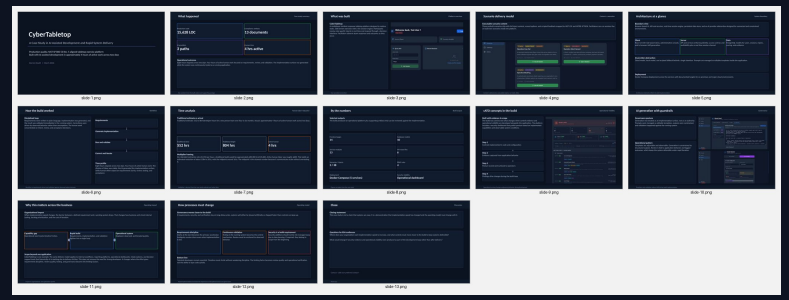
ATARC

Cyber Tabletop

A Case Study in AI Assisted Development and Rapid System Delivery

Building a production quality, NIST SP 800 53 Rev. 5 aligned tabletop exercise platform in approximately four hours of active work.



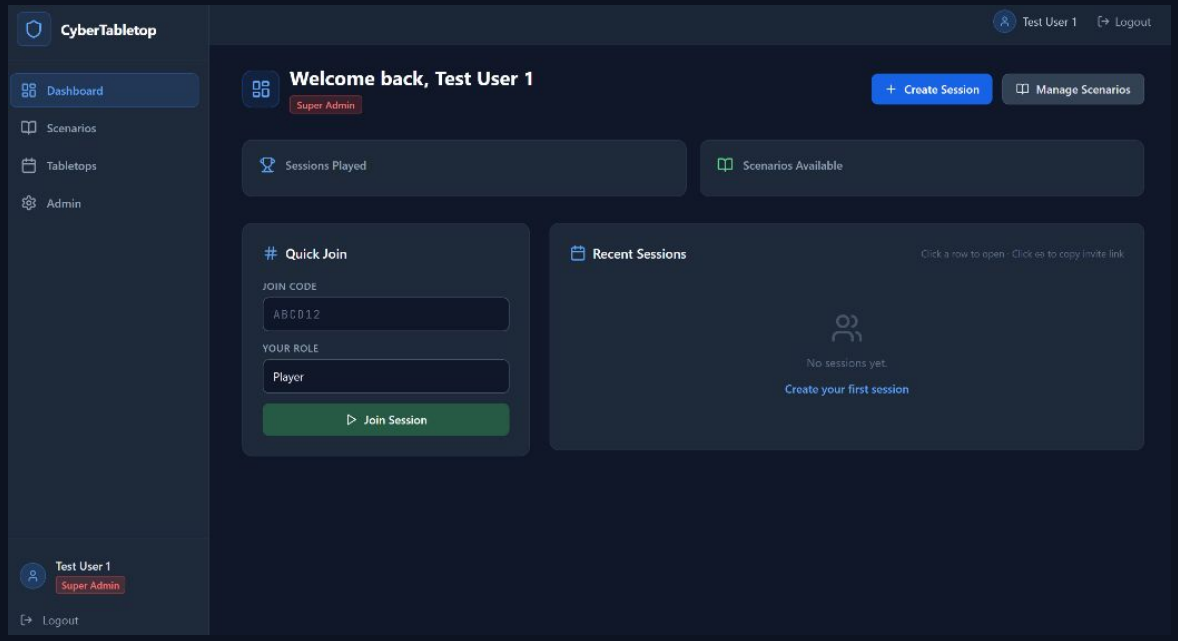
15,628	13	3	4 hours	2 days
lines of production code	NIST compliance documents	AI provider paths	active human work	elapsed time

What was actually built

The result was a working system, not a proof of concept artifact.

The finished system was running, deployed, and documented. It included the application, service layer, authentication stack, real time session handling, reporting functions, administrative interfaces, and an operational security dashboard that graded the application against implemented capabilities and observable conditions.

The documentation package was generated against the actual implementation, which meant the platform could be reviewed as an operational system with a defined boundary, a control structure, and supporting evidence.



The significance of the experiment

I used the build to test AI assisted development and cATO concepts at the same time.

Application build

Could a real internal application be built to a meaningful level of maturity with AI doing most of the implementation work?

Security and documentation

Could security architecture, documentation, and control evidence be built as part of development instead of after it?

Operational visibility

What changes when the system includes a dashboard that grades security posture while the application is still taking shape?

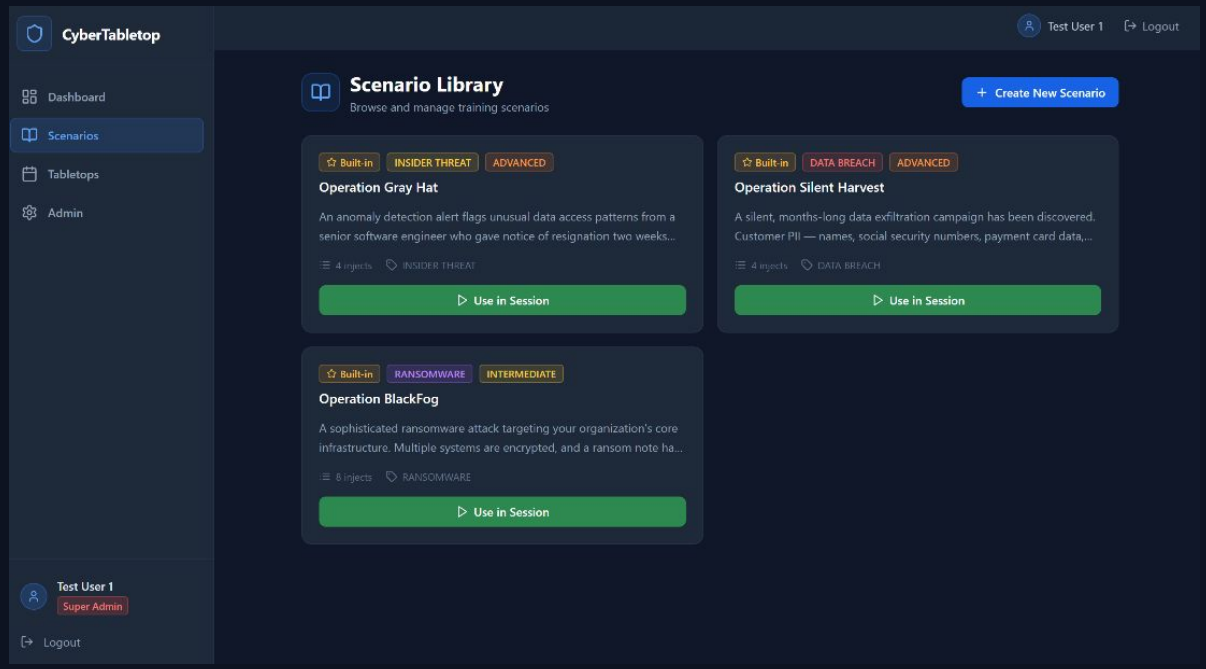
This made the project more than a build exercise. It became a test of how the development process changes when delivery, security, and evidence generation move together.

What the application does

CyberTabletop turns incident response exercises into a live, role based system.

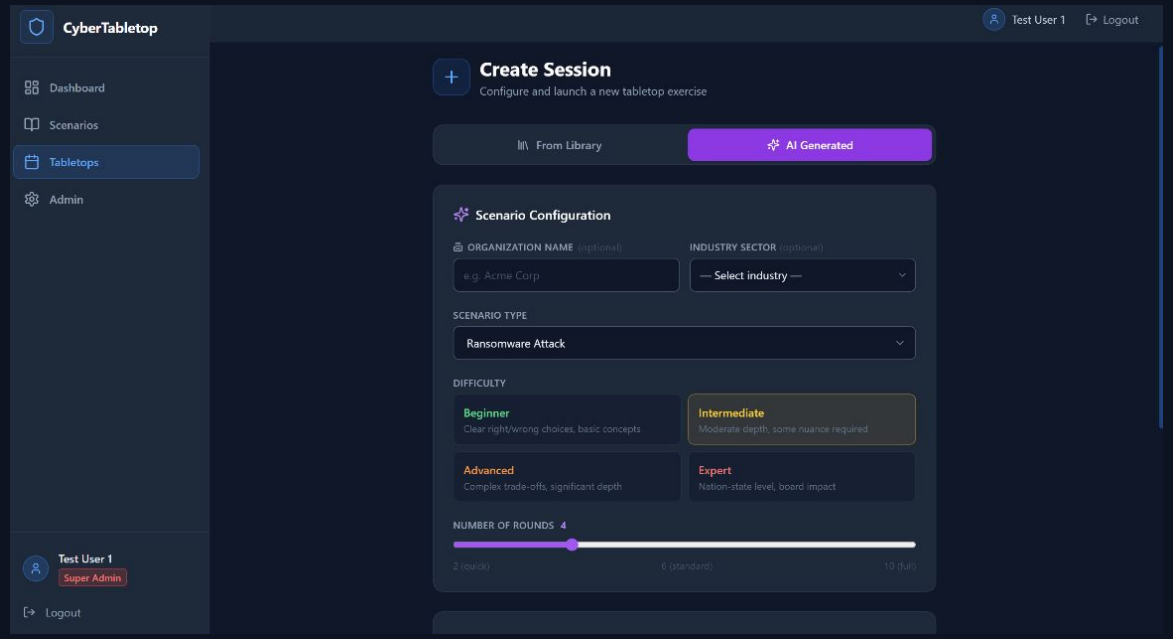
The platform supports live multiplayer sessions where participants join from separate devices, receive role specific injects in real time, and make decisions through a click based interface. Each response is scored and can produce immediate AI generated feedback tied to recognized cybersecurity frameworks.

At the end of an exercise, the system generates an in browser After Action Report. Facilitators can also create and modify scenarios through a built in scenario builder instead of through code changes.



What shipped in the initial build

Three scenarios, thirty eight injects, operational reporting, and multiple generation paths.



Initial release scope

Three prebuilt scenarios shipped with the platform: ransomware, PII exfiltration, and insider threat. Together they contained thirty eight decision injects with scored response options and scripted feedback tied to realistic operational consequences.

The platform also supported a cloud model, a local model for controlled environments, and a scripted fallback path so scenario generation could continue even when live generation was unavailable.

How the build actually worked

The elapsed time was about eight hours. The active human work was about four.

What I did

Defined requirements, reviewed generated code, tested the running application, and described defects precisely enough for the implementation to be corrected.

What the AI did

Generated the implementation across the codebase, maintained structure across files, and iterated rapidly when bugs or requirement changes were identified.

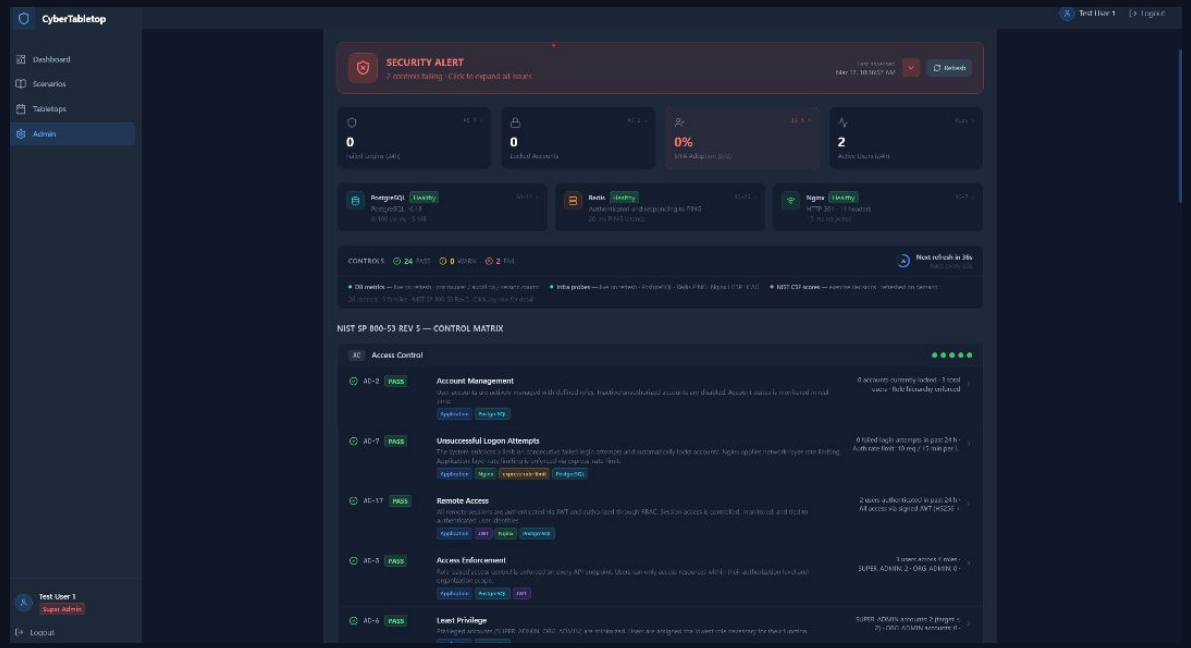
Outcome

The human work shifted away from writing every component directly and toward intent, review, architecture judgment, testing, and acceptance.

The workflow only worked because the requirements were specific, the review cycle stayed immediate, and the output was validated against the actual application instead of being accepted at face value.

Security and cATO were part of the build

The application, the evidence structure, and the operational view were developed together.



I did not treat security documentation and operational visibility as downstream deliverables. The platform included a structured security model for authentication, authorization, auditability, and traceable user activity, and it generated documentation against the actual system architecture.

The operational security dashboard graded the application against implemented capabilities and observable conditions. That changed the build because it pushed the system toward clearer control boundaries, stronger traceability, and better discipline around what had actually been implemented.

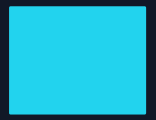
Time analysis and labor reduction

The traditional estimate and the actual active human effort were not in the same range.

Traditional estimate

552 to 804 developer hours

Active human effort



Approximately 4 active hours

170x

midpoint reduction in direct human labor

Using the midpoint of the traditional estimate, 678 hours divided by 4 active human hours yields approximately 170x. At $150 per hour, the traditional build range is about $82,800 to $120,600, compared with roughly $600 of active labor.

What changes across the business

The effect does not stay inside engineering or cybersecurity.

Teams can build more of the system at once

CyberTabletop was not built in isolated phases. The application, security model, documentation, and operational visibility were delivered together.

Leaders can evaluate working systems earlier

Program and mission teams do not have to rely only on requirements documents, backlog estimates, or vendor claims before making decisions.

Backlog economics start to change

Capabilities that used to look too slow, too expensive, or too difficult to justify can now be built and tested directly against the actual use case.

How development processes need to change

Implementation speed is increasing. Governance and expectations have to move with it.

Organizations still need strong developers. What changes is where their time is spent. The bottleneck shifts away from typing code and toward architecture, review, testing, and validation against a running system.

Timelines have to shrink. A process built around long implementation cycles, delayed review points, and late stage security involvement will not hold up when working capability can be assembled in hours or days.

Security additions should not be risk managed away because they were not planned for. If a control is necessary for secure operation, then it is a build requirement. Faster development should remove excuses for deferring security, not create new ones.

What comes next

The larger point is the delivery model, not whether this specific platform becomes a product.

CyberTabletop was built as an experiment to understand what AI assisted development can produce when the requirements are clear, the review process is disciplined, and security, documentation, and operational visibility are included from the start.

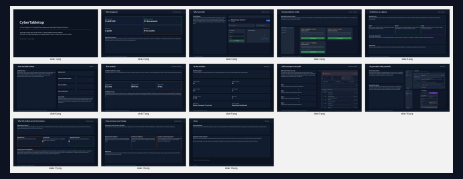
I may continue developing it, or I may apply the same approach to other use cases. Either way, the value was in testing the capability and understanding its effect on the development process.

The practical takeaway for organizations is simple: once requirements are clear and the build is being led by someone who can direct the work, evaluate the output, and validate the system as it takes shape, the distance between identifying a need and delivering a functioning application is much smaller than it used to be.

That applies well beyond cybersecurity to workflow tools, reporting platforms, dashboards, intake systems, and other internal applications that have historically been left in backlog.

Key takeaway

AI assisted development changes more than coding speed. It changes how quickly a disciplined team can move from clear requirements to a running system with security, documentation, and operational visibility built in from the start.



CyberTabletop demonstrates the delivery model. The application is the example, not the limit.

Darren Death
CISO, CPO, DCAIO