Text to SWIFT - making data from prose (What possible use could Gen AI be to me? - Part 2)

06 August 2024 · ai, automation, iso, iso20022, MT, MX, qa, sql, swift, testing

As I write this, my dog is grumpily moving around the room pausing intermittently to give me disappointed looks - looks that only my elderly mother could compete with. She (my dog) is annoyed by the robot vacuum cleaner. Its not been run for a while in that room - and its making a noisy foray into dark corners in a valiant effort to cleanse the mess. Its grinding gears and the cloud of dust in its wake is not helping to ease the dogs nerves.

The dog's pleading puppy dog eyes & emotions have of course been anthropomorphised - at least a bit - by me (My dog is 7 years old and weighs over 20kg - so has little to fear). That is - I've taken human feelings and mapped them onto my dog. I know she has emotions - but she lacks language - or at least a language that (1) we humans understand, (2) maps to the same phrases or concepts I'm using. But I'm human, That's how I think and how I interact with people and sometimes - machines.

Deciphering the problem and representing that in code and tests is a large part of software development.

Similarly when I give her a cheaper & less tasty brand of dog food - she (hopefully) isn't making highly critical judgements about my recent life choices and desire to economise.

“Pete, maybe if you worked a little harder some of us wouldn’t have to eat like we are… well… animals”.

As with dogs, when we work with machines we often like to think they are working or can work (read: think and talk) like us. This is particularly true in Software Engineering, where we go through cycles of trying to represent human concepts in machines. Like time, a very human idea, is typically handled in ways that can exhibit problems e.g. The Y2K bug.

But lets go further, more abstract. We are usually trying to solve a problem in software. That problem might be described as a new app or a fix to an old one - but how do we describe it? how do we describe the data it uses? Will the data it reads - be in the format it needs - will it understand the language of the data?

If we take Cucumber Features & Scenarios (often used in Software Testing and/or Behaviour Driven Development). We often find our selves trying to describe complex real world rules and behaviours in fairly rigid framework of Given, When & Then. Or describing our complex hierarchical data in a data-table consisting of 2 simple columns. That is - just like my dog and me - we don’t have a good way to communicate - especially as one of the parties lacks human understanding of the problem.

Text to the rescue!

The written word or ‘prose’ isn’t perfect, but its often easier to describe what we want to do, want to test, the data we care about at a high level - in plain English/Spanish etc. (While the low level steps to do that - have traditionally required code) And that’s where things stood for quite a while, we could write and describe what we wanted - but to translate that into Software or Data that we wanted - took a lot of effort. Effort that took Engineers, BAs, Project Managers etc.

The rise of Large Language Models (like ChatGPT, Claude, Gemini, Llama etc) have not done away with those engineers, and other specialists - but given us new tools to smooth that process. Take the area I work a lot in, Banking & Payments. The market is dominated by large players, is highly regulated and has a mixture of tech dating from the 1960s to now. (And yes before you ask - there is still a some COBOL around - running in production)

The problems we solve are very human. E.g. How can I give money (a made up human concept) to a company on the far side of the planet (who are currently asleep), in a different currency than I use - and comply with the appropriate rules around money transfers in both our jurisdictions - and ensure the money does not go missing en-route. And did I mention my bank and their bank have no direct business relationship?

Take something like a street address. Addresses maybe fairly standard in somewhere like London - where I live. But what about in your country? What about the addresses of my

payment above? Large language models can help with these types of problems. Take this extract from a Payment Initiation (PAIN XML) message:

Mr Sherlock Holmes

Baker Street

221c

NW1 5RT

London

UK

This and the other required information could be represented in plain English text, and converted to a an XML document using a large language model. In fact that's how this was created, using this prompt to a specially setup LLM, using this:

Create a PAIN.001001.08 message where the creditor's name was Mr Sherlock Holmes, and his address is 221c Baker Street, London, NW1 5RT, UK. The initiating party is Holmes Investigations Inc. and they have the ID "GB123H7"…

See how the model has correctly parsed the text and populated the fields intelligently - like Building Number (BldgNb). While that specific example would not be difficult to do in code, it would be difficult to do that for all the variations of address that exist in various global locations. (Also note the AI model did not try and correct my deliberately wrong address for Sherlock - his stated address in the novels was 221**b** not c)

Take another example like this Spanish address - note the building number spans a number of houses/buildings and is at the end of the street - and not at the start:

Create a payment from Jose Smith at Avenida de Andalucía, 5 y 7. Málaga to…

Is translated without change to the model or prompt to:

Jose Smith

Avenida de Andalucía

5 y 7

Málaga

ES

I’ve used the address here, but we can produce a full XML message from plain English. For example, the other fields like remittance info were populated similarly:

The debtor is Winston Churchill, 10 Downing Street, London, W1 7UI. The purpose of the payment is to buy Investigative Services.

Payment for Investigative Services

Translating ideas to data - Text to Swift

So now we have technology that can go from text to data, or text to XML in this specific case. Or to put it another way - Text to SWIFT. We can take unstructured prose and deliver machine readable data - usable by back office payment systems.

For Software Test Engineering & automation this sort of capability is very useful, we can replace layers of abstraction, rules, step definitions and regular expression code - with a model that can turn our high level tests into the underlying data we need.

Whether we use the LLM directly in our automation frameworks - or as an assistant - helping generate or check data for us - either way (probably both) having a system that can simplify test automation code - and replace it with something trained and prompted with examples could provide great benefits.

Why? Typically this sort of complex cross system / End to End development & testing uses examples. Each example represents a scenario that might trip-up or activate one of the dozens (hundreds in some cases) systems that are involved in a complex feature. Those examples should be the hard part - and they are. They require knowledge of Payments (so you need a subject matter expert) the processes of the companies involved (possibly another SME or business analyst) the tech involved end to end (Technical SME) and a Test or Quality Engineer / Architect. Depending on your companies’ size and structure that could constitute 1 or up to a dozen or even more people.

That’s just the ‘What’, then you have the ‘How’ - How do I create the data for all those examples? How do I query the systems involved? How do make this tech reusable, fast - and able to talk to the dozens of systems involved (yes including those 1960s mainframes and your latest SaaS cloud system). How do I report on all those checks - across all those systems? - how do I do that well and quickly…

While my previous post touched on how AI tools can help (https://www.investigatingsoftware.co.uk/2024/02/what-possible-use-could-gen-ai-be-to-me.html) with domain knowledge by providing a way to talk to your documents, manuals and regulatory material. Here the data provided by your SMEs can be directly used to instruct an AI / LLM tool to generate more examples to help fully test your system. The Large Language Models are readily able to understand and parse plain English (or Spanish etc) descriptions in a way that can be used directly without tools like

Cucumber. This in turn helps to reduce the size and scope of that engineering step where we build custom tools for every type of test, system interaction or query.

Sceptical? Just take a look at Text to SQL, a popular area of model development (https://aws.amazon.com/blogs/machine-learning/build-a-robust-text-to-sql-solution-generating-complex-queries-self-correcting-and-querying-diverse-data-sources/) right now. Why rely on rigid pre-defined queries, when you can build custom queries from plain English and examples?

While these technologies don’t do away with traditional engineering roles, they do change the nature of the work and can help close the gap between domain knowledge and technical knowledge - while possibly reducing the long term engineering overhead.