Testing *at the speed* of AI.

AI agents handle the repetitive work. Testers stay in control.

Quality Assurance
Gabriel Canache, QA Engineer, Centric

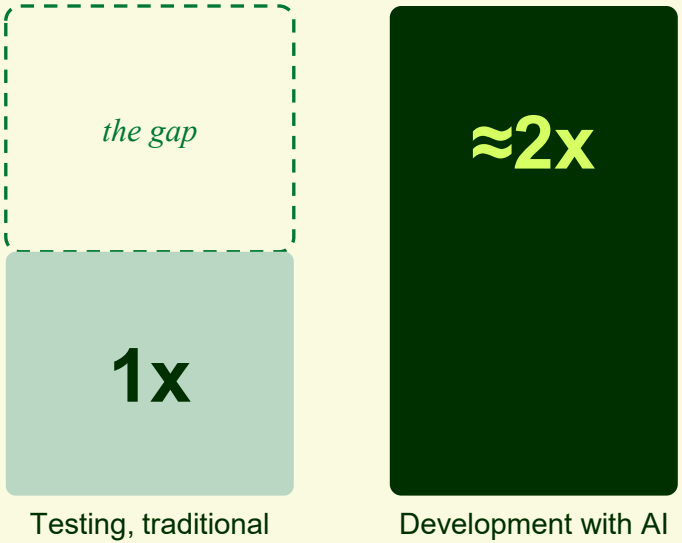
centric

Agenda
Today's session

1. When Development Speeds Up, Testing Must Too
2. A New Way to Test
3. The Toolbox: MCP Servers for Testing
4. Meet the Squad: A Proposal for AI Agents
5. Making Your Agents More Efficient
6. What Can Go Wrong
7. Live Demo

centric

When development speeds up, testing must too.



Illustration, based on our own experience.

1. **Developers deliver more in each sprint.**
   In our team, development output went up close to 2x.

2. **Traditional testing cannot keep the pace.**
   The same manual steps are done at the same speed.

3. **Testing becomes the bottleneck.**
   Quality work now waits for testing capacity.

We need AI for testers, too.

centric

The traditional testing flow.

5 of 7 **steps are done by hand, by a tester.**

This worked well, as long as development and testing moved at the same speed.

centric

A new way to test.

Agents do the repetitive work

- Turn requirements into a test plan
- Explore standard scenarios in the live app
- Report confirmed bugs
- Write the test cases
- Automate UI tests

Testers focus on what matters

- Complex, business-critical scenarios
- Scenarios the AI did not think of
- Reviewing and deciding at the human gate
- Showing the real value of testing

Not one big AI. *A squad of small agents, each with one narrow job.*

centric

The toolbox: MCP servers for testing.

What is MCP?

Model Context Protocol, an open standard.

Think of a USB port for AI. Plug in a server, and the AI can use its tools.

Not the model.
The context.

Our rule: if a required server is missing, the agent stops. It does not improvise.

- **Work tracker**
  Work items, test plans and test cases
- **API contract**
  Endpoints, requests and responses
- **Database, read-only**
  Tables, columns and relations
- **Chrome DevTools**
  Debugs the live app: network, console, speed
- **Playwright**
  Drives the app in the browser
- **Designs, optional**
  Screens and flows from a design tool

centric

Why context matters.

The code looks right. But it can test an API that does not exist.

Without context, the AI guesses

- Wrong paths and field names: tests fail for the wrong reason
- Weak assertions: only the status code is checked
- Test data stays in the database
- Hours of manual fixing

With MCP servers, the AI asks

- Requests match the real contract
- Assertions on body, types and status codes
- Cleanup deletes data in the right order
- Up to date, if the contract is generated from the code

The model is not the problem. The missing context is.

centric

Work tracker MCP.

Examples: Azure DevOps MCP or Jira MCP. In Jira, test cases need an add-on, like Xray or Zephyr.

What it gives the AI

- Read work items, acceptance criteria and links
- Create and update test plans, suites and test cases
- Create bugs and link them to work items
- Search by iteration or query

Who uses it

- Agents that plan: read requirements
- Agents that explore: read the test suite
- Agents that report bugs or write test cases: write to the tracker
- Agents that automate UI tests: find planned cases

Good to know

- Writes only after the human gate
- Load only the tool groups you need
- Official Azure DevOps server: cloud only. On-premises: community servers

centric

API contract: the map of your API.

OpenAPI (formerly Swagger). Two ways to give it to the AI.

Option A: a contract MCP server

- The AI asks small questions: list endpoints, get one endpoint, request schema, response schema, search
- Short answers: fewer tokens, less guessing
- Best for large APIs and teams that use it every day

Option B: a simple method

- Give the AI the contract file or its URL
- No setup. Good to start with
- A big contract fills the context: ask for one part at a time

centric

Database MCP, read-only.

Examples: servers for PostgreSQL, SQL Server, MySQL or Oracle. Ready-made or built in-house.

What it gives the AI

- Schemas and tables
- Columns, types and keys
- Foreign keys: the right order for cleanup
- A few sample rows, to learn the data format
- Read-only queries (SELECT)

Who uses it

- Agents that explore: check test data and cleanup
- Agents that automate UI tests: seed data, check cleanup

Good to know

- Main control: a read-only database user
- Second layer: a guardian rejects writes
- Connect to a test environment, not production
- Cleanup code runs in the tests, not in the server

centric

Chrome DevTools MCP.

Official, from the Chrome DevTools team. Answers the question: why did it happen?

What it gives the AI

- Explores the app like Playwright: navigate, click, fill forms
- Page snapshots and screenshots
- Network requests and console messages
- Performance traces, audits and throttling

Who uses it

- Agents that explore: main tool for scenarios
- Agents that explore: collect evidence for bugs
- Agents that explore: check failed API calls
- Agents that automate UI tests: only to debug

Good to know

- Works with Chrome only
- Sees tokens and cookies. So does the AI provider
- Runs locally, started by your AI assistant

centric

Playwright MCP.

Official, from Microsoft. Answers the question: what happened, and what do I do next?

What it gives the AI

- Opens pages and walks the flow
- Reads an accessibility snapshot: structured text with element references
- Click, type, fill forms, wait, switch tabs
- Chrome, Firefox, WebKit and Edge

Who uses it

- Agents that automate UI tests: find real steps
- Agents that automate UI tests: write the tests

Good to know

- Cheaper than screenshots. Big pages still cost tokens
- Keep sessions short: long flows fill the context

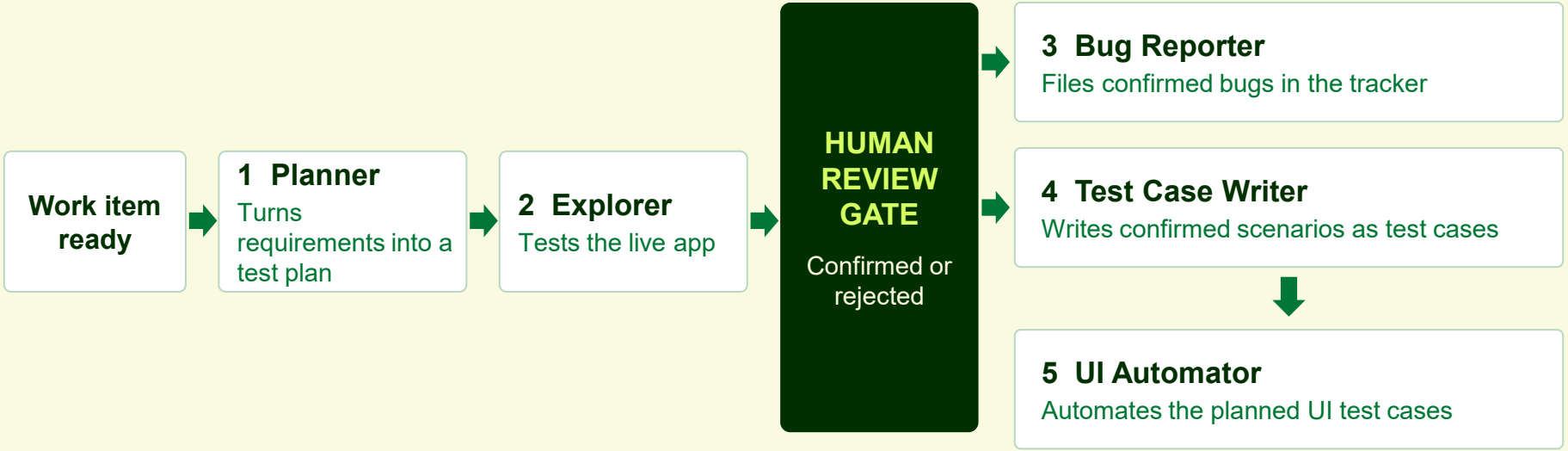
centric

Chrome DevTools or Playwright?

	Chrome DevTools MCP	Playwright MCP
Built for	Diagnosis: why did it happen?	Automation: do it, check it, repeat
Browsers	Chrome only	Chrome, Firefox, WebKit, Edge
Unique strength	Traces, Lighthouse, memory, throttling	Cross-browser. Same API as your tests
Best at	Speed, network and console problems	Walking flows and writing UI tests
In our testing	Exploring scenarios. Evidence for bugs. Debugging	Real steps and locators. UI tests
Rule of thumb	**Use it to explain.**	**Use it to do.**

centric

Meet the squad: a proposal for AI agents.



centric

Zoom in: the Planner and the Explorer.

1 Planner

Turns requirements into a test plan.

Reads

- The work item, acceptance criteria, designs and API contract

Writes

- A test plan with stable scenario IDs

Rules

- Never invents acceptance criteria
- Every scenario links to a requirement
- Critical scenarios, not many tests

2 Explorer

The only agent that tests the running app.

Tests

- UI scenarios, judged like a human tester
- API scenarios: wrong role, missing token, bad payload

Keeps it clean

- Synthetic test data only
- Checks cleanup. Stops if cleanup fails

Produces

- Results and one bug file per root cause, with evidence
- Never writes to the tracker

centric

The human review gate.

Bug file

Status: PENDING

Steps to reproduce

Actual result

Expected result

Evidence

HUMAN REVIEW GATE: the only approval point for findings

CONFIRMED
Goes on to the tracker

REJECTED
Never reaches the tracker

- A person opens every bug file and every scenario.
- Case and spaces are ignored.
- Missing or unclear status means not approved.
- Test code still goes through normal code review.

No AI finding reaches the tracker without a human decision.

centric

Zoom in: Bug Reporter and Test Case Writer.

3 Bug Reporter

Files confirmed bugs in the tracker.

1. Reads only confirmed bug files that are not filed yet
2. Checks the tracker for duplicates first
3. Reads recent bugs to copy the field conventions
4. Writes a short bug: steps, actual, expected
5. Links the bug to the work item and writes the ID back

4 Test Case Writer

Writes confirmed scenarios as test cases.

1. Rewrites each scenario to work on any environment
2. Skips duplicates and near-duplicates
3. Reuses the Planner's regression decision, and explains any change
4. Adds the case to the requirement's suite
5. Writes the ID back. UI test cases only, for now

centric

Zoom in: the UI Automator.

5 UI Automator

Automates the planned UI test cases.

Scope
- Planned test cases linked to the work item
- Skips cases that are already automated

Writes
- Walks the flow in the live app for real steps
- Team style: page objects, normally one expect per test

Runs
- New tests first, then the full suite once
- Each test creates and deletes its own data
- **Max 3 attempts per test. Never hides a failure**

centric

Seven principles for every agent.

Human in the loop
Nothing reaches the tracker without a human decision.

Single responsibility
One job per agent. No agent redoes another's decision.

Looks human-written
Matches the team's conventions.

Clean environment
Synthetic data only. Every fixture is cleaned and checked.

Security
Test credentials only. Read-only database user.

Continuous learning
Lessons files capture human corrections.

Fail loudly
A required server is missing? The agent stops.

centric

Documentation comes first.

Agents do not know your application. Your docs teach them.

1

Flows

Pages, routes and the steps a user takes. This is how an agent understands how the application works.

2

Business rules

What must always be true. Give each rule an ID, so every scenario can point to it.

3

Roles and states

Who can do what. Which states to cover: empty, error, locked, expired.

Without docs

The agent explores blind. It guesses the flow and spends many tokens.

With docs

The agent starts from the real flow. A wrong doc becomes a wrong test.

centric

Why use multiple agents?

1. **The right model for each job**
   A stronger model for hard thinking. A lighter model for simple, repeated work.

2. **The right effort for each job**
   High effort for planning and exploring. Low effort for filling a template.

3. **Fewer tokens**
   Each agent loads only its own skill and docs. No giant shared context.

Agent	Model	Effort
Planner	Strongest	High
Explorer	Strong	High
Bug Reporter	Lighter	Low
Test Case Writer	Lighter	Low
UI Automator	Balanced	Medium

Illustration only. Tune it for your own tasks.

centric

How an agent is built.

Keep each agent small. Let files do the talking.

WHO

Agent file

Who the agent is, and which tools and servers it may use.

HOW

Skill file

The playbook: inputs, workflow, output format, guardrails and examples.

HANDOFF

Files between agents

One folder per work item. IDs are written back, so re-runs are safe.

KNOWLEDGE

Read-only docs

Product knowledge, stored for search. Agents read it, and never edit it.

centric

Writing good skills.

A good skill has three parts

1. **Description**
   What the skill does, and when to use it.

2. **Expected output**
   The exact format of the result.

3. **Guidelines and rules**
   What to do, and what never to do.

Best practices

- Do not let AI write the skill alone. Read every line it writes
- AI adds rules you did not ask for. Remove them
- Start small. Add a rule after a real mistake
- Test with real examples. Fix the skill, not the output
- **Use references every time**

centric

Skills: use references.

Keep the skill short. Link to references. Say when to open each one.

Example: a bug-writing skill

```
bug-reporter/
    SKILL.md
    references/
        severity-guide.md
        bug-template.md
        good-examples.md
        field-conventions.md
```

Severity guide
How to choose the severity, with examples for each level.

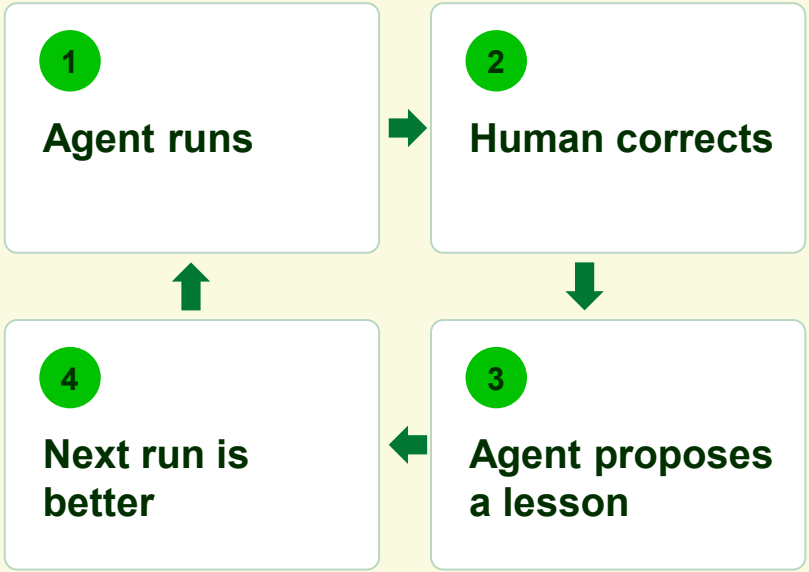
Bug template
How to write the bug: title, steps, actual, expected, evidence.

Good examples
Real bugs the team wrote, to copy the style.

Field conventions
Tags, areas and naming rules of the tracker.

centric

Agents that learn from corrections.



One lessons file per agent. A human approves each lesson. Agents read it at the start of every run.

What helps the agents

- Work items written with a high level of detail
- Main flows also designed, not only described
- Clear conventions that the agents can copy

Better context = less time, fewer tokens, better results.

centric

What can go wrong.

Risk	What we do about it
Agents are not perfect at the start	They learn: every human correction becomes a lesson
Some bugs may escape to production while the team gets used to the flow	The human gate, and testers focused on complex scenarios
The AI guesses without context	MCP servers give the real contract, data and designs. A missing server stops the agent
Test data stays behind	Synthetic data only. Cleanup is checked after every scenario
Unsafe access	Read-only database, no credentials in chat, writes only after the gate
Prompt injection: text in the app or a work item gives the agent orders	Least-privilege tools. Writes only after the gate. Test code is reviewed
A failure is hidden to get green tests	Max 3 attempts per test, then the agent reports. A failure is never hidden

centric

Live *demo.*

centric

Questions?

Quality Assurance
Gabriel Canache, QA Engineer, Centric

centric