Your software sucks (any data you give it)

21 December 2015 · breaking things, testing, tools, tricks, unicode

At 1524h, On the afternoon of January 15th 2009, US Airways Flight 1549 was cleared for takeoff from Runway 4 at New York's La Guardia airport. The airplane carried 150 passengers and 5 flight crew, on a flight to Charlotte Douglas, North Carolina. The Airbus A320's twin CFM56 engines had been serviced just over a month prior to the flight. The plane climbed to a height of 859m (2818 feet) before disaster struck.

Passengers reported hearing several loud bangs and then flames being visible from the engines' exhaust. Shortly thereafter the 2 engines shut-down, robbing the Airbus of thrust and its primary source of electrical power.

At this point the Captain took over from the First officer and between them they spent the next 3 minutes both looking for somewhere to land, while also desperately trying to restart their aircraft's engines.

What Happened?

A flock of birds had crossed the path of the Airbus and several had struck the plane. Both engines had ingested birds and shut-down as a result. A shut-down is the FAA required minimum standard behaviour for a Jet engine.

The safe automatic shut-down of a jet engine is a scenario tested for by engine manufacturers before they can be certified for use.

Worse things might happen, e.g. the broken unbalanced blades might continue blowing air into the fuel rich combustion chamber while red-hot engine fragments are jettisoned outwards into other parts of the fuel-laden plane.

Viewed in that light: a graceful shut-down is not a bad minimum safety requirement.

If we think about it, jet engines need a good deal of testing, after-all they

- Are mission critical.
- Work faster than humans can think and react.
- Are expensive and time consuming to build.
- Have to be integrated with other complex systems
- Have to accept un-validated inputs (like birds)

Does any of that sound familiar? That last one in particular is relevant to the field of software development and testing.

Un-validated input? How do they test that?

One of the tests that can be performed on a Jet Engine is to fire frozen poultry into the engine. The engine ingests a turkey at high speed, in an attempt to simulate a bird being sucked into the engine during flight.

Like many technical systems that deal directly with the outside world, software can have serious problems when exposed to unusual inputs. Like the Jet, the point of ingest literally can not be protected - something has to ‘process’ what’s coming in.

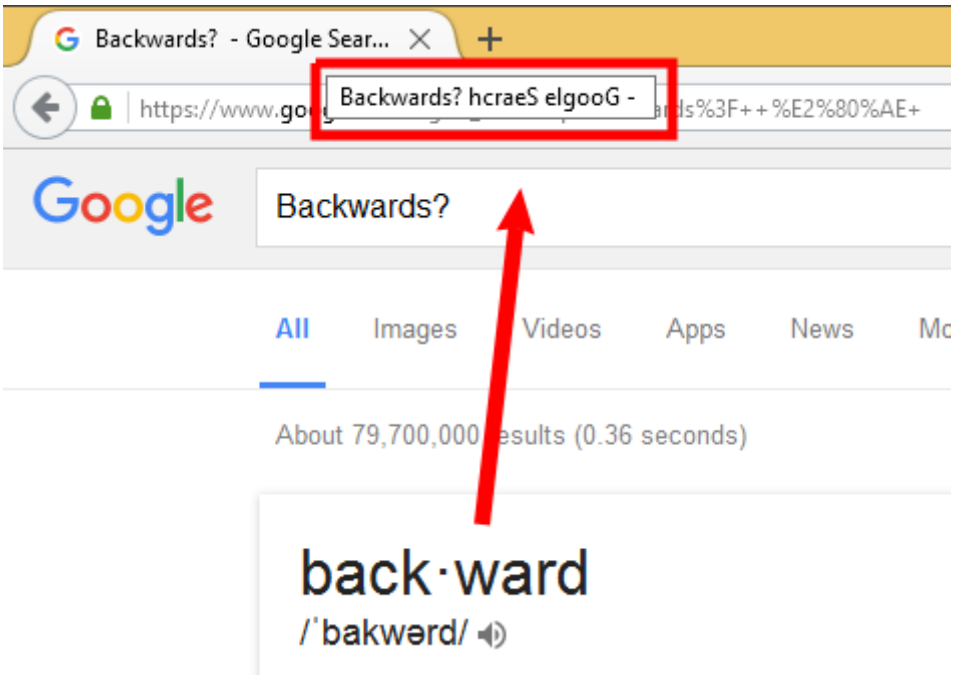
As software testers working with applications that need to handle these situations, we need to learn how to perform our own frozen-turkey-tests and examine how our complex systems handle them.

- Do they crash?
- If they crash, is that OK?
- What have I learned?
- What were the side effects?
- Can I restart it? or is it now ‘corrupted’ ?
- What is the likelihood of failure?

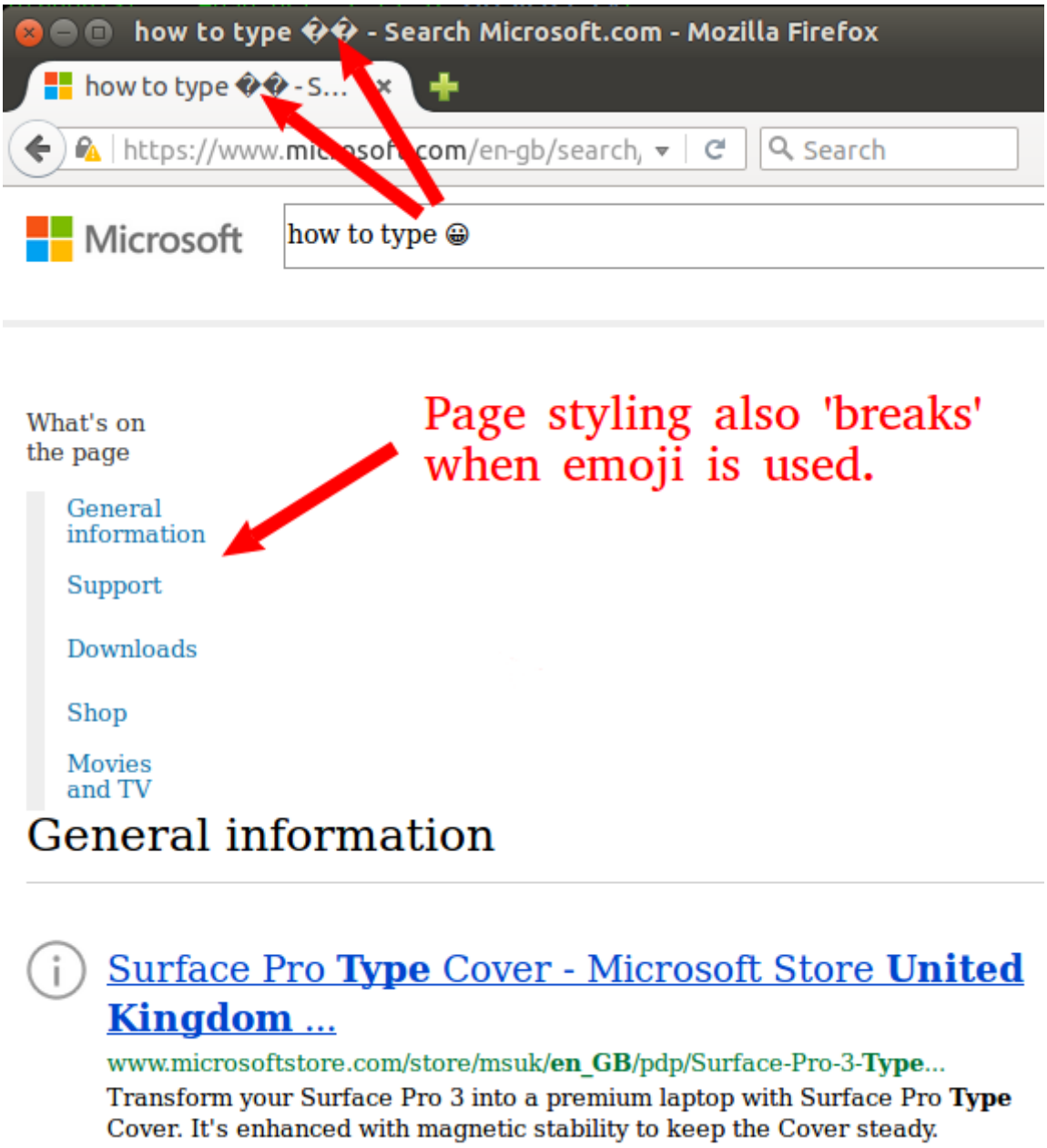
The sort of websites we use every day have to accept largely un-validated inputs, they are on the internet and anything our computer can send - they have to deal with.

But surely its just text, right?
If its not ‘normal’ block it!

That isn’t going to work for long… For example Google has to handle anything you want to find on any website. Even if you accidentally include some right-to-left data in your search (https://www.google.com/?gws_rd=cr#q=Backwards%3F++%E2%80%AE+):



...Or you want to find out how to do that cool Emoticon on your new Microsoft Surface notebook keyboard... Microsoft.com then needs to handle that query (https://www.microsoft.com/en-gb/newsearch/result.aspx?q=How%20to%20type%20%F0%9F%98%80).

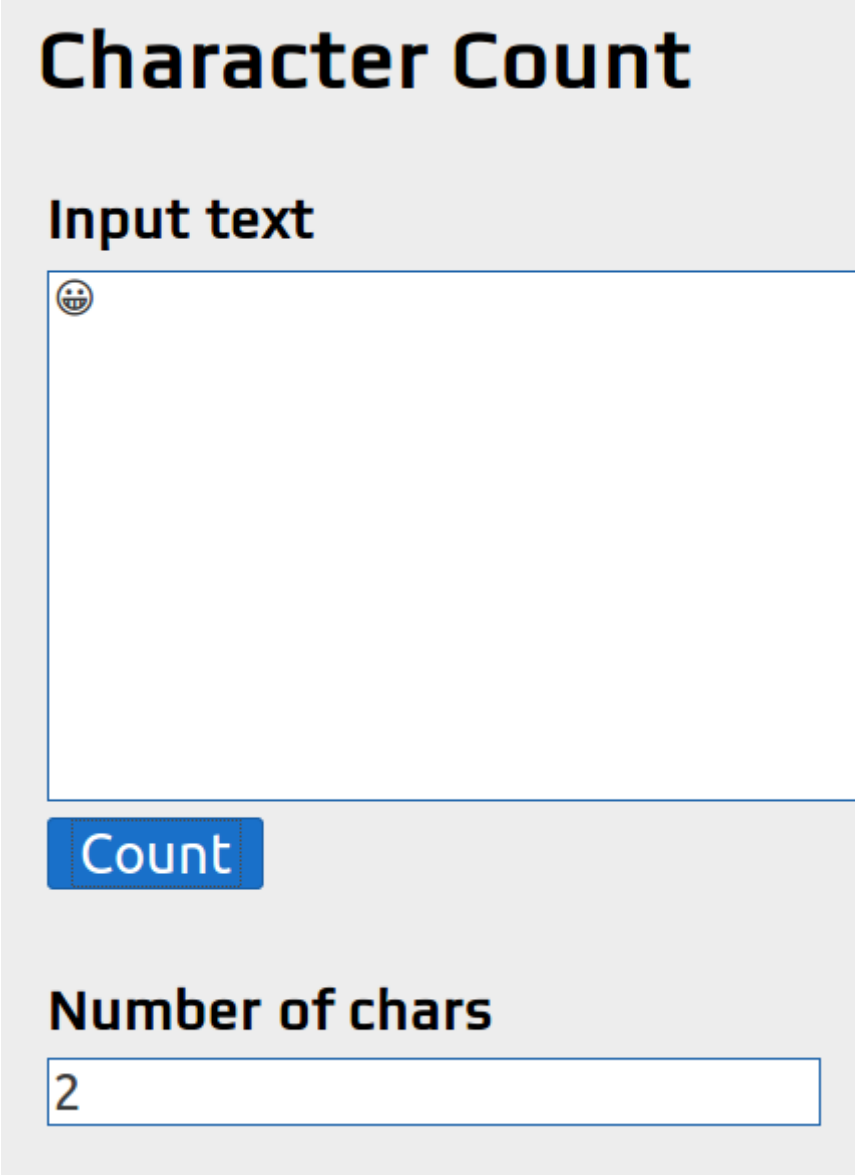


...Or you don't want to pay extra on your phone bill just because you used a smiley (http://www.bbc.co.uk/news/technology-31148424) face in your text message.

These are real world examples of things people use their software for, every day. Hence they are the sort of things we need to test for, lest our users end up going elsewhere or find they are being over charged.

Tools such as No More ASCII (https://addons.mozilla.org/firefox/addon/no-more-ascii-text-inserter/) can help us test websites, by giving us direct access to a range of Unicode 'code-points' that may cause problems for our software.

The problems can be subtle, more than just something ‘not looking right’. The complex nature in which languages are represented in your application can mean that simple things such as measuring the length of a string can fail. (string.online-toolz.com (http://string.online-toolz.com/))

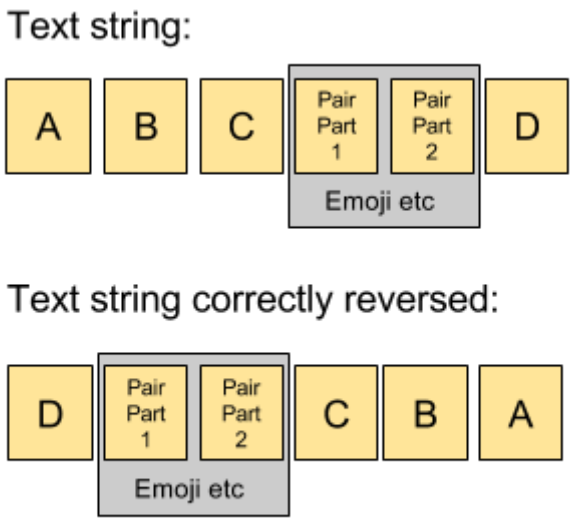


Sorting can also fail. If your text is reversed for example if may not render correctly afterwords:

These 2 issues are caused by the website not being able to properly process Unicode text, in particular the UTF-16 flavour of Unicode. Some characters (or Graphemes as they are called in

Unicode) are in fact made-up of 2 parts or ‘code-points’. So whilst many characters tend to be 1 code-point, some are pairs. These pairs are referred to as ‘surrogate pairs’.

Why does the reverse-string function fail? It appears to be putting the emoticons 2 code-points in reverse order, when it shouldn’t. They should be treated together as one character when a reverse or sort is performed. (When the individual code-points in a surrogate pair are swapped, they become meaningless).



How to reverse a UTF-16 text string with a Surrogate Pair in it.

These ‘surrogate pairs’ cover things like Emoticons (http://www.unicode.org/charts/PDF/U1F600.pdf) or Musical notation (http://unicode.org/charts/PDF/U1D100.pdf) etc. While not used widely on computers in North America in the 1960s, therefore not in ASCII (https://en.wikipedia.org/wiki/ASCII), they are now widely used all around the globe.

Un-validated text input is great example of where tools-assisted-testing can discover a wealth of knowledge about our applications. Given the wide domain of possible inputs and unknown-complexity of the app, this is an inherently exploratory process. Have the right tools on-hand helps you gain that knowledge quicker.

You can read more about how to explore how your browser/app handles Unicode (http://www.investigatingsoftware.co.uk/2012/10/simple-test-automation-with-no-moving.html).