An Introduction to text-based test automation and the TextTest tool

JEPPESEN
Making Every Mission Possible

Contentions

1. That there are circumstances where xUnit-style testing isn't the best choice.
2. That the text-based approach is an obvious alternative candidate in many of these cases.
3. That there are advantages to operating Acceptance testing in this way even in general.
4. That text-based test-driven development is possible and even desirable.
5. That TextTest is the best free tool out there that tests this way.

Agile test automation == The API-assertion paradigm?

```
public void testAdd() {
   Number x = new Number(1);
   Number y = new Number(1);
   Number result = new Number(2);
   Assert.assertEquals(x.add(y), result);
}
```

- Classic, near universal xUnit
- We assume an API to the Number object
- We assert that it returns certain hardcoded values

JEPPESEN
Making Every Mission Possible

Acceptance test tools have the same approach

Fixtures.Addition		
x	y	add()
1	1	2
1	2	3

- Fit table. Because customers don't write code.
- So we write them a "fixture", give them a table and let them fill in the numbers.
- Under the covers it's more or less xUnit with variable data and a nice interface.

Example applications where this paradigm is less than ideal

- ✘ **Anything in a non-mainstream language.**
- ✘ **UNIX-style command-line scripts. Wide language variety (cshunit anyone?). Command line/textual output key parts. Design often haphazard.**
- ✘ **Legacy systems. Often legacy language. Design optional. Retrofitting APIs possible but hazardous. Correct behaviour maybe unknown.**
- ✘ **Jeppesen's airline crew schedule planner. Correct behaviour subjective and volatile. Large amounts of data for interesting tests.**

JEPPESEN
Making Every Mission Possible

(Re-)Introducing
The text-based paradigm

config.expr:
```
executable:/usr/bin/expr
```

options.expr:
```
1 + 1
```

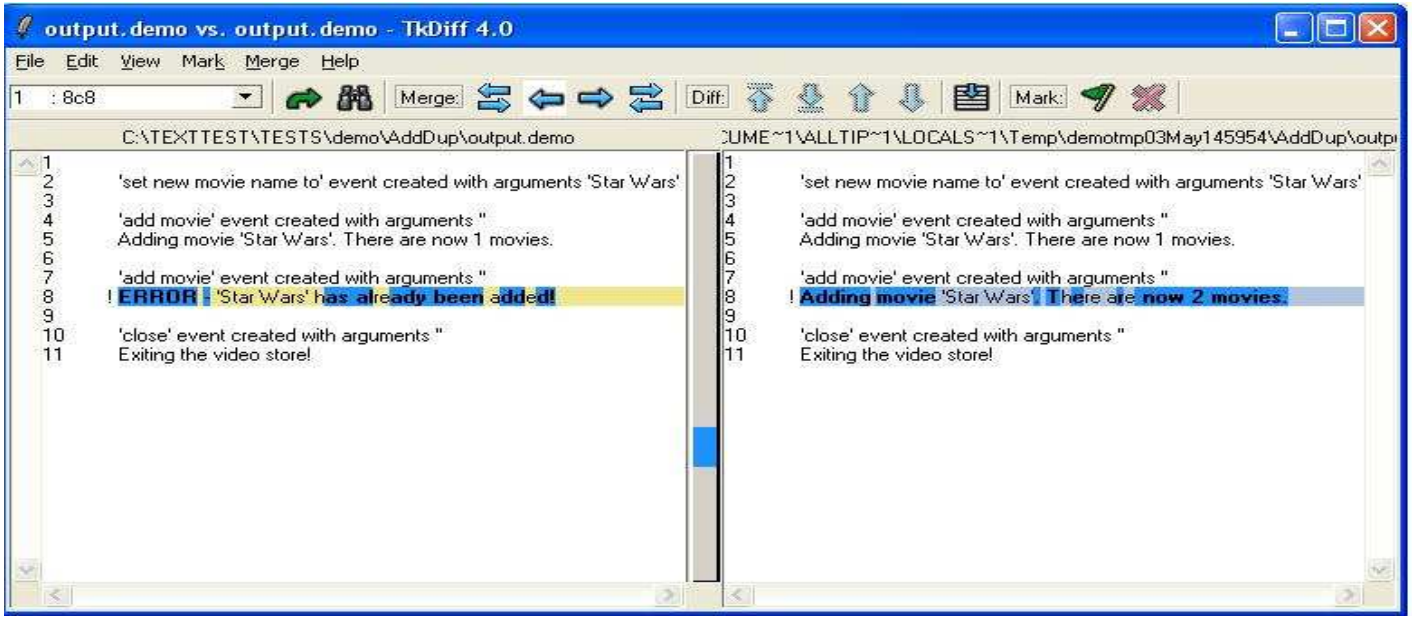
output.expr:
```
2
```

- Run the system under test from the command line
- Define tests in terms of different command lines
- Compare produced text files to equivalent files from previous runs.

(and save them when appropriate)

JEPPESEN
Making Every Mission Possible

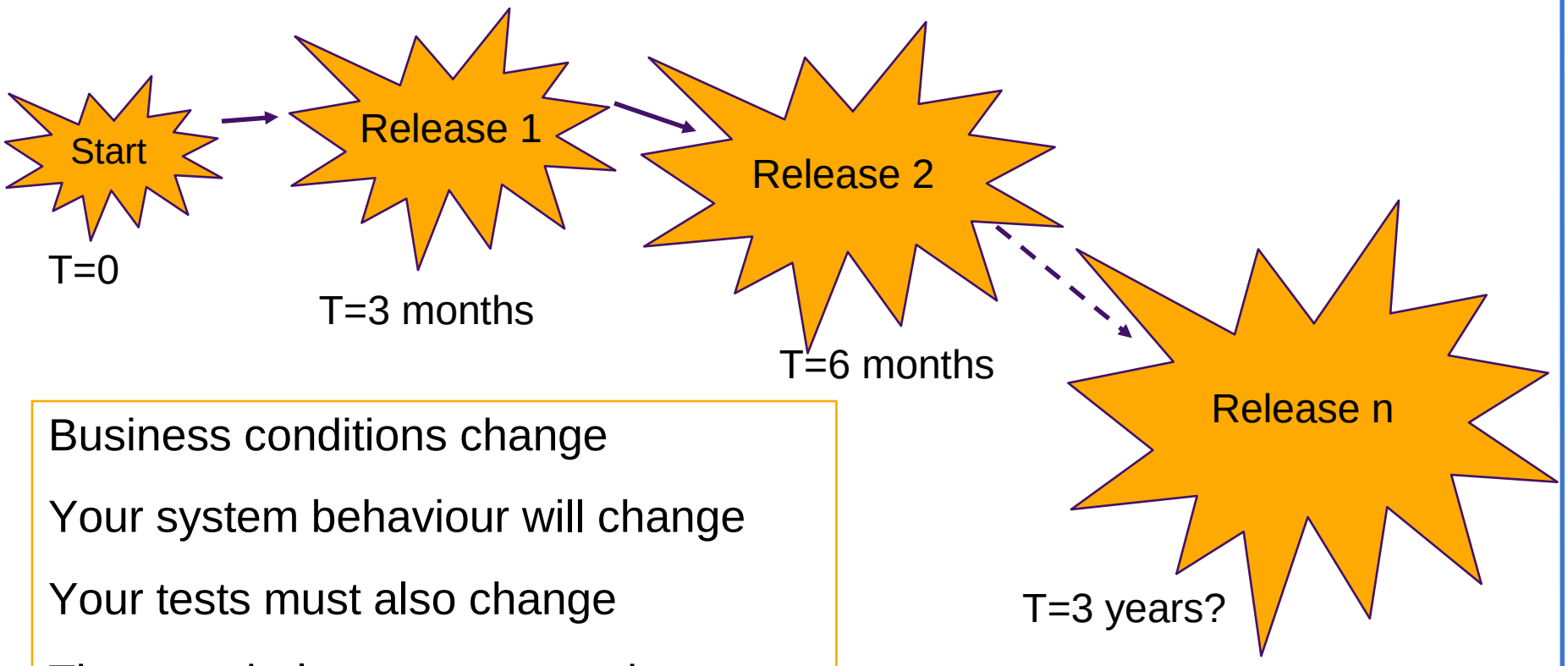
Behaviour Change Management by Comparing Plain Text



- Use produced “result files” and internal logs as a measure of system behaviour.
- Invest in them so they are easy to read and have the right level of detail.
- Testing becomes a matter of *Behaviour Change Management*.

JEPPESEN
Making Every Mission Possible

Seeing testing as Behaviour Change Management – not Correctness Assertion



Business conditions change

Your system behaviour will change

Your tests must also change

The tests help you *manage* change

What could be more Agile?

JEPPESEN
Making Every Mission Possible

Assumptions of the text-based paradigm

- **Can have one tool for all languages past, present and future**
- **Do not compel tests to bypass any parts of the system**
- **Can handle any amount of data**
- **Independent of the system design and APIs**
- **Does not require any hand-coded assertions**

But we still have (different) assumptions...

- ✓ **That system logging will not be a performance burden**
- ✓ **That the system produces suitable files, or can easily be made to.**
- ✓ **That the developers know what to log and how to log it**
- ✓ **That the command line interface and logging are not too volatile**

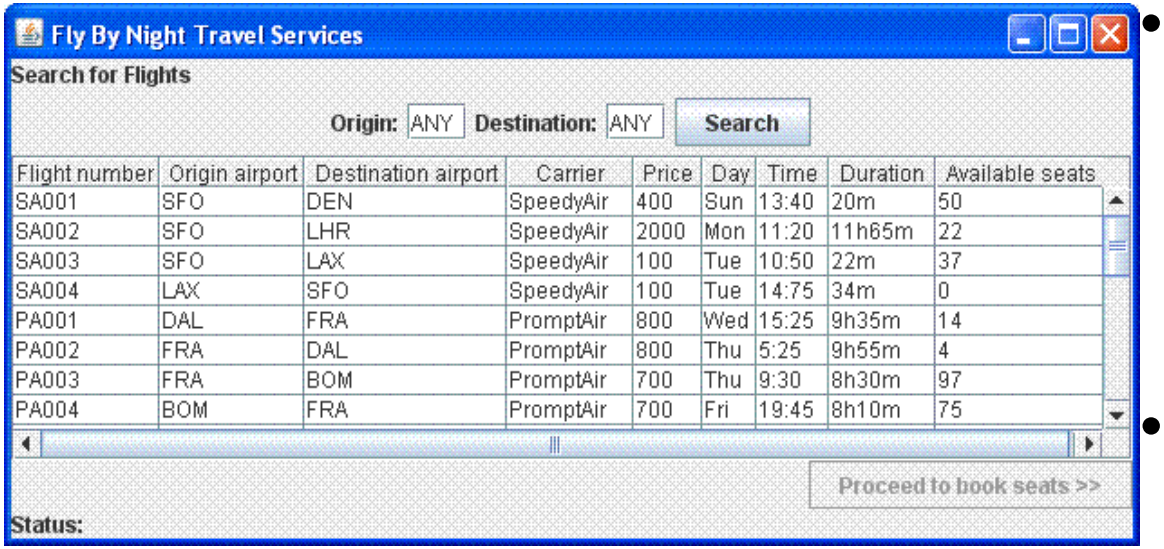
JEPPESEN
Making Every Mission Possible

What is TextTest?

- A tool for automated text-based testing
- Free, open source and written in Python (GUI written with PyGTK).
- Works on UNIX systems and Windows XP/Vista
- System under test is driven via the command line.
- Compares text files produced by the system under test against those produced by previous runs
- Continually developed and improved by Jeppesen.
- Short demo follows…

JEPPESEN
Making Every Mission Possible

Text-based Acceptance Testing and the GUI

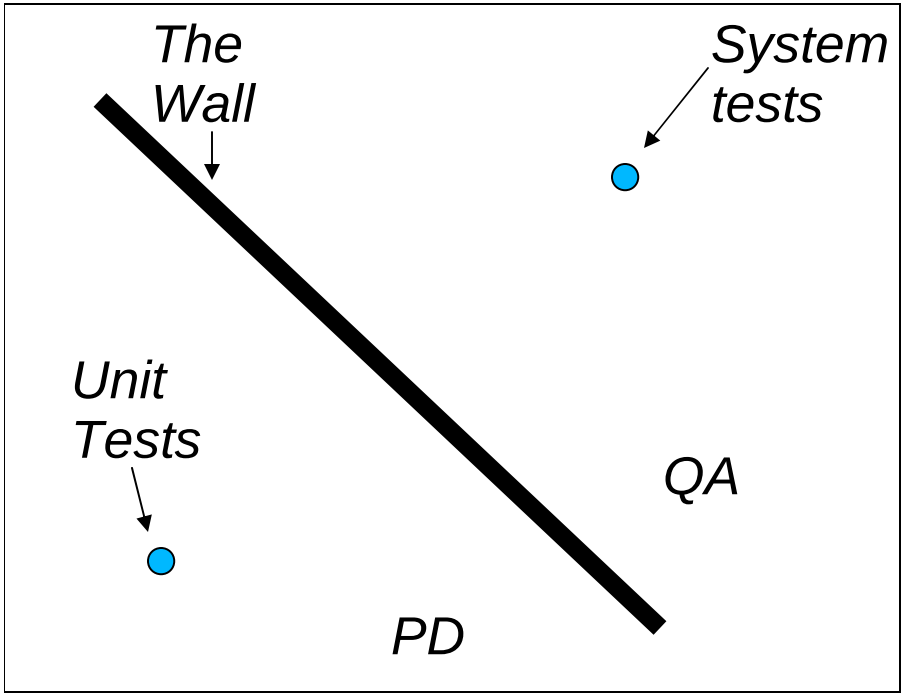



```
wait for flight information to load
select flight SA004
proceed to book seats
accept error message
quit
```

- Use, or write, a record/replay library that can map GUI controls to a domain language (e.g. xUseCase)
- Testers record sensible use-cases and critique system behaviour.
- Developers make sure system logs everything that can be observed externally.

JEPPESEN
Making Every Mission Possible

Text-based testing for the whole team

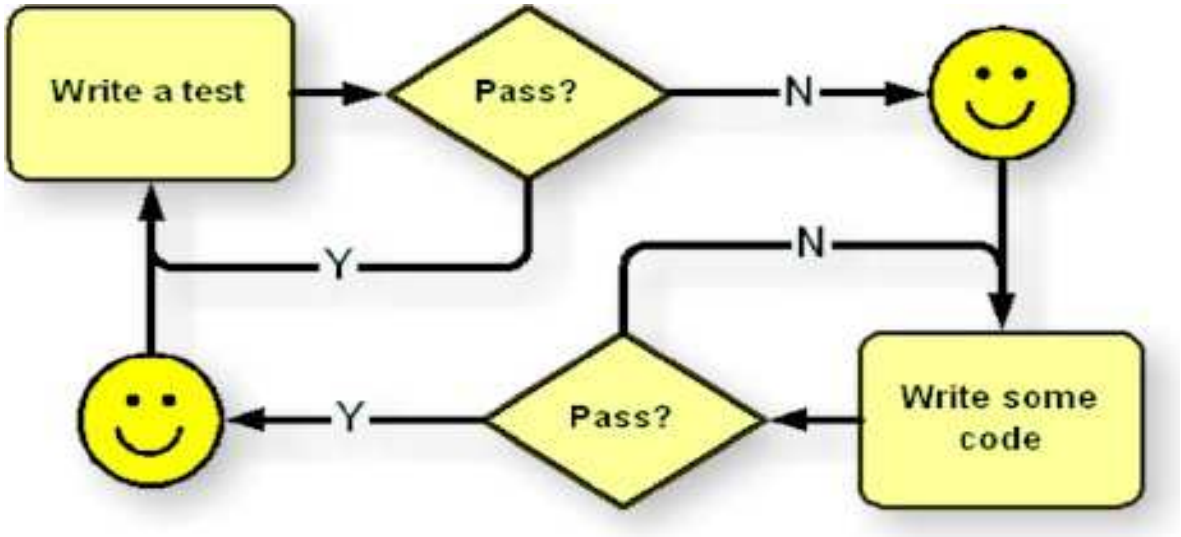


How automated tests are often organised.

- Mr Team Leader - tear down this wall!
- The tests only run at the system level, but we can log at any level.
- Developers also create logs of lower level detail behaviour
- These are normally disabled but can be easily enabled for debugging
- Gradually build a knowledge base, unlike using debuggers

JEPPESEN
Making Every Mission Possible

Test-Driven Development



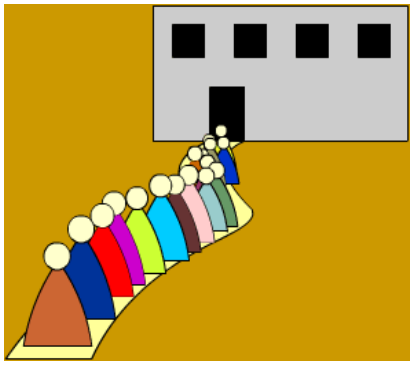
- We want to provide rapid feedback to the developer at every build
- These tests are usually unit tests, but what if they weren't?
- Wouldn’t it be nice to use our acceptance tests in this role?

Text-based test-driven development (really behaviour-driven development)

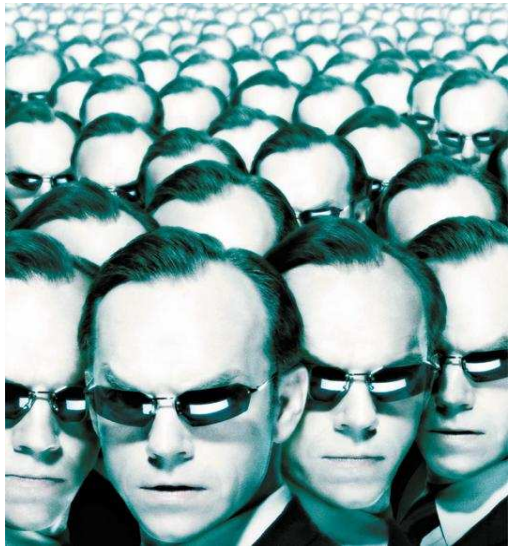
- Vital to take macro-level effects and the business perspective into effect when testing
- Greatly enhance our feedback if we see them every build instead of every iteration/release
- Enhance communication between developers and domain experts
- Everything is text files, so easy to write test sketches or suggest test changes by just editing them by hand
- But some things will be rather different…

JEPPESEN®
Making Every Mission Possible

“But system tests will be way too slow to run at every build”



- Maintaining mock test environments or interdependent tests consumes human resources.
- Running tests in parallel consumes hardware resources. And hardware is cheap.



- Grid Engines are widely available, easy to use, and often free. TextTest integrates with SGE and LSF.
- Running more tests faster becomes simply a matter of buying more hardware.

JEPPESEN
Making Every Mission Possible

“But I want to use my tests to drive my design!”

- Customer-perspective tests are unlikely to help drive design or prevent overdesign
- Much overdesign can be limited by “Usage-driven design”
- Before writing a class or method, write the code that make use of it (calls it).
- Ultimately, good design is mostly driven by thinking hard and refactoring well.

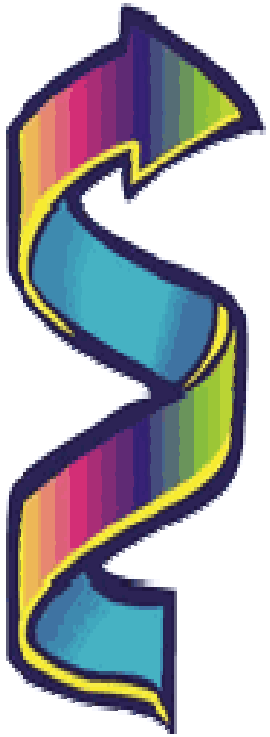
JEPPESEN®
Making Every Mission Possible

“But I want to verify unit behaviour before trying to run the system!”

- When the intended behaviour of a method, class or subsystem is clear it can be useful to test it on its own.
- We can reach for the interactive interpreter or the “main method” for this purpose.
- Aim is to drive development, not to bake microdesign decisions into the test suite.
- So we only make permanent the test that makes sense in a wider and longer-term perspective.

JEPPESEN®
Making Every Mission Possible

“What ever happened to test-first?”



- Unit-tests are predictive in that they assert up-front exactly what is expected.
- For larger-scale or requirements-focussed tests this is often not possible.
- Specify the *test interaction* first, but be prepared to frequently adjust the *test behaviour* as the problem is fleshed out.
- Testing then becomes less about correctness assertion and more about *behaviour change management*.

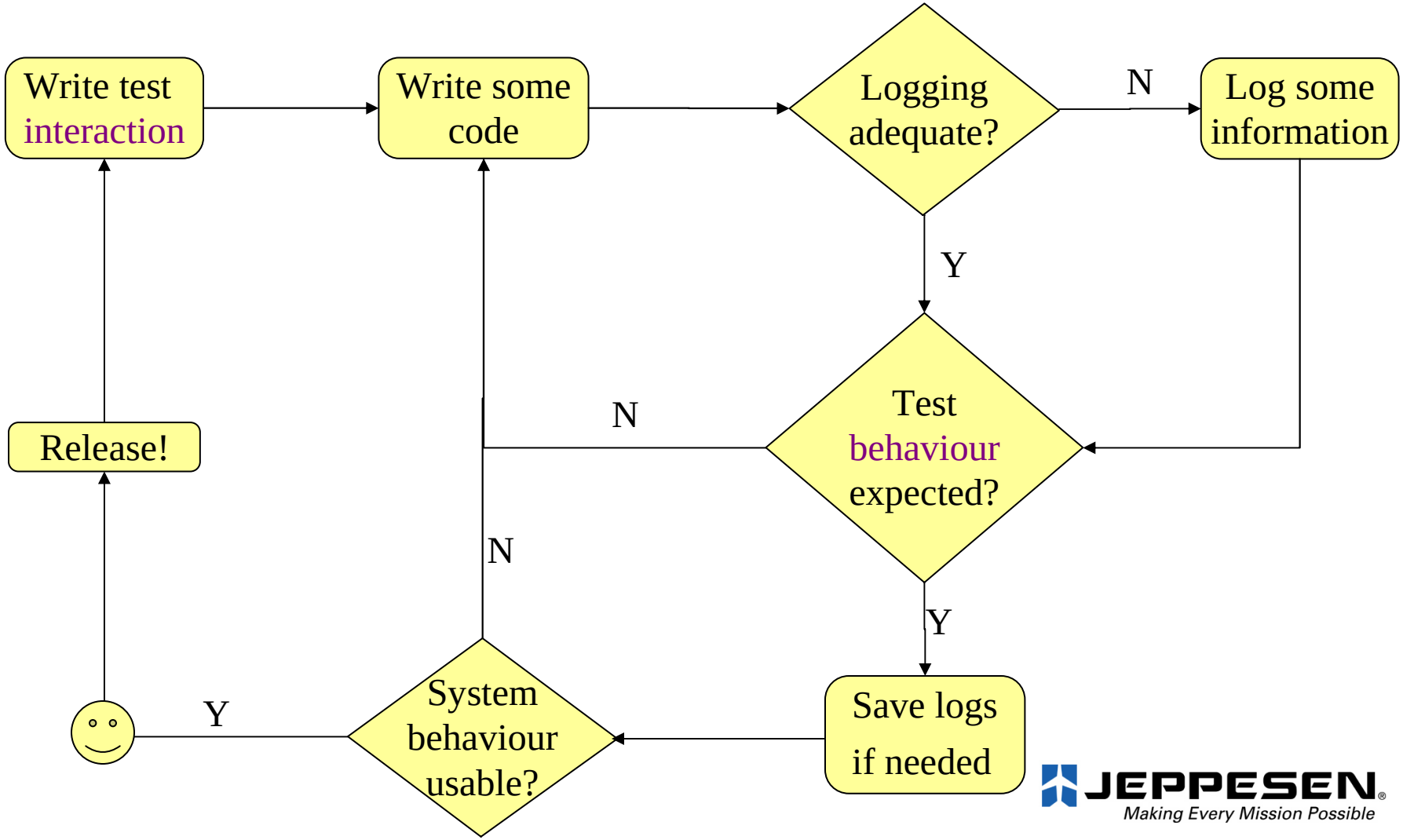
JEPPESEN®
Making Every Mission Possible

“What about the ‘Expert Reads Output’ antipattern?”

- ”You need a person to read the output. But people are lazy and after a while will miss things. Then your tests are proving the wrong thing!”
- Configure TextTest to find error messages automatically.
- Don't read the logs, observe the system!
- TextTest will group similar changes.
- Behaviour Change Management : changes are more important than contents.

JEPPESEN
Making Every Mission Possible

Text-based ATDD : the process



JEPPESEN
Making Every Mission Possible

Conclusions

- Acceptance tests in the development cycle provide feedback on both macro effects and business perspective
- There are obstacles but all can be overcome:
  - slowness best conquered by parallelism
  - design can still be driven, top-down
  - need to relax the test-first concept and introduce that of Behaviour Change Management
- Text-based testing is a much-underrated technique which works well in this role
- Tool support available in the form of TextTest.

JEPPESEN
Making Every Mission Possible

TextTest Features

- Filters output to avoid false failure
- Manages test data and isolation from global effects
- Automatic organisation of test failures
- “Nightjob website” to get a view of test progress over time
- Performance testing
- Integrates with Sun Grid Engine for parallel testing (and LSF)
- Various “data mining” tools for automatic log interpretation
- Interception techniques to automatically “mock out” third-party components (command line and network traffic).
- Integrates with xUseCase tools for GUI testing

JEPPESEN
Making Every Mission Possible

The coding exercise

- We are now going to try to use the techniques outlined here to solve a toy coding problem in Python.
- We will not use unit testing as well. This is primarily to keep things simple.
- We will proceed from simple behaviour and manage the changes until it does what we want. We won't expend (much) effort predicting behaviour in advance.
- We will try to design top-down, but will use the python interpreter for bottom-up exploration when appropriate.
- You, the audience, should interrupt if:
  - We take larger steps than you're comfortable with
  - We over-design the code, or don't drive design from actual usage
  - We don't test it adequately

JEPPESEN
Making Every Mission Possible