Floating Code	**HowTo Exactly?**	**Handcrafted Serializers**	**The Infinite Void of User Feedback**	**Orphaned Deployments**
The Everythingy	**Love of Threads**	**Failed Side Gig**	**Managed Memory Leaks**	**Oops, it's Async**
Also And Also And Also	**Fables from a Distance Past**	**What Tests?**	**Fables from a Distance Land**	**Load Bearing Conditional**
LLM Credit Farming	**The Islanders**	**It Should Build**	**Same Language Barrier**	**The Master Control Program**
Pedal Powered Release Train	**Universe Simulator**	**Handcrafted SQL**	**Using Horizontal Space**	**Fear of Retirement**
The Never Merge	**Stuck in the 80s**	**High Upkeep Copies**	**Fix Everything**	**Whatever Ben's Doing**

0.0: Floating Code

Code vital to the production system is not (or not entirely) stored in a CMS. Maybe an important patch exists only as an attachment to some e-mail somewhere.

0.1: The Everythingy

A single, enormous code module that contains the implementation details to do, pretty much anything the system can do. Bonus points: It's used by different parts of the code base, just to give the illusion that responsibilities are sensibly separated.

0.2: Also And Also And Also

A single conditional statement that checks at least 4 different components of local or system state.

0.3: LLM Credit Farming

When the LLM generates proper, functional code, I did it. When it screws everything up, it's the LLM's fault.

0.4: Pedal Powered Release Train

The effort of producing a new release requires the undivided attention of all or most team members for at least one full day. Bonus points: Developer machines are needed to produce some or all of the release artifacts. Bonus bonus points: Many manual actions are needed that are not documented anywhere.

0.5: The Never Merge

There's at least one long-running branch that should have been closed a long time ago. Its changes have diverged so significantly from the main branch that no one is eager to attempt the merge. Yet it contains vital changes that we definitely need, some day, for sure, I promise.

1.0: HowTo Exactly?

There's no documentation or user guides describing how the system is supposed to be used. Everyone involved simply already knows. Bonus points: If you ask them, you get inconsistent responses.

1.1: Love of Threads

Everyone loves threads, right? So any part of the code just spawns some whenever, without consideration for the system at large. Somehow, this hasn't solved any performance issues.

1.2: Fables from a Distance Past

Comments or commented-out code speak of a time when procedures implemented mythical behavior, certain concepts were held to be true, or a design was adhered to. We may never know if such tales were ever true or mere fiction.

1.3: The Islanders

Everyone has their own island inside the code. You could work on someone else's part, but comfort zones are comfortable.

1.4: Universe Simulator

The system is so flexible/configurable, it could in theory be used to construct a simulation of the universe. In practice, no one truly needs nor uses this level of flexibility.

1.5: Stuck in the 80s

The system uses technology that's been outdated for years, but it's so tightly coupled that the effort to modernize hasn't been started. Bonus points: The modernization effort has been started and abandoned more than half a year ago.

2.0: Handcrafted Serializers

We wrote our own serializer/deserializer for a well-known format, for which support definitely exists. Bonus points: We've actually done this to implement our own customizations to the well-known format, making our data incompatible with the wider world. Bonus bonus points: We realized that we didn't actually need those customizations, solved the problem a different way, but kept the custom serializers.

2.1: Failed Side Gig

A feature has been partially implemented but was never finished, integrated, or used. The code is still around, mixed in with the rest of the system.

2.2: What tests?

There are no automated tests, or there aren't nearly enough to cover vital functionality to a sufficient level of detail. Developers can't make changes to large sections of code without serious risk of A) breaking something vital, or B) introducing subtle bugs that won't be found for months.

2.3: It Should Build

The source code should build. Other people are working on it. Maybe it's being used in production already. But when you get the code from the repository, it doesn't build and there are no build instructions to fix this problem. After talking to the right person, you'll receive the secret wisdom of how to fix your build.

2.4: Handcrafted SQL

Screens upon screens full of string-manipulation code to create and/or parse your SQL statements. Bonus points: A nearby comment that mentions the word "performance".

2.5: High Upkeep Copies

State is duplicated in multiple places across the system. The code spends considerable effort trying to keep everything in sync. The reason for the duplication is unclear, so no one dares remove it.

3.0: The Infinite Void of User Feedback

No one in or near the team has any exposure to user feedback. We don't know how or even if the system is being used. Bonus points: We've got some very strong opinions about which features are definitely critically important to our users, though.

3.1: Managed Memory Leaks

Despite being in a memory-managed environment, the system still leaks memory over time. Bonus points: The issue is dealt with by restarting the system periodically. Everyone has accepted this as a fact of life.

3.2: Fables from a Distance Land

Variable names, function names, type names, and/or comments written in a language other than English. Bonus points of it mentions literal translations of the programming-language specific keywords.

3.3: Same Language Barrier

Even though everyone speaks the same natural language, we don't understand each other. We speak in technical detail with others in the org who don't have the background to understand it. Others in the org speak their domain-specific jargon to the software team.

3.4: Using Horizontal Space

A function or procedure that has four or more nested conditional scopes, putting that horizontal resolution of your monitor to good use.

3.5: Fix Everything

Pull-requests or change sets that touch more than 50 code files. Often they are bugfixes. Often they introduce at least one bug for each one they solve.

4.0: Orphaned Deployments

It is not possible to review the exact source code for a deployment that's currently being used in a production environment. The version/revision information is simply not available, or there's no way of knowing if it's correct or whether modifications have been made manually.

4.1: Oops, it's Async

The surprise appearance of locks, semaphores, mutexes. Any clear indication that some code was apparently running async or parallelized, and it needed to be stopped!

4.2: Load Bearing Conditional

A conditional statement that invokes a function or procedure which performs a serious part of the system's logic.

4.3: The Master Control Program

A single module is responsible for controlling every aspect of the system. This module is exposed to all the interfaces and data types and possibly has access to the detailed data fields of its subjects.

4.4: Fear of Retirement

One person exclusively knows everything, takes all technical decisions, assigns the work to other team members. We all live in fear of the day this person retires.

4.5: Whatever Ben's Doing

Team members don't know what their colleagues are working on, or how it's going. The on-going work isn't tracked anywhere anyone can see it, if at all.