



Goodman
TECHNICAL DILIGENCE

— A FREE TOOLKIT FOR DEAL TEAMS & FOUNDERS

The Technical Red Flag Checklist

A twenty minute self screen for spotting technical risk before you commit resource, or before you sit down with someone who will.

Code Quality

Technical Debt

AI

Architecture

Security

Team

6

Categories covered

30

Direct questions

~20 min

To work through it

BEFORE YOU START

How to use this checklist

This is a compressed version of the same six category framework I use in paid engagements. Code Quality, Technical Debt, AI, Architecture, Security, and Team. It won't replace a proper read of the code, but worked through honestly, it takes about twenty minutes and tells you whether a target is worth taking further, or whether you need someone to actually open the codebase first.

- 1 Work through it with whoever runs engineering** at the target, or apply it yourself if you already have repo access. Six categories, five or six questions each.
- 2 Score honestly.** This only works if you're not marking your own homework. A defensive or vague answer to a direct question is itself a data point. Note it as a flag.
- 3 Tally red flags per category** and use the scoring guide on page 8 to turn your answers into a plain verdict. Proceed, proceed with conditions, or get a proper technical read before you go further.

What "red flag" means here. Not every "no" is a deal breaker. The same gap can be a non issue in a slow moving internal tool and a serious problem in something processing card payments or health data. Weight each answer against what the business actually does and how much is riding on it.

THE SIX CATEGORIES, NEXT FOUR PAGES

01 Code Quality p.03
Is it well written, and safe to change?

03 AI p.04
Is AI actually shipping, or just being claimed?

05 Security p.05
What happens if something goes wrong?

02 Technical Debt p.03
Do they know what they owe, and to whom?

04 Architecture p.04
Built to grow, or just built to survive?

06 Team p.05
How dependent is it on any one person?

<> 01 · Code Quality

Is the code well written, and safe to change?

Code that's hard to change safely is a business risk, not just an engineering opinion. Every roadmap item after close moves slower and costs more than the deck implies.

- ☐ Is there an automated test suite, and does it actually run in CI, not just exist in the repo?
- ☐ Can they point to one recent feature that shipped without breaking something else?
- ☐ Could a new senior engineer navigate the codebase within a week, unassisted?
- ☐ Is there significant duplicated logic, the same thing solved three different ways?
- ☐ Has anyone besides the original author reviewed the core, business critical paths?

RED FLAGS IN THIS CATEGORY / 5

\$ 02 · Technical Debt

Do they know what they owe, and to whom?

Every shortcut has a bill attached. Every codebase carries some debt, so the question isn't whether it exists. It's whether anyone can actually tell you the size of it.

- ☐ Can engineering name their top three technical debt items, unprompted?
- ☐ Is there a plan, even an informal one, to address the most serious one?
- ☐ Are there parts of the system everyone privately avoids touching?
- ☐ Is the stack something the team can still hire for in this market?
- ☐ Has debt ever directly caused a missed deadline or a customer facing incident?

RED FLAGS IN THIS CATEGORY / 5

03 · AI

Is AI actually shipping, or just being claimed?

Nearly every target now says AI makes their engineering faster, or that AI is the product. Very few buyers can test either claim. This can't fully replace that test, but it's a start.

- ☐ Can they show specific commits or pull requests where AI tools were actually used?
- ☐ Was AI generated code reviewed by a human before merging, consistently rather than occasionally?
- ☐ Is there any policy on what proprietary or customer data can go into third party AI tools?
- ☐ If AI is part of the product, can they explain what happens when the model is wrong?
- ☐ Is there a costly dependency on a single AI vendor or model they couldn't easily unwind?

RED FLAGS IN THIS CATEGORY / 5

04 · Architecture

Built to grow, or just built to survive?

The question isn't whether the system works today at current load. It's whether it's built to grow, or whether it's one of those two things by accident.

- ☐ Could the system handle 10x current load without a rewrite?
- ☐ Can the team name their single points of failure, unprompted?
- ☐ Could a new engineer reason about the data model, or does it require tribal knowledge?
- ☐ Are third party integrations contained, or wired through the whole codebase?
- ☐ Does understanding of the system depend on one or two specific people?

RED FLAGS IN THIS CATEGORY / 5

 05 · Security

What happens if something goes wrong?

How secrets and access are handled, and what happens when something goes wrong, tells you more about engineering discipline than almost anything else on this list.

- ☐ Are secrets and API keys kept out of source control, with no hardcoded credentials?
- ☐ Is there rate limiting or hardening on sensitive endpoints such as login and password reset?
- ☐ Has there been a security review or penetration test in the last twelve months?
- ☐ Is there a clear incident response process if a breach happened tomorrow?
- ☐ Do departing employees actually lose access on their last day, provably?

RED FLAGS IN THIS CATEGORY  / 5 06 · Team

How dependent is it on any one person?

How the team is set up, and how dependent it is on any one person, is a risk you're buying whether or not it appears anywhere on the cap table.

- ☐ Would the system survive the departure of any single engineer next month?
- ☐ Does documentation exist beyond what's in one person's head?
- ☐ What has engineering turnover looked like in the last 12 to 18 months?
- ☐ Is hiring and onboarding a repeatable process, or ad hoc every time?
- ☐ Who actually holds the keys (infra, domain, admin accounts), and is that documented anywhere but their head?

RED FLAGS IN THIS CATEGORY  / 5

TURNING ANSWERS INTO A VERDICT

What your flags actually mean

Count the "no" or red flag answers within each of the six categories, then read across. This is a blunt instrument on purpose. It's designed to tell you when to stop and get a proper read, not to replace one.

0 to 1

Low risk

Nothing here should slow the deal down. Standard commercial terms, proceed as planned.

2 to 3

Moderate

Worth a note in the term sheet and a deeper look at this specific category before signing.

4+

High risk

Do not proceed on this category's strength alone. Get a technical read before you commit further.

One category outweighs the average. A codebase that scores well everywhere except Security, with 4+ flags there, is a worse bet than one with 2 flags spread evenly. Concentrated risk in one area is usually the one that actually detonates. Don't just sum the categories and average them out.

How a paid engagement scores it differently. In a real audit, every finding I raise carries three things. A severity, a cost and time estimate to fix, and whether it's a pre close condition or a year one budget line. That way it drops straight into your model or the price conversation, not left as an opinion for someone else to interpret.

YOUR SCORECARD, FILL IN AS YOU GO

Code Quality	____ / 5	Architecture	____ / 5
Technical Debt	____ / 5	Security	____ / 5
AI	____ / 5	Team	____ / 5

Overall verdict ☐ Proceed ☐ Proceed with conditions ☐ Get a technical read first

WHAT THE REAL THING LOOKS LIKE

A finding from an actual screen

This checklist gets you a plain "yes / no / flag it." A paid engagement gets you this. The exact line, the exact fix, and what it costs to sort before you sign.

```
auth/session.rb
```

```
- token = request.headers['x-api-key']  
+ raise Unauthorized unless hmac_valid?(request)  
// no rate limiting on password reset
```

Verdict. Fixable pre close, low effort. Flagged for the report.

Two lines, one comment, and a deal team now knows exactly what's wrong, exactly what fixing it looks like, and exactly where it sits on the pre close / year one line. That specificity is the actual product. A checklist can point you at the right questions; only reading the code gets you an answer like this one.

READ THIS BEFORE YOU RELY ON IT

What this checklist won't catch

A checklist tells you what to ask. It can't read the code for you, can't verify that an answer is accurate rather than optimistic, and can't catch the things a team doesn't know to mention.

-
- ✗ It can't tell you whether test coverage is real or theatre. Tests that exist but don't test anything meaningful.
-
- ✗ It can't verify that AI generated code was actually reviewed properly, only that someone says it was.
-
- ✗ It can't tell you whether the architecture holds up under the load you're planning to put through it in eighteen months.
-
- ✗ It can't put a number on any of it. How much a finding actually costs to fix, and by when.
-

That's the gap between a checklist and a proper technical read. Someone who has actually built and led engineering teams looking at the code itself, testing the claims against it, and putting a costed number on what they find, ready for the room where the decision gets made.

→ That's exactly what a Red Flag Screen or Full Review closes. See what's on offer on the next page.

ABOUT

Hands on senior engineering leader



I'm **David Goodman**. I've spent 20+ years in software engineering, the last 9 years leading teams of up to 50 engineers across fintech, cybersecurity and enterprise SaaS, from American Express through to fast moving scale ups and startups.

You're not hiring a faceless firm running a generic checklist. You're hiring someone who has built the systems and led the teams he's now being asked to judge, and who will be the one sitting in your investment committee call explaining what he found.

I've led AI transformation at organisation scale. Rolling out Claude Code and agentic development workflows, automated PR reviews, and coding standards across whole engineering, data engineering and data science functions, rather than talking about it in a slide deck. My first degree was in Artificial Intelligence, back in 2006, with an MSc in Information Technology since.

This checklist is the free edition of that same judgement, compressed into something you can run yourself in twenty minutes. The next page covers what happens when you want the full version.

WHAT COMES AFTER THE CHECKLIST

Want the full read?

Four ways in, depending on how far along you are. A quick screen before you commit resource, a targeted audit of the AI story, a full review for signing, or an ongoing technical voice for the portfolio. Every engagement is fixed fee, agreed in writing before any work starts.

EARLY STAGE

Red Flag Screen

A fast early check before you go too far into a deal. It flags anything that could be a serious problem.

ANY STAGE

AI Usage Audit

Tests what AI actually shipped in the codebase, whether it was reviewed, and what it really costs in debt.

FOR SIGNING

Full Review

The complete read across code, architecture, security and team, with costed findings ready for your IC.

ONGOING

Portfolio Technical Voice

A standing technical read across your portfolio, sized to your cadence, not mine.

hello@goodmandiligence.co.uk · goodmandiligence.co.uk · LinkedIn
linkedin.com/in/dgwebit

Screens turned around in days, sized to your deal clock, not mine.