BIus Working Paper No 2

Rules As Code

8th August 2023

1 About BIus

BIus - Basic Law-Making For Legislative Computer Systems is a research project looking systemically at how the state creates the digital systems underpinning its services.

Author/contact: gordon.guthrie@gov.scot

2 Purpose

The BIus working paper is designed to stimulate discussion about key elements of the relationship of the state to digital systems and their delivery. Your feedback, input, and particularly criticisms of this paper are most welcome. Feel free to distribute it however you wish.

Working papers are unashamedly technical, where that is required.

3 Executive Summary

Rules as Code is a movement looking at annotating legislation in machine readable formats that enable various technical transformations and tests to be performed on it.

The goal is to variously make law into one or more of the following:

- executable production code
- basic expositionary systems to drive shared understanding and iterative development
- systems that have had formal consistency proofs applied
- navigable information architectures that are machine traversable
- executable inputs into macro-economic statistical models
- as a feed to be interrogated by AI and made more comprehensible to people without legal training
- reference systems for regulated industries that can reduce their compliance costs

This paper identifies two new possible outputs which are important for digital transformation:

- test first development using property-based system tests
- a shared catalogue of data sources

The significance of this is that test-first development can dramatically reduce development times and costs – savings in the 10's of percents and not single percents. See Appendix 1 for worked examples with measured costs.

4 Table Of Contents

5 Context

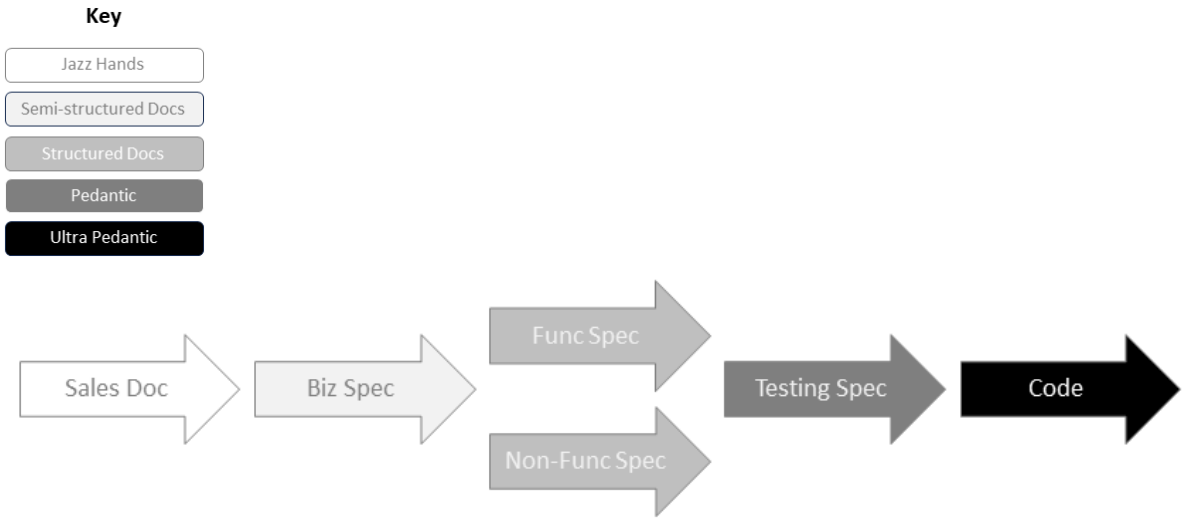
I have been talking to Denis Merigoux and his Catala team at INRIA, Bridget Hornibrook at the DWP and Adrian Kelly who has been working on LogLaw and Matthew Waddington and Flora Leather in the Bailiwick of Jersey.

Adrian was part of the OG Rules As Code endeavour - the New Zealand Government's *Better Rules for Government Discovery* Report - which is an oldie-but-goldie that is well worth your time reading.

However, no comprehensive study has been made of legal tech. This working paper is observational.

6 Background

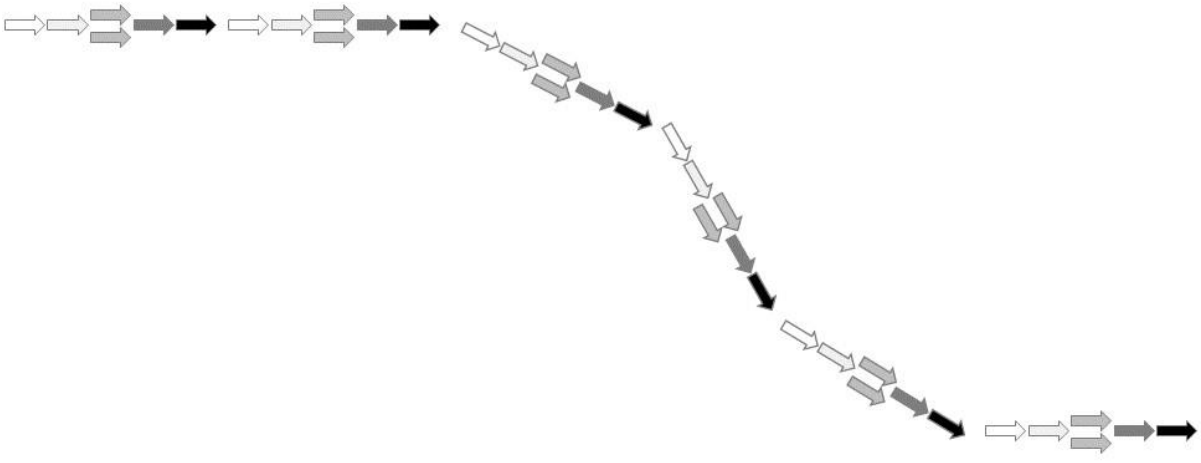
In the private sector the process of specifying a new computer system can be regarded as a process of empendantifying a series of documents, taking them from being fairly free form to very strictly structured:



Code is very pedantic – consisting as it does of 0's and 1's in a strict order – swapping any pair of them can cause systemic meltdown.

In this diagram I split out the technical specification into 2 parts – the functional specification and the non-functional one – for reasons that will become clear later.

Agilists might here be looking knowingly and saying "ah but that's waterfall!". All software development is waterfall – agile is just lots of them.



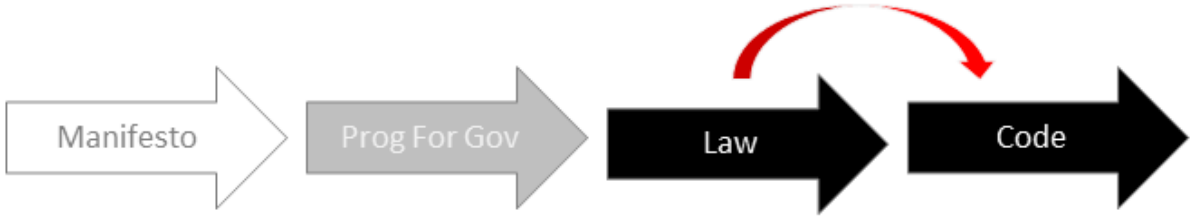
The quantum of work is not affected by doing it agile – the benefit comes from early course correction, eliminating rework and fix-ups and arriving at a better outcome faster and cheaper. Each agile sprint is a waterfall in its own right.

The public sector by contrast can be doubly-pendantifying:

Note: in the context of the Scottish parliament the Programme for Government is a 5 year living programme which includes all the primary legislation (Bills/Acts) passing through the parliament. Law pertaining to state systems is also defined in secondary legislation (ministerial orders).

Law, like code, is ultra-pedantic – a comma will support enough rope to hang a man.

The ***fever dream*** of Rules as Code (or to be fair, as it comes across to me) is to capture the core essence of the law in a machine-readable form and transform it into code, a great leap forward:



When we look at what Rules as Code people are doing with their tools – it isn't this.

The initial work in New Zealand was a lawyer-led approach to rethinking the development of law to enable simpler and better development of regulations, entitlements, calculations.

They pioneered cross-team working with parliamentary counsel, policy makers, service designers and delivery people working in cross-disciplinary teams.

The project also identified and struck down key barriers between policy intent and deliverability and demonstrated value and velocity by using rules as code.

Rule as Code tech can be used as way of building quick prototyping tools that:

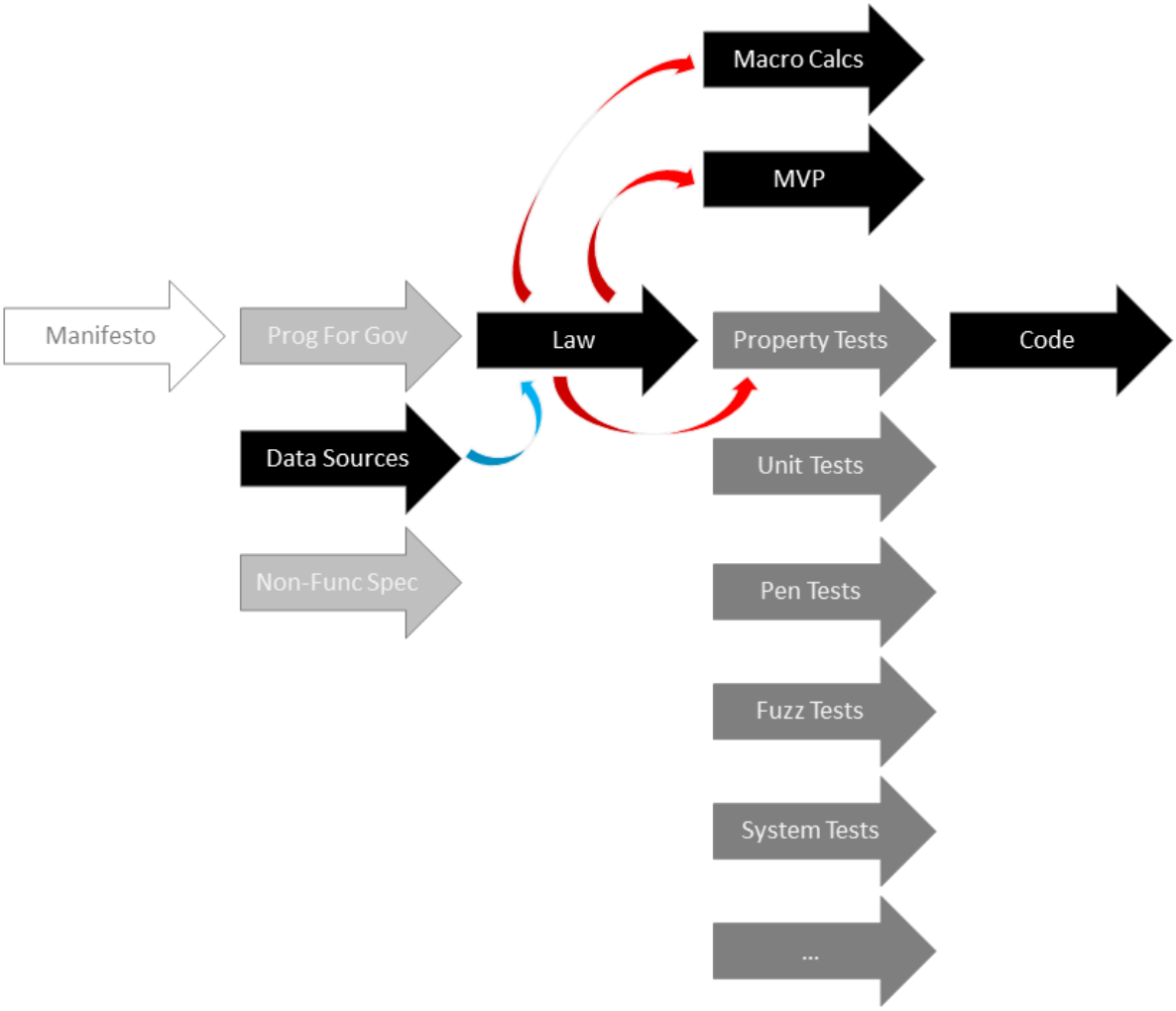
- enable fast design feedback loops in the development of policy and legislation – having a common 'surface' that members of different professions can engage with is an excellent tool for collapsing getting-on-the-same-page discussions and associated costs
- provide plug-in entitlement and calculation elements for financial modelling covering take-up and impact of benefits, monte-carlo exploration of tapers and better design of hardship/compensation schemes, tax base modelling etc, etc
- are a very useful quick'n'dirty first pass usability tool that can be used with co-design communities

But the leap to write-once/deploy is never going to happen. And its obvious why looking at the flow of pendantification. The law only covers the *functional spec*. In the case of social security that would be who gets what money in what circumstances.

It doesn't cover the *non-functional spec* – things like: you need to log in, the data needs to go into a database, there has to be rules-based-access-control, and dashboards, and cloud deployment and it must work in browsers and disburse actual payments to actual bank accounts.

I think a more realistic approach should be called *Rules As Tests*. In this world the machine-consumable annotated law consumes existing data sources (and their legal and regulatory definitions). It then possibly generates three testing outputs:

- macro-economic calculations – code that can be deployed without the normal non-functional requirements because it is used in economic modelling systems where individuals are treated statistically and not as individuals (economic testing)
- an MVP (single user, basic GUI) which can be used to iteratively seeking consensus during the development of the law – saving time and effort in developing regulations whilst increasing quality and effectiveness is a worthy goal in its own right
- as a property test generator for the final live production system



The use of these technologies to make MVPs is proven by the work of Bridget Hornibrook and Adrian Kelly at the DWP and elsewhere. Denis Merigoux and his team have demonstrated its possibilities in Macro Calculations.

Consequently the rest of this paper will focus on two things:

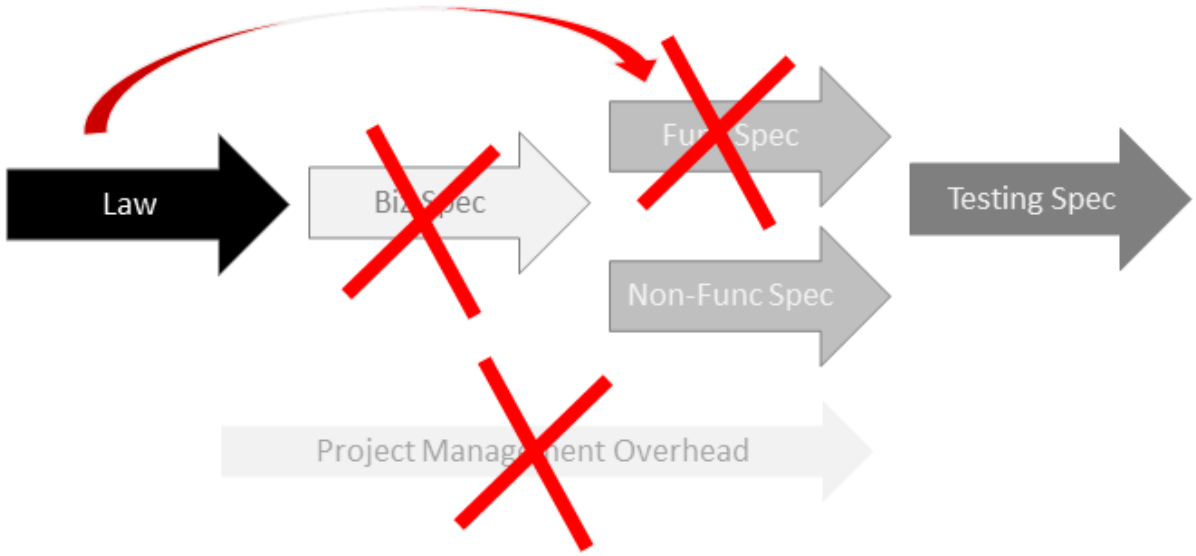
- Test First Development
- the consumption of Data Sources
    - (this will include the pull-back of the Non Functional Specs in the diagram – not strictly related technically, but organically related organisationally)

Test First development brings tremendous gains – the fact that the test type is Property Tests is just an added bonus.

7 Future State

7.1 Test First Development

Test First Development is a common technique to improve software delivery and reduce rework (and hence costs). However in this instance by generating tests it adds additional value by eliminating work:



With the right tooling and appropriate care at the design-of-the-law level the entire business and functional specification steps can be eliminated. Well designed, comprehensive and appropriately annotated system tests can perform the majority of this work.

Project management in major IT deliveries imposes huge reporting demands on software developers and other professionals to assemble a picture of progress that is comprehensible by non-technical managers. Much of this can be expressed in terms of *simple tests passing Vs tests not passing* – if and only if the test suite is known to be functionally complete.

See Appendix 1 for a measure of the size of the cost savings – but be cognisant that previously measured costs in different circumstances can't be simply read across.

For systems like social security and taxation there are additional benefits. It is possible to generate property tests. A property test takes a set of inputs and states that the output must have these properties. In the case of a social security system you could generate a test along the lines of:

> *Caroline is 42, has two children aged 11 and 7, one with special needs. Her husband earns £2,400 a month and she is entitled to X in benefits*

If the end system produces the same value of X – good to go.

But these tests are generatable – there is also a *Caroline with 3 kids, and an income of £1,200* and so on and so on. And Caroline can give birth, and her middle daughter can be paralysed suddenly in a car crash, and her man can get cancer and rolling on and over all the edge cases endlessly.

Generative property-based tests are unbounded in number. Most software testing problems can be summarised as "not enough tests" – test first property generative tests have the opposite problem – "too many to run".

A huge quantity of the cost of major software programmes consists of a number of things:

- driving agreement and understanding amongst all stakeholders that they are talking about the same thing
- building a model of progress towards a goal that can be used in communication with stakeholders
- testing that the developed software actually does what it is supposed to go – and delivering confidence to stakeholders and team members that it is reliable

Rules As Code/Tests can significantly reduce costs by addressing each of these areas – by generating MVPs that professionals from different disciplines can share, by generating completion figures in the form of tests passing/not-passing and by actually writing huge and flexible test suites.

(Readers should beware of going over the top – the annotations to law that Rules As Code uses are not in themselves judiciable – the fact that a system passes the system tests does not, in itself, mean that the system is legal or complies to the rule of law. As long ago as 1970 Djikstra told us *Program testing can be used to show the presence of bugs, but never to show their absence!* It follows that tests suites generated by Rules As Code can only tell us when the system under test violates the law, not that it conforms to it.)

7.2 Consumption of data sources (and common non-functional specification)

Moving towards using code annotation in law opens up two other potential savings. The functional specifications describe what the software does – what makes it a social security system as opposed to a tax system. They are different for each system.

By contrast the non-functional specifications usually are similar (if not identical) for different systems. It is perfectly possible for a tax and a social security system to have identical log-in mechanisms, reuse the same payment rails, work in the same browers, use the same underlying database technologies.

So simply splitting functional and non-functional requirements enables the partial Lego/IKEAisation of systems. The expectation that systems run by government have common and standardised non-functional requirements eliminates rework in its own right.

The Rules As Code language typically define entities in an abstract sense ‘a person’, ‘a taxpayer’ or ‘a child under 16’. What we want in government is not the abstract person – but the reified one – ‘a person with an identity on the government identity service’ or ‘a person who has a medical certificate issued by a recognised national health system’.

If the definitions in different parliamentary acts can be harmonised and systematised, then the implementations of that data can be merged. The move to services that encapsulate and expose single sources of data under an appropriate API will become possible.

How this process would be embedded in a language like Catala needs to be determined. It might be a simple case of using include/header files – so common definitions are stored in their own legislation which is annotated in the usual way – but lacks process or rules and contains only entity definitions. These definition can be included in other legislation and their Catala entity relations imported. There is then (outside the language/law definition) a presumption that these entities are implemented as services somewhere and offered as APIs.

These two problems merge because some of the core barriers to merging data sets don’t like in the functional or calculation aspects of a particular statute but in regulations that are formally non-functional – so access controls (who can see what data) or data retention policies (how long the data must be kept for) and a myriad of other seemingly insignificant things that act as a barrier to consolidation.

Data consolidation is a key activities for two reasons:

- it brings simplicity to the user – if only one system holds your address and surname then changing them when you get married is much simpler
- it reduces cost – every instance of data has to be maintained

8 Barriers to uptake

Testing as a discipline is fairly low profile in Scottish and UK governments. GDS doesn't have explicit testing standards much beyond "test things" in the Service Manual[1] and there isn't a service community for quality[2]. So jumping from here to Property Based Testing is a not-insignificant leap and would require a training/education programme. (This is on top of the migration of policy and legislation people from the old world to the new.)

For the Scottish Government with its stated policy aim of independence, building these capabilities now in order to support the creation of new national institutions (central bank, main tax office, etc) would seem to be a sensible option.

But there are also outstanding technical issues – I have explored them in the context of Catala in Technical Appendix 2 – Outstanding technical issues.

In summary there are both skills and technical barriers to moving in this direction.

9 Further work

And there is a second element of the modern state that Rules As Code can help in. We substantially live in a regulated world. *Rules As Tests* could also potentially be used to publish compliance test suites for regulated organisations to use. It has been reported to me that Angus Moir at the Bank Of England is exploring this use.

In addition Pia Andrews reports that publishing a reference example of banking regulations using Rules As Code saved a single regulated bank $16m a year (not sure if that's $US or $AUS there…)

[1] https://www.gov.uk/service-manual/technology/quality-assurance-testing-your-service-regularly
[2] https://www.gov.uk/service-manual/communities

10 Technical Appendix 1 – Test First Development Worked Examples

10.1 Test Case 1 – Hypernumbers

Hypernumbers was a startup that aimed to build a web-native spreadsheet (Google Sheets is an open source desktop spreadsheet under the covers). Every cell, every page, every range would have its own URL and these URLs would be composable in functions (making a functional programme of the web).

To that end a goal of Excel 95 compatibility was set and a test framework was developed that could convert Excel spreadsheets into system tests.

A function would be inserted into a cell and Excel would resolve that function (and any dependency tree it was involved in) in the usual manner returning a value.

A programme was written to traverse all the test spreadsheets in a directory and then make of every populated cell a systems test.

The tests were hand written – but that was simply a large set of spreadsheets. After about 2 weeks there were 100,000 of them. At the time of the first test run the results were *6 tests passing, 99,994 tests failing*. (The 6 passing tests had the formulas `1`, `1.0`, `1.1e+1`, `-1`, `-1.0`, `-1.1e+1`

During the development some hundreds or thousands of unit tests were written alongside the hundred thousand of end-to-end system tests.

The cash-equivalent costs (what we would have spent if we had paid market salaries and had an office, etc, etc) were between £1.25 and £1.5m – and the COCOMO II Estimate (based on the Open Office spreadsheet source code) was £8m to £20m. These figures need to be treated with appropriate caution. Reading across savings on this scale to the public sector would be a mistake – but the possibility of very significant costs savings is very real.

10.2 Test Case 2 – bet365

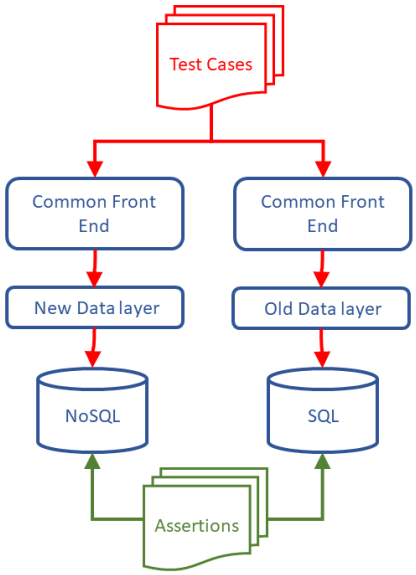
bet365 is the largest and most successful internet company in the UK. bet365 had its growth blocked at about £30bn turnover because its data layer (based on Microsoft SQL) just couldn't scale at peak. Betting is a very bursty business with lowish daily traffic, weekend peaks and Himalayan traffic at the Grand National, the World Cup etc.

We put a slip into production and logged the Germany-Brazil semi-final in the World Cup and captured over 8 million discrete events.

The logs were processed to anonymise them before copying them down from the production zone in the data centre. They had to be post-processed to create setup activities (create all the users, create all the markets in all the fixtures that they betted on, price those markets, etc, etc).

These events were then replayed side-by-side into one version of the code with the existing data layer and one with the new data layer.

The test case was run twice and the results from the two systems were compared.



If the two systems returned the same results for all posts then the tests themselves passed. As belt-and-braces a job that ran over the post test databases was created. It asserted that the persisted data was consistent (same number of users, same number of bets, total wagered, total won and totals lost identical in both cases, and so on and so forth).

I do not have working cost estimates for this work, but the live/no-down-time replacement of the data layer was a small team of less than 10 working for less than a year.

11 Technical Appendix 2 – Outstanding technical issues

Apart from the training issue – there are three outstanding technical issues that Catala would need to solve to be fully useful for this approach:

- generating the tests
- making the tests anti-fragile
- decoupling the system under test from Catala

I am using Catala as the example because it's the one I understand best, and the one with a fully articulated parse chain[3].

11.1 Issue: generating tests

Testing is substantially about 2 things:

- applying a defined payload to a point of application
- matching the response to an expectation

Its clear that Catala has enough information to generate the data payload and it is already capable of generating code to calculate the expected response – the problem is the point of application – and this in itself brings fragility issues.

Both the examples in Technical Appendix 1 – Test First Development Worked Examples have one thing in common – the test suite and the system under test share a routing table – ***by design***.

In the online spreadsheet we can express a route to a particular formula/value pair in the Excel spreadsheet that contain the test definitions as a generalised path (which by definition is directed and acyclic):

```
directory -> file name -> sheet -> cell addressed as row/column
```

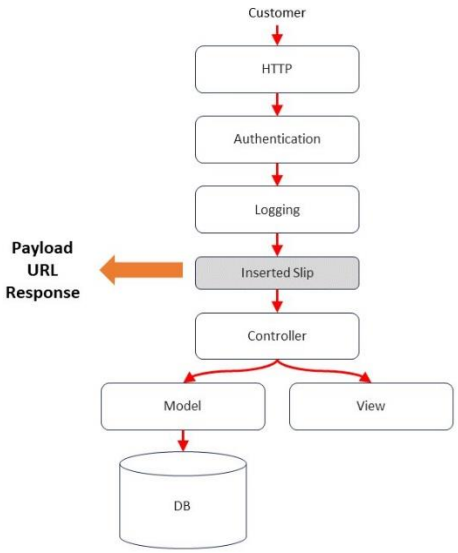
In writing the tests this can be transformed into a URL which has the self-same properties:

```
http://testsystem.local/directory/file name/sheet name/cell address
```

It is demonstrably trivial to apply the payload at the point of application and get back a result with can be compared. The entire test suite can run and where the formula isn't parsable (as **=1+1** wasn't on day 1) the system under test returns an error value and the assertion, and hence test, fails.

[3] https://catala-lang.org/ocaml_docs/catala/index.html

In the case of bet365 the same capability arose from where we recorded the test case:



Because the test was a side-by-side test with the same front end and only the DB layer (ie the model and below) changed out, we were able to capture the payload and the URL (the point of application) as well as the result. This meant tests could be generated.

One problem with Rules as Test is not that a routing table cannot be constructed from the law – the sort of law under consideration – propositional logic – can be expressed as a directed acyclic graph (Catala will throw a consistency error if that is not the case) and thus the application of data items and the retrieval of calculated values ***could*** be done via a URL structure which ***could*** be generated. The problem is that that URL structure is unlikely to have the appropriate affordances for a user-friendly system.

A second problem is that there are actually two sorts of tests that we can generate:

- simple property tests
- hysteresis property tests

A simple property test scenario is *Caroline has a child* and applies for benefit.

By contrast a hysteresis one is *Caroline is childless and applies for a benefit, then she has a child, then the child gets a terminal disease, then the child dies*.

These two tests capture different aspects of systems behaviour and ideally we want both of them – writing tests for both of them automatically in an anti-fragile manner will be a challenge.

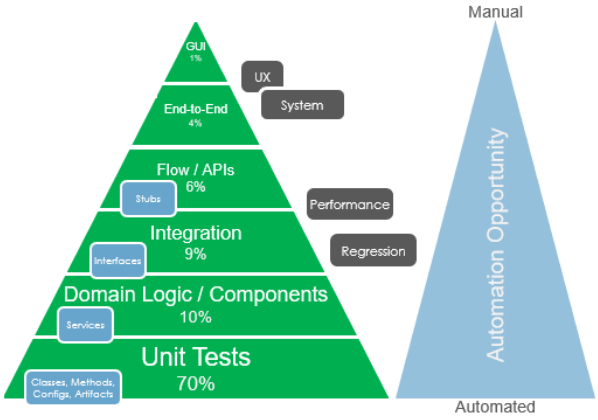
It would be trivial to have Catala generate a skeleton set of tests and a complete set of generators and then let software developers assemble a test suite as they went along and implemented features.

The problem with this approach is that the day 1 progress report would not be ***6 tests passing, 99,994 tests failing*** but ***6 tests passing, 0 tests failing***. The lack of insight into the

missing *99,994* would in and of itself conjure up a complete project/progress management apparatus whose elimination we are seeking.

11.2 Issue: fragility

The second issue arising is fragility. The property-based tests are a sub-class of system or end-to-end tests which are frowned upon in most software shops. Here is the test pyramid from the UK Hydrographic Office[4]:



The strategy here is to test discrete components and then do the minimum amount of testing to ensure that they are plumbed together. It builds on the principle of shadowing.

Shadowing is when the failure of one test guarantees the failure of another.

Consider the following spreadsheet formulae written as tests:

Formula	Expected Value
1e+1	10
=1e+1*10	100
=sum(1e+1, 1)	11

If the first test fails because the spreadsheet under test doesn't not yet understand scientific notation then the 2nd and 3rd will also fail – they are shadowed by the root test.

A test suite where one error causes a cascade of failing tests is a fragile suite – fragility is experienced by software developers when a small change to the code causes a much larger amount of work to make the test suites pass again.

The UKHO pyramid deals with test fragility in two ways.

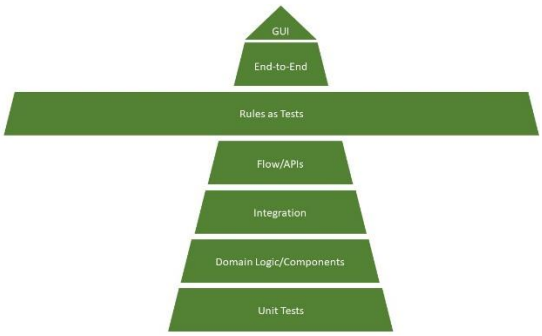
Firstly it tries to reduce shadowing to a minimum – which is why it's a pyramid and not a column or a vase.

[4] https://github.com/UKHO/docs/blob/main/quality-assurance/test-strategy.md

Secondly is it uses the knowledge of shadowing to triage the tests. The pyramid roughly is a hierarchy of shadowing: if there is a unit test failure in the bottom layer, there is an expectation that might be failures higher up in the domain logic/component tests, and integration tests, and flow/api tests and end-to-end and gui tests.

In the event of a regression failure and a sudden massive amounts of tests no longer passing the developer has their failure triaged by the shadowing structure – first fix all the failing unit tests and then retest. Usually that will fix it. If not move on to the failing domain logic/components test and systematically work up the pyramid.

By using generated property-based end-to-end testing we don't see a simple pyramid but a table:



(Generated property-based tests would augment but not replace normal testing protocols.) With property-based generators there might be 10,000,000 rules as tests generated – 99.99% of all tests.

This brings a couple of issues:

- when to run the test suite
- what the develop does when confronted by 95,000 failing tests

The first is a set of practical problems arising from how long it takes. Where in the build chain does it run – not on the client side pre-comit, but:

- on each commit?
- daily overnights?
- weekly over-weekends?

If it runs daily or weekly then it runs against a basket of commits – so who is responsible for fixing the faults?

The second problem is our old friend shadowing – generated tests are slight variations on each other and the amount of shadowing is enormous. If the tests use a reducing framework like QuickCheck – it will triage for you. In the absence of that the test system will need to have some simple-to-complex naming convention (at Hypernumbers we had test suites named with prefixes **a_**, **b_**, **c_** etc, so fix the **a**'s first, tests were also arranged simple-to-complex within our hand-written test suites too – fix the top ones first and work your way down).

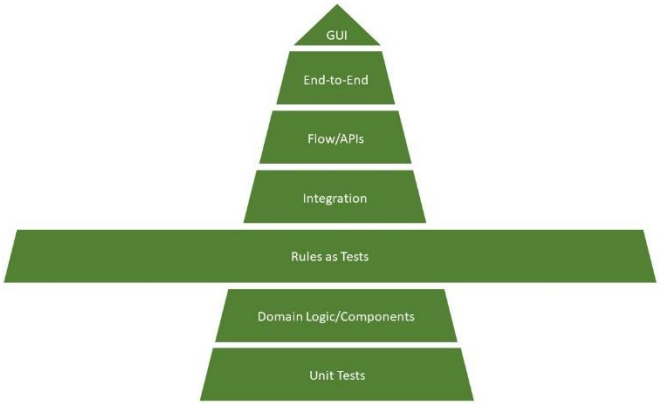
Developing a test system that can generate self-triaging tests is not insoluble, but it requires someone to do it.

11.3 Issue: decoupling

Catala is a programming language – its parser chain is written in OCaml, but it has hooks for transpilers to produce outputs in a range of languages other than OCaml, most notably Python and Javascript[5].

If it is to be used to generate test suites it needs to respect the fact that the systems under test may or may not be written in any particular language. The mechanism that Catala uses for the calculations (transpile them to your target language) may or may not be appropriate for testing (generate a test runner in your target language) – or it may make sense to expect the system under test to expose URLs to which payloads can be applied – which would enable a single test runner executing generated tests for all development languages. As noted earlier this brings its own.

If Catala were to output not calculations but state machines it might be possible to rejig the property-based tests as component tests:



This would solve the problem of the routing table – you would be applying state transitions at some internal level – the state transitions (sans logic) could probably be generated too as stubs. I can see a way to do this for Elixir/Erlang and the Beam in general, where there is strong and native support for state machines – but this violates the principle of decoupling.

11.4 In Summary

There needs to be a substantial programme of work to address and work through these issues before Rules as Test could be production ready. Building a test runner with the right affordances to fend of project management demands, and also be integratable in automated build processes on GitHub is non-trivial.

[5] I had a pop at writing first a transpiler first to Gleam and then Elixir (both Beam languages) for fun, but only having 2 or 3 days, and not speaking either OCaml or Catala (or Gleam) proved a bit of a barrier to making a lot of progress – but adding transpilers to other languages in Catala is, as they say, *only a small matter of code*.