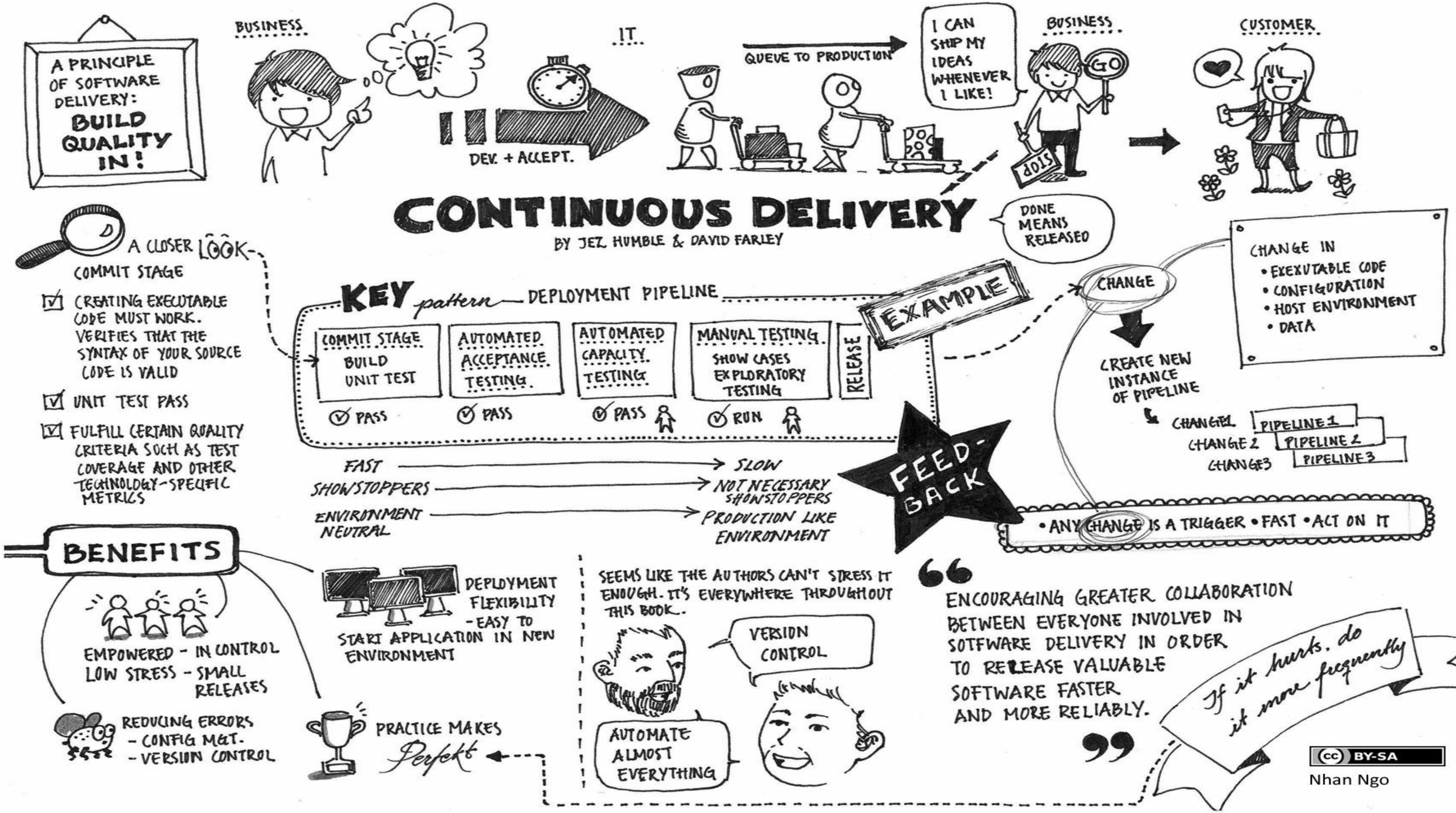
Jenkins

...

Main agenda

- Short brief on testing, some terms
- Nice lovely graphics (!) nicely showing why you want continuous integration and continuous delivery (ty Nhan Ngo at Spotify)
- Install jenkins
- Install the php-template
- Use php-template on an example project
- Have tests run on pull requests before merging into mainline
- “Deploy code”



CONTINUOUS DELIVERY

BY JEZ HUMBLE & DAVID FARLEY

A PRINCIPLE OF SOFTWARE DELIVERY: **BUILD QUALITY IN!**

BUSINESS

IT

DEV. + ACCEPT.

QUEUE TO PRODUCTION

I CAN SHIP MY IDEAS WHENEVER I LIKE!

BUSINESS

GO

STOP

CUSTOMER

DONE MEANS RELEASED

A CLOSER LOOK

COMMIT STAGE

☑ CREATING EXECUTABLE CODE MUST WORK. VERIFIES THAT THE SYNTAX OF YOUR SOURCE CODE IS VALID

☑ UNIT TEST PASS

☑ FULFILL CERTAIN QUALITY CRITERIA SUCH AS TEST COVERAGE AND OTHER TECHNOLOGY-SPECIFIC METRICS

KEY *pattern* — DEPLOYMENT PIPELINE

COMMIT STAGE	AUTOMATED ACCEPTANCE TESTING	AUTOMATED CAPACITY TESTING	MANUAL TESTING	RELEASE
BUILD UNIT TEST			SHOW CASES EXPLORATORY TESTING	
✓ PASS	✓ PASS	✓ PASS	✓ RUN	

EXAMPLE

FAST → SLOW

SHOWSTOPPERS → NOT NECESSARY SHOWSTOPPERS

ENVIRONMENT NEUTRAL → PRODUCTION LIKE ENVIRONMENT

FEED-BACK

CHANGE

CHANGE IN
- EXEXUTABLE CODE
- CONFIGURATION
- HOST ENVIRONMENT
- DATA

CREATE NEW INSTANCE OF PIPELINE

CHANGE 1 — PIPELINE 1

CHANGE 2 — PIPELINE 2

CHANGE 3 — PIPELINE 3

• ANY CHANGE IS A TRIGGER • FAST • ACT ON IT

BENEFITS

EMPOWERED - IN CONTROL

LOW STRESS - SMALL RELEASES

DEPLOYMENT FLEXIBILITY - EASY TO START APPLICATION IN NEW ENVIRONMENT

REDUCING ERRORS - CONFIG MGT. - VERSION CONTROL

PRACTICE MAKES *Perfekt*

SEEMS LIKE THE AUTHORS CAN'T STRESS IT ENOUGH. IT'S EVERYWHERE THROUGHOUT THIS BOOK.

VERSION CONTROL

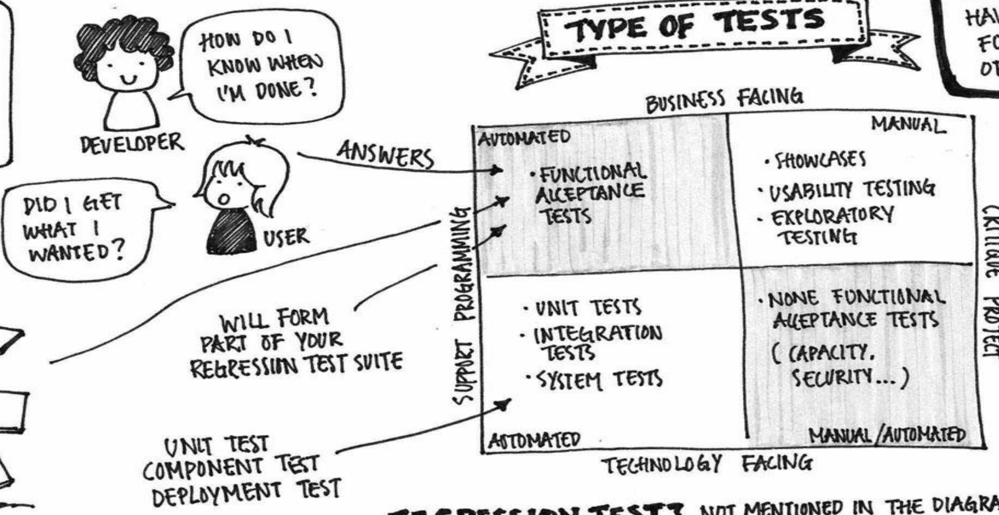
AUTOMATE ALMOST EVERYTHING

“ ENCOURAGING GREATER COLLABORATION BETWEEN EVERYONE INVOLVED IN SOTFWARE DELIVERY IN ORDER TO RELEASE VALUABLE SOFTWARE FASTER AND MORE RELIABLY. ”

If it hurts, do it more frequently

What does CONT. DEL. *say about* TEST STRATEGY

"TESTING IS A CROSSFUNCTIONAL ACTIVITY THAT INVOLVES THE WHOLE TEAM, AND SHOULD BE DONE CONTINUOUSLY FROM THE BEGINNING OF THE PROJECT."



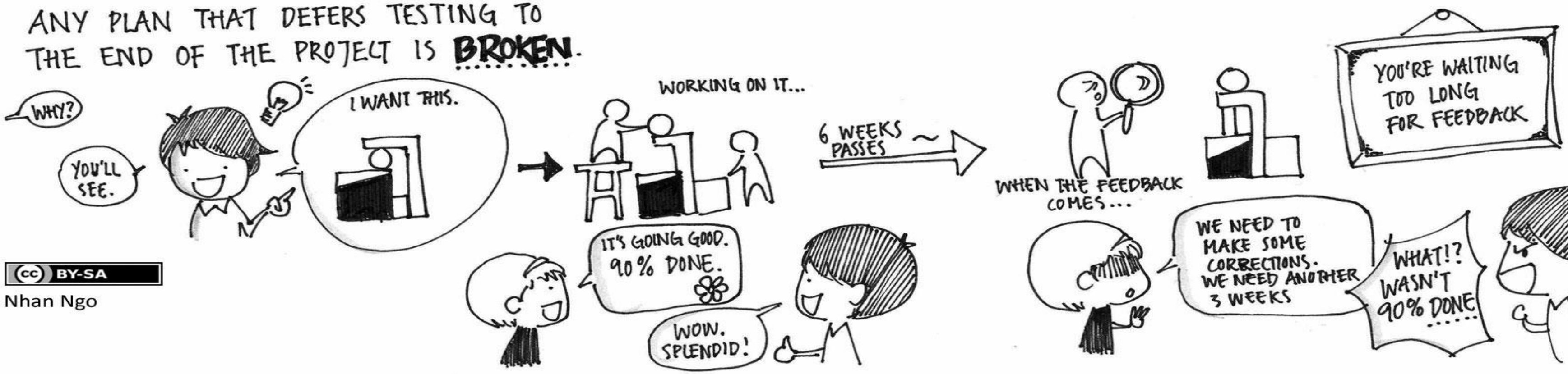
YOUR DEPLOYMENT PIPELINE SHOULD HAVE ALL THESE FOUR TYPE OF TESTS.

? NOT MUCH INFORMATION REGARDING THIS TYPE OF TESTS IN THE BOOK.

INTEGRATION TEST - TEST THAT ENSURE THAT EACH INDEPENDENT PART OF YOUR APPLICATION WORKS CORRECTLY WITH THE SERVICES IT DEPENDS ON.

REGRESSION TEST? NOT MENTIONED IN THE DIAGRAM. THEY ARE CROSSCUTTING CATEGORY.

ANY PLAN THAT DEFERS TESTING TO THE END OF THE PROJECT IS **BROKEN**.


WHY?
YOU'LL SEE.
I WANT THIS.
WORKING ON IT...
IT'S GOING GOOD. 90% DONE.
WOW. SPLENDID!
6 WEEKS PASSES ~
WHEN THE FEEDBACK COMES...
YOU'RE WAITING TOO LONG FOR FEEDBACK
WE NEED TO MAKE SOME CORRECTIONS. WE NEED ANOTHER 3 WEEKS
WHAT!? WASN'T 90% DONE

Term explanations: unit test

- unit test:
    - Test the smallest unit of functionality. Usually a method or function in a class with a given state.
    - (Example: throw InvalidOperationException if trying to pop the stack and no elements available)
    - Should not call into other parts of the system (dependencies are stubbed / mocked)
    - Access network
    - Hit a database
    - Use the file system

Term explanations: integration test

- Test parts/components of the system as used in production
- They find wiring bugs of your application, examples:
  - reading from a database, different format on rows than expected
  - serializing and deserializing state to disk, maybe we forgot to close the file descriptor?
- Problems with integration tests
  - Harder to diagnose failures
  - Touches more code
  - Tests are harder to maintain

What does CONT. DEL *say about* AUTOMATED ACCEPTANCE TESTING

OBJECTIVE: PROVE THAT OUR APPLICATION DOES WHAT THE CUSTOMER MEANT IT TO, NOT THAT IT WORKS THE WAY IT'S PROGRAMMERS THINK IT SHOULD.

UNIT TESTS SHOW THAT A SINGLE PART OF THE APPLICATION DOES WHAT THE PROGRAMMER INTENDS IT TO.

FAIL FAST ⇩ FAST FEEDBACK

REFACTOR TESTS → ATOMIC TESTS

PERFORMANCE

- CREATE A CLEAN RUNNING INSTANCE OF THE SYSTEM UNDER TEST AT THE BEGINNING OF THE ACCEPTANCE TEST RUN, RUN ALL OF THE ACCEPTANCE TESTS AGAINST THAT INSTANCE AND SHUT IT DOWN AT THE END.
- USING COMPUTE GRIDS
- PARALLELL TESTING

EVALUATION

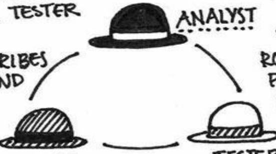
COST A LOT

RESPONSE: CAN BE COST EFFECTIVE IF WE DESIGN IT SMARTLY.

CREATING ACCEPTANCE TEST IS A COLLABORATING PROCESS.

Q: WHY?
A: TRANSPARENCY + TAKE AWAY ASSUMPTIONS + SHARE KNOWLEDGE

DEFINE ALL CRITERIA IN COLLABORATION WITH TESTER



ANALYST DESCRIBES REQUIREMENT AND BUSINESS CONTEXT + GO THROUGH ALL CRITERIA WITH DEVELOPER AND TESTER

DESIGN TESTS

ROLES: ONE PERSON CAN PLAY MORE THAN ONE ROLE

MAINTAINABLE ACCEPTANCE TEST SUITE

- LAYERS
- ATOMIC: NO DEPENDENCIES BETWEEN TESTS. THE ORDER IN WHICH THEY EXECUTE DOES NOT MATTER.
- USE TEST STUBS
- OWNED BY DEVELOPERS & TESTERS

ACCEPTANCE CRITERIA
GIVEN...
WHEN...
THEN...

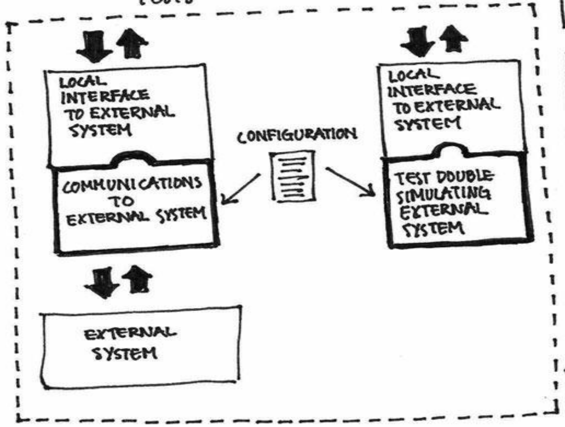
TEST IMPLEMENTATION
CODE USES DOMAIN LANGUAGE.
NO REF. TO UI ELEMENTS

APPLICATION DRIVER LAYER
UNDERSTADS HOW TO INTERACT WITH THE APPLICATION TO PERFORM ACTIONS AND RETURN RESULTS.

EXTERNAL INTEGRATION POINTS - (INTEGRATION TEST STRATEGY)

1. CREATE SMALL NUMBER OF TESTS TO COVER OBVIOUS SCENARIOS.
2. WE WILL MISS PROBLEMS → WE WILL ADDRESS BREAKAGES AS WE FIND THEM BY WRITING TEST TO CATCH EACH CASE.

⚠ NOT A PERFECT STRATEGY, BUT TO ATTEMTING TO GET PERFECT COVERAGE IN SUCH SCENARIOS IS USUALLY VERY DIFFICULT AND THE RETURNS OF EFFORT VERSUS REWARD DIMINISH VERY QUICKLY.



LOCAL INTERFACE TO EXTERNAL SYSTEM
COMMUNICATIONS TO EXTERNAL SYSTEM
CONFIGURATION
LOCAL INTERFACE TO EXTERNAL SYSTEM
TEST DOUBLE SIMULATING EXTERNAL SYSTEM
EXTERNAL SYSTEM

Term explanations: acceptance tests

- Specification written for the application
- Follows pattern of Context -> Action -> Outcome (Gherkin , Behat)
- Example from Behat's docs which suits Work-Work:

```
Feature: Serve coffee
  In order to earn money
  Customers should be able to
  buy coffee at all times

  Scenario: Buy last coffee
    Given there are 1 coffees left in the machine
    And I have deposited 1 dollar
    When I press the coffee button
    Then I should be served a coffee
```

```
Feature: Some terse yet descriptive text of
what is desired
  In order to realize a named business value
  As an explicit system actor
  I want to gain some beneficial outcome which
furthers the goal
  Scenario: Some determinable business
situation
    Given some precondition
      And some other precondition
     When some action by the actor
      And some other action
     Then some testable outcome is achieved
      And something else we can check happens
```

Test double:

Test doubles is a common technical term used for saying this object is an stand in for the real object. Comes from “stunt man” in movies.

Mocks: Implements same interface as real object the SUT needs (System under test) This is usually pre-programmed objects with expectations. (expects at least two calls on given method with the following arguments)

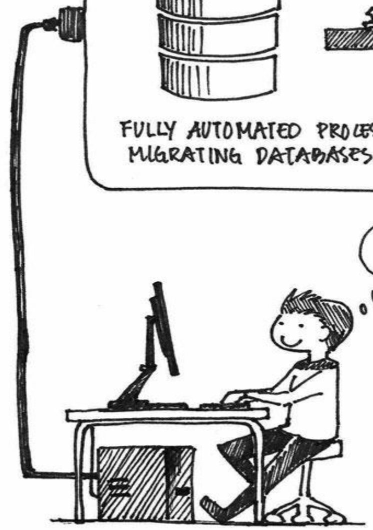
Stubs: Provide canned answers for calls made under the test, ie record if a email service ‘sent’ a message or not.

Fake: Have working implementations, but take shortcuts which deems it unusable for production. Ie, in memory database vs real database.

What does CONT. DEL. say about

MANAGING DATA

FULLY AUTOMATED PROCESS FOR CREATING AND MIGRATING DATABASES.



MANAGING TEST DATA
2 CONCERNS
- TEST PERFORMANCE
- TEST ISOLATION

NO REAL DATA BASE
BENEFIT : (LAYERS)
FOCUS ON BUSINESS BEHAVIOUR
+
DATA ACCESS CODE KEPT TOGETHER

IN MEMORY DATABASE
- CONFIGURABLE (ALLOW YOU TO SWITCH TO ANYTHING SUITABLE)
- BENEFIT: DECOUPLED CODE

MANAGING THE COUPLING BETWEEN TEST AND DATA

TEST ISOLATION
EACH TEST'S DATA IS ONLY VISIBLE FOR THAT TEST.

SETUP & TEAR DOWN

ADAPTIVE TEST
EACH TEST IS DESIGNED TO EVALUATE IT'S DATA ENVIRONMENT AND ADAPT IT'S BEHAVIOUR TO SUIT THE DATA IT SEES.

CONSEQUENCE
MORE COMPLEX AND LARGER TESTS.

TEST SEQUENCING
TEST ARE DESIGNED TO RUN KNOWN SEQUENCES, EACH DEPENDING FOR INPUTS ON THE OUTPUTS OF IT'S PREDECESSORS.

CONSEQUENCE

$$\bullet \rightarrow \bullet \rightarrow \bullet \rightarrow \bullet$$

FAIL CAUSING SUBSEQUENT TEST NOT TO BE RUN

IF YOU WANT TO TEST DIFFERENT VARIATIONS OF THIS TEST YOU ARE FORCED TO RUN THE PREDECESSORS

COMMIT STAGE → AUTOMATED ACCEPTANCE TEST → CAPACITY TESTING → MANUAL TEST

COMMIT STAGE
MUST RUN QUICKLY
- MINIMUM TEST DATA TO ASSERT THAT THE UNIT UNDER TEST EXHIBIT THE EXPECTED RESULT
- TEST NOT CLOSELY TIED TO IMPLEMENTATION. WILL OTHERWISE INHIBIT CHANGE.

+

AUTOMATED ACCEPTANCE TEST
3 TYPES OF DATA
- TEST SPECIFIC - TEST ISOLATION STRATEGY
- TEST REF. DATA - SUPPORTING CAST
- APPLICATION REF DATA - IRRELEVANT TO BEHAVIOUR UNDER TEST. NEEDS TO BE THERE FOR APPLICATION TO START UP.

+

CAPACITY TESTING
AMPLIFY TO GET THE LARGE SCALE

MANUAL TEST
SUBSET OF PRODUCTION DATA

Jenkins plugins

- Checkstyle (for processing PHP_CodeSniffer logfiles in Checkstyle format)
- Clover PHP (for processing PHPUnit's Clover XML logfile)
- Crap4J (for processing PHPUnit's Crap4J XML logfile)
- DRY (for processing phpcpd logfiles in PMD-CPD format)
- HTML Publisher (for publishing documentation generated by phpDox, for instance)
- JDepend (for processing PHP_Depend logfiles in JDepend format)
- Plot (for processing phploc CSV output)
- PMD (for processing PHPMD logfiles in PMD format)
- Violations (for processing various logfiles)
- Warnings (for processing PHP compiler warnings in the console log)
- xUnit (for processing PHPUnit's JUnit XML logfile)

Jenkins up and running with plugins

```
$ docker run -p 8080:8080 -p 50000:50000 jenkinsci/jenkins


$ wget http://localhost:8080/jnlpJars/jenkins-cli.jar

$ java -jar jenkins-cli.jar -s http://localhost:8080 install-plugin \

checkstyle cloverphp crap4j dry htmlpublisher jdepend plot pmd \

 violations warnings xunit
$ java -jar jenkins-cli.jar -s http://localhost:8080 safe-restart
```

Install jenkins php template job

```
$ curl -L \ https://raw.githubusercontent.com/sebastianbergmann/php-jenkins-template/master/config.xml \ |
java -jar jenkins-cli.jar -s http://localhost:8080 create-job php-template
```

We now have the basic php-template up and running, but we still need additional plugins:

Enhancements to look into

- Use job-dsl to write jenkins jobs and let em reside together with repositories
- Use job-dsl to write pipeline for continuous delivery