Root Cause Analysis of Software Aging and Decay

CORE Group - Philip König, Fabian Obermann, Dennis Toth, Alexander Schatten

Background & Motivation

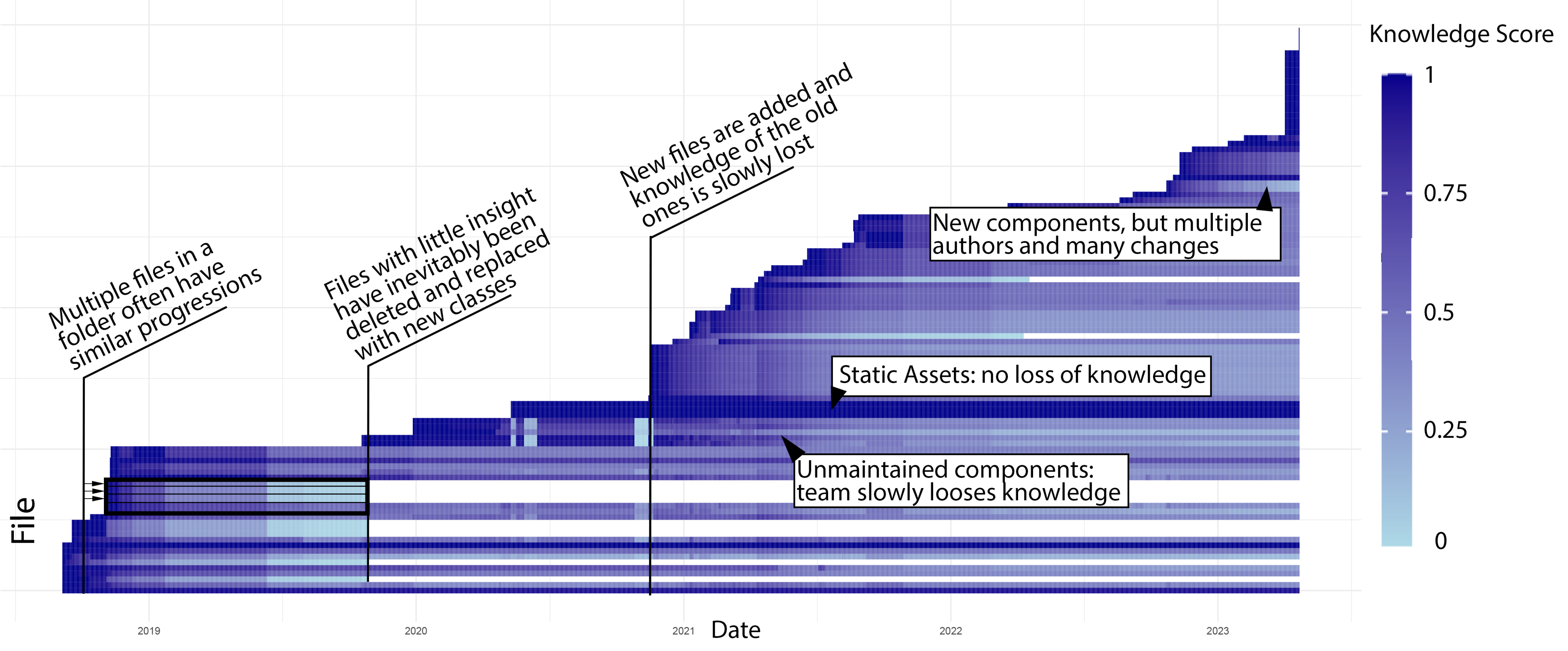
- **What is the goal for this work?**
  We aim to understand software decay and aging in complex software systems using principles from biological aging.
- **What is our research question?**
  Can we develop a method to measure systemic aging processes and predict future risks in software systems using bioinspired metrics?
- **What did we do?**
  In biological systems, aging as in loss of function occurs when maintenance mechanisms themselves age and degrade. We approximate this phenomenon in software systems using Degree of Knowledge[1] and Code Change Entropy[2] to model potential degradation of software maintenance mechanisms and thus aging and decay.

Contact Us

Degree of Knowledge

- The DoK score incorporates developer contributions, such as lines of code added, modified, or deleted, as well as file history, which considers commit frequency and recency.
- To effectively capture a developer's familiarity with a file, the score also integrates an exponential decay factor, which assigns less weight to older contributions.
- These factors are combined using a weighted formula that balances their relative importance. This allows for a comprehensive assessment of a developer's familiarity with a file.

Knowledge Decay



Multiple files in a folder often have similar progressions

Files with little insight have inevitably been deleted and replaced with new classes

New files are added and knowledge of the old ones is slowly lost

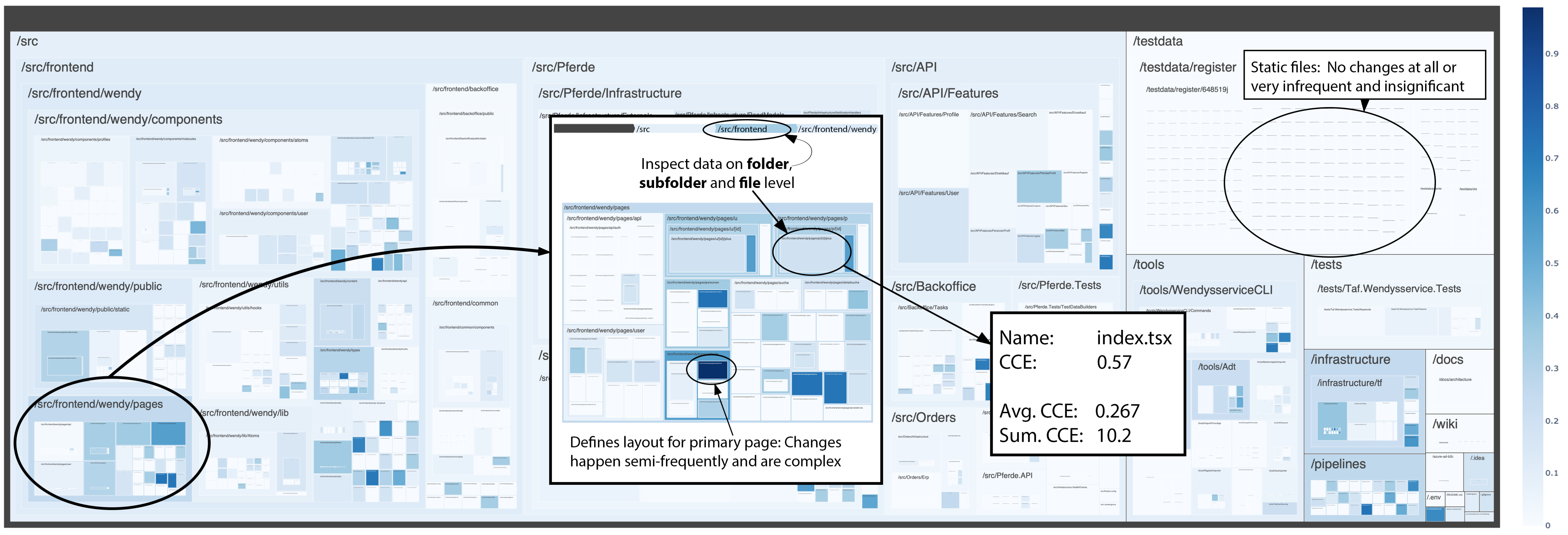
New components, but multiple authors and many changes

Static Assets: no loss of knowledge

Unmaintained components: team slowly looses knowledge

Figure 1: x-axis represents time, y-axis represents individual files within a software project. Each horizontal cell and thus file is color-coded based on the Degree of Knowledge score, ranging from 0 (low familiarity) to 1 (high familiarity). This highlights the temporal evolution of developers' familiarity with different parts of the software, revealing potential areas of knowledge decay and increased risk for introducing bugs.

Code Change Entropy



/src /src/frontend /src/frontend/wendy

Inspect data on **folder**, **subfolder** and **file** level

Defines layout for primary page: Changes happen semi-frequently and are complex

Static files: No changes at all or very infrequent and insignificant

Name: index.tsx
CCE: 0.57

Avg. CCE: 0.267
Sum. CCE: 10.2

- Code Change Entropy (CCE) is calculated via factors such as the frequency and complexity of code changes made by developers in a software project. A high CCE value indicates a more complex code change pattern, which implies a higher risk of future faults.
- Low CCE values represent simpler, more localized changes, suggesting a lower likelihood of faults and a more stable maintenance process.
- A treemap effectively represents hierarchical data, allowing us to display the distribution of CCE across files, folders, and subfolders, with darker shades of blue indicating higher entropy values.

[1] Patrick Eric Carlson. *Engaging developers in open source software projects: harnessing social and technical data mining to improve software development*. PhD thesis.
[2] Ahmed E. Hassan. Predicting faults using the complexity of code changes. In *2009 IEEE 31st International Conference on Software Engineering*, pages 78–88, 2009.

SBA Research (SBA-K1) is a COMET Centre within the framework of COMET – Competence Centers for Excellent Technologies Programme and funded by BMK, BMDW, and the federal state of Vienna. The COMET Programme is managed by FFG.