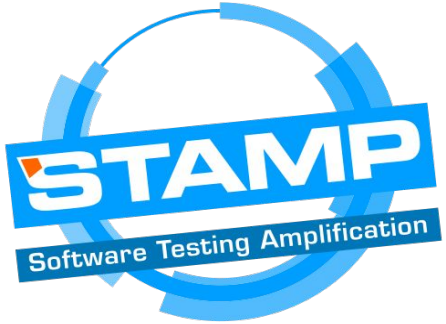


WP5 Industrial Evaluation

H2020 LEIT RIA - ICT-10-2016 – Software Technology

2016/12/01 – 2019/11/30

Outline

- WP5 Introduction
  - Deliverables
  - KPIs
  - Validation Questions
- Industrial Validation
  - Activeeon
  - Atos
  - OW2
  - Tellu
  - XWiki

Deliverables

T5.3-T5.7: Use Case validation

- D5.7 Use Case Validation Report v3 - M36, Nov 19

	Descartes	DSpot	CAMP	Botsing/RAMP
Activeeon	√-√	X-√	X-√	√-√
Atos	√-√	√-√	√-√	√-√
Tellu	√-√	√-√	√-√	√-√
XWiki	√-√	√-√	√-√	√-√
OW2	√-√	√-√	X-√	X-√

Use Case Experimentations during Period 2 (X -√) and Period 3 (√)

KPIs

- **K01: More execution paths**. Measures test coverage
- **K02: Less flaky tests**. Measures ability to recognize and handle flaky tests so that they don't impact developer's confidence in tests
- **K03: Better test quality**. Measures test quality through mutation score
- **K04: More unique invocation trace**s. Measures new paths taken thanks to configuration testing
  - New ways to measure this KPI for non micro-service architectures.
  - Using Flamegraphs
    - Differences of flamegraph calls between configurations
  - Counting combinations of different configurations and tests.
    - (#Configs) * (#Docker tests)
- **K05: System-specific bugs**. Measures ability to discover system specific bugs.
- **K06: More Configuration/Faster Tests**. Measures quantity of configurations tested and how much time is won vs manual testing of configurations.

KPIs

- **K07: Shorter Logs**. Dropped.
- **K08: More crash replicating test cases**. Measure ability to automatically generate tests reproducing a crash.
- **K09: More production level test cases**. Generate new tests based on a static and dynamic analysis of code and existing tests.
  - New metric.
  - (#Tests generated by RAMP) / (#manually-written unit tests) * 100
  - Target: 10% increase

Validation Questions (1/4)

VQ1 - ***Can the STAMP tools assist software developers to cover areas of code that are not tested?***

Related to STAMP **O1: Provide an approach to automatically amplify unit test cases when a change is introduced in a program**

Challenge**: to reduce the amount of untested code**

- **K01: More execution paths:** code coverage
- K04: More unique invocation traces: how configuration test amplification can help to test more paths through the SUT
- K08: More crash replicating test cases: create tests which reproduce runtime crashes not detected beforehand by the existing test suites.
- K09 - More production level test cases: STAMP tools may be able to create tests after observing the runtime behaviors

Validation Questions (2/4)

VQ2 - ***Can STAMP tools increase the level of confidence about test cases?***

Related to STAMP **O1: Provide an approach to automatically amplify unit test cases when a change is introduced in a program**

Confidence relates to**:**

- **False positives:** tests fail without actual errors in the code
- **Test verdict**: to what extent can we be sure that the code executed by tests is indeed correct?

- K02 - Less flaky tests: test sometimes fails and sometimes passes, without changes in the SUT configuration, internal or external (environment) state or test
- K03 - Better test quality: mutation score. This score tells us to which extent changes to the code is detected by tests.

Validation Questions (4/4)

VQ4 - *Can STAMP tools speed up the test development process?*

Related to STAMP **O3: Provide an approach to automatically amplify, optimize and analyze production logs in order to retrieve test cases that verify code changes against real world conditions.**

Challenge**: reduce the time spend in developing and executing test cases (unit, configuration-dependent and crash replicating)**

- K06 - More Configuration/Faster Tests: execute more tests than before per amount of time
- K08 - More crash replicating test cases: Generation of crash replicating test cases vs manual creation of such test cases
- K09 - More production level test cases: the generation of production level test cases (driven by the observation of the SUT behavior) will be evaluated against the manual creation of such tests.

Validation Questions (3/4)

VQ3 - *Can STAMP tools increase developers confidence in running the SUT under various environments?*

Related to STAMP **O2: Provide an approach to automatically generate, deploy and test large numbers of system configurations**

Dependency on the environment. To what degree will the tested software run correctly in various real deployments?

- K04 - More unique invocation traces: execute more paths through the SUT by varying configurations
- K05 - System-specific bugs: find bugs which only occur on specific configurations
- K06 - More Configuration/Faster Tests: run the SUT on more configurations
- K08 - More crash replicating test cases: create tests which reproduce crashes caused by configuration testing amplification
- K09 - More production level test cases: if the behavior of the SUT can drive the generation of new test cases, these can be used to verify the correct behavior of the SUT under different target environments.

Industrial Validation

- Industrial Validation
  - Activeeon
  - Atos
  - OW2
  - Tellu
  - XWiki

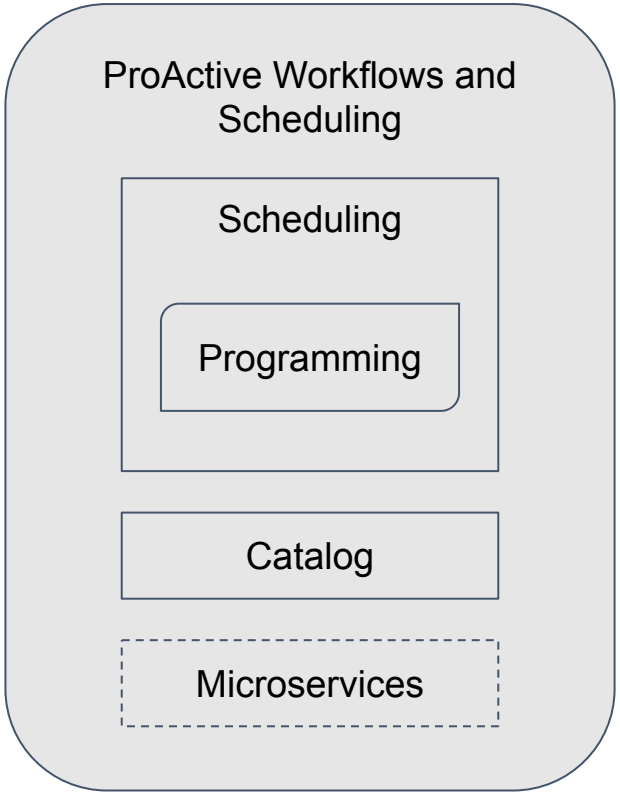
UC ActiveEon (1/7) – UC introduction – expectations from STAMP

ProActive Workflows and Scheduling (**PWS**)

- One core project: **Scheduling** (1770 tests)
- 40 projects: microservices or libraries
  - **Programming** (library, 808 tests)
  - **Catalog** (microservice, 225 tests)

Expectations for this reporting period:

- Use Dspot and RAMP to increase the test suite quality
- Use CAMP to discover bugs
- Use Botsing with Programming to clarify unresolved reported issues



UC ActiveEon (2/7) – KPI Summary

KPI	Measure			Difference with objective
	Baseline	**Treatment**	**Difference**	
K01-Execution Paths (Catalog)	42% (Code Coverage) **58**% uncovered	52% (Code Coverage) **48**% uncovered	**+21%**	**-13.2%** (-40% reduction)
K02-Flaky Tests (PWS)	**46 detected**	**23 identified and handled**	**+50%**	**+30%** (20%)
K03-Better Test Quality (Catalog)	**35%** (mutation score)	**43%** (mutation score)	**23%**	**+3%** (20%)
K04-More unique traces (PWS)	**38%**	**69%**	**82%**	**+42%**(40%)
K05-System specific bugs (PWS)	**0**	**5**	**+5**	**N/A** (30%)
K06-More config / Faster (PWS)	**1**	**8**	**+7 (+233%)**	**+183%** (50%)
K08-More crash tests (Catalog)	**16 crashes**	**1 replicated**	**6.3%**	**-64.7%** (70%)
K09-More prod tests (Catalog)	**184 unit tests**	**149 unit tests generated**	**+81%**	**+71%** (10%)

UC ActiveEon (3/7) – Descartes (KPIs, VQs)

K03-Better Test Quality (Catalog)	**35%** (mutation score)	**43%** (mutation score)	**23%**	**+3%** (20%)

Type of test	Score	Score after focusing on adding tests
line coverage	42%	52%
mutation score	35%	43%
pseudo-tested	36	48
partially-tested	5	7

VQ2: By increasing the mutation score, Descartes enabled to increased the confidence in the test suite.

VQ4: Descartes helps developers to quickly identify how to increase the mutation score in order to reach the level of confidence required.

UC ActiveEon (4/7) – Dspot (KPIs, VQs)

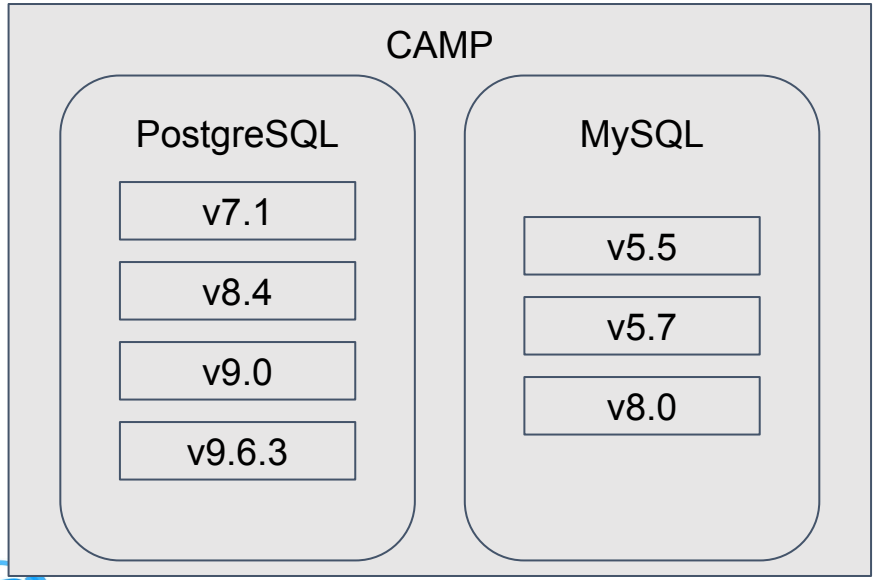
K01-Execution Paths (Catalog)	42% (Code Coverage) **58**% uncovered	52% (Code Coverage) **48**% uncovered	**+21%** **+17%**	**-23%** (40%)
K03-Better Test Quality (Catalog)	**35%** (mutation score)	**43%** (mutation score)	**23%**	**+3%** (20%)

	Before Dspot	After Dspot
Lines Covered	871/2020	891/2020 **(+20)**
Line Coverage	43%	44% **(+1%)**
Mutation Score	192/546	220/546 **(+5%)**
Mutation Coverage	35%	40% **(+5%)**

VQ2: Dspot generates tests that can be added to the tests suite. These tests increase the confidence in the test suite by increasing the code coverage and the mutation score.

UC ActiveEon (5/7) – CAMP (KPIs, VQs)

K04-More unique traces (PWS)	**38%**	**69%**	**82%**	**+42%**(40%)
K05-System specific bugs (PWS)	**0**	**5**	**+5**	**N/A** (40%)
K06-More config / Faster (PWS)	**1**	**8**	**+7 (+233%)**	**+183%** (50%)



VQ1: The CAMP experiment showed that running tests over different configurations leads to discover more unique execution traces that were not previously reached by the baseline version.

VQ3: CAMP increased our confidence in PWS resilience under several configurations.

VQ4: CAMP have replaced our manual configuration tests by automatic configuration test.

UC ActiveEon (6/7) – Botsing/RAMP (KPIs, VQs)

K08-More crash tests (Catalog)	**16 crashes**	**1 replicated**	**6.3%**	**-64.7%** (70%)
K09-More prod tests (Catalog)	**184 unit tests**	**149 unit tests generated**	**+81%**	**+71%** (10%)

	Scheduling	Programming/Catalog
Number of execution	23	45
Execution failed/No test generated	15 (65%)	31 (69%)
Empty test generated	5 (22%)	14 (31%)
Test generated	3 (13%)	0 (0%)

VQ1: Botsing generated tests reproducing issues that were not detected during the initial testing phase. By running the generated tests, some areas of the code, that were not previously tested, will be executed.

UC ActiveEon (7/7) – Industrial benefits of STAMP adoption

STAMP had a huge impact on Activeeon's testing practices. Our engineers learnt new solutions to improve our test suites' quality. We learnt about the concept of mutation testing and the tools that can be used on our projects.

Among all the improvements, those with the biggest impact are the following:

- Descartes is integrated and used to validate the test suite quality in Activeeon projects.
- CAMP helped us to test new configurations automatically and to discover bugs with specific databases.
- During STAMP project, we increased our code quality by fixing bugs and our tests quality by increasing our code coverage and improving our projects mutation score.

UC Atos (1/7) – UC introduction – expectations from STAMP

Atos industrial validation has targeted the SUTs:

- **SUPERSEDE IF:** inter-service middleware, Java, 200 test suites, 25493 NSLOC
- **CityGo CityDash:** city dashboard for travel planning, Python/Django, no test suites, 15473 NSLOC

Expectations for this reporting period:

- ***CityGO CityDash***
  - **Deliver an optimal CityGO platform** for the current runtime infrastructure conditions.
  - **Deliver CityGO CityDash onto different compatible runtime platforms** provisioned by city administrators
- ***SUPERSEDE IF***
  - **Ensure the runtime functional consistency of the SUPERSEDE backend platform** through a robust test-driven QA process

UC Atos (2/7) – KPI Summary

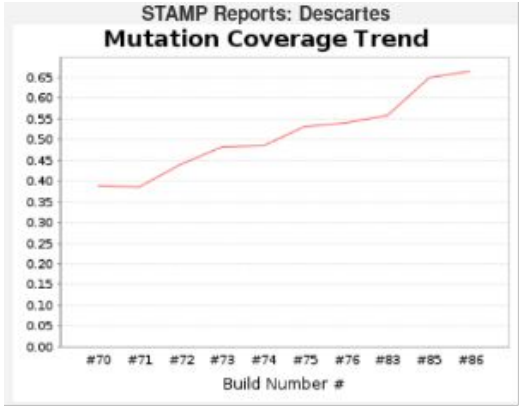
KPI	Measure			Difference with objective
	Baseline	**Treatment**	**Difference**	
K01-Execution Paths (IF)	47.7% (Code Coverage) **52.3%** uncovered	78.7% (Code Coverage) **21.3%** uncovered	+65.68% **+59.27%**	**+19.27%** (40%)
K02-Flaky Tests (IF)	**No Flaky tests**	N/A	N/A	N/A (20%)
K03-Better Test Quality (IF)	**39%** (mutation score)	**66%** (mutation score)	**69,23%**	**+49,23%** (20%)
K04-More unique traces (CityGo)	**629** (suboptimal config)	**683** (optimal config)	**8.59%**	**-31.41** (40%)
K05-System specific bugs (CityGo)	**0** (one single default config)	**5** (of 32 configs)	**N/A**	**N/A** (30%)
K06-More config / Faster (CityGo)	**1** (#config)	**42**(#config)	**+4100%**	**+4040** (50%)
K08-More crash tests (IF)	**0**	**3 (3/6)**	**50%**	**- 20%** (70%)
K09-More prod tests (IF)	**196**	**2180**	**1012.24%**	**+1002.24** (10%)

UC Atos (3/7) – Descartes (KPIs, VQs)

KPIs:

- **K01**: Descartes seems to **have no influence** on the overall code coverage
- **K03**: Descartes **has strong positive impact on both mutation score and test quality**

Iteration	Mutation score
Baseline 1 (D5.6)	39%
D5.6 Descartes treatment	43%
D5.7 Descartes treatment	53%



Iteration	Mutation Score	Issues	
		Pseudo-tested	Partially tested
Baseline - 2019/07/01	38%	20	21
Code refactoring: 2019/09/25	48%	7	4
Code refactoring: 2019/09/27	54%	2	2
Variation	**42%**	**-90%**	**-90%**

VQs:

- **VQ1**: Descartes shows **low positive impact** in reaching not tested code areas.
- **VQ2**: Mutation score increases up to **69,23%** with the combined STAMP treatment for addressing the reported **Descartes** issues, which decreases **90%**

UC Atos (4/7) – Dspot (KPIs, VQs)

KPIs:

- **K01**: **DSpot increases the code coverage (from 47% to 54.9%)** thanks to the new tests cases it created by amplifying most of the existing IF test suites. DSpot coverage improvement (**7.4%**) required 50 amplified test suites and 1317 new test cases.
- **K03**: Significant DSpot contribution to the **mutation score improvement (5.66%)** thanks to its amplified tests that supported the detection of a larger set of mutants.

Treatment	Code Coverage (Clover)
Baseline 1 (D5.6)	47.5%
D5.6 Dspot Treatment	52%
D5.7 DSpot Treatment	54,9%

Iteration	Mutation score
Baseline 1 (D5.6)	39%
D5.7 Descartes treatment	53%
D5.7 DSpot treatment	56%

VQs:

- **VQ1**: DSpot **is significantly contributing for a positive answer** to this VQ1. DSpot is still producing a significant high number of (quite similar) amplified test cases, though
- **VQ2**: DSpot have a relative impact on increasing the mutation score. Significantly better was the **DSpot management, in amplified tests, of the exceptional code behavior**.

UC Atos (5/7) – CAMP (KPIs, VQs)

KPIs:

- **K04**: CityDash under different configurations s**hows a significant (5% - 10%) increment** in the total number of unique CityDash test executions, **despite its monolithic nature**
- **K05**: Tests executed by CAMP spotted **5 failing configurations**
- **K06**: CAMP significantly reduces **the overall time required to test CityDash under multiple given configurations**

VQs:

- **VQ3**: CAMP **increases our confidence** in CityDash under different workloads (e.g. number of concurrent users) requiring **optimal configurations**, and classified them between those that are reliable and those that are not.
- **VQ4**: *the execution of functional and stress tests can be **fully automated** by CAMP in out CI/CD*

K04-More unique traces	**Measure**		
	Baseline	**Treatment (CAMP)**	**Difference (%)**
#unique traces (performance)	**92** (suboptimal config)	**102** (optimal config)	**+10.86%**
#unique traces (functional)	**537** (suboptimal config)	**581** (optimal config)	**+5.57%**

K05-System specific bugs	**Measure**	
	Treatment (CAMP)-Performance	**Treatment (CAMP)-Functional**
#failing configurations	**4 (10)**	**1 (32)**

K06-More config / Faster	**Measure**		
	Baseline	**Treatment (CAMP)**	**Difference (%)**
#Amplified configurations	**1**	**10/32**	**+900%/3100%**
CAMP execute time	**1722/1588 s**	**1016/476 s**	**-41.00%/-70.03%**

UC Atos (6/7) – Botsing/RAMP (KPIs, VQs)

KPIs:

- **K01**: RAMP offers the most significant code coverage improvement
- **K03**: Significantly larger is the RAMP contribution, supported by the newly generated production level tests: a relative mutation score improvement of 22.64% (from 53% to 65%)
- **K08**: Botsing succeeded on generating crash replicating tests in 50% of the tried cases
- **K09**: RAMP could generate lots of (somehow repetitive and simple) test cases for the IF helper and model packages in over 2 hours. RAMP could not generate production level tests for IF business classes, those that offer communication proxy capabilities, since RAMP failed in correctly mocking the proxy objects

VQs:

- **VQ1**: **High contribution of RAMP** for a positive answer to this question. Although RAMP generated tests look rather simple, we acknowledge the high potential of its behavioral model seeding approach to increase the level of richness of the generated tests
- **VQ2**: Favorable RAMP impact for a positive answer to this question.
- **VQ4**: **Botsing** offers some assistance to developers **to speed up the generation of test cases for some exceptional situations**. **RAMP** is **fast and efficient in the generation of new test cases** for some IF classes, although generated test cases are rather simple executions of a few class API methods

Treatment	Code Coverage (Clover)
D5.7 baseline	53.4%
D5.7 RAMP treatment	78.3%

Treatment	Mutation score
D5.7 Baseline	56%
D5.7 RAMP treatment	65%

KPI8 More crash tests	Treatment (Botsing)
Number of crash replicating test cases	**3 (3/6)**

KPI9: More production tests	Baseline	Treatment (RAMP)
Number of test cases	**196**	**2180**

UC Atos (7/7) – Industrial benefits of STAMP adoption

STAMP technologies and tools show a **great potential impact on the QA of the Atos Research & Innovation (ARI)** software development lifecycle methodology. The benefits of STAMP adoption in ARI can be classified into two groups:

- **Know-how**
  - **TDD related technologies** have been promoted among the Atos ARI team
  - Atos is internally promoting **learnt strategies to improve our test bases**
  - Adopting **QA indicators such as mutation score and code coverage** to strengthen our test base in order to reduce the occurrence of regression bugs
- **Improvements in software development methodology**
  - Two main STAMP technologies that are promoted within ARI internal methodology because of its straightforward application:
    - **Descartes' test quality analysis** to be incorporated into our pipelines
    - **CAMP test execution support against multiple environments**. It has been evaluated by Seville testing factory.

OW2 UC (1/7) - expectations from STAMP

- OW2 is an open-source software community
  - OW2 provides infrastructure (hosting and quality assessment resources) to open-source projects.
- STAMP tools are expected to:
  - Enhance OW2's capability to evaluate and classify projects, by adding value to our MRL (Market Readiness Level) methodology.
  - Offer OW2 project leaders ready-to-use test amplification tools, that can be activated in the project build and/or continuous integration process.

OW2 UC (2/7) – KPI Summary

KPI	Measure			Difference with objective
	Baseline	**Treatment**	**Difference**	
K01-Execution Paths (Joram)	14.0% (Code Coverage) **86%** uncovered	18.4% (Code Coverage) **81.6**% uncovered	+31.3%	-30% (40%)
K02-Flaky Tests (Joram)	**Not quantifiable** (Multiple random exceptions in existing test suite)	**Flaky tests not fixed** (reproduced, but skilled human dev required from team)	**0%**	**- 20%** (20%) Great help for dev team (hints in generated tests)
K03-Better Test Quality (Joram)	**20%** (mutation score)	**27.2%** (mutation score)	**+36.2%**	**+16.2%** (20%)
K04-More unique traces (Lutece)	**Initial configuration: config0**	**88%** difference with config0 (in best config tested)	**+88%**	**+48%** (40%)
K05-System specific bugs (Lutece)	**0**	**0**	**0%**	**- 30%** (30%)
K06-More config / Faster (Lutece)	**1** (#config)	**7** (#config) Variations: 2 on DB, 3 on app server, 2 on java platform	**+600%**	**+550%** (50%)
K08-More crash tests (Joram)	**0**	**7 exceptions out of 12 successful** (result: 64 tests in 4 cases)	**+58%**	**- 12%** (70%)
K09-More prod tests (Joram)	**93 test cases**	**123 test cases**	**+43%**	**+33%** (10%)

OW2 UC (3/7) – Descartes (KPIs, VQs)

- Stable, industry-grade: recommend to community.
  - VQ 1, 2 (all dev process + confidence) / K01, 03 (execution paths, test quality)
    - Joram / Lutece
  - Suitable for project managers: build (break upon mutation score breach), development (detect poor testing patterns like pseudo-tests), CI (when fast enough: depends on projects and the way tests are coded).
  - Configurable outputs, for human readers, machine processing, or automation (we could generate Gitlab issues, which proves Descartes is also extensible).

OW2 UC (4/7) – Dspot (KPIs, VQs)

- Some increase in mutation and line coverage
  - VQ 1, 4 (dev speedup + coverage) / K01, 03, 09 (execution paths, test quality, prod level tests)
    - Joram / Lutece
  - Result difficult to interpret for a human developer (applied to a real Joram bug: fixed, but team rewrote tests by hand).
- Trade-off between computation time and generated tests too low for massive deployment at OW2
  - DSpot punctually efficient to fix something (particularly when mixed with Descartes, to auto-suggest DSpot commands).
  - More science needed to improve legibility of generated tests

OW2 UC (5/7) – CAMP (KPIs, VQs)

- For projects with multiple configuration schemes
  - VQ 3 / K04, [05], 06 (multi-environment test + SUT confidence)
    - Joram / Lutece
  - Many external components: servers, databases, platforms…
    - *Lutece: from 1 to 7 variations (2 DB / app servers/ 2 platforms)*
    - *Perf tests + unique traces (stability assessed: no shift, many paths)*
- CAMP dedicated to high configuration complexity
  - For complex multi-configuration projects (like Lutece), when more basic solutions don’t fit (like plain Dockerfiles).
  - More suitable for validation / deployment tests on releases than everyday use by dev teams.

OW2 UC (6/7) – Botsing/RAMP (KPIs, VQs)

- Provides accurate crash-reproduction code.
  - VQ 1, 3, 4 (all dev process) / K01, 02, 03, 08, 09 (all except config tests)
    - Joram / Lutece
  - RAMP, added to Botsing, provides important increases of mutation and line coverage, at little cost.
    - *58% success (7 exceptions out of 12, 64 tests in 4 test cases)*
- Recommended to skilled people for specific hard debugging (like flaky behaviour).
  - Brings new ideas and innovative fixes, but needs skilled post-processing (JUnit porting, cleanup, understanding...)
  - Difficult to automate (long processing / medium success rate)

OW2 (7/7): Industrial benefits of STAMP adoption

- Promote new approaches and better understanding of testing in the community
- OW2 projects adopting STAMP in production...
  - … obtain better scores in MRL (Market Readiness Level)
  - Done for XWiki + ActiveEon, in progress for Joram and Lutece
- CI tools to integrate STAMP with our issue tracking system (Gitlab).
  - Tools + doc to exploit Descartes in production
    - Descartes Gitlab issue generator released on Maven Central
    - Config + doc provided in OW2 new maven project template

UC Tellu (1/7) – UC introduction – expectations from STAMP

- TelluCloud - IoT platform for collection and processing of data
  - Edges
  - Data processing and storage
  - Rule engine
  - REST APIs, web applications
- Unit test experiments on three projects
  - TelluLib (8.245 NCLOC)
  - Core (27.462 NCLOC)
  - FilterStore (1.777 NCLOC)
- Deployment with Kubernetes

Expectations from STAMP

- Improve and automate testing
- Move towards DevOps
- Ensure correctness and robustness in a cost-effective manner
- Amplification, tools and techniques
  - Generate unit tests
  - Increase test coverage & quality
  - Tests for micro-service and system level
  - Amplify Kubernetes configuration files

UC Tellu (2/7) – KPI Summary

KPI	Measure			Difference with objective
	Baseline	**Treatment**	**Difference**	
K01-Execution Paths	29,1% coverage **27932 uncovered**	42% coverage **21725 uncovered**	44,5% cov. increase **22,2% decrease in uncovered code**	**-18%** (40%)
K02-Flaky Tests	3 flaky of 6 tests	Fixed 3	100% fixed (50% of total)	**+80%** (20%)
K03-Better Test Quality	**24,7%** (mutation score)	**35,9%** (mutation score)	**45,5%**	**+25,5%** (20%)
K04-More unique traces	**1 message path**	**5 message paths**	**500%**	**+460%** (40%)
K05-System specific bugs	**10**	**3 new**	**20%**	**-10%** (30%)
K06-More config / Faster	**1 configuration** **4 hours to deploy**	**48 configurations** **10 minutes to deploy**	**4800%** **2300%**	**+2250%** (50%)
K08-More crash tests	12 crashes	3 replicated	25% replicated **No test cases**	**-45%** (70%)
K09-More prod tests	**162 tests**	**1407 generated tests**	**869%**	**+859%** (10%)

UC Tellu (3/7) – Descartes (KPIs, VQs)

Project	K01 (coverage) improvement	K03 (mutation) improvement	Total issues found	Issues fixed
TelluLib	1,49%	4,2%	34 (4 partial, 30 pseudo)	15
Core	1,51%	14%	125 (15 partial, 110 pseudo)	19
FilterStore	0%	0%	4 (2 partial, 2 pseudo)	0

- Issue report shows exactly which methods are not properly covered.
- Provides knowledge, and led to fix of critical issues.
- Resulting in some improvements in mutation score.
- Tellu decided to include Descartes in the DevOps toolchain to check new tests as they are written
- **VQ1**: Somewhat - Manual fixing of issues give a small increase in coverage.
- **VQ2**: **YES**. Main benefit is finding test issues.

UC Tellu (4/7) – DSpot (KPIs, VQs)

Project	Basel. tests	Treatment tests	K01 (coverage) improvement	Baseline issues	Treatment issues	K03 (mutation) improvement
TelluLib	77	306	**2,7%**	30 (3 partial, 27 pseudo)	30 (6 partial, 24 pseudo)	**6,3%**
Core	83	95	**1,9%**	125 (20 partial, 105 pseudo)	132 (35 partial, 97 pseudo)	**14%**

Compared to Descartes + manual fixes (DSpot in green)					
Mutation improv.		**Issues fixed**		**Coverage improv.**	
4,2%	6,3%	15	3	1,5%	2,7%
14%	14%	19	1 partial was fixed, 12 pseudo to partial	1,5%	1,9%

- Worked well, amplified most test cases tried.
- Needs knowledge and time to use well.
- Better than manual fixing of Descartes issues!
- **VQ1**: **YES** - automatic generation of tests reaching previously untested code.
- **VQ2**: **YES** - generated tests improve mutation score.
- **VQ4**: **YES** - additional code coverage and mutation score with minimal human effort.

UC Tellu (5/7) – CAMP (KPIs, VQs)

- Docker (single-node) config. amplification (D5.6):
  - Speeds up test deployments.
  - Little variability in the Docker level for TelluCloud.
- Kubernetes (cloud) config. amplification (D5.7):
  - Variations in 3 performance-related params.
  - Generated 48 different configurations.
  - Found large performance increase (495%).

K04 - More unique invocation traces
Message paths through the system

- Baseline runtime 94 seconds.
- Best configuration gave runtime of 19 seconds.
- No relation between config. and alarm sequence.
- Sequence deviation also quite random.
- Up to 5 instances of node - 5 paths.

K05 - System-specific bugs Docker-compose amplification	
Baseline bugs	10
New bugs discovered	2
Improvement	20%

K06 - More configs./Faster tests	
Baseline time to deploy	4 hours
Time to deploy using CAMP	10 min
Baseline configurations	1
Kubernetes amplification	48

- **VQ3**: **YES** - CAMP helps us explore the configuration space.
- **VQ4**: **YES** - Speeds up testing of different configurations.

UC Tellu (6/7) – Botsing/RAMP (KPIs, VQs)

Botsing
Collected 20 unique stack traces from operational TelluCloud instances. All caused by external factors and handled correctly - no crashes.
Validation with introduced errors: • 12 crashes with stack traces • Botsing replicated 3 (25%)

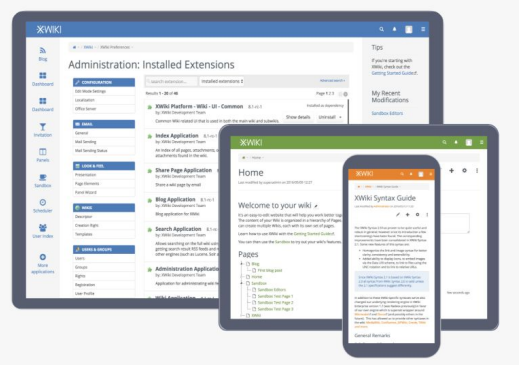
RAMP	**Baseline**	**Treatment**	**Result**
K01 - Code coverage	32,5%	41,7%	**+28,2%**
K03 - Mutat. score	23,6%	34,4%	**+45,7%**
K09 - Number of tests	162	1407	**+869%**

- **Botsing** worked well in lab validation, but we have not found use for it for TelluCloud.
- **VQ1**: **YES** - RAMP generates tests reaching previously untested code.
- **VQ2**: **YES** - RAMP generated tests improve mutation score.
- **VQ4**: **YES** - RAMP generates additional code coverage and mutation score with minimal human effort.

UC Tellu (7/7) – Industrial benefits of STAMP adoption

- Tellu got new **insights** into testing and test quality.
- Learned **testing practices**, and how important testing is to automation of DevOps.
- The use case projects saw very significant increases in code coverage (**44,5%** increase) and mutation score (**45,5%** increase) thanks to STAMP work and tools.
- Developed a **testing framework** for micro-services and cloud deployments, and tests on these levels.
- STAMP has shown that **amplification is feasible**.
- Helped ensure the quality as TelluCloud has been deployed commercially and further developed, now having been **deployed** in three different cloud infrastructures.

UC XWiki (1/7) – UC introduction – expectations from STAMP



Use case:

- Enterprise wiki and web development platform
- Full code base: 3 Git repos (xwiki-commons, xwiki-rendering, xwiki-platform)
- Size: **500K** NCLOC, **7K** java files, **1K** Maven modules, **9K+** tests before STAMP (**11K+** after)

Expectations for this reporting period:

- Continue increasing the global test coverage on the live and full XWiki codebase
- Finish implementing Configuration testing on all supported XWiki configurations and continue migrating existing functional tests to the new CAMP/TestContainers framework.
- Improve the CAMP/TestContainers framework to stabilize it by removing false positives and by adding new features and improvements
- Setup RAMP to generate KPIs for K09

UC XWiki (2/7) – KPI Summary

KPI	Measure			Difference with objective
	Baseline	**Treatment**	**Difference**	
K01-Execution Paths	**65.29%** (Code Coverage)	**71.33%** (Code Coverage)	**+9.25%** (on live codebase of ~1M LOC!)	**-7.85%** (40% reduction)
K02-Flaky Tests	**0.010%** (% flaky fixed vs total # tests)	**0.290%**	**+2890.5%**	**+2870.5%** (20%)
K03-Better Test Quality	**61%** (mutation score)	**68%** (mutation score)	**+11.47%**	**-8.53%** (20%)
K04-More unique traces	**194** (unique traces)	**1315** (unique traces)	**+577.83%**	**+537.83%** (40%)
K05-System specific bugs	**56** (config-related bugs)	**72** (config-related bugs)	**+28.57%**	**-1.43%** (30%)
K06-More config / Faster	**1** (#config)	**32** (#config)	**+3100%**	**+3050%** (50%)
K08-More crash tests	**9** (#issues reproduced by Botsing)	**13** (#issues reproduced by Botsing)	**+44.44%**	**- 25.56%** (70%)
K09-More prod tests	**40** (#tests)	**213** (#tests)	**+432%**	**+422%** (10%)

UC XWiki (3/7) – Descartes (KPIs, VQs)

KPIs

- **K01**: Increased by Descartes (+2.4% average) but low
- **K03**: Increased by Descartes (+6.7% average) but low. However XWiki already had high coverage (60%+)

VQs:

- **VQ1**: When increasing mutation, code coverage is also increased
- **VQ2**: Descartes successfully increases the mutation score and thus improves the quality of existing tests.
- **VQ4**: Helps write better tests and thus improve test dev process

- Descartes integrated into XWiki's build and CI
  - Automatically fails the build if the mutation score is reduced compared to the current level
  - Ensures that test quality can only go up
- Bugs found thanks to Descartes (e.g. https://bit.ly/380UR2I)

Iterations	**Mutation score** increase on modules
D5.6 Descartes treatment	+116%
D5.7 Descartes treatment	+125%
Total Descartes treatment	+241%
Average per module	**+6.69%**

Iterations	**Coverage increase** on modules
D5.6 Descartes treatment	+40%
D5.7 Descartes treatment	+53%
Total Descartes treatment	+93%
Average per module	**+2.4%**

European Commission

UC XWiki (4/7) – Dspot (KPIs, VQs)

KPIs

- **K01**: Very small increase but when tests are generated the coverage is increased.
- **K03**: Almost no mutation score increase even when tests are generated.

VQs:

- **VQ1**: Coverage is increased but slowly
- **VQ2**: No significant increased test quality
- **VQ4**: Not mature enough to provide significant test dev help

- DSpot integrated into XWiki's build
  - Generated tests execute next to manual tests
- DSpot doesn't often generate tests for the XWiki code base, most likely because the XWiki coverage is already quite high. However it works and it's automatic, so still providing value.

Iterations	**Mutation score** increase on modules
D5.6 DSpot treatment	+0%
D5.7 DSpot treatment	+3%
Total DSpot treatment	+3%
Average per module	+0.11%

Iterations	**Coverage increase** on modules
D5.6 DSpot treatment	+1.139%
D5.7 DSpot treatment	+7.69%
Total DSpot treatment	+8.82%
Average per module	+0.32%

European Commission

UC XWiki (5/7) – CAMP (KPIs, VQs)

KPIs

- **K04**: Good results
- **K05**: Slow increase but normal since bugs are now found before issues are created by users!
- **K06**: Very good results. Speed increase of 96% compared to manual for a single config test (thus huge when including all configs).

Iteration	K04 - More invocation traces	K05 - System specific bugs	K06 - More configs/Faster
D5.6 CAMP treatment	N/A (0%)	+7.14%	+2600% 26 configs with 1 test
D5.7 CAMP treatment	+577.83%	+28.57%	+3200% 32 configs with 86 tests (129 now!)

VQs:

- **VQ3**: Huge boost thanks to the integration of config testing in the build and in the CI.
- **VQ4**: Also huge boost thanks to reduced time in testing configs on developer's laptops.

- Integrated into XWiki's development process and build/CI.
- Configuration testing is what brought the most value to the XWiki project in testing.
- 15 bugs found (https://bit.ly/2UyfoI1) thanks to CAMP

UC XWiki (6/7) – Botsing/RAMP (KPIs, VQs)

KPIs

- **K08**: Lowered percentage due to more reported issues with stack traces than Botsing was able to reproduce.
- **K09**: Lots of tests generated
- **K01:** Increased coverage (+6% average)
- **K03:** Increased mutation score (+11% average)

VQs:

- **VQ4**: Not enough maturity to really help test dev speed

- Both Botsing and RAMP have proven able to generate tests for a complex project such as XWiki. They're still missing some maturity to be put in production and providing a positive cost/benefit ratio.

Iterations	K08 - More crash replicating tests
D5.6 Botsing treatment	60%
D5.7 Botsing treatment	33.33%

Iterations	K09 - More production level tests
D5.6 RAMP treatment	N/A
D5.7 RAMP treatment	+430%

UC XWiki (7/7) – Industrial benefits of STAMP adoption

- Coverage increased from 65.29% to 71.4% over the course of STAMP, which is a substantial achievement on such a large and live code base.
- Descartes proved to be useful for monitoring test quality and the XWiki's build and CI now fail when either coverage or mutation score are reduced. They can only go up from now on.
- Before STAMP the XWiki project was only testing a single configuration automatically. Thanks to CAMP/TestContainers, it's now tested on 30+ configurations automatically in the CI, finding problems before the software is released in production.
- In addition, CAMP/TestContainers allows much faster debugging of issues since configurations can be reproduced and tests executed on them on developer machines. They've become as simple as unit tests.
- Thanks to STAMP, the XWiki project has also added a Docker distribution, increasing its reach and number of Active Installs. More globally STAMP has raised the quality of XWiki and the awareness in testing for the whole XWiki community.

Questions & Answers