CodeScene

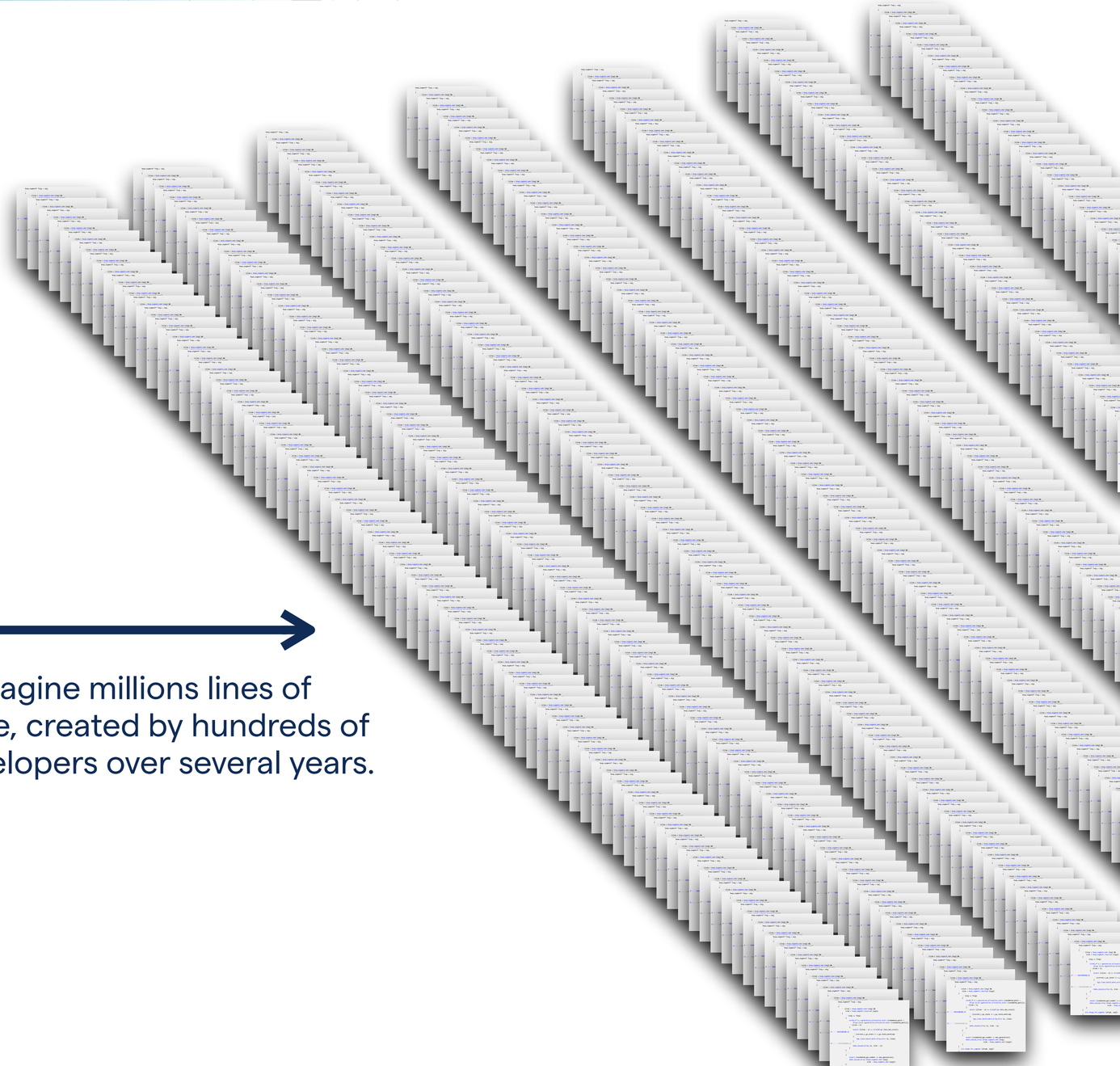
Socio-Technical Smells:
How Technical Problems Cause Organizational Friction

@AdamTornhill

Why Software is Hard

Is source code hard to understand?

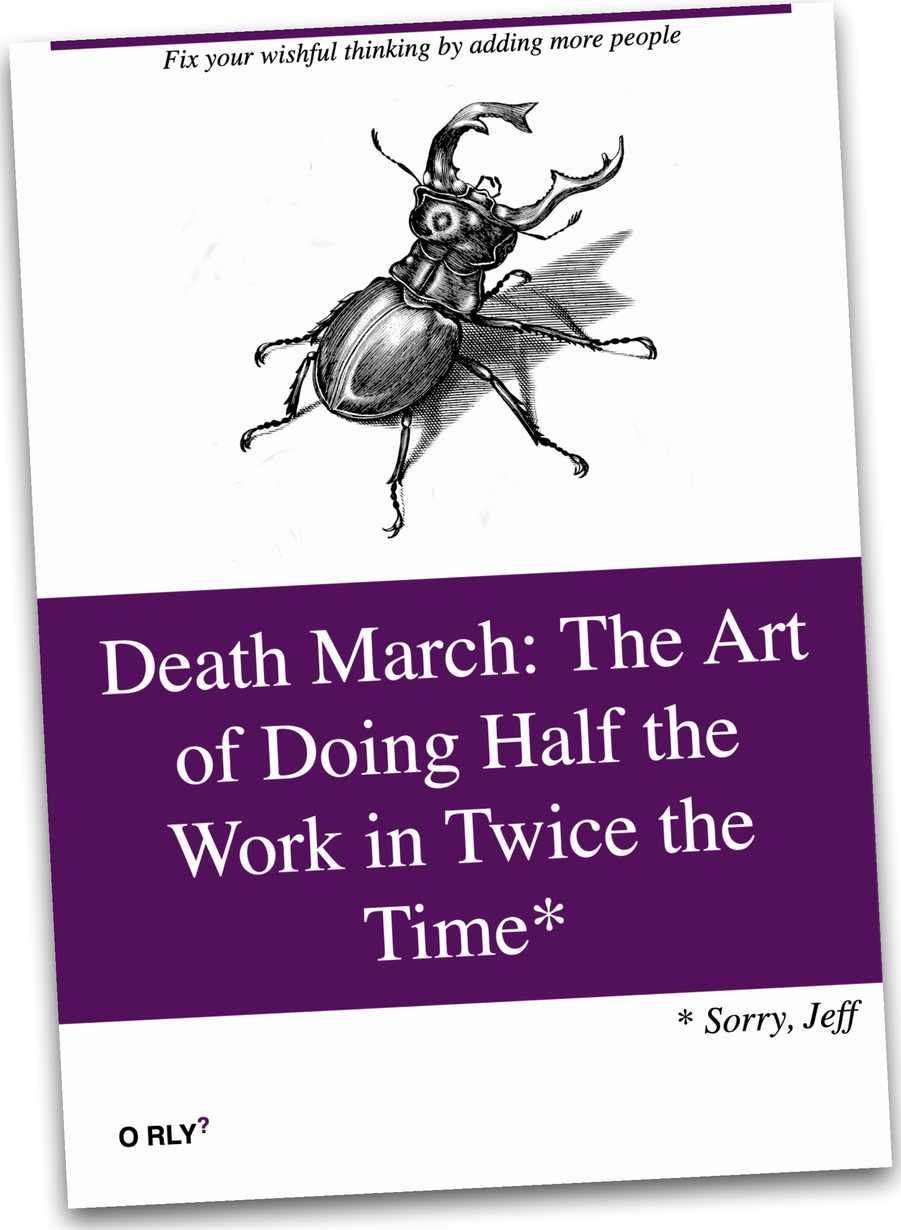
```
        heap_segment* fseg = seg;
        do
        {
            if (slow > heap_segment_mem (seg) &&
                slow < heap_segment_reserved (seg))
            {
                if (seg == fseg)
                {
                    uint8_t* o = generation_allocation_start (condemned_gen1) +
                        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.

Socio-Technical Smell #1:

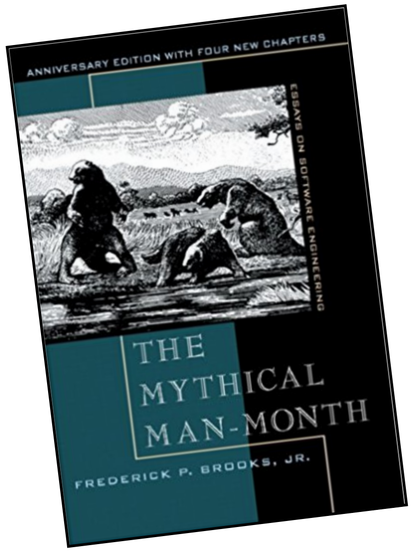
The Overcrowded System



@AdamTornhill

Generated via https://arthurbeaulieu.github.io/ORlyGenerator/

The Mythical Man-Month



<u>Brooks's Law:</u>

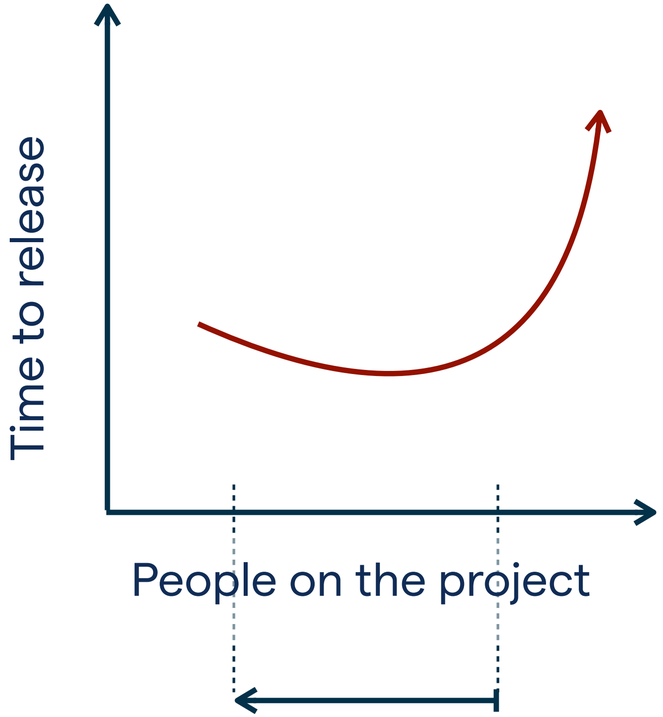
Adding people to a late software project makes it later.

@AdamTornhill

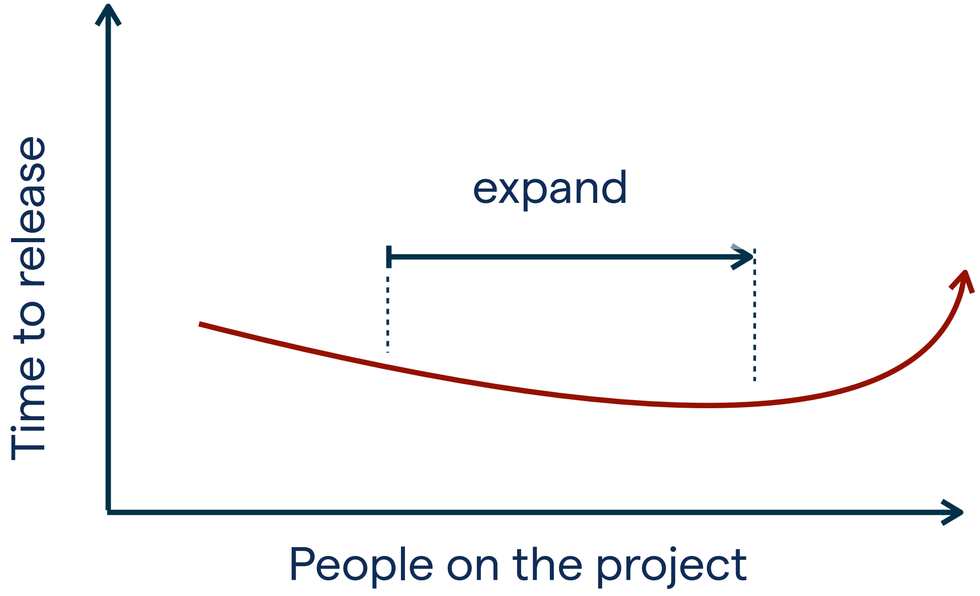
Rescuing a Death March

Scope with Brooks's Law in Mind

Option #1: shrink the organization to fit the modularity of the work to be done.



Option #2: re-design/architect to accommodate more work in parallel



A Coordination Bottleneck?

```
export function init(options : ViewOptions) {
  const fileSummaryContainer = $('#filecontent');

  const evolutionContainer = $('#evolution');
  xhr.csv(options.evolutionUrl, parseEvolutionaryMetricRow)
    .then(data => {
      const table = $('<table>')
        .addClass('table table-striped');

      $.each(data, (i, row) => {
        const tr = $('<tr>')
          .append($('<th>').text(row.statistic))
          .append($('<td>').text(row.value.toLocaleString()));
        table.append(tr);
      });

      tableSorter.sort(table, {});

      evolutionContainer.append(table);
    });

  xhr.csv(options.fileSummaryUrl, parseFileSummaryRow)
    .then(data => {
      var fileTable = tabulator.tabulate(fileSummaryContainer.get(0), data, ['lan

      fileTable
        .attr('id', 'summarytable')
        .classed('file-content', true)
        .selectAll('thead th')
        .text(function(column) {
          return column.charAt(0).toUpperCase() + column.substr(1);
        });

      tableSorter.sort($(fileTable.node()), {});
    });
```

@AdamTornhill

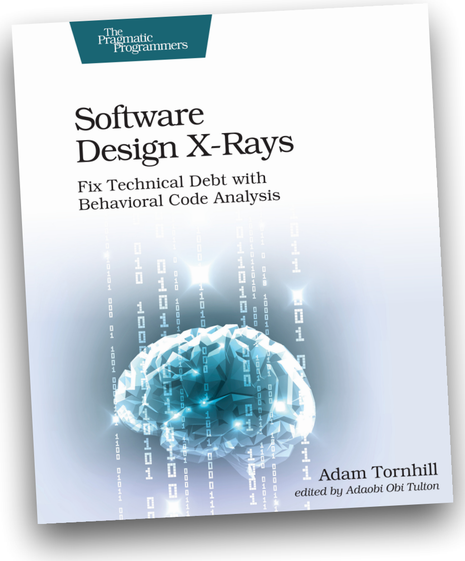
```
export function init(options : ViewOptions) {
  const fileSummaryContainer = $('#filecontent');

  const evolutionContainer = $('#evolution');
  xhr.csv(options.evolutionUrl, parseEvolutionaryMetricRow)
    .then(data => {
      const table = $('<table>')
        .addClass('table table-striped');

      $.each(data, (i, row) => {
        const tr = $('<tr>')
          .append($('<th>').text(row.statistic))
          .append($('<td>').text(row.value.toLocaleString()));
        table.append(tr);
      });

      fileTable
        .attr('id', 'summarytable')
        .classed('file-content', true)
        .selectAll('thead th')
        .text(function(column) {
          return column.charAt(0).toUpperCase() + column.substr(1);
        });

      tableSorter.sort($(fileTable.node()), {});
    });
```

The Tragedy of Software Design:

The Organisation that builds the System is Invisible in the Code itself

@AdamTornhill

What is a **Behavioral Code Analysis?**

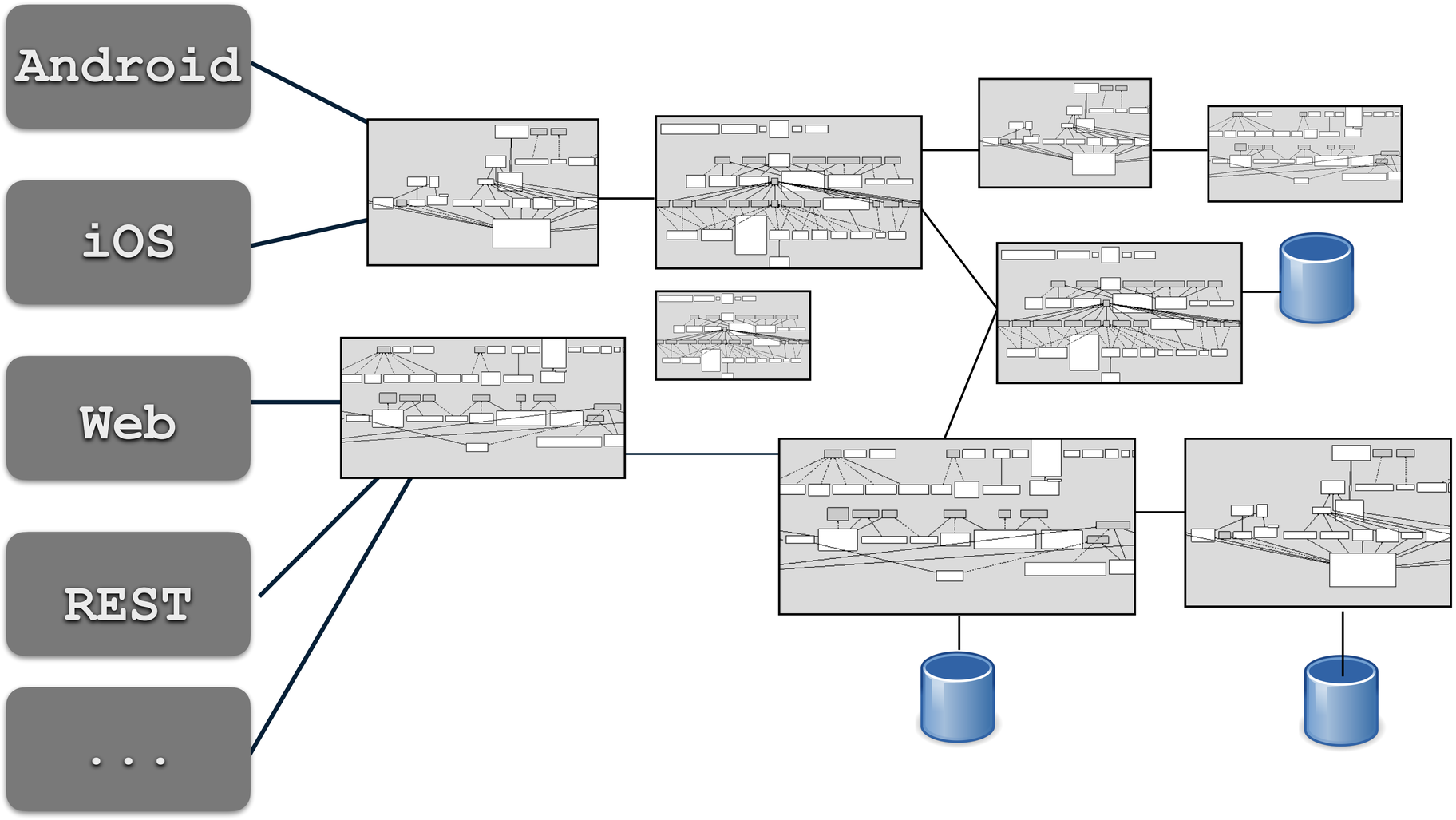


“While the code is important, it’s even more important to understand how we — as a development organisation — interact with the system we’re building.”

— Software Design X-Rays, 2018

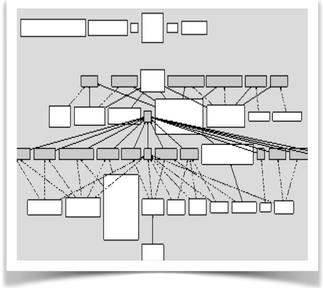
Socio-Technical Smell #2:

Coordination Bottlenecks in the Code

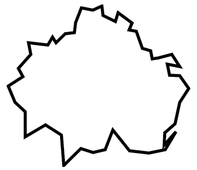

Android
iOS
Web
REST
...

Social Factors Influence how we Perceive a Codebase

Reality



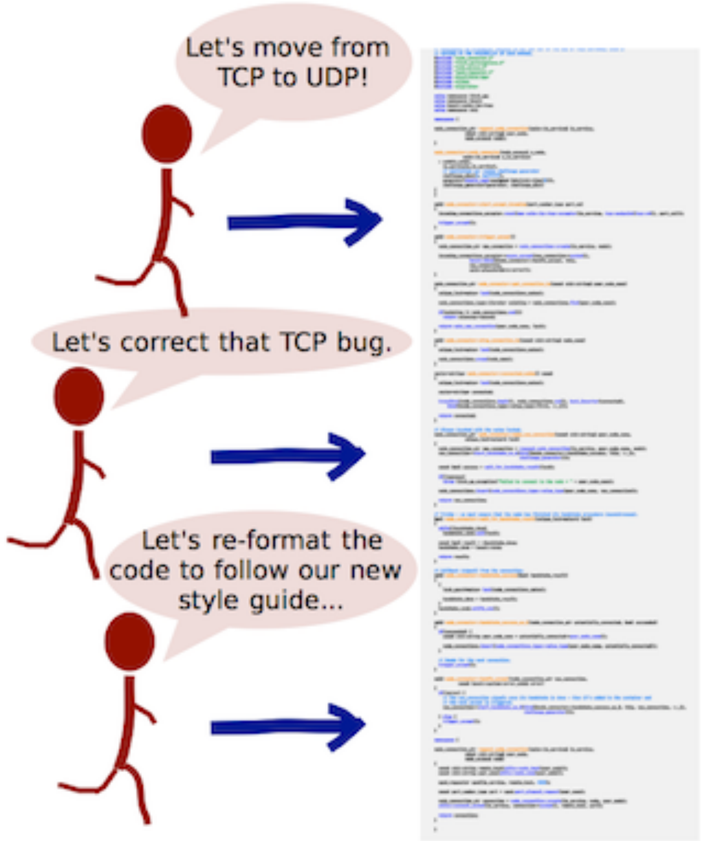
Mental Model



The more developers working on the same code, the harder to maintain stable mental models

Watch out:
a poor socio-technical fit invalidates our mental models

Software is rarely built in isolation



Discover Coordination in Code:
Collecting the Evidence

David has modified the FuelInjector file...

..and so has Stephen.

```
commit 796d31809b3683083d3b62ccbab4f00dec8ffb1f
Author: David Lightman
Date:   Fri Aug 12 13:27:53 2022 -0700

    Support multiple grades of fuel (small fix, right).

402     2       src/detectors/FuelGradeDetector.cs
1       0       src/engine/FuelInjector.cs

commit 32baab38f8f48f629ccd3f7564251b91dc2d777
Author: Stephen Falken
Date:   Thu Aug 11 23:04:45 2022 -0400

    Increase the pressure to use the fuel more efficiently.

12      5       src/engine/FuelInjector.cs
1270    0       test/engine/FuelInjectionTest.cs
```

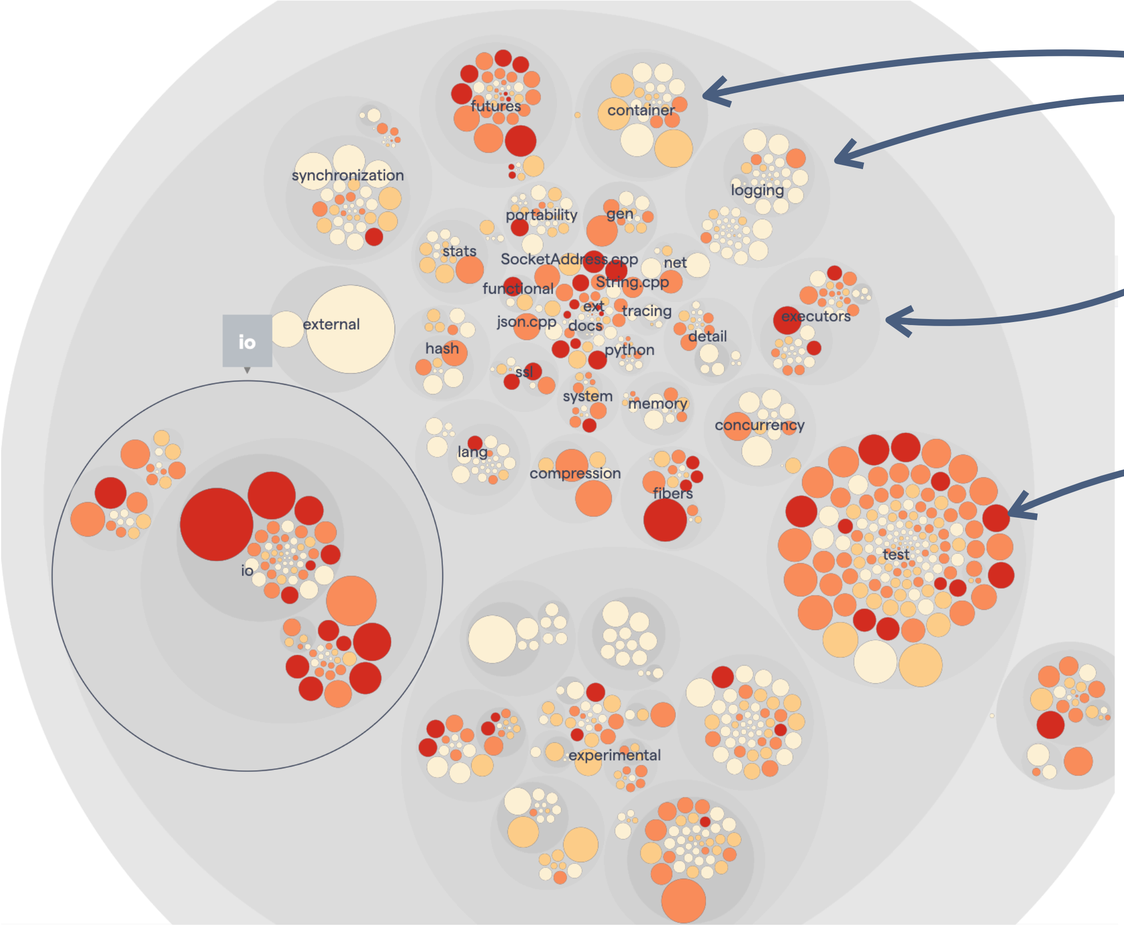
A file modified by multiple authors.

Coordination Analysis:
How many authors have touched each file?

```
file,                        n-authors, n-commits
build/builder.py,            13,        38
io/AsyncSSLSocketTest.cpp,   12,        15
io/AsyncSocketTest2.cpp,     10,        17
folly/AsyncSSLSocket.cpp,     9,        10
...
```

Git analytics via Code Maat https://github.com/adamtornhill/code-maat

Case study:
Visualizing Coordination Bottlenecks in Folly



Enclosure diagram: a hierarchical structure that follows the structure of the source code.

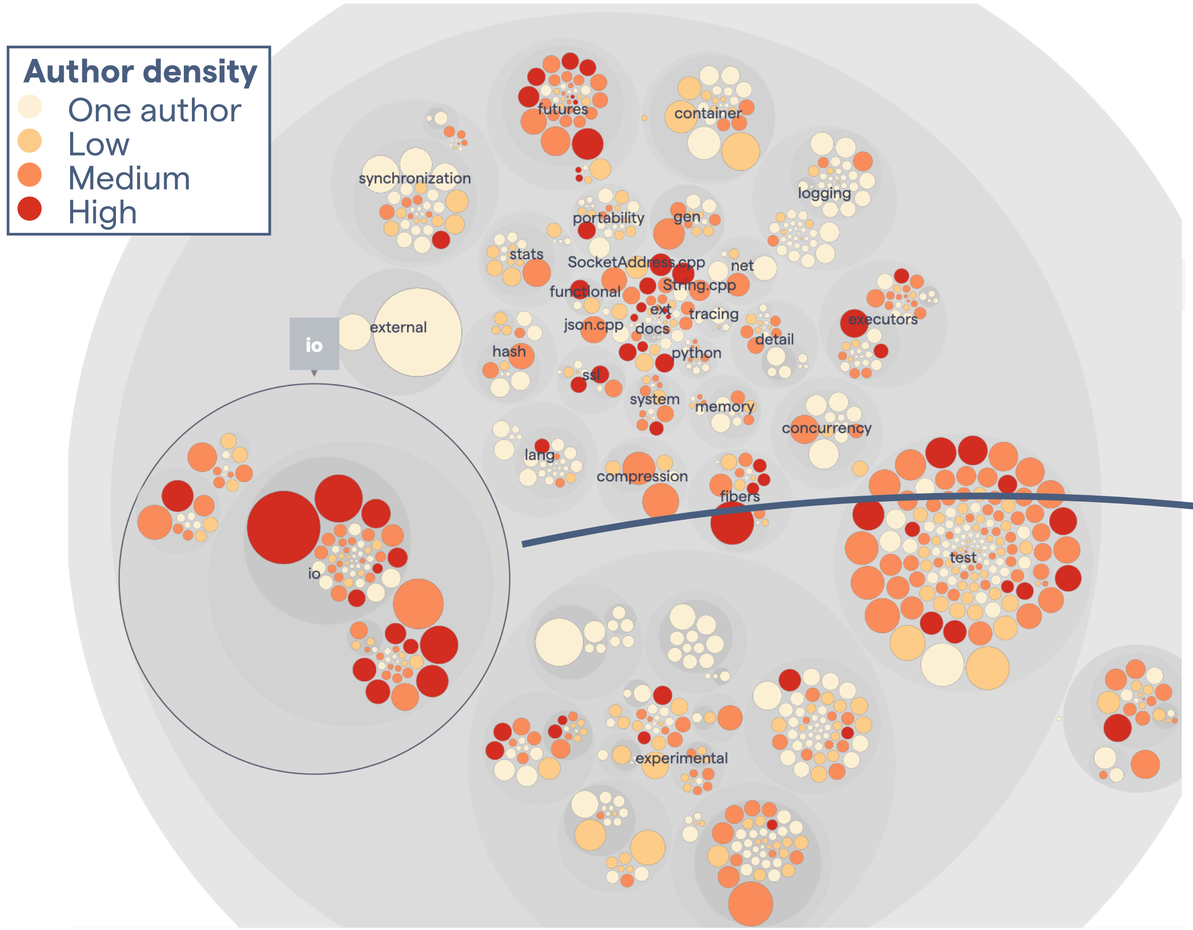
Each file is visualized as a coloured circle:

- Size: lines of code in that file.
- Color: author density

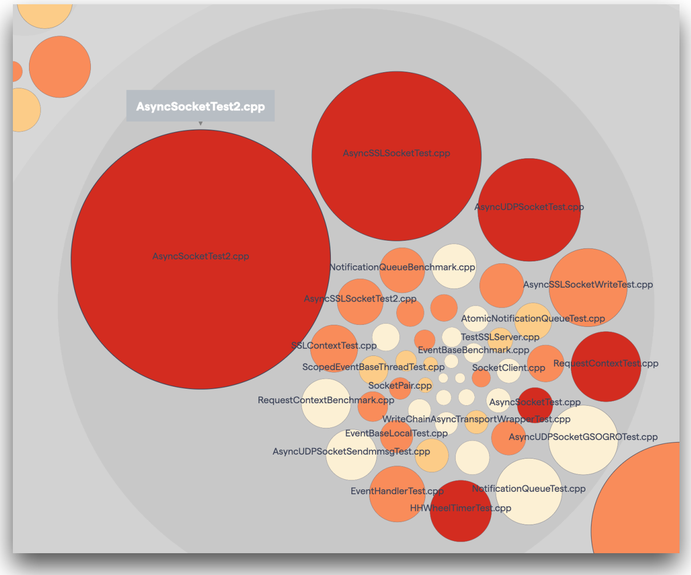
Author density
- One author
- Low
- Medium
- High

https://github.com/facebook/folly

Case study:
Visualizing Coordination Bottlenecks in Folly



58 (!) authors contributing to the same piece of code.



Explore the **Technical** Root Cause of Excess Coordination

A tiny fragment of *AsyncSSLSocket.cpp* — the class has more than 100 functions of similar complexity:

```
ssize_t bytes;
uint32_t buffersStolen = 0;
auto sslWriteBuf = buf;
if ((len < minWriteSize_) && ((i + 1) < count)) {
  // Combine this buffer with part or all of the next buffers in
  // order to avoid really small-grained calls to SSL_write().
  // Each call to SSL_write() produces a separate record in
  // the egress SSL stream, and we've found that some low-end
  // mobile clients can't handle receiving an HTTP response
  // header and the first part of the response body in two
  // separate SSL records (even if those two records are in
  // the same TCP packet).

  if (combinedBuf == nullptr) {
    if (minWriteSize_ > MAX_STACK_BUF_SIZE) {
      // Allocate the buffer on heap
      combinedBuf = new char[minWriteSize_];
    } else {
      // Allocate the buffer on stack
      combinedBuf = (char*)alloca(minWriteSize_);
    }
  }
  assert(combinedBuf != nullptr);
  sslWriteBuf = combinedBuf;

  memcpy(combinedBuf, buf, len);
  do {
    // INVARIANT: i + buffersStolen == complete chunks serialized
    uint32_t nextIndex = i + buffersStolen + 1;
    bytesStolenFromNextBuffer =
        std::min(vec[nextIndex].iov_len, minWriteSize_ - len);
    if (bytesStolenFromNextBuffer > 0) {
      assert(vec[nextIndex].iov_base != nullptr);
```

Heuristic: a comment block close to a special case — denoted by an if-statement — indicates a separate responsibility which would be better expressed in its own function.

Notice how multiple chunks of Deep Nested Logic work to form the Bumpy Road code smell. Cognitively challenging.

Meet the God Class

A God Class is a design smell used to describe classes which compulsively collect responsibilities and contain complex logic.

Consequence: God Classes centralizes the behavior of the system, which impacts the organizational dynamics.

God Classes are the Traffic Jams of the Software World

☠️ Cognitive Consequences of Developer Congestion

God Classes force programmers working on independent features into the same parts of the code, making it virtually impossible to maintain effective mental models.

Cohesion is a fundamental property for scaling the people side of code

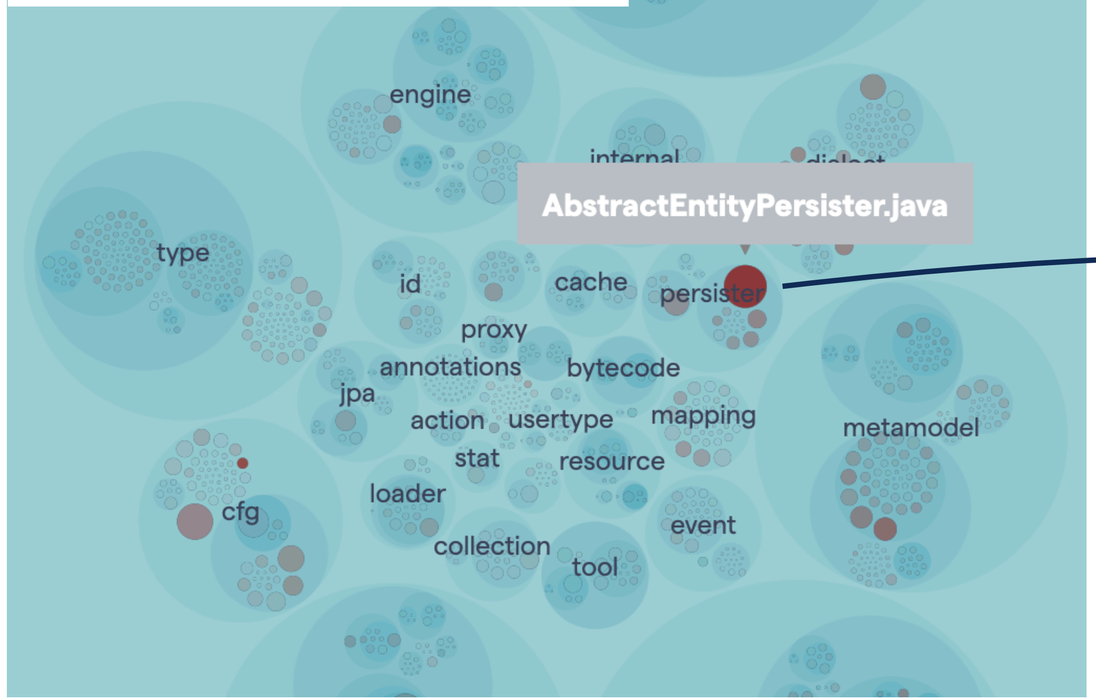
> “The gaps between the computed coordination requirements and the actual coordination activities had major implications on software failures.”

Cataldo, M., & Herbsleb, J. D. (2012).

Coordination breakdowns and their impact on development productivity and software failures.

Case Study: Cohesion in Hibernate

God Class in the Hibernate ORM



+5000 lines of code in the hotspot, indicating low cohesion

`AbstractEntityPersister.java` has 380 methods

```
getSQLUpdateByRowIdStrings
isAffectedByEnabledFilters
initializeEnhancedEntityUsedAsProxy
afterInitialize
doLoad
instantiate
getNaturalIdentifierSnapshot
getConcreteProxyClass
loadEntityIdByNaturalId
postInstantiate
handleNaturalIdReattachment
getMappedClass
getIdByUniqueKey
createProxy
getDatabaseSnapshot
...
```

Interactive visualization: https://codescene.io/projects/29298/jobs/594242/results/code/hotspots/system-map

Refactoring for Cohesion: a socio-technical fit

`AbstractEntityPersister.java` has 380 methods

Regroup them according to responsibilities, and use the groups to determine cohesive design elements (classes, modules)

```
getSQLUpdateByRowIdStrings
isAffectedByEnabledFilters
initializeEnhancedEntityUsedAsProxy
afterInitialize
doLoad
instantiate
getNaturalIdentifierSnapshot
getConcreteProxyClass
loadEntityIdByNaturalId
postInstantiate
handleNaturalIdReattachment
getMappedClass
getIdByUniqueKey
createProxy
getDatabaseSnapshot
...
```

Proxy

Improves performance by offering lazy loading

```
getConcreteProxyClass
createProxy
```

LifeCycle

Each entity passes through the different stages, e.g. transient, persistent.

```
afterInitialize
postInstantiate
```

NaturalEntityIdentity

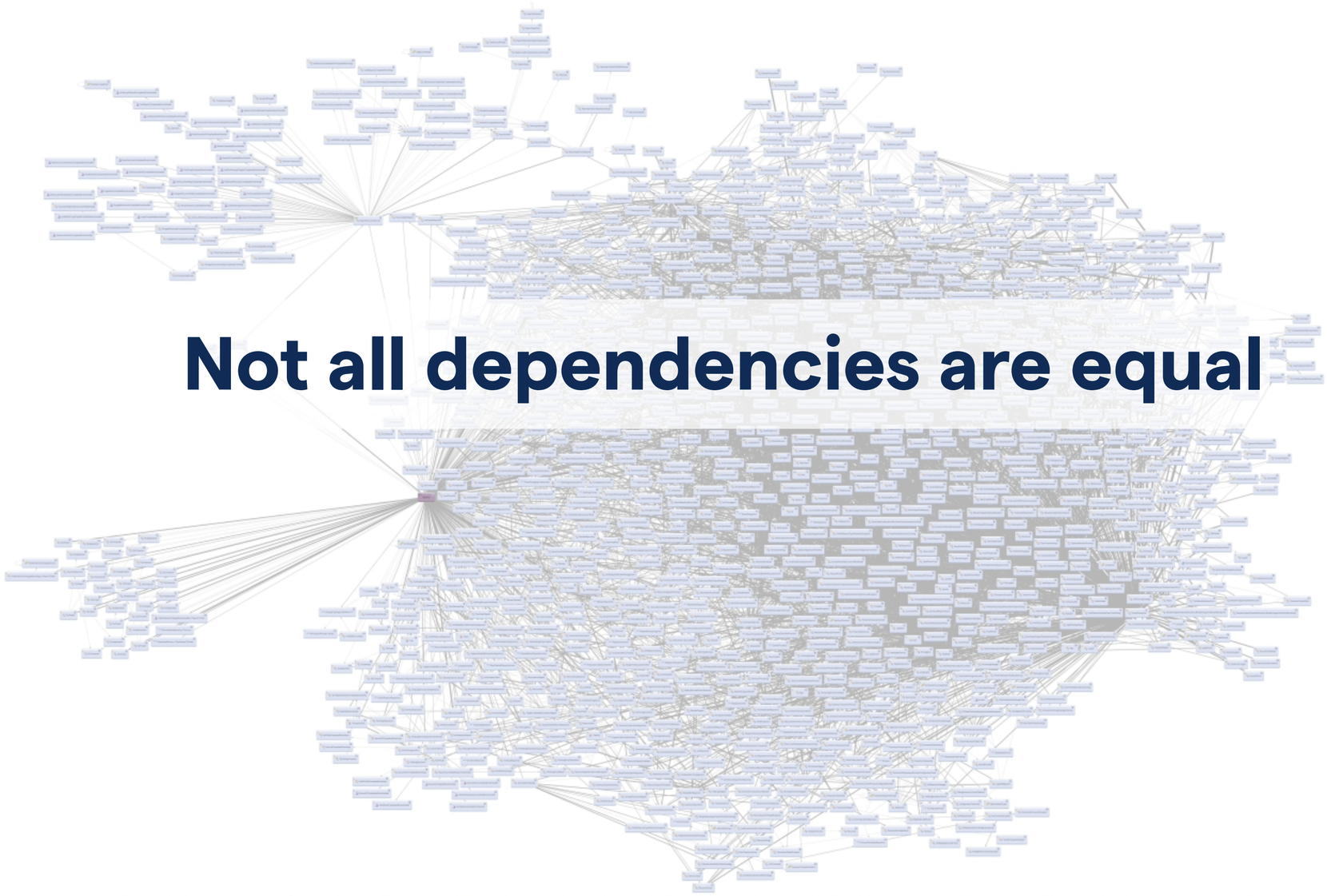
A natural ID uniquely identifies each entity.

```
getNaturalIdentifierSnapshot
loadEntityIdByNaturalId
handleNaturalIdReattachment
```

Natural boundaries for developers

Socio-Technical Smell #3:

A Propagating Cost of Change

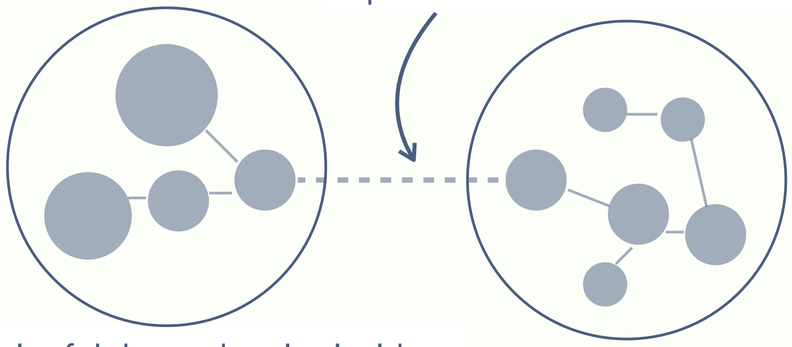


Not all dependencies are equal

Cohesion Constraints the People Organization

High cohesion

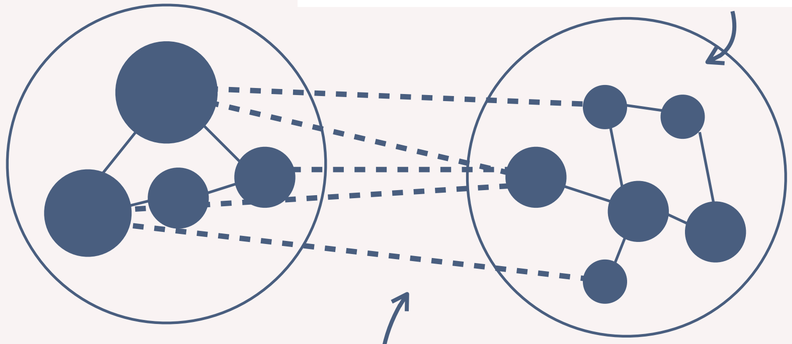
Keep inter-package dependencies to a minimum.



Meaningful dependencies inside a package indicate that the design elements belong together.

Low cohesion

In low cohesion designs, the intra-package dependencies tend to be tight too.



Unmanaged dependencies lead to code changes rippling across modular boundaries.

Code changes for a reason

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

Introducing Change Coupling: logical dependencies

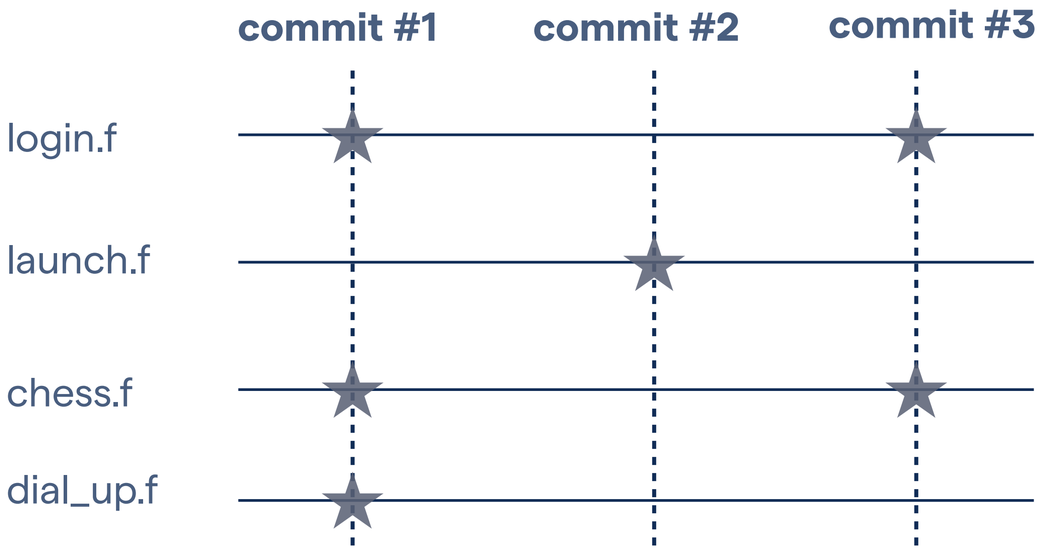
```
commit 796d31809b3683083d3b62ccbab4f00dec8ffb1f
Author: David Lightman
Date:   Fri Aug 12 13:27:53 2022 -0700

    Support multiple grades of fuel (small fix, right).

402     2       src/detectors/FuelGradeDetector.cs
1       0       src/engine/FuelInjector.cs
13      1       src/engine/Engine.cs
65      3       src/statistics/Diagnostics.cs
2       1       test/engine/FuelInjectionTest.cs

commit 32baab38f8f48f629ccd3f7564251b91dc2d777d
Author: Stephen Falken
Date:   Thu Aug 11 23:04:45 2022 -0400

    Increase the pressure to use the fuel more efficiently.

12      5       src/engine/FuelInjector.cs
16      82      src/engine/Engine.cs
6       3       src/statistics/Diagnostics.cs
270     0       test/engine/FuelInjectionTest.cs
```

Second commit

First commit

Files which are co-changing as part of multiple commits

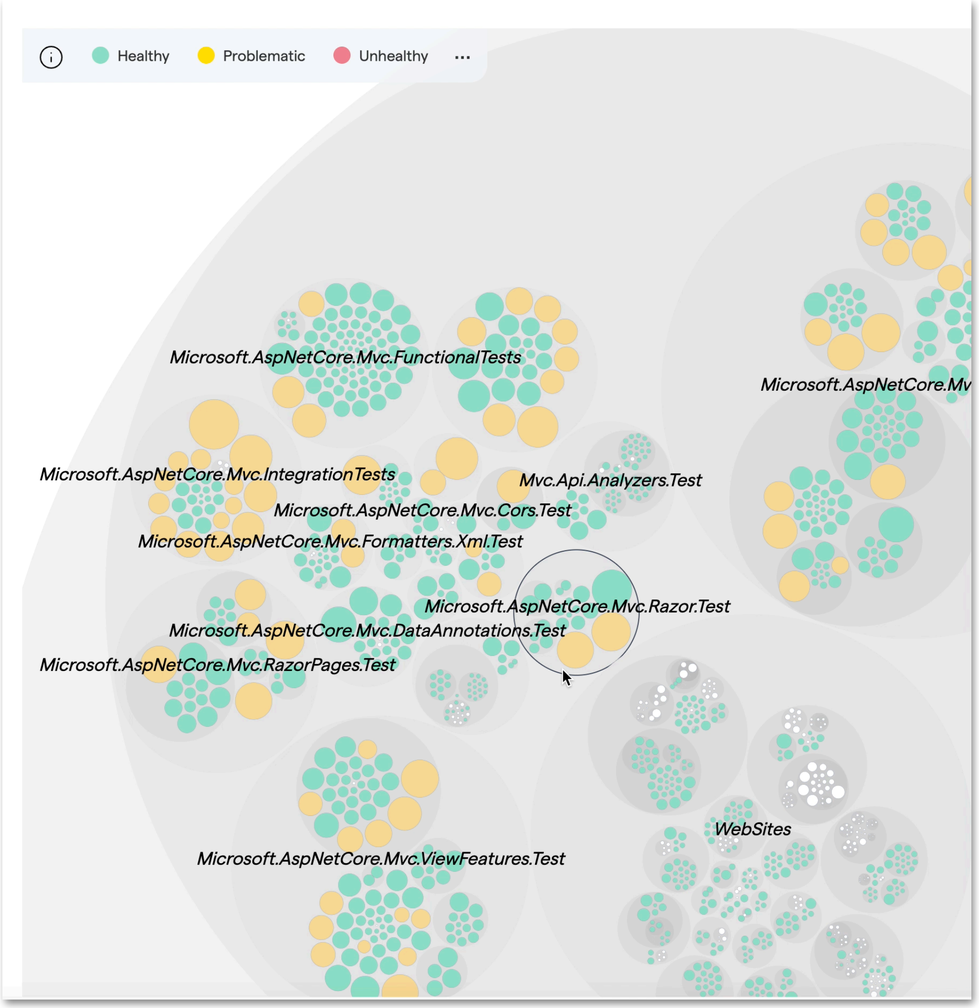
Introducing Change Coupling: logical dependencies



The `login` and *`chess`* files keep co-changing — there has to be some logical relationship between them.

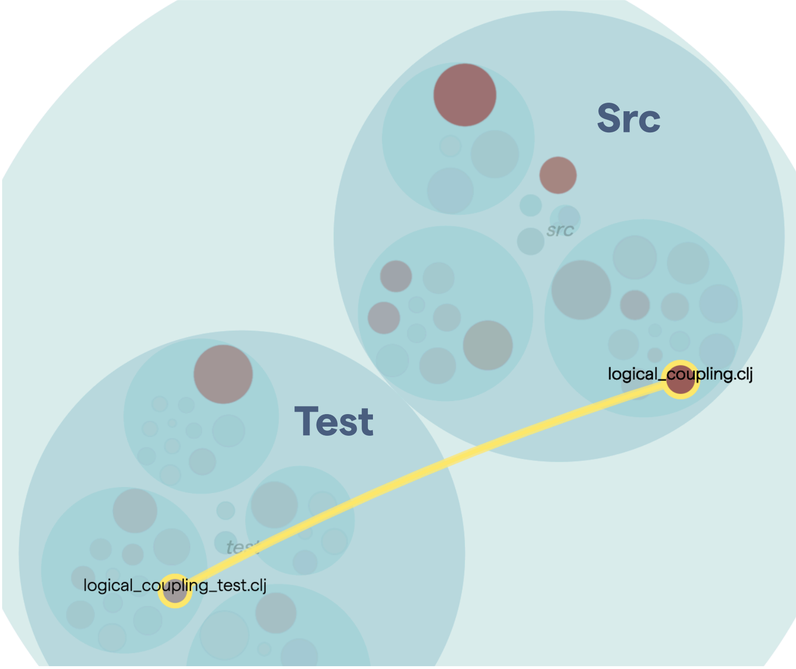
Example inspired by https://en.wikipedia.org/wiki/WarGames

Use Change Coupling to Visualizing Package Cohesion



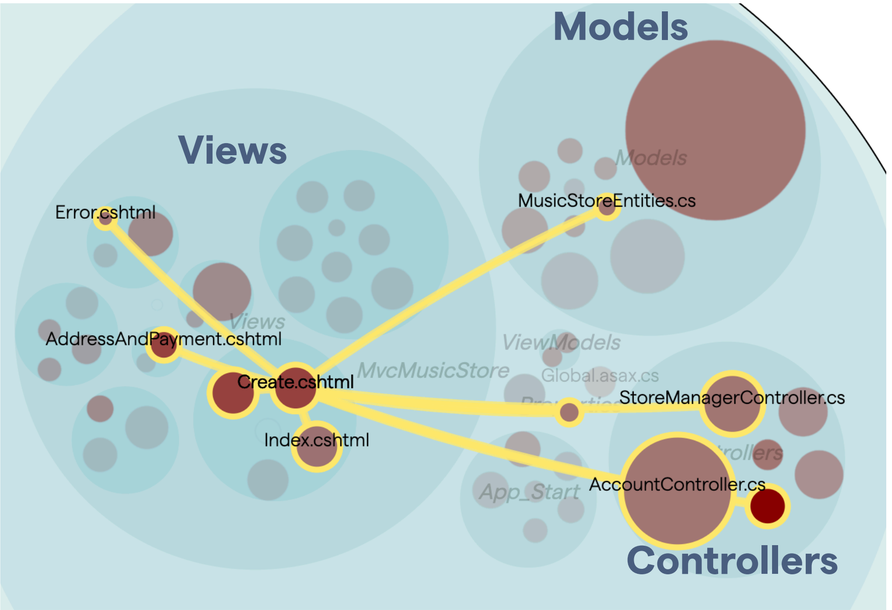
https://github.com/dotnet/aspnetcore

Acting on Change Coupling: **Contextualize**



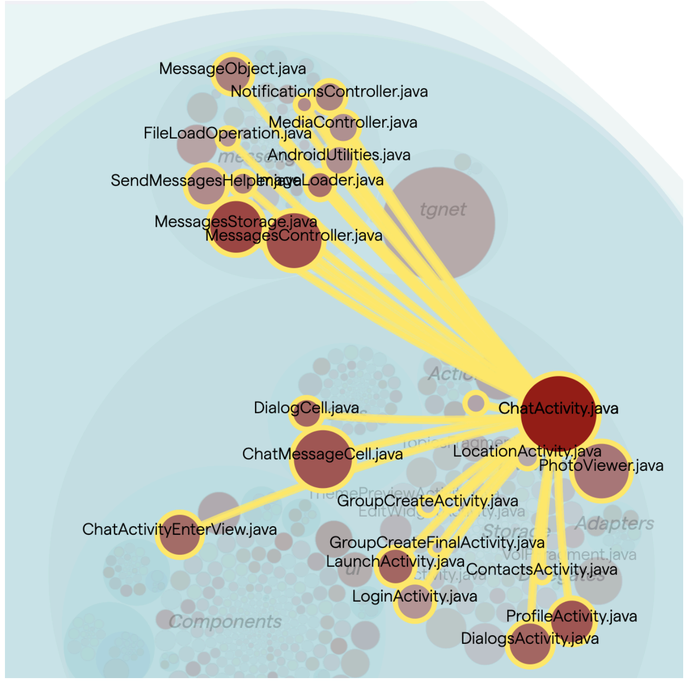
Desired change patterns: a unit test co-evolves with the code under test.

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



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

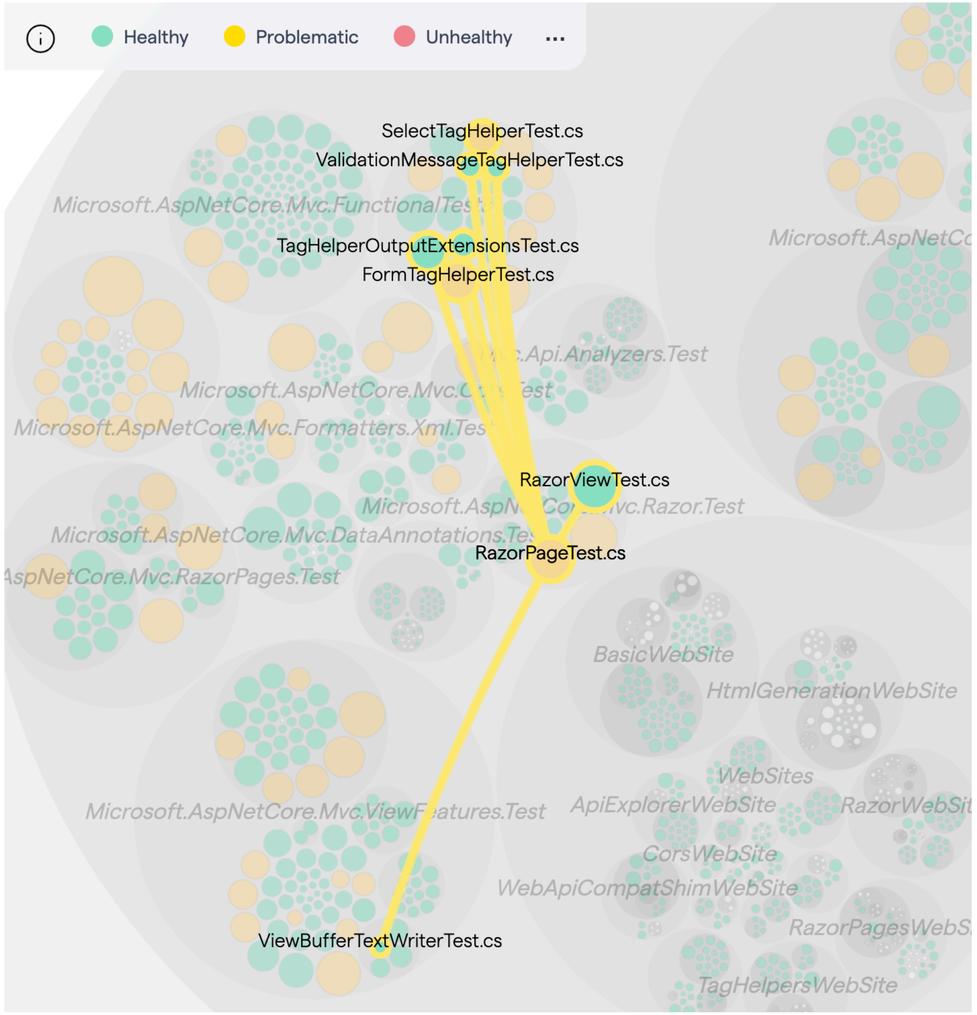
https://github.com/SebastiaanLubbers/MvcMusicStore



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

https://github.com/DrKLO/Telegram

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.

Socio-Technical Smell #4:

Dependent Work Crossing Team Boundaries

Avoid Shotgun Surgery

Align your architecture with the problem domain to create natural team boundaries.

Socio-Technical Smell #5:

Unhealthy Code with a Low Truck Factor

The Technical Debt That Wasn't

Product #1

Product #2

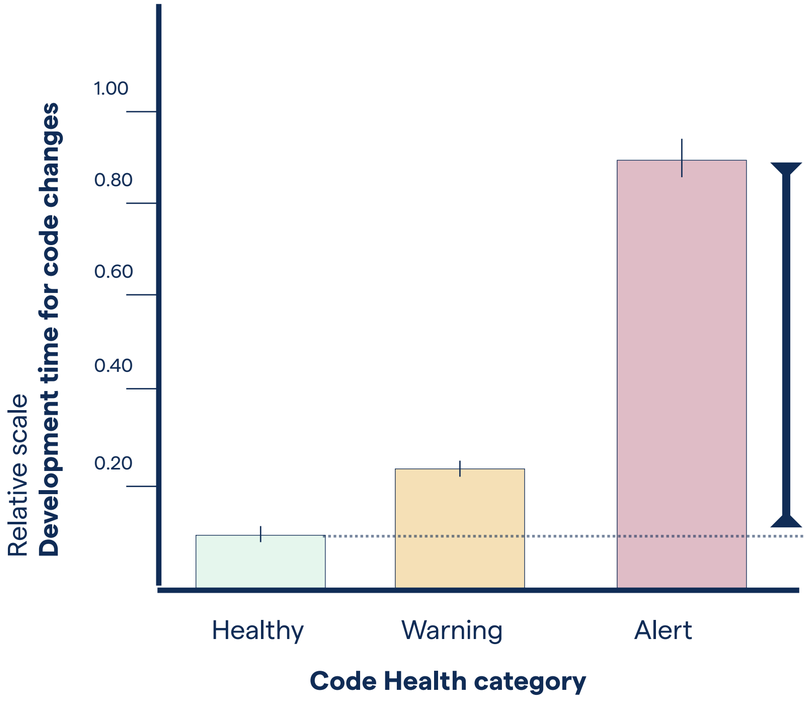
Product #3

?

Don’t Confuse a lack of Familiarity with Complexity

Unhealthy Code: Unfamiliarity breeds risk

Task completions times in unhealthy code can be up to 10x longer than 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

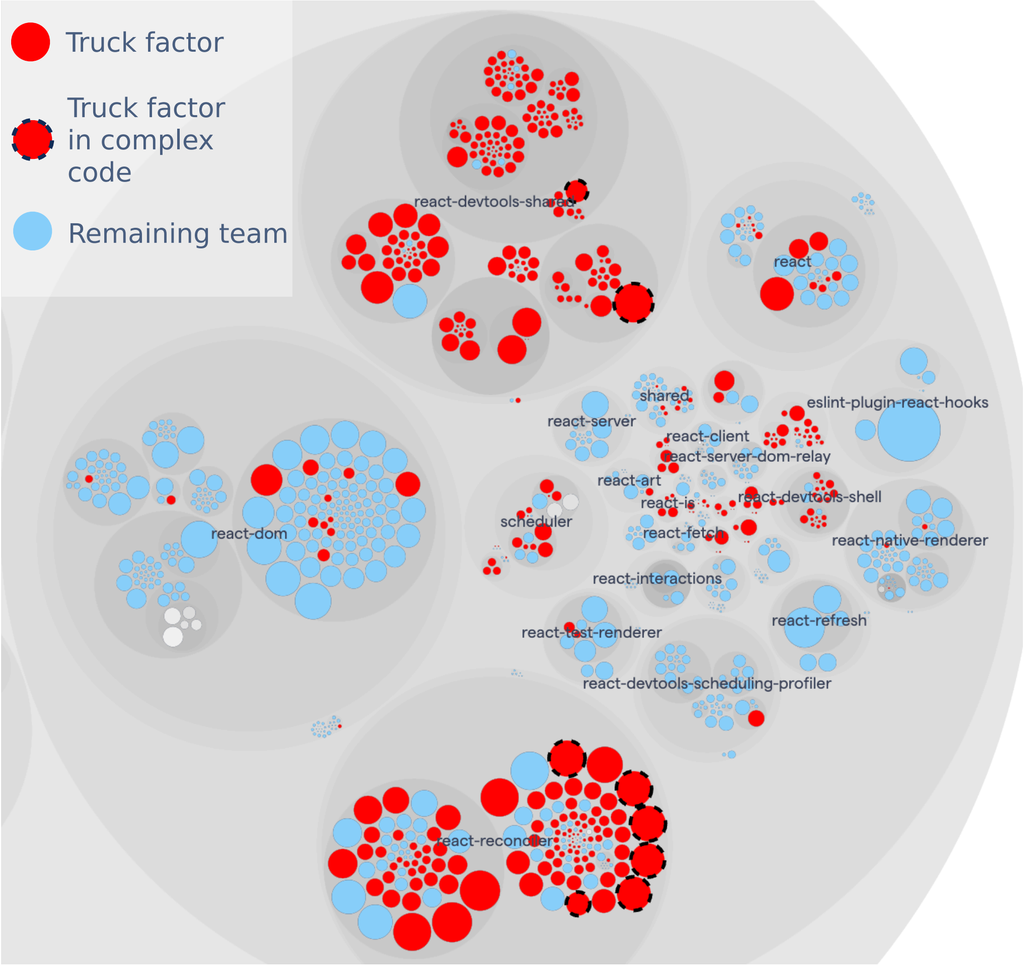
- 45% more time for small tasks, and
- 93% more time for large tasks compared to Green Code.

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

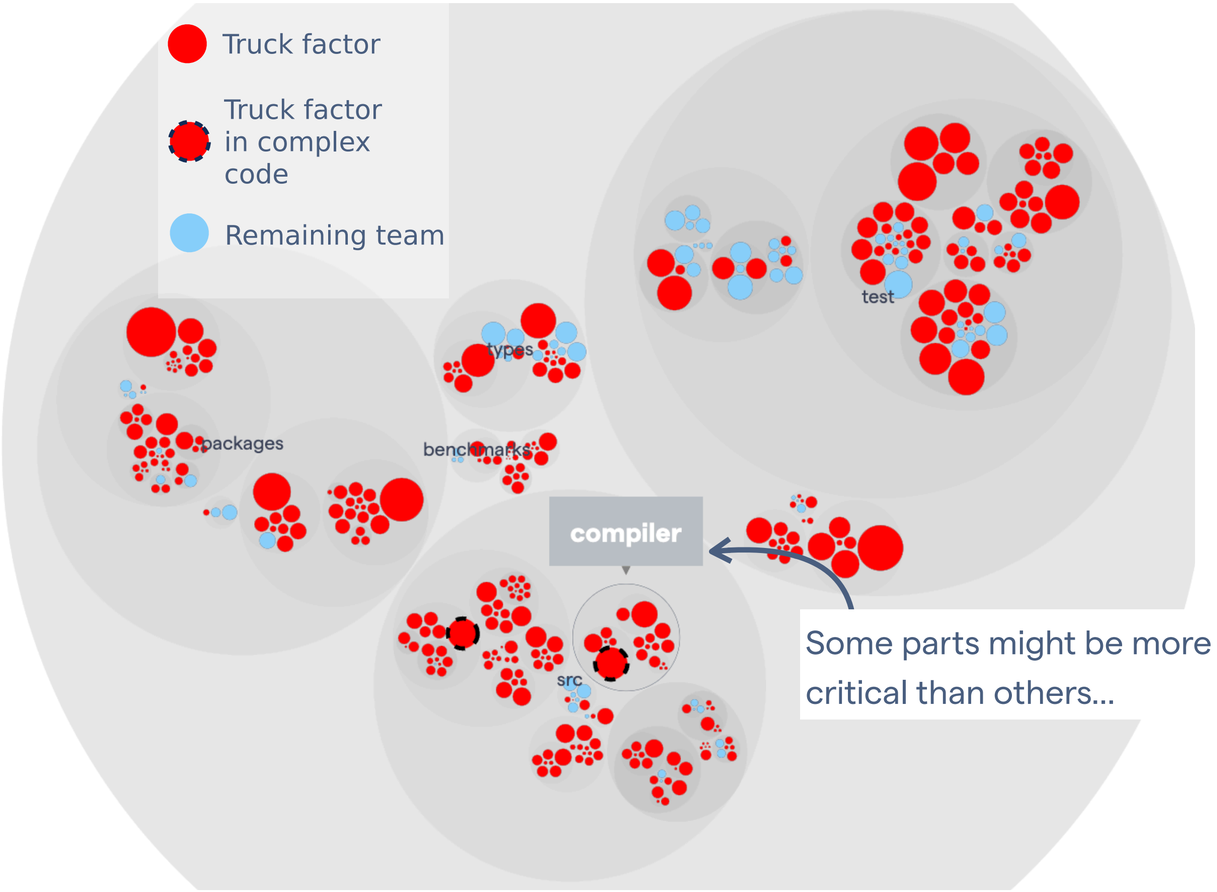
Meet the ~~Bus~~ ~~Lottery~~ Truck Factor

CodeScene

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

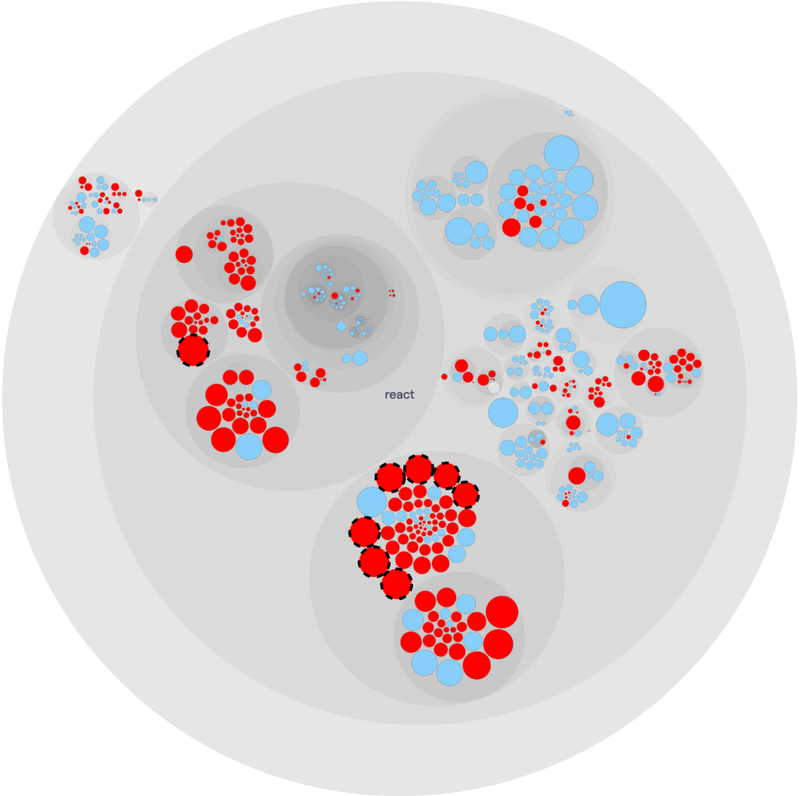


The Truck Factor in React: 2 people



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.

Socio-Technical Smells cause **Organizational Friction**

#1 — The Overcrowded System

#4 — Dependent Work Crossing Team Boundaries

#2 — Coordination Bottlenecks in the Code

#5 — Unhealthy Code with a Low Truck Factor

#3 — A Propagating Cost of Change

Summary

Improving code reduces organizational friction

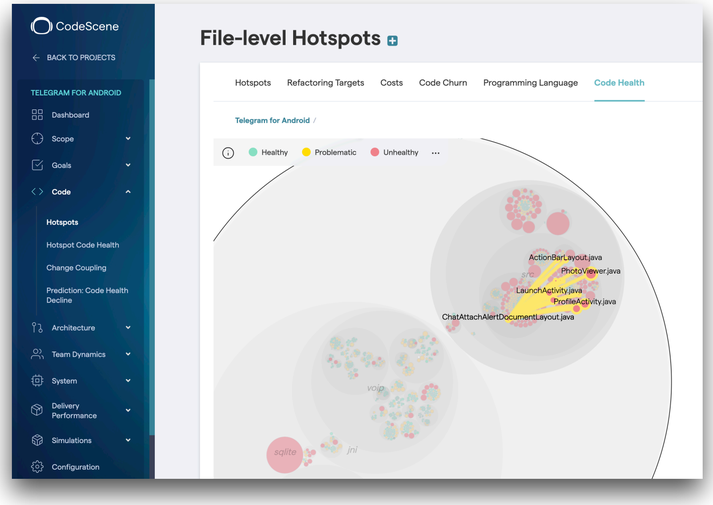
Design for a socio-technical alignment:

- **simpler on-boarding** by flattening the [technical] learning curve,
- **reduce key person risks** by making code cognitively affordable,
- **decouple teams via cohesive designs** to protect features from each other,
- **minimize defects** by avoiding excess coordination in code, and
- **improve team morale** by increasing developer happiness.

Learn more: references, books, and tools

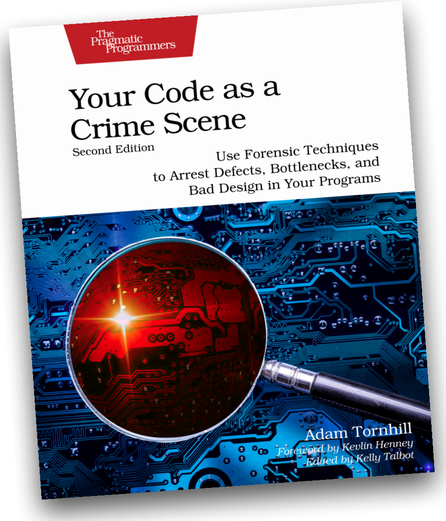
Research papers:

- **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
- **Happiness and the Productivity of Software Engineers**:
  https://arxiv.org/pdf/1904.08239.pdf
- **The Influence of Organizational Structure On Software Quality**:
  https://www.microsoft.com/en-us/research/publication/the-influence-of-organizational-structure-on-software-quality-an-empirical-case-study/



The CodeScene tool for analysis + visualizations:

https://codescene.com/



Your Code as a Crime Scene, 2nd ed (2023):

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

Adam Tornhill

https://twitter.com/AdamTornhill