Are you sure you've "completed" testing? A Guardian Content API example.

16 August 2011 · automation, exploratory testing, guardian, tools, unicode

Testing doesn’t complete, it might end, it might finish, but it doesn’t complete. There’s too much to test. If you ever need confirmation of this, test something, something that’s been tested already. Better still test a piece of software, you know has been tested by someone you think is a brilliant tester. A good tester like you, will still find new issues, ambiguities and bugs.

That’s because the complexity of modern software is huge: as well as all the potential code paths of your code, there’s all the other underlying code’s paths and the near infinite domain of data it might process. Thats part of the beauty of testing, you have to be able to get a handle on this vast test space. That is, review a near infinite test-space in a [very] finite time-frame.

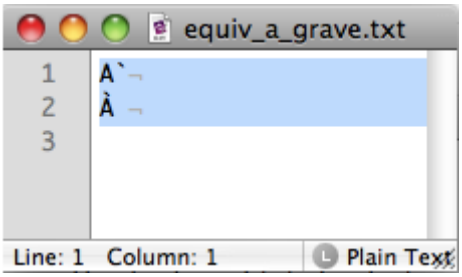
We are unable to give a complete picture of the product to our clients. But we are also free to find out new issues, that have so far eluded others. In fact the consequences are potentially more dramatic. We will always be sampling a sub-section of the potential code, data and inputs. The unexplored paths will always out number the mapped paths. As such the number of un-discovered issues is always going to be greater than the number already found. Or at least, we will not have time and/or resources to prove otherwise. As such, it’s the tests you haven’t run or even dreamed of, that are probably most significant.

As I learn, I become better equipped to see more issues in the software. My new knowledge allows me to better choose regions of the software’s behaviour to examine. I can realise questions that previously I did not even think of. Each new question opens up a new part of that near-infinite set of tests, I’ve yet to complete.

For example, I learned that some Unicode characters can have multiple representations (http://www.macchiato.com/unicode/nfc-faq#TOC-What-is-NFC-). The two representations are equivalent, but for example may utilise 2 codepoints to represent one character, rather than one codepoint representing one character. A good example would be the letter A, with a Grave accent:
À
À

Depending on your browser/OS they might look the same or different. Changing the font might help distinguish between them:
À
À

My text editor actually renders them quite differently, even though they are meant to display the same:



Until I knew about this feature of unicode; I didn't know to ask the right questions. How would the software handle this? Could it correctly treat these as equal? This whole area of testing would not of been examined, if I hadn't taken the time to learn about this 'Canonical Equivalence' property of Unicode normalisation.

This is a situation when I would actively avoid using most test automation, until I was clear as to the my understanding of the potential issues. Therefore I stopped using my previous scripts, and used cURL (http://curl.haxx.se/). The benefit of cURL is that it gives me direct and visible control of what I request from a site/API. It will make the exact request I ask of it, with very little fuss and certainly no frills. I can be sure its not going to try and encode or interpret what I'm requesting, but rather, repeat it verbatim.

This example had an interesting result when used against the Guardian Content API (http://www.guardian.co.uk/open-platform). My first tests included this query to the Guardian's Content Search:

The non-combining query, including the letter À (Capital A with a grave accent or %C3%80 in a single character):

Query:
http://content.guardianapis.com/search?q=%C3%80lex&format=json (http://content.guardianapis.com/search?q=%C3%80lex&format=json)

Response:

```
{
  "response":{
    "status":"ok",
    "userTier":"free",
    "total":0,
    "startIndex":0,
    "pageSize":10,
    "currentPage":1,
    "pages":0,
```

“orderBy”:”newest”,
“didYouMean”:”alex”,
“results”:[]
}
}

...and the query with the combining characters, consisting of the regular ‘A’ and a separate grave accent (%CC%80):

Query:
http://content.guardianapis.com/search?q=A%CC%80lex&format=json (http://content.guardianapis.com/search?q=A%CC%80lex&format=json)

Response:
{
“response”:{
“status”:”ok”,
“userTier”:”free”,
“total”:0,
“startIndex”:0,
“pageSize”:10,
“currentPage”:1,
“pages”:0,
“orderBy”:”newest”,
“results”:[]
}
}

At first glance these two results look fairly similar, but a closer look shows the first response includes a didYouMean field. In theory these two queries should be treated equivalently. This minor difference suggests they were not being treated so, but this was also a fairly minor issue. As a tester I knew I had to examine this further, find out how big/bad could this difference be?

Rather than slip back into automation, I realised that what I needed was an example that demonstrates the potential magnitude of the issue. This was a human problem, or opportunity, I needed an example that would clearly diagnose an issue in one representation of the characters and not in the other. So I needed a query that could be affected by these differences and if interpreted correctly, deliver many news results. The answer was Société Générale (http://en.wikipedia.org/wiki/Soci%C3%A9t%C3%A9_G%C3%A9n%C3%A9rale) a high profile and recent news story, with a non-ASCII, accented company name.

The non-combining query, using a single codepoint to represent the accented ‘e’:
Query:

http://content.guardianapis.com/search?q=Soci%c3%a9t%c3%a9+G%c3%a9n%c3%a9rale&format=json (http://content.guardianapis.com/search?q=Soci%c3%a9t%c3%a9+G%c3%a9n%c3%a9rale&format=json)

Response (partial):

{
“response”:{
“status”:”ok”,
“userTier”:”free”,
“total”:**536**,
“startIndex”:1,
“pageSize”:10,
“currentPage”:1,
“pages”:54,
“orderBy”:”newest”,
“results”:[{
“id”:”business/2011/aug/14/economic-burden-debt-crisis-euro”,
“sectionId”:”business”,
“sectionName”:”Business”,
“webPublicationDate”:”2011-08-14T00:06:13+01:00”,
“webTitle”:”The financial burden of the debt crisis could lead countries to opt out of the euro”,
“webUrl”:”http://www.guardian.co.uk/business/2011/aug/14/economic-burden-debt-crisis-euro”,
“apiUrl”:”http://content.guardianapis.com/business/2011/aug/14/economic-burden-debt-crisis-euro”
},{
…

As you can see there are over 500 results with this query.

The combining query, using a two codepoints to represent the accented ‘e’:

Query:

http://content.guardianapis.com/search?q=Socie%cc%81te%cc%81+Ge%cc%81ne%cc%81rale&format=json (http://content.guardianapis.com/search?q=Socie%cc%81te%cc%81+Ge%cc%81ne%cc%81rale&format=json)

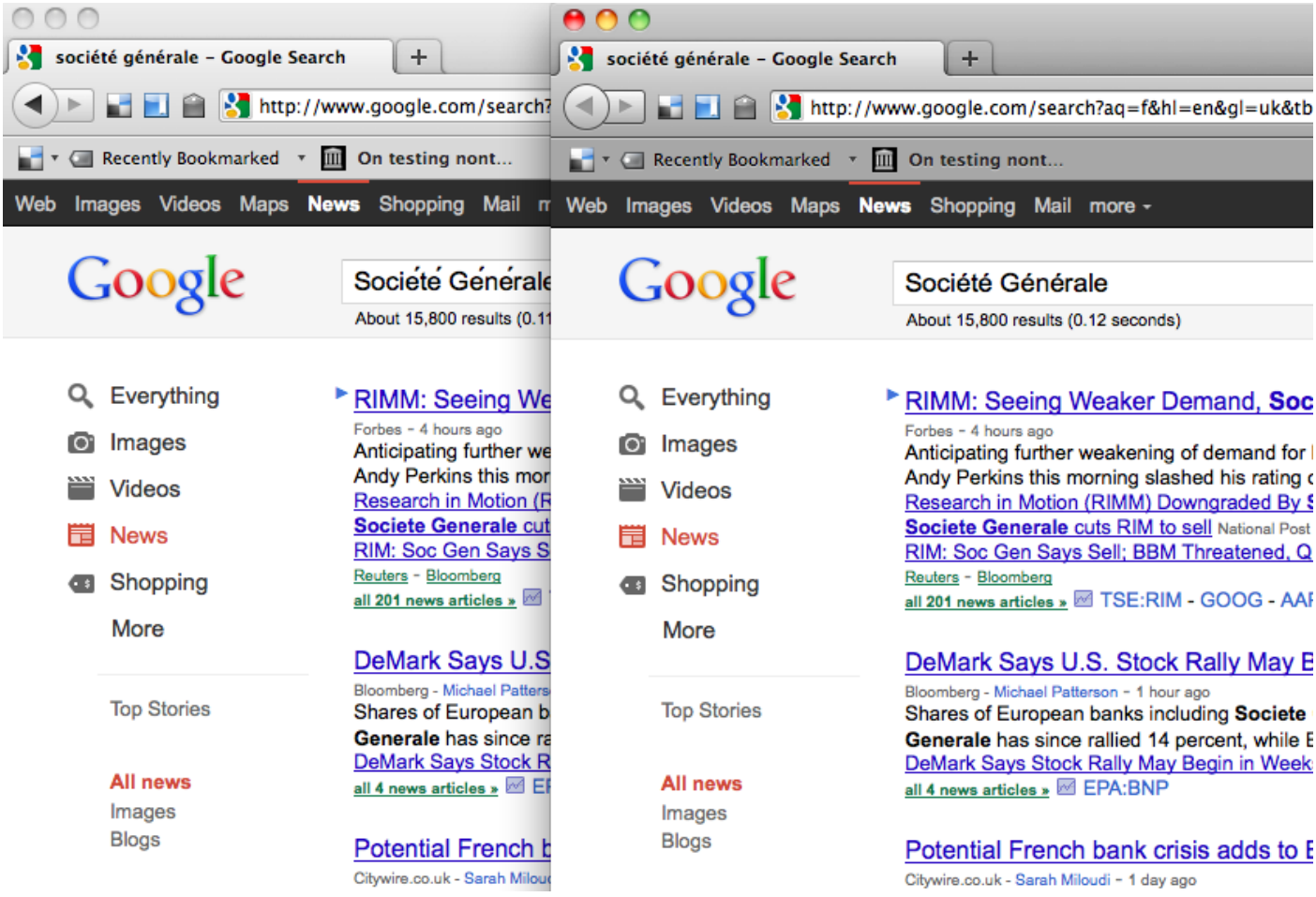
Response:

{
“response”:{
“status”:”ok”,
“userTier”:”free”,

```
    “total”:0,
    “startIndex”:0,
    “pageSize”:10,
    “currentPage”:1,
    “pages”:0,
    “orderBy”:”newest”,
    “didYouMean”:”sofiété Générace”,
    “results”:[]
  }
}
```

This response shows that the query found 0 results, and suggested something else.

At this point it looked like there was an issue. But how could I be sure? maybe the Unicode NFC behaviour was purely hypothetical, not used in reality. So I needed an oracle, something that would help me decide if this behaviour was a bug. I switched to another news search system, one that generally seems reliable and would be respected in a comparison: Google News.



FireFox 5 (Mac OS X) Renders these characters differently, but google returns the same results.

Note: Google Chrome renders no discernable difference.

I used cURL to make two queries to the Google News site, using the two different queries. This requires a minor tweak, to modify the user-agent of cURL, to stop it being blocked by Google. The results showed that google returned almost the same results, for both versions of “Société Générale”. There were some minor differences, but these appeared to be inconsistent, possibly unrelated. The significant feedback from these google news pages was that Google returns many results for both forms of character representation, and those results are virtually identical. It would therefore appear that there is an issue with the Guardians handling of these codepoints.

Thanks to this investigation, we have learned of another possible limitation in the Guardian Search API. A limitation that could mean a user would not find news related to an important and current news event. This kind of investigation is at the heart of good testing, results learned from testing are quickly analysed, compared with background knowledge and used to generate more and better tests. Tools are selected for their ability to support this process, increasing the clarity of our results, without forcing us to write unneeded code, in awkward DSLs.