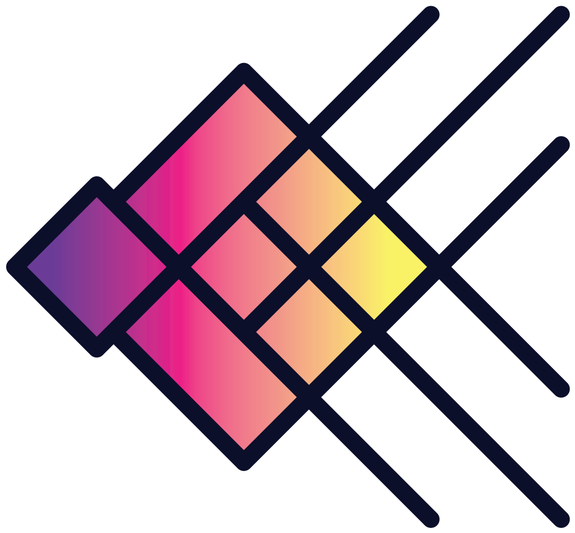


provengo

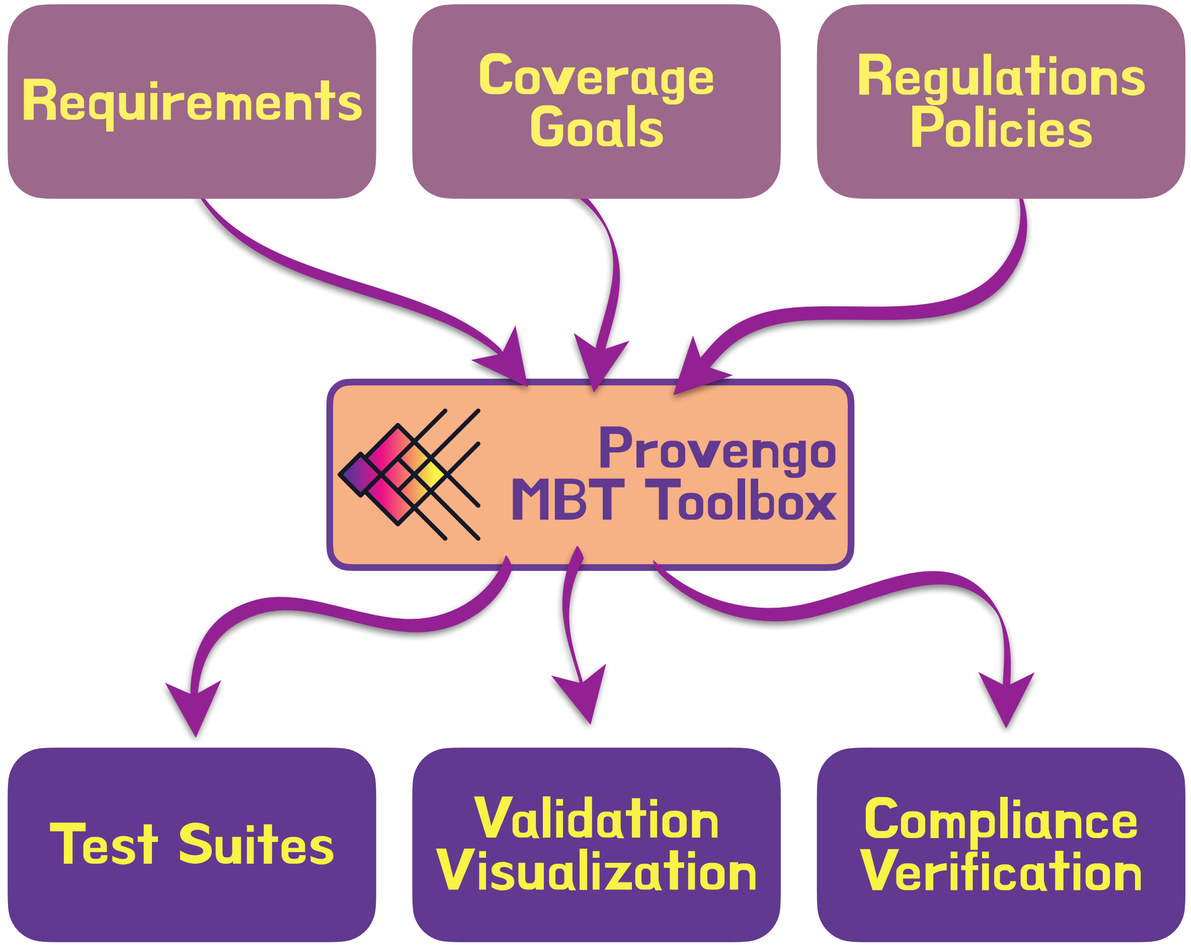
Power Tools *for* Model-Based Testing

Model-Based Testing

Model-Based Testing is a set of tools and techniques to **automatically derive test suites and test scenarios** from system specifications.

Derived tests can be executed using test automation, exported as manual test books, or generated as code for 3rd party tools.

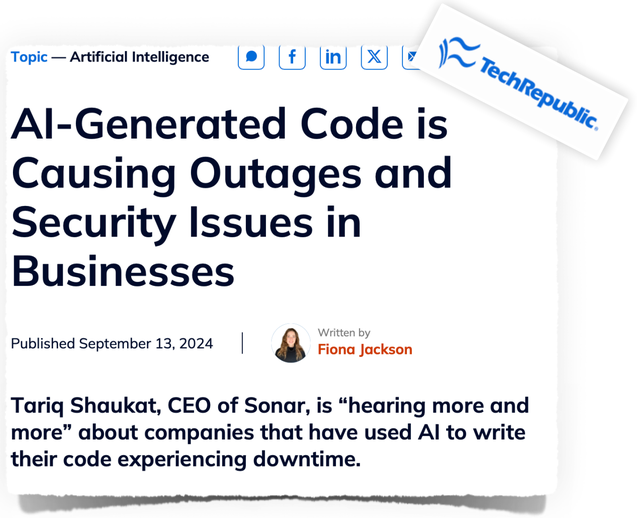
Model-Based Testing can also validate and verify system or feature specifications, even before implementation begins.

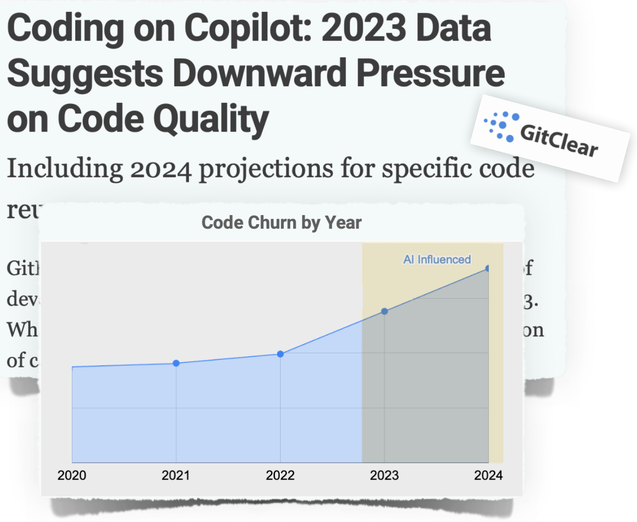


Upsides of Model-Based Testing with Provengo

Provengo's Model-Based Testing Toolbox helps you:

- Reduce time to market
- Stay within budget
- Improve quality *without sacrificing* agility
- Reduce development risks and deploy with confidence
- Get comprehensive test coverage faster
- Drastically reduce test maintenance efforts
- Uncover design errors earlier and at a lower cost
- De-risk usage of GenAI coding tools



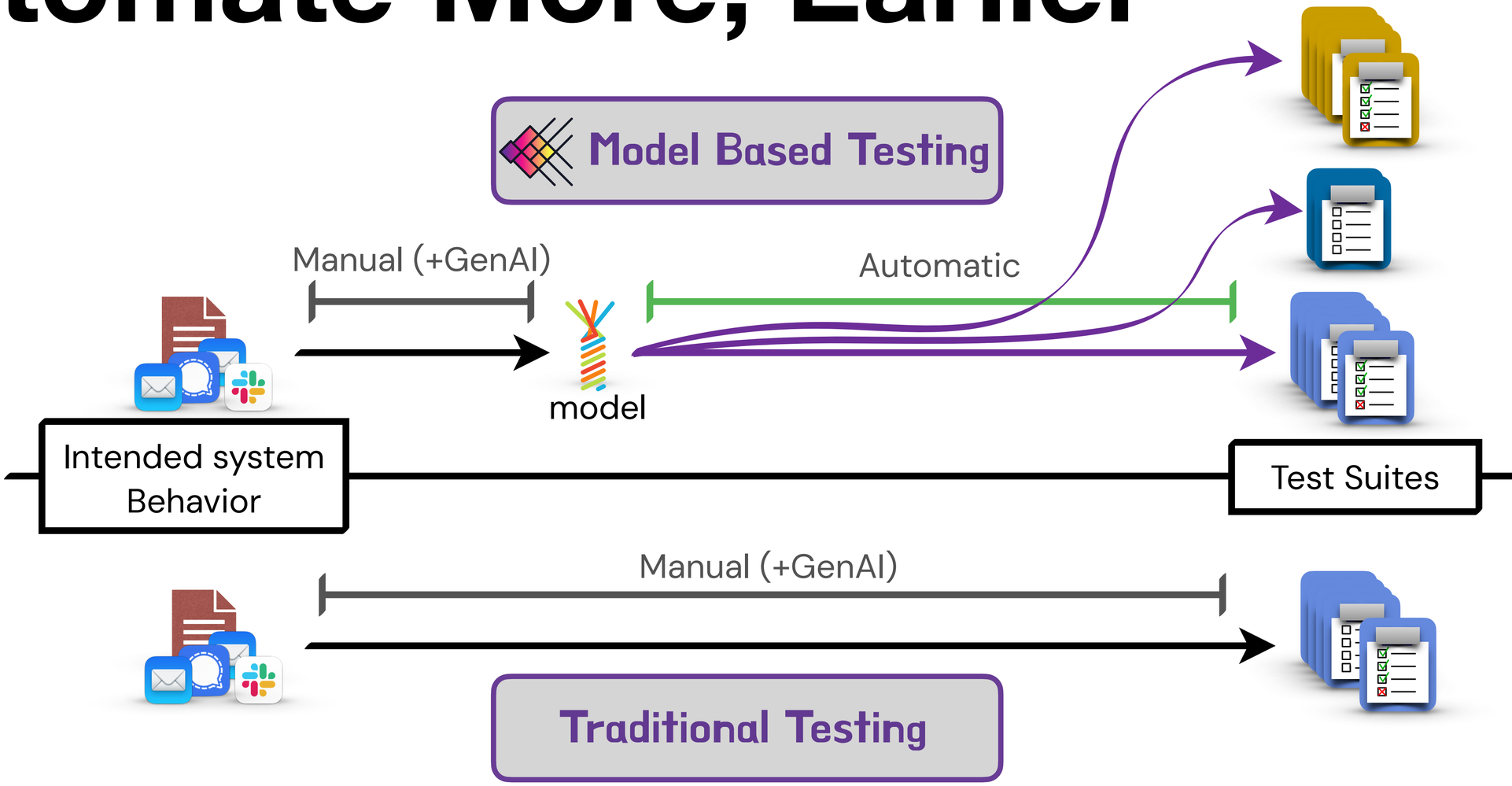


Automate More, Earlier

Traditional testing methods rely on the QA team writing maintaining test scenarios manually. Maintaining alignment between the requirements and the test suites is tricky and laborious.

Model-based testing captures intended system behavior in a formal model. It then uses this model to automatically generate test scenarios, and compose optimized test suites. Maintaining alignment between the requirements and the test is done automatically.

Since test suite generation is automatic, it is very easy and fast to generate and maintain test suites focused on specific goals.

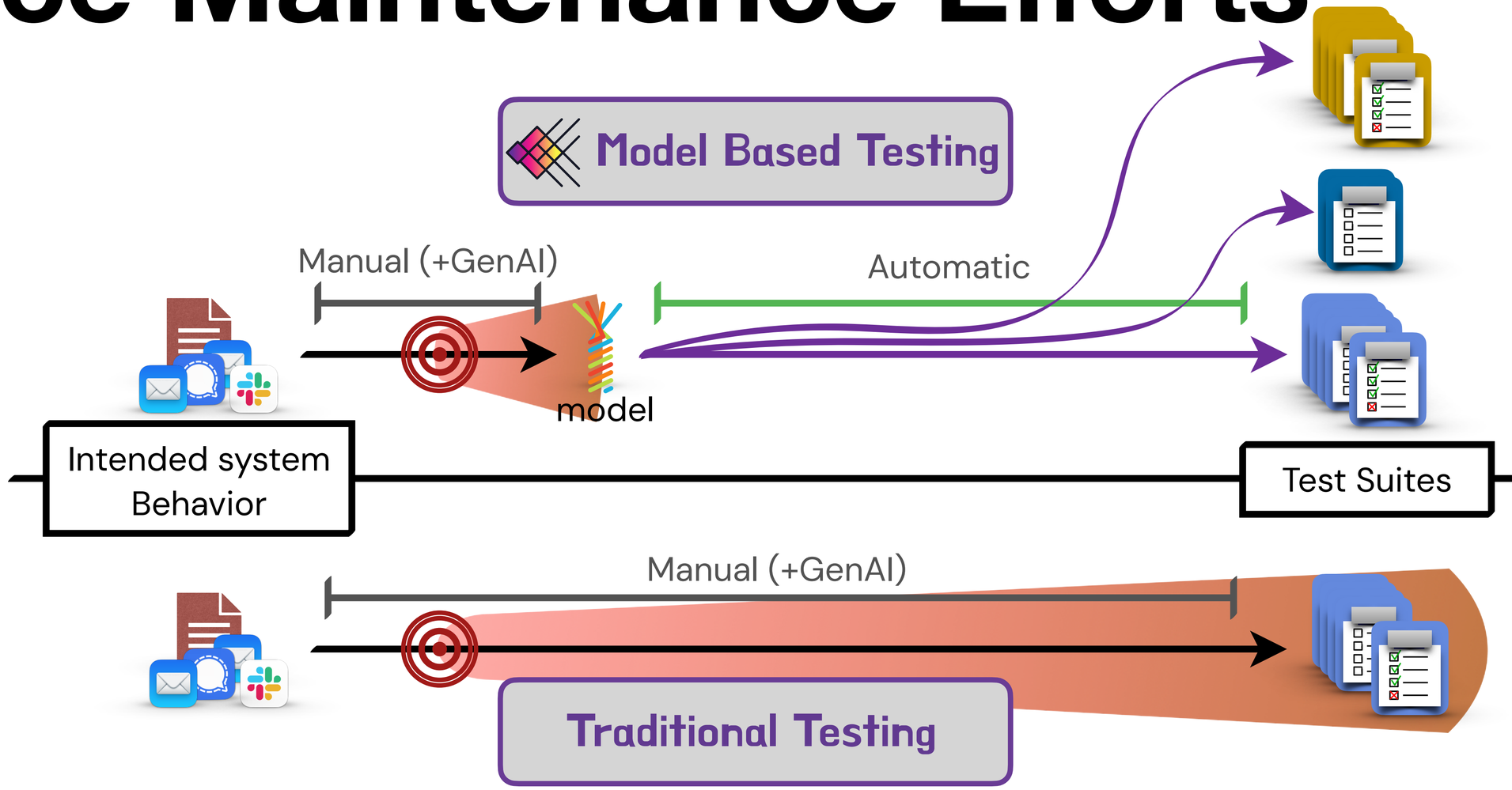


Reduce Maintenance Efforts

Changes* in system requirements or interfaces require changes to test suites and individual scenarios.

In traditional testing techniques, this effort is proportional to the amount of test scenarios, which forces QA team to choose between being agile and having a good test coverage.

When using Provengo, users can discard the old scenarios, update the model, and re-generate the test suites. So you can be both agile *and* have good test coverage. With less effort.



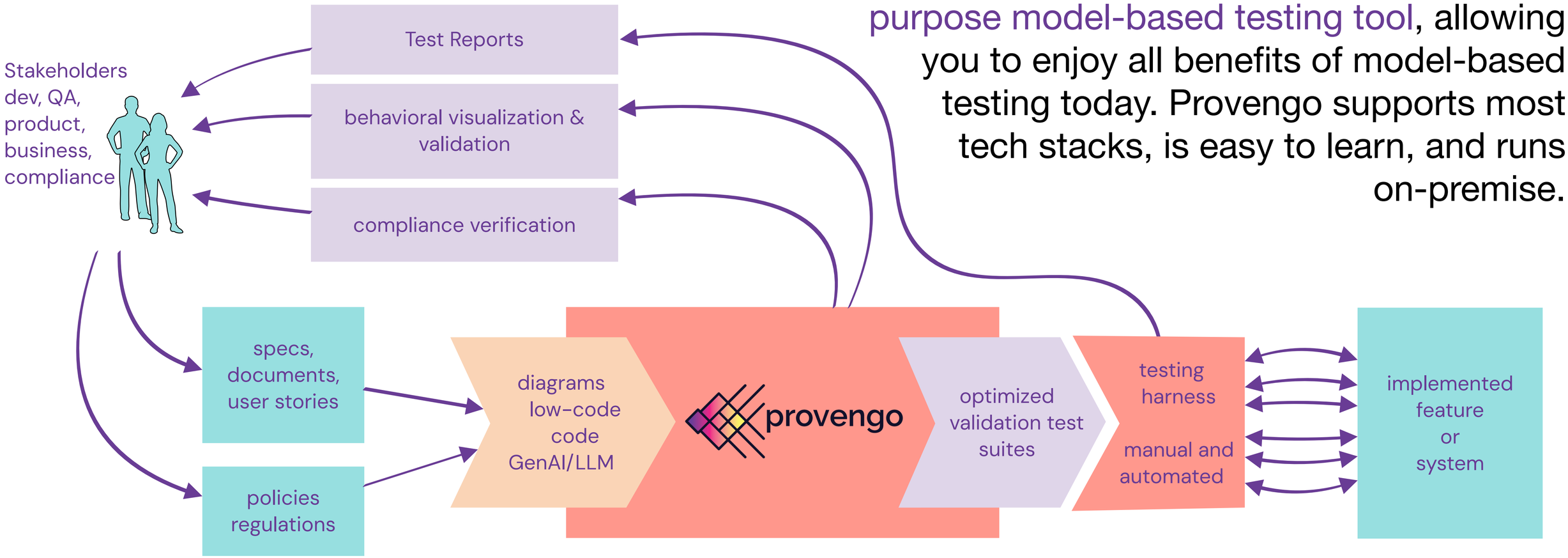
* Or mis-alignments, ambiguities, and general mis-understandings between stakeholders. We know, we've been there ;-)

Testing: Model-Based vs. Traditional

	Traditional	Model Based Testing
Test Scenario Generation	Manual	Automatic
Test Plan Generation	Manual	Automatic
Test Plan Maintenance	Manual, laborious	Automatic
Requirement Validation	n/a	Automatic
Scenario Visualization	n/a	Automatic
Stakeholder Alignment	n/a	Helps align teams using automatically generated visualizations and scenarios
Measurable Functional Coverage	n/a	Automatic
Test suite optimization	Manual, lots of work	Automatic
System and test documentation	Manual	Accurately documents intended system behavior
Effect on Agile	Test maintenance decreases team velocity	Faster+earlier feedback cycles, increased quality at no cost to team velocity
Automation	After system development	*Starts with design*

Provengo System Overview

Provengo created a breakthrough general-purpose model-based testing tool, allowing you to enjoy all benefits of model-based testing today. Provengo supports most tech stacks, is easy to learn, and runs on-premise.

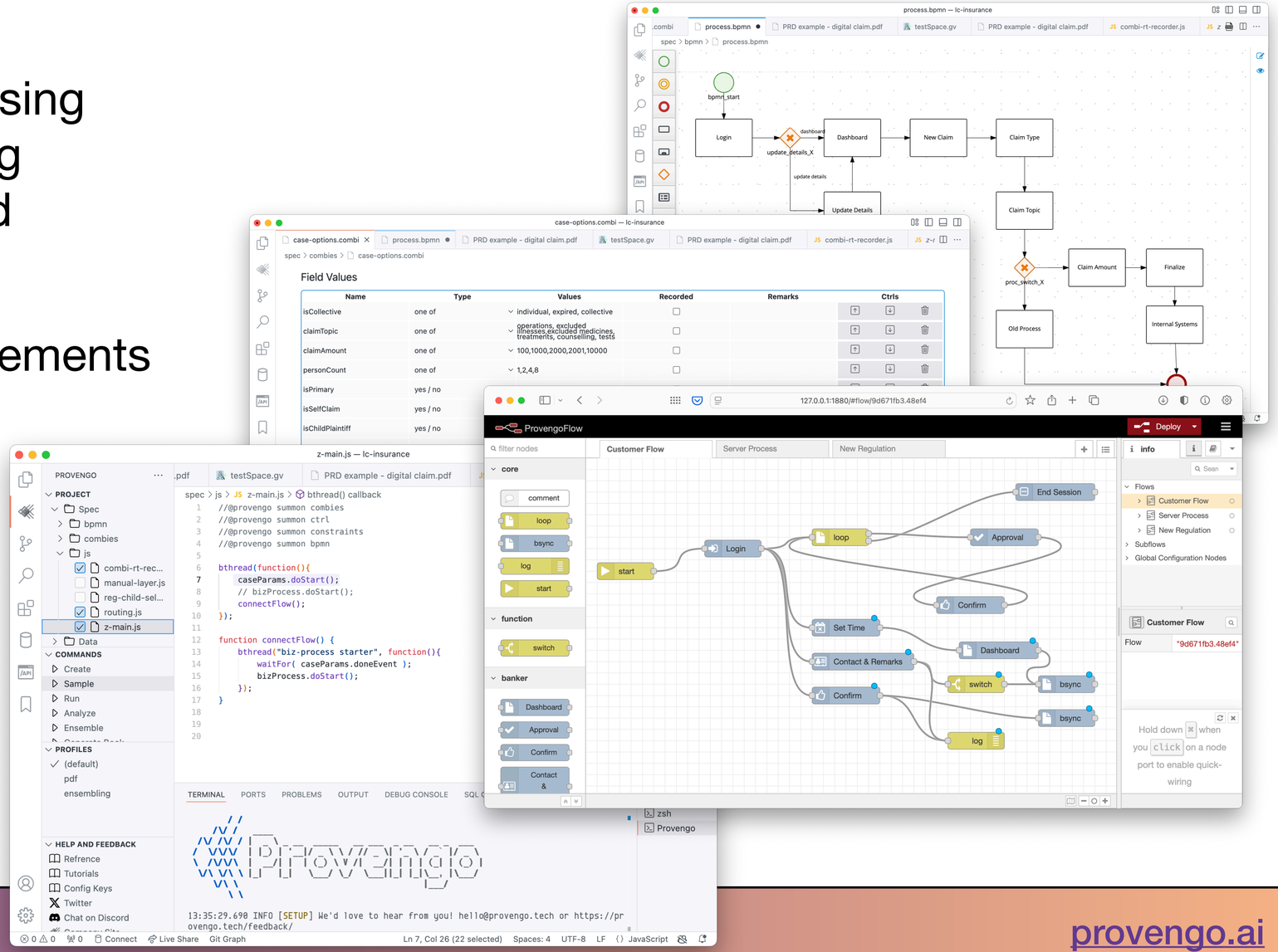


Modeling with Provengo

Describe intended system behavior using various modeling languages, including diagrams, low-code, form-based, and specialized JavaScript.

Use Generative AI tools to turn requirements into code.

Provengo compiles multiple languages into a single, complete model. This allows users to create comprehensive models by mixing and matching intuitive, complementary descriptions.



Visualization and Validation

Model visualizations and generated scenarios are great ways to validate intended system behavior with non-technical stakeholders, *before development even begins*.

Model visualizations and scenario tables on this page automatically generated using Provengo.

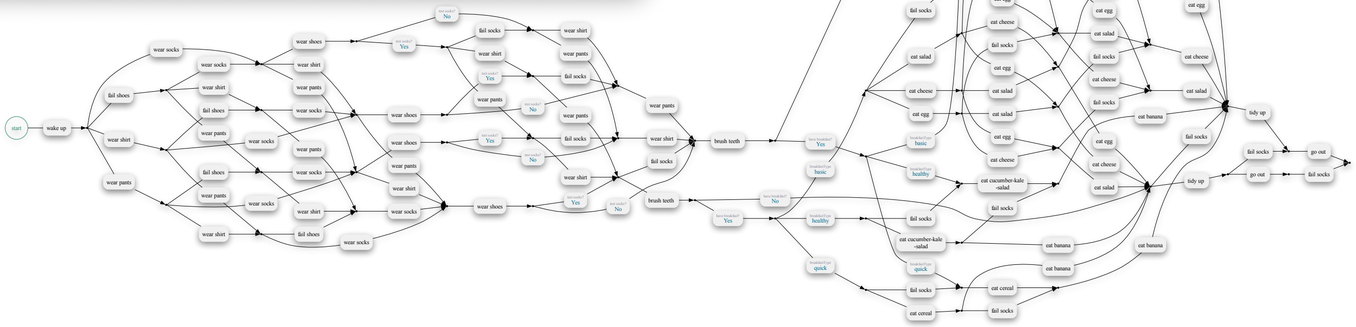
Provengo Test-Book

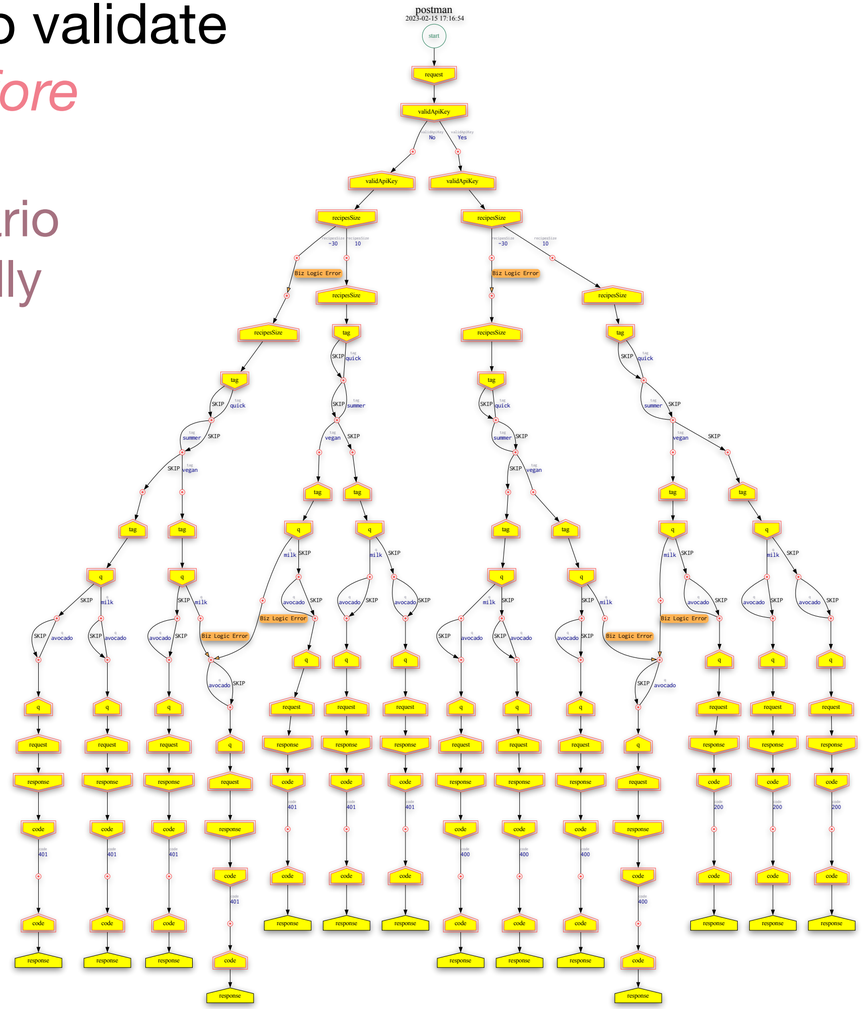
Test Book for *lc-insurance-gwt*

Path: `/Users/michael/Documents/Consulting/Provengo/ProvengoGH/clients/lc-insurance-gwt`

▸ Filter

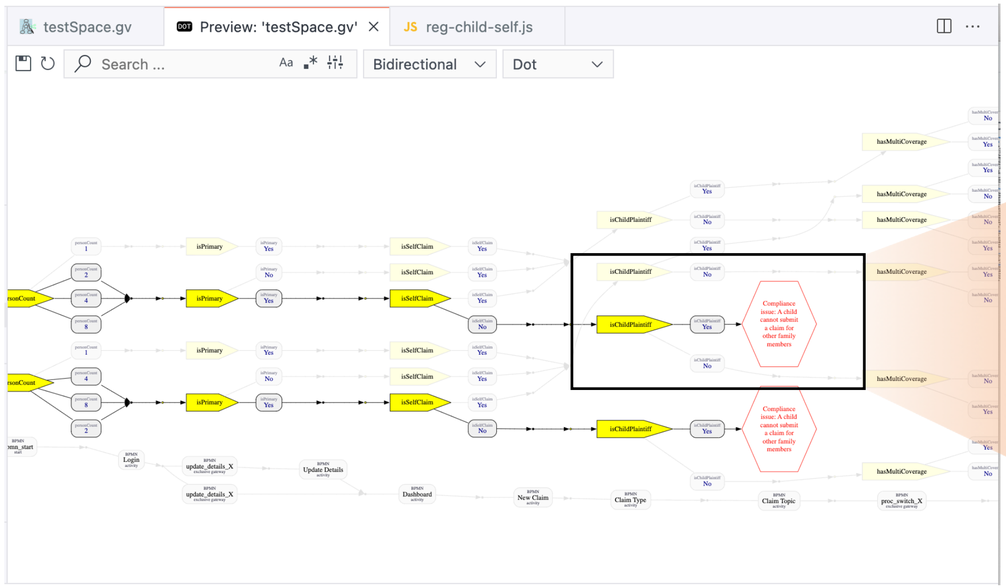
Test	Properties										02-WHEN		03-THEN	Labels
	01-GIVEN													
	hasDebts	hasExistingClaims	hasMultiCoverage	isChildPlaintiff	isCollective	isPolicyValid	isPrimary	isSelfClaim	isTariffValid	personCount	claimAmount	claimTopic	result	
Test 1	No	No	Yes	No	expired	No	Yes	Yes	No	4	1000	counselling	reject	
Test 2	No	Yes	No	Yes	collective	Yes	Yes	Yes	Yes	4	2001	treatments	manual	
Test 3	No	Yes	No	No	gold	No	No	Yes	No	4	1000	counselling	reject	
Test 4	Yes	Yes	No	No	collective	No	No	Yes	No	8	1000	treatments	reject	
Test 5	No	Yes	Yes	No	gold	Yes	No	Yes	No	8	100	counselling	reject	
Test 6	No	No	Yes	No	gold	No	No	Yes	No	2	2001	treatments	reject	
Test 7	No	Yes	Yes	Yes	collective	Yes	No	Yes	No	2	1000	tests	reject	
Test 8	Yes	Yes	Yes	No	expired	Yes	No	Yes	No	2	2000	treatments	reject	
Test 9				Yes	individual		Yes	No		10	2001	tests		Violation
Test 10	No	No	Yes	No	collective	No	No	Yes	No	4	100	treatments	reject	
Test 11	Yes	No	No	No	expired	No	Yes	Yes	No	4	10000	counselling	reject	
Test 12	No	Yes	Yes	Yes	individual	Yes	No	Yes	No	8	2000	counselling	reject	
Test 13	Yes	No	No	Yes	individual	Yes	No	Yes	No	4	10000	treatments	reject	
Test 14	No	Yes	Yes	No	gold	Yes	Yes	Yes	Yes	1	1000	counselling	manual	





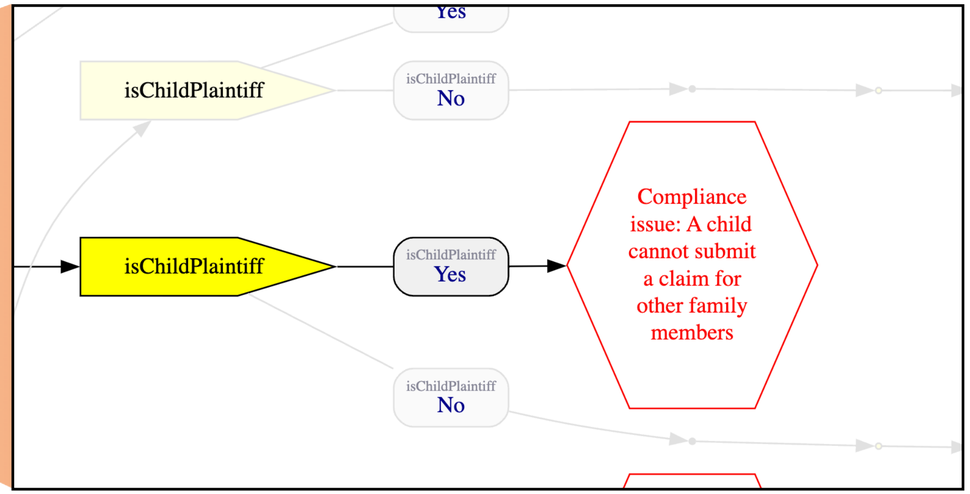
Requirement Verification

Provengo detects cases where the intended system behavior does not comply with regulations and company policies - essentially finding bugs in the requirements before implementation begins.



Provengo detects both *safety violations* ("something bad happened", e.g. money transferred to an unknown customer) and *liveness violations* ("something good did not happen", e.g. a user can remain forever at a free tier that supposed be offered for a limited time only).

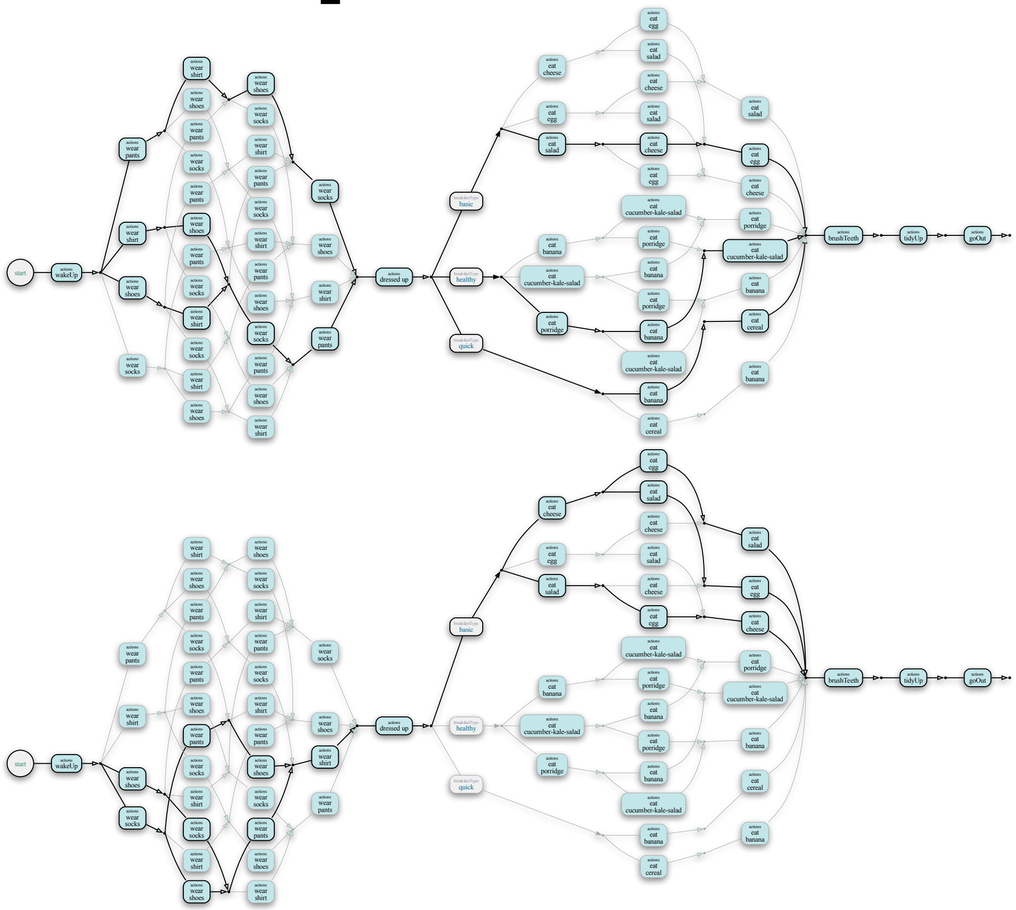
Provengo provides the series of steps leading to the violation, *helping product definition teams correct requirements before sending them to developers*.



Automatic Test Suite Composition

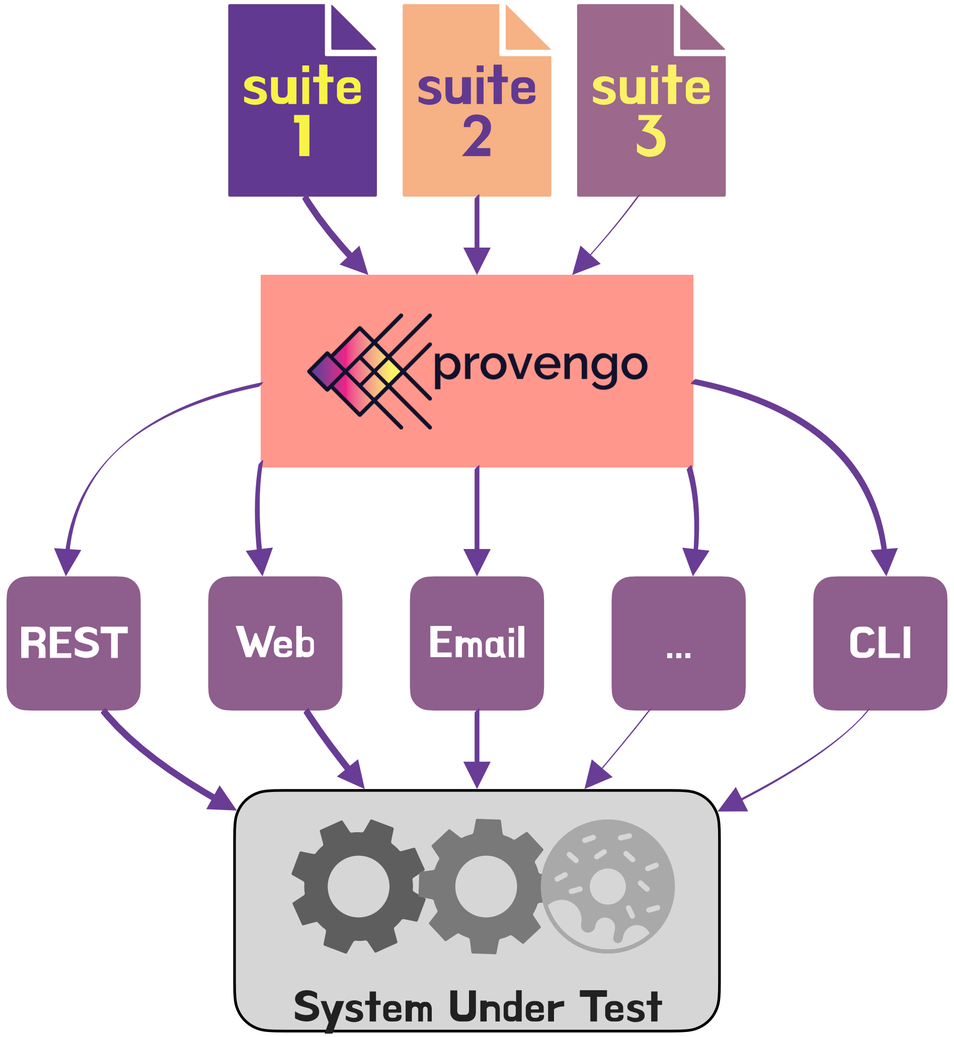
Define test coverage goals such as functionalities, feature usages, events, sub-processes, or pairwise, and let our advanced algorithms compose a test suite optimized to your goals.
In seconds.

Provengo uses advanced algorithms such as *evolutionary programming* to compose optimized test suites based on user-defined criteria. This means that test suite generation becomes *easy*, so you can focus on testing a specific problematic feature, generate a system-wide sanity suite, or a thorough UAT suite - without all the time and effort this would normally take.



Same system, different test plans
Two tests suites for the same system, generated using different coverage goals. Visualizations generated using Provengo.

Test Automation



Test suite ready? Now execute it using our many automation options, including Selenium, Playwright*, REST API, Email (IMAP), Text Messages (SMS), AS/400 (TN5250), and command-line execution.

Provengo allows you to combine different automations in a single test scenario, so you can kick off a process from a web interface, check that emails were sent using our IMAP integration, and validate that the back-end (or mainframe) got the messages using REST. All on the same test.

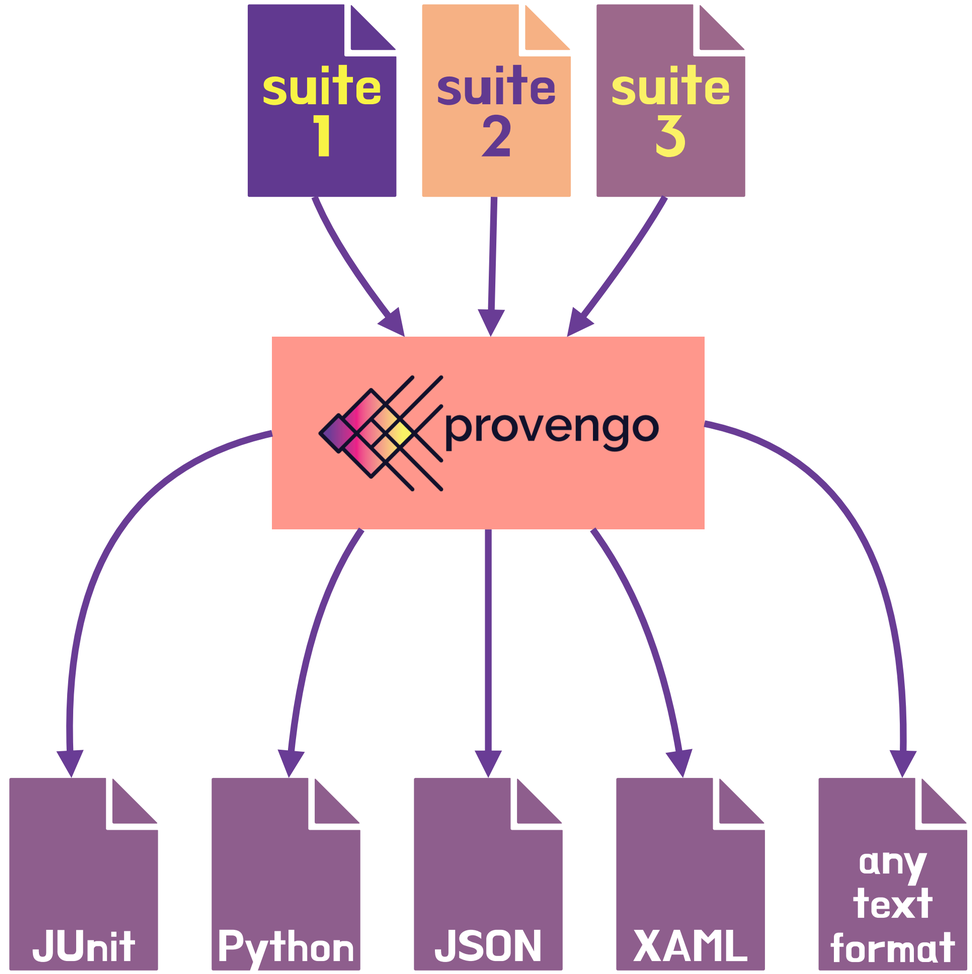
Need another automation connector? Talk to us, we're happy to connect!

* Coming Soon

Script Generation

Generate tests in Python, Java, Gherkin, JSON, XML... or any other text-based format.

Provengo allows users to convert test suites into files in any text-based format. This way you can use Provengo to generate test suites, and run these suites using your existing test infrastructure.



Testimonials

Provengo has confirmed again and again why we brought them on – their solution is extremely valuable and has shown that it can meet our needs and help us reduce development costs and time while improving our coverage. I strongly believe in Provengo and their ability to make a dramatic change in the development world.

***Mikhail Korman**, CTO,*
Ayalon Insurance

The Provengo concept of Model Based Testing for the masses is the way of the future. It reflects tools and languages that allow to reach complete coverage, Risk Based Testing, easy integration with the new AI Era, explainability and relatively easy test maintenance. Provengo is on the right path to improve and change how QA is done.

***Tamir Zano**, CTO DevOps & Automation Manager,*
Ness Technologies

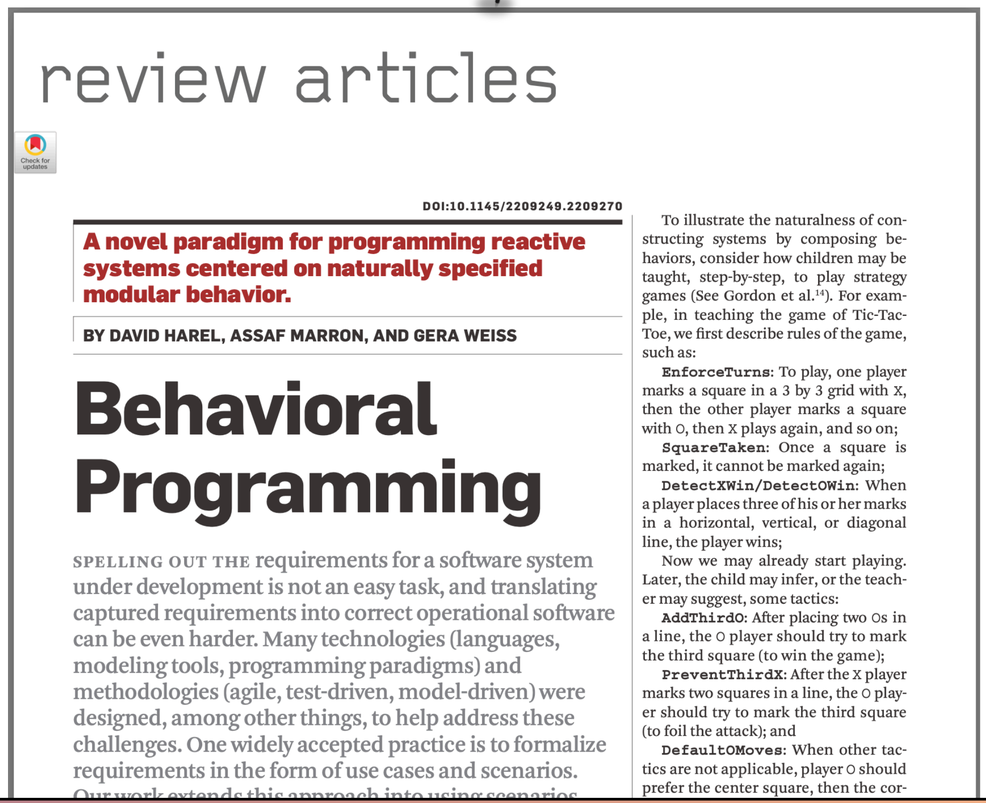
Solid Scientific Foundations

Provengo is based on *Behavioral Programming*, a novel executable modeling paradigm developed by David Harel, Assaf Marron, and Gera Weiss* circa 2010. Using this paradigm, users create intuitive descriptions of complex systems by combining multiple simple behaviors. A lot of academic and field work has gone into Behavioral Programming since the original papers were published, including notable contributions from members of the Provengo founding team.

Provengo is proud to be the first company to bring Behavioral Programming-based tools to mainstream markets

* One of Provengo's founders

Read the original 2012 paper, published on *Communications of the ACM*, here: https://doi.org/10.1145/2209249.2209270



review articles

Check for updates

DOI:10.1145/2209249.2209270

A novel paradigm for programming reactive systems centered on naturally specified modular behavior.

BY DAVID HAREL, ASSAF MARRON, AND GERA WEISS

Behavioral Programming

SPELLING OUT THE requirements for a software system under development is not an easy task, and translating captured requirements into correct operational software can be even harder. Many technologies (languages, modeling tools, programming paradigms) and methodologies (agile, test-driven, model-driven) were designed, among other things, to help address these challenges. One widely accepted practice is to formalize requirements in the form of use cases and scenarios. Our work extends this approach into using scenarios

To illustrate the naturalness of constructing systems by composing behaviors, consider how children may be taught, step-by-step, to play strategy games (See Gordon et al.[14]). For example, in teaching the game of Tic-Tac-Toe, we first describe rules of the game, such as:

EnforceTurns: To play, one player marks a square in a 3 by 3 grid with X, then the other player marks a square with O, then X plays again, and so on;

SquareTaken: Once a square is marked, it cannot be marked again;

DetectXWin/DetectOWin: When a player places three of his or her marks in a horizontal, vertical, or diagonal line, the player wins;

Now we may already start playing. Later, the child may infer, or the teacher may suggest, some tactics:

AddThirdO: After placing two Os in a line, the O player should try to mark the third square (to win the game);

PreventThirdX: After the X player marks two squares in a line, the O player should try to mark the third square (to foil the attack); and

DefaultOMoves: When other tactics are not applicable, player O should prefer the center square, then the cor-

Resources

Provengo's main site

https://provengo.ai

Reference Documentation

https://docs.provengo.tech/

Tutorials

https://provengo.github.io/Tutorials/

Sample code repo

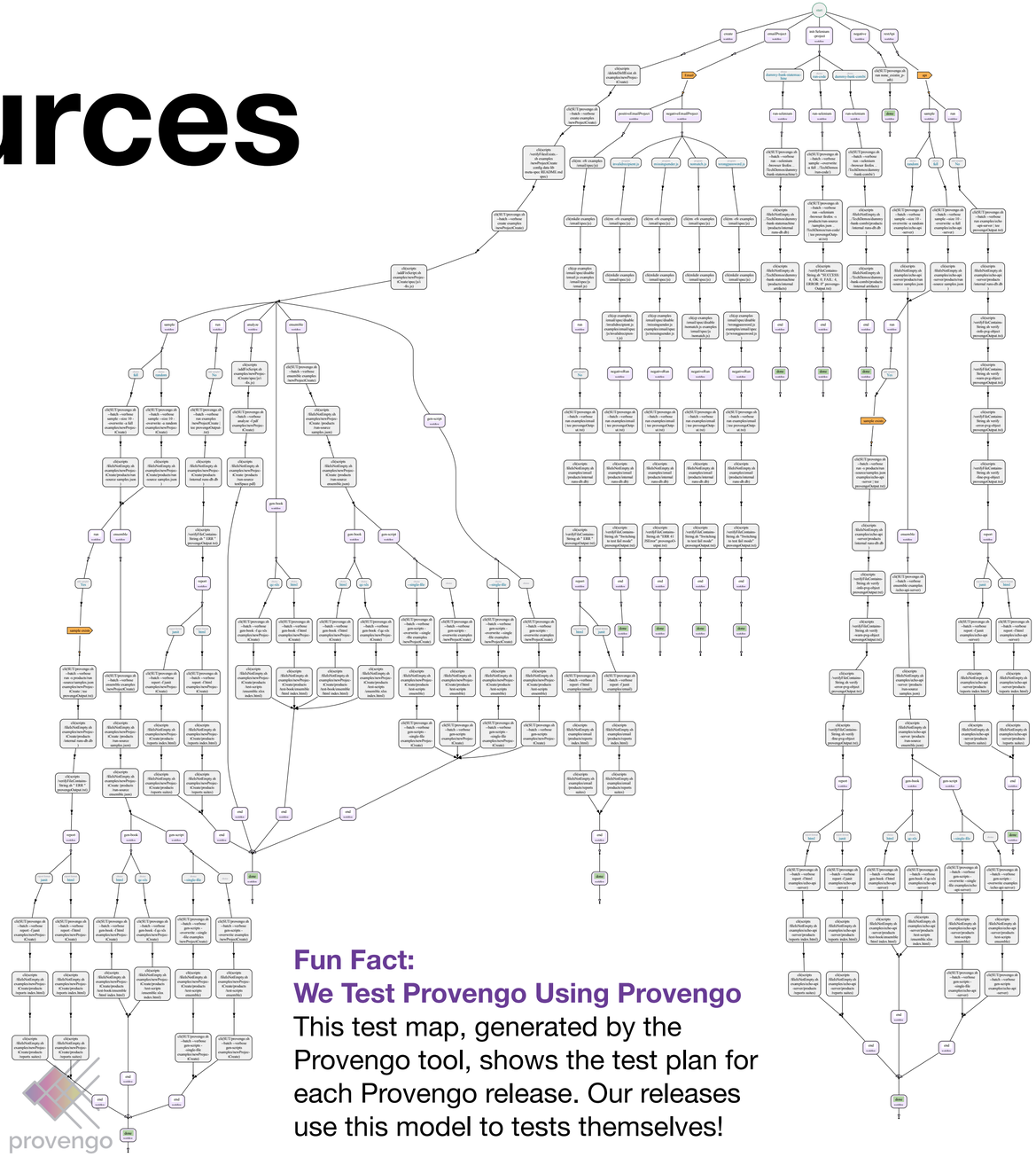
https://github.com/Provengo/TechDemos

Twitter/X account

https://twitter.com/ProvengoTech

YouTube channel

https://www.youtube.com/@provengo/



Fun Fact:
We Test Provengo Using Provengo
This test map, generated by the Provengo tool, shows the test plan for each Provengo release. Our releases use this model to tests themselves!

https://provengo.ai

hello@provengo.ai