# Detecting Knowledge Islands in Software Development Processes

Christoph Gote
*Chair of Systems Design*
*ETH Zurich*
Zurich, Switzerland
cgote@ethz.ch

A knowledge island in a software development team is an individual or a small group of team members performing tasks that no other team member can perform. They can represent a severe threat to the maintenance and future development of a software product, a risk commonly expressed as the truck factor in software engineering. However, knowledge islands can remain undetected in complex development processes, e.g., the one of the German IT security company *genua* shown in Figure 1.

At *genua*, issues (i.e., bugs or feature requests) are first collected in an issue tracker. Here, they are discussed and eventually assigned to a team member who develops a change to resolve the issue. The development takes place on a code review platform, where, following a six-eye principle, the change is subsequently reviewed and integrated by two other developers. Once this process is completed, the issue is marked as resolved on the issue tracker. After a quality assurance test, the issue is finally closed. Notably, every step in this process can fail (e.g., code review or quality assurance) or be retracted (e.g., an issue assignment or a developed change), resulting in extended issue-specific action sequences, i.e., paths.

In this work, we mine the paths for all issues related to one of *genua*'s core products from 1999 to 2019, yielding over 40,000 paths. We subsequently assess the centralities of individual team members on these paths over time. To do so, we use path centrality measures based on MOGen [1], a multi-order generative model specifically designed to capture temporal patterns in path data. Such centrality measures allow us to identify which team members are essential. Employing path centrality measures is essential, as simple network models underfit the temporal characteristics of path data, whereas directly analysing the paths would result in us overfitting these patterns instead [2]. Using this approach, we detect two knowledge islands in the team's development process. First, only a single person performs quality assurance tests. Second, change integration is only performed by three team members.

In semi-structured interviews, we find that the top five team members our path analysis identified as highly central are all included in the six members the team itself considers essential. This congruence is remarkable, as we identified these team members from 176 candidates. We further find that the team is well aware of the substantial knowledge and project overview required to perform change integration. They

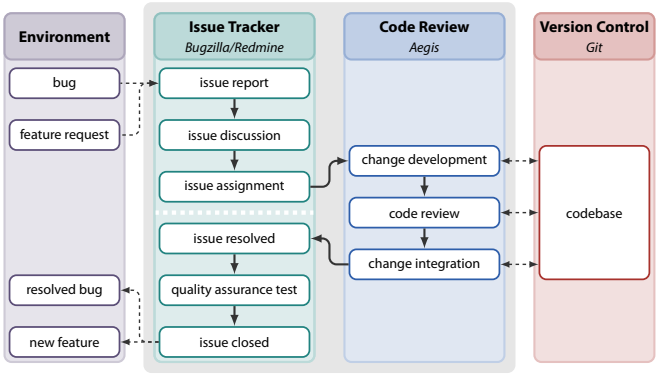


Fig. 1. Simplified representation of the development process at *genua*. The process takes place over multiple platforms. Backward loops are not displayed.

actively promote collaboration and knowledge sharing between team members through a strict code review process, Scrum, and pair programming. With three different (and changing) team members performing integrations throughout the 20-year observation period, we can validate the success of these efforts in our data. In contrast, the team is largely unaware of the single developer $D$ performing the quality assurance process. When specifically asking about $D$, we learned:

> "If [$D$] is absent or unable to perform the work, we have a massive problem." "If there are any steps taken to moderate the consequences if [$D$] was no longer there? I don't know; I haven't witnessed any."

In conclusion, we show that the complexity of software development processes can result in undetected knowledge islands in teams. By analysing how issues traverse through the development process, we can support teams in their identification and resolution.

## References

[1] Gote, C.; Casiraghi, G.; Schweitzer, F.; Scholtes, I. (2020). Predicting Sequences of Traversed Nodes in Graphs using Network Models with Multiple Higher Orders. *arXiv preprint arXiv:2007.06662* .

[2] Gote, C.; Perri, V.; Scholtes, I. (2021). Predicting Influential Higher-Order Patterns in Temporal Network Data. *arXiv preprint arXiv:2107.12100* .