Parallel Run Selenium Tests in a Good / Bad Way

Anton Semenchenko

Anton Semenchenko

Anton Semenchenko

EPAM Systems, Software Testing Manager

Creator of communities www.COMAQA.BY and www.CoreHard.by, founder of company www.DPI.Solutions, «tricky» manager at EPAM Systems. Almost 15 years of experience in IT, main specialization: Automation, C++ and lower development, management, sales.

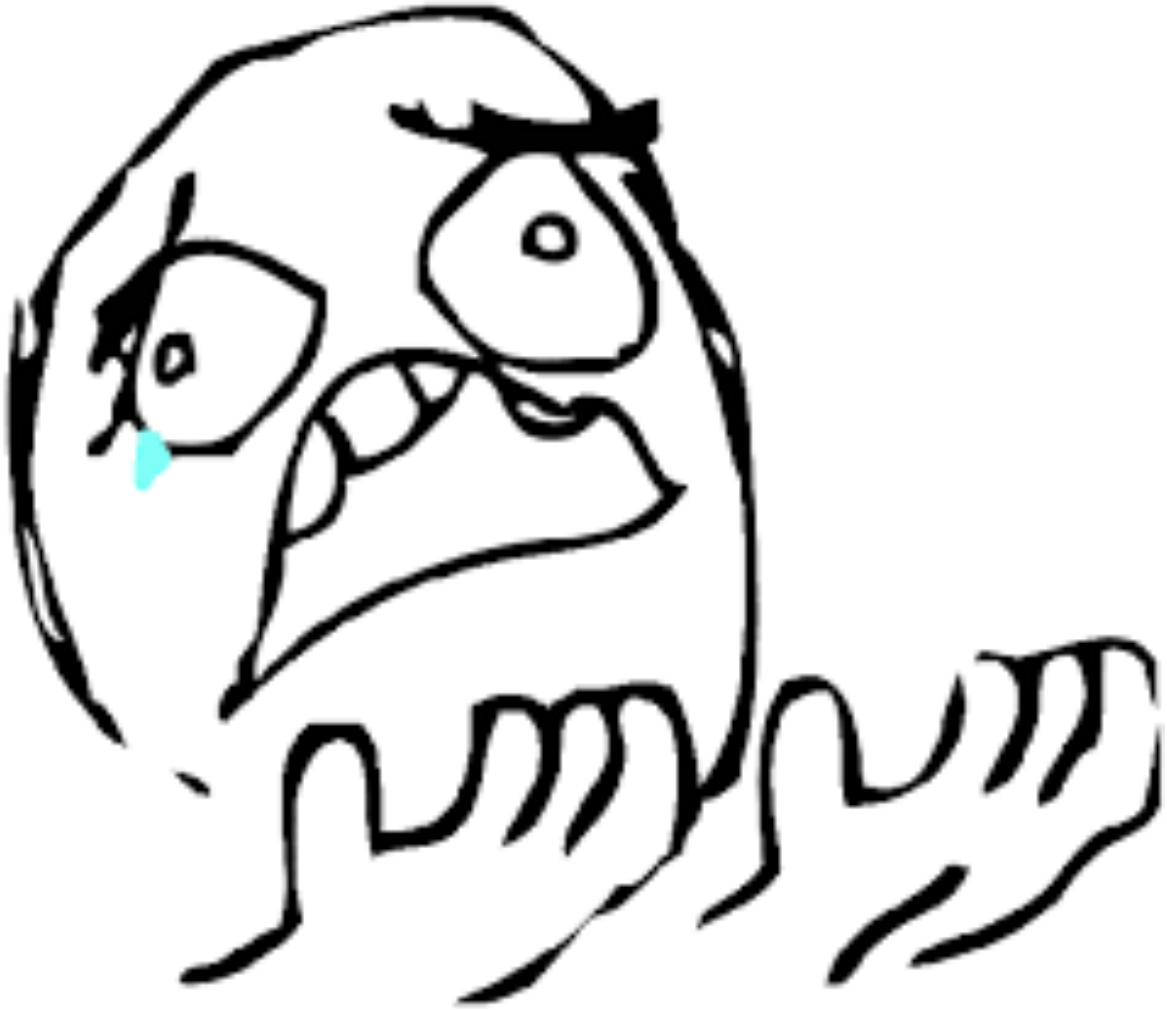
Agenda

COMAQA.BY

- Why do we need run tests in parallel?
- Challenge
- Solution
- Algorithm
- Static or Dynamic; stateless or statefull: Architecture related questions
- Metrics definition
- *Custom Test Runner: General scheme*
- Risks based “good” examples of customized Test Runners
- “Bad” examples of customized Test Runners
- Refactoring as a criteria – detailed information
- Metrics definition

Why do we need run tests in parallel?

COMAQA.BY



Challenge

Challenge

- How to invest min amount of time \ money to run tests in parallel
- How to maximize QA Automation ROI
- ROI >> 0

Solution

Let’s define Algorithm “How to run tests in parallel efficiently”

Algorithm

- Define Tests properly \ Tests attributes
- Define All shared entities
- Select proper Selenium WebDriver Wrapper
- Select proper Architecture
- Test Parallel approach or combination
    - Some standard Test Runner
    - Build instruments
    - Several Processes
    - Selenium Grid
    - OS \ Language specific multithreading

Why do we need run tests in parallel?

COMAQA.BY

2 types of reasons:

- Process related reasons
- Project specific reasons
    - Risk based reasons

Meaning:

- Test == Feedback Mechanism
- Test Value == Feedback Value

Feedback's in Scrum example

COMAQA.BY

Methodology – as a predictable way of risk management for some context

- **Pre planning grooming – run-time feedback from customer side**
- **Planning poker during Iteration planning – run-time feedback from team side mostly + customer side too**
- **Daily Stand up – daily feedback, Team side**
- **Iteration Demo – per iteration feedback, customer side**
- **Iteration Retrospective - per iteration feedback, Team side**
- **Pair programming (as an example) – run-time technical feedback**
- **Unit Tests + CI – close to run-time technical feedback**
- **QA Automation Tests + CI + Report + Report Analyses – daily feedback mechanism**
- ***And so on***

Feedback's in Scrum example

COMAQA.BY

Methodology – as a predictable way of risk management for some context

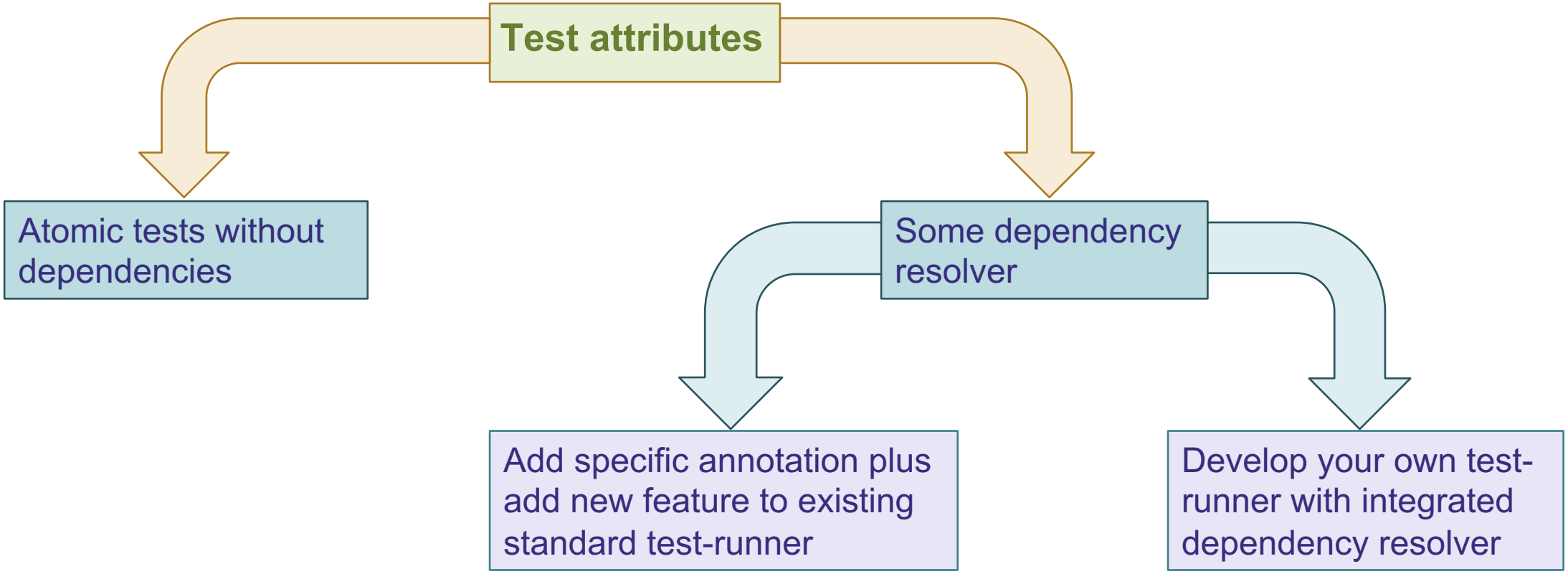
Goals

1. Decrease QA Automation “Window”
2. Decrease Test results analysis “Window”
3. Increase Regression frequency
4. Decrease \ Optimize hardware utilization
    1. Electricity
    2. Hardware costs (buy or rent)
5. Increase QA Automation ROI >> 0

Algorithm

- Define Tests properly \ Tests attributes
- Define All shared entities
- Select proper Selenium WebDriver Wrapper
- Select proper Architecture
- Test Parallel approach or combination
    - Some standard Test Runner
    - Build instruments
    - Several Processes
    - Selenium Grid
    - OS \ Language specific multithreading

Algorithm – The best way



Algorithm – The best way

COMAQA.BY

How to choose – ROI calculator as a solution

Atomic tests without dependencies

- Decreased test debug time
- Decreased test update/support time
- No time spent on dependency definition
- No time spent on dependency resolver
- Lower entry barrier for newcomers
- Less documentation

Some dependency resolver

- Decreased time to run
- Lower hardware costs + electricity

Algorithm – The best way

In most cases – no dependencies:

- More expensive specialists
- Higher hardware and electricity costs

Algorithm – The best way

- **Define All shared entities**
- **Improve Architecture**
- **For example:**
  - Pre-steps using DB
  - DB Layer:
    - Singleton
    - Thread-safe
    - Optimize (use profiler)
    - Migrate to Singleton-Bus (instance per thread, in a thread-safe way)
    - Solution: in an iteration-based way, start from the simplest singleton

Algorithm – The best way

- **Define All shared entities**
- **Improve Architecture**
- **For example:**
  - Logger
  - Tracer
  - Report Engine:
    - +1 more reason to integrate Test Runner and Report Engine, knows how to run in parallel and integrate reports pieces into one document

Algorithm – The best way

- **Define All shared entities**
- **Improve Architecture**
- **For example:**
  - Data-provider
  - Any other orthogonal entity:
    - Define
    - Isolate
    - Remember

Algorithm – The best way

- **Select proper Selenium WebDriver Wrapper**
  - Mature
  - Thread-safe
  - Easy-to-use “downcast”
  - Examples:
    - Selenide – easy to use
    - JDI evolution
    - Serenity – more complicated to use

Algorithm – The best way

- **Select proper Architecture**
  - Stateless architecture
  - Static Page Object
  - Isolate all orthogonal shared entities
  - Use “From Conditional to Patterns and reverse” Refactoring as a metric + ROI to prof

Algorithm – The best way

- **Select proper Architecture**
  - Convert to Stateful architecture
  - Dynamic Page Object
  - Update all isolated orthogonal shared entities
  - Re-calculate ROI and reuse “From Conditional to Patterns and reverse” in a systematical way

Algorithm

- Define Tests properly \ Tests attributes
- Define All shared entities
- Select proper Selenium WebDriver Wrapper
- Select proper Architecture
- Test Parallel approach or combination
    - Some standard Test Runner
    - Build instruments
    - Several Processes
    - Selenium Grid
    - OS \ Language specific multithreading

Algorithm – The best way

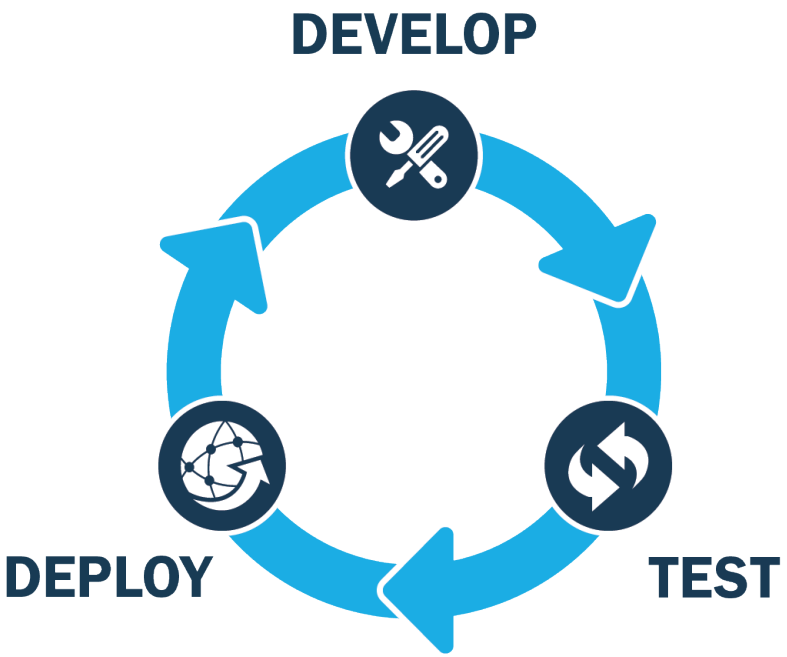
- **Some standard test-runner**
  - The lowest layer
  - Standard (or some specialized), stable, efficient, simple, most of instruments use it as a basement

Algorithm – The best way

- **Build instruments**
  - Use parameters to configure test run
  - Use Test Runner as a basement
  - Some kind of Test Runner wrapper
  - + 1 layer => less stable, less efficient, sometimes easier to use
  - Anyway, doesn't work without Test Runner
  - Maven example: Thread Count and param: Test Method, Test Class, both

Algorithm – The best way

- **Several processes**
  - Using CI
    - Team City (Job Configuration)
    - Jenkins (Job Configuration)
    - Extensive way

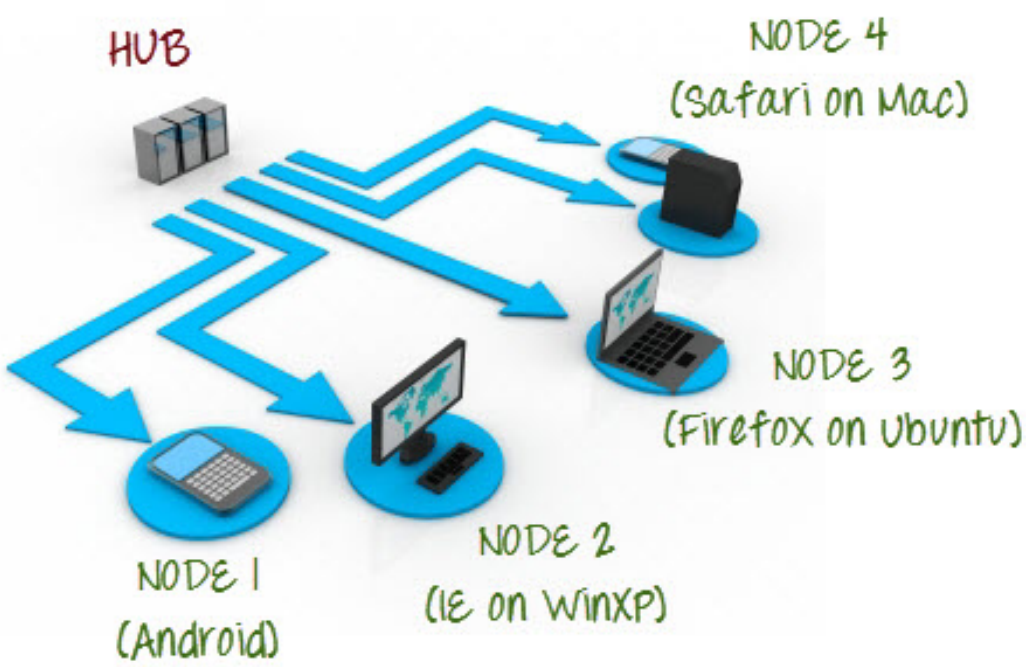


Jenkins

TeamCity

Algorithm – The best way

- **Selenium Grid**
  - Infrastructure
  - Could be used for indirect test run, but this is not primary purpose
  - Could be combined with all other solutions



Algorithm – The best way

- **Language / OS specific mechanisms**
  - Language, focusing on multithreading
  - Java
    - Process
    - Thread / Fork
    - JVM (Java property config or command line arguments)

Out of scope

OS and Language related questions of multi-threading

Reasons

COMAQA.BY

Transition complexity

With proper architecture (plus mature, not self-developed, WebDriver wrapper, like Selenide) -> 0, just 3-5-10 places in a solution, QA Automation Architect or Developer should invest several hours

With improper architecture (without any wrapper or with a self-developed WebDriver wrapper) -> complicated question

Invest days or weeks to Update Architecture and wrapper (better, to use, mature one)

Invest weeks or months to update tons of source code places

Without any architecture -> nightmare

Invest weeks to redesign tests using proper Architecture and mature wrapper

Invest months or even years just to update tests

ROI as a metric

Architecture related questions

Static or Dynamic; stateless or statefull: Architecture related questions?

Architecture related questions

1. Static: transform to parallel run
2. Dynamic: transform to parallel run
3. Static <=> Dynamic: transformation criteria
4. Transformation example
5. Detailed information about transformation
6. Static or Dynamic – as an architecture basement

Leak of stateless examples

COMAQA.BY

Leak of stateless examples in standard Selenium documentation

IMHO: due to Selenium development process

A tiny group of extra professionals \ developers

No processes

No backlog

No priorities

No ccommittee for backlog and backlog items prioritization

No “iterations”

How to add new feature

Just to implement and then add for review

This is an example !

State-less or state-full solution?

COMAQA.BY

1. Let’s compare:

Photo

Share – looks like parallelism (easy parallelism).

Video

Share – looks like parallelism (not trivial parallelism).

State-less or state-full solution?

COMAQA.BY

1. How easy transform solution from “single” to “multi” threading (to decrease “QA Automation Windows”)

State-less – like share a photo

Just 5 minutes of work.

State-full – like share a video

Not trivial task, could be a night mare.

2. Summary

prefer state-less solutions to state-full solutions in moooooost cases;

before start implementation a state-full solution, please, take a break for a minute, and re-thing everything again, possibly you can find a proper state-less solution.

Object or static class \ State-full or state-less solution?

COMAQA.BY

1. Static class

could be implemented as a state-less solution easily

2. Object

State-full solution in 99,99% cases

3. Summary

prefer static class based solutions (state-less) to object based (state-full) in moooooost cases;

before start implementation based on objects, please, take a break for a minute, and re-thing everything again, possibly you can find a proper solution based on static classes.

Replace Conditional with Polymorphism as criteria

COMAQA.BY

“Replace Conditional with Polymorphism refactoring” as a Static <=> Dynamic: transformation criteria

Replace Conditional with Polymorphism and vice versa

COMAQA.BY

1. You have a conditional that chooses different behavior depending on the type of an object.
2. Move each leg of the conditional to an overriding method in a subclass. Make the original method abstract.
3. And vice versa
4. Example

Replace Conditional with ... more sophisticated options

COMAQA.BY

1. Replace Conditional Dispatcher with Command Design Pattern

Create a Command for each action. Store the Commands in a collection and replace the conditional logic with code to fetch and execute Commands.

2. Replace Conditional Logic with Strategy Design Pattern

Create a Strategy for each variant and make the method delegate the “calculation” to a Strategy instance.

3. Replace Conditional Logic with State Design Pattern

Create a State for each variant as a part of “State Machine” and make the method delegate tricky “calculation” to the “State Machine”.

Replace Conditional with Polymorphism – detailed description

1. Problem:

You have a conditional that performs various actions depending on object type or properties.

2. Solution:

Create subclasses matching the branches of the conditional.

In them, create a shared method and move code from the corresponding branch of the conditional to it.

Replace the conditional with the relevant method call.

The result is that the proper implementation will be attained via polymorphism depending on the object class.

Example plus some details

1. Transformation example
2. Detailed information about transformation
3. Static or Dynamic – as an architecture basement

Custom Test Runner: General scheme

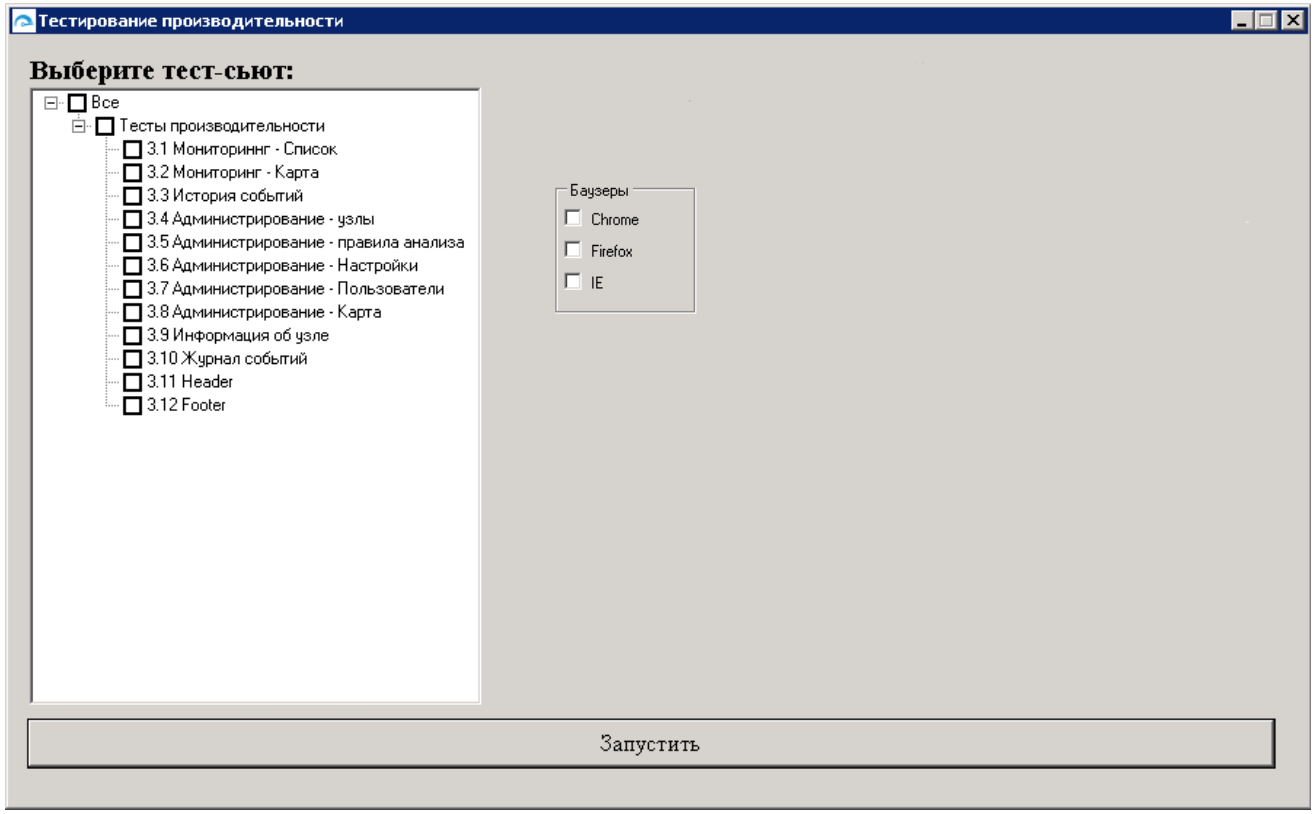
Risks based “good” examples of customized Test Runners

Custom Java test runner first iteration

COMAQA.BY

The runner allowed selecting test suite and browser

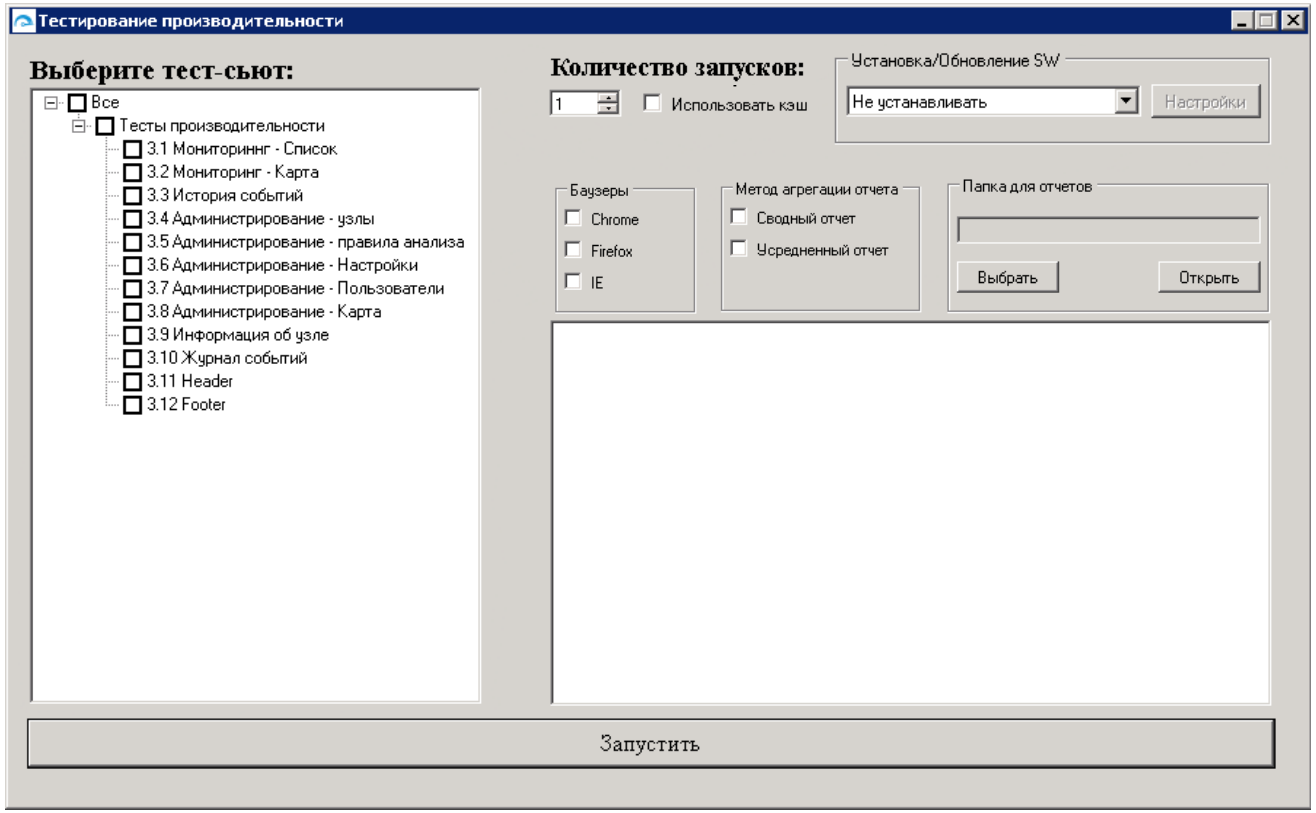
CSV-report as s result



Custom Java test runner final iteration

COMAQA.BY

Added various features for test-run configuration



Custom Java test runner parallel engine

COMAQA.BY

We get the number of nodes from Selenium Grid

Get node ip’s from Grid

Get the number of cores on each node (use the smallest number)

Use this info to configure the number of parallel threads using standard test-runner (Test-NG)

Use the power of both Test-NG and Selenium Grid

Custom Java test runner next iteration

- Reworked the re-run mechanism
- Changed CSV-reports for detailed Excel spreadsheets with color coding
- Integrate with load emulator

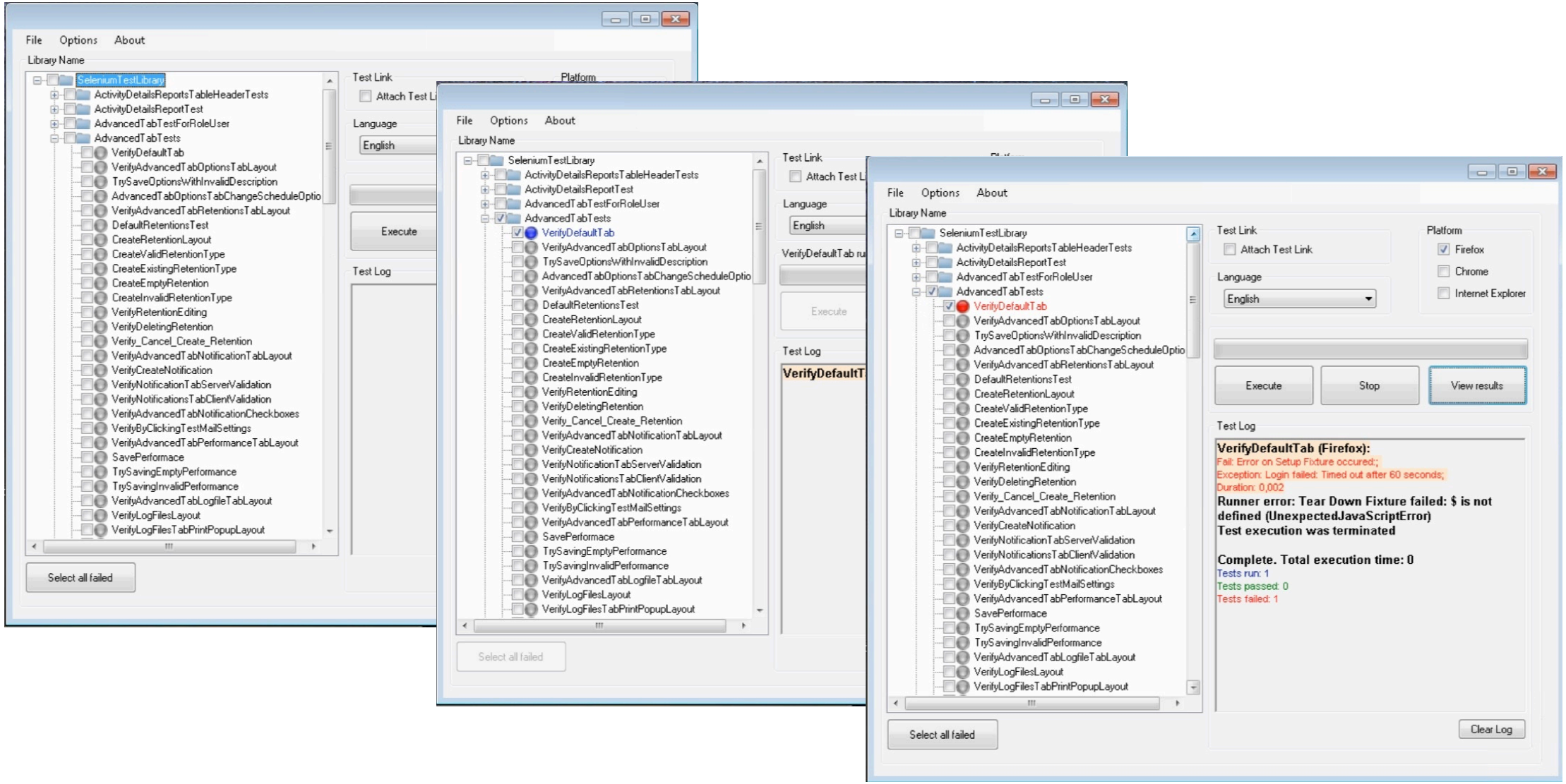
	A	B	C	D	E
1		Операционная система:	Windows 7		
2		Версия операционной системы:	6.1		
3		Архитектура:	amd64		
4		Общий объем памяти:	132902241		
5		Свободно памяти:	112817810		
6					
7	№	Наименование теста	Chrome	Firefox	InternetExplorer
8	1	3.5.1 Загрузка страницы: Администрирование - Правила анализа	9.68	10.38	11.648
9	2	3.5.10 Переключение режима редактирования правила	10.221	10.652	10.892
10	3	3.5.11 Сохранение простого правила	13.56	13.473	13.517
11	4	3.5.12 Сохранение сложного правила	13.124	13.035	13.249
12	5	3.5.13 Удаление простого правила	0.267	0.248	0.311
13	6	3.5.14 Удаление сложного правила	3.573	3.64	3.726
14	7	3.5.15 Создание копии простого правила из простого правила	10.942	11.336	11.612
15	8	3.5.16 Создание копии сложного правила из простого правила	12.513	12.897	13.015
16	9	3.5.17 Сохранение редактированного простого правила	10.214	10.784	10.643
17	10	3.5.18 Создание копии из сложного правила анализа	12.024	11.964	12.345
18	11	3.5.19 Нажатие: Передать правило вниз по каскаду	0.246	0.314	0.351
19	12	3.5.2 Загрузка правила по клику	3.128	3.309	3.512
20	13	3.5.20 Отменить выбор: Передать правило вниз по каскаду	0.281	0.317	0.312
21	14	3.5.3 Открытие окна создания простого правила	1.641	1.863	1.945
22	15	3.5.4 Открытие окна создания сложного правила	1.663	1.763	1.688
23	16	3.5.5 Настройки правил анализа: Сохранить	3.866	4.026	4.12
24	17	3.5.6 Настройки правил анализа: Отменить	12.779	12.76	13.043
25	18	3.5.7 Создание правила анализа: Сообщение об ошибке	11.341	11.523	11.527
26	19	3.5.8 Вызов диалога подтверждения удаления	Failed	Failed	Failed
27	20	3.5.9 Переход между правилами анализа	3.425	3.27	3.611
28					

“Bad” examples of customized Test Runners

“Bad” examples of customized Test Runners

- Custom test-runner
- XML-configuration-based
- Features:
    - Test, not test-suite level orientation
    - XML-config for multithreading

“Bad” examples of customized Test Runners



“Take away” points

Algorithm

1. **Define Tests properly \ Tests attributes**
2. **Define All shared entities**
3. **Select proper Selenium WebDriver Wrapper**
4. **Select proper Architecture**
5. **Test Parallel approach or combination**
    1. Some standard Test Runner
    2. Build instruments
    3. Several Processes
    4. Selenium Grid
    5. OS \ Language specific multithreading

Summary

COMAQA.BY

Step by step summary

1. **Algorithm**
2. **Static or Dynamic; stateless or statefull: Architecture related questions**
3. **Metrics definition**
4. ***Custom Test Runner: General scheme***
    1. *Risks based “good” examples of customized Test Runners*
    2. *“Bad” examples of customized Test Runners*

What’s next?

COMAQA.BY

Let’s go and discuss all open questions in an informal way ☺

Refactoring by Martin Fowler

COMAQA.BY

1. “Refactoring is a controlled technique for improving the design of an existing code base.”
2. “Its essence is applying a series of small behavior-preserving transformations, each of which "too small to be worth doing".”
3. “The cumulative effect of each of these transformations is quite significant.”
4. “By doing Refactoring in small steps you reduce the risk of introducing errors. You also avoid having the system broken while you are carrying out the restructuring - which allows you to gradually refactor a system over an extended period of time.”

Encapsulation – the most important OOP principle

COMAQA.BY

1. **Ask yourself "how can I hide some details from the rest of the software?“**

2. **What is encapsulation?**

hide variability

hide complexity

Details

"conflict of interests“

“tech” discussions

3. **Example of public member or private member + setter/getter**

What is really hidden?

Where is simplicity?

Refactoring and Design Patterns by Martin Fowler

COMAQA.BY

1. “There is a close relationship between refactoring and patterns.”
2. “Often the best way to use patterns is to gradually refactor your code to use the pattern once you realize it’s needed.”
3. “Joshua Kerievsky’s Refactoring to Patterns explores this topic, making this a great topic to learn about once you’ve got the basic refactoring's under your belt.”
4. “From Refactoring To Design Pattern” path – from pure design to adequate design
5. “From ~Design Patterns To Refactoring” path – from over design to adequate design

Refactoring and Design Patterns by Joshua Kerievsky

COMAQA.BY

1. “Refactoring to Patterns is the marriage of refactoring - the process of improving the design of existing code - with patterns, the classic solutions to recurring design problems.”

2. “Refactoring to Patterns suggests that using patterns to improve an existing design is better than using patterns early in a new design. This is true whether code is years old or minutes old.”

3. “We improve designs with patterns by applying sequences of low-level design transformations, known as refactoring's.”

4. And vice versa

Refactoring Catalog / Language

COMAQA.BY

1. **There are more then 90 types of refactoring**
2. **Refactoring types that relate to a particular field is called a ”Refactoring Language”**
3. **”Refactoring Language” gives a common terminology for discussing the situations specialists are faced with:**

“The elements of this language are entities called Refactoring types”;

“Each type of Refactoring describes a problem that occurs over and over again in our environment”;

“Each type of Refactoring describes the core of the solution to that “~low level” problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice!”

Replace Conditional with Polymorphism and vice versa

COMAQA.BY

1. You have a conditional that chooses different behavior depending on the type of an object.
2. Move each leg of the conditional to an overriding method in a subclass. Make the original method abstract.
3. And vice versa
4. Example

Replace Conditional with ... more sophisticated options

COMAQA.BY

1. Replace Conditional Dispatcher with Command Design Pattern

Create a Command for each action. Store the Commands in a collection and replace the conditional logic with code to fetch and execute Commands.

2. Replace Conditional Logic with Strategy Design Pattern

Create a Strategy for each variant and make the method delegate the “calculation” to a Strategy instance.

3. Replace Conditional Logic with State Design Pattern

Create a State for each variant as a part of “State Machine” and make the method delegate tricky “calculation” to the “State Machine”.

Replace Conditional with Polymorphism – detailed description

1. Problem:

You have a conditional that performs various actions depending on object type or properties.

2. Solution:

Create subclasses matching the branches of the conditional.

In them, create a shared method and move code from the corresponding branch of the conditional to it.

Replace the conditional with the relevant method call.

The result is that the proper implementation will be attained via polymorphism depending on the object class.

Why refactor

COMAQA.BY

1. **This refactoring technique can help if your code contains operators performing various tasks that vary based on:**

Class of the object or interface that it implements

Value of an object's field

Result of calling one of an object's methods

2. **If a new object property or type appears, you will need to search for and add code in all similar conditionals. Thus the benefit of this technique is multiplied if there are multiple conditionals scattered throughout all of an object's methods.**

Benefits

1. This technique adheres to the *Tell-Don't-Ask* principle: instead of asking an object about its state and then performing actions based on this, it is much easier to simply tell the object what it needs to do and let it decide for itself how to do that.

2. Removes duplicate code. You get rid of many almost identical conditionals.

3. If you need to add a new execution variant, all you need to do is add a new subclass without touching the existing code (*Open/Closed Principle*).

Preparing to Refactor

1. For this refactoring technique, you should have a ready hierarchy of classes that will contain alternative behaviors. If you do not have a hierarchy like this, create one. Other techniques will help to make this happen:

2. Replace Type Code with Subclasses. Subclasses will be created for all values of a particular object property. This approach is simple but less flexible since you cannot create subclasses for the other properties of the object.

3. Replace Type Code with State/Strategy. A class will be dedicated for a particular object property and subclasses will be created from it for each value of the property. The current class will contain references to the objects of this type and delegate execution to them.

4. The following steps assume that you have already created the hierarchy.

Refactoring Steps

COMAQA.BY

1. If the conditional is in a method that performs other actions as well, perform Extract Method.
2. For each hierarchy subclass, redefine the method that contains the conditional and copy the code of the corresponding conditional branch to that location.
3. Delete this branch from the conditional.
4. Repeat replacement until the conditional is empty. Then delete the conditional and declare the method abstract.

Regression frequency (RF)

COMAQA.BY

Definition

- ***Regression Frequency (RF) = How frequent does automated regression run?***

«Meaning»

- ***The value of product use is better as higher it is. That is ok for automated tests, if automation test runs are frequent, their importance for customer is bigger. Because of that that metric is one of the key metrics while valuing ROI.***

Regression frequency (RF)

COMAQA.BY

Boundaries

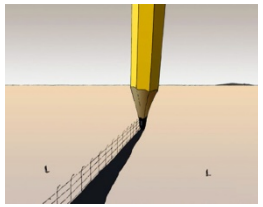
- ***Widespread boundaries / recommendations:***

smoke – every night

full-regression – every weekend

Where do we get info from

- ***Automation reports***
- ***Continuous Integration (CI)***



Regression frequency (RF)

COMAQA.BY

Examples:

- ***RF and Economical expediency of AT(ROI);***
- ***Facebook and Bamboo***
- ***HeadHunter***
- ***Kanban: RF and WarGaming experience***
- *Contra example «Absolute» «recommendations»*
  - *Contra example «Commit window»*

Regression frequency (RF)

COMAQA.BY

Visualization

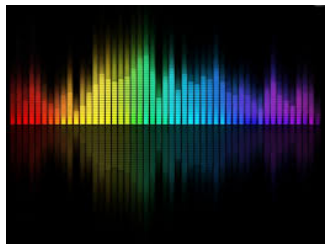
- *Not less than one a week – green color*
- *Not less than once in two weeks – yellow color*
- *Less than one a month - red color*
- *More frequent than once a day - red color*

Connection between other metrics

- *Automation testing window (ATW);*
- *Test results analysis window (TRAW);*
- *Economical expediency of AT (ROI)*
- *“Commit window“*

Category:

- *Quality*
- *Automated testing*



AT «Window»

COMAQA.BY

Definition

- ***Automated testing «Window» – how much physical time does Automated test run take (full run or subset)***

- ***Automated testing «Window» – how much system \ «lab» time does Automated test run take (full run or subset)***

AT «Window»

COMAQA.BY

«Meaning»

- ***Time, that is required to be calculated while estimating economical expediency of AUT while analyzing ROI in comparison with manual testing. Metric is required as for making decision about introduction of Automation and as for valuing current state of implemented automation with the aim of looking for narrow places.***

AT «Window»

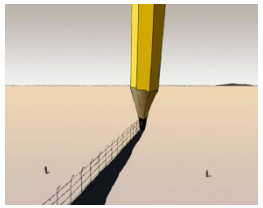
COMAQA.BY

Boundaries

- ***Depends on the size of the project, might take from couple of hours to many hours. In general, Smoke after commit should be not longer than one hour, full Regression not more than two days (weekend).***

Where do we get info from

- ***Test Reports***
- ***Continuous Integration (CI)***



AT «Window»

COMAQA.BY

Examples:

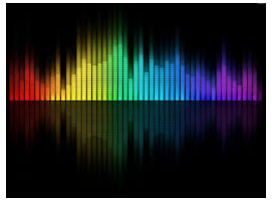
- ***Social networks (Facebook, Bamboo), CMS, CMS templates – before automation tools for vizual testing automated test cases percent was not big;***
- ***HeadHunter example***
- ***Counterexample – physical time***
- ***Counterexample – machine time (Cloud)***
- ***Technical details: Stateless and Statefull Automation, parallel run***
- ***Technical details: Effective waiters***
- ***Technical details: Premature optimization***

AT «Window»

COMAQA.BY

Visualization

- ***Smoke <= 1 hour, Full Regression <= 12 hours (night) – green color***
- ***Smoke <= 2 hours, Full Regression <= 2 days (weekend) – yellow color***
- ***Smoke > 2 hours, Full Regression > 2 days(weekend) – red color***



AT «Window»

COMAQA.BY

Connection between other metrics

- ***Automation progress (AP)***
- ***Automated tests coverage percentage***
- ***Regression Frequency (RF)***
- ***Automated tests stability (ATS)***
- ***Economical expediency of AT (ROI)***

Category:

- ***Cost / Time***
- ***Automated testing***

Test results analysis “Window” (TRAW)

COMAQA.BY

Definition

- ***Analyzing «Window» of automation test results = How much time does it take to analyze received data?***

«Meaning»

- ***Metric shows how exhaustive and readable are reports, how stabile AT and AUT. When the window is too big, less time would be devoted to tests development, or analysis will be performed not thoroughly enough, which will decrease Automation value.***

Test results analysis “Window” (TRAW)

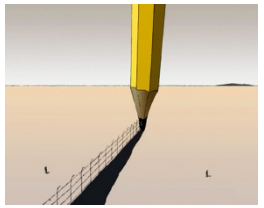
COMAQA.BY

Boundaries

- ***In dependency of the project can last from some minutes to many hours. In general, analyzing results of Smoke test after commit – should take couple of minutes, analyzing results of full Regression – should take couple of hours, ideally, less than an hour.***

Where do we get info from

- ***Test Reports***
- ***Continuous Integration (CI)***
- ***Task Tracking Systems***



Test results analysis “Window” (TRAW)

COMAQA.BY

Examples:

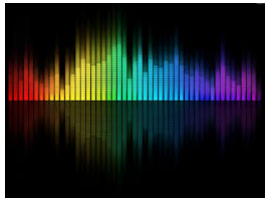
- ***Social networks (Facebook, Bamboo), CMS, CMS templates – before automation tools for vizual testing automated test cases percent was not big;***
- ***HeadHunter example***
- ***Mature Data Protection Solution, new SQL Denali plug-in, close to 100%;***
- ***Mature Secure VPN (R), technological stack;***
- ***Counterexamples;***

Test results analysis “Window” (TRAW)

COMAQA.BY

Visualization

- ***Smoke <= 10 minutes, Full Regression <= 2 hours – green color***
- ***Smoke <= 20 minutes, Full Regression <= 4 hours – yellow color***
- ***Smoke > 20 minutes, Full Regression > 4 hours – red color***



Test results analysis “Window” (TRAW)

COMAQA.BY

- **Connection between other metrics**
- ***Automation progress (AP)***
- ***Automated tests coverage Percentage (ATC)***
- ***Regression Frequency (RF)***
- ***Automated Tests stability (ATS)***

- **Category:**
- ***Cost / Time***
- ***Automated testing***

ROI

COMAQA.BY

Definition

- **Economical expedience of AT (ROI) = Manual efforts – (Automation efforts + Automation investment) / QA investment * 100%**

«Meaning»

- ***Shows does it have sense to implement automation on the current project in the current time. It might happen, that at some conditions, automation on the project can be economically inappropriate, because manual testing, even in long term future can be cheaper.***

ROI

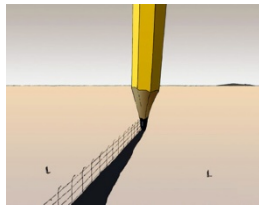
COMAQA.BY

Boundaries

- ***Out of scope***

Where do we get info from

- ***Test Strategy***
- ***Test Plan***
- ***Test Management Systems (TMS)***
- ***Task Tracking System***



ROI

COMAQA.BY

Examples:

- ***Variety of projects***
- ***Standard «problem» while working with middle+ automation specialists of «old formation»***
- ***A set of "alternative" ways of ROI usage (out of scope)***

ROI (+ additional profit)

COMAQA.BY

Visualization

- ***Comparing trends***
  - *Manual testing vs Automation*
  - *Whole variant of option of implementing / developing automation*
  - *Different investment options*
  - *Choosing optimal team-trend here and now*

ROI

COMAQA.BY

Connection between other metrics

- **% of Tests, suitable for AT**
- ***Regression frequency (RF)***
- ***Automated test creation time (ATDT)***
- ***Automated test support time (ATST)***
- ***Automated tests stability (ATS)***
- ***Automation testing window (ATW)***
- ***Test results analysis window (TRAW)***

Category:

- ***Price / time***
- ***Automation testing***

CONTACT ME

semenchenko@dpi.solutions

dpi.semenchenko

https://www.linkedin.com/in/anton-semenchenko-612a926b

https://www.facebook.com/semenchenko.anton.v

https://twitter.com/comaqa

Community's audience

Testing specialists (manual and automated)

Automation tools developers

Managers and sales specialists in IT

IT-specialists, thinking about migrating to automation

Students looking for perspective profession.

Community goals

Create unified space for effective communication for all IT-specialists in the context of automated testing.

Your profit

Ability to listen to reports from leading IT-specialists and share your experience.

Take part in «promo»-versions of top IT-conferences in CIS for free.

Meet regularly, at different forums, community «offices», social networks and messengers.

www.COMAQA.BY

COMAQA.BY

info@comaqa.by

https://www.facebook.com/comaqa.by/

http://vk.com/comaqaby

+375 33 33 46 120

+375 44 74 00 385

www.CoreHard.by

COMAQA.BY

Community's audience

«Harsh» C++ developers & co, IoT, BigData, High Load, Parallel Computing

Automation tools developers

Managers and sales specialists in IT

Students looking for perspective profession.

Community goals

Create unified space for effective communication for all IT-specialists in the context of «harsh» development.

Your profit

Ability to listen to reports from leading IT-specialists and share your experience.

Take part in «promo»-versions of top IT-conferences in CIS for free.

Meet regularly, at different forums, community «offices», social networks and messengers.

www.CoreHard.by

COMAQA.BY

info@corehard.by

https://www.facebook.com/corehard.by/

+375 33 33 46 120

+375 44 74 00 385