CodeScene

Treat Your Code as a Crime Scene

aka: Using forensic techniques to identify and prioritize technical debt, eliminate expensive change patterns, and visualize organizational risks in your code.

October 2024

@AdamTornhill

codescene.com

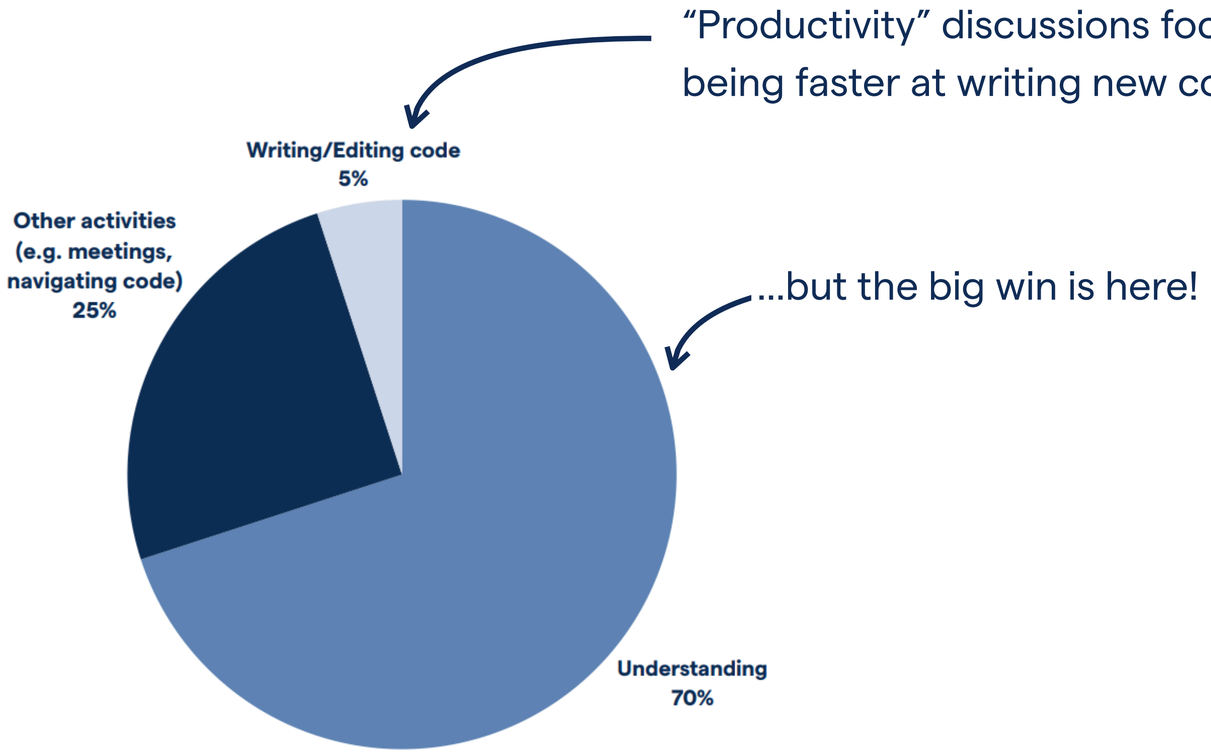
What’s our main challenge as software developers?

“A bad system will beat a good person every time”

W. Edwards Deming

Typing Isn’t the Bottleneck in Programming:

Optimize for understanding

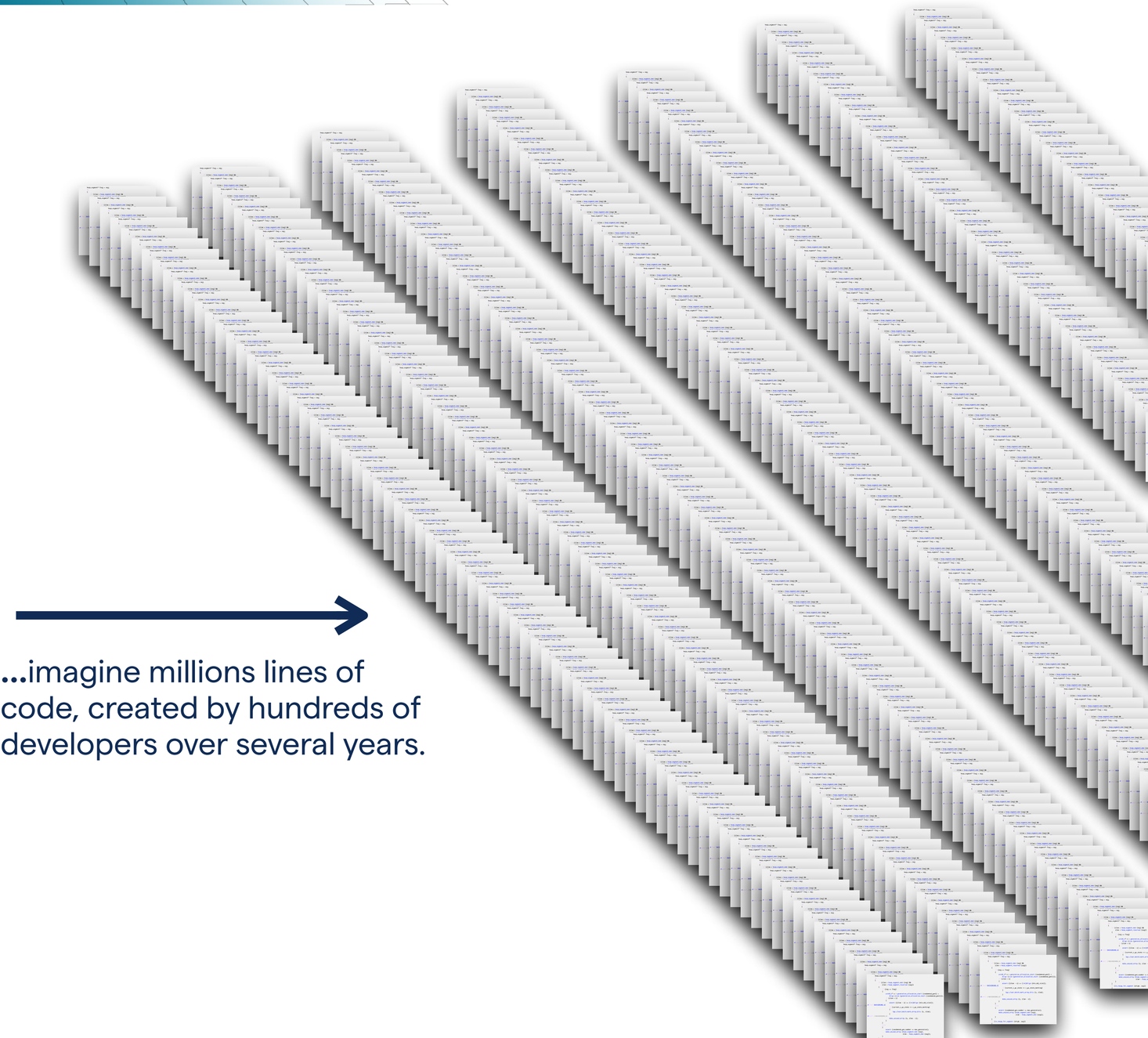


The majority of a developer’s time is spent trying to understand the existing system (data from Minelli, et. al., 2015)

Why Software is Hard

Is source code hard to understand?

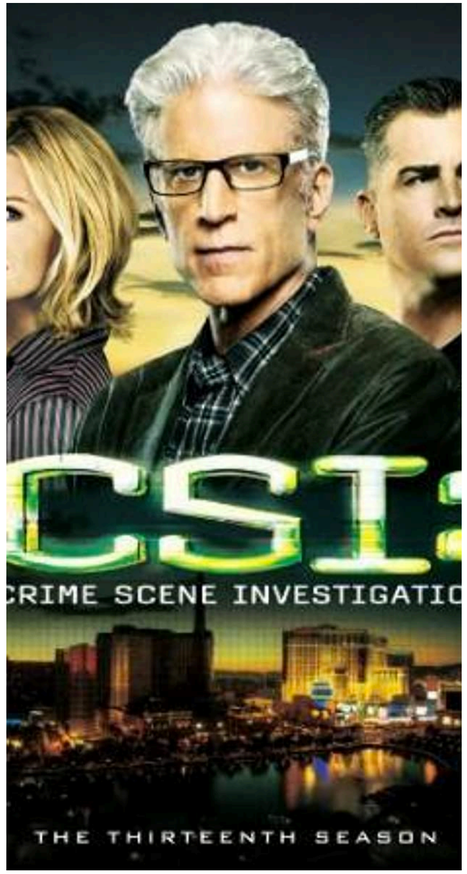
```
        heap_segment* fseg = seg;
        do
        {
            if (slow > heap_segment_mem (seg) &&
                slow < heap_segment_reserved (seg))
            {
                if (seg == fseg)
                {
                    uint8_t* o = generation_allocation_start (condemned_gen1) +
                        Align (size (generation_allocation_start (condemned_gen1)));
                    if (slow > o)
                    {
                        assert ((slow - o) >= (int)Align (min_obj_size));
#ifdef BACKGROUND_GC
                        if (current_c_gc_state == c_gc_state_marking)
                        {
                            bgc_clear_batch_mark_array_bits (o, slow);
                        }
#endif //BACKGROUND_GC
                        make_unused_array (o, slow - o);
                    }
                }
                else
                {
                    assert (condemned_gen_number == max_generation);
                    make_unused_array (heap_segment_mem (seg),
                                       slow - heap_segment_mem (seg));
                }
            }
            if (in_range_for_segment (shigh, seg))
            {
```



...imagine millions lines of code, created by hundreds of developers over several years.


Intuition Doesn’t Scale

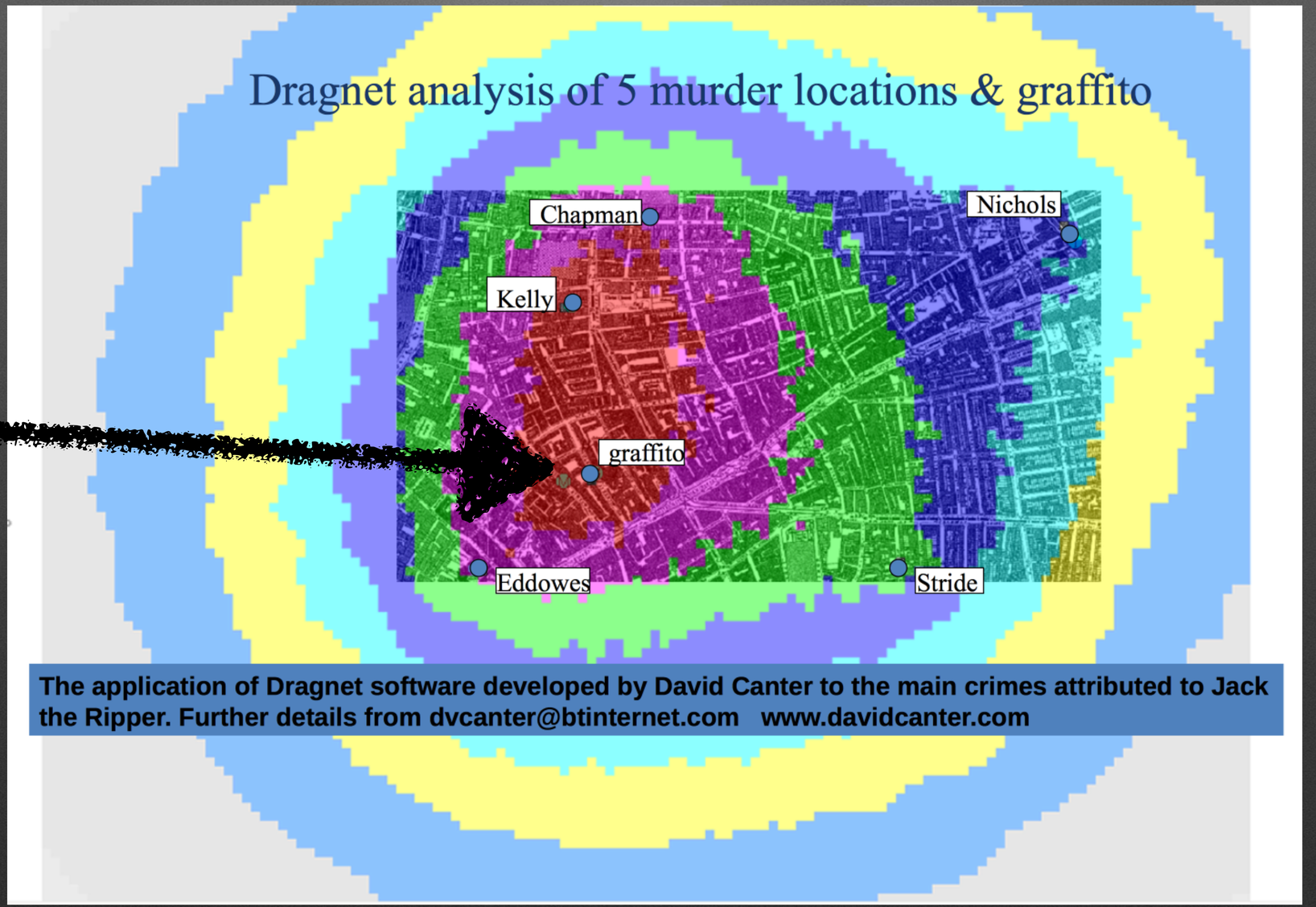
Forensic Psychology




HOLLYWOOD

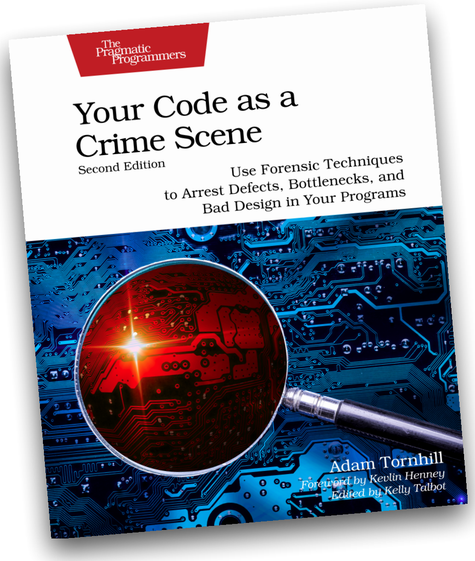

Geographical Offender Profiling
-
a 2 Minutes Introduction
@AdamTornhill

Profiling the Ripper




How can we use this in Code?

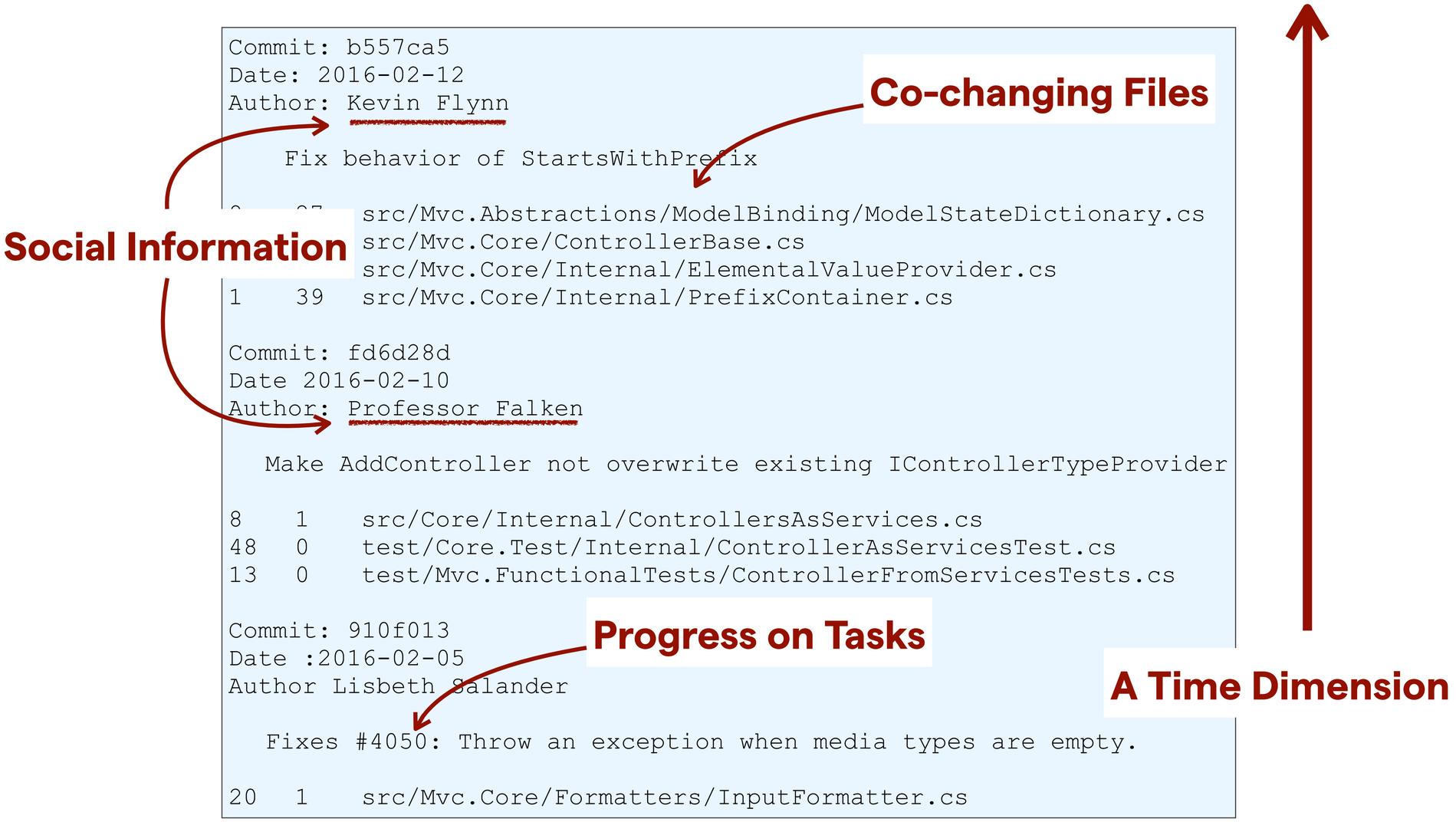
What is a **Behavioral Code Analysis?**



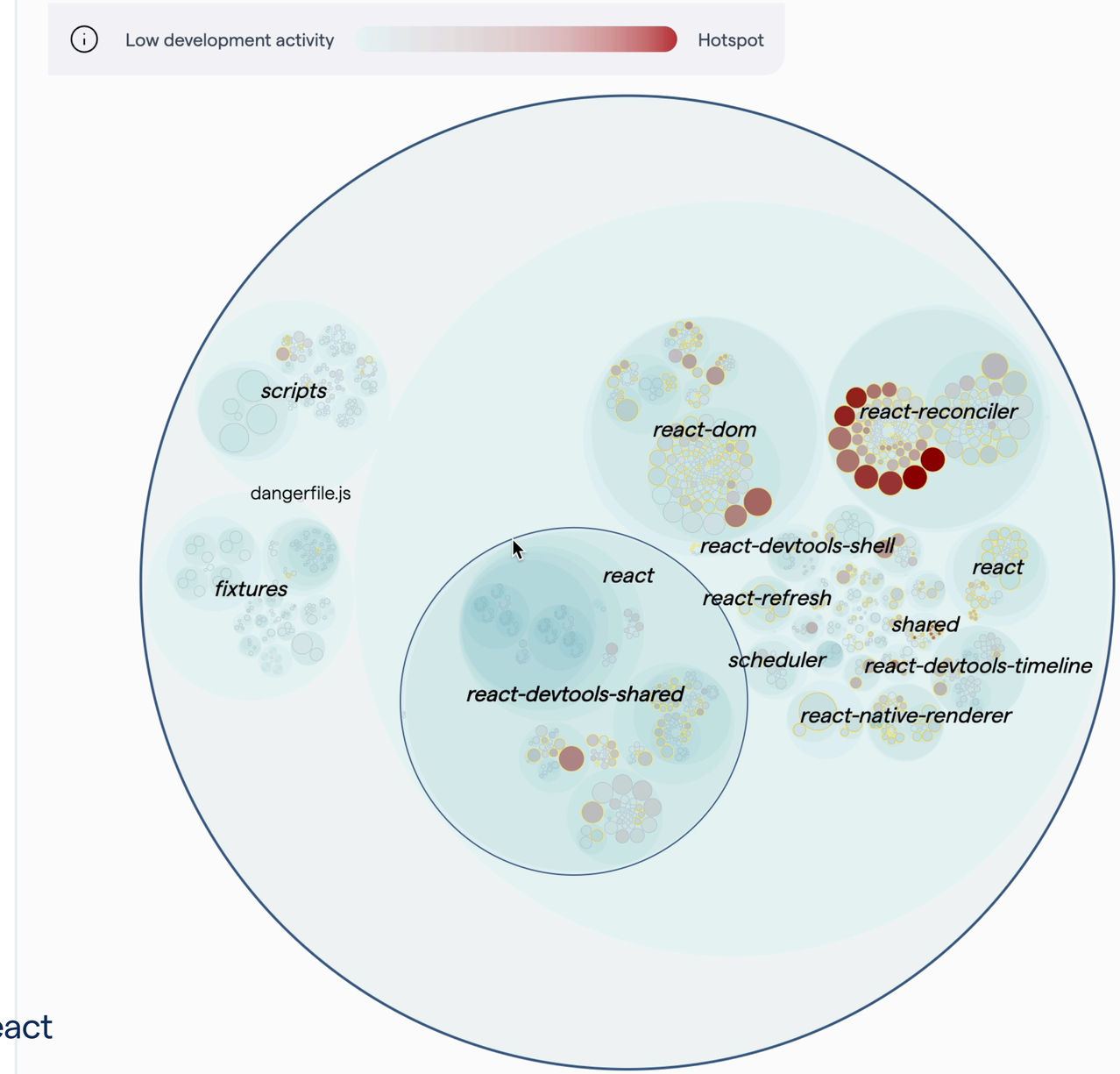
Behavioral Code Analysis = code + people + context

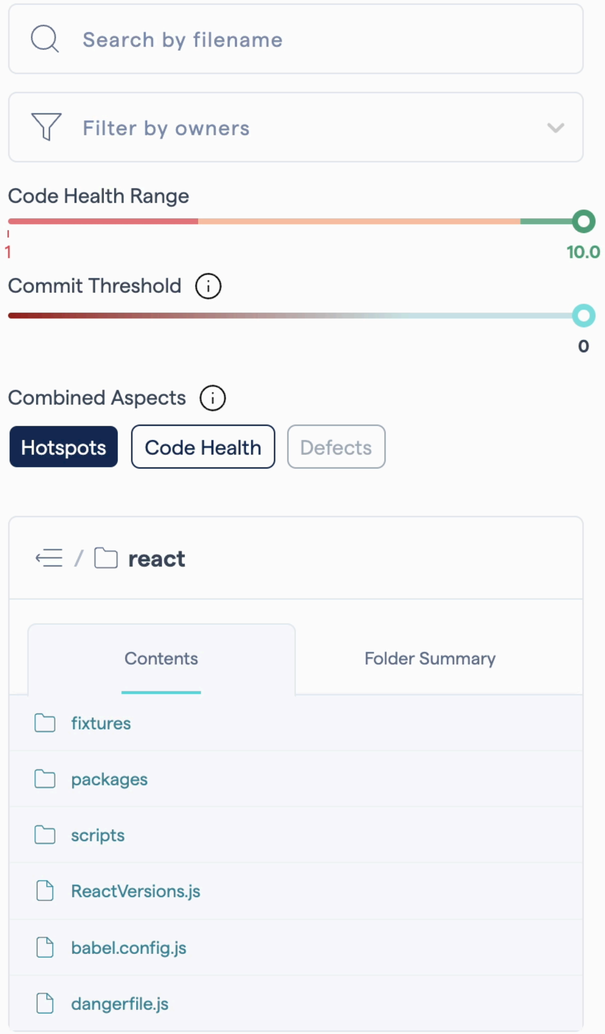
While the code is important, it's even more important to understand how we — as a development organisation — interact with the system we're building.

Version-Control — **A Behavioral Data Source**



Hotspots in Code: Visualizing the geographical offender profile





React: JavaScript UI library

400,000 lines of code

https://github.com/facebook/react

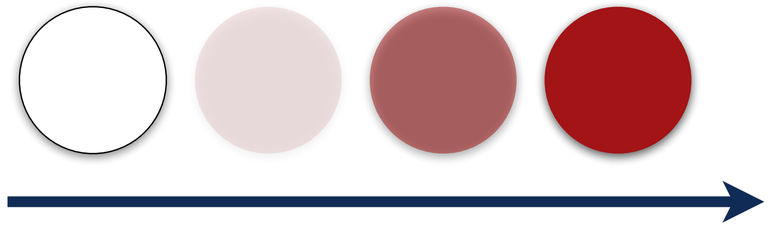
CodeScene

Case Study:

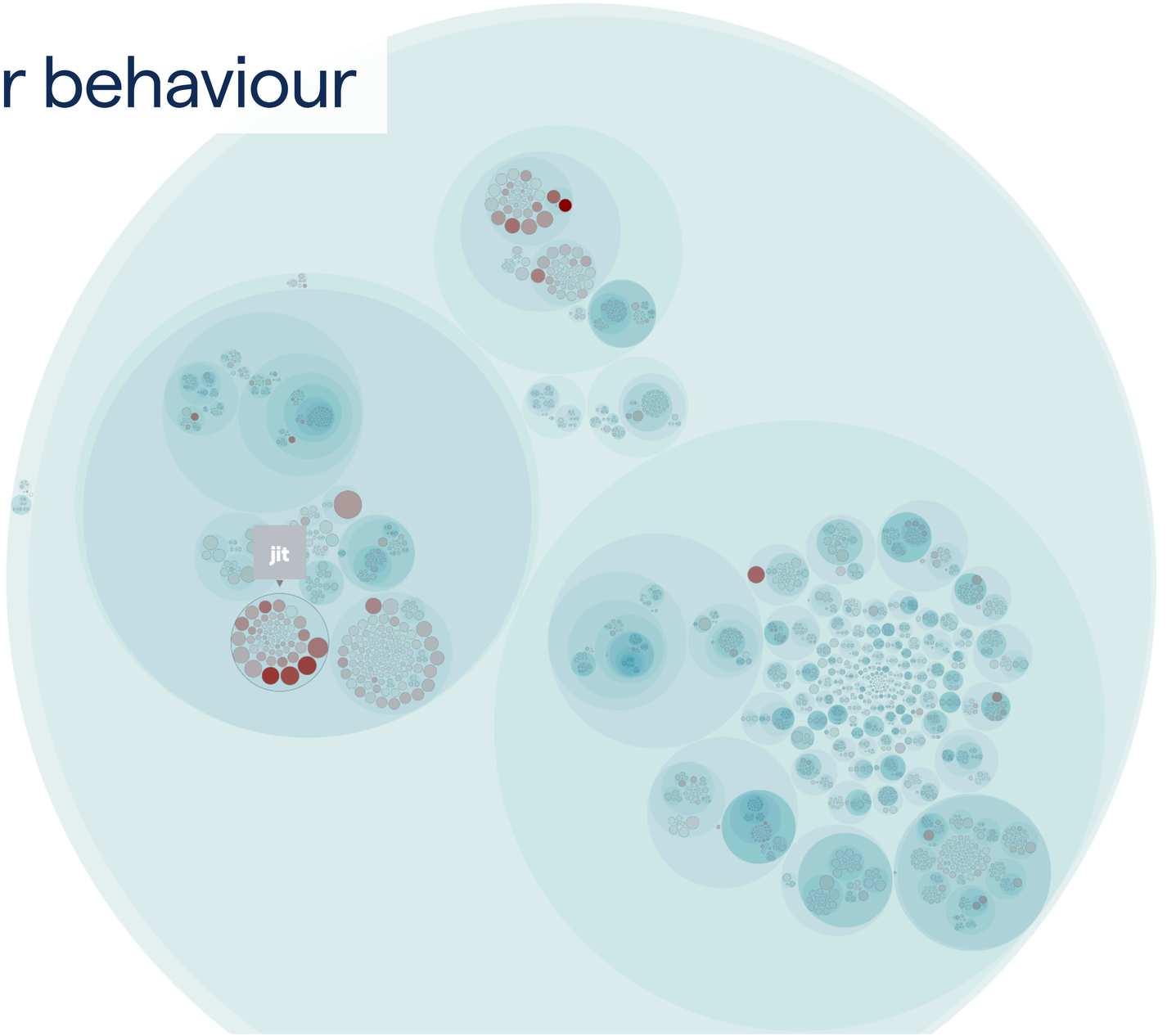
Use Hotspots to Prioritize Technical Debt

Hotspots:

Prioritize based on developer behaviour



Interest rate: Code Change Frequency



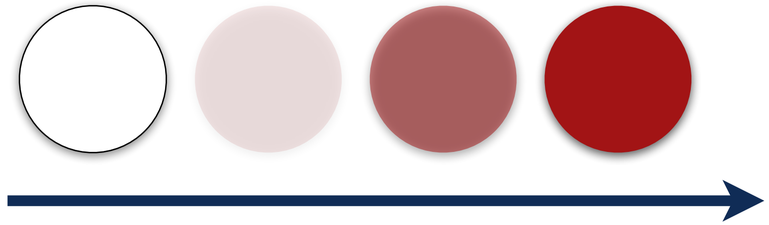
CoreCLR: the runtime for .Net

8.5 million lines of code

https://github.com/dotnet/coreclr

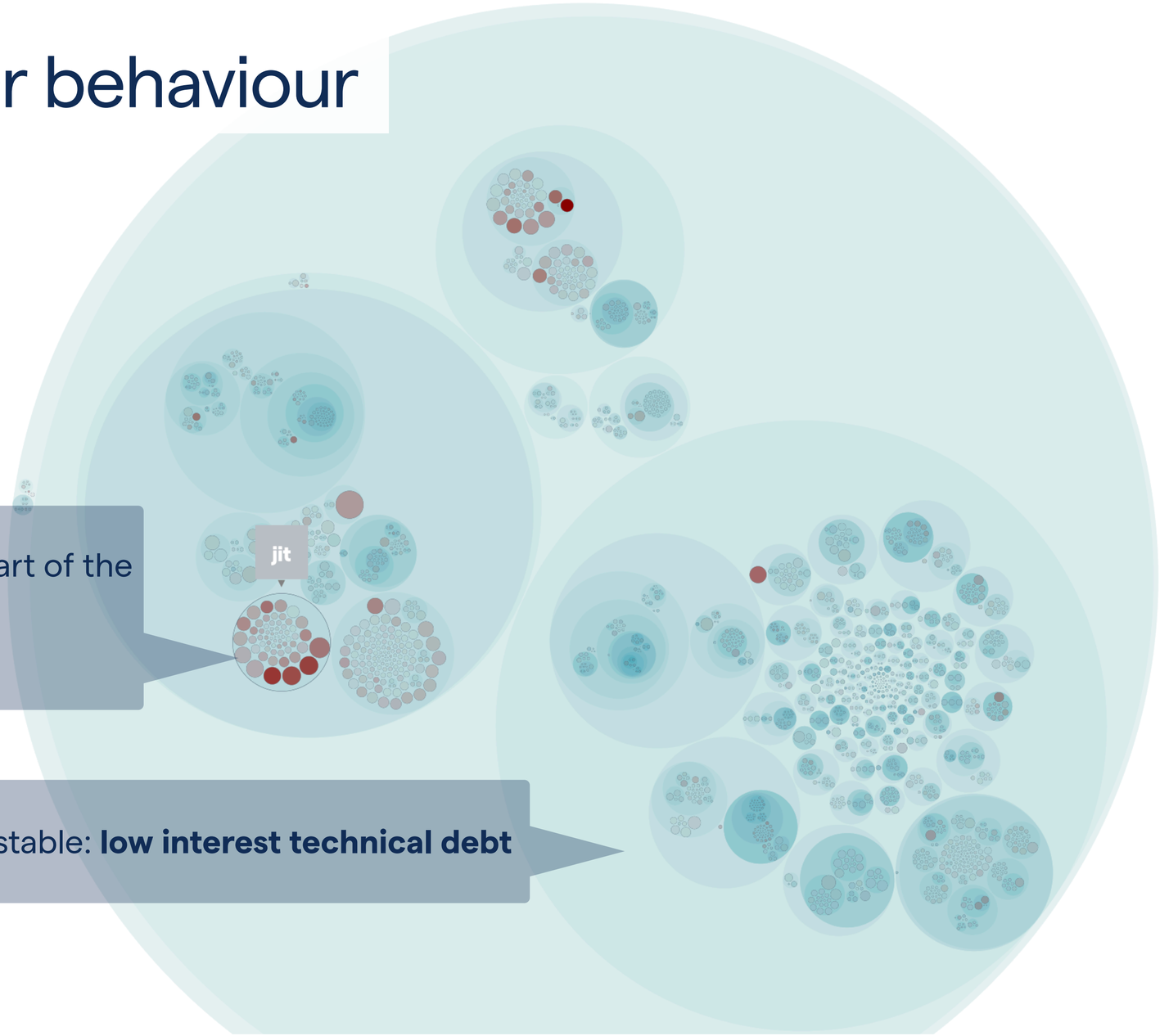
Hotspots:

Prioritize based on developer behaviour



Interest rate: Code Change Frequency

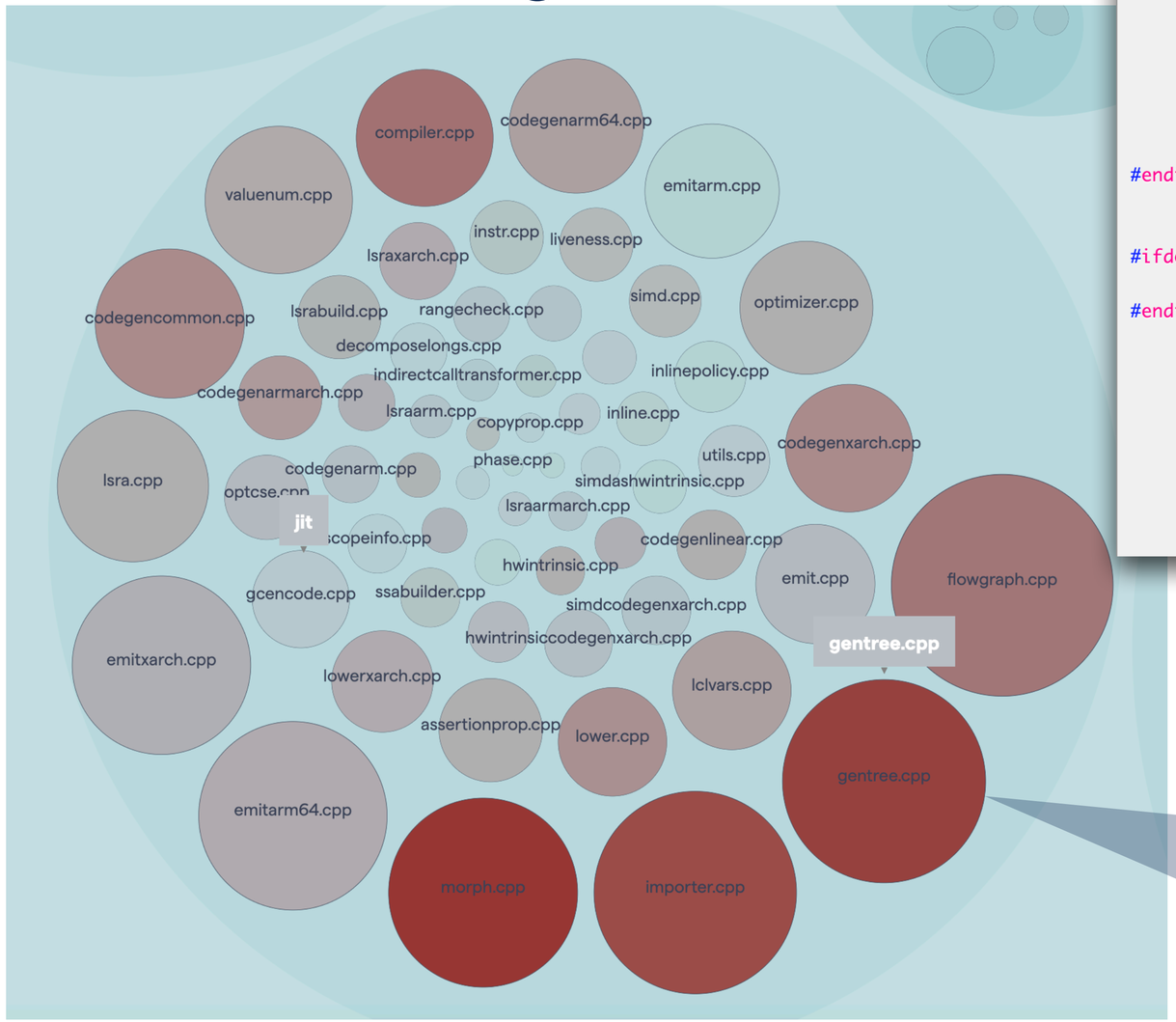
Most development activity is in a small part of the codebase: **high interest technical debt**



Most code is stable: **low interest technical debt**

CodeScene

A look into the *Jit* package:
Actionable Insights?

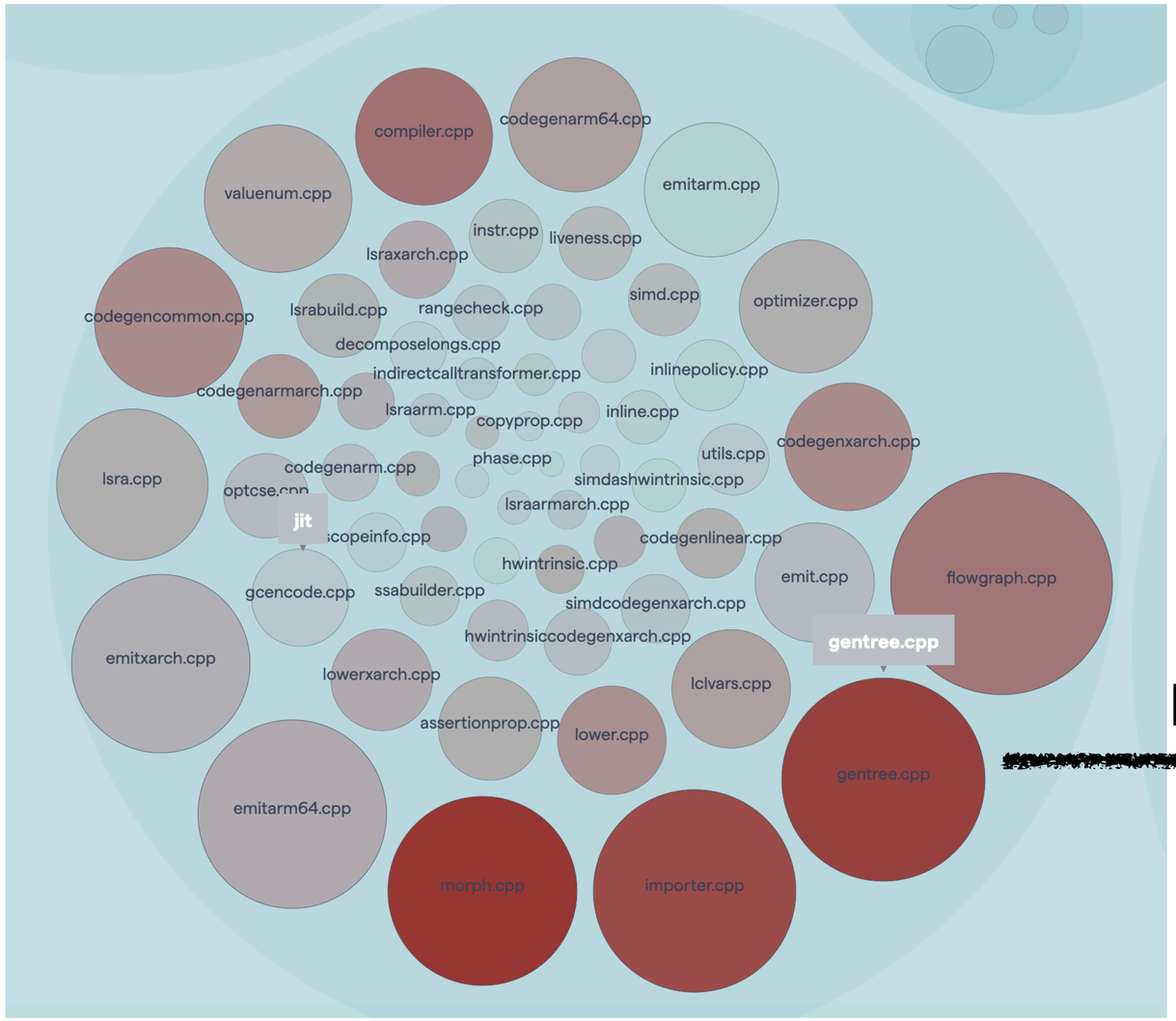



```
    if (kind & (GTK_CONST | GTK_LEAF))
    {
        switch (oper)
        {
            case GT_CNS_INT:

#if defined(LATE_DISASM)
                if (tree->IsIconHandle())
                {
                    copy =
                        gtNewIconHandleNode(tree->AsIntCon()->gtIconVal, tree->gtFlags, tree->AsIntCon(
                    copy->AsIntCon()->gtCompileTimeHandle = tree->AsIntCon()->gtCompileTimeHandle;
                    copy->gtType                          = tree->gtType;
                }
                else
#endif
                {
                    copy = gtNewIconNode(tree->AsIntCon()->gtIconVal, tree->gtType);
#ifdef DEBUG
                    copy->A
#endif
                    copy->A
                    copy->A
                }
                goto DONE;

            case GT_CNS_LNG
                copy = gtNe
                goto DONE;
```

```
case GT_LCL_VAR:

    if (tree->AsLclVarCommon()->GetLclNum() == varNum)
    {
        copy = gtNewIconNode(varVal, tree->gtType);
        if (tree->gtFlags & GTF_VAR_ARR_INDEX)
        {
            copy->LabelIndex(this);
        }
    }
    else
    {
        // Remember that the LclVar node has been cloned. The flag will
        // be set on 'copy' as well.
        tree->gtFlags |= GTF_VAR_CLONED;
        copy = gtNewLclvNode(tree->AsLclVar()->GetLclNum(),
                             tree->gtType DEBUGARG(tree->AsLclVar()->gtLclILoffs));
        copy->AsLclVarCommon()->SetSsaNum(tree->AsLclVarCommon()->GetSsaNum());
```

14,000 Lines of Code!

Hotspots: X-Ray: `gentree.cpp`



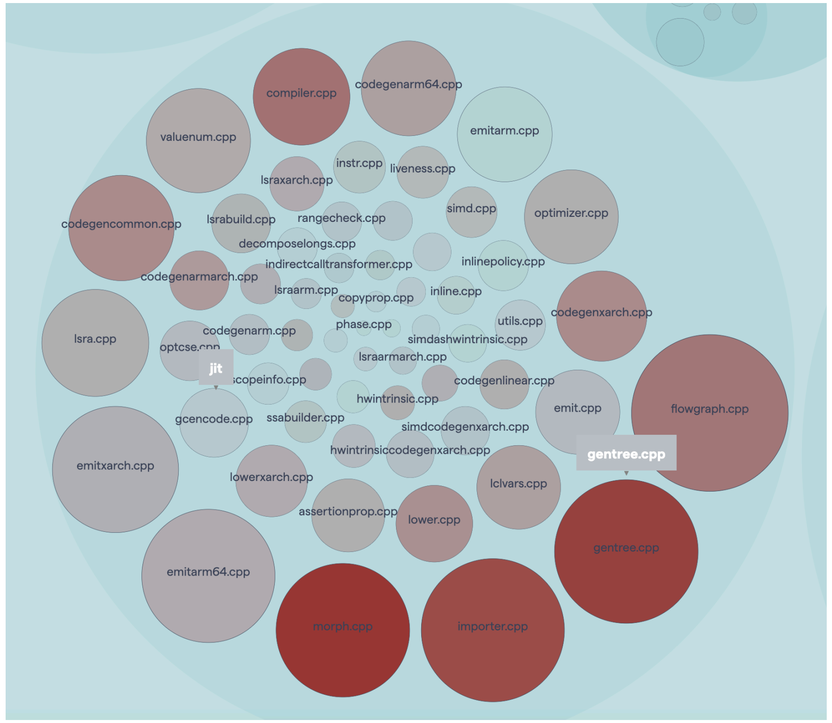
Parse

Function Level Hotspots

Recommended functions to improve.

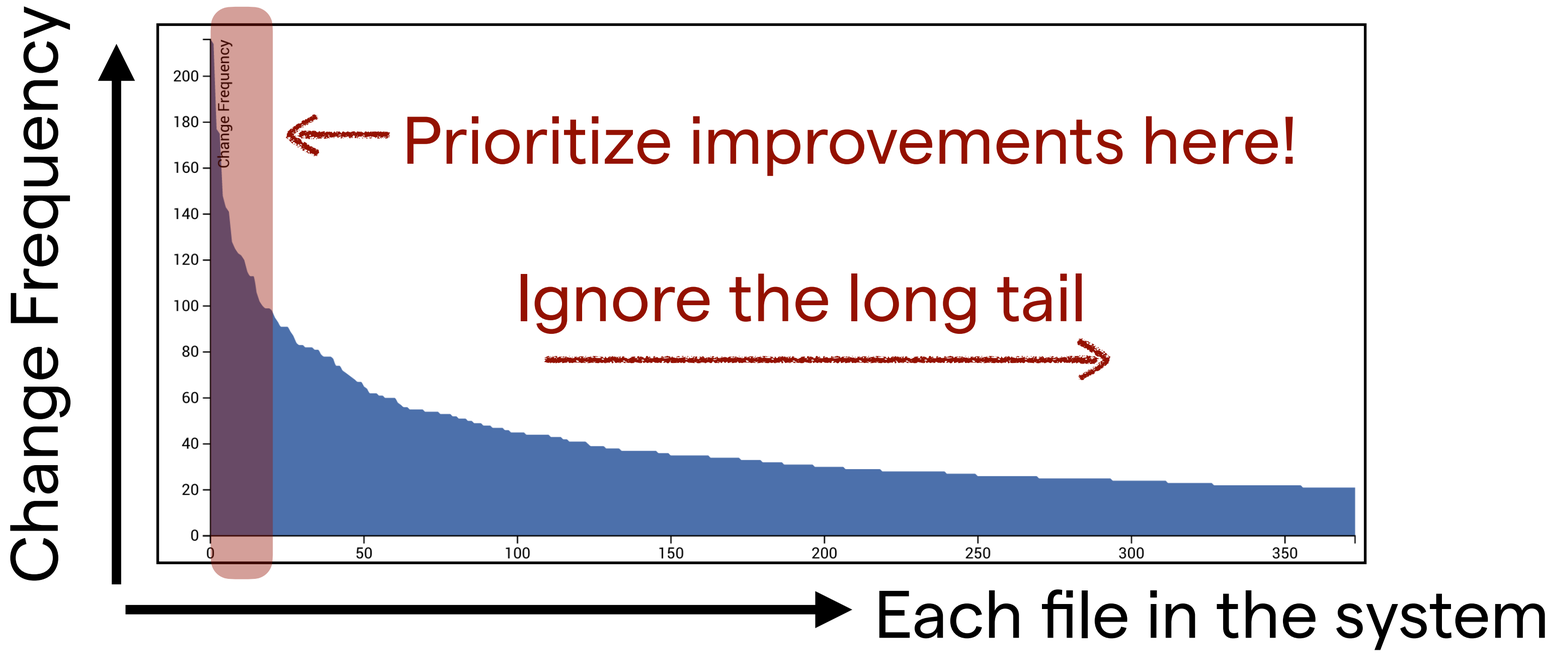
From https://pragprog.com/book/atevol/software-design-x-rays

X-Ray of `gentree.cpp`



Function	Change Frequency	Lines of Code	Cyclomatic Complexity
`Compiler::gtCloneExpr` View Complexity Trend | View Function Code	42	586	112
`Compiler::gtHashValue` View Complexity Trend | View Function Code	40	335	66
`GenTree::Compare` View Complexity Trend | View Function Code	34	390	110
`Compiler::gtSetEvalOrder` View Complexity Trend | View Function Code	29	1,531	291
`Compiler::gtDispTree` View Complexity Trend | View Function Code	23	578	130
`Compiler::gtDispNode` View Complexity Trend | View Function Code	18	510	125

Hotspots: why you don't have to fix all tech debt



Key take-aways:

- Most code is in the long-tail. This is low-interest debt.
- Hotspots only make up 2-4% of the total codebase, but attract 20-70% of all development activity!

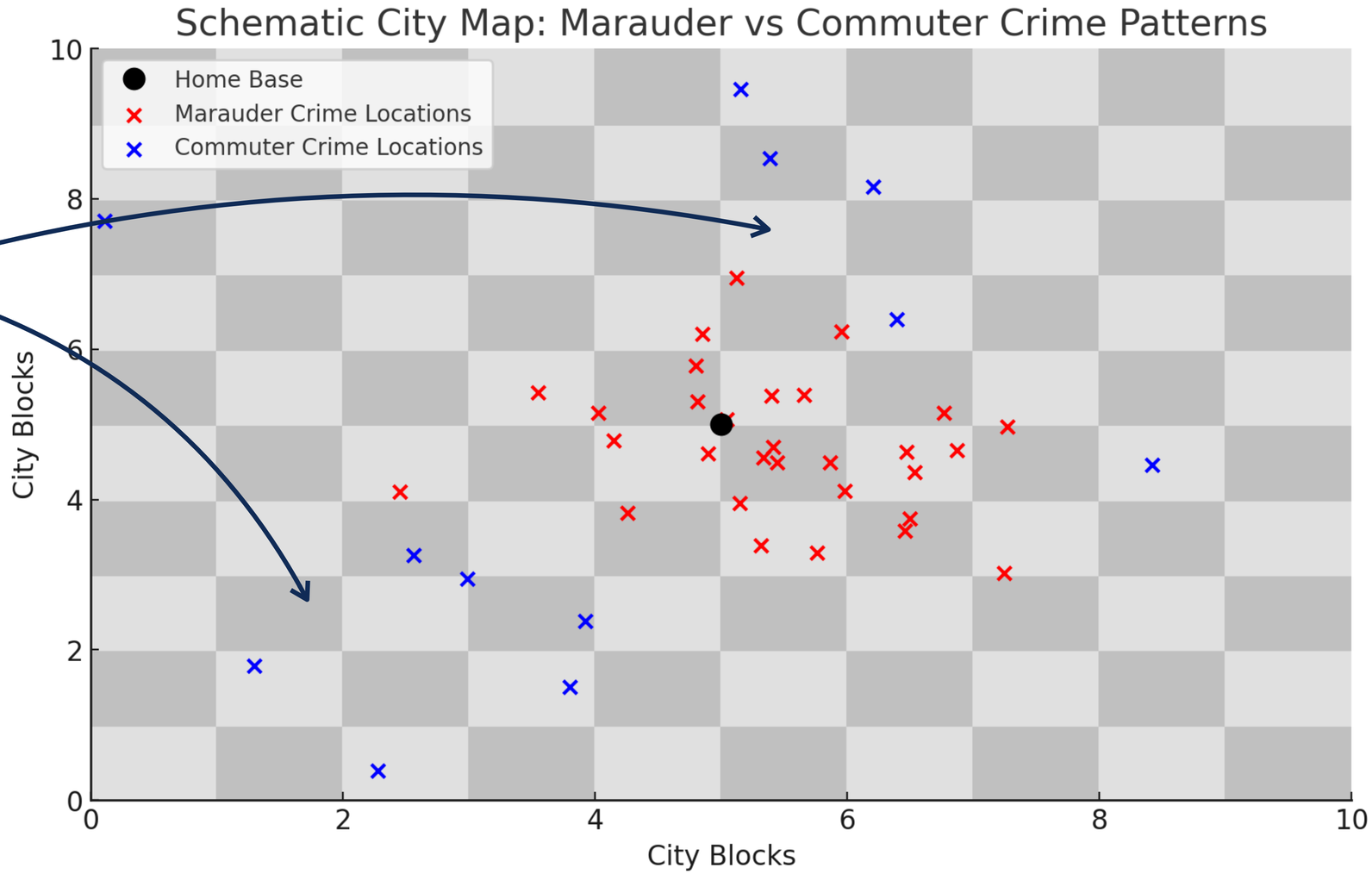
🎗 **Code health issues in a hotspot are expensive. This is high-interest debt.**

Beyond hotspots:

Multiple types of Offender Behaviour

Understanding Spatial Behaviour:
Marauders and Commuters

How can we tell if these hotspots belong to the same offender?

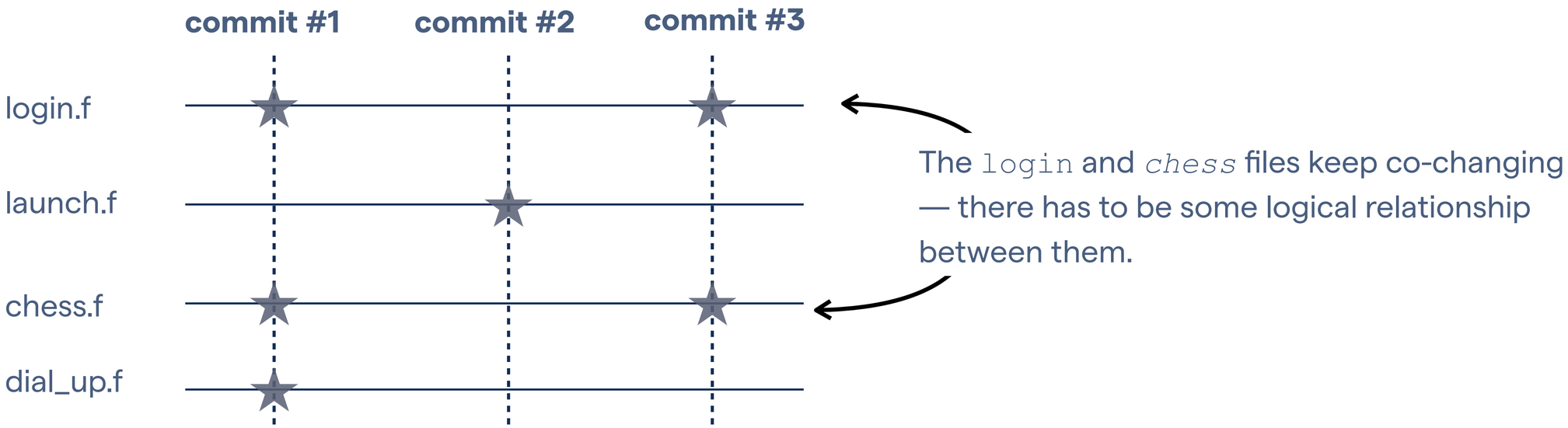


Not all dependencies are equal

Code changes for a reason

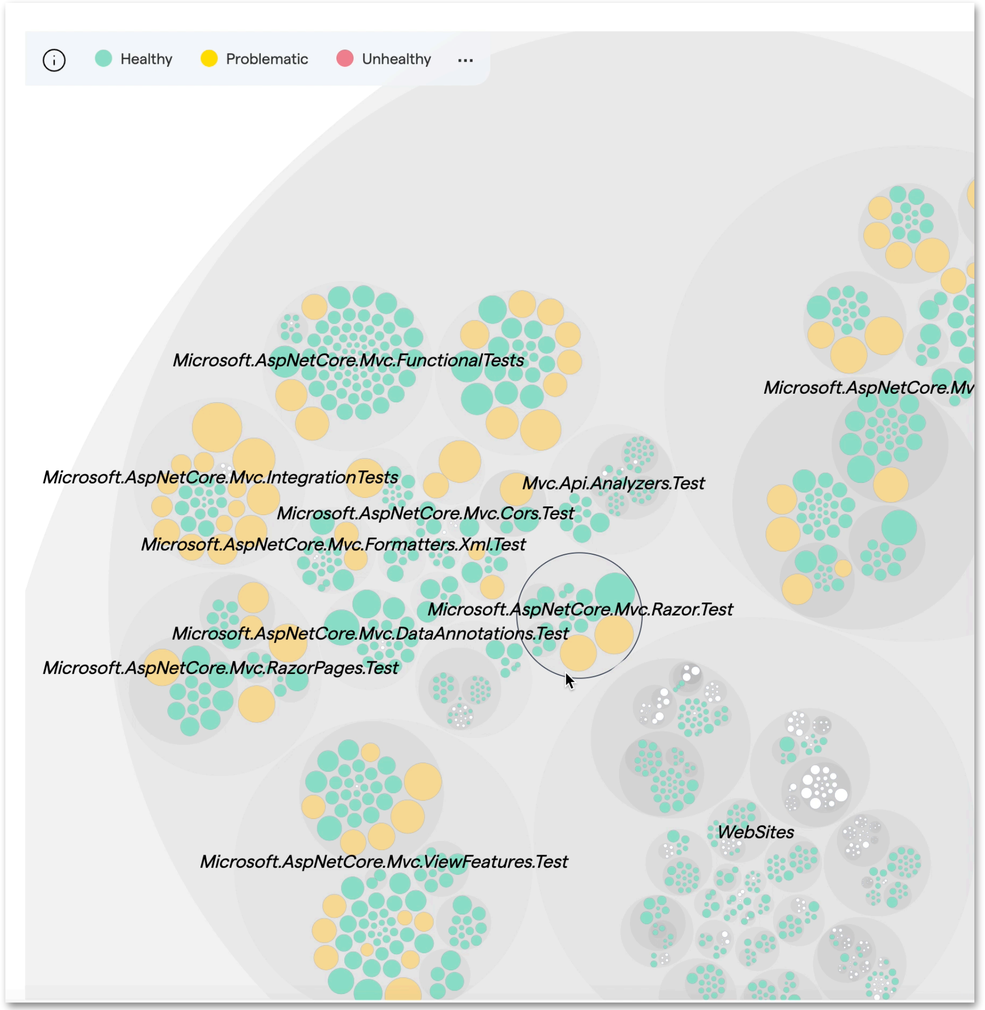
What if we could evaluate our actual modifications against the desired patterns?

Introducing Change Coupling: logical dependencies



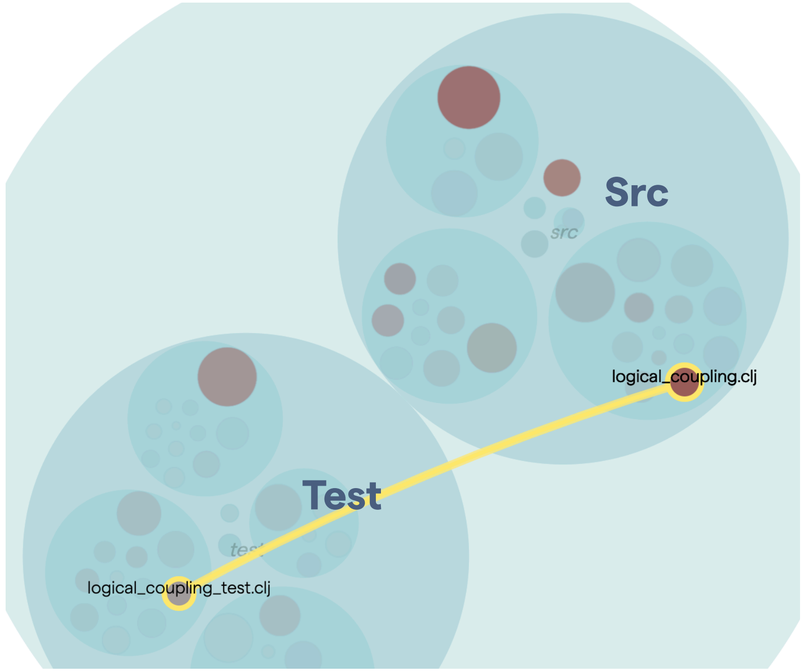
Example inspired by https://en.wikipedia.org/wiki/WarGames (the movie)

Visualize the Cost of Change



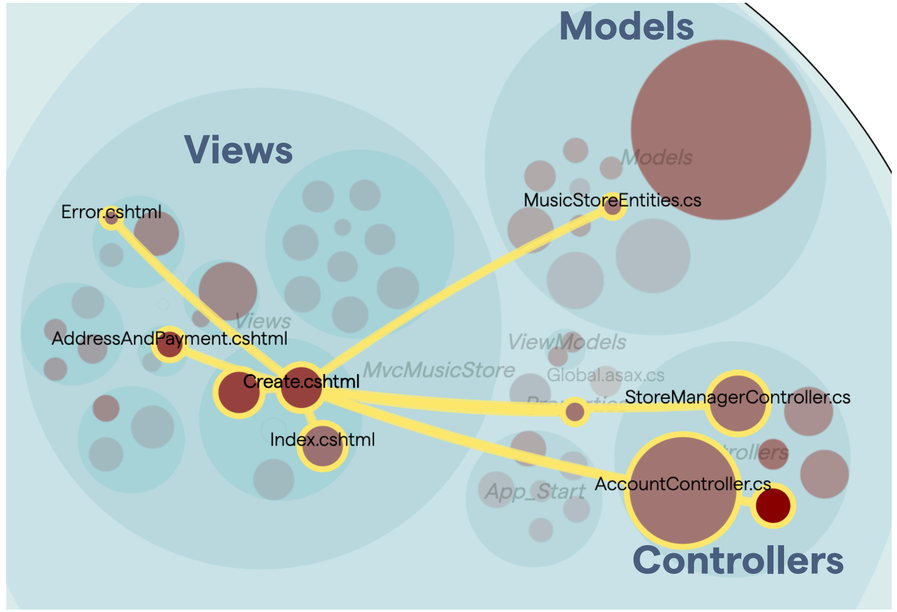
https://github.com/dotnet/aspnetcore

Acting on Change Coupling: **Contextualize**



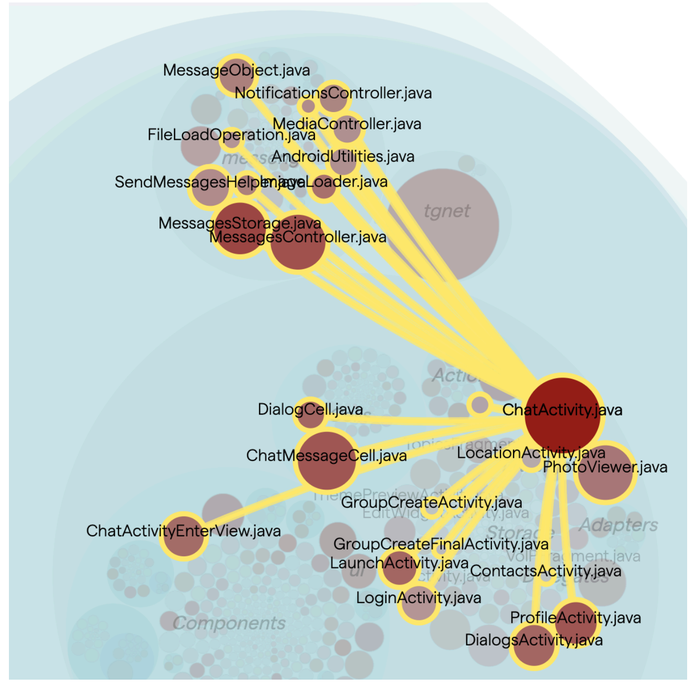
Desired change patterns: a unit test co-evolves with the code under test.

https://github.com/adamtornhill/code-maat



Expected change patterns ✓, **but** does this architectural pattern really support the way the system evolves?

https://github.com/SebastiaanLubbers/MvcMusicStore

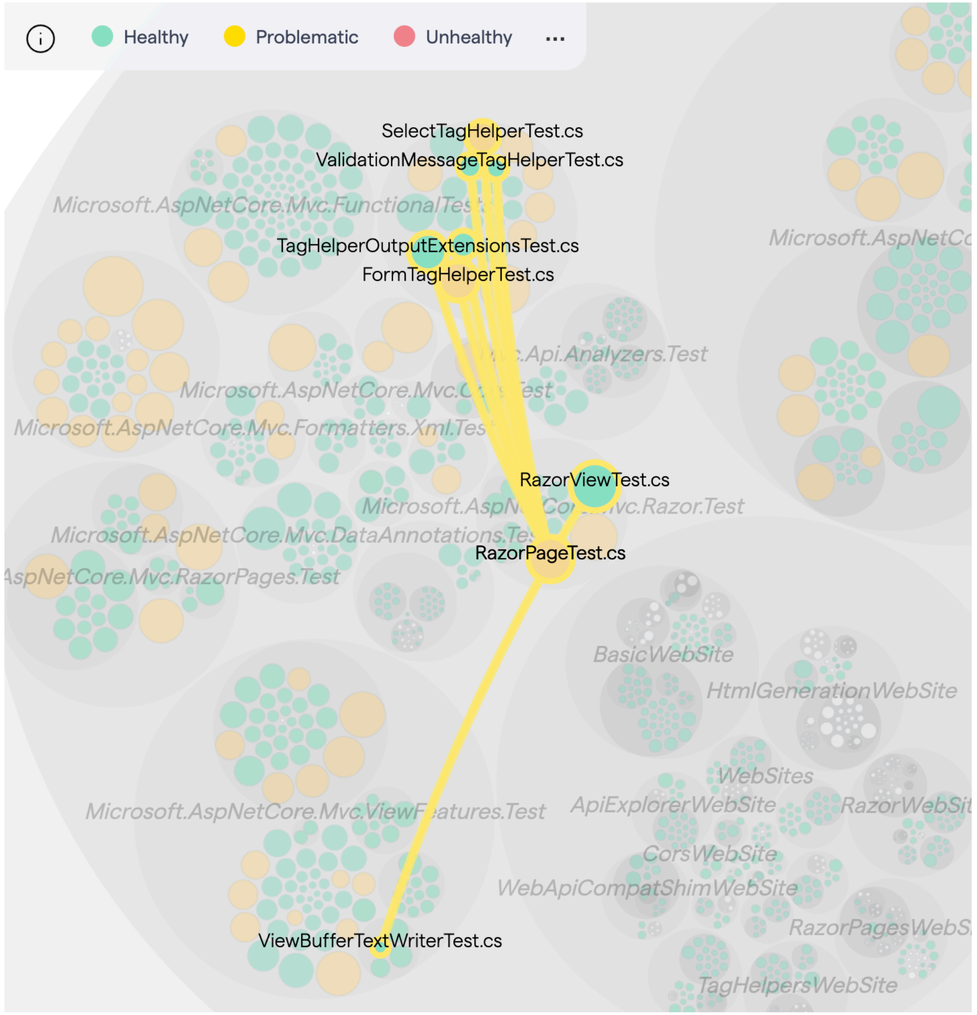


God Class: tight coupling without obvious patterns, nor benefits.

https://github.com/DrKLO/Telegram

A spiralling cost of change ➡

Tip: **Evaluate Change Coupling according to your Architectural Patterns**



Change coupling is problematic when it violates your architectural principles.

Surprise is one of the most expensive things you can put into a software architecture.

CodeScene

Case Study:

Unhealthy Code with a Low Truck Factor

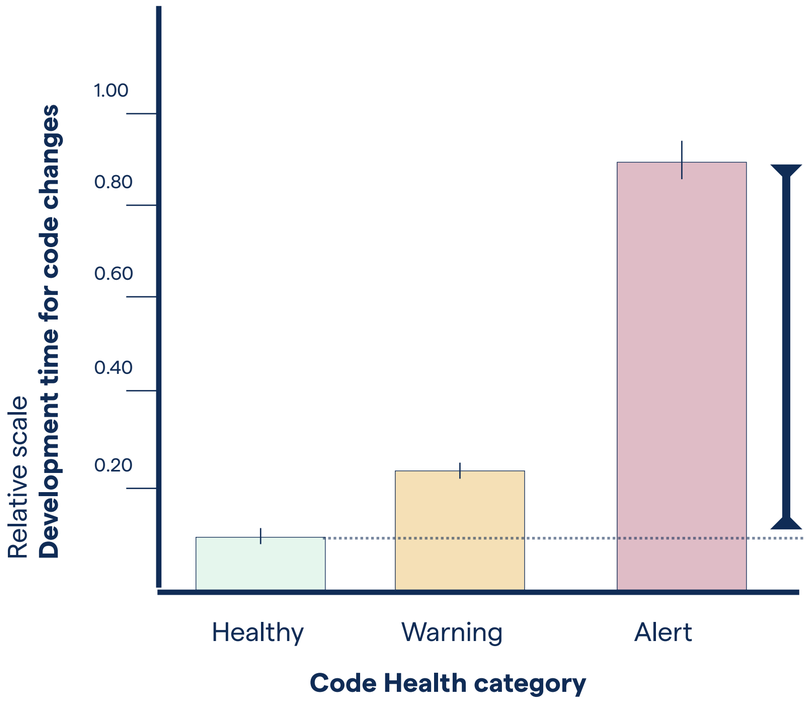
The Technical Debt That Wasn't

Product #1	Product #2	Product #3
		?

Don’t Confuse a lack of Familiarity for Complexity

Unhealthy Code: Unfamiliarity breeds risk

Task completions times in unhealthy code are up to 10x longer compared to green, healthy code



Code Red: The Business Impact of Code Quality:
https://arxiv.org/abs/2203.04374

Uncovering the people side:

Does it matter which programmer that makes a change or fixes a bug?

Unhealthy Code comes with a significant on-boarding cost:

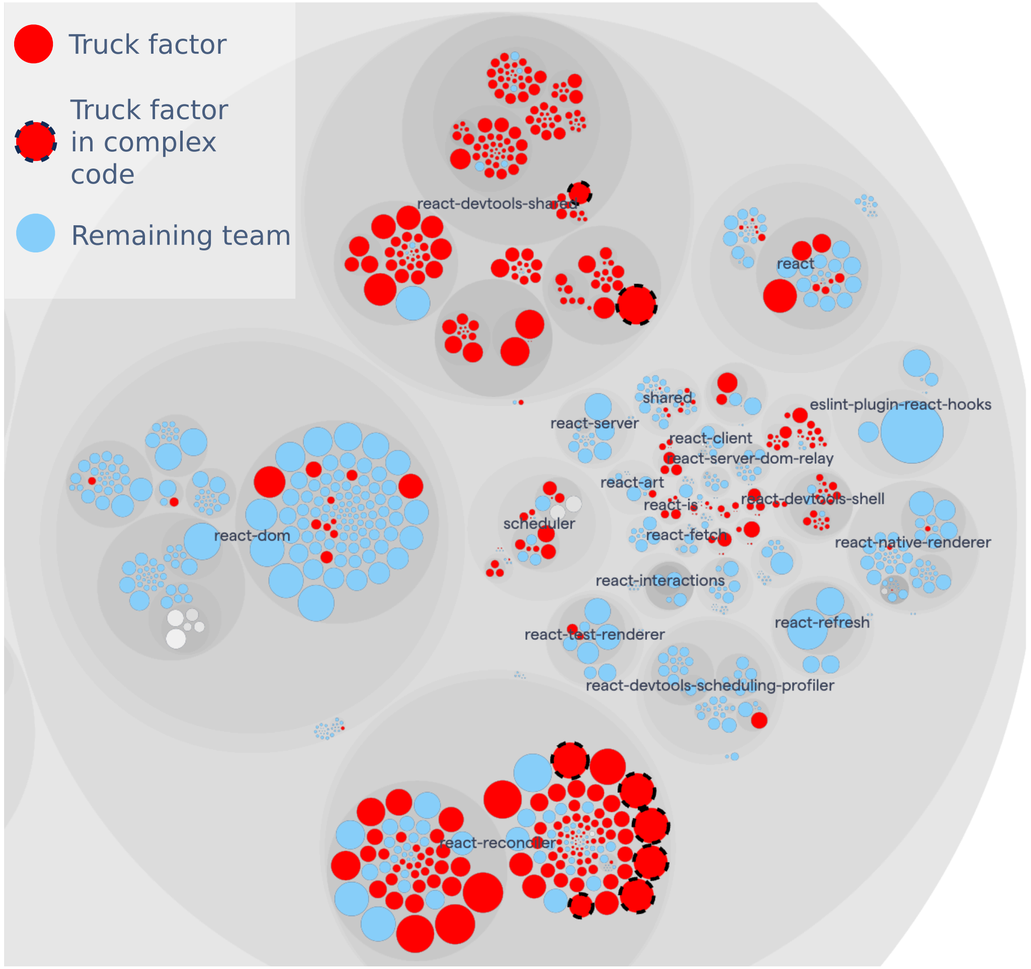
unless you're the main developer, you need

- 45% more time for small tasks, and
- 93% more time for large tasks compared to Green Code.

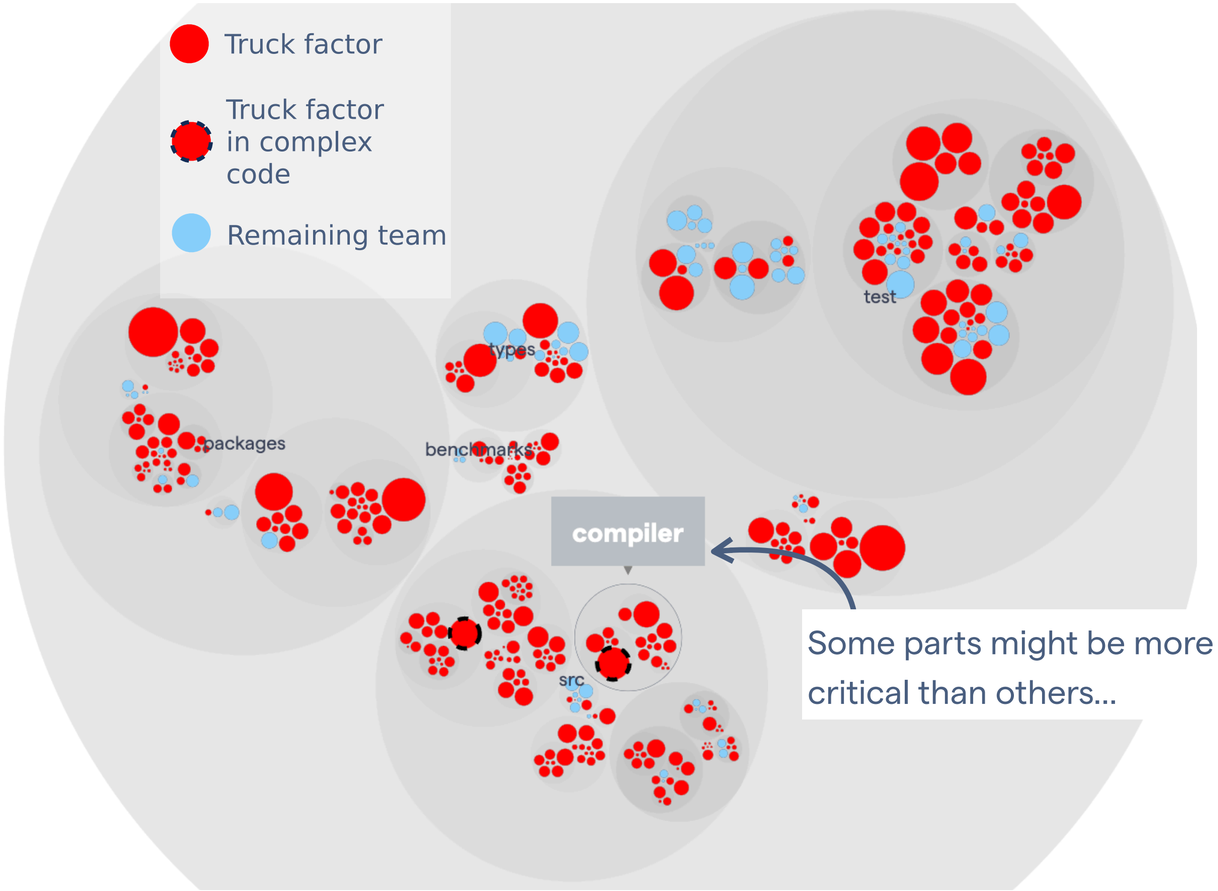
Borg, M., Tornhill, A., & Mones, E. (2023). U Owns the Code That Changes and How Marginal Owners Resolve Issues Slower in Low-Quality Source Code: https://arxiv.org/pdf/2304.11636.pdf

Meet the ~~Bus~~ ~~Lottery~~ Truck Factor

When Unhealthy Code meets the Truck Factor:
The Highway to Legacy Code

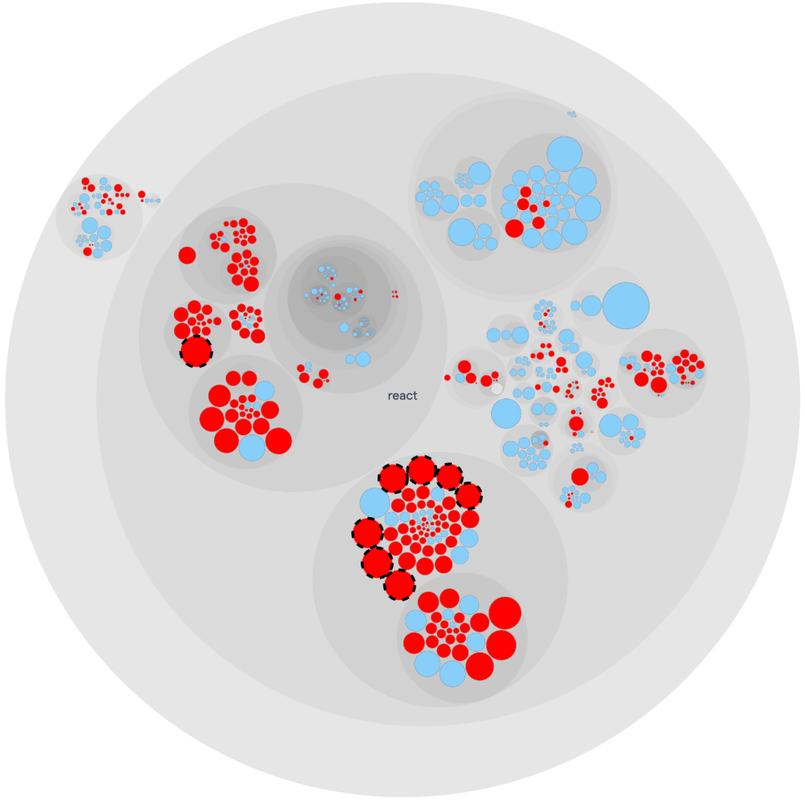


The Truck Factor in React: 2 people



Visualizing the truck factor in vue.js: 1 person

Code Quality issues amplify Organizational Problems

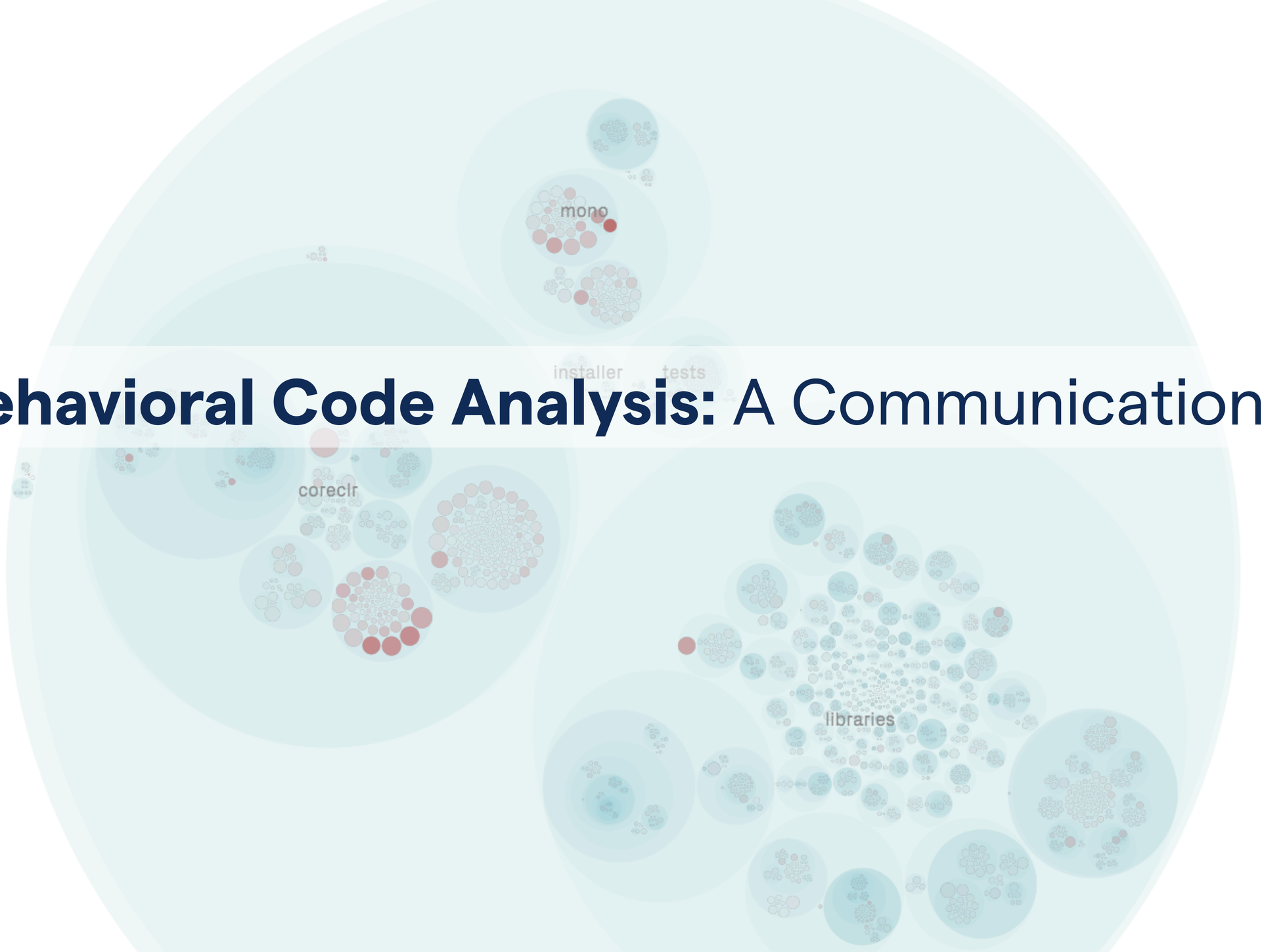


Let community smells be a driver for refactoring: prioritize improvements to unhealthy code with a low truck factor.

Summary

Improving code reduces organizational friction

- **simpler on-boarding** by flattening the [technical] learning curve,
- **reduce key person risks** by making code cognitively affordable,
- **shorter development cycles** in healthy code,
- **minimize defects** by avoiding excess coordination in code, and
- **improve team morale** by increasing developer happiness.



Behavioral Code Analysis: A Communication Tool

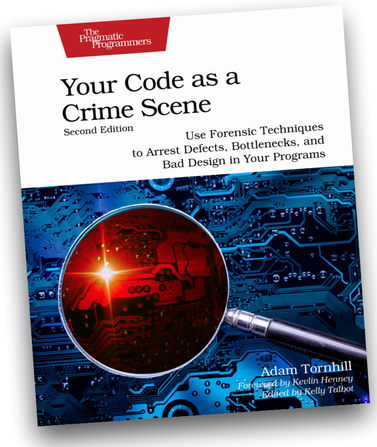
Learn more: references, books, and tools

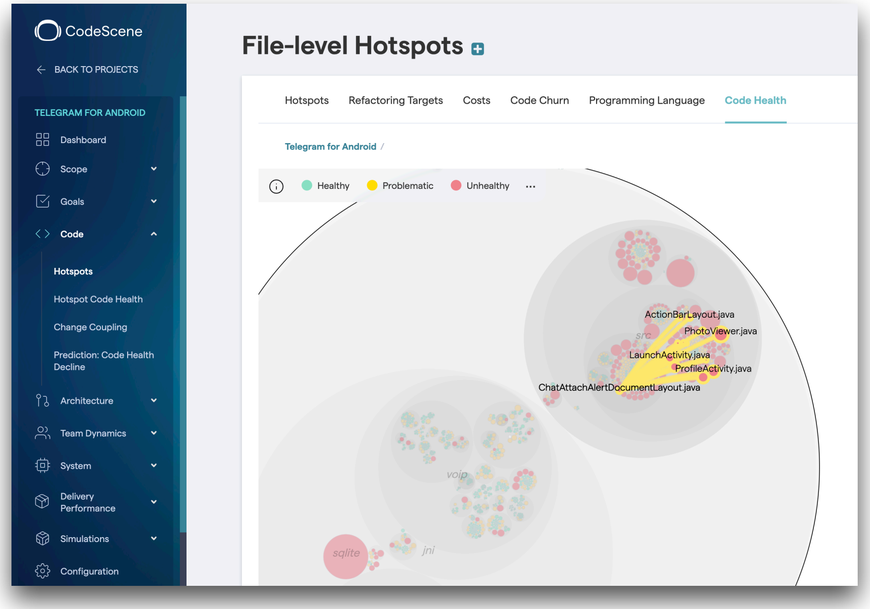
Research papers:

- **The business impact of Code Quality**:
  https://arxiv.org/abs/2203.04374
- **On-boarding costs in unhealthy code**:
  https://arxiv.org/pdf/2304.11636.pdf

Code quality metrics:

- https://codescene.com/code-health





Where are your Hotspots?

Use the CodeScene tool for analysis + visualizations:

https://codescene.com/

Your Code as a Crime Scene, 2nd ed (2024):

https://pragprog.com/titles/atcrime2/your-code-as-a-crime-scene-second-edition/

Get in touch:

https://se.linkedin.com/in/adam-tornhill-71759b48

https://twitter.com/AdamTornhill