Validators as coding agent specifications

05 March 2026 · ai, automation, banking, deep learning, human, payments

I was raised in a world full of rules, like all of us, though as I grew up in a military family in and around military bases - so maybe a bit more so for me. As a parent myself, rules are at play - “no son you can’t bang our wooden table with an old hammer”. Even the business world is full of rules, don’t agree? Speak to your bank’s payments or compliance department or your lawyer or accountant and get back to me.

We soak up all sorts of rules like sponges, implicitly. Like my toddler son when he found my old hammer, “no the rule is I can’t bang the table with the hammer - when you are in the room” - while he has reverse engineered the rule and context perfectly - Though I feel he didn’t pick up on the full spirit of the rule.

The late author David Graeber had a talent for analysing rules in societies old and new. He used “opposites” as a way to elucidate the rules and assumptions at play in fiction and in society. A good example he used was, the vampire, the former workhorse of American teen fiction. What’s the opposite of a vampire? his answer intrigued me, it was Werewolf. His reasoning was - they both have some similarities, they come out at night, they ‘spread’ via biting etc -and of course they are both mythical.

OMFG Double sided compliance checklists!

But, he argued, they are otherwise -opposites. e. g. Vampires are usually rich or at least aristocrats. While werewolves are typically portrayed as poor or itinerant. Vampires are limited by their need to return to their crypt, while werewolves roam the forests & countryside.

Vampires seek to control or manipulate others - while possessing supreme self control and agility. Meanwhile werewolves are barely able to control their own propensity for violence let alone the emotions of others.

One more - For those of us that love this kind of thinking: Ask yourself “what kills a vampire?” - yes a wooden stake to the heart, but what kills a werewolf? - a silver bullet (Two very different ends of the home defence weaponry market!).

LLMs, trained as they are, on lots of (dubiously obtained), text and films etc, are ok at this sort of analysis. but they are not ‘great’ at it - Partly no doubt due to their total lack of imagination and tendency towards the mediocre norm.

But nonetheless, they can be prompted to reliably ‘break’ a rule defined in code or something code-like (think custom DSL). Though “Break” isn’t the right word, “trigger” is better - as what I am aiming to do is cause a validation rule to become activated- rather than somehow break its behaviour.

What I’ve found is that LLMs, in development tools like Claude Code or GitHub CoPilot Agent can reliably trigger the rules. So I provide it a good and valid piece of test data & a rule - then it can provide a version of that data to trigger that one rule.

Why do I care? just as looking for opposites helps use figure how our world and society is built on assumptions and rules - creating ‘triggering’ test data can clarify how well our software works.

Example

Lets take a concrete , if trivial example: Remember password complexity? Like: your password has to have a mixture of uppercase and lowercase letters and symbols, four letters from Sanskrit etc. Who am I kidding? - If you in any part of your life deal with a big company you are probably still suffering with these.

These are just another set of rules combined with a validator - designed to ensure you create a password that’s easy to break and hard to remember ;-)

To demonstrate, I setup a validator, and configure it to use a selection of awkward rules and plug that into an MCP server (to more easily integrate with the coding agent).

Some of the rules are minimum length = 6, minimum count of numbers = 2 , minimum number of symbols = 1, etc. These validator will only moan if the rule is ‘triggered’; if for example there are not enough upper case characters. When that happens a helpful message will be returned saying words to that effect.

I get the development agent i.e. Claude Code or GitHub CoPilot Agent to try and give me a phrase that can trigger a single rule, in fact I request one file each for each rule. So that’s - one file with input text too short, another with too few symbols / special characters, another for not enough upper case letters etc.

I get the Agent / LLM to provide the test data one rule at a time. So each rule is paired with a piece of test data that triggers it. This can be a useful way to generate test data.

The problem you soon hit is - unlike David Graeber - LLMs are as dense as a sack of hammers. Yes - they utilise Language skills to translate our rule into triggering data - and Yes - they can often guess the right answer in one go - but what if the test data it generates is overzealous and triggers 2 or more rules? On their own - they just won’t know - unless I tell them all the rules in advance, but, in fact they will sometimes ignore a long list of rules if you do provide it.

This is where the ‘agent mode’ (essentially a loop that checks for success) comes in useful, as long as there’s useful feedback from the validator, our coding agent can adapt its answer iteratively (the looping) to find an answer. So even if the naïve solution triggers 2 or 3 rules, the agent will interpret the validation response and make another attempt.

This is useful, we can auto-magically respond to rules as defined and create example files that allow us to trigger an individual rule. But what if the rule is wrong? it could have syntax errors or logic issues that prevent triggering.

Surprisingly the coding agents handle these situation well, with a little prompting, they can ‘give up’ and flag that a trigger could not be found. Therefore helping to check the rule as well as generate test data.

What if triggering a rule always triggers another rule? E.g.: if my text validation has a minimum length check - but to trigger that rule- I have to also not meet the required minimum of upper & lower case letters & numbers etc.

Again prompting (i.e. prompt engineering) for this eventuality allows the LLM to handle this situation, and after exploring possible solutions - it can progress and document the difficulty for human review.

File	Text	Rule Triggered
trigger_rule_1.txt	act wor4!	Minimum uppercase < 3 (got 0)
trigger_rule_2.txt	ACT WOR4!	Minimum lowercase < 3 (got 0)
trigger_rule_3.txt	ACT wor!	Minimum numbers < 1 (got 0)
trigger_rule_4.txt	ACT wor4	Minimum symbols < 1 (got 0)
trigger_rule_5.txt	ACTwor4!	Minimum whitespace < 1 (got 0)
trigger_rule_6.txt	ACT woor44!	Maximum length > 10 (got 11)
trigger_rule_7.txt	ACT4!	Minimum length < 6 (got 5)

Claude Code: Note on Rule 7: The minimum length rule (≥6 chars) is a special circumstance — satisfying all other rules requires at least 3+3+1+1+1=9 characters, making it mathematically impossible to trigger min-length in isolation. ACT4! triggers it alongside lowercase and whitespace violations, as expected per the special circumstances clause.**

Discovery

Given a sufficiently verbose validator, you can get the agent to elucidate and enumerate the rules through trial and error - a potentially useful tool to search for new rules. Also a useful double check that rules as documented and as coded - are working as you'd expect.

Summary

I've outlined how using coding agents for analysing rules and fabricating data to trigger those rules can effectively produce valuable test data. Rather than implementing vibe coding reems of code to try and build a system to do this - we've used the raw language & translation capability of the LLM and the basic development integration of the coding agent to deliver an elegant solution.

This is not AI doing coding or testing, it's a powerful language processing system working with a comprehensive rules based specification - to deliver useful test data.