From Waterfall to SAFe: How Software Actually Gets Built

Fri, Sep 18, 10:00 AM EDT · https://scottslab.io/posts/software-development-methodologies

TL;DR

Waterfall, spiral, and agile aren't "more" or "less" secure than each other. They attach security work to different points in the process, and each has a specific failure mode when nobody accounts for that. Waterfall's own inventor called the strict linear version "risky and invites failure" back in 1970, which is worth knowing given how it's still cited as the disciplined option. Agile's speed is real, but "we're agile so we don't document" quietly deletes the paper trail regulators and incident responders both need. And the reason regulated, safety-critical software still runs something waterfall-shaped isn't nostalgia. It's that certification demands a document chain agile has to work harder to produce.

Before the methodology, know the terrain, because where software runs changes the threat model more than how it was built. Endpoint software runs on a user's machine; client-server splits logic across a network; web apps live behind a browser and a server; mobile adds app stores and device constraints; embedded and IoT run as firmware on hardware you often can't patch in the field. A web XSS bug and a firmware flaw you can't push an update for are not the same category of problem, so pin down the environment before you argue about process.

Waterfall's origin is more interesting than its reputation

Waterfall gets treated as the strict, document-heavy, no-going-back option, and it's usually blamed on Winston Royce's 1970 paper "Managing the Development of Large Software Systems." Read the paper and the joke is on the reputation: Royce described the purely sequential version (requirements, design, build, test, deploy, no backtracking) as "risky and invites failure," specifically because testing only happens at the very end, after every earlier mistake has had months to compound. His actual recommendation was to do it twice if you can afford to, keep the customer involved throughout, and let testing feed back into design. Almost none of that made it into how the industry actually used his diagram. The rigid version people mean when they say "waterfall" is a simplification of a paper that argued against rigidity.

That history matters for security because it explains waterfall's real strength: each phase produces a document, and a document is something a security requirement, a threat model, or a test plan can attach to. Security requirements live in the requirements phase. Architecture review lives in design. Security testing is its own gate before deployment, mirrored against the earlier phases in what's usually drawn as a V rather than a straight line: each verification stage checks against the design stage it corresponds to. That's exactly why regulated and safety-critical software still runs something waterfall- or V-shaped: aviation software certified under DO-178C and medical device software under IEC 62304 both require objective evidence that traces every requirement forward to a test result and every test backward to a requirement. That's a document-chain requirement, not a personality trait of the industry, and a fixed sequence of phases is the easiest way to produce it. Agile teams can build that traceability too, but it takes deliberate tooling and discipline the framework doesn't hand you for free, which is why the path of least resistance in a certification-heavy shop still looks like Royce's diagram, a quarter-century of agile advocacy later.

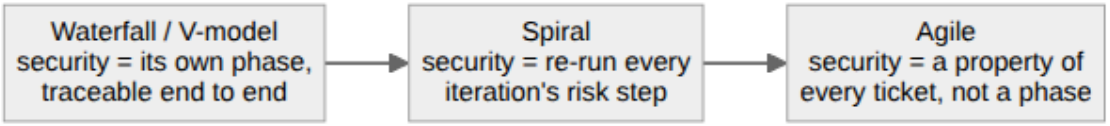
Spiral put risk analysis on a loop

Barry Boehm's spiral model (1986, with a wider-audience version in 1988) answered waterfall's rigidity by turning the process into a loop you cycle through repeatedly: determine objectives, evaluate alternatives and resolve risk, build and verify the next piece, plan the next pass. The part worth keeping is that risk analysis is a *named, recurring step*, not a one-time exercise at the start. Applied to security, that's the natural home for re-running your threat model every iteration instead of once at kickoff and never again. The spiral doesn't just tolerate revisiting your assumptions, it's built around doing exactly that on a schedule.

Agile moves the target, and that has a cost

Agile, codified by seventeen people at a ski resort in Snowbird, Utah in February 2001, and boiled down to the Agile Manifesto, trades the document chain for speed and continuous customer feedback, and it's how most teams actually work now. The trade-off is real: nobody hands you a

requirements document to attach a security review to, because requirements are explicitly allowed to change mid-project. That's a feature for delivery speed and a liability for security unless you replace the missing phase boundary with something else, usually a security checklist baked into the team's definition of done, so "secure" is a property every ticket has to satisfy rather than a gate the whole project passes through once.



"We're agile so we don't document" is the sentence that turns a legitimate speed advantage into a security problem. It isn't that agile can't produce a threat model, a security requirement, or an audit trail. It's that nothing in the framework forces you to, the way a phase gate does. A control that isn't written down is a control an incident responder can't find at 2am and an auditor won't believe existed. And a requirement that's allowed to move mid-sprint means a threat model done once at kickoff is stale the moment the design changes underneath it. The threat model has to move with the requirement, not sit frozen in a document from week one.

Scaling it up

Two more show up once an org outgrows a single team. Integrated Product Teams, introduced across the U.S. Department of Defense in 1995 as part of a deliberate shift in how it acquires software, put every relevant discipline in one room to make decisions in parallel instead of passing work down a chain. And the Scaled Agile Framework exists because plain agile strains at enterprise size; it's organized into four configurations (Essential, Large Solution, Portfolio, and Full) so an organization can adopt only as much structure as its scale actually requires, rather than importing all of it on day one.

None of these methodologies is secure by default, and none of them is insecure by default either. Each one just decides, structurally, where security has to attach: a phase, a recurring risk step, or a property of every unit of work. The failures show up exactly where that structure got skipped.

Written by Scott S. — Contact: scott@scottschlangen.com — scottslab.io — https://scottslab.io/posts/software-development-methodologies