Synecdoche

02 August 2016 · testing

A common but often unnoticed figure of speech is the synecdoche. When I say “Beijing opened its borders”. We know I mean “The People’s Republic of China has opened its borders.”) That’s a Synecdoche, in this case I named part of something (Beijing) to mean the whole (P.R.C.).

Conversely, I might say “Westminster is in turmoil” when anyone with knowledge of British politics will know I mean, “The politicians in the Houses of Parliament are in turmoil”. The reader will know I am not referring to The City of Westminster, a region of London. (Or the place in Canada etc.)

Synecdoche can be a useful and illustrating tool of conversation. Helping to convey the size or importance of the subject or illustrate in more detail a subtlety of the situation. For example: “Beijing opened its Borders” also indicates the power of that country’s central government. Some residents of one city in China, can open [or close] the borders of a vast country spanning thousands of miles and comprising over 1.3 billion people.

Synecdoche can also lead to ambiguity, and are particularly dependant on context. For example the same phrase “Westminster is in turmoil” accompanied by a picture of a de-railed train, smoke and ambulances would lead the reader to assume the geographic region of Westminster was being referred to.

Just this sort of language and potential for confusion exists within software development. For example, a Product Owner might ask a team to code a feature for her App. A technical lead would likely know her team will actually: analyse, converse, script, code, test, fix, report, document, review etc. And probably do this across multiple systems before she can agree with the Product Owner that the App’s feature is complete or ‘coded’.

Why don’t technical leads get annoyed by this narrow description of the work? Well, actually they do, all the time. When working as a Scrum Master and Program Manager I frequently had to smooth these sorts of negotiations. Often a technical lead or test lead would take the product owners choice of the word (e.g.: “code” or “develop”) to mean that the work required was not significant. When the Product Owner’s words could have been translated as “do clever stuff to make it happen”.

Product Owners were often not from a programming or testing background. Occasionally they would not use the same jargon as developers or, more often, they used the same terms but with their own meanings. For example, using ‘code’ to mean the whole software development and release process.

While some friction would be caused in circumstances where someone might use the wrong or, to the team, misleading jargon, the team usually adapted. The team might use the jargon between themselves, but then adopt a less ‘technical’ (their words) language style when talking to others. That is, people outside the core team.

Testing also has situations where we frequently say one thing, and rely on context to mean so much more or less. ‘Test automation’ for example. This simple term can covers a range of tools, techniques and even approaches.

In my experience, ‘Test automation’ has for example referred to test data generators or shell scripts. These would check data-outputs were within a valid range, given data-inputs of historical purchases. I have also worked with successful teams where the term test automation meant random input generators combined with a simple run-until-crash check.

Furthermore, I have worked on systems where ‘Test Automation’ results could be red / green / pass / fail style messages reported from a GUI or API based test tool. In another team our results could only have been usefully discerned with the aid of graphing software. On some projects the skilled expertise of a statistician was required to decide whether our test code had uncovered an issue. On occasion, the term ‘Test Automation’ could mean several or all of the above.

When talking with my team, I need to be more specific. I, like them, have to be able to describe what I’m doing and why. I could just say “I’m doing test automation” but that would be like a developer stating “I’m doing feature X”. Having a precise way to describe my work, and how it relates to the work of my team members is valuable and time saving. Not just in the time spent not re-explaining and clarifying concepts. But more importantly, not having to re-do things we thought were complete or correct the first time.

Having the words to describe in detail our work is invaluable. The sorts of things we talk about within a team are jargon heavy E.g.: I need to explain to my team that I’m coding a check for the products UTF-16 surrogate pair handling, to be added to the Continuous Integration process, this might mean we don’t complete a feature this sprint. I may need to clarify that I’m writing a script to be used as an oracle - to aid our User Interface testing, or ask the programmers to include a testability hook to aid our log file analysis.

The language used to communicate these ideas is important. The language and terms themselves are worthy of at least some discussion. If we as a team are unfamiliar with the terms. Or their differing contextual meanings, we will likely end up very confidently and quietly not knowing what we are doing all day.