

The first two days **on someone else's build.**

What to do in the two days before you change a line of an inherited codebase, and before you give anyone an estimate. For the developer picking it up, and for whoever is paying them.

Nothing here is a fix. The point of these two days is to know what you are dealing with, so the estimate is worth something.

BEFORE THE FIRST HOUR: WHAT TO ASK FOR

- ☐ **The repository, with its history**
Not a Zip of the files. The history is half of what tells you why the code is the way it is.
- ☐ **Every credential and environment variable, in a vault**
Each missing one is a third-party service you will find out about the hard way.
- ☐ **Hosting, domain and platform accounts, and who holds them**
If they are in a former employee's name, sorting that out comes before any code.
- ☐ **Half an hour with whoever built it, if they are still reachable**
The cheapest half hour in the project. Ask what they would have done next, and what they were nervous of.

HOURS 1 TO 4: GET IT RUNNING LOCALLY

- ☐ **The application runs on your machine and does something**
Everything else waits for this. No fixes yet, however tempting.
- ☐ **Every place the README differs from what actually happens is written down**
The size of that gap tells you roughly when the project stopped being maintained.
- ☐ **Every environment variable you had to ask for is on a list**
That list is often a better map of the site's dependencies than any document.
- ☐ **How long it took you to get it running is noted**
It is the fairest measure of the handover, and worth telling whoever is paying, early and kindly.

HOURS 4 TO 8: MAP THE SURFACE

- ☐ **Every route a human can reach, from the router, the CMS or the sitemap**
Then open a sample. Pages that quietly throw an error are where the unfinished work usually is.
- ☐ **Every third party the site calls, and what breaks without each one**
A site that falls over when an analytics script is slow carries more risk than one that carries on.

☐ **Where the data lives, and who else writes to it**

A database another system also writes to is a very common reason something works on staging and not live.

DAY 2, MORNING: FIND WHAT IS LOAD-BEARING

☐ **The file with the most recent changes and no tests**

Lots of change with no safety net is where the risk sits.

☐ **The function or module everything else imports**

You will lean on it eventually. Better to know where it is first.

☐ **Anything commented "don't", "careful" or "temporary"**

Somebody left a warning. Read it before you find out why.

☐ **Anything somebody was part way through when they left**

Half-finished work looks finished until it is used.

DAY 2, AFTERNOON: THE HONEST REPORT

☐ **What is genuinely risky, and will cause an outage, lose data or block launch**

This is the only list that has to be fixed.

☐ **What is a matter of taste, and works fine**

Keep it separate, or a preference ends up sounding like a risk.

☐ **What you could not work out yet, each with a cost to find out**

After two days there are always a few. Saying so is the honest part.

☐ **The report is sent before any number is given**

Any estimate made before this one is a guess with a decimal point.

ONLY THEN

☐ **An estimate, with the assumptions it rests on written next to it**

So everyone can see which assumption changed when the number does.

☐ **A rewrite is proposed only if the platform is genuinely at the end of its life**

Rewrites have to rediscover every small thing the old code already handles, and that is what they overrun on.

Take it, change it, use it on your own projects. No need to credit me. adamroe.co.uk/blog/taking-over-someone-elses-codebase