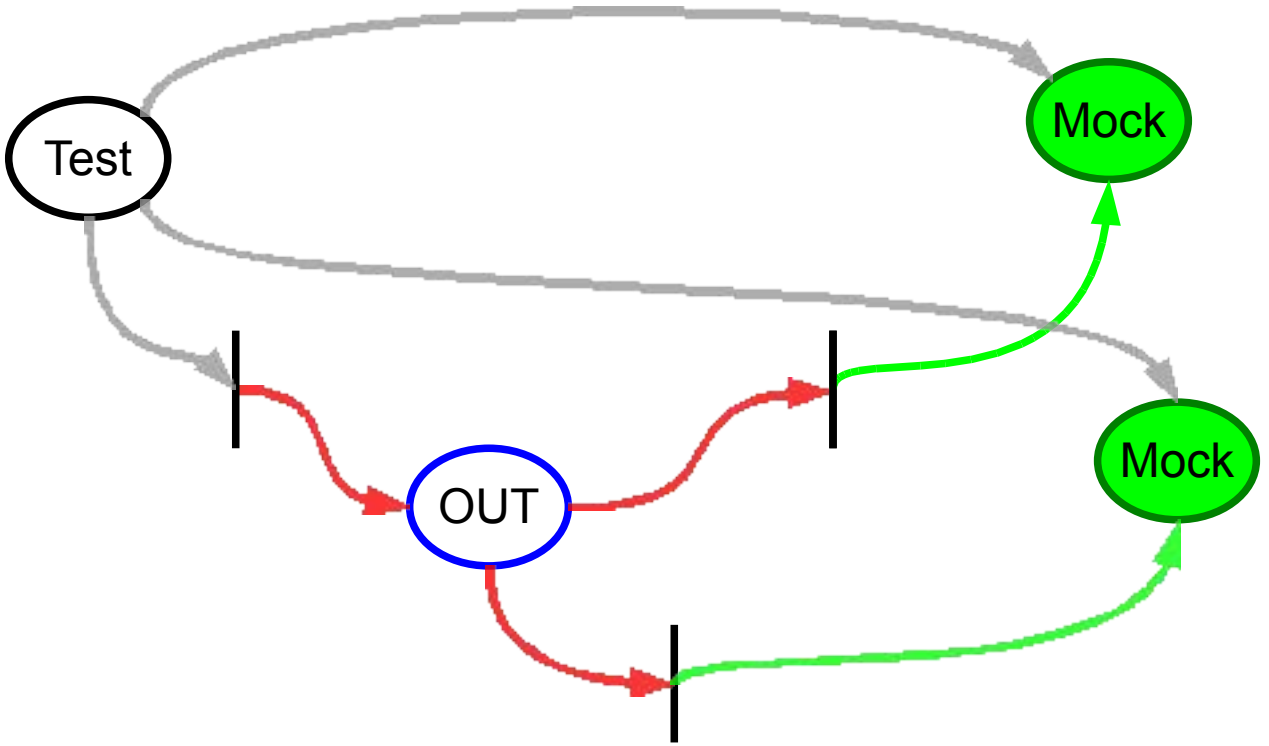


Testing with mocks

test interactions rather than state changes

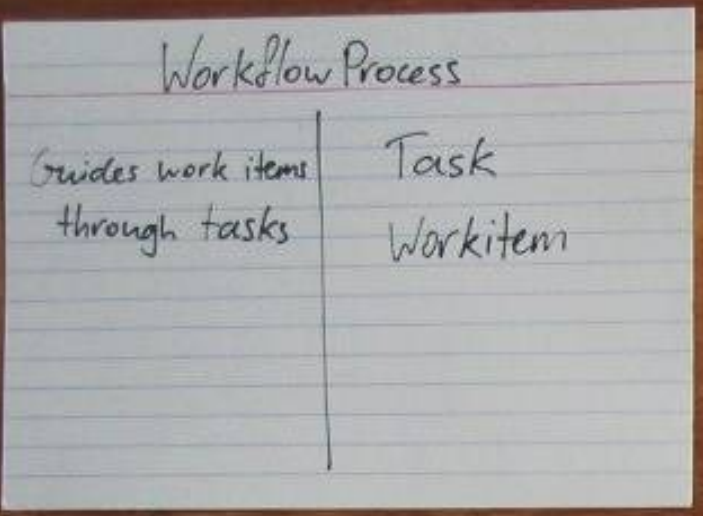


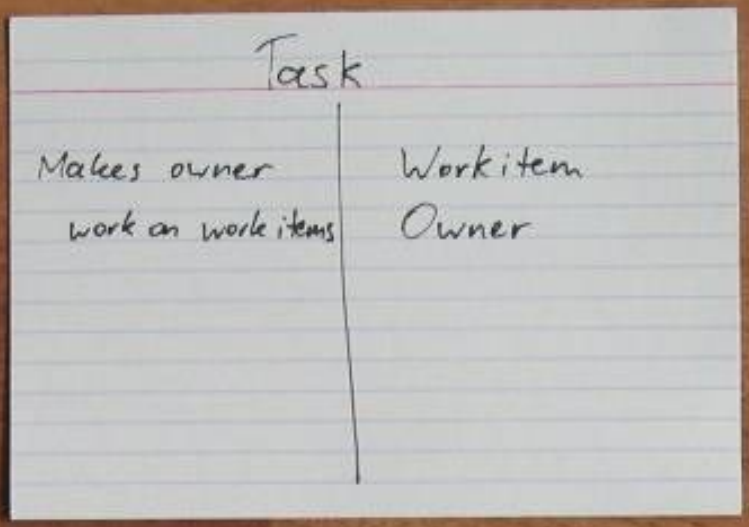
Testing with mocks

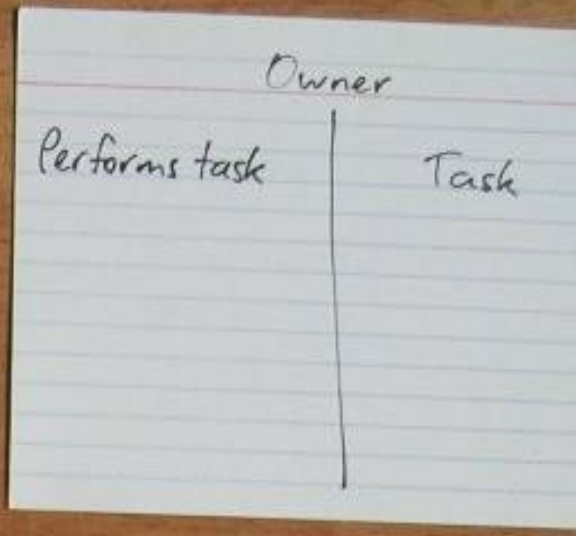


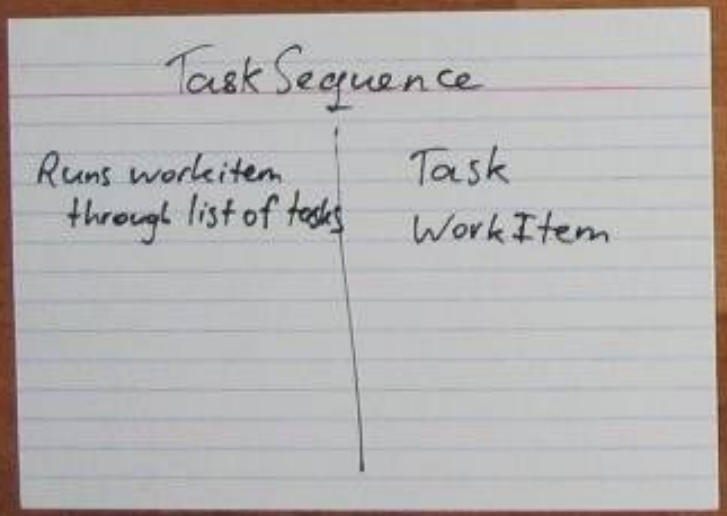
Mock

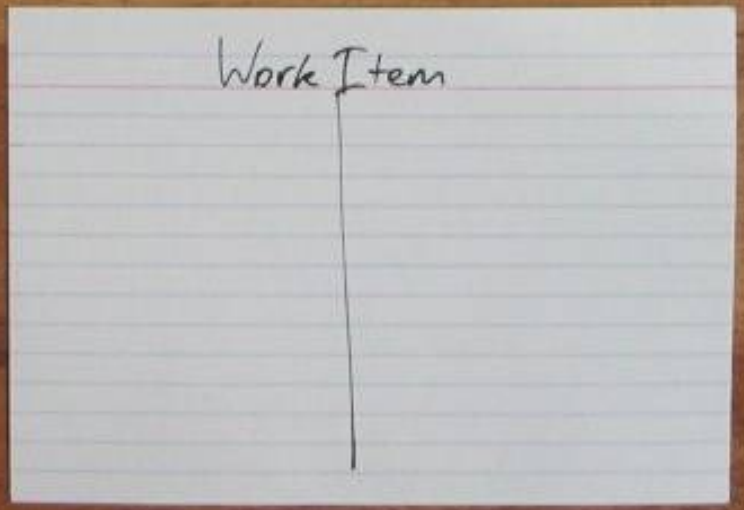
object pre-programmed with expectations which form a specification of the calls it is expected to receive


Workflow Process
Guides work items through tasks
Task
Workitem


Task
Makes owner work on work items
Workitem
Owner


Owner
Performs task
Task


Task Sequence
Runs workitem through list of tasks
Task
WorkItem


Work Item

Mockito

```
public class WorkflowProcessTest {

    @Test
    public void startingAProcessInitiatesATask() {


        WorkflowProcess process = new WorkflowProcess(task);

        process.start(workItem);


    }
}
```

Mockito

```
public class WorkflowProcessTest {

    @Test
    public void startingAProcessInitiatesATask() {



        WorkflowProcess process = new WorkflowProcess(task);

        process.start(workItem);

        verify(task).start(workItem);
    }
}
```

Mockito

```
import static org.mockito.Mockito.*;

public class WorkflowProcessTest {

    @Test
    public void startingAProcessInitiatesATask() {
        Task task = mock(Task.class);
        WorkItem workItem = mock(WorkItem.class);

        WorkflowProcess process = new WorkflowProcess(task);

        process.start(workItem);

        verify(task).start(workItem);
    }
}
```

Live demo!

Steps

1. Starting a Process should initiate a Task
2. Starting a SimpleTask should notify its Worker
3. Finishing a Task should notify its Parent
4. Finishing all tasks in a process should finish WorkItem
5. Starting a sequence task should start first task in sequence
6. Finishing a task in a sequence should start next task
7. Last finishing task in sequence should notify its parent
8. Starting an empty sequence should notify its parent directly that it is done
9. A Worker should notify the responsible person by email

Guidelines

- Tell, don't ask
  - getters are evil
  - no asserts
- Be also explicit what should *not* happen
- Specify as little as possible
- Don't add behaviour to mocks
- Only mock types you own
  - use wrappers for others
- Don't mock everything

Issues

- Duplicating implementation in tests
- Brittle tests
- Keeping specs and implementation in sync
- Too many mocks or complex mock setup
  - hint: too many dependencies ...
- Making assumptions about collaborators
  - might turn out to be wrong
- Too many interfaces

Summary

- Taking TDD a step further
- OO is about responsibilities, not state
  - focus on roles & interactions
- More specific tests
- Better design
  - low coupling, high cohesion
- Beware of brittle tests
- Don't mock what you don't own

More information

www.jmock.org, www.mockito.org

Steve Freeman, Nat Pryce, Tim Mackinnon, Joe Walnes, *Mock Roles, not Objects* (2004)

Martin Fowler, *Mocks Aren't Stubs* (2004)

Rebecca Wirfs-Brock & Alan McKean, *Object Design: Roles, Responsibilities, and Collaborations* (2003)

Rob Westgeest & Marc Evers, *Responsibility Driven Design met Mock Objects* (2008)

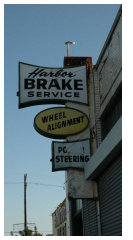
Michael Feathers, *Working effectively with Legacy Code* (2004)

Mocking in other programming languages:
Mocha (Ruby), NMock (.NET)

Credits

Train Wreck - Prob. Early 1900's © rcstanley (cc-by)
http://flickr.com/photos/rcstanley/30366555/

SP Signs - Harbor Brake Service © Marshall Astor (cc-by-sa)
http://flickr.com/photos/lifeontheedge/125657086/



traffic light obsession.3 © slopjop (cc-by-sa)
http://flickr.com/photos/slopjop/2391468254/

books in a stack (a stack of books) © austinevan (cc-by)
http://flickr.com/photos/austinevan/1225274637/