As near as damn it.

15 November 2019 · automation, bug, precision, python, squish

It’s 1982 and there’s a bull market in the western stock exchanges. After being in the doldrums for 6 years the Dow Jones Industrial Average index is climbing steeply. In London, the FTSE 100 index is also witnessing a steady climb, despite the ongoing war in the South Atlantic. The rise in share prices also leads to an increase in the popularity and prominence of these stock ‘indices’, the algorithmically derived snapshots of leading stock prices, frequently, used as an indicator of overall market health.

At that time a relatively small North American exchange decides to institute its own new index, allowing investors to discern, at a glance, the state of the market. The Vancouver Stock Exchange creates its index at an arbitrary 1000.0 starting value. The index value is then recalculated thousands of times a day as transactions are processed through the exchange.

Fast forward to late 1983, western markets have continued to boom, stimulating sustained increases in their share index values. For example between August 27, 1982, and August 5, 1983, the Dow Jones has risen over 33% (from 883.47 to 1183.29). Though the Vancouver Stock Exchange index was having no such luck. By November 1983 it had dropped to just over 500, approximately half its initial value of 1000.

(https://previews.123rf.com/images/moravia/moravia1510/moravia151000039/47485887-market-crash-chart.jpg)

Investigators uncovered the cause, a bug in the algorithm used to update the share index. The bug was skewing the result slightly each time it was calculated. Interestingly the algorithm was not calculating the index anew from the raw data after each transaction:

> Instead a “clever” analyst decided it would be more efficient to recompute the index by adding the net change of a stock after each trade. This computation was done using four decimal places and truncating (not rounding) the result to three.

https://introcs.cs.princeton.edu/java/91float/ (https://introcs.cs.princeton.edu/java/91float/)

This combination of an error-prone algorithm, poor precision and not using a sensible rounding method meant the gap between the correct index value and the reported value diverged quickly.

While modern programming languages have features that allow you to handle issues such as rounding and precision, many teams are not using them or don’t feel they need them in their

project. That's understandable, but its a risk. If that risk is discussed and accepted you at least know what you are doing and hopefully can handle the impact if any of the consequences arise.

But many teams are running full steam ahead and are oblivious to the risks. As a tester or SDET, it's your job to highlight that risk and make sure those judgment calls are made.

Precision bugs can be messy and are often caused inadvertently across system/API boundaries. I once worked on a team where we took great effort to increase the precision of our backend systems to match the limits of our database and allow business logic calculations to yield suitably precise answers. In fact, we made it so precise that it was more 'precise' than our front end could handle.

The backend and front-end communicated via JSON, and JSON numbers are by default, Doubles. Therefore even though we were calculating and serving numbers at greater than double-precision, they would be displayed and saved back to the database as 'doubles'. This is because the Javascript front end code would parse the numbers into doubles, sacrificing the greater 'detail' we had served from the back-end. Luckily I caught that issue in testing before we released, but it serves as a reminder to me of the value of precision testing and integration testing.

As for your automated tests, you'll need to ensure that you work with the expected level of precision. And by expected, I guess I mean whatever your team has deemed as acceptable risk mitigation given the data you work with. It's the sort of conversation that should involve your product owner.

Note, your programming language's default precision is often not that used by the applications you are testing. For example, this simple Squish Python script drives the standard Windows Calculator (https://github.com/phoughton/squish_decimals) and even here I need to adjust to the app's own precision configuration.

It's definitely worth investigating the rounding and precision of your applications and check you are doing it right.

Further Reading: A list of related real-world failures: https://web.ma.utexas.edu/users/arbogast/misc/disasters.html (https://web.ma.utexas.edu/users/arbogast/misc/disasters.html)

A thought experiment: Do you think they would have noticed the issue if the index value had been increasing by mistake, instead of decreasing?