2025-03-15

All Code Sucks

Joe J Collins
j.collins@zengenti.com
linkedin.com/in/joejcollins

Zengenti

21 March 2025

Thanks for taking the time to come out tonight.
I'm Big Joe from Zengenti.
We are a small company in Shropshire. About 70 nerds.
We do websites for universities and local authorities.
I actually don't do any websites, I work in the hosting team.
We maintain a private cloud to run the websites.
We use a combination of Ansible and Python
maintaining about 3000 servers.
But, tonight Matthew,
I am going to talk about, **why all code sucks**.

We all know good code,
or at least we think we do.
But I should probably define what I mean by sucky code.

"Programs are meant to be read by humans and only incidentally for computers to execute."
— Donald Knuth

What is sucky code?

My short answer is . . . **sucky code is hard to read**.
No need take my word for it.
Donald Knuth, the Yoda of Computer Science
says that code is for humans to read
and sometimes for computers to run.
He is all about the readability.

All Code Sucks

└The Great Hunt for Non Sucky Code

For about about 25 years now,
I have been looking for code that doesn't suck.
And trying to produce code that doesn't suck.
I've worked in companies large and small.
But pretty much, all the code sucked.

This does beg the question, why does it all suck?

All Code Sucks

- Half of everything is below average

└Why Code Sucks

On the whole I think the odds are against us.
Straight out of the gate,
half of everything is going to be below average.
Well, below the median.

All Code Sucks

- Half of everything is below average
- Sturgeon's Law

└Why Code Sucks

Then there is Sturgeon's Law.
Sturgeon was explaining why most science fiction is low quality.
And came up with the pithy answer
"90% of everything is shit".
The observation works here too.

All Code Sucks

- Half of everything is below average
- Sturgeon's Law
- The 3 Year Old Programmer

└Why Code Sucks

When I started out
the average programmers experience was 3 years.
25 years later that hasn't changed.
Probably (and fortunately) because demand outstrips supply.

So if the odds are against us
maybe the organizations we work for will help.
Or perhaps not.

- Software Startups

Why Code Sucks

The romantic image of a software startup is
a couple of guys in a garage.
I have actually see this quite a bit.
For most start ups the two guys are the dad and the son.
The dad is the salesman.
And the son is the programmer.
They are inherently under resourced.
Writing readable code is a luxury they can’t afford.

Why Code Sucks

The other kind of startup I have seen
occurs in big companies.
The summer student project.
Alternatively called the unsupervised use of new technology.
All the experienced programmers are on holiday or busy.
So they give the new technology to the summer students,
who give it a go.
And if it runs they put it into production.

- Software Startups
- Summer Student Projects
- Prototypes in Production

Why Code Sucks

This last point is also a general point.
Any software that appears to work goes into production.
Not because anyone thinks it's a good idea
but because there is a commercial imperative.
Having learnt from the prototype,
the plan was to throw it away and build it for real.
But that never happens.
It is always put into production.

Just in case you haven't lost all hope
I would say that suck
is actually built into human psychology.

All Code Sucks

Why Code Sucks

We are unaware of our ignorance.
Because we are ignorant of it.
We don’t know what we don’t know.
But writing code is about explaining things in detail.
So it is no surprise that we struggle.

All Code Sucks

- The Illusion of Explanatory Depth
- Availability Bias

Why Code Sucks

A couple of years ago we used a static analysis tool on our code.
To my surprise is said most of the code was good.
But it pointed to the five worst files.
Those were the files I spend most of my time working on.
Those will be the bits you remember.
So it turns out all code doesn't suck,
I just feels like it.

Nevertheless, makes sense to tool up for sucky code.

- **Discovery** Log behavior
- **Exploration** Scripted investigation
- **Automating** Test harnesses (or clamps)
- **Deconstruction** Separate the target code
- **Enable** Switching with a feature flag
- **New Code** Side by side rewrite

What To Do

I should say that this isn't my idea.
This is based on the work of Michael Feathers.
in his book **Working Effectively with Legacy Code**.
It is the definitive guide to working with sucky code.
But it is over 20 years old now.
But it is still the best guide I have found.

Also I have skipped the step
getting control of the development environment.
At Zengenti we use ephemeral environments
either Docker Containers or Virtual Machines.
But that is a whole other talk.

All Code Sucks

- **Discovery** Log behavior
- **Exploration** Scripted investigation
- **Automating** Test harnesses (or clamps)
- **Deconstruction** Separate the target code
- **Enable** Switching with a feature flag
- **New Code** Side by side rewrite

What To Do

Example, API with one end point, see the squiggles.

make app

Is is a valuable must preserve the behavior.

Open http://127.0.0.1:8000/docs

I will just mimic some behavior.

Try 2 and 42

The behavior isn't abstract, it is in context of use.

We need to discover how the end point is used.

demo01log

Feathers don't change, but sometimes you have to.

What To Do

Now we know how the end point is used.
We can start to explore the code.
Open tests/app.http
We use http scripts to mimic the behavior.
demo2http01
There's a Jetbrains client or a VSCode extension.
demo2http02
This allows us to do some scripted investigation.
Looks like a string is coming back.
Not a number.
Fix inverted commas
And to get a feel for the behavior.
We might start to get an insight into the author's intent.

What To Do
Adapted from Michael Feathers

- **Discovery** Log behavior
- **Exploration** Scripted investigation
- **Automating** Test harnesses (or clamps)
- **Deconstruction** Separate the target code
- **Enable** Switching with a feature flag
- **New Code** Side by side rewrite

What To Do

Using this knowledge we can write some automated tests.
Open tests/app_test.py
These are the clamps that will hold the code in place.
As Michael Feathers calls them.
We run these tests every time we make a change.
To make sure we haven't broken anything.
demo3test01
demo3test02
This will give us the confidence to make changes.
Every time we make a change we run the tests.
make test
We get a little report each time.
To let is know it is alright to continue.

- **Discovery** Log behavior
- **Exploration** Scripted investigation
- **Automating** Test harnesses (or clamps)
- **Deconstruction** Separate the target code
- **Enable** Switching with a feature flag
- **New Code** Side by side rewrite

What To Do

We the automated tests in place.
We are ready to start deconstructing the code.
Open app.py
We can start to separate the target code.
demo4function01
Since we don't really know what the code does.
We can't give it a meaningful name.
Michael Feathers suggests just mashing the keyboard.
foo and bar would be just as good.
We can check that it works in all it's ugliness.
By rerunning the tests.
make test

- **Discovery** Log behavior
- **Exploration** Scripted investigation
- **Automating** Test harnesses (or clamps)
- **Deconstruction** Separate the target code
- **Enable** Switching with a feature flag
- **New Code** Side by side rewrite

What To Do

Stay with me we are getting close.
Now we can set up a switch.
There are lots of tools to do this.
We favour Unleash because it comes for free with Gitlab.
Open /joejcollins/alan-tracy/-/feature_flags
Here I have a feature flag to turn a function on and off.
I can use this to switch between the old and the new code.
demo5flag01
make test
So let rewrite the code.
Cut and paste ChatGPT

"It's not that hard..."
— Billy Beane

"It's incredibly hard"
— Ron Washington

The Challenge

To quote Moneyball,
the challenge is both easy and difficult.
The example was one function and one file.
But the code we are working with has 3500 files.
Some is hard to read,
some has been improved and some it half way between.
But it often feels like we have three different code bases.
All jumbled up.

It turns out the improving code is easy
but managing the process is hard.
Knowing which bits to improve really hard.
I have no answer to that, I am kind of hoping you do.

Thank You!

Joe J Collins
Zengenti

Thank you for listening.
I’m Big Joe from Zengenti.
If you thought this was interesting,
and would like to work with us,
please get in touch.
If you think you have a better way to work with sucky code,
I’m all ears.
Or if you have an idea how to manage the process,
Let’s talk.