PERFORMANCE TESTING FROM THE FIRST STEPS TO CONTINUOUS INTEGRATION

itera

MAKE A DIFFERENCE

TaaS: Performance testing

Description:

Some huge Icelandic bank, had a request for checking their application performance during their migration from archaic technology to .NET services. Application is to be integrated with a lot of third party system and consist of three layers. The test team was responsible not only for searching bugs and creating performance testing framework, but also for teaching client for performance testing skills

Challenges:

- No documented knowledge about the system
- No special environment and no any test data
- No knowledge and expectation about performance testing
- No accesses
- No monitoring system

MAKE A DIFFERENCE

Platforms / Technologies:

Web-UI performance:

- *Python*
- *Selenium*
- *Robot*
- *Browser mob proxy*
- *Har-storage*
- *mongoDB*

Backend performance:

- *Jmeter*
- *groovy*
- *R*
- *MsSql*
- *Shiny*
- *Dynatrace*
- *DotMemory*
- *DotTrace*

TaaS: Performance testing

Approach:

- Implementation of structured performance testing life cycle
- Integration of performance testing in continuous integration
- Automation of repetitive performance testing pipeline (smoke, load, scalability, stability, error detection)
- Maximal metric measurement with afterward analysis
- Maximal automatize of routine activities (data preparation and first analysis)
- Tight communication with onshore development team and customer teams

Scope:

- Describe performance strategy for the bank
- Build solid performance testing framework
- Establish reliable and effective performance testing process which can guarantee reliability and speed of all types of banking operations
- Fast and transparent feedback regards connections quality with third-party systems.

itera MAKE A DIFFERENCE

Initiation

- Define critical business values and goals
- Project schedule

Information gathering

- Infrastructure analysis
- Test Strategy
- Performance requirements analysis
- Design check-lists

Manual testing

- Script generating
- Manual preparation
- Manual run and bottleneck searching

Result Analysis & Reporting

- Recommendations how to fix performance issues
- Summary report

Automated run

- CI (TFS + teamcity)
- Distributed run
- Configuring test plan from CI tool
- CI pipeline

Connect monitoring

- Jmeter monitoring tool
- Dynatrace api
- Ckeck_mk monitoring

Automatization routine operation

- Data preparation
- Data cleaning
- Data saving
- Data wrangling

Report generation

- Static report
- Web-report

Automatized comparation with requirements

- Define metrics for comparations
- Defined requirements

Automatized requirement generation

- Arima forecasting expected results

Semiautomatic bottleneck detection

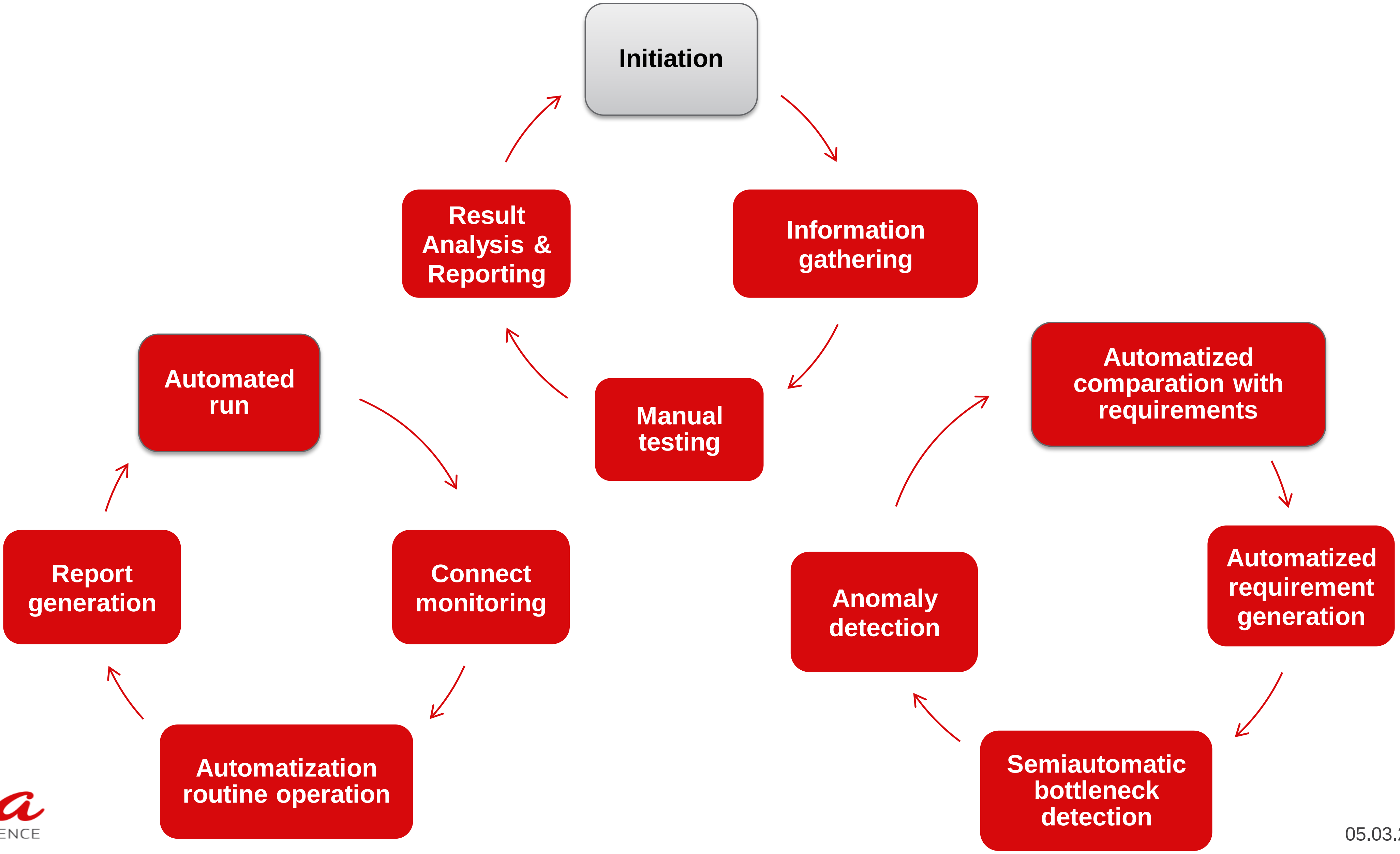
- Cluster analysis
- Correlation heatmaps

Anomaly detection

- Context depending
- Moving median
- Robust statistic

MAKE A DIFFERENCE

Project Timeline in real life


Initiation
Information gathering
Manual testing
Result Analysis & Reporting
Automated run
Connect monitoring
Automatization routine operation
Report generation
Automatized comparation with requirements
Automatized requirement generation
Semiautomatic bottleneck detection
Anomaly detection

itera MAKE A DIFFERENCE

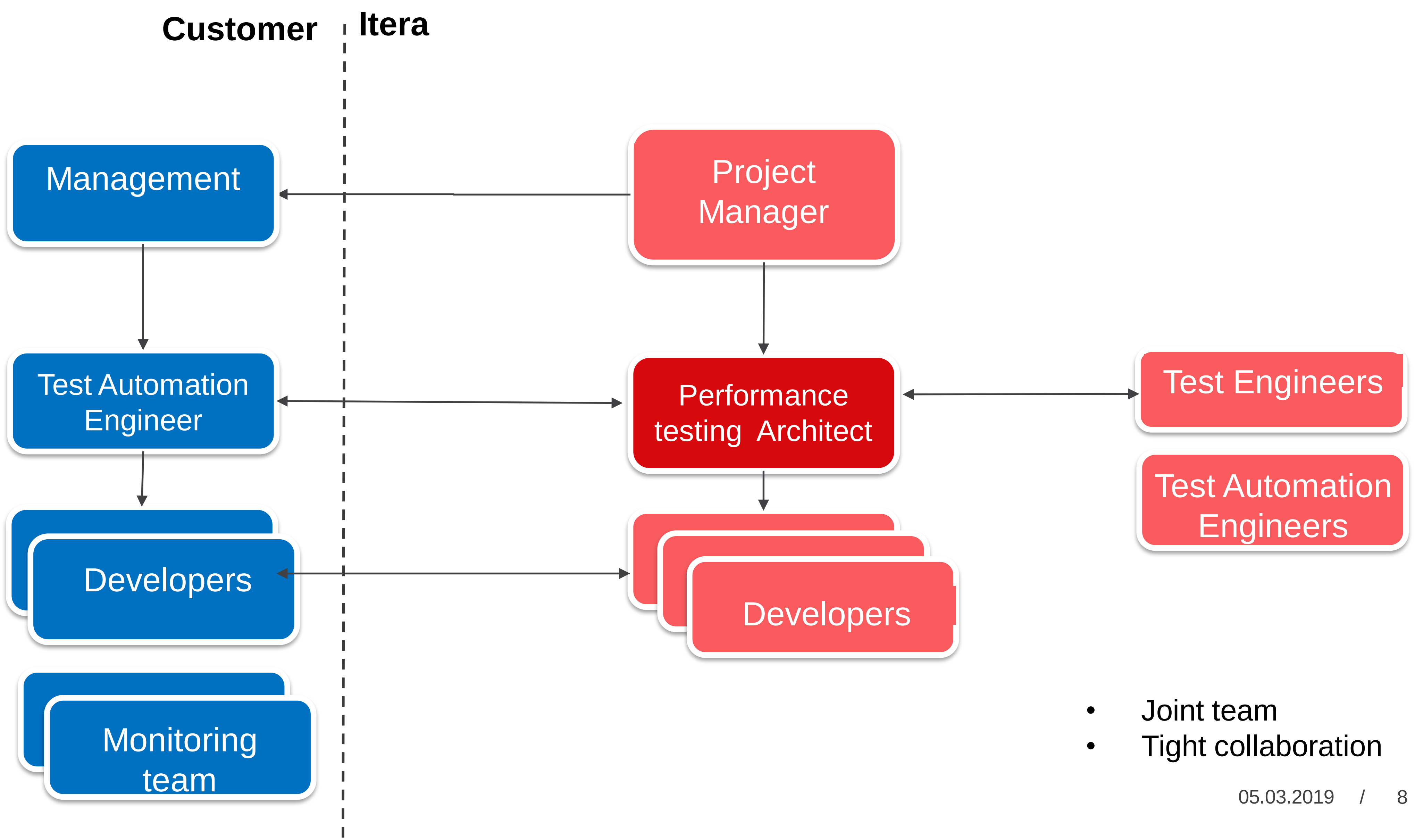
Results:

- Performance testing framework
- Performance testing CI
- Automated analysis
- Automated report generation
- Semiautomatic bottleneck detection (using ML)
- Integration with monitoring system

Deliverables:

- Test Strategy is introduced and followed
- Overall Test Scope is identified
- Performance testing framework (description and documentation)
- Scenarios and scripts automated (Smoke, load, scalability, stability)
- Customer issues analyzed (application, DB, infrastructure)
- Recommendation provided
- CI configured
- Daily ongoing automated reports and recommendation

Performance team integration



- Joint team
- Tight collaboration

Continuous PT

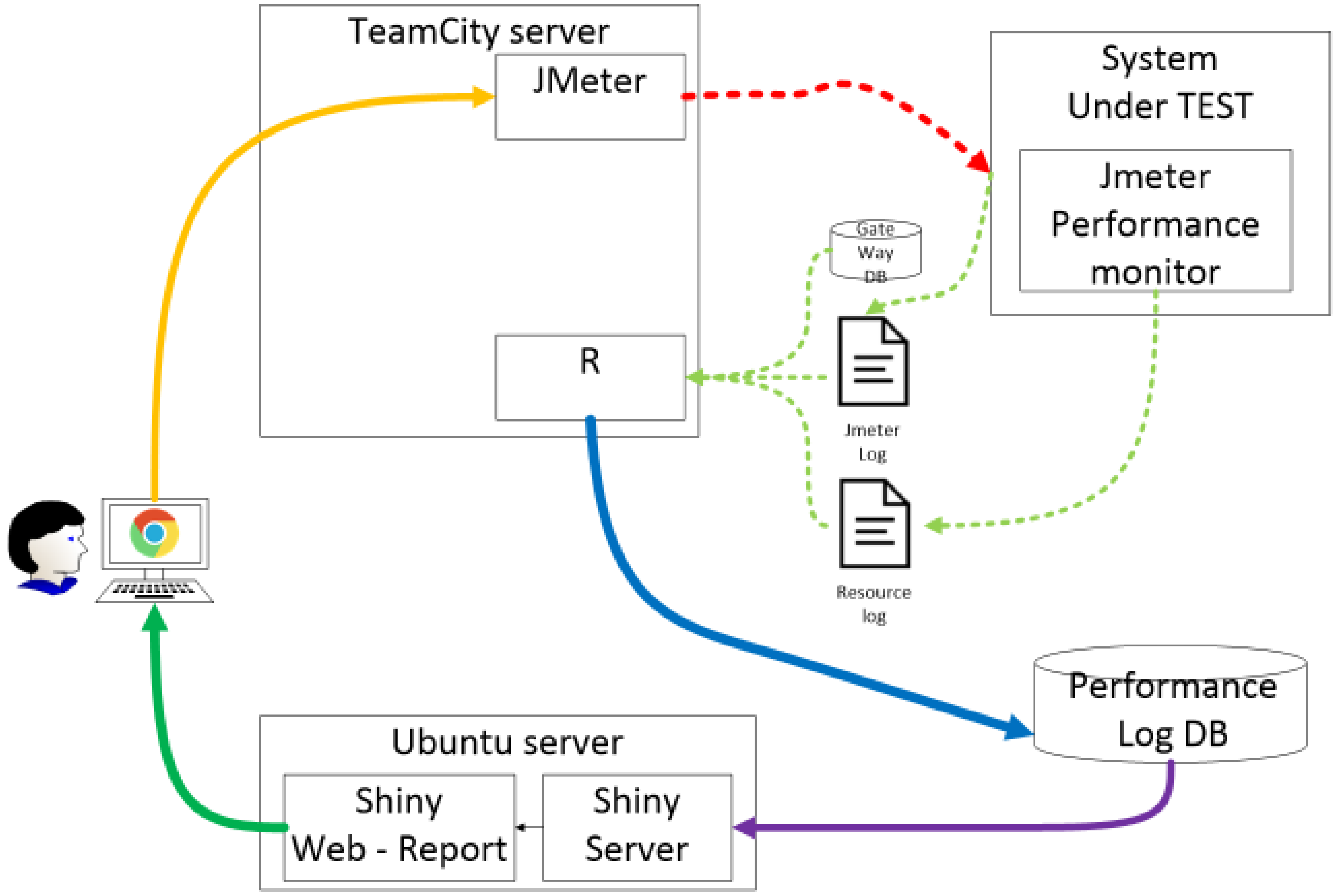
Report Generation

Machine learning in PT

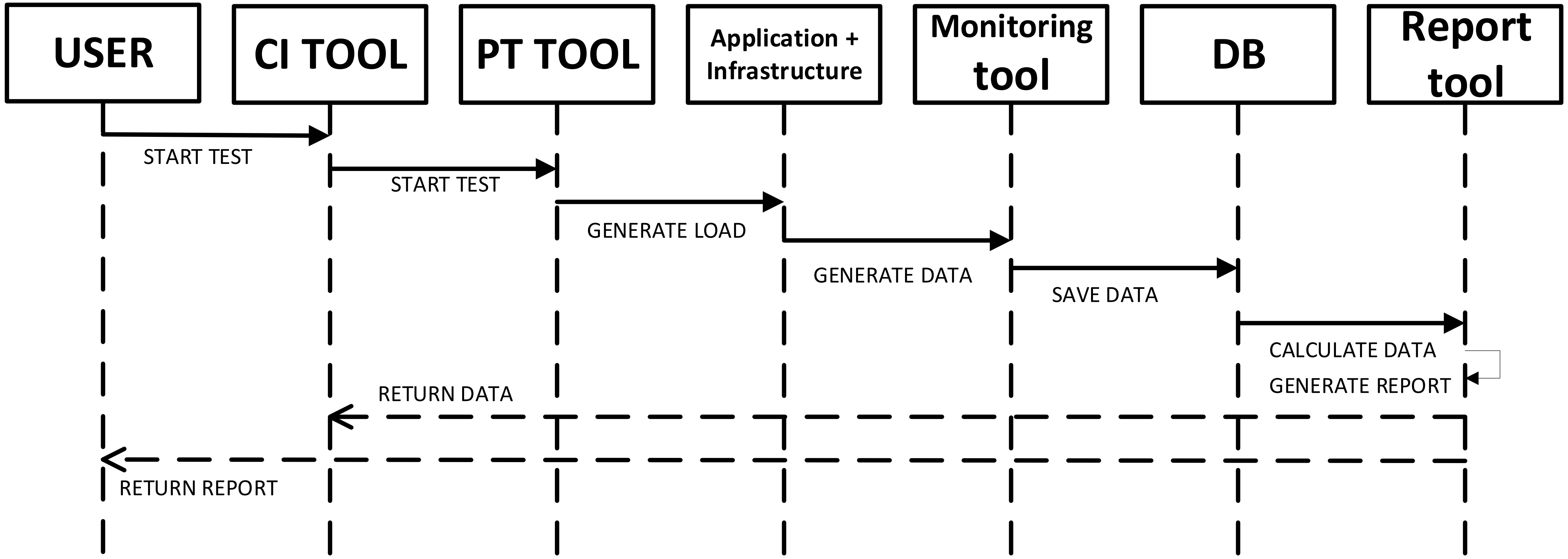
Recommendation for bottleneck detection

PT fairy ~~fails~~ tales

MAKE A DIFFERENCE


TeamCity server
JMeter
R
System Under TEST
Jmeter Performance monitor
Gate Way DB
Jmeter Log
Resource log
Performance Log DB
Ubuntu server
Shiny Web - Report
Shiny Server

Continuous performance testing



Collected data

Response times:

- Client
- Servers
- DB
- Gateways

Resources usage

- CPU
- RAM
- Network
- Disk IO

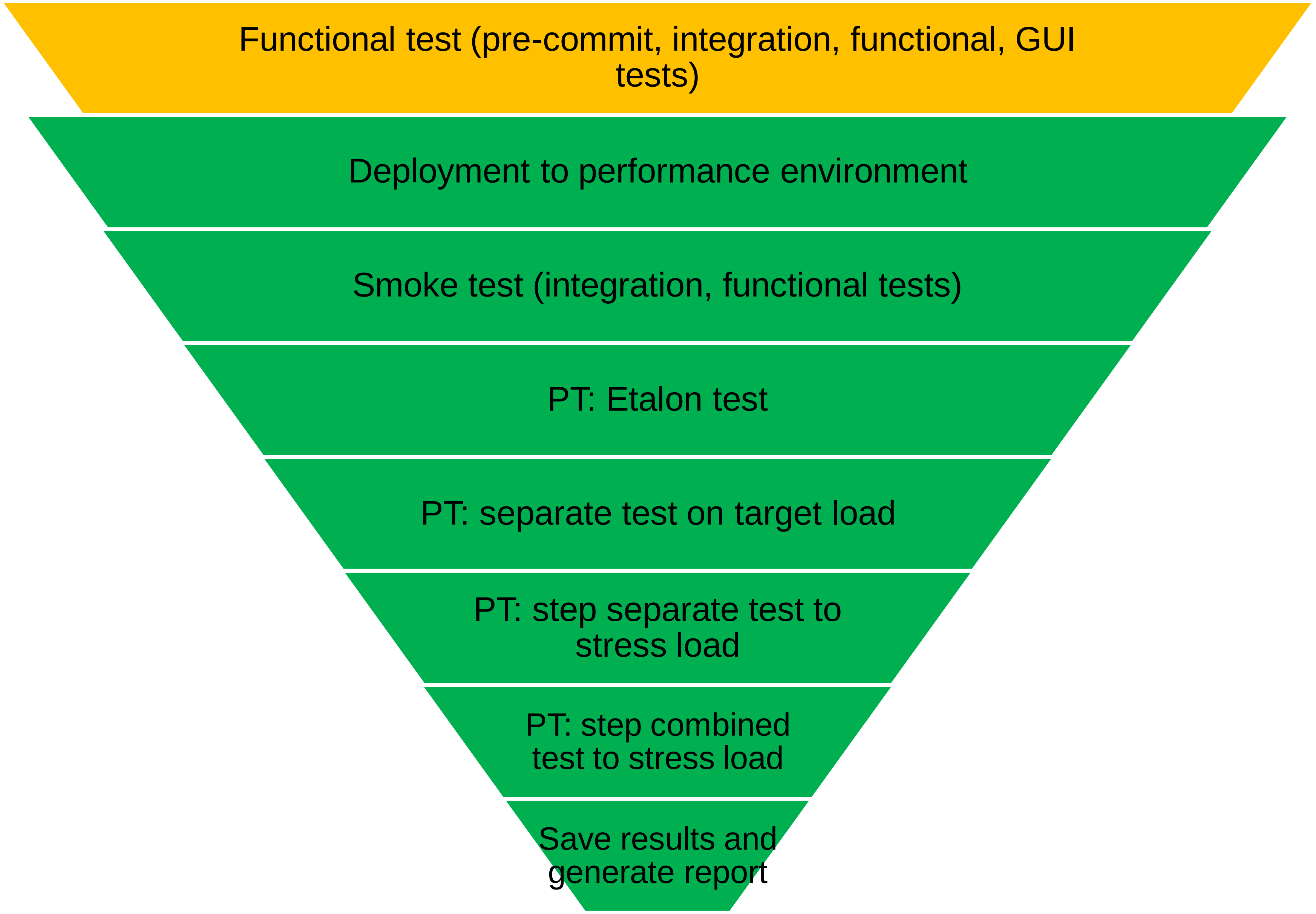
Request phases:

- DNS lookup
- Connecting
- Sending
- Waiting
- Receiving

Continuous performance testing – description

- User interact and configure everything from CI tool (web-ui)
- CI tool interact with all elements via it’s agents and instruments API
- Everything possible to configure from CI tool (Load, duration, versions, type of testing, report and log level)
- CI tool start load tools and load tools generate necessary load in one agent or distributed load, which also configured from CI tool
- CI tool gather all necessary information afterward performance test
- CI tool run analytical and report generator scripts
- CI tool save all information and logs (raw, cleaned, aggregated) into DB
- CI save all aggregated reports into artifacts
- User may interact with report tool vi web-ui
- Report tool should support filtering, comparing by many parameters (load, date, ifrastructure)

Continuous performance testing

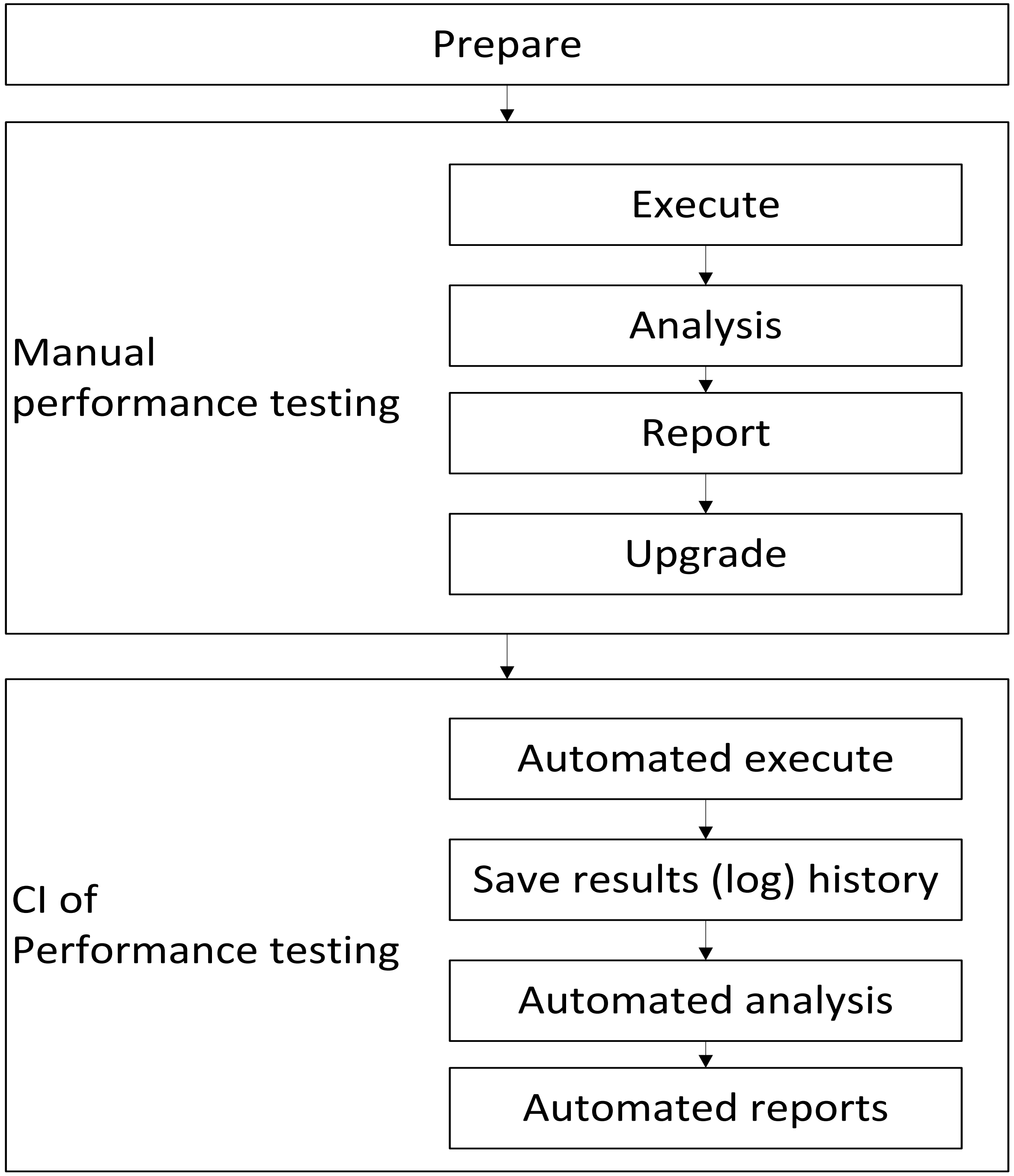

Functional test (pre-commit, integration, functional, GUI tests)
Deployment to performance environment
Smoke test (integration, functional tests)
PT: Etalon test
PT: separate test on target load
PT: step separate test to stress load
PT: step combined test to stress load
Save results and generate report

MAKE A DIFFERENCE

Performance testing pipeline description

- Each steps should based on information from successful previous step
- Successful functional tests is precondition for start performance testing (PT takes too much time for idle run)
- Smoke test check availability of functionality and make warm up
- Smoke tests duration should non be more than minimal time execution (e.g. 5 minutes)
- Etalon test gather information about measurements in perfect conditions (e.g. 1 RPS, no more than open connection etc.). Most of comparation will be done with etalon conditions
- Separate tests shows how system works without other impact. It gives possibility compare and exclude additional impact in future tests.
- Target load gives possibility check system under expected load.
- But overload during stress tests gives possibility get overload point and detects systems thresholds
- Saving all information gives possibility to detect bottlenecks and not to waste time for rerun all tests

Evolution Continuous performance testing



itera
MAKE A DIFFERENCE

Evolution Continuous performance testing

- Hardcoded scripts
- Manual executions
- Manual (eye analysis)
- Manual checks
- Manual report generators
- Configured scripts
- Configured runs
- Automatic executions
- Configured tests
- Results saving
- Automatic results analysis
- Automatic report generation
- Automatic anomaly detection
- Automatic recommendation generation

Continuous PT

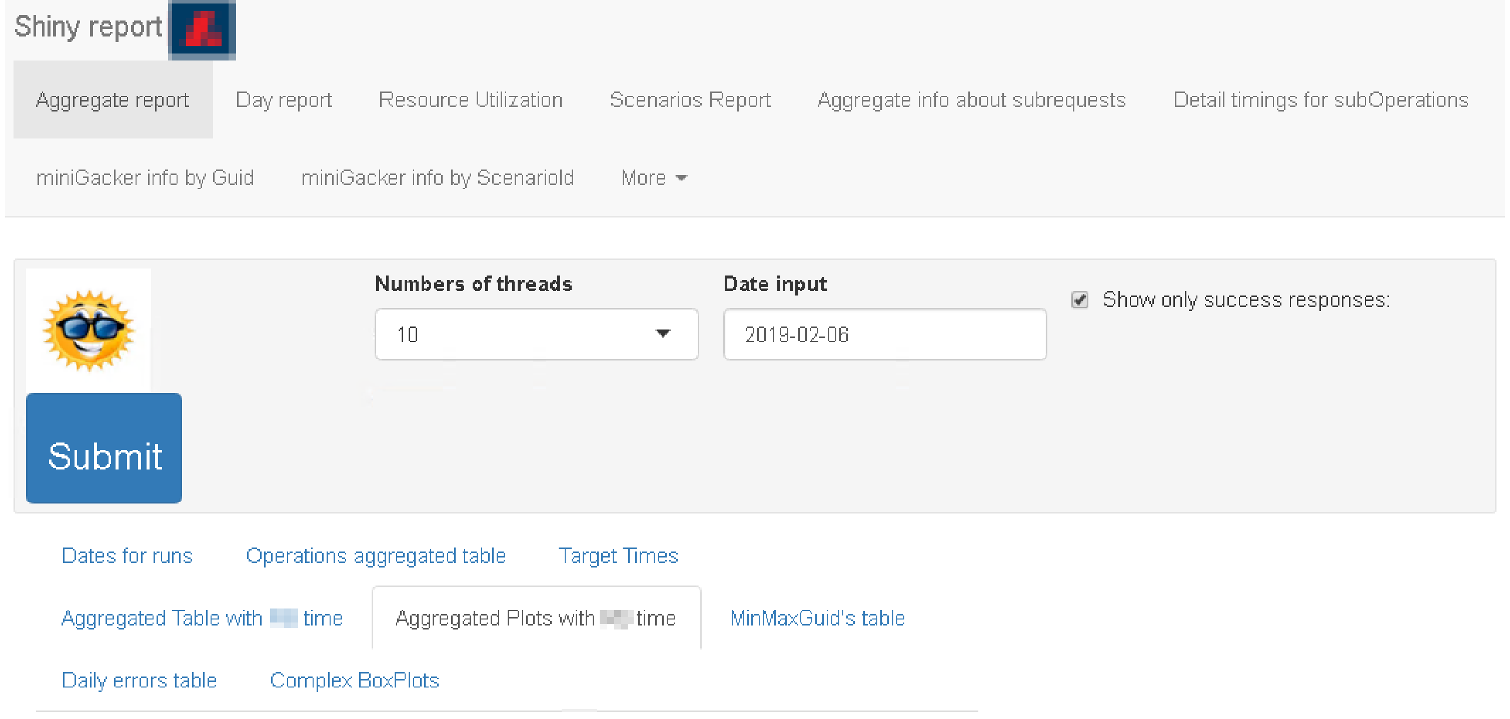
Report Generation

Machine learning in PT

Recommendation for bottleneck detection

PT fairy ~~fails~~ tales

Report generation

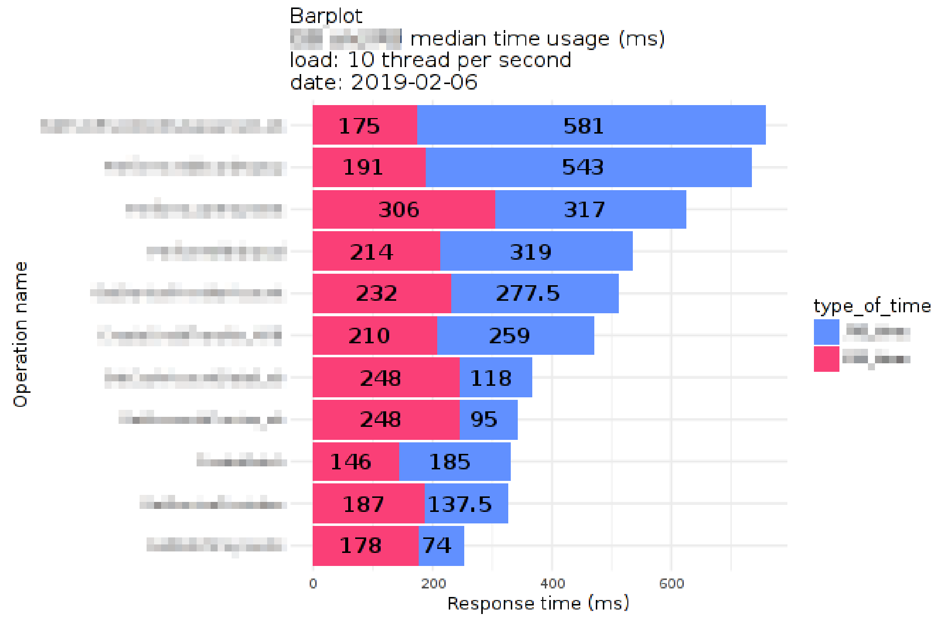

Shiny report
Aggregate report
Day report
Resource Utilization
Scenarios Report
Aggregate info about subrequests
Detail timings for subOperations
miniGacker info by Guid
miniGacker info by ScenarioId
More
Submit
Dates for runs
Operations aggregated table
Target Times
Aggregated Table with [redacted] time
Aggregated Plots with [redacted] time
MinMaxGuid's table
Daily errors table
Complex BoxPlots

Numbers of threads: 10

Date input: 2019-02-06

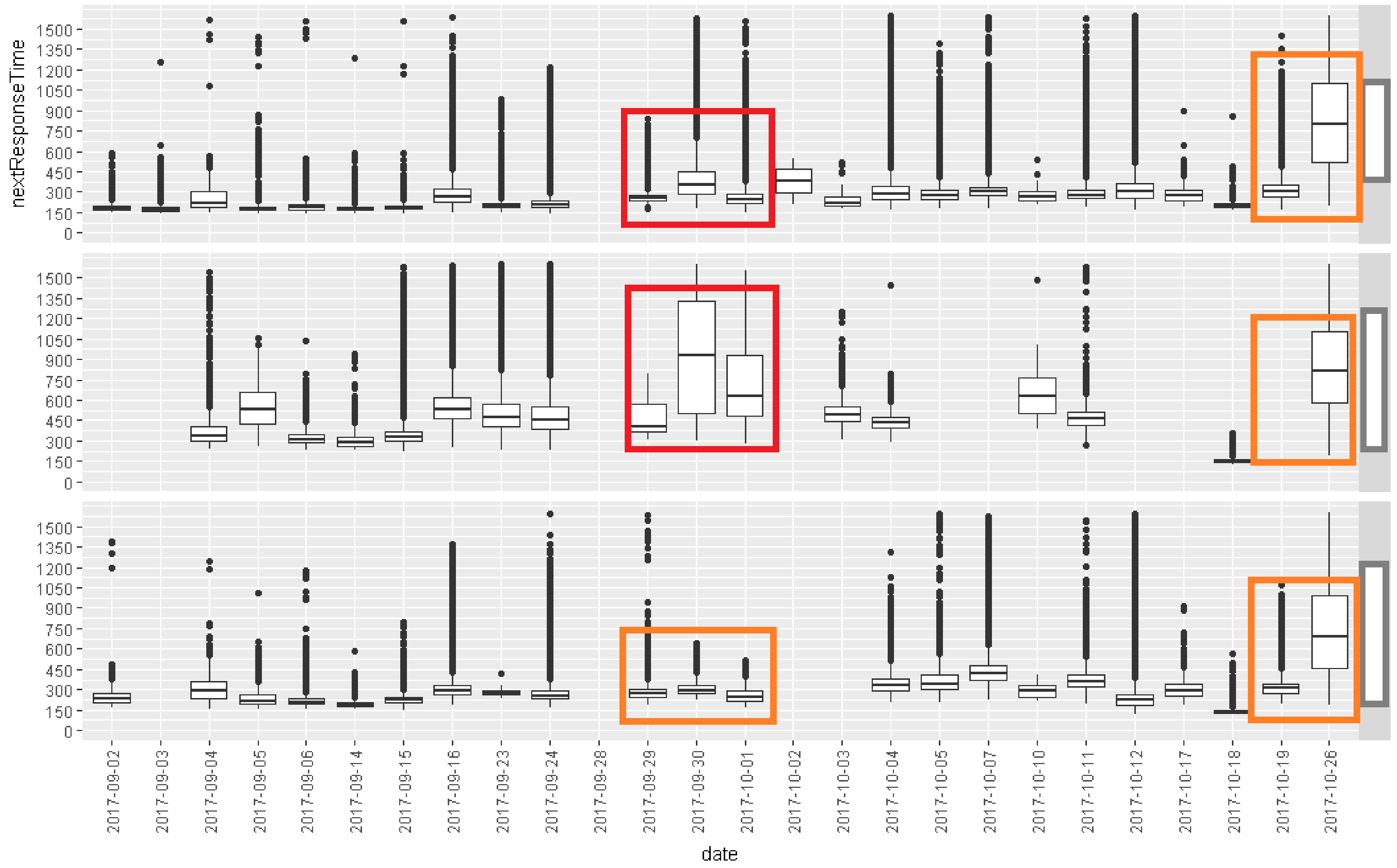
☑ Show only success responses:

Ratio between [redacted] times



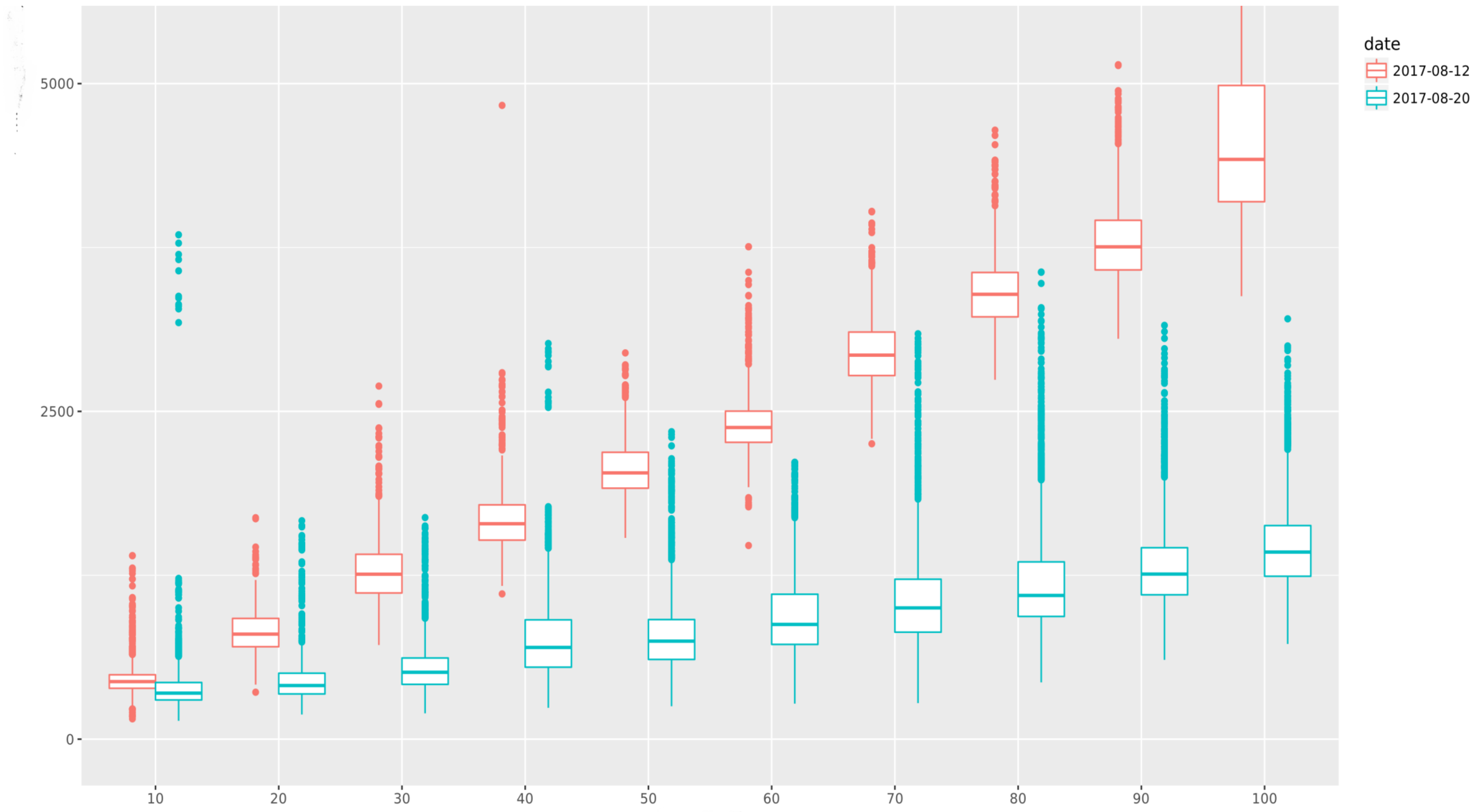
	🟥	🟦
[redacted]	175	581
[redacted]	191	543
[redacted]	306	317
[redacted]	214	319
[redacted]	232	277.5
[redacted]	210	259
[redacted]	248	118
[redacted]	248	95
[redacted]	146	185
[redacted]	187	137.5
[redacted]	178	74

Report generation



MAKE A DIFFERENCE

Report generation



MAKE A DIFFERENCE

Report generation

- Based on typical log files reports should be divided on next parts:
- Summary report – show only things which go bad
- Aggregated report -- describe all measurements with all statistics and graphs
- Interactive report – should give full access to all historical data with possibility to compare, filter, split and unite different data types and measurements.
  - It should be some application with visualization of data
  - Should provide access to all history of data
  - Should provide access for all involved person
  - Should provide export raw data for analysis in external tools
  - Should be possibility to compare results and measurement between dates, loads, layers and others….

Continuous PT

Report Generation

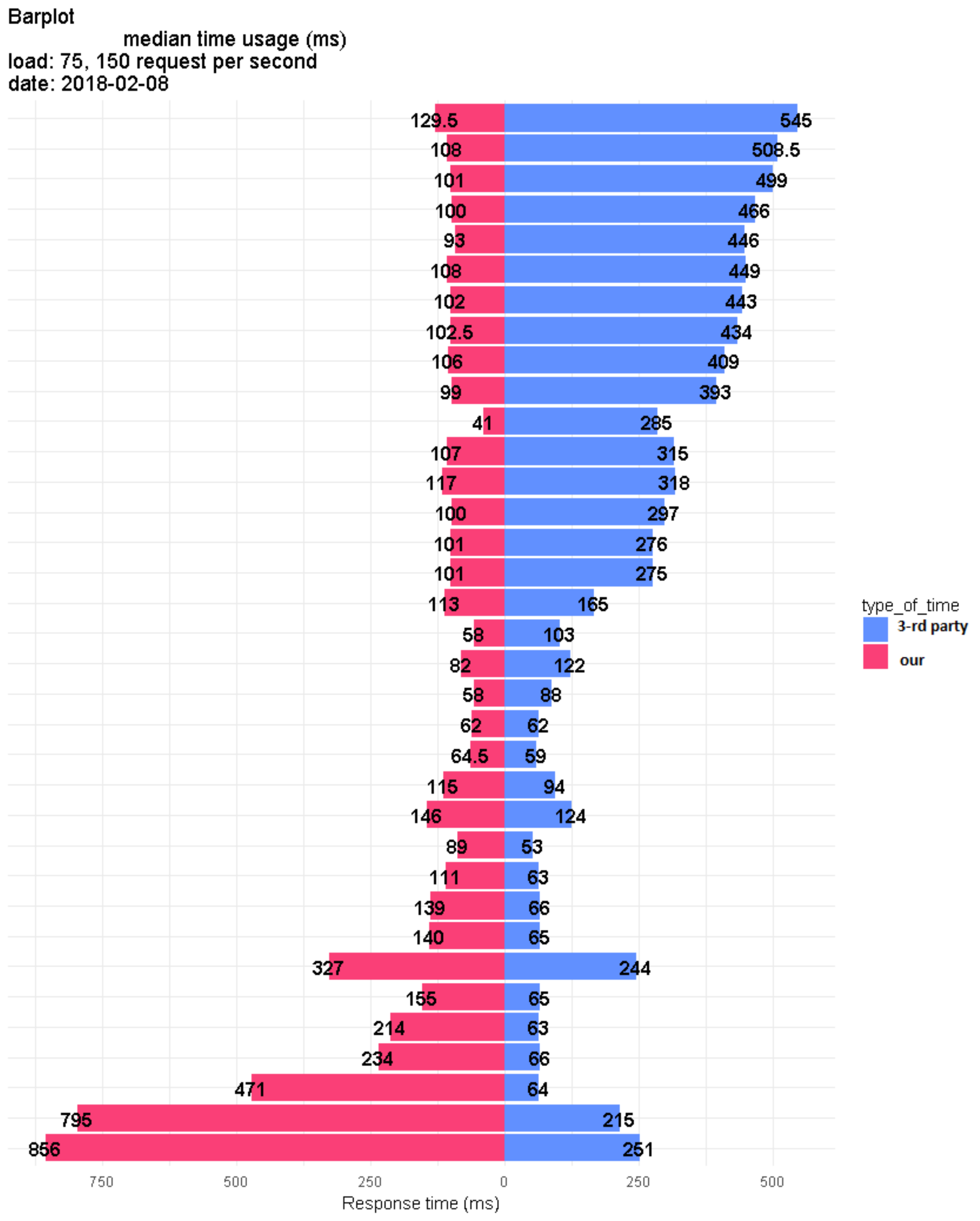
Machine learning in PT

Recommendation for bottleneck detection

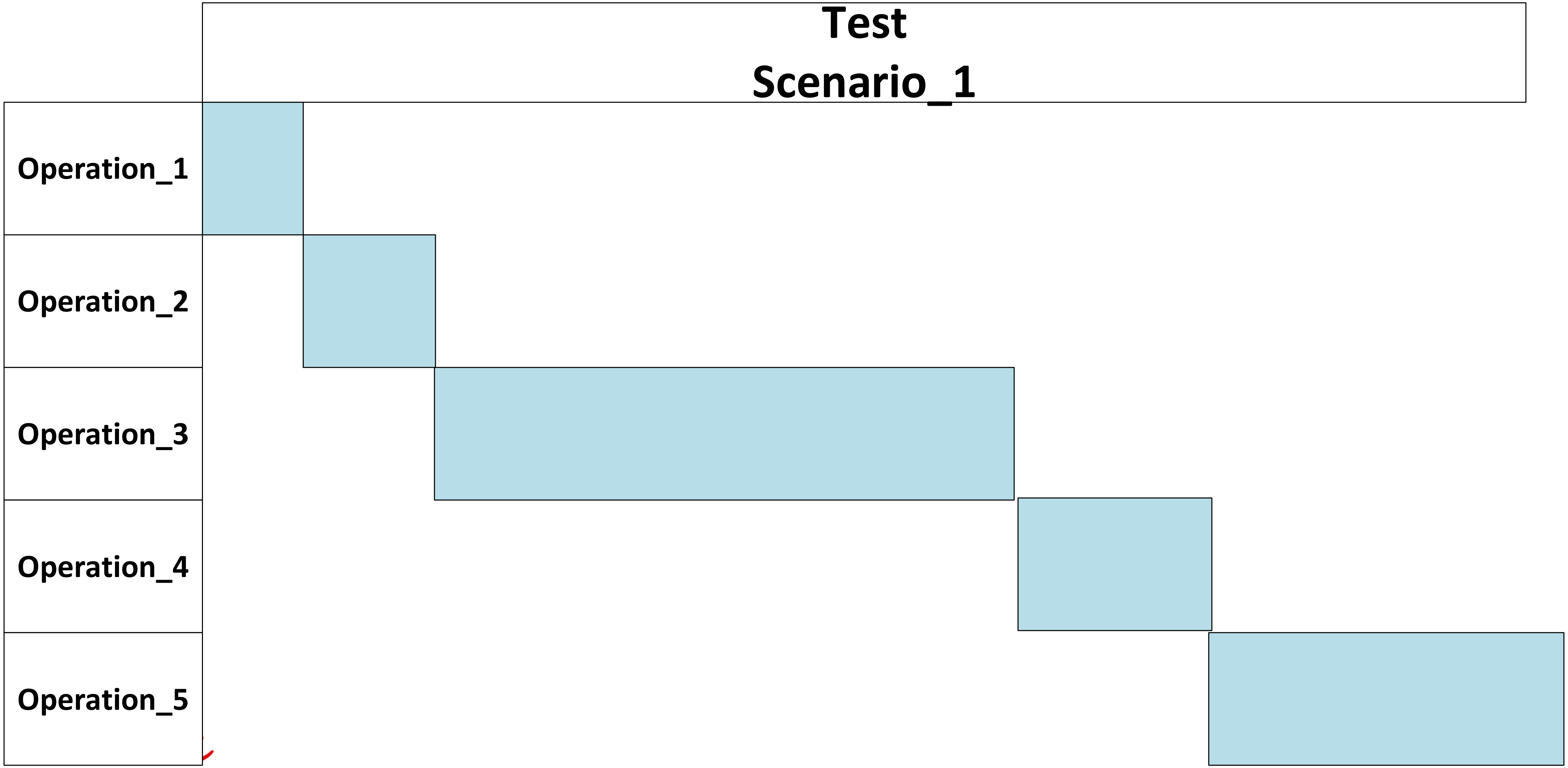
PT fairy ~~fails~~ tales

itera
MAKE A DIFFERENCE

Recommendation for bottleneck detections



Recommendation for bottleneck detections



Recommendation for bottleneck detections

	Test Scenario_1
Operation_1	0.1
Operation_2	0.15
Operation_3	0.95
Operation_4	0.2
Operation_5	0.7

Recommendation for bottleneck detections

	Test Scenario_1
Operation_1	0.1
Operation_2	0.15
Operation_3	0.95
Operation_4	0.2
Operation_5	0.7

Recommendation for bottleneck detections

	Test Scenario_1	Test Scenario_2	Test Scenario_3	Test Scenario_4	Test Scenario_5
Operation_1	0.1	0.99	0.2	0.99	0.3
Operation_2	0.15	0.9	0.25	0.1	0.2
Operation_3	0.95	0.1	0.95	0.3	0.86
Operation_4	0.2	0.5	0.75	0.4	0.7
Operation_5	0.7	0.1	0.99	0.77	0.25

Old build

	REST Response time	SOAP Response time	SQL Response time	CPU usage	RAM usage
REST Response time	▨	**0.75**	0.6	**0.7**	0.3
SOAP Response time		▨	**0.8**	**0.78**	0.2
SQL Response time			▨	0.4	0.2
CPU usage				▨	0.32
RAM usage					▨

−

New build

	REST Response time	SOAP Response time	SQL Response time	CPU usage	RAM usage
REST Response time	▨	**0.73**	0.6	**0.7**	0.3
SOAP Response time		▨	**0.9**	**0.58**	0.2
SQL Response time			▨	0.4	0.5
CPU usage				▨	0.32
RAM usage					▨

=

Difference

	REST Response time	SOAP Response time	SQL Response time	CPU usage	RAM usage
REST Response time	▨	0.02	0	**0**	0
SOAP Response time		▨	**-0.1**	**0.2**	0
SQL Response time			▨	0	**-0.3**
CPU usage				▨	0
RAM usage					▨

MAKE A DIFFERENCE

Recommendation for bottleneck detections

- To decrease time for bottleneck detection it is necessary gather information about all layers and all resources.
- Next it is necessary build correlation matrix predictor and predicted variables.
- Next step it is visualize it in heat map – and it shows what impact more on dependent variable.
- It is easiest way to drop all unimportant variables in variety of variables.
- For detecting what was changes with previous releases it just needed count differences between previous matrix and newest

Continuous PT

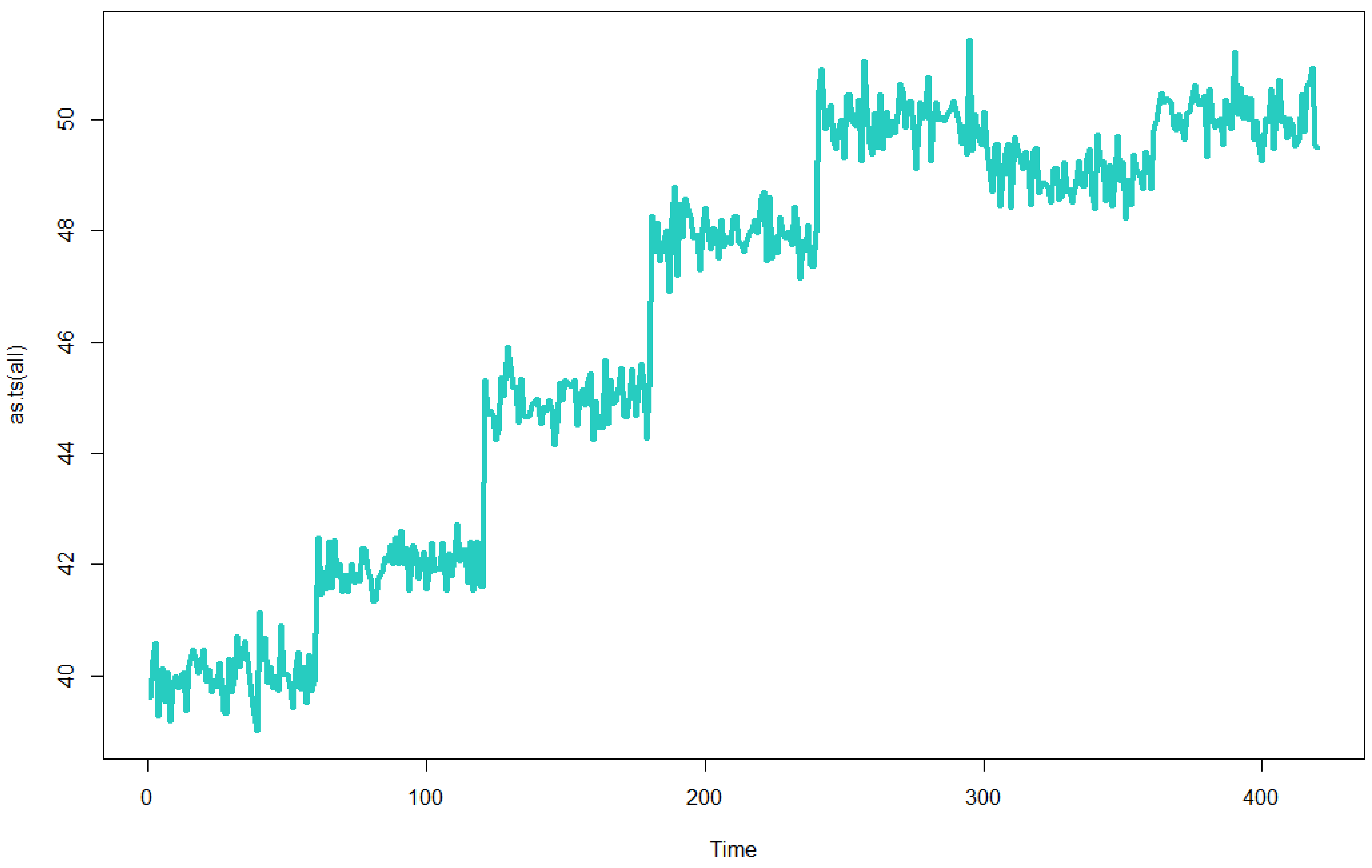
Report Generation

Machine learning in PT

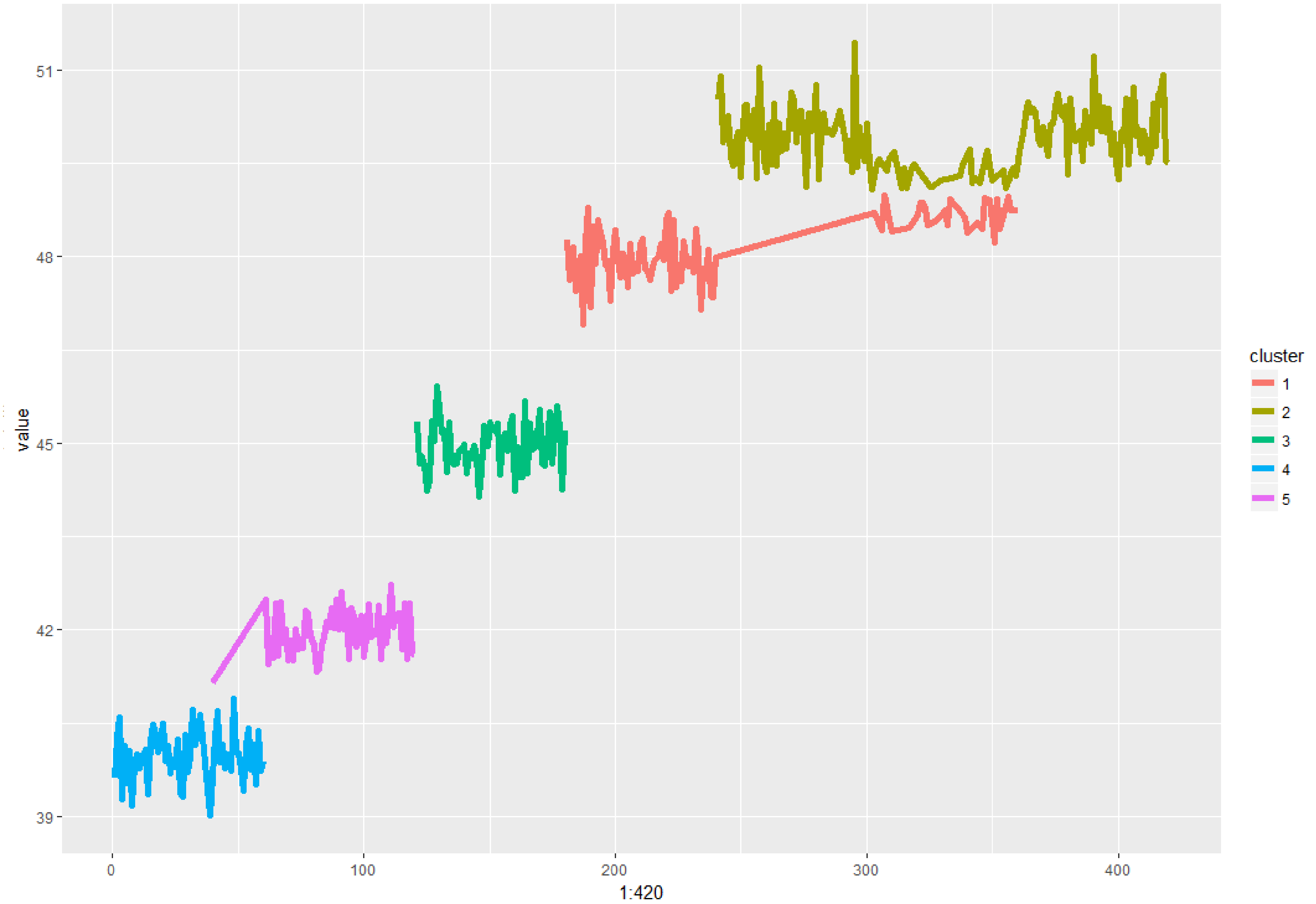
Recommendation for bottleneck detection

PT fairy ~~fails~~ tales

MAKE A DIFFERENCE



MAKE A DIFF



MAKE A DIFF

Machine learning in performance testing

- Using clustering in timeseries it is possible to detect bottlenecks-platoe, if plate of resources usage appears before that biggest load in performance tests

MAKE A DIFFERENCE

Continuous PT

Report Generation

Machine learning in PT

Recommendation for bottleneck detection

PT fairy ~~fails~~ tales

itera
MAKE A DIFFERENCE

Performance testing fairy ~~fails~~ tales

- Mailing from STAGE environment to real users
- Limit connections for STAGE environments
- Run performance test on TEST environments

?