Data quality assurance project document

Author: Amédée Souquet

Date: 01/09/2010

Version: 0.3



1. **Glossary**

Constraint: More or less specific validation rule set on SiteCatalyst data.

Action: Interaction (mouse or keyboard) on a webpage.

Scenario: Set of actions performed on a webpage.

Extension: Program allowing to extend the functionality of another program.

SiteCatalyst: Software-as-a-service application that offers web analytics.

Google analytics: Software-as-a-service application that offers web analytics.

Wasp: Web Analytics Solution Profiler firefox extension.

Selenium: Integrated (Firefox Extension) development environment for Selenium tests.

Web beacon: Image request that uses a transparent pixel image to pass data from web pages to Omniture data center.

Description: String attached to certain SiteCatalyst parameters to give more information.

Regular expression: concise and flexible means for matching strings of text

Namespace: Identifier present in URLs allowing to distinguish customers.

UI: User interface.

UX: User experience.

XUL: XML-based User Interface language.

CVS: Control version system.

XPI: Extension for Firefox using the XPInstall API in order to get installed.

1. **Goal**

In one sentence, our goal is to ensure integrity and quality of the data

contained on our clients websites.

Our aim is to facilitate the web analytics internal testing process, automating

it and reducing redundant operations.

1. **Product overview and use cases**

Given a URL or a scenario (a set of URLs), our product will have the function of checking the SiteCatalyst content on these URLs, against the constraints defined for these URLs.  
  
Constraints are set upon a given scenario or a given visualization.

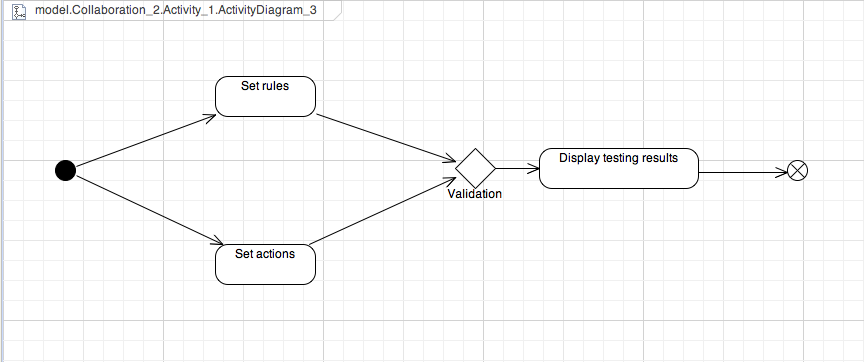
If for instance, we want to specifically test the checkout operation on a shopping website, we will have to:

1. Define the scenario (**set of actions**) corresponding to the checkout operation (for instance, select items, insert them in your cart, and click on the checkout button).
2. Define the expected parameter values and possible ranges (**set of constraints**) that SiteCatalyst should be sent via the web beacon.

As a summary, we have a scenario and database constraints as input, and the output will be the result of the test.

Our product should, given this input, specify if the test has succeeded or not,

and if not, specify why.



1. **Functional requirements**

|  |  |
| --- | --- |
|  | Feature Requirement 1.  Installation. |
| 1 | User can install the XPI via drag and drop in Firefox. [MUST] |
| 2 | User can install the XPI clicking on a link on a public webpage. [MAY] |

|  |  |
| --- | --- |
|  | Feature Requirement 2.  User can visualize current website web analytics data. |
| 1 | User can automatically see Omniture SiteCatalyst parameter values for all requests sent to the Omniture domain from a page. A specification matrix will be delivered describing what variables should be visualized and how they can be gathered [MUST] |
| 2 | User can upload the description (string) of specific SiteCatalyst variables. This is defined in the variable matrix, where also the list of visualized variables is defined. [MUST] |
| 3 | User can select a namespace, only data for this namespace will be shown. [MUST] |
| 4 | The application is always collecting data for visualization, but can be hidden. [MUST] |
| 5 | All visualization information is customer (report suite) specific (different websites can have different settings) [MUST] |
| 6 | If a current website does not have a description set, but a description is available for a different report suite, then user sees a suggestion link which will copy the settings. [TBD] |
| 7 | An history section gathers Site Catalyst requests and allows to retrieve them via report ids or  chronologic order. [MUST] |

|  |  |
| --- | --- |
|  | Feature Requirement 3.  User can validate the integrity of the data contained on our customer websites. |
| 1 | User can set constraints for every SiteCatalyst variable through the application |
| 2 | The constraints include the type of the variable (the types will be specified in the matrix) |
| 3 | The constraints may include a number range for numbers or a regular expression for all types |
| 4 | A different color is used if the constraint is met (green) compared to the constraint not being met (red) as background for the variable when visualizing the variable. |
| 5 | User can turn constraint on and off. |
| 6 | Constraints are specific for a report suite, but can be exported and imported. |
| 7 | Constraints are automatically loaded after they’ve been saved. |

|  |  |
| --- | --- |
|  | Feature Requirement 4.  User can save, import and play scenarios. |
| 1 | User can define a scenario which consist of a start URL and a list of actions (link click, form field value, refresh, button submit) |
| 2 | Scenarios can be grouped. |
| 3 | Scenarios can be imported and exported locally. |
| 4 | Definition of a scenario can take place by recording the actions taken on a website by a user. |
| 5 | For every action in a scenario a list of constraints on the SiteCatalyst variables can be defined. These constraints are specific for the action, scenario and customer website |
| 4 | It is possible to run a scenario by selecting it by a name and clicking a button run. |
| 5 | It is possible to run a group of scenarios. |
| 6 | The result of the run and the constraints (they hold or do not hold) is reported at the end of a scenario. |
| 7 | The results of a scenario is reported in XML |
| 8 | A scenario can be viewed in which case the actions are listed. |
| 9 | The sidebar visualization panel shouldn’t be automatically opened when a scenario is created or played.  Instead, only SC parameters should be saved.. |

|  |  |
| --- | --- |
|  | Feature Requirement 5.  Uninstallation. |
| 1 | User can uninstall the XPI via Tools | Add-ons menu in Firefox. [MUST] |

1. **Usability requirements**

The following table specifies what kind of interface will be used to fulfill our requirements.

|  |  |
| --- | --- |
|  | Usability requirements. |
| 1 | A sidebar will have a container (tree) allowing to visualize SC requestsmade to the current URL. |
| 2 | A button will allow to launch an interface to attach constraints to the parameters visualized in the sidebar. |
| 3 | User can select a customer via a listbox, and display parameters corresponding to this customer. |
| 4 | Have a menu allowing to import / record / save scenarios and open reports. |
| 5 | User can visualize the different report suites associated to a SC request. |
| 6 | Descriptions can be set via a textbox for fields / parameters which are classified. |
| 7 | Descriptions, types and constraints will be saved in a file referring to the current request/URL |
| 8 | User can set constraints, types, and descriptions for actions associated to a scenario. |
| 9 | Saving scenario will be done via the selenium interface (including the selenium XML extension). |
| 10 | A scenario can be played via a button, with or without any constraint attached. |
| 11 | User can open reports, saved under the XML format, for a given action of a specific scenario. |
| 12 | If a report is existing for an action of a specific scenario, an icon will appear.  Clicking on this icon will open the report (format/where TBD). |
| 13 | A tab will be used to display the different report suites available for the current SC request. |
| 12 | If a report is existing for an action of a specific scenario, an icon will appear.  Clicking on this icon will open the report (format/where TBD). |

1. **Technical requirements**

**Technical references:**

Our product will be developed on a browser-level using Mozilla technologies, implementing a similar layer to the existing Firefox extension, Wasp (see <https://addons.mozilla.org/en-US/firefox/addon/4001/> ).

The extension will be compatible with Firefox 3.0+ versions and cross-platform.

Some of the Selenium Firefox extension code might be re-used, in order to implement the scenario recorder interface and logic (see <https://addons.mozilla.org/en-US/firefox/addon/2079/> ).

We will save scenarios under an XML format, using the Selenium XML extension (see <https://addons.mozilla.org/nl/firefox/addon/106193/> ).

Multi-item packaging (merging Selenium and Wasp) is also an alternative

(see <https://developer.mozilla.org/en/multiple_item_packaging> ).

It means that instead of having 2 extensions installed independently, we install the extensions together, modify their code, to make only one product.

**Client-server architecture:**

Most of the implementation can be done client-side without the use of any server.

A server might be required at a later stage to:

* Store the Firefox extension on a “public” web page.
* Share the code on a version control system.
* Publish the testing results on a public web page.
* Give shared access to the datasources.

The SiteCatalyst requests will be retrieved via the “http-on-modify-request” event observer. Each time an HttpRequest is sent, and event is fired and the corresponding URI can be caught. From then, we will use the matrix to parse URIs and detect Site Catalyst requests.

When SiteCatalyst requests are detected, we will extract its parameters to visualize them.

A matrix, describing the list of parameters that needs to be extracted with associated characteristics, is attached to this document.

**Entities relationship and lifecycle:**

**1** URL 🡪**n** SC request (most of the times, it will be only 1 request).

**1** SC request 🡪**1** namespace.

**1** namespace 🡪**n** report suites.

**1** report suite 🡪**n** variable constraints.

**1** scenario step 🡪**n** variable constraints.

**1** scenario 🡪**n** scenario steps.

Since we are working with a limited amount of data, XML files will be used to represent above data object relationship.

**1)** Detect SC requests 🡪 Visualize SC parameters 🡪 Add constraints to the fields 🡪 Save constraints 🡪 Validate

**2)** Create scenario 🡪 Save scenario 🡪 Visualize scenario 🡪 Add constraints to steps of the scenario 🡪 Save constraints 🡪 Run scenario with contraints 🡪 Validate.

**Data design:**

Visualization of the data will be represented by a tree/container.

This data is only supposed to be read, so it won’t be stored on the file system.

Once the data visualized, some constraints can be added on this data.

Contraints will represent ranges between which data will be considered as correct, or not.

Four main modules/object will be manipulated for this project:

\* The constraint object.

\* The scenario object.

\* The report object.

\* The SiteCatalyst request object.

Visualizationconstraints XML file format:

P = prop

G = group

V = value

R = regexp/rule

D = desc

T = type

C= classified?

<?xml version="1.0"?>

<constraints id="xml\_data" xmlns="">

<cons id="1" p="c1" g="traffic" v="1" r=”” d=”” t=”” c=”true”/>

<cons id="2" p="c2" g="traffic" v="2" r=”” d=”” t=”” />

<cons id="3" p="v1" g="conversion" v="3" r=”” d=”” t=”” />

<cons id="4" p="c3" g="traffic" v="4" r=”” d=”” t=”” />

<cons id="5" p="v2" g="conversion" v="5" r=”” d=”” t=”” />

</constraints>

The file will be contained in a visualization > constraints folder.

The file name will be in correlation with current URL/SC request.

Scenario constraints XML file format:

This will be exactly the same format than above.

<?xml version="1.0"?>

<constraints id="xml\_data" xmlns="">

<cons id="1" p="c1" g="traffic" v="1" r=”” d=”” t=”” c=”true” />

<cons id="2" p="c2" g="traffic" v="2" r=”” d=”” t=”” />

<cons id="3" p="v1" g="conversion" v="3" r=”” d=”” t=”” />

<cons id="4" p="c3" g="traffic" v="4" r=”” d=”” t=”” />

<consid="5" p="v2" g="conversion" v="5" r=”” d=”” t=”” />

</constraints>

Note that only classified parameters will have the “descr” attribute in the XML file create by default when starting to edit constraint for a visualization or a scenario.

Meaning of the type attribute will be explained in the new version of the parameter matrix that Peter will send to me.

The file will be contained in a scenarios > constraints folder.

The file name will be in correlation with current URL/SC request.

Report files XML format:

<?xml version="1.0"?>

<errors id="xml\_data" xmlns="">

<error id=”1” param=”” errorCode=”” errorMessage=”” />

<error id=”2” param=”” errorCode=”” errorMessage=”” />

</errors>

Report files will be saved in a scenario OR visualization > reports folder.

The file name will be in correlation with current URL/SC request.

1. **Constraints / Risks**

The following table gathers constraints and risks, specifying their potential impact level.

|  |  |
| --- | --- |
|  | Constraints / Risks. |
| 1 | Define/maintain constraints, descriptions and types properly. [HIGH] |
| 2 | Define/maintain scenarios (set of actions) properly. [HIGH] |
| 3 | The constraints are completely dependent of the customer URLs and the website implementation. [LOW] |
| 4 | Site Catalyst implementation dependencies. [LOW] |
| 5 | Firefox implementation dependencies. [LOW] |
| 6 | Wasp implementation dependencies. [LOW] |
| 7 | Selenium implementation dependencies. [LOW] |

1. **Workflow / Planning**

The implementation of the testing framework can be decomposed functionally in four parts, as following:

1. Visualization of SiteCatalyst requests without constraints.
2. Visualization of SiteCatalyst requests with constraints.
3. Scenario manipulation without constraints (recording and saving can be done, at first, via Selenium)
4. Scenario manipulation with constraints.

Planning (see the attached Gant file):

Pre-development phase

* Requirements and need definition
* Study the Selenium extension
* Study the Wasp extension
* Design UI/UX

Phase 1

* Create XPI Makefile
* Create data extraction framework
* Create datasource Framework
* Create an extension sidebar
* Create visualization UI
* Documentation
* Testing

Phase 2

* Create constraints interface
* Create constraints validation UI
* Ability to import/export constraints
* Documentation
* Testing

Phase 3

* Ability to import/export scenarios
* Specify new scenarios
* Add ability to record a scenario (will be done via selenium for our first version)
* Add ability to play a scenario
* Documentation
* Testing

Phase 4

* Create scenario constraints interface
* Scenario validation XML reports
* Create report UI
* Create UI allowing to open reports once created
* Put XPI on public web page
* Store code on CVS

- Publish testing results

- Documentation

* Testing

1. **References/Links**

<http://rkapoor.blogspot.com/2009/04/my-take-on-web-analytics-testingquality.html>

<http://blog.immeria.net/2008/02/quality-assurance-using-wasp-tag-all.html>

<http://www.webproductblog.com/web-product-management/the-most-popular-quality-assurance-testing-mistakes/>

<https://developer.mozilla.org/en/multiple_item_packaging>

<https://addons.mozilla.org/en-US/firefox/addon/4001/>

<https://addons.mozilla.org/en-US/firefox/addon/2079/>

<https://addons.mozilla.org/nl/firefox/addon/106193/>