CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Early detection of tipping points in Software Engineering

A Cross-Domain Perspective on Software System Instability using Early Warning Signs

Master's Thesis in Computer science and engineering

Gustav Jungstedt & Isak Borovac

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2026

Master's Thesis 2026

Early detection of tipping points in Software Engineering

Master's Thesis in Computer science and engineering

Gustav Jungstedt & Isak Borovac

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2026

Early detection of tipping points in Software Engineering
Gustav Jungstedt & Isak Borovac

Supervisor: Hans-Martin Heyn, Department of Computer Science and Engineering
Examiner: Philipp Leitner, Department of Computer Science and Engineering

Master's Thesis 2026
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2026

Gustav Jungstedt & Isak Borovac
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

This thesis investigates tipping points in Software Engineering (SE) and explores whether concepts and methods from other complex systems domains can be applied within a SE context. Software projects can experience sudden transitions from stable development into states characterized by instability, declining activity, coordination problems, or project abandonment. While tipping point theory and early warning signs (EWS) have been extensively studied in domains such as ecology, climate science, and finance, they have received limited attention within SE.

A mixed methods approach combining a systematic literature review with empirical analysis of SE datasets was applied. The literature review identified tipping point characteristics and tipping point types associated with SE activities. To evaluate the applicability of existing tipping point detection methods, three open source software datasets were analyzed using the *ewstools* framework.

The results show that tipping point behavior in SE can be characterized using a set of EWS, where variance, autocorrelation, and skewness appeared most frequently. Their distribution closely aligned with findings from domains such as ecology and climate science, suggesting that software projects may exhibit similar dynamical behaviors to other complex systems. Tipping point characteristics were identified across multiple SE activities, particularly within Management and Requirements related processes, highlighting the importance of socio-technical and organizational dynamics in software project instability.

The empirical analysis demonstrated that existing tipping point detection methods can capture meaningful instability patterns within SE datasets. However, irregular and highly variable repository activity affected the consistency of several indicators across project populations, suggesting that existing methods may require adaptation before they can be reliably applied within SE contexts.

This thesis contributes a foundation for understanding tipping points in SE and provides a basis for future research on instability detection and EWS tools within software projects.

Keywords: tipping points, early warning signs, EWS, software engineering, SE, complex systems, critical slowing down, CSD

Acknowledgements

This master's thesis was conducted during the spring of 2026 as the final part of the Master's Program Software Engineering and Technology at Chalmers University of Technology.

First of all, we would like to thank our supervisor Hans-Martin Heyn, for his support and dedication throughout the thesis process. We would also like to thank our families and friends for their continued support, not only during this thesis, but throughout our five year journey at Chalmers.

Gustav Jungstedt & Isak Borovac, Gothenburg, June 2026

Acronyms

The following list displays an alphabetical order of the acronyms used in this paper.

CSD	Critical Slowing Down
EWS	Early Warning Signs
OSS	Open Source Software
RQ	Research Question
SE	Software Engineering
SLR	Systematic Literature Review
SWEBOK	Software Engineering Body of Knowledge

Contents

List of Figures

List of Tables

1

Introduction

Software systems are becoming increasingly complex and play a critical role in modern society. As software projects evolve over time, small issues in development, coordination, or project structure can gradually accumulate and eventually lead to sudden changes in system behavior. Understanding how such instability emerges, and whether it can be identified before escalating into critical failure, has therefore become an important challenge within software engineering (SE).

To address this challenge, this study investigates tipping points in SE through a mixed methods approach that combines a systematic literature review with empirical analysis of SE datasets. The results indicate that tipping point behavior in software systems can be characterized using a small set of recurring early warning signs (EWS), and that such transitions are not confined to specific development phases but occur across multiple SE activities. The findings further suggest that several EWS established in other complex systems domains also appear in SE projects, although differences in project activity patterns may affect how reliably existing detection methods can be applied across SE datasets.

1.1 Problem Description

Software projects are complex systems that evolve continuously and depend on interactions between code, processes, tools, and people. Research in other domains, such as climate, ecology, and finance, shows that these transitions, referred to as tipping points, are often preceded by EWS that reveal declining system stability long before the transition occurs [9], [26]. Despite the conceptual relevance of tipping point theory to complex software systems, this perspective has been only minimally explored in SE, leaving a gap in how such transitions are understood and managed. This study explores the applicability of theories and methods derived from research fields such as climate science and economics to SE [9], [12].

SE lacks a coherent understanding of how tipping points manifest throughout the software development lifecycle, even though such shifts can carry potentially severe consequences. While related concepts such as project failure and system instability have been studied, they are rarely framed in terms of tipping point dynamics. Without this perspective, early signs of instability may go unrecognized, limiting

the ability of teams to intervene before conditions worsen beyond recovery. This study addresses this gap by adapting tipping point theories from other fields to the context of SE, enabling a clearer understanding of whether such transitions occur and how they can be detected early enough to prevent avoidable project failures.

To address the problem of detecting tipping points, this study first identifies their characteristics in SE and examines where they occur. These findings are then used to evaluate whether existing tipping point detection methods from other fields are applicable in a SE context.

1.2 Purpose of the Study

The goal of this research is to establish an understanding of tipping points in SE in order to develop a strategy to identify them. The study aims to define what characterizes a tipping point, as well as where and when such transitions occur within a software project. It also examines metrics that may indicate an approaching tipping point and compares these characteristics with those observed in other fields to evaluate the applicability of existing detection methods.

To achieve this, tipping point characteristics observed in SE are compared with those identified in other domains, such as climate research and financial systems. The comparison focuses on identifying similarities and differences in order to determine which attributes characterize tipping points in SE and whether existing theories and methods can be applied in this context.

To apply these concepts in practice, the study analyzes datasets identified in the primary studies included in the systematic literature review. These datasets, derived from previously published research, are used to examine whether tipping point behavior can be observed in real world SE scenarios and to evaluate the applicability of the identified characteristics and indicators.

By introducing early tipping point detection into SE, the study provides an overview of the state of tipping point detection in the domain. The results aim to support earlier intervention, reduce the risk of cascading failures, and help software projects remain stable for longer.

1.3 Significance of the Study

In the current SE literature, there is a need for research that explains why projects sometimes undergo sudden transitions instead of gradual decline. While prior work has examined project failure and project instability, these topics have not been systematically analyzed using tipping point theory from other complex system domains such as climate research. There has also been no consistent effort to define what constitutes a tipping point in SE.

This study contributes by introducing tipping point theory for understanding abrupt

transitions in SE. It provides insight into how tipping points can be defined, characterized, and detected using concepts and methods established in other domains such as climate and finance.

Given the increasing complexity and societal importance of software systems, as well as the high costs associated with missed deadlines and project failure, improving the ability to anticipate tipping points is relevant both for research and for practice. The results contribute to bridging the gap between theoretical tipping point research and practical SE applications by providing a foundation for future tools and methods aimed at improving project stability and early risk detection.

1.4 Research Questions

To fulfill the purpose of the study, three research questions (RQs) were identified.

RQ1: What characterizes tipping points in different SE activities? The purpose of this question is to establish an understanding of what defines a tipping point across a range of SE activities by identifying the key properties, patterns, and conditions that distinguish tipping points from normal project behavior.

To address this research question, a theoretical framework was established by selecting seven cross domain EWS derived from other complex systems. Through a systematic literature review, 40 primary studies were analyzed to identify how these signals behave in SE. Each instance was also classified into one of four tipping types.

The results show that variance, autocorrelation, and skewness are the most frequently observed EWS. This aligns with findings from other domains and indicates that tipping points in SE are primarily associated with increasing instability and fluctuations. Systems become more unpredictable as they approach a tipping point.

RQ2: Where in SE do such tipping points occur? This question explores which aspects of software projects and repositories exhibit tipping point characteristics.

To address this research question, tipping point occurrences in the identified primary studies were analyzed. Each characteristic was mapped to a SWEBOK knowledge area to ensure consistent categorization.

The results show that tipping points occur across most knowledge areas, with Management and Requirements being the most prominent. Requirements related tipping points are often associated with propagating behavior, meaning they tend to cascade and affect other parts of the system. Management related tipping points are frequently linked to societal impact, highlighting the importance of socio-technical factors such as communication and organizational structure.

RQ3: To what extent can methods for tipping point detection from other fields be applied to SE? This question investigates whether existing detection methods can be used to identify tipping points in SE datasets.

To address this research question, a Python package originally designed for detecting transitions in ecology and climate science was applied to SE datasets. The EWS of variance, autocorrelation, skewness and kurtosis were calculated across multiple repositories to evaluate the tipping points of project survival and developer turnover.

The results show that while these methods can successfully profile the collapse of individual projects, their universal predictive reliability in software is currently limited. The sparse data from the datasets mathematically infuses instability, frequently causing disturbances in the calculations. While the theoretical crossover is highly promising, existing ecological tools require domain-specific calibration to handle human activity data before they can be reliably applied.

1.5 Limitations & Delimitations

This study is subject to a number of limitations and delimitations that define the scope and influence the interpretation of the results.

The study is delimited to the development of a conceptual framework for detecting tipping points in SE, rather than the implementation of a fully functional detection tool. The outcome is intended as a framework that identifies relevant indicators and relationships, serving as a foundation for future work.

The analysis is limited to a selected set of EWS derived from existing literature, as well as to the use of SWEBOK knowledge areas for categorization. Additionally, the study is based on a systematic literature review covering 2015 to 2026 and selected datasets, which may not fully represent industrial or proprietary software environments.

A key limitation is the limited availability of direct research on tipping points in SE, requiring interpretation of related concepts such as project instability and failure factors.

2

Background

Complex systems across many scientific domains exhibit sudden, nonlinear, and often irreversible shifts from a stable into a chaotic behavior known as tipping points. Such abrupt transitions have been observed in ecosystems, climate systems, financial markets and medicine, where minor changes can escalate and cause major and rapid transformations of a system [26]. Software systems are similarly complex, adaptive and dependent on each other, and they too can undergo abrupt changes with little advance indication. These changes are not necessarily negative. For instance, a project may experience a sudden spike in contributions or user adoption due to increased popularity or a commercial transition. In other cases, however, rapid changes can instead be associated with instability and degradation. Despite extensive research on tipping points in other fields, comparatively little attention has been given to SE from this perspective.

In SE, tipping points can arise from isolated factors, or from interactions between several factors that gradually form a cascading chain of events. By ignoring minor issues in software, the risk of issues escalating into larger project problems increases [23]. From a software economics perspective, errors which appear early in a project have been studied to cost 50 to 200 times more to correct later in development compared to addressing them immediately [31]. As unresolved issues accumulate, the complexity of a software project may eventually reach a threshold where development becomes increasingly difficult to manage.

Tipping points may also trigger cascading effects throughout a project. As complexity increases, developers may need to shift resources away from planned development activities in order to manage growing instability. Over time, sustained exposure to high workloads and stress may also contribute to burnout among developers [13]. A defining characteristic of tipping points is that they are often difficult to reverse once a transition has occurred, which in severe cases may contribute to project abandonment.

Tipping points in SE have yet to be examined explicitly, but the topic has indirectly been approached through studies of EWS and system stability. Research on the failures of software projects shows that these projects often display subtle indications of instability long before major breakdowns occur [23]. These early indications have previously been referred to as “weak signals” and defined as “A weak signal is a

factor for change hardly perceptible at present, but which will constitute a strong trend in the future" [23]. Another closely related concept is "Early Warnings," defined as "An early warning is an observation, a signal, a message or some other item that is or can be seen as an expression, an indication, a proof, or a sign of the existence of some future or incipient positive or negative issue. It is a signal, omen, or indication of future developments." [30]. Indicators of weak signals and early warnings include communication breakdowns, poor leadership, and lack of expertise within development teams, all of which are commonly associated with project instability.

More recent work has begun treating software repositories as dynamic systems through proposed stability indicators. These indicators include commit patterns, issue resolution rate, pull request processing and community engagement. The goal is to understand how these indicators correlate to a system's ability to return to an equilibrium following sudden disturbances [6]. While these studies do not explicitly address tipping points, they demonstrate increasing interest in understanding instability and abrupt transitions in software projects.

Research on tipping points in fields such as ecology, economy and medicine has developed a wide range of concepts and analytical tools that could potentially be applied to SE. Studies have shown that systems approaching critical transitions often exhibit identifiable patterns regardless of the underlying domain [26]. One important concept used to describe such transitions is bifurcation, where gradual changes in underlying conditions cause a system to lose stability and shift abruptly into an alternate state. This loss of stability is connected to changes in a system's dominant eigenvalue, which approaches zero as the system nears a critical threshold, leading to slower recovery from disturbances, a phenomenon known as critical slowing down [26]. As a result, systems approaching a tipping point often exhibit increased fluctuations, including rising variance and autocorrelation. Because these indicators have been observed across several different domains, similar early warning patterns may also exist within SE systems.

In practice, software projects may experience sudden spikes in issues, declines in productivity, or abrupt reductions in repository activity. These disruptions are often connected to accumulated technical debt, coordination problems, or changes in contributor dynamics that gradually build up over time. Such behavior may appear through increasing issue resolution times, growing variance in commit activity, or increasing dependence on a small number of contributors. While these changes may appear manageable when viewed individually, their combined effect may indicate declining system stability. When additional disturbances occur, such as major requirement changes, tightened deadlines, or the departure of key developers, projects may rapidly transition from stable development into increasingly unstable states.

2.1 Tipping Points

Tipping points represent critical thresholds where even a 'slow drift' in external conditions can fundamentally alter a system's dynamical tendencies. These points are significant because they govern the possible behaviors of the system. While tipping point theory remains an under-explored framework within the SE domain, it is a well-established paradigm in ecology used to identify the thresholds that govern the possible behaviors of complex dynamical systems. An example of a tipping point in that field is the transition of a shallow lake. They are often found in one of two states. Either being dominated by aquatic vegetation or by algae. This transition is dependent upon nutrients in the lake. A lake which is dominated by aquatic vegetation can transform in to the other state, for example by an influx of phosphorus. When the phosphorus reaches a specific threshold, the system enters a critical transition from a clear state dominated by aquatic vegetation to a turbid state dominated by algae [14].

2.2 Critical Slowing Down

The phenomenon of critical slowing down (CSD) in regards to systems is defined as the systems inability to regain stability after being forced towards a tipping point. It becomes slower at recovering from minor disturbances [9]. A healthy system would appear to have a higher resilience, not being as affected by perturbations as CSD systems. A CSD system can eventually become weak enough that one small disruption can send the system over a specific threshold, leading to a critical transition which can be defined as reaching a tipping point. Most established tipping points and early warning theories signs are related to this phenomenon [12]. There are also tipping points which are non-CSD related. While CSD measures the diminishing resilience of a system, non-CSD related EWS focus on the distributional properties or structural organization of the systems state. These signals can identify a collapse which occurs independently of the systems ability to regain stability. In Table 2.1 is a categorization of the EWS and if they are CSD related or not.

Table 2.1: Categorization of CSD and Non-CSD Related EWS [9]

CSD Related	**Non-CSD Related**
Variance	Skewness
Autocorrelation	Kurtosis
Return Rate	Fisher Information
Power Spectrum	

2.3 Tipping Types

Tipping points can be interpreted as four types, namely macro, propagating, clustered or societal. In this section the types of tipping will be explained and also how

they are interpreted in this study for the SE domain. The tipping point types are all derived from the 2023 paper by Lenton et al. called “Remotely sensing potential climate change tipping points across scales” [12].

Macro

Macro tipping are considered “major” tipping point events. In the case of Earths system, it would involve atmospheric events like monsoons or the destruction of large ice sheets, where the crucial reinforcing feedback mechanisms that drive tipping points operate across vast spatial scales [12]. In terms of the SE domain, macro tipping would represent a large scale transition that affects the entire ecosystem of the project. Unlike a small bug in the code, this is an event where the entire platform shifts into a completely different and often worse way of functioning. This transition is usually driven by a negative cycle where problems feed into each other and grow automatically. For example, a project might hit a “point of no return” when years of messy code or outdated technology become so widespread that they weaken the entire system’s ability to handle even minor changes. When this happens, the project’s natural behavior changes. It enters a phase where the team can no longer build new features and must spend all their time just trying to keep the system from collapsing.

Propagating

Propagating tipping occurs when a system is composed of a network of smaller, connected components where causal interactions allow a localized transition to spread. In these systems, tipping one component alters the likelihood of tipping another. In some cases, tipping of the most sensitive components can destabilize the entire network, creating a “domino cascade” [12]. In the field of SE, this could be due to volatile requirements. Changing a core requirement can cause a ripple effect which leads to many parts of the project needing to change. A propagating tipping point starts as a localized failure, but eventually affects other interconnected parts of the system.

Clustered

Clustered tipping occur when smaller localized parts of a system reach their tipping points at the same time. Unlike propagating tipping points, where one tipping point event can trigger another, clustered tipping points transpire due to each part of the system being struck by the same external force. The parts being affected may not be “physically” connected, but they all cross their thresholds together because they are influenced by the same factor [12]. An example of a clustered tipping in SE would be a sudden spike in usage of the product. If the servers are not prepared for that influx of traffic, it might slow down every part of the platform all at once.

Societal Impact

Some tipping points can have a great effect on societal systems. An example of a societal impact tipping point in another domain is heatwaves. Heatwaves can impact many vital systems in human society, for example by drying out agricultural land. This can lead to the whole agricultural system to tip into failure, leading to other cascading social issues like food crisis [12]. While these events can share structural mechanisms with macro, propagating, or clustered tipping points, they are specifically categorized as "societal impact" when they primarily affect human societies. Societal impact tipping points inside of the SE domain would be related more to the socio-technical side as opposed to raw technical metrics. Instead of monitoring the resilience of the project itself, it is about the resilience of the engineering teams. Examples of this would be managerial issues which can lead to various tipping points, such as developers leaving their positions due to not feeling satisfied with their work.

2.4 Early Warning Signs

EWS are potential indicators of tipping points. They are statistical indicators which arise from the universal behavior of complex systems as they lose resilience near a critical threshold. There are many kinds of EWS, but they are divided in to two categories, CSD related and non-CSD related. The EWS are all derived from the 2024 paper by Dakos et al. called “Tipping point detection and early warnings in climate ecological, and human systems” [9]. This section will start by going over the four EWS which are considered CSD related, followed by the three which are not. Why these seven EWS were chosen is explained later in the Methodology chapter.

Variance

Variance measures the spread of a systems state variables around their mean. As a system approaches a bifurcation point, the resilience become weaker. Consequently, each disturbance will push the system further away from equilibrium than it would in a healthy resilient system. Its the raw mathematical measurement of how far the system’s states spread out from their average behavior over a specific moving time window [9].

Autocorrelation

When a system is approaching a tipping point, each time step is more correlated to the previous time step. This measurement is calculated with lag-1 autocorrelation, known as $AR(1)$. The term "lag-1" simply means that the data at a specific time step is compared directly to the data from the single preceding time step. Tipping point research indicates that when a system operates far from a critical threshold, there is little to no correlation between steps [9], [14].

Return Rate

Return rate is a straightforward but very fundamental measurement of a systems resilience. It is the time it takes for a system to return to equilibrium following a disturbance. As the system loses resilience, the return rate decreases, signaling that the system is *critically slows down* [9].

Power Spectrum

The power spectrum, or spectral density, is the measurement used to determine how the fluctuations of a system are distributed across different frequencies. In a stable system, these fluctuations usually appear as white noise, meaning the systems vibrations consist of fast, short-term changes that are spread evenly across all frequencies. However, when a system experiences CSD, it begins to recover slower, which results in the system to shift from fast short term changes to slow and long term trends [8], [9], [14].

Skewness

Skewness is a statistical measure of the asymmetry in the distribution of a systems state variables around their mean. Variance focuses on the magnitude of fluctuations while skewness captures the direction. In a stable regime the system is equally likely to be pushed in either direction by random noise, resulting in a skewness value near zero. As a threshold nears, the system begins to "lean" toward the approaching alternative stable state [27] [25] [9].

Kurtosis

Kurtosis measures the prevalence of extreme outliers in the distribution. In the context of tipping points, kurtosis identifies when a system begins to experience high-magnitude fluctuations which deviate greatly from its normal behavior. The system stays close to the mean most of the time, but when it moves violently, lacking the flexibility of a healthy system. While kurtosis is a valuable metric, research indicates that dramatic increases often occur late in the transition process, when counteractive measures often is hard to carry out [24]. Furthermore, it is also known that real world systems often have a high kurtosis due to their inherent complexity, meaning they are predisposed to extreme events even outside of a tipping point event [25].

Fisher Information

Fisher Information is a tool from information theory used to measure the structural order within a complex system. While indicators like variance look at the size of a systems fluctuations, Fisher Information looks at whether they still follow a pattern or if they have turned into total chaos. Fisher Information identifies when a system is in a transition phase by turning many different parameters into a single index. It can spot the exact moment a system stops behaving like its old self and starts drifting

towards a new regime. Fisher Information typically drops significantly during a transition, signaling the system is breaking down [29].

3

Related Work

This chapter presents research related to tipping points and instability in SE, together with established tipping point theory from other domains. Existing systematic literature reviews and related concepts are also discussed.

3.1 Related Concepts in SE

After conducting a systematic literature review on the topic of tipping points in SE, one major observation occurred. Currently there are no primary studies that directly investigate or define tipping point transitions within the SE domain. The detailed methodology and implementation of the systematic literature review is presented in the next chapter, but the results show that there is currently a research gap on this topic within the SE domain.

In SE, there is research regarding project instability and other similar topics which could relate to tipping point theory. One such paper, made by Destefantis et al., proposes a new "Stability Index" in regards to software repositories. The stability index which measures "a system's ability to return to an equilibrium" can be directly compared to the EWS "return rate", mentioned in established tipping point theory papers [6], [9]. However, there have not yet been any research utilizing established tipping point theory as a formal analytical framework.

A secondary observation derived from the SLR is the existence of several research streams that document SE "failure factors" and related concepts. These are typically defined as elements that negatively contribute to the eventual collapse of a software project. However, they are most often treated as isolated or linear risks, rather than as interacting dynamics that may lead to sudden system-level transitions.

One prominent area is scope management and scope creep, which is consistently identified as a major driver of project failure. Uncontrolled changes in requirements lead to cascading effects on time and cost, ultimately resulting in instability. Poor scope management has been shown to strongly correlate with project failure, as increasing requirement changes introduce compounding complexity over time [S10]. Similarly, scope creep is reported as a primary cause in a large proportion of failing projects, affecting not only requirements but also cost, timeline, and quality [S6].

This suggests a gradual accumulation of changes that eventually pushes the project beyond a manageable threshold.

Another related concept is the volatility and complexity of requirements engineering (RE). Requirements are inherently uncertain and evolve over time, making systems difficult to stabilize. Empirical studies show that RE problems are highly interconnected and driven by uncertainty and stakeholder dependencies [S13]. At the same time, increasing complexity in requirements has been identified as a key driver of failure, due to its nonlinear and unpredictable nature [S16].

Research on integration and system-level failures further highlights how problems often remain latent and only surface in later stages. Integration failures are typically caused by combinations of technical and socio-technical factors, leading to delays, increased costs, and in some cases complete project breakdown [S8].

A similar pattern can be observed in open-source project dynamics, where projects frequently experience abrupt decline. Studies show that failures are often linked to factors such as lack of contributors and poor maintenance practices [S9]. In addition, many projects depend on a small number of core developers, making them structurally fragile and vulnerable to sudden collapse if these contributors leave [S7].

Closely related to this is developer turnover and socio-technical instability. Changes in team composition, particularly the loss of experienced developers, have been shown to negatively impact software quality [S15]. These effects often interact with other factors such as communication issues and poor coordination, which are also known contributors to project failure [S12].

Finally, broader research on IT project failure highlights uncertainty, volatility, and unknown factors as key contributors to unsuccessful outcomes [S12]. Rather than being caused by a single issue, failure is typically the result of interacting factors that collectively degrade system stability.

Taken together, these findings suggest that SE research already captures many of the underlying mechanisms associated with tipping points. However, these phenomena are typically studied in isolation and described as linear cause-effect relationships. What is missing is a unifying perspective that treats software projects as complex systems capable of undergoing nonlinear transitions.

3.2 Existing Systematic Literature Reviews

Several studies have explored software project success and failure through systematic literature reviews and related approaches. Across these studies, a fairly consistent set of factors appears. Common causes of failure include poor cost and time estimation, inadequate planning, scope creep, incomplete requirements, and limited user involvement [1], [15]. On the other hand, studies focusing on successful projects highlight factors such as stable requirements, skilled teams, effective management, and strong stakeholder involvement [16].

More focused reviews investigate specific mechanisms, such as scope creep, which has been shown to significantly affect project cost, timeline, and overall quality when not properly managed [S6]. A common theme across these studies is that project outcomes are rarely caused by a single issue. Instead, multiple factors tend to interact and influence each other, reinforcing the view that software projects exhibit characteristics of complex systems.

Despite these contribution, existing literature primarily conceptualizes project failure as the result of accumulated issues rather than as a dynamic transition in system behavior. Failures are typically described in terms of observable outcomes, such as delays, cost overruns, or unmet requirements. While interactions between factors are sometimes acknowledged, there is limited attention to how these interactions evolve over time or whether they can lead to sudden and irreversible changes in project performance.

In particular, concepts such as tipping points and EWS, which are well established in other domains, are largely absent from the SE literature. This highlights a gap in existing research, where the focus remains on identifying contributing factors rather than understanding how systems transition from stable to unstable states. Addressing this gap motivates the need for a cross disciplinary approach that applies tipping point theory and EWS detection methods to SE contexts.

3.3 Tipping Point Theory in Other Domains

While tipping point theory is an emerging concept in SE, it is an established cornerstone of research in other disciplines. Specifically in domains which experience a multitude of nonlinear transitions between stable states of the system. This theoretical framework has been extensively documented in ecology, climatology, and physiology. For instance, researchers have used these methods to anticipate the collapse of marine populations, the eutrophication of shallow lakes, and even spontaneous systemic failures in medicine, such as asthma attacks and epileptic seizures. [9], [26].

A central theme in the literature is the search for generic EWS which arise from the phenomenon of critical slowing down. CSD refers to a systems decreasing rate of recovery from small perturbations as it approaches a bifurcation point. Because CSD is a universal property of dynamical systems, it produces model-independent signals like increased variance and autocorrelation [26]. Guttal and Jayprakash demonstrated that changing skewness in time series data also serves as a warning of a regime shift [27]. They later expanded the theory into the spatial domain, showing that an increase in spatial variance and skewness often precedes a collapse in spatially extended systems [25]. Beyond simple statistical moments, more complex metrics have been proposed to handle system noise. Mayer et al. advocate for the use of Fisher Information as a way to collapse multiple variables into a single index of system stability [29].

Recent advancements have moved the field toward large scale operational monitor-

ing. Significant progress has been made in using satellite remote sensing to detect resilience loss in global climate "tipping elements," such as ice sheets and the Amazon rain forest, across vast spatial and temporal scales [12]. Moreover, comprehensive reviews of tipping point detection across climate, ecological, and human systems demonstrate that these generic EWS have successfully anticipated transitions in a significant majority of reported case studies, confirming their practical use in identifying approaching instabilities [9].

4

Methodology

This chapter describes the methodology applied to address the research questions of this study. The methodology consists of two main components. The first involves the identification and synthesis of tipping point characteristics through a systematic literature review. The second involves the application of these findings to empirical data in order to evaluate their relevance in real world SE contexts. Figure 4.1 shows the overall research process.

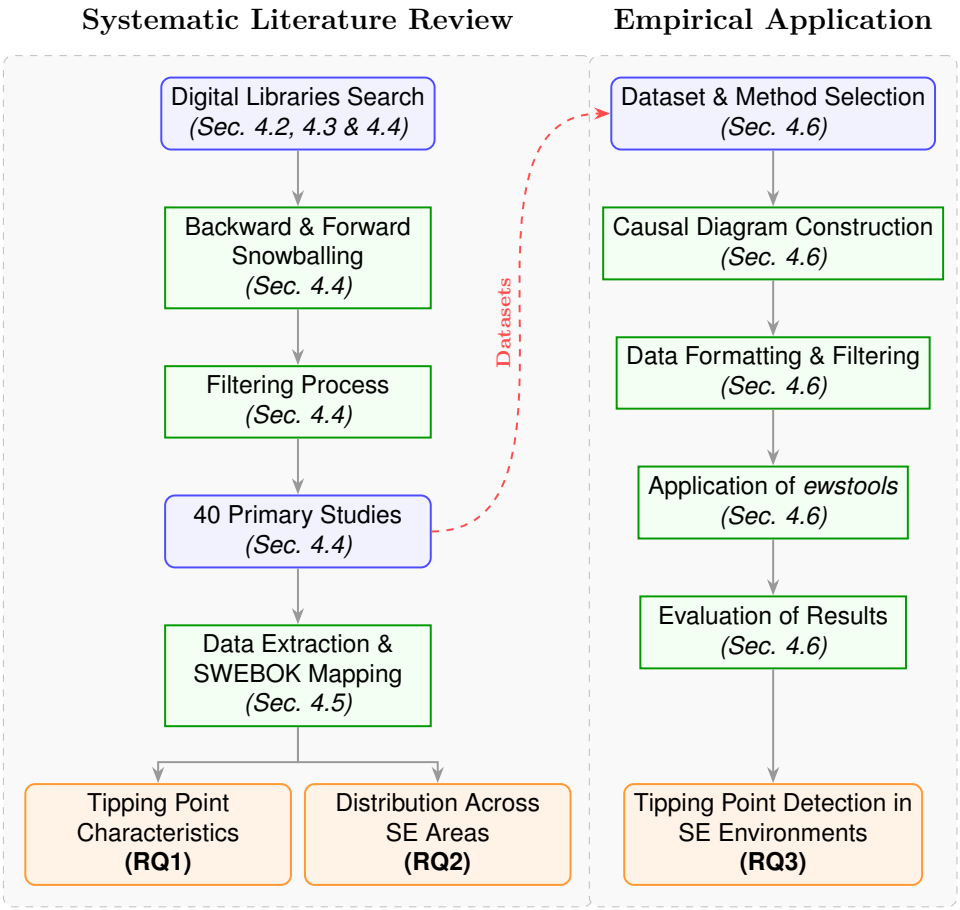

Figure 4.1: Overview of the research methodology

4.1 Systematic Literature Review

The Systematic Literature Review (SLR) was done by following the guidelines made by Kitchenham and Charters [28]. To answer the research questions, both qualitative and quantitative data were needed. Therefore, a mixed-method approach was thought to be the most appropriate method to gain insight into the current available research.The quantitative analysis was used to gather statistical patterns and distributions needed to answer the research questions. The qualitative approach was employed to interpret the statistics and to find connections between established tipping point theory from other domains to SE research. The qualitative synthesis was essential in the process of thematic mapping of the identified EWS to their SE context. The SLR focused on identifying relevant literature for RQ1 and RQ2. For RQ3, a smaller targeted literature search was conducted to find suitable methods and analytical tools.

4.2 Inclusion and exclusion criteria

To ensure that only relevant and high quality studies were included in the review, explicit inclusion and exclusion criteria were defined prior to the screening process. These criteria were applied consistently during the title, abstract, and full text screening phases.

We used the following *inclusion criteria*:

1. Research regarding negative development in a software project leading to a sudden change (e.g., sudden or cascading failures of test, unstable architecture, etc.).

2. Research regarding positive development in a software project leading to a sudden change (e.g., sudden popularity).

We used the following *exclusion criteria*:

1. Non english literature;

2. Tipping point research outside of the SE domain;

3. Secondary studies;

4. Literature with three or less pages;

5. Gray literature.

The limitation of papers with three or less pages is due to those papers typically representing extended abstracts or summaries, not sufficient enough to be used as material in this study. The reason for excluding gray literature is to ensure reliability and avoid bias, since non peer reviewed sources can lack quality control. Secondary

studies are also excluded since the objective of the study is to analyze first-hand data rather than interpretations of other literature reviews. They are also excluded to avoid duplicated data, which can occur when the literature reviews refer to the same studies as the ones found in the SLR.

4.3 Data sources

To find the most relevant literature available to answer the RQs, multiple digital libraries had to be explored. Ultimately, the sources which were used were:

- ACM Digital library (dl.acm.org)
- Scopus (scopus.com)
- Springer (link.springer.com)
- IEEE Xplore (ieeexplore.ieee.org)
- Web of Science (webofscience.com).

This study aims to be based on the most recent and advanced research regarding points within the domain of SE, therefore all of the reviewed literature is at earliest published in 2015.

4.4 Data collection

The data collection process was conducted in accordance with established SLR guidelines by Kitchenham et al. [28], following a structured and replicable workflow consisting of search, deduplication and multi level screening.

To collect the necessary research in accordance with the procedure of a SLR, one string query were made based on RQ1 and RQ2. Since the overarching theme of the first two RQs is the same, one single search query was used across the different research libraries. The third RQ is different since literature outside of SE the field is needed. Therefore, as mentioned previously, it will have its own literature search separate to the SLR.

The query for RQ1 and RQ2 was based on keywords which are derived from the RQs themselves. Because the term “tipping points” is currently not widely used within the research field of SE, substitute terms were applied as well, to find appropriate material. The substitute terms which were added to the query were “critical transition*”, “phase transition*”, “collapse” and “failure*”. Since the expression “tipping points” is common in other domains, the query had to consist of a block representing the correct domain as well. The resulting query corresponding to RQ1 and RQ2 can be seen in figure 4.2.

The search results obtained from the selected digital libraries were exported to a

```
("tipping point*" OR "critical transition*" OR "phase transition*" OR
"collapse" OR "failure*") AND ("project health" OR "project failure*"
OR "project decline") AND ("software" OR "software engineering" OR
"software project*" OR "open source" OR "OSS")
```

Figure 4.2: The query used on the various data sources for RQ1 & RQ2

spreadsheet for further processing. The initial query for RQ1 and RQ2 yielded 820 results. After deduplication based on DOI, 661 unique studies remained.

A three stage screening process was then applied. Title screening excluded 457 studies, of which 32 were secondary studies. Abstract screening further reduced the dataset by 108 papers, leaving 96 studies for full text assessment. Following the full text screening, 13 studies were deemed relevant.

To complement the database search, backward and forward snowballing was conducted on the included studies. Forward snowballing was conducted by systematically reviewing the "Cited by" feature for each paper within its respective digital library. This resulted in 7 additional papers being identified and included.

The number of identified studies was considered insufficient for a comprehensive analysis. To address this, the search strategy was extended by broadening the publication time frame from 2021–2026 to 2015–2026, resulting in 797 additional records, of which 664 remained after de-duplication.

The same screening procedure was applied. Title screening excluded 501 studies, while abstract screening reduced the set to 52 papers. Full text assessment resulted in 7 additional relevant studies. Subsequent snowballing identified 3 further papers.

In total, the extended search contributed 20 additional studies. Combined with the initial results, the final dataset consisted of 40 primary studies covering the period 2015–2026. The complete study selection process is illustrated in Figure 4.3.

Each research paper was also classified into SE knowledge areas based on what was covered in the paper themselves. In Figure 4.4, this distribution is displayed in a pie chart. Notice that most papers were classified as covering multiple knowledge areas and therefore the total amount is a lot greater than 40.

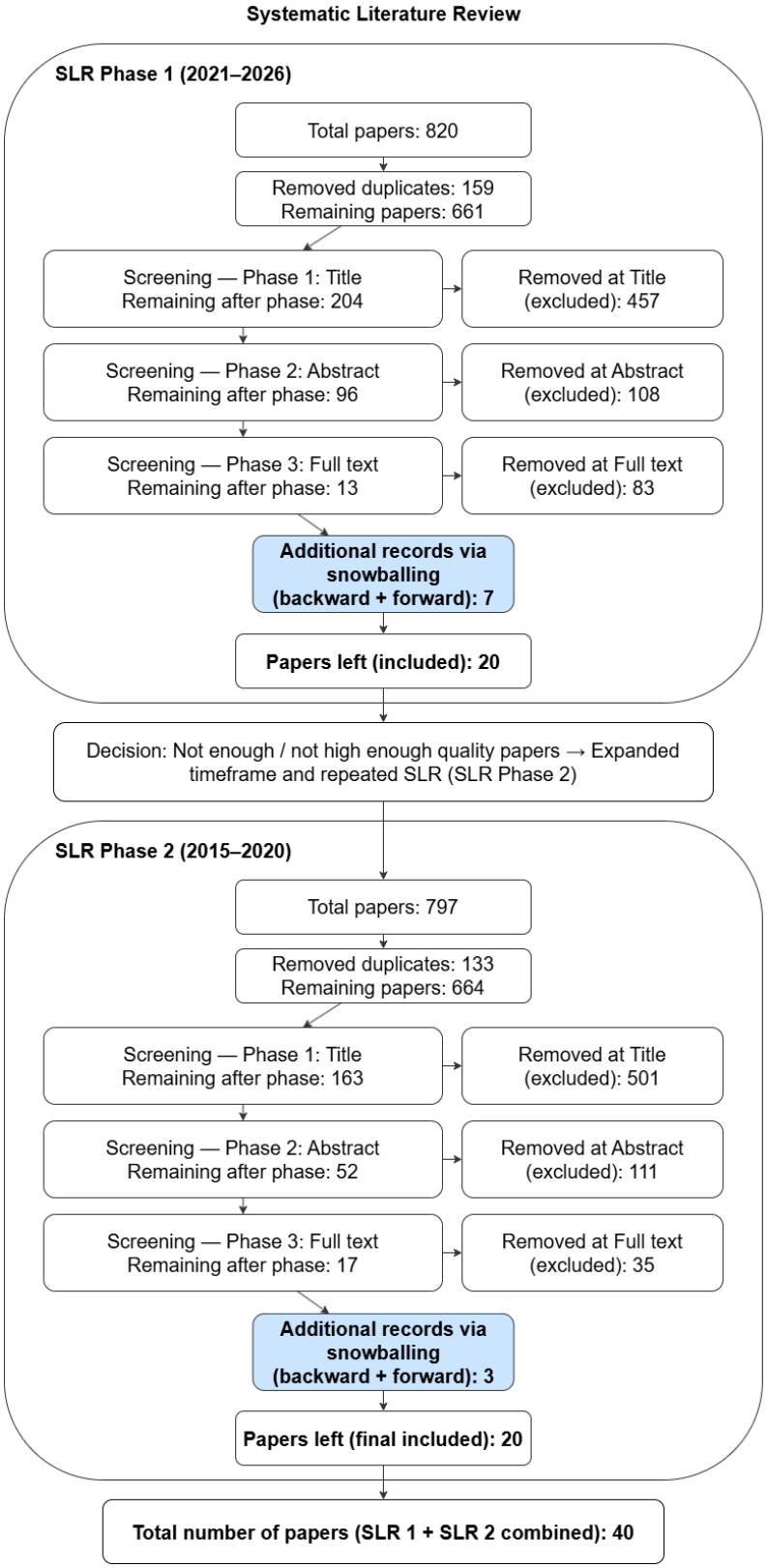

Figure 4.3: Systematic literature review process

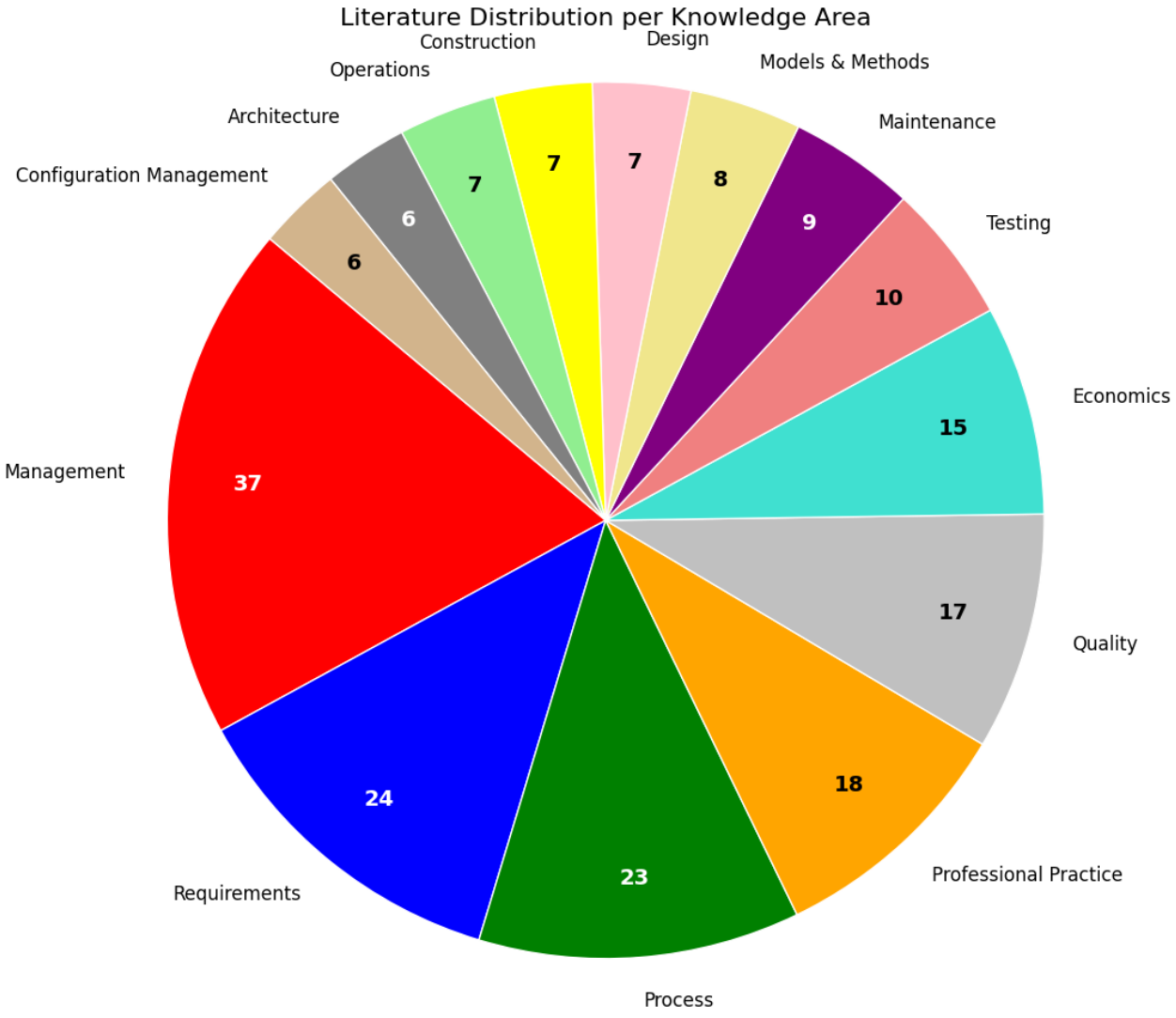

Figure 4.4: Piechart of literature divided into SWEBOK Knowledge Areas

4.5 Data Synthesis

This section describes the synthesis of the data extracted from the primary studies identified from the SLR process.

RQ1 - Characterization and Identification of EWS

The thematic coding strategy for RQ1 was made possible by applying a content analysis on to the papers found during the SLR [5]. This method was particularly appropriate for RQ1 because it allowed for the interpretation of the research sources to identify patterns that align with tipping point theory.

To start off, established tipping point theory was reviewed. This was done in order to initiate a theoretical framework. The goal was to identify what could characterize a tipping point. This stage resulted in the identification of seven EWS. These EWS were chosen based on a research paper by Dakos et al. [9]. This research explores five different domains, namely climate, ecology, health, physical sciences and social sciences. They find a total of 65 EWS across all the five domains. Out of these 65 EWS, only 21 of them occurred more than once. These 21 EWS represent the primary focus of the original study and served as the data pool from which we selected the most relevant EWS for our study. To do that, a filtering process was applied in order to select the most cross-domain and frequently occurring indicators.

Firstly we reclassified the findings, merging together some similar types of EWS. The exact reclassification of the EWS can be seen in the appendix section B.1. This reclassification made the analysis easier to follow. It ensured that overlapping mathematical definitions did not confuse the main system behaviors we were looking at. Secondly, a minimum amount of appearances of at least five times was set, which left nine EWS. This threshold was set to ensure that the EWS selected were backed by a larger amount of empirical evidence. By excluding the less appearing EWS, we could increase the reliability and the robustness of the data. Lastly, we wanted to ensure a multi-domain application of the EWS. Therefore, out of the last nine EWS, we chose the seven which was found in more than one domain. This was done since those are theoretically more likely to be applicable to another domain as well. The seven EWS used for the data synthesis of RQ1 can be seen in Table 4.1 together with their mathematical definitions.

Table 4.1: EWS used in the data synthesis of RQ1

EWS	**Mathematical Definition**
Variance	$\mathrm{Var}(X) = \mathbb{E}[(X-\mu)^2]$
Autocorrelation (Lag-1)	$\rho_1 = \frac{\mathrm{Cov}(X_t, X_{t-1})}{\sigma^2}$
Return rate	$\lambda = -\ln(\alpha)$, where α is the AR(1) coefficient
Power Spectrum	$S(f) = \int_{-\infty}^{\infty} R(\tau) e^{-i2\pi f\tau}\, d\tau$
Skewness	$\mathrm{Skew}(X) = \mathbb{E}\left[\left(\frac{X-\mu}{\sigma}\right)^3\right]$
Kurtosis	$\mathrm{Kurt}(X) = \mathbb{E}\left[\left(\frac{X-\mu}{\sigma}\right)^4\right]$
Fisher Information	$I(\theta) = \mathbb{E}\left[\left(\frac{\partial}{\partial\theta}\ln f(X;\theta)\right)^2\right]$

Each tipping point characteristic found in the SE domain during our analysis was also labeled as a tipping type. There are a total of four tipping types. These are called macro tipping, propagating tipping, clustered tipping and societal impact tipping [12]. This let us characterize the potential impacts of the SE tipping points as well. This establishes a characterization framework which both identifies the EWS of a tipping point and also their structural implications.

Based on the content analysis, a set of requirements was created. These requirements were used to classify each tipping point contribution. The requirements can be found in Table 4.2 and in Table 4.3. For a tipping point contribution to be classified as a specific EWS or tipping type, it has to follow the behaviors described. The requirements exist to ensure a systematic and repeatable mapping process.

Table 4.2: Characterization requirements for SE tipping point contributions regarding EWS

Early Warning Sign	Requirement
Variance	Must show signs of increased fluctuation intensity, such as frequent changes, sudden deviating patterns, and inconsistent spikes in activity.
Autocorrelation	Must resemble a Critical Slowing Down (CSD) system, where the contribution shows signs of increasingly being "stuck" in a state without improvement.
Return Rate	Must show a decline in system resilience, specifically through signs that the system fails to return to a healthy state after experiencing disturbances.
Power Spectrum	Must show a loss of regularity, where periodic patterns that were previously regular occurrences become irregular or non-periodic.
Skewness	Must show clear indications of asymmetry in the system's distribution, with data heavily leaning toward a specific direction as it approaches a transition.
Kurtosis	Must show events of significant outliers that deviate far from the historical mean of a stable system, indicating "heavy-tailed" behavior.
Fisher Information	Must show signs of systemic collapse through the loss of order and structural integrity within the project's organization or architecture.

Table 4.3: Characterization requirements for SE tipping point contributions regarding tipping types

Tipping Type	Requirement
Macro Tipping	Must show signs of a fundamental large-scale shift in the systems state. It is characterized by a single transition that significantly changes the overall project status.
Propagating Tipping	Must document a transition in one specific area that triggers a subsequent shift in a different part of the system.
Clustered Tipping	Must show how multiple independent disturbances occurring in close proximity pushes the system toward a critical threshold.
Societal Impact	Must show socio-technical impacts which leads to a significant shift in the human/social side of the project.

RQ2 - Localization of EWS within SE Knowledge Areas

A content analysis was also applied to answer RQ2. While RQ1 found the characteristics of EWS leading to potential tipping points, RQ2 answers where they appear within the domain of SE. To answer that question, a last step was added each time a tipping point characteristic was found. That step was to identify in which SWEBOK [10] knowledge area the characteristic would fit in. The SWEBOK definitions was the theoretical framework on which we based the mapping of where each characteristic appeared in the SE domain. This allowed us to use real established standardized SE terms. This also made sure that we had a structured approach which could be applied consistently and repeatably across all of the literature derived from the SLR.

A total of 15 knowledge areas from the SWEBOK was used. The three unused areas, namely “Computing foundations”, “Mathematical foundations” and “Engineering foundations” were excluded. These areas were excluded because they consist of fundamental principles and logic within the domain rather than the actual development areas where tipping points can be found. The knowledge areas used in the data synthesis and their requirements for categorization can be seen in Table 4.4.

Table 4.4: SWEBOK knowledge areas used for SE tipping point contribution localization

Knowledge Area	**Requirement**
Software Requirements	Needs to explicitly and mainly affect SE requirements and stakeholder needs.
Software Architecture	Needs to affect architectural desgins and patterns.
Software Design	Needs to affect component logic, internal system interfaces, or design abstractions.
Software Construction	Needs to affect the implementation of source code, coding standards, or debugging processes.
Software Testing	Needs to affect verification strategies, test coverage, or validation procedures.
Software Operations	Needs to affect system deployment, runtime performance, or production environment.
Software Maintenance	Needs to affect software evolution and changes.
Configuration Management	Needs to affect version control, baseline integrity, or build management.
SE Management	Needs to affect project planning, resource allocation, or risk mitigation strategies.
SE Process	Needs to affect development lifecycles, workflow efficiencies, or methodology adoption.
SE Models and Methods	Needs to affect modeling abstractions, formal specifications, or methodological frameworks.
SE Quality	Needs to affect quality assurance standards or process metrics.
SE Security	Needs to affect vulnerability management, data protection, or system resilience.
Professional Practice	Needs to affect team ethics, professional conduct, or organizational communication.
SE Economics	Needs to affect cost estimation or business value analysis.

4.6 RQ3 - Empirical Application on Open-Source Datasets

The final stage of the methodology was to find the right assets needed to complete RQ3. RQ1 and RQ2 were grounded in a SLR to establish a theoretical framework. However, to answer RQ3, needs an empirical approach. The methodology of RQ3 was divided in to three phases. To start off the process, relevant SE related datasets had to be found. Secondly, it was necessary to identify tools and methods which could be applied on SE datasets. After that, the last step was to actually apply the tools and methods to the datasets, and to evaluate the performance.

SE Datasets

The outcome of the SLR resulted in a total of 40 papers. From these 40 distinct papers, a total of five different datasets were derived. These five datasets contained data in different parts of SE, both strict technical data but also socio-techincal data. The following datasets were identified and marked as potential candidates for the empirical analysis.

- Core Developer Turnover in the Rust Package Ecosystem: Prevalence, Impact, and Awareness [S2]
- An Empirical Study on the Survival Rate of GitHub Projects [S5]
- Cascading Effects: Analyzing Project Failure Impact in the Maven Central Ecosystem [S3]
- How Do Communities of ML-Enabled Systems Smell? A Cross-Sectional Study on the Prevalence of Community Smells [S1]
- On the abandonment and survival of open source projects: An empirical investigation [S7]

Method Identification

A targeted search was conducted to identify established tools and methods which could be applied to the identified datasets. The search focused on software packages capable of detecting tipping points and critical transitions. The tools and methods also had to be compatible with the data available.

The search was conducted on various search engines but also by analyzing the previously identified research about tipping point detection. The following software tools were identified and marked as potential candidates for the empirical analysis.

- **earlywarnings (R package):** This is one of the most widely used libraries for detecting EWS of critical transitions. It provides functions for rolling window analysis of autocorrelation, variance, skewness, and kurtosis. It was developed

by Dakos et al [21].

- **spatialwarnings (R package):** This tool focuses on spatial patterns of transitions. While primarily used in ecology, it was considered for its ability to analyze structural integrity and Fisher Information equivalents in non-temporal datasets. It was Developed by Génin et al [18].

- **ewstools (Python package):** A modern Python-based implementation designed for the analysis of tipping points in time-series data. This tool is particularly useful for its integration with Python's data science ecosystem. It was Developed by Thomas Bury [11].

Selection Process

The third and final step in the process of answering RQ3 was to choose which datasets to use and which method to apply to them. The method was identified first, followed by the selection of datasets that met its requirements and format. Among the options presented in the previous subsection, the Python package *ewstools* was selected as the primary tool for this part of the study.

The choice of *ewstools* was based on a multitude of reasons. Firstly, it is the most modern of the identified tools. It is designed with the Python data science ecosystem in mind, integrating industry-standard libraries such as *NumPy* and *Pandas*. It provides a "comprehensive, object-oriented framework for computing EWS across a given time series" [11]. It also offers robust detrending methods, like LOWESS, alongside a wide array of CSD metrics. These include statistical metrics like variance, autocorrelation, skewness, and kurtosis, as well as more advanced measures involving power spectra and various entropy computations [11]. Furthermore, the package is supported by extensive documentation, which facilitated a more reliable implementation of these methods.

The choice was also driven by the team's experience in Python. Utilizing a familiar programming environment was essential for efficiency, allowing resources to be focused on analysis rather than the acquisition of new software competencies within the study's constraints.

After the decision was made to use the Python package *ewstools*, the datasets were analyzed. The goal of this analysis was to determine which of the datasets would best complement the tool. Since *ewstools* is a framework for computing EWS for a given time series, time-series data were required. After an initial screening of the datasets and the available metrics, two were excluded. The excluded datasets were "How Do Communities of ML-Enabled Systems Smell? A Cross-Sectional Study on the Prevalence of Community Smells" [S1] and "Cascading Effects: Analyzing Project Failure Impact in the Maven Central Ecosystem" [S3]. The main reason for their exclusion was that their data could not be formatted into a time series, making it impossible to use the tool. The other three datasets had a more fitting format, with metrics that could be manipulated in such a way that they would work well

with the Python package.

Causal Diagrams

In order to correctly identify and predict a tipping point based on the variables from the datasets, causal diagrams were constructed using Directed Acyclic Graphs. These diagrams provide a qualitative graphical representation of the system, showcasing how various variables potentially influence the outcome. We utilize these models primarily to make contextual assumptions and imply causal relationships. The goal is not to perform a finalized causal discovery, but rather to support the discovery of the operational context from a causal perspective.

Before constructing the diagrams, it was necessary to formalize assumptions based on the dataset and previous research from RQ1 and RQ2. These assumptions are referred to as *Prior Knowledge* and the process follows the framework introduced in Heyn et al. (2026) [3]. The following example uses the dataset from "Core Developer Turnover in the Rust Package Ecosystem: Prevalence, Impact, and Awareness" [S2] to explain the process.

After the identification of PKs, which can be seen in Table 4.5, a causal model was iteratively constructed. The model grew as each newly identified causal mechanism is integrated. This process ensures that the relationship between the outcome and the other variables is captured systematically.

The outcome of the diagram is the Core Developer Turnover Event (CDTE), which signifies the critical tipping point for the project (PK1). Since the dataset is limited and we can only base the diagram on what is available, latency was chosen as one of the primary exposures of which indicate CDTE (PK4). To capture the context fully, competing causes such as workload, driven by downloads, issues and the number of contributors is also added (PK2). The number of core developers as well as the knowledge concentration is also two variables which can influence the likelihood of a developer leaving (PK3).

Figure 4.5 illustrates the iterations of the causal diagram construction. Nodes are color coded by variable type: blue indicates the outcome, green the exposure, and red, gray, and white represent observable, unobservable, and adjusted variables, respectively. The iterative construction of the causal diagram is illustrated in Figure 4.5. Phase (A) involved identifying the initial observable variables and the primary outcome. In (B), causal relationships were hypothesized by connecting these variables based on domain assumptions. Latent proxy variables KC and WL were introduced in (C) to provide structural coherence to the DAG. A causal connection from NOC to commits was also added in this step. Followed by the full integration of the outcome variable in (D). In the final stage (E), Latency was defined as the primary exposure. Since Latency is derived from PRs, the latter is identified as a key ancestor in the causal chain. To neutralize biasing paths, the DAGitty tool [22] was employed, identifying that NOCD and Commits required adjustment to accurately measure the effect of Latency on a CDTE. All variables listed above can be

found alongside their definitions in Table 4.6.

Table 4.5: Prior Knowledge for the Causal Diagram of Rust Developer Turnover

ID	Prior Knowledge
PK1	Core developer turnover serves as a primary driver for the system reaching a critical tipping point.
PK2	High volumes of issues and downloads increase the maintenance workload, which may influence turnover rates.
PK3	Knowledge concentration is directly affected by the size of the core developer team and the number of contributors.
PK4	PR latency (processing time) acts as an observable proxy variable of a project prior to a transition event.

Table 4.6: Variables for the Rust Developer Turnover Causal Diagram

Variable	Definition
DTOE	Developer Turn Over Event
Latency	Time it takes to process a pull request
Knowledge Concentration (KC)	Knowledge of the core developer compared to the other
Downloads (DLs)	The amount of downloads between a specific time point
Issues (OI)	The amount of open issues
Workload (WL)	The amount of work available
NOCD	The number of core developers for the project
NOC	The number of contributors to the project
Commits	The number of commits
Pull Requests (PRs)	The number of opened pull requests

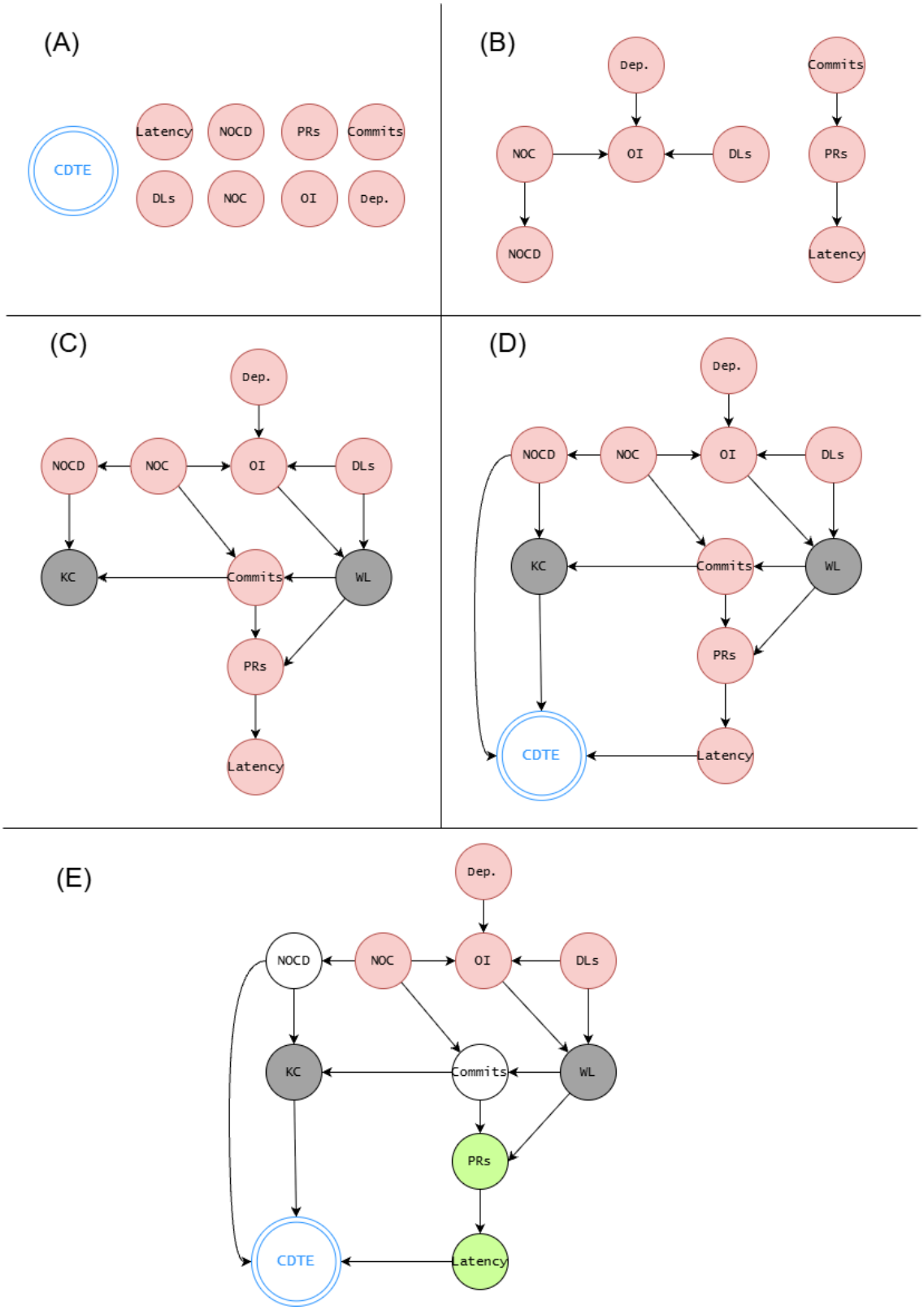

Figure 4.5: Causal diagram and subsequent iterations derived from the "Core Developer Turnover" dataset [S2]

Application

The empirical application of the methodology was conducted on the three datasets that met the time-series requirements: the Rust Package Ecosystem dataset [S2], the GitHub Survival dataset [S5], and the Open Source Abandonment dataset [S7]. For each dataset, a three-step analytical pipeline was followed. Firstly, a dataset-specific causal diagram was constructed, as explained in a previous section. Secondly, all the data were transformed through filtering and formatting. The last step was the application of the *ewstools* package to detect EWS.

The first application targeted the Rust dataset. Projects were selected based on a two-year observation window occurring after 2021. This boundary was established to minimize the socio-technical anomalies and implications caused by the initial COVID-19 outbreak. To ensure sufficient data for a time-series analysis, projects were required to meet minimum activity thresholds within this window: at least 100 commits, 50 merged pull requests, and 500 total downloads. From the resulting pool, a representative sample of five "solo" projects (a single core developer) and five "team" projects (multiple core developers) was selected for analysis.

Two primary metrics were extracted as indicators, commit volume and latency. Latency was defined as the duration in days between a pull request's creation and its successful merge. These metrics were resampled into weekly intervals to create a uniform time series. Following the *ewstools* analysis, interactive dashboards were generated to visualize transition dynamics and evaluate the predictive power of the causal variables identified in the DAG.

The second application analyzed a survival dataset across the Laravel, npm, R, and WordPress ecosystems to compare the EWS of "Surviving" versus "Non-surviving" projects. The primary objective was to compare the EWS of "Surviving" projects against "Non-surviving" projects. To ensure statistical validity, only projects with a minimum lifespan of seven months were considered, providing sufficient data points for meaningful time-series analysis. The time-series extraction was limited to a maximum of two years directly preceding the tipping point event. By capping the historical data at two years, the analysis isolates the critical window where project dynamics actually begin to destabilize, ensuring the metrics capture the transition rather than a project's entire lifecycle. Notably, the construction constraints of this specific dataset made it impossible to filter out the COVID-19 pandemic timeframe.

Raw event data was cleaned by filtering out bot-generated activity and then resampled into weekly intervals. This was a simple process since the dataset had already tagged bot activity. An additional filter was applied to exclude "zombie" projects, requiring each time series to contain at least ten weeks of confirmed human activity. To move beyond individual project observations, a comparative statistical analysis was performed. The Kendall Tau trends for both groups were compared using the Mann-Whitney U test to determine if indicators of a tipping point were significantly more pronounced in non-surviving projects. The distribution of these trends was visualized using violin plots to highlight the statistical difference between the two

populations.

The final application was conducted on the dataset regarding open-source abandonment [S7]. This phase aimed to identify whether commit-based EWS could distinguish projects approaching a transition toward abandonment from those maintaining stability. Only projects with a minimum lifespan of seven months and a total observation length of at least 30 weeks were included. All data was captured before Covid-19, therefore there was no need to add a filter for that. To isolate transition dynamics, only commits occurring prior to the identified tipping point date were considered. Similar to the survival analysis, a population level statistical comparison was conducted using the Mann-Whitney U test to evaluate the significance of the differences between surviving and non-surviving groups. The resulting trends were also visualized via violin plots.

All time-series data were detrended using the LOWESS method. This approach was selected because SE metrics are frequently non-linear and characterized by sudden, high-intensity bursts of activity. LOWESS constructs a baseline by analyzing local data segments, which prevents isolated spikes from disproportionately shifting the trend line or creating false indicators of instability.

A small sensitivity analysis was conducted on the rolling window size parameter as well as the LOWESS span parameter. This was done in order to ensure validity in the procedure. Sizes of 0.125, 0.25 and 0.375 of the total time-series length were tested for the window size. While a LOWESS span of 0.2, 0.3 and 0.4 was also tested. The analysis revealed that the EWS indicators remained stable across these variations, suggesting that the results are not overly sensitive to the window and span selection within reasonable bounds for SE contexts. The final results will be presented with the window size of 0.25 and a span of 0.2, due to them being the standard values in the package *ewstools*.

The Mann-Whitney U test was chosen because it is specifically designed for non-normally distributed data. Technical SE metrics rarely follow a normal distribution, as they frequently exhibit bursts of activity followed by prolonged periods of inactivity. A standard significance threshold of $p \leq 0.05$ was applied, aligning with established empirical research practices. Furthermore, violin plots were selected to visualize the distributions. This direct visual representation helps validate the significance of the tests by illustrating the actual shape and spread of the underlying data.

By leveraging the specific computational capabilities of the *ewstools* library, four primary EWS indicators were analyzed: variance, autocorrelation, skewness, and kurtosis. These four metrics were chosen as they provide a comprehensive statistical profile of a system's stability. While variance and autocorrelation measure the "sluggishness" and fluctuations associated with critical slowing down, skewness and kurtosis capture the increasing asymmetry and extremeness of activity bursts as a project approaches a tipping point.

5

Results

This chapter presents the results of the study in relation to the research questions. The findings from the SLR and the empirical analysis are presented through the identified tipping point characteristics, SE activities, and the application of existing EWS detection tools the chosen datasets.

5.1 Characteristics of Tipping Points (RQ1)

To address RQ1, the identified tipping point instances from the selected primary studies were analyzed and mapped to a set of seven EWS, as described in Section 4.5. Figure 5.1 presents the distribution of these EWS across the analyzed studies. The results show that variance is the most frequently observed early warning sign, followed by autocorrelation and skewness.

This indicates that increasing instability and fluctuations in system behavior are commonly reported characteristics of tipping points in the existing SE literature. In particular, an increase in variance indicates that the system becomes more unpredictable prior to a transition, which aligns with established tipping point theory in other domains. This pattern is visible across multiple SE activities, suggesting that rising variability is a general characteristic of systems approaching critical transitions.

Autocorrelation is also prominently represented, indicating that systems approaching a tipping point tend to exhibit slower recovery from perturbations. This behavior, commonly referred to as critical slowing down, suggests that deviations persist longer over time, making the system more vulnerable to further disturbances. In a SE context, this may reflect situations where issues accumulate faster than they can be resolved, such as growing technical debt or unresolved defects.

Skewness further contributes to the characterization of tipping points, reflecting asymmetries in the distribution of system states. This may indicate that extreme events or outliers become more likely as the system approaches a critical transition. In software projects, this could correspond to sudden spikes in failures, delays, or unexpected changes in system behavior.

While these three indicators dominate, other EWS such as Fisher information, return

rate, and kurtosis appear less frequently. This indicates that they are less commonly reported in the analyzed literature. One possible explanation is that measures such as Fisher information and kurtosis are less familiar to SE researchers and therefore have received less attention in the existing literature. The power spectrum is the least represented indicator, indicating limited empirical evidence of frequency based early warning behavior in the analyzed literature.

An important observation is that different EWS appear across different SE activities, rather than being confined to a single context. This indicates that tipping points are not characterized by one universal signal, but rather by a combination of indicators that together reflect a loss of system stability. The variation in EWS across activities suggests that tipping point behavior may manifest differently depending on the context, even if the underlying dynamics are similar.

Overall, the results indicate that tipping points in SE are primarily characterized by increasing variance, autocorrelation, and skewness in system behavior. These findings support the notion that EWS identified in other complex systems are also observable in SE.

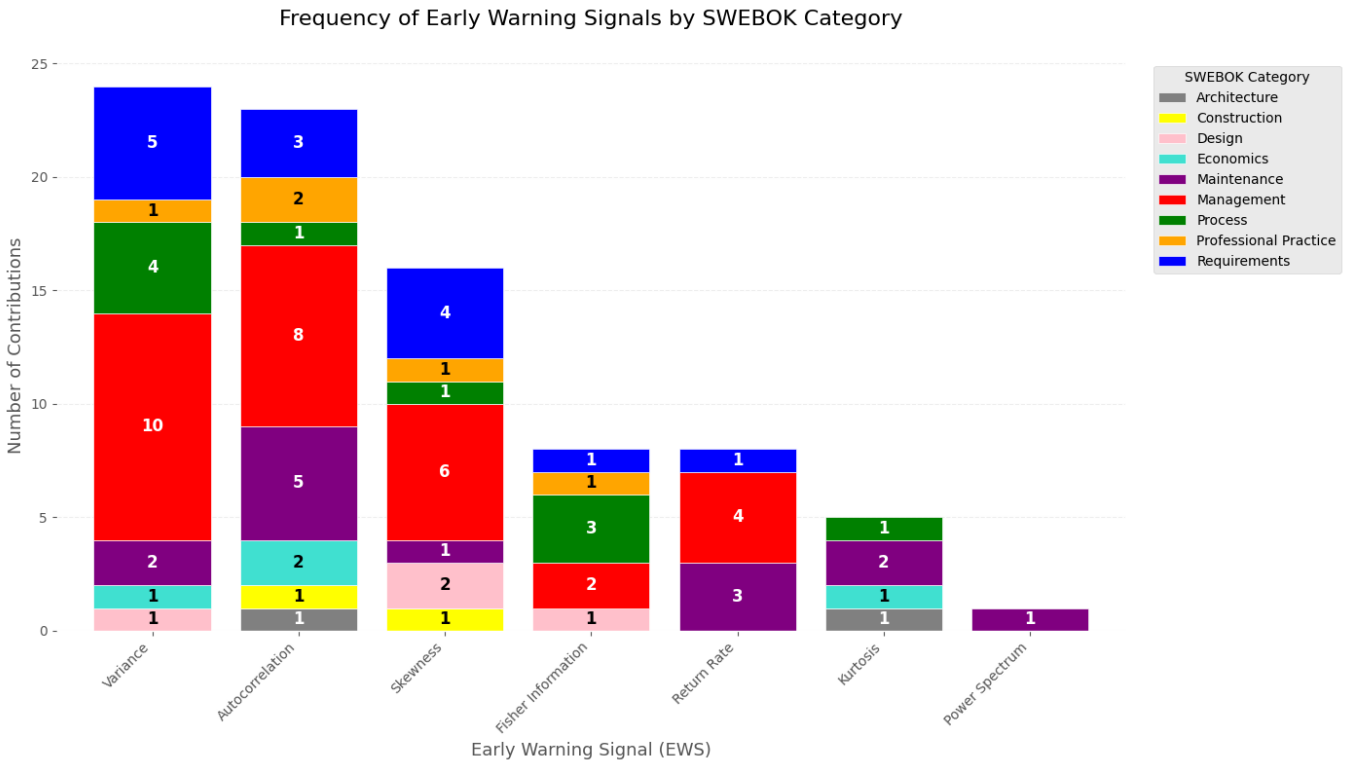

Figure 5.1: Bar graph showing relationships between SWEBOK categories, established EWS, and the amount of tipping contributors found in the SE domain.

The identified characteristics are also reflected in individual observations from the analyzed studies. Several papers explicitly describe increasing instability in software projects, where delayed handling of issues leads to escalating consequences. For example, one study notes how defect related problems "significantly increase the cost

and development time" [S4, p. 1], illustrating how variability and unpredictability increase as the system evolves.

This behavior is closely tied to the timing of detection and response. In many cases, issues are only addressed once they have already escalated, rendering corrective actions less effective. As noted in prior research, "The main causes behind failed software projects are that it is often too late to correct the problems by the time they are detected" [S14]. This suggests that systems may remain seemingly stable until a critical threshold is crossed, after which recovery becomes exceedingly difficult.

Similarly, the persistence and accumulation of issues is evident in descriptions of project failure dynamics, where failure is not attributed to a single cause, but rather to "many failure factors working together" [20, p. 2]. This reflects a gradual reduction in system resilience, where interacting factors collectively push the system toward a state in which it can no longer absorb disturbances.

In addition, propagation effects are explicitly described in software ecosystems, where "Even smaller, less connected libraries can trigger significant cascading effects through complex dependency chains, demonstrating ecosystem vulnerability extends beyond just critical components" [S3]. This demonstrates how localized issues can escalate and affect multiple parts of a system, rather than remaining isolated. Such behavior reinforces the interconnected nature of software systems, where dependencies between components allow disturbances to spread and amplify across the system.

This aligns with the view that software projects operate within complex and dynamic environments. As described by Alami, "The e-Borders project failed when its ecosystem became unbalanced; hence survival became difficult. Projects' ecosystems are a reflection of the real world and are characterized by volatility, uncertainty, complexity, and unknowns." [S11], highlighting the conditions under which such transitions can occur.

Notably, the distribution of EWS observed in this study closely aligns with patterns reported in other complex system domains. In particular, the three most prominent EWS in the results, variance, autocorrelation, and skewness, are also the most frequently identified indicators in fields such as climate science, ecology, and health sectors, as reported by Dakos et al. [9], and illustrated in Figure 5.2. This similarity suggests that, despite differences between domains, SE systems may exhibit comparable behavior as they approach critical transitions.

Taken together, these observations strengthen the interpretation of the identified patterns as indicators of tipping point behavior, where increasing variability, slower recovery, and propagation effects signal a system that is approaching a critical transition.

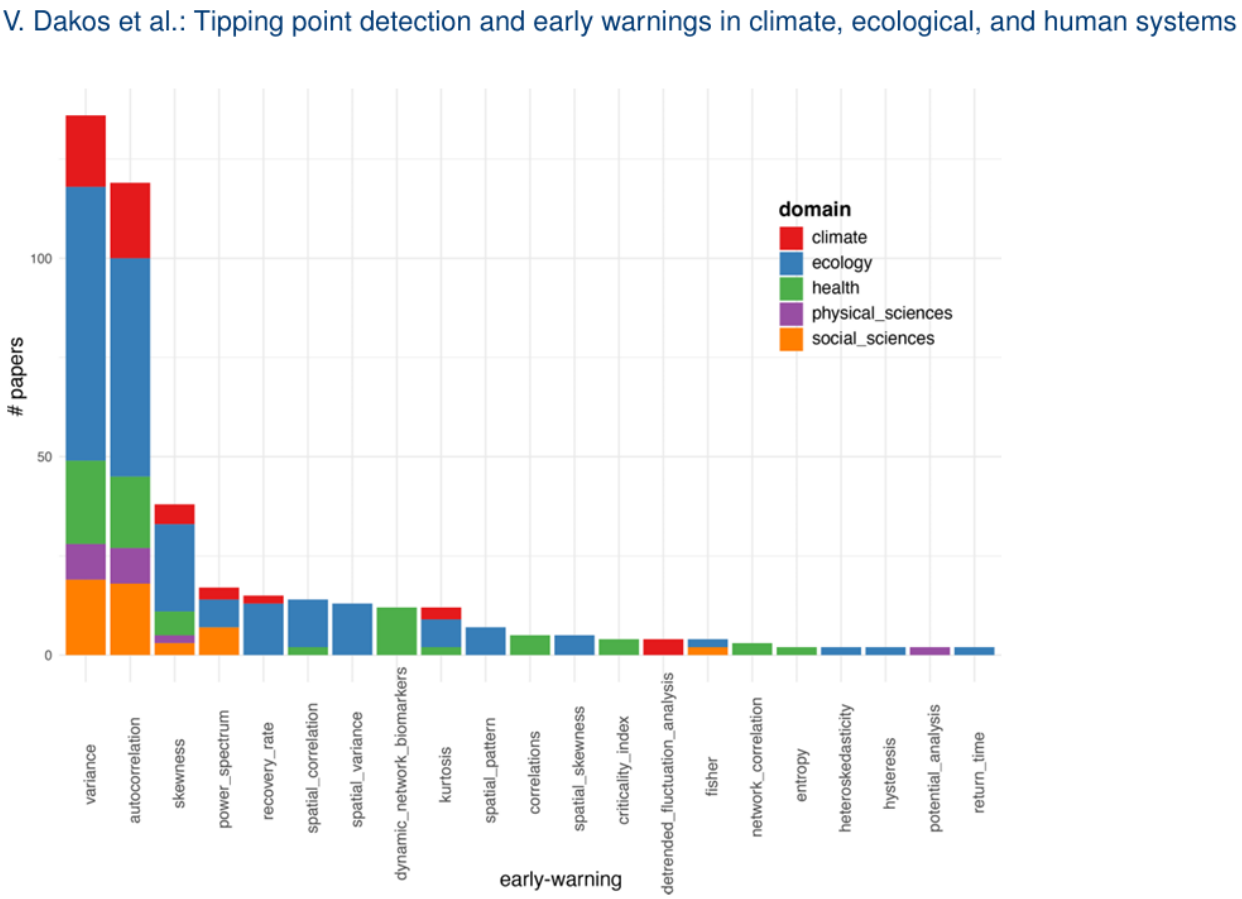

Figure 5.2: Bar graph showing the distribution of EWS across multiple domains, adapted from Dakos et al. [9]

Summary for RQ1

The results of RQ1 demonstrate that tipping points in SE can be characterized using EWS and tipping point behaviors derived from other complex systems domains.

Most Common EWS: Variance, autocorrelation, and skewness were identified as the most frequently occurring EWS across the data. Their prevalence suggests that software systems approaching tipping points often exhibit increasing instability, stronger fluctuations, and growing asymmetry in system behavior.

Alignment with Other Domains: The distribution of EWS observed in SE closely aligns with findings from domains such as ecology, climate science, and finance, where variance and autocorrelation are also among the most common indicators of critical transitions. This suggests that software projects may exhibit similar dynamical behaviors to other complex adaptive systems.

Broad Presence of EWS Types: All seven selected EWS were identified within the SE domain, although with varying frequency. This indicates that tipping point behavior in SE is multifaceted and cannot be explained through a single indicator alone, but rather through combinations of different instability patterns.

5.2 Distribution Across SE Activities (RQ2)

To answer RQ2, each identified tipping point contribution was mapped to a SWEBOK knowledge area based on the affected SE activity. The resulting relationships between SWEBOK categories, EWS, and tipping point types are visualized in the alluvial plot shown in Figure 5.3.

The results show that tipping point characteristics are distributed across a broad range of SE activities rather than being isolated to a single development phase or technical area. Contributions were identified in areas including Requirements, Management, Maintenance, Professional Practice, Process, Design, Architecture, Construction, and Economics. This indicates that tipping point related behavior can emerge in both technical and socio-technical parts of software projects.

Management and Requirements exhibit the strongest presence in the dataset. These areas are connected to a wide range of EWS and tipping types, suggesting that tipping points frequently emerge in parts of the development process involving decision making, coordination, planning, and stakeholder interaction. This indicates that activities which define project direction and structure are particularly vulnerable to instability and critical transitions.

A clear pattern can be observed for Requirements related tipping points, which are predominantly associated with propagating behavior. These contributions are frequently linked to variance, autocorrelation, and skewness, indicating increasing instability and growing asymmetry in project behavior as requirement related issues accumulate. This suggests that instability originating in RE rarely remains localized and instead spreads throughout interconnected parts of the project. Since requirements influence architecture, implementation, testing, and maintenance, even small disruptions may cascade into larger system wide effects. The results therefore indicate that requirements engineering acts as a structurally sensitive area within software projects, where unresolved issues may trigger broader transitions in system stability.

Management related tipping points are strongly associated with societal tipping behavior and are distributed across nearly all identified EWS. In particular, variance and autocorrelation appear frequently in management related contributions, suggesting repeated fluctuations and reduced resilience in organizational processes over time. These findings indicate that many tipping points in SE are rooted in socio-technical dynamics rather than isolated technical failures. Several studies describe instability caused by communication breakdowns, poor coordination, workload imbalance, unclear responsibilities, or shifting priorities. Such factors often interact and reinforce each other, gradually reducing the ability of teams and projects to recover from disturbances.

The alluvial plot also shows that variance and autocorrelation are widely distributed across multiple SWEBOK categories. Unlike more localized indicators such as kurtosis and power spectrum, these two EWS appear consistently across both technical

and organizational areas. This suggests that they may function as more general indicators of instability within SE systems. Similar patterns have previously been observed in domains such as ecology, climate science, and finance, where increasing variance and autocorrelation are associated with systems approaching critical transitions. The results therefore indicate that software projects may exhibit similar dynamical behaviors to other complex adaptive systems.

Maintenance and Process also show notable connections to several EWS and tipping types, although to a lesser extent than Management and Requirements. These areas primarily reflect long term system evolution, accumulated technical complexity, and the continuous adaptation of software over time. Several contributions in these areas describe instability caused by increasing maintenance burden, unresolved technical debt, or gradually deteriorating development processes. Such behavior resembles the loss of resilience associated with CSD, where systems become increasingly unable to recover efficiently from disturbances.

In contrast, Architecture, Design, Construction, and Professional Practice show fewer connections in the alluvial plot. However, this does not necessarily imply that tipping points are less relevant in these areas. Instead, it may indicate that tipping behavior in SE is more commonly discussed through organizational, process oriented, or project level consequences rather than isolated technical transitions. Technical instability may therefore manifest indirectly through downstream effects in management, requirements, or maintenance related activities.

Across all SWEBOK categories, societal tipping points appear most frequently. This reinforces the observation that tipping points in SE are closely tied to human and organizational dynamics. In contrast, clustered tipping points occur less frequently, suggesting that tipping behavior is rarely confined to isolated subsystems and more commonly involves interactions across multiple components or teams. Propagating tipping points are also strongly represented, particularly in Requirements and Process related activities, further supporting the interpretation of software projects as highly interconnected systems where instability can spread between components.

Overall, the results indicate that tipping points in SE primarily emerge in areas characterized by strong interdependence, coordination, and evolving complexity. The findings suggest that tipping behavior in software projects should be understood as a systemic and socio-technical phenomenon rather than solely a technical problem. The broad distribution of EWS across multiple SWEBOK categories further supports the applicability of tipping point theory as a framework for understanding instability and critical transitions in SE systems.

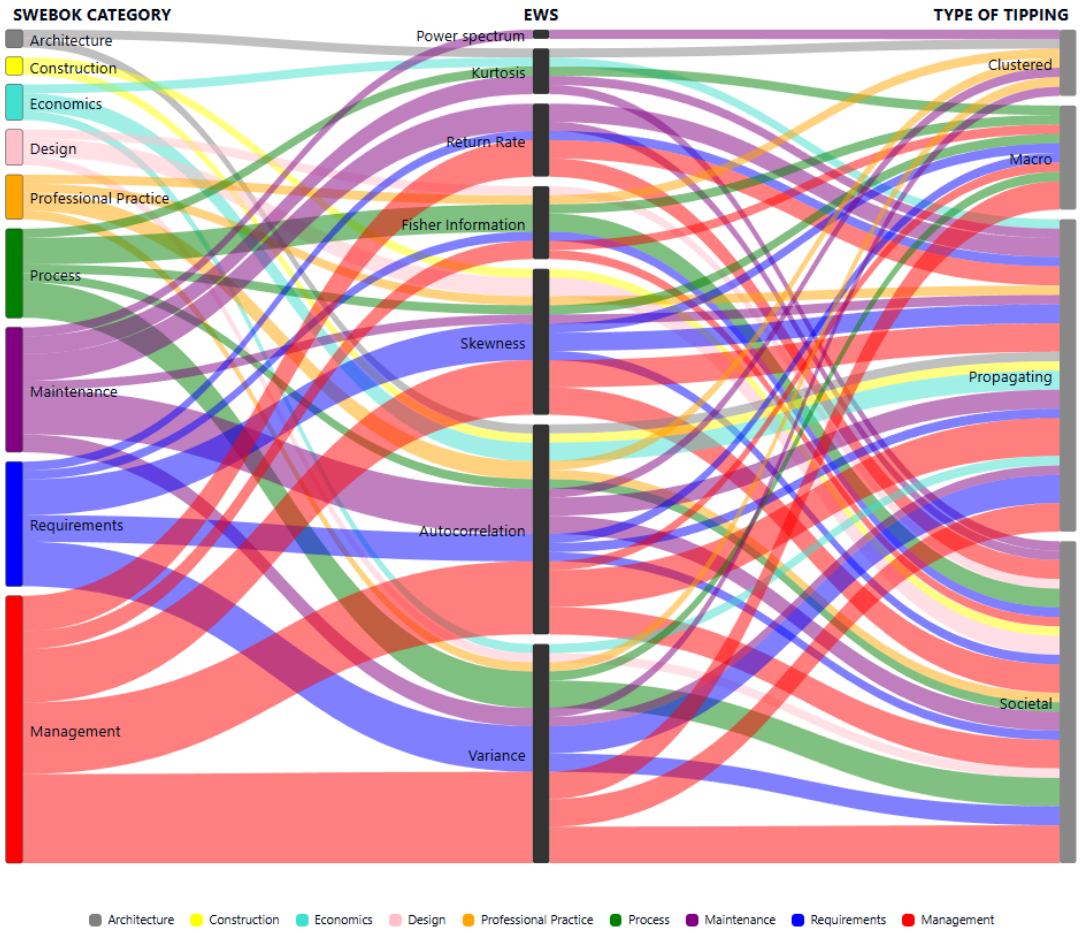

Figure 5.3: Alluvial plot showing relationships between SWEBOK categories, EWS, and tipping point types.

Summary for RQ2

The results of RQ2 demonstrate that tipping points in SE are not limited to isolated technical failures, but instead emerge across both technical and socio-technical activities throughout the software development lifecycle.

Dominance of Management and Requirements: Management and Requirements were identified as the most prominent SWEBOK categories associated with tipping point behavior. Requirements related tipping points were primarily connected to propagating tipping, while Management was strongly associated with societal tipping, highlighting the importance of organizational and coordination related instability.

Most Common Tipping Types: Societal and propagating tipping points appeared most frequently across the dataset. This suggests that tipping behavior in SE commonly spreads across interconnected parts of a project and is often influenced by human, organizational, and communication related factors.

Distribution of EWS: Variance and autocorrelation appeared across a wide range of SWEBOK categories and tipping types. Their broad distribution suggests that these EWS may represent general indicators of instability in SE systems, similarly to their role in other complex systems domains.

5.3 Application of *ewstools* to SE Datasets (RQ3)

To address RQ3, the study implemented an analytical framework using the *ewstools* Python package [11]. This methodology was applied in two distinct ways. Firstly, as an analysis of ten sampled repositories within the Rust turnover dataset. This sample size was selected not due to computational constraints, but to allow for a thorough qualitative inspection of individual repository behaviors. Secondly, the analytical framework was used to conduct a broad population comparison across the remaining two datasets.

To evaluate the effectiveness of the EWS leading up to a project's tipping point, this study utilizes the Kendall Tau (τ) rank correlation coefficient. Kendall Tau is used to indicate the strength of a trend along a time-series [21]. It has been used in previously studied literature regarding tipping point theory and it is what *ewstools* provide [12], [14]. Although there is no universal standard for interpreting the Kendall Tau values, it does provide a clear indication. A positive Tau value, indicates an increasing trend in the metric over time. In the context of this study, a high correlation is the expectation for a system approaching a critical transition. A negative Tau value indicates a decreasing trend in the EWS metric. A negative correlation generally suggest the absence of the traditional EWS sign. Negative and weak Tau values can also be the result of oscillating trends and they should be further investigated upon before interpreting the results of the analysis [21].

Core Developer Turnover in the Rust Package Ecosystem

The causal diagram for this dataset is given in Figure 5.4. The process of creating it is explained in the methodology chapter. The resulting diagram tells us that in order to measure the effect Latency has on the CDTE, we also have to measure the effect of commits and NOCD. For this reason, we decided to apply the *ewstools* computational methods on the latency and commit metrics. It was not possible to compute it on the NOCD metric since it was a static value. Therefore, it was also not possible to turn into a time series.

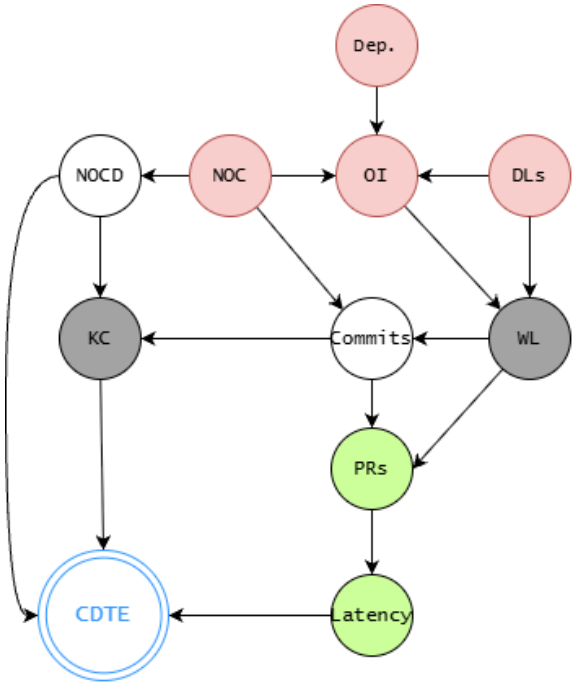

Figure 5.4: Causal diagram of the Rust Turnover dataset. Abbreviations used: OI (Open Issues), NOC (Number of Contributors), NOCD (Number of Core Developers), WL (Workload), DLs (Downloads), PRs (Pull Requests), and Dep. (Dependencies). Node colors indicate variable classifications: green (exposure), blue (outcome), red (observable), gray (unobservable), and white (adjusted).

Ten repositories were randomly sampled from the dataset to serve as observable case studies. By applying *ewstools* to metrics from the datasets, the analysis identifies if the tool is applicable to SE datasets. Each projects latency and commit metrics were analyzed using *ewstools*. All the resulting graphs are available in the Appendix at Section B.2.1 and Section B.2.2.

Table 5.1 displays the ten projects and their respective number of NOCD, commits and PRs during the measured time period. As can be seen, the amount of data differs for each projects. Because some projects have a much higher volume of commits than PRs, the most effective metric for predicting tipping points can vary. Tables 5.2 and 5.3 detail the EWS variance, autocorrelation, skewness, and kurtosis computed via *ewstools* from all the sampled projects.

Rather than exhaustively reviewing every generated graph one by one, specific individual projects are selected for detailed inspection. RP1 is chosen to highlight the differences between EWS derived from commits versus PR latency, and RP7 is examined to demonstrate how bursty data can cause variance and autocorrelation to diverge.

Table 5.1: Rust Repository Metrics During the Captured Window

ID	Name	NOCD	Commits	PRs
RP1	calypso	1	186	241
RP2	cli-batteries	1	133	75
RP3	whitelist	1	385	53
RP4	farcaster-core	1	597	255
RP5	linear-algebra-rs	1	233	58
RP6	sol-did	3	694	217
RP7	macro	2	309	393
RP8	sn_launch_tool	4	111	1082
RP9	tools	3	571	895
RP10	akd_core	3	182	56

Table 5.2: Kendall Tau (τ) Values by Rust Project (Latency)

ID	Project	Type	Var_τ	$AC1_\tau$	$Skew_\tau$	$Kurtosis_\tau$
RP1	calypso	Solo	0.8414	0.3621	-0.3509	-0.3411
RP2	cli-batteries	Solo	-0.4974	-0.2692	-0.5040	0.3548
RP3	whitelist	Solo	-0.1138	0.2667	-0.2095	-0.0585
RP4	farcaster-core	Solo	0.0933	-0.4537	0.1726	0.2822
RP5	linear-algebra-rs	Solo	0.3779	-0.0883	0.3457	-0.3445
RP6	sol-did	Team	0.3593	-0.0816	0.1489	0.1602
RP7	macro	Team	0.3736	0.0068	-0.2632	-0.2288
RP8	sn_launch_tool	Team	0.0006	0.0848	-0.2316	0.1184
RP9	tools	Team	0.0551	-0.2247	0.0741	0.2880
RP10	akd	Team	-0.4842	0.0827	-0.3146	-0.2702

A comparison of project RP1 across both tables highlights significant discrepancies between the two approaches. Under latency (Table 5.2), EWS variance and autocorrelation show strong and moderate positive correlations, respectively, while skewness and kurtosis are moderately negative. However, under commits (Table 5.3), the relationship inverts. Skewness and kurtosis become positive, whereas variance and autocorrelation turn negative. The latency based graphs of RP1 can be seen in Figure 5.5. In this project, the solo developer noticeably increases the latency of the PRs a couple of months before they leave, resulting in a high Tau value for variance. By looking at the same project commits based graph, Figure 5.6, we can see that the developer stops committing to the project around six months prior to the tipping point event. This time skewness and kurtosis are the Tau values which indicates of a tipping point.

Figure 5.5: EWS for project RP1. The indicators (autocorrelation, variance, kurtosis, and skewness) were estimated within a 0.25 rolling window. For the first graph, "state" represents the raw weekly commit volume, while "smoothing" shows the trajectory after applying a LOWESS de-trending filter.

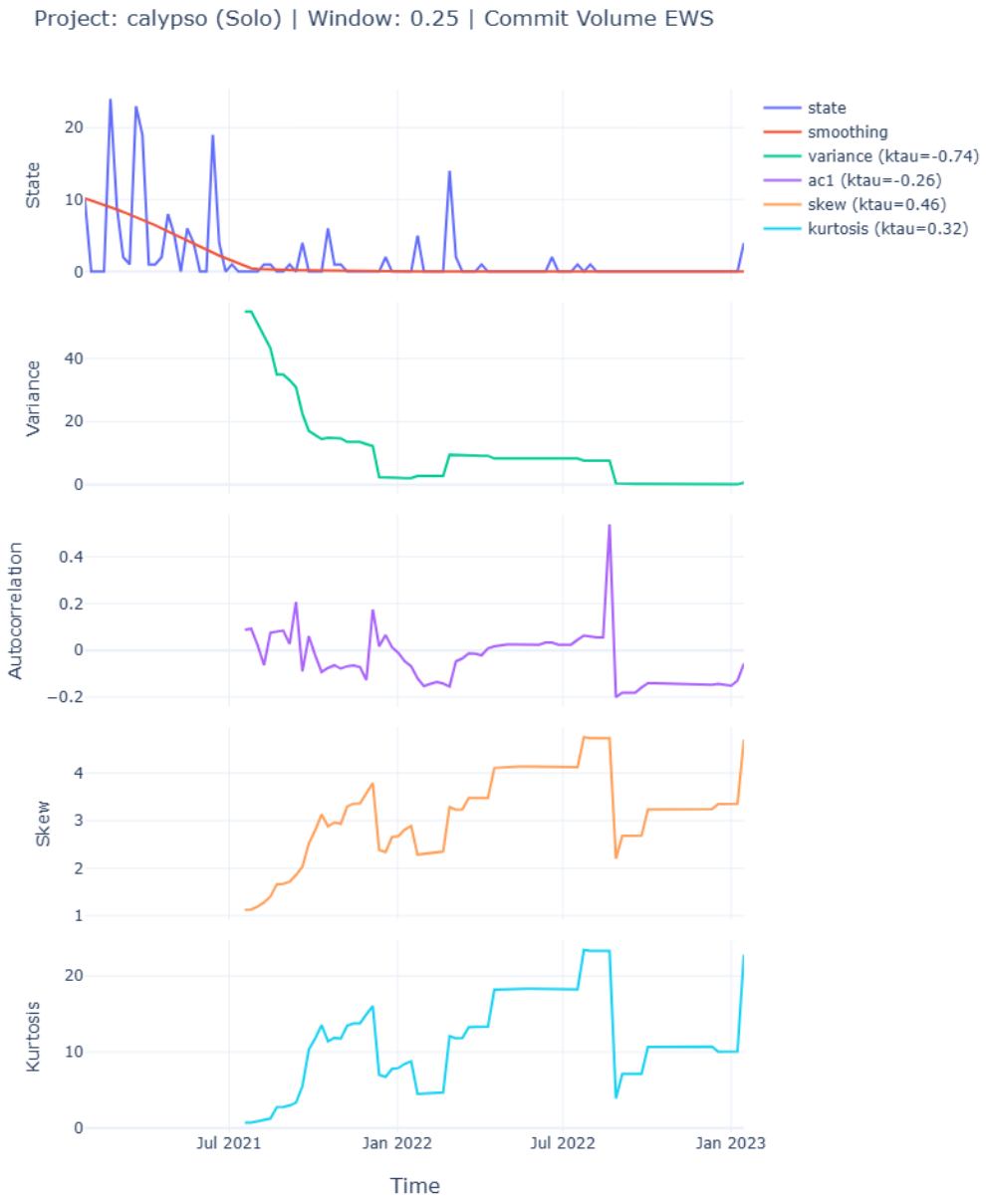

Figure 5.6: EWS for project RP1. The indicators (autocorrelation, variance, kurtosis, and skewness) were estimated within a 0.25 rolling window. For the first graph, "state" represents the raw weekly commit volume, while "smoothing" shows the trajectory after applying a LOWESS detrending filter.

Table 5.3: Kendall Tau (τ) Values by Rust Project (Commits)

ID	Project	Type	Var_τ	$AC1_\tau$	$Skew_\tau$	$Kurtosis_\tau$
RP1	calypso	Solo	-0.7423	-0.2574	0.4580	0.3216
RP2	cli-batteries	Solo	-0.7728	-0.2558	0.2576	-0.1341
RP3	whitelist	Solo	0.1217	0.4392	0.2381	-0.1587
RP4	farcaster-core	Solo	-0.4898	-0.2891	0.3895	0.3425
RP5	linear-algebra-rs	Solo	-0.4635	0.1715	-0.2994	-0.2909
RP6	sol-did	Team	0.5108	0.0797	-0.0532	-0.0361
RP7	macro	Team	0.6646	-0.6386	0.3203	0.2880
RP8	sn_launch_tool	Team	-0.5291	0.2867	-0.1975	-0.1867
RP9	tools	Team	0.0924	0.4411	-0.1038	0.0880
RP10	akd	Team	-0.3683	-0.3521	0.0742	0.1901

In Table 5.3, RP7 exhibits a strong positive correlation for variance, but also a strong negative correlation for autocorrelation. In most EWS theory regarding critical slowing down, both variance and autocorrelation are generally expected to move together and increase as a system approaches a transition. The graphs for the EWS RP7 can be seen in Figure 5.7. Right before the tipping point is reached, the amount of weekly commits quickly spikes, which increases the variance. The autocorrelation slowly decreases throughout the measurement window, which is why the Tau value is strongly negative. Because of the raw commit data oscillating through out the weeks, which can be seen in the state line of the first graph. Both skewness and kurtosis spikes at the same time, and by inspecting the state line of the first graph, a big spike occurs each time. This happens since skewness and kurtosis are both sensitive to outliers. For RP7, the variance Kendall Tau value correctly identifies the increasing likelihood of a tipping point occurring, while the kurtosis and skewness shows a small indication. The autocorrelation test however fails, due to it being sensitive to the flickering of the commit data.

When comparing projects with a single core developer against projects with multiple core developers, some things differ in the commit-based EWS. The solo projects seem to have a high negative variance. This is due to the total amount of commits often getting closer to zero as the tipping point approaches. While for teams, there still are other developers who may commit. What also has a big difference is the skewness value for the commits. There is a noticeable difference in how the skewness indicator behaves between project types. For solo projects, the Kendall Tau value for skewness is never very strong, but four projects exhibits a low to medium positive trend. For team projects however, the skewness Tau values sits close to zero, with three projects showing a negative correlation with skewness and the tipping point. The presence of an increasing trend for skewness in solo projects indicates that, although their average activity is dropping as the tipping point approaches, they are still experiencing occasional bursts of activity that deviate sharply from their declining average. Consequently, rising skewness appears to be a significantly more reliable early warning indicator for solo developer projects than the other EWS.

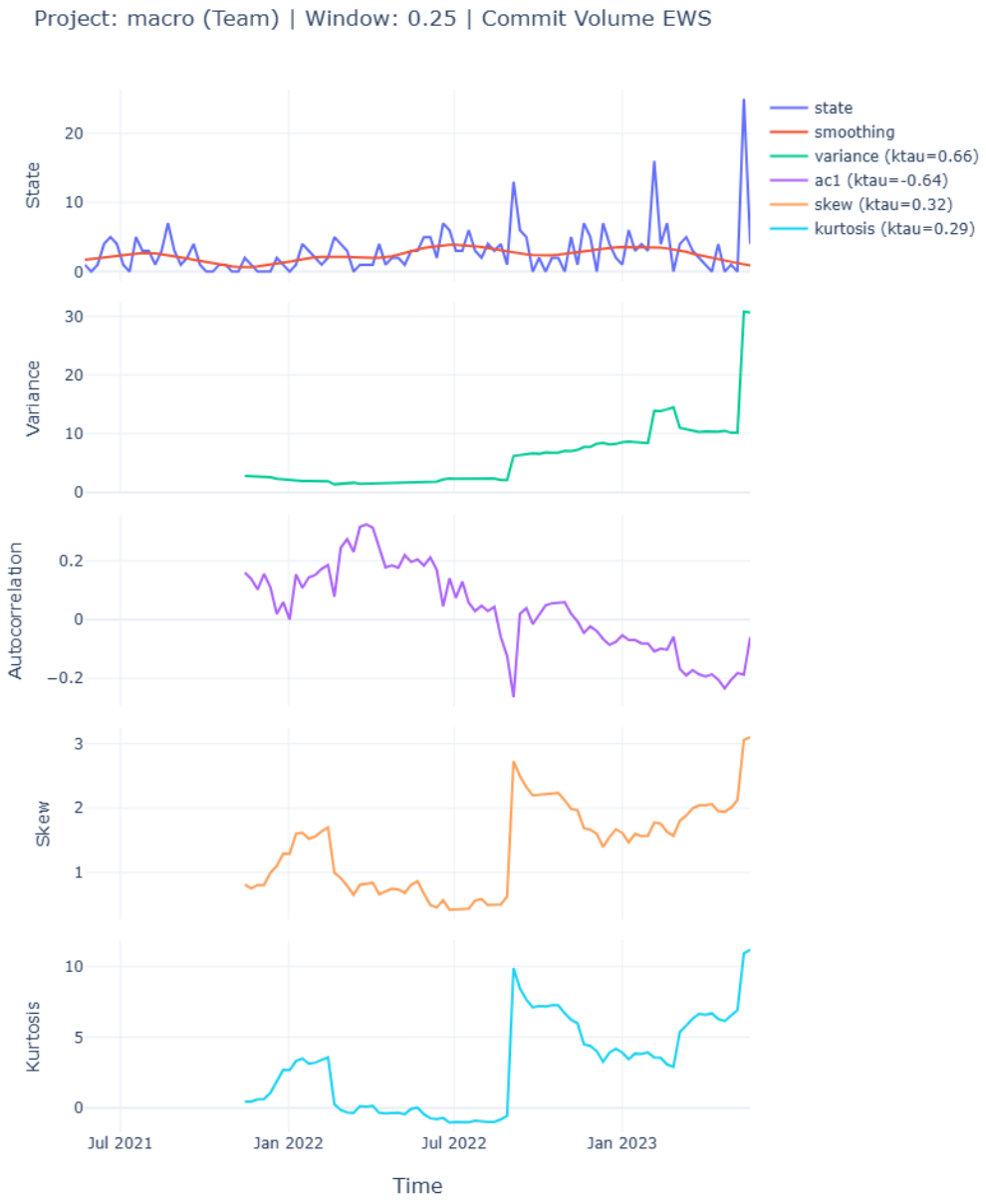

Figure 5.7: EWS for project RP7. The indicators (autocorrelation, variance, kurtosis, and skewness) were estimated within a 0.25 rolling window. For the first graph, "state" represents the raw weekly commit volume, while "smoothing" shows the trajectory after applying a LOWESS detrending filter.

Survival Rate of GitHub Projects

The second application of the *ewstools* package focused on projects from four software ecosystems. Those are Laravel, npm, R, and WordPress. This analysis was conducted on eight randomly sampled repositories, with one "surviving" and one "non-surviving" project from each ecosystem. In addition to analyzing these individual samples, a population-level comparison was performed to evaluate the broader differences between surviving and non-surviving projects following the tipping point.

The derived causal diagram for this dataset can be found in Figure 5.8. The observable data is divided into code and non-code types. Code activities consist of pull requests and commits, while non-code activities include open issues, reviews, and comments. Together, these metrics form the overall repository activity. The tipping point derived from this dataset is defined as the survival or non-survival of the project. The observable exposure in this case is the combination of code and non-code activities. Using this combined metric, the *ewstools* package was then employed to calculate Kendall Tau trends across the structured time series. All of the resulting individual project graphs can be found in the Appendix Section B.2.3.

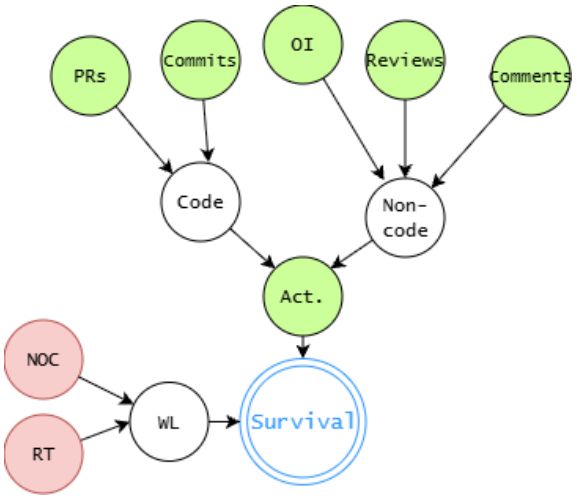

Figure 5.8: Causal Diagram of the Survival Survival Rate of GitHub Projects Dataset. Abbreviations used: PRs (Pull Requests), OI (Open Issues), Act. (Activities), NOC (Number of Contributors), RT (Repository Type), and WL (Workload). Node colors indicate variable classifications: green (exposure), blue (outcome), red (observable), gray (unobservable), and white (adjusted)

In Table 5.4, all calculated EWS for the eight sampled projects are collected. The total amount of non-coding and coding activities can also be seen there. By inspecting the table, we can see that the Tau values do not follow a certain pattern which is coherent for all projects and their status.

Table 5.4: Kendall Tau (τ) Values of Sampled Projects

ID	Project	Eco	Status	Activities	Var_τ	$AC1_\tau$	$Skew_\tau$	$Kurt_\tau$
SP1	simple-scroll-to-top-button	WP	Alive	91	0.310	0.012	-0.156	-0.136
SP2	npm-boilerplate	npm	Alive	88	-0.418	0.109	-0.492	-0.554
SP3	tufte	R	Alive	124	-0.335	-0.219	-0.059	-0.067
SP4	laravel-viva-payments	Laravel	Alive	82	-0.690	0.130	0.571	0.583
SP5	automate-mautic	WP	Dead	84	0.375	0.476	-0.198	-0.189
SP6	eslint-config-fluct	npm	Dead	100	-0.760	-0.683	-0.522	-0.522
SP7	travis	R	Dead	840	-0.184	0.303	-0.560	-0.481
SP8	laravel-tracer	Laravel	Dead	96	-0.954	0.011	0.416	0.370

After inspecting the produced graphs, a commonality can be found for most of the projects. When inspecting their weekly activities, a lot of empty weeks can be found, with bursts of activities in others. To illustrate this widespread issue of data sparsity, project SP5 was specifically selected for detailed inspection. As can be seen in Figure 5.9, this issue heavily affects the calculations and resulting Tau values. The data sparsity is a reoccurring phenomenon in open-source development, but it has large implications for the mathematical reliability of calculations. For example, the applied LOWESS filter can not work as intended when applied to sparse discrete activity, and the sudden bursts will be interpreted as extreme outliers.

The next step of the analytical process of this dataset was to compare the two groups of surviving and non-surviving projects. The paper which describes the dataset defines a non-surviving project as a project without any activity the last six months of measuring it [S5]. In Table 5.5 the means of all the projects of the dataset can be seen. The p-value column is based on a Mann-Whitney U test. The only significant difference according to the test is in the variance. Surviving projects seem to have a correlation closer to zero than its non-surviving counterpart. Due to most projects probably lacking consistent data, these results might not be reliable enough to make any significant claims. Figure 5.10 illustrates the differences as violin plots which are very similar to each other. This further suggests that the difference between surviving and non-surviving projects does not seem to have much of a difference in this dataset.

Table 5.5: Statistical Comparison of Kendall Tau (τ) Trends Between Surviving and Non-surviving Projects

Indicator	Non-surviving (Mean)	Surviving (Mean)	p-value	Sig.
Var_τ	**-0.2493**	**-0.1617**	**0.0220**	**Yes**
$AC1_\tau$	-0.0551	-0.0758	0.5232	No
$Skew_\tau$	0.0529	0.0162	0.2097	No
$Kurt_\tau$	0.0865	0.0016	0.1792	No

Yes = $p < 0.05$ using Mann-Whitney U test

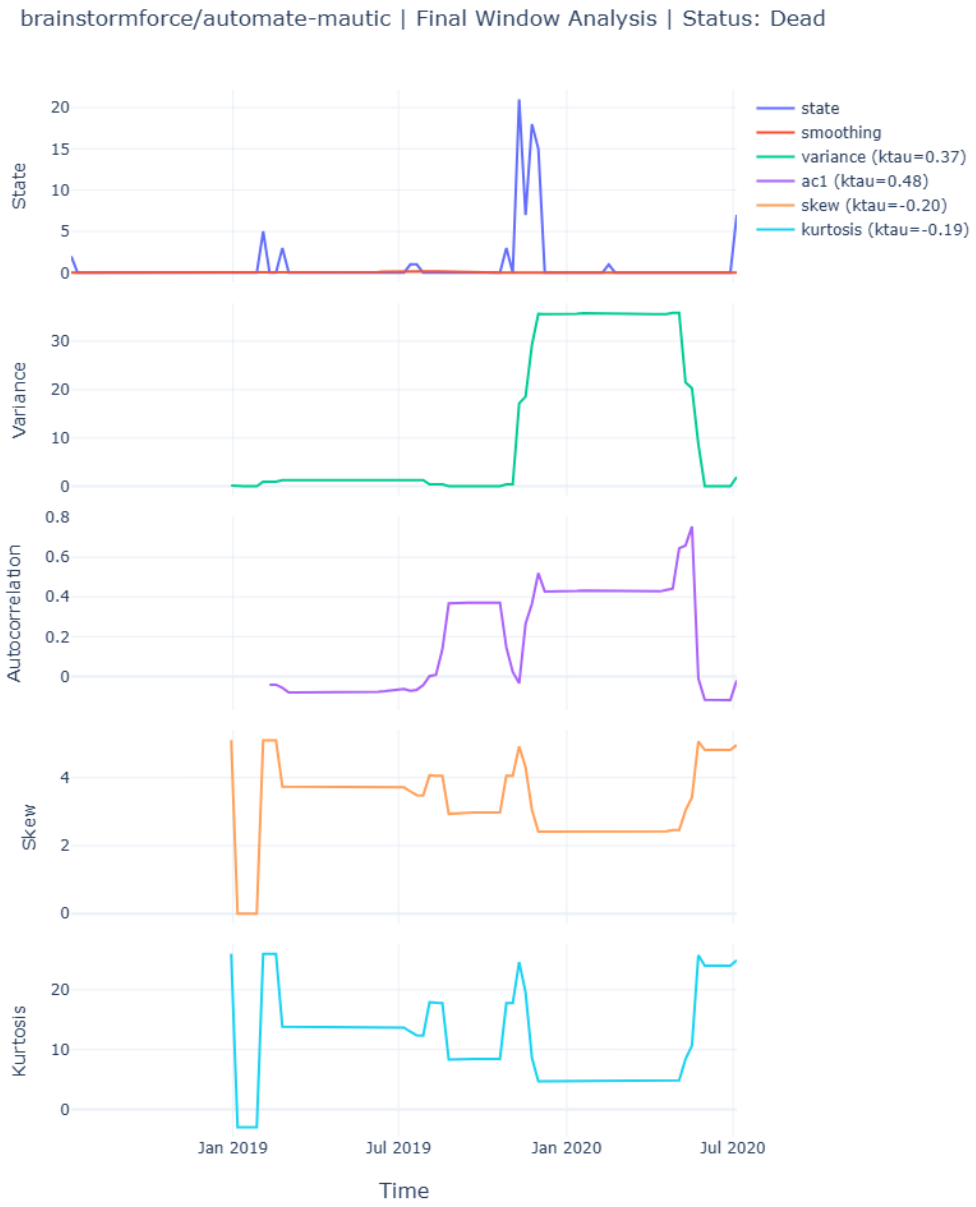

Figure 5.9: EWS for project SP5

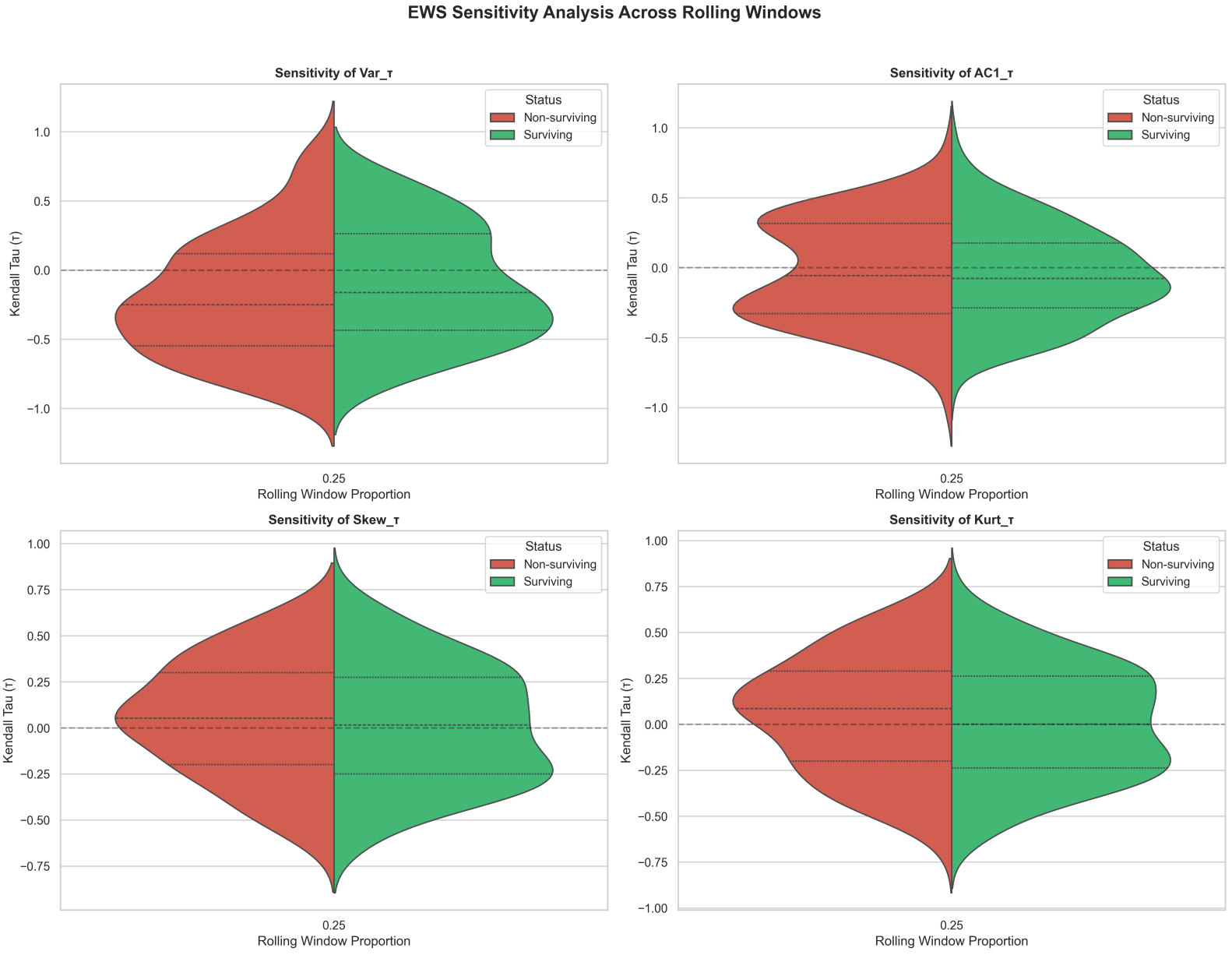

Figure 5.10: Violin Plots showcasing the distribution difference between Surviving and Non-Surviving Projects

On the Abandonment and Survival of Open Source Projects

The third application of the *ewstools* functionalities was conducted on a dataset focusing on the abandonment and survival of open-source software. As outlined in the causal diagram in Figure 5.11, the observable exposures for this dataset are the project's commit volume and the NOCD. The analyzed tipping point is the truck factor developer detachment (TFDD) event. This concept originates from the paper from which the dataset is derived, and it refers to the event of all core developers having left the project [S7]. Similar to the previous dataset, we firstly evaluated ten randomly sampled repositories, followed by a population-level statistical comparison utilizing the Mann-Whitney U test. The two groups evaluated against each other are projects which were abandoned after the TFDD event, and projects which did not get abandoned. The projects that were not abandoned survived by subsequently attracting new developers. All of the resulting individual project graphs can be found in the Appendix Section B.2.4.

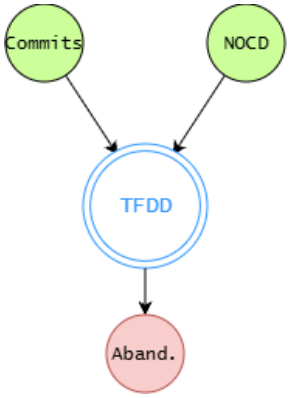

Figure 5.11: Causal diagram On the Abandonment and Survival of Open Source Projects: NOCD (Number of Core Developers) and Aband. (Abandonment). Node colors indicate variable classifications: green (exposure), blue (outcome), red (observable), gray (unobservable), and white (adjusted).

Table 5.6 lists the calculated EWS values for the ten sampled projects in this dataset. By inspecting the table, we can see that similar to the survival dataset of the previous analysis, the Tau values vary heavily. There is no uniform pattern that clearly separates the abandoned projects from the ones that survived.

When looking closer at the individual projects, it becomes clear that abandonment does not happen the exact same way every time. The EWS metrics show two very different types of collapse. A more volatile flickering phase and a slower fade. To show these two different behaviors, AP5 and AP4 were specifically selected to be inspected in detail.

The abandoned project AP5 exhibits an example of a system approaching a critical transition, with positive Tau values across 3 out of four indicators. This suggests that the commit activity became both more repetitive and more erratic right before abandonment. By looking at Figure 5.12 we can observe this dynamic in detail. The overall variance fails to capture a consistent trend due to the sudden jump and

Table 5.6: Kendall Tau (τ) Values of Sampled Projects

ID	Project	Status	Commits	Var_τ	$AC1_\tau$	$Skew_\tau$	$Kurt_\tau$
AP1	former	Abandoned	319	-0.144	0.227	0.242	0.172
AP2	dalli	Abandoned	268	0.003	0.415	0.594	0.580
AP3	savon	Abandoned	704	0.518	0.257	-0.688	-0.692
AP4	picturefill	Abandoned	704	-0.642	0.063	-0.497	-0.268
AP5	sinatra	Abandoned	394	0.222	-0.142	0.476	0.467
AP6	js-xlsx	Not Abandoned	116	-0.562	-0.547	0.477	0.586
AP7	capistrano	Not Abandoned	528	0.482	0.095	-0.077	-0.285
AP8	minimagick	Not Abandoned	121	-0.554	0.608	-0.219	-0.222
AP9	shoulda-matchers	Not Abandoned	330	-0.158	0.126	-0.413	-0.352
AP10	commander.js	Not Abandoned	106	-0.318	-0.008	0.177	0.138

subsequent drop from the commit data. Both skewness and kurtosis begin a steep steady climb starting in mid 2013, months before the actual commit spike occurs. This steady rise indicates that the project's activity distribution was developing fat tails, indicating that extreme outliers were becoming more frequent compared to the decreasing baseline activity. The dashboard illustrates a strong flickering signature, where skewness and kurtosis rise sharply months prior to a final burst of commit activity in early 2014, preceding the project's eventual abandonment.

On the other hand, the abandoned project AP4 exhibits the opposite behavior. It has negative values for variance, skewness, and kurtosis, indicating a quiet stagnation of activity rather than a volatile collapse. As shown in Figure 5.13, there is significantly less flickering occurring in the data. While there is an initial high spike in commits, it is followed by a relatively consistent low activity pattern for the remainder of the timeline.

The next step of the analytical process for this dataset was to compare the two groups at population level to see if these EWS metrics could serve as universal predictors. Table 5.7 displays the statistical differences between the abandoned and not abandoned projects. The p-value column is based on the calculations from running a Mann-Whitney U test.

By inspecting the table, it is clear that there are no statistically significant differences between the abandoned and not abandoned populations across any of the four indicators. In Figure 5.14, this lack of significance is visually confirmed by the violin plots. The distributions for both groups are highly similar, heavily overlapping in both their medians and overall spread.

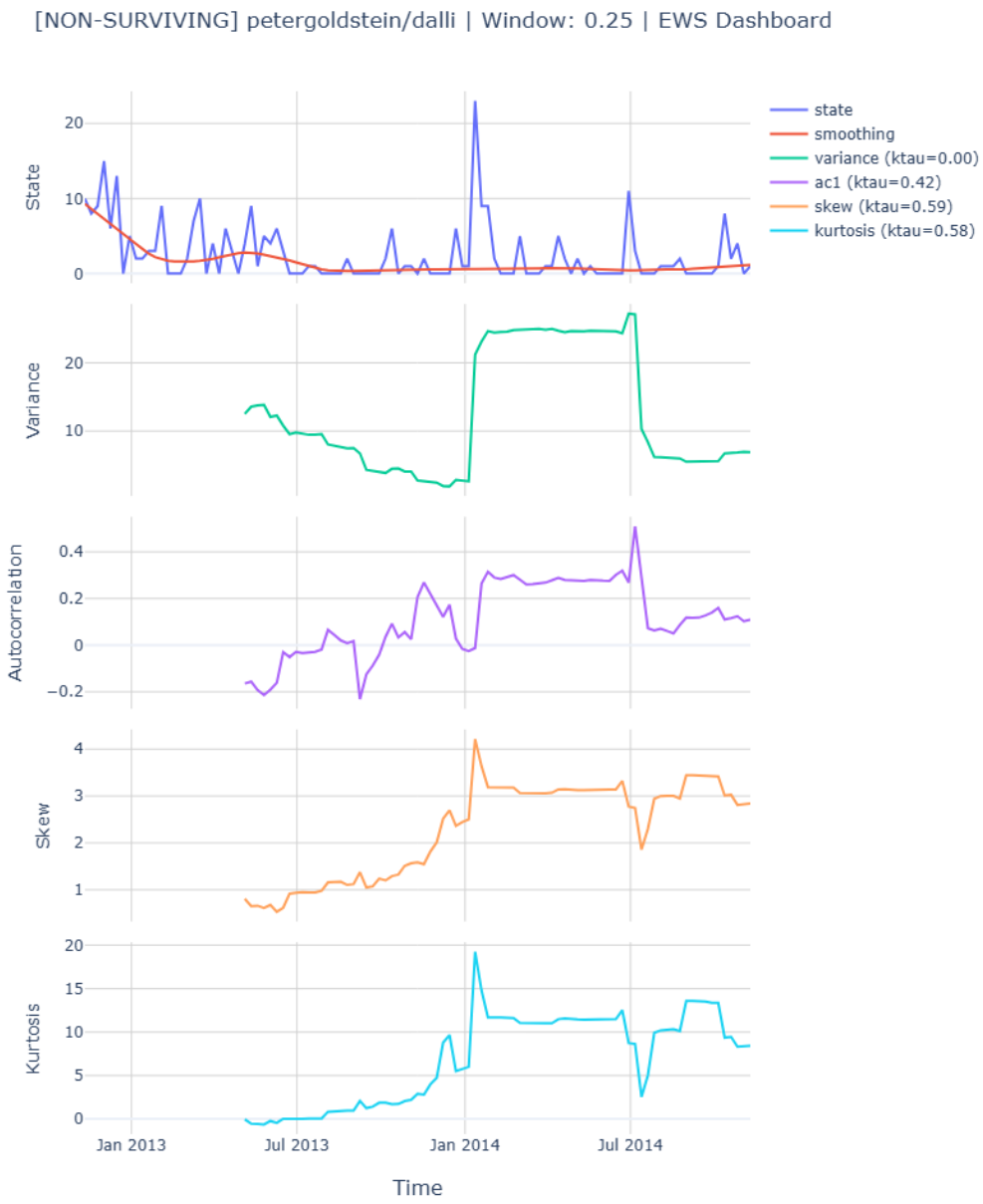

Figure 5.12: EWS for the abandoned project AP5

Table 5.7: Statistical Comparison of Kendall Tau (τ) Trends : Abandoned vs. Not Abandoned Projects

Indicator	Abandoned (Mean)	Not Abandoned (Mean)	p-value	Sig.
Var_τ	-0.189	-0.140	0.5767	No
$AC1_\tau$	0.044	0.095	0.8682	No
$Skew_\tau$	0.081	0.004	0.4391	No
$Kurt_\tau$	0.080	-0.015	0.4630	No

Yes = $p < 0.05$ using Mann-Whitney U test

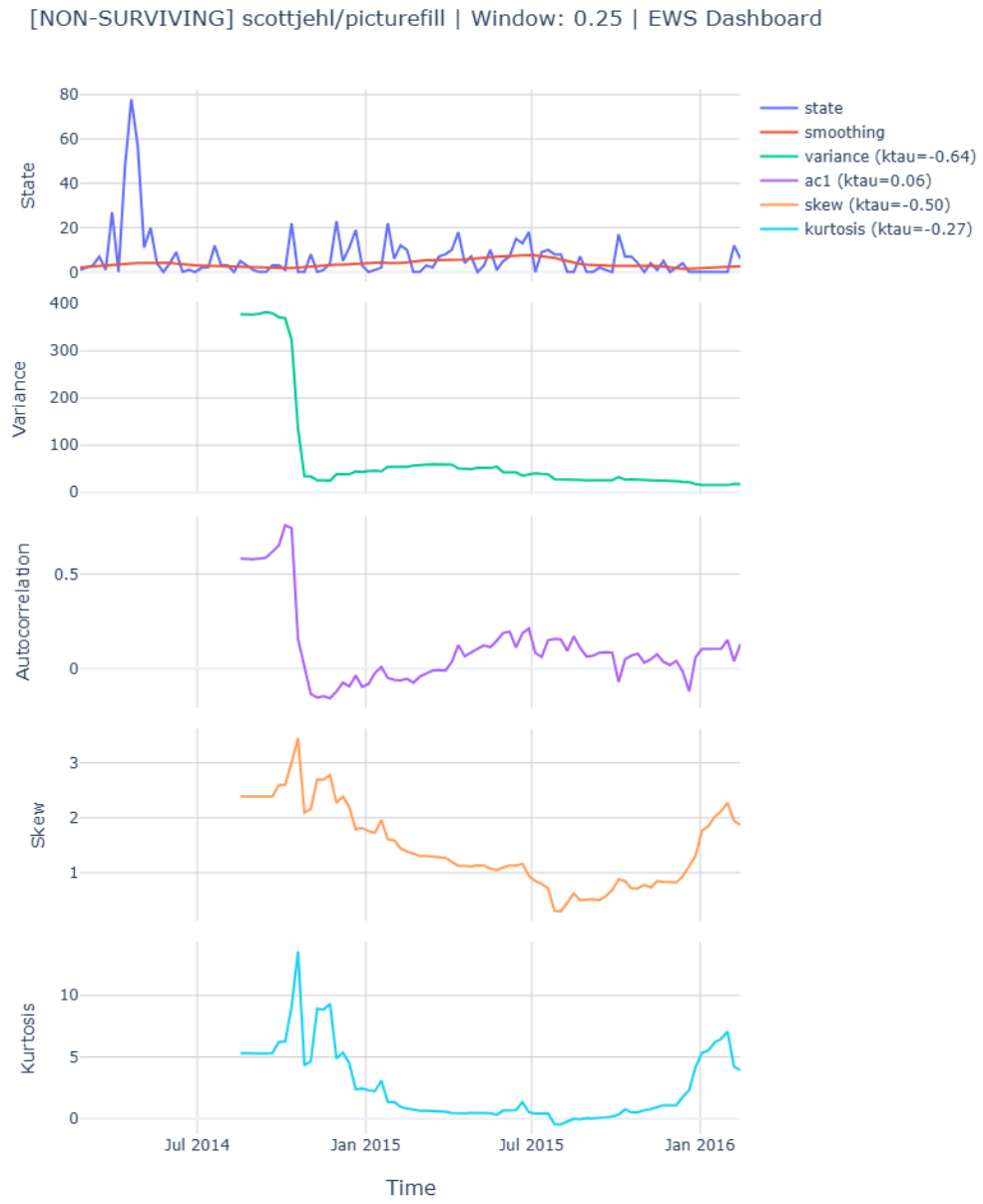

Figure 5.13: EWS for the abandoned project AP4

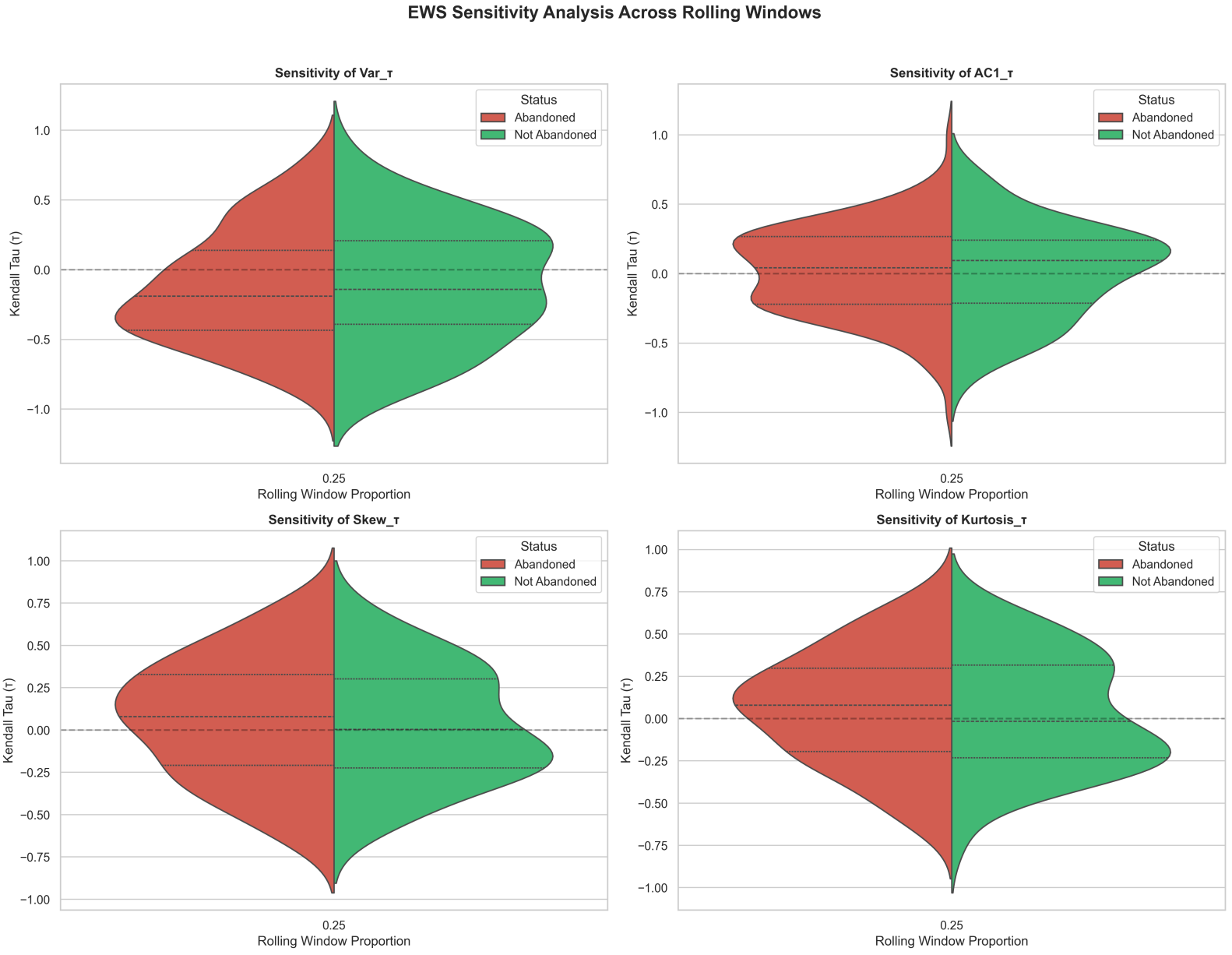

Figure 5.14: Violin Plots showcasing the distribution difference between Abandoned and Not Abandoned Projects

Summary for RQ3

The application of the *ewstools* framework across three distinct datasets demonstrates that while EWS in other domains, such as ecology, can describe individual project behaviors, their universal predictive reliability in SE is currently limited.

Individual Project Dynamics: EWS metrics successfully capture different trajectories leading to a tipping point. Some projects exhibit a volatile collapse with rising skewness and kurtosis (e.g., sudden bursts of late activity), while others experience a slow decrease characterized by stagnation and a negative Kendall τ value for variance.

The Challenge of Data Sparsity: Open-source development is inherently discrete. Frequent empty weeks interspersed with sudden commit spikes disrupt traditional trend-smoothing filters like LOWESS. These natural bursts can mathematically mimic instability, artificially inflating the EWS kurtosis and skewness.

Population-Level Inconsistency: Statistical comparisons between surviving and abandoned projects yielded mostly insignificant results. Because projects die in fundamentally different ways (flickering vs. fading), and healthy projects also experience bursty activity, the EWS indicators do not form a uniform pattern across a population. Therefore, while highly useful for profiling individual repository behavior, EWS metrics require domain-specific adaptation before they can reliably predict software tipping points.

6

Discussion

This chapter discusses the findings presented in the previous chapter in relation to the research questions, existing literature, and established tipping point theory from other domains. Furthermore, the chapter discusses the practical relevance of the findings, potential limitations of the study, and directions for future research.

6.1 RQ1 - Characteristics of Tipping Points in Software Engineering

The results from RQ1 show that tipping points in SE do not occur all at once. Instead, they build up over time. The most common signals, variance, autocorrelation, and skewness, all point in that direction. They suggest that the system becomes more unstable step by step, rather than suddenly breaking down.

An important takeaway from tipping point theory is that these signals reflect how complex systems behaves as they become unstable. Increasing variance indicates that a system becomes harder to predict. At the same time, higher autocorrelation suggests that problems persist rather than disappear. Together, these points to a reduced ability to recover from disturbances, where a system gradually moves toward a more fragile state.

From a SE perspective, this highlights a limitation in how project problems are often handled. Teams often solve issues one at a time through temporary fixes. This can involve patching bugs, changing requirements, or moving resources to deal with immediate problems. These actions may help stabilize the project in the short term, but they can also hide deeper structural issues in the system. Because of this, the underlying instability may continue to grow even when the visible problems seem resolved. The findings suggest that when larger symptoms finally appear, the project may already have reached a state where recovery becomes much more difficult.

Another point is that failure is rarely caused by one event. The studies included in the SLR describe failure as the result of many factors interacting over time. This fits well with the idea of tipping points. It is not a single event that causes the transition, but the combined effect of many small changes. As these changes build up, the system reaches a point where even a small disturbance can have a large

impact.

The findings closely match what has been observed in other domains. The same types of signals appear in areas such as climate science and ecology, with a similar distribution of EWS. This suggests that software systems are not as unique as they are sometimes treated, but instead behave like other complex systems as they approach instability.

At the same time, some EWS appear less often in the results. This does not necessarily mean that they are not relevant. A more likely explanation is that they are harder to detect with the type of data used in SE studies. Many of the analyzed papers were not written to capture system-level behavior in this way. This points to a gap in how software systems are currently studied.

It should also be noted that not all analyzed studies showed clear signs of EWS. Out of the total set of papers included in the SLR, a small subset did not exhibit patterns that could be interpreted as EWS. One possible explanation is that these studies did not capture system behavior over time in a way that makes such signals visible. Another possibility is that some systems do not reach a state where these patterns become detectable. This highlights that while EWS appear frequently, they are not guaranteed to be present in all cases.

Overall, the results provide a clearer understanding of what characterizes tipping points in SE. Rather than being defined by a single event, they are marked by a gradual loss of stability, most commonly reflected through increasing variance, autocorrelation and skewness. These signals capture how instability develops over time. Fluctuations increase and problems persist, which makes extreme events more likely. Tipping points are therefore not sudden, but instead emerge because recovery from disturbances becomes increasingly difficult.

6.2 RQ2 - Tipping Points Across Software Engineering Activities

The results of RQ2 suggest that tipping point characteristics are not confined to a single activity or stage within SE, but rather occur across a broad range of SWEBOK knowledge areas. This further suggests that tipping point behavior in software projects should be understood as a systemic phenomenon rather than a purely technical issue isolated to implementation or architecture. The findings support the view presented in earlier research that software projects behave as complex socio-technical systems, where interactions between people, processes, and technical components collectively influence project stability.

The most notable result was the prominence of Management and Requirements related tipping points. Existing SE literature has identified poor planning, unstable requirements, scope creep, and communication issues as major contributors to project failure. However, these factors are typically discussed as contributors to

failure rather than as signals of declining system resilience approaching a tipping point. While previous research acknowledges interactions between organizational and technical factors, these dynamics are rarely interpreted through a tipping point perspective focused on resilience loss and critical transitions. These observations, instead, suggest that such phenomena may represent early stages of broader tipping processes, where interacting instabilities reduce project resilience until an abrupt transition occurs.

The strong association between Requirements related tipping points and propagating tipping behavior is particularly important. Requirement instability rarely remains isolated within the requirements engineering phase itself. Instead, changes in requirements often propagate into other parts of the project, affecting architecture, testing, timelines, workload, and overall project complexity. This observation aligns with previous research on scope creep and requirements volatility, where requirement related changes have been shown to create cascading effects throughout software projects. This study extends this perspective by suggesting that these cascading effects may represent propagating tipping dynamics similar to those observed in other complex systems.

The prominence of Management related tipping points associated with societal impact further highlights the importance of socio-technical factors in SE stability. Existing literature has connected communication breakdowns, developer turnover, and organizational issues to unsuccessful project outcomes. The findings suggest that these factors may not only contribute to project failure individually, but may collectively reduce the resilience of the broader software system over time.

Taken together, these findings suggest that instability in SE projects may be better understood as an interconnected systems phenomenon rather than as a collection of isolated risks or failures.

6.3 RQ3 - The Tipping Point Dynamics of Software Systems

The application of the *ewstools* Python package to the datasets resulted in mixed results. It demonstrated the theoretical potential but also the limitations of transferring general tipping point detection methods based on EWS into SE domain. Tipping point theory usually relies on the continuous monitoring of natural systems, where a constant measurement is always available [9]. However, the application of this framework to software repositories reveals, among other things, a friction regarding data compatibility. Metrics derived from open-source repositories, namely commits and latency, are fundamentally discrete and characterized by episodic bursts of activity. For instance, it is common to observe several low- or zero-activity weeks followed immediately by intense bursts of contributions within a short amount of time. The lack of continuity disrupts mathematical detrending techniques such as the LOWESS filter used in this analysis. Consequently, normal developer behavior can be misinterpreted by the algorithm as outliers, influencing

the calculations of the EWS significance. This dynamic can be the cause of the false positives observed in this study, where entirely healthy repositories triggered, or did not trigger, the exact same EWS as failing ones. This points to a fundamental difference established in complex system theory. Unlike for example ecological systems, where a dying system exhibits increased variance as its destabilizes, a software project may exhibit less variance because the activities simply fade out over time with less overall fluctuation. A highly active and health software project naturally produces high variance due to release crunches and urgent fixes, explaining why these traditional EWS may be misinterpreted.

As highlighted by Avelino et al. [S7] and Foucault et al. [S15], many open-source projects are incredibly fragile because they depend on a very small number of core developers. In an ecological system, CSD is affected by a multitude of interacting organisms and systems. This is in contrast to software projects which can be susceptible to behaviors of single developers. The departure of a core developer can instantly destabilize the system, making it much harder to measure the resilience of the system itself rather than the individual human patterns. While this makes small open-source projects highly prone to sudden disruption, it is important to note that tipping point theory is likely much more relevant for larger software ecosystems. Projects with hundreds of developers possess the capacity to absorb individual shocks, meaning their eventual failure would much more closely mirror the gradual decline and critical slowing down seen in traditional ecological tipping points.

There is a notable discrepancy between the theoretical findings of RQ1 and RQ2 and the empirical results of RQ3. The SLR established that tipping points in SE are predominantly driven by socio-technical factors, such as requirements volatility and scope creep. However, the datasets used in the empirical evaluation were strictly limited to technical metrics, such as commit volumes and PRs. Consequently, it is plausible that EWS were present within the socio-technical dynamics of these projects, but were simply not captured by purely technical measurements.

The limitations of the *ewstools* package itself must also be considered. The tool is highly effective at computing the EWS such as the ones used in this analysis. However, it lacks functionality for other EWS such as Fisher Information, which Mayer et al. advocate the user for [29]. Additionally, it does not directly measure the return rate, which is a heavily emphasized metric in established literature such as Scheffer et al [26]. The return rate would be interesting to analysis, since there are many parallels with the SE "Stability Index" introduced by Destefantis et al [6].

Despite these limitations, there is still a promising foundation. The literature review highlighted that software project failure is rarely a linear event, but rather driven by interactions between dynamic systems, both technical and socio-technical. However, a key takeaway from this study is that factors such as data burstiness, the incompatibility of smoothing functions, and the heavy impact of single actors make SE a fundamentally unique ecosystem. Because of these distinct complex system dynamics, conventional EWS, as they are for example used in the *ewstools* package,

do not work without significant modification. Ultimately, bridging the current research gap requires more than just borrowing tools from other fields: the SE domain needs the development of its own dedicated class of EWS, tailored specifically to discrete, socio-technical human behavior rather than continuous complex systems.

6.4 Relevance to Practitioners

The findings of this study offer several insights for SE practitioners and researchers interested in studying tipping points within the SE domain. While an universal method for software projects is not yet viable, the concepts of tipping point theory can still be practically applied if adapted correctly.

Raw code volume metrics proved susceptible to noise and false positives. Therefore, practitioners should also explore alternative data sources. This study did not incorporate any socio-technical datasets, and that may have been a reason behind the discrepancy of results between RQ3 and the other two RQs. It is important for practitioners to measure all parts of a project, capturing not only the purely technical metrics. Software projects are also socio-technical systems, and measuring human interaction might provide more continuous and reliable data. Monitoring the sentiment of developer communications, the frequency of requirement changes, or the density of developer collaboration networks may yield more stable EWS than repository statistics alone. Therefore, it is highly recommended for projects to start implementing socio-technical monitoring systems, which could later be used in future studies and empirical applications.

The findings from the empirical application also highlight the impact of individual developers on overall system stability. Unlike ecological systems, which rely on interactions between a multitude of systems and organisms, software projects are more sensitive to single-actor dynamics. Therefore, software projects might have a more unpredictable nature, since the sudden departure of an individual can cause a ripple effect which can instantly destabilize the entire project.

For practitioners to successfully apply tipping point theory to SE, they must recognize a fundamental difference in system dynamics. Natural ecosystems rely on continuous interactions between a multitude of components. In contrast, software development is inherently discrete. Activity often occurs in sudden bursts, resulting in sparse data. Therefore, the most critical recommendation for future applications is to shift focus away from sparse repository statistics and apply these detection methods to socio-technical metrics that naturally produce continuous data streams. Because human interactions drive software systems, these socio-technical metrics will likely serve as far more accurate predictors of tipping points in this domain.

6.5 Threats to Validity

This section outlines potential threats to the validity of the study. The aim is not to remove all limitations, but to make clear how they may influence the results.

Construct validity: A central challenge is how tipping points are defined in a SE context. There are no primary studies that explicitly study tipping points in this domain. Because of this, the analysis relies on related concepts such as failure factors, instability, and project decline. These concepts are close to tipping point behavior, but they are not identical. The mapping is therefore based on interpretation, which means that some observations may not fully capture tipping point dynamics.

Internal validity: The identification of EWS depends on how patterns are interpreted from the selected studies. Most papers do not explicitly measure signals such as variance or autocorrelation. Instead, these patterns are inferred from descriptions of system behavior. This introduces a risk of interpretation bias, since different readers might draw different conclusions from the same material.

External validity: The results are based on a limited set of studies and datasets. Many of these focus on open-source projects, which may differ from industrial environments in terms of scale and available data. Because of this, the findings may not fully generalize to all SE contexts. While the results indicate that tipping point behavior exists, this study cannot fully determine to what extent the findings apply across different types of SE projects.

Reliability: The study follows a structured SLR process with defined search strategies and screening steps. However, parts of the process involve manual decisions, such as study selection and data interpretation. Even though consistent criteria were used, another researcher might make slightly different choices. This means that the results may vary to some extent if the study is repeated.

Overall, these limitations should be considered when interpreting the findings. They do not invalidate the results, but they highlight areas where further research is needed.

The usage of generative AI: Generative AI was used to enhance already written text by improving grammar and sentence structure. It was also used in the process of writing Python code in the empirical application process. All of the generated text and code was validated by the authors of this thesis. The generative AI models used were Gemini (Google) and also ChatGPT (OpenAI) [2], [4].

6.6 Future Research

This study serves as a first step toward introducing tipping point theory into the SE domain. Rather than providing a complete solution, it establishes a conceptual foundation and limitations that future work can build on.

One natural next step is to apply the identified concepts and EWS in real world SE contexts. While this study shows that tipping point behavior can be observed in existing datasets, further work is needed to evaluate how these signals behave in active projects. This includes studying how EWS evolve over time and whether they can be used to support decision making in practice.

Another important direction is the development of domain specific tools for tipping point detection in SE. Existing domain-agnostic tools, such as the *ewstools* used in this study, rely on general time series analysis. While they can be applied to SE data, they are not tailored to the specific characteristics of software projects. Future work could focus on designing tools that integrate SE metrics more directly, for example by combining repository data, team dynamics, and process related information.

Overall, future research should move from conceptual understanding toward practical implementation, with a focus on refining and applying tipping point detection in real SE environments.

7

Conclusion

This thesis investigated the application of tipping point theory and EWS to the SE domain. Project failure in SE has been treated as the linear accumulation of isolated risks and defects. This study sought to determine whether software projects instead behave as complex systems, exhibiting detectable signals of instability prior to a sudden collapse or abandonment.

Through a systematic literature review, the study established that software projects do experience EWS as they approach a tipping point. The loss of resilience is characterized by a gradual increase of instability, most frequently observable through increasing variance, autocorrelation, and skewness. The analysis revealed that tipping point dynamics are not confined to technical implementation. They often occur within the Requirements and Management knowledge areas, where localized instabilities, such as scope creep or core developer turnover, frequently cascade into propagating or societal tipping points that destabilize the entire socio-technical system.

The empirical application of the *ewstools* package on open-source datasets exposed limitations in transferring tipping point detection methods directly into SE. Traditional EWS successfully profiled the collapse of individual repositories, identifying both volatile "flickering" and gradual "fades". However, they failed to serve as reliable and universal predictors at the population level. The data generated by open-source development is inherently discrete and "bursty" and driven by human patterns rather than continuous ecological systems. These sudden bursts of activity disrupt mathematical smoothing filters, artificially inflating statistical moments and triggering false positive alarms in completely healthy projects. The fragility of small open-source teams means that indicators often track individual human behavior rather than true systemic resilience.

Ultimately, this research concludes that SE constitutes a fundamentally unique socio-technical ecosystem. While the theoretical foundation of tipping points holds significant promise for shifting industry risk management from isolated defect tracking to systemic resilience monitoring, ecological EWS cannot be applied blindly. To bridge the current research gap, the SE domain must move beyond borrowing tools from other fields and focus on developing its own dedicated class of EWS, tailored specifically to the discrete, human-driven dynamics of software development.

Bibliography

[1] K. Bhadauria, "From analysis to action: Categorizing and addressing it project failures for enhanced success rates," in *Proceedings of International Conference on Computing Systems and Intelligent Applications (ComSIA 2025)*, 2026, pp. 65–77.

[2] Google, *Gemini*, Large language model software, Accessed January 2026 – May 2026, 2026. [Online]. Available: `https://gemini.google.com`.

[3] H.-M. Heyn, Y. Mao, R. Weiß, and E. Knauss, "Causal models for specifying requirements in industrial ml-based software: A case study," *Journal of Systems and Software*, p. 112 691, 2026.

[4] OpenAI, *Chatgpt*, Large language model software, Accessed January 2026 – May 2026, 2026. [Online]. Available: `https://chatgpt.com`.

[5] J. Ungvarsky, *Content analysis.* Mar. 2026. [Online]. Available: `https://research.ebsco.com/linkprocessor/plink?id=00409752-62be-3efd-9478-b41571604b31`.

[6] G. Destefanis, S. Bartolucci, D. Graziotin, R. Neykova, and M. Ortu, "Introducing repository stability," in *Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering*, ser. FSE Companion '25, Association for Computing Machinery, 2025, pp. 555–560.

[7] M. Robredo, M. Esposito, D. Taibi, R. Peñaloza, and V. Lenarduzzi, *Squad: The software quality dataset*, 2025.

[8] V. Dakos et al., "Supplement of tipping point detection and early warnings in climate, ecological, and human systems," *Earth System Dynamics*, 2024.

[9] V. Dakos et al., "Tipping point detection and early warnings in climate, ecological, and human systems," *Earth System Dynamics*, pp. 1117–1135, 2024. DOI: `10.5194/esd-15-1117-2024`.

[10] *Guide to the Software Engineering Body of Knowledge (SWEBOK Guide), Version 4.0.* 2024.

[11] T. M. Bury, "Ewstools: A python package for early warning signals of bifurcations in time series data," *Journal of Open Source Software*, vol. 8, no. 82,

p. 5038, 2023. DOI: 10.21105/joss.05038. [Online]. Available: https://doi.org/10.21105/joss.05038.

[12] T. M. Lenton et al., "Remotely sensing potential climate change tipping points across scales," *Nature Communications*, pp. 1–14, 2023.

[13] B. Trinkenreich et al., "A model for understanding and reducing developer burnout," pp. 48–60, 2023.

[14] J. M. Drake, S. M. O'Regan, V. Dakos, S. Kéfi, and P. Rohani, "Alternative stable states, tipping points, and early warning signals of ecological transitions," in *Theoretical Ecology: concepts and applications*, 2020.

[15] M. Hamid, F. Zeshan, A. Ahmad, and E. Aimeur, "Factors contributing in failures of software projects," *International Journal of Computer Science and Network Security*, pp. 62–70, 2019.

[16] M. Ibraigheeth and S. A. Fadzli, "Core factors for software projects success," *International Journal on Informatics Visualization*, 2019.

[17] B. Komal, U. I. Janjua, and T. M. Madni, "Identification of scope creep factors and their impact on software project success," in *IEEE Conference Proceedings*, 2019.

[18] A. Génin et al., "Monitoring ecosystem degradation using spatial data and the r package spatialwarnings," *Methods in Ecology and Evolution*, pp. 2067–2075, 2018.

[19] K.-J. Stol and B. Fitzgerald, "The ABC of software engineering research," *ACM Transactions on Software Engineering and Methodology*, vol. 27, no. 3, 2018. DOI: 10.1145/3241743.

[20] G. Guillaume-Joseph, "Improving software project outcomes through predictive analytics," *IEEE Engineering Management Review*, 2015.

[21] V. Dakos, S. R. Carpenter, W. A. Brock, A. M. Ellison, V. Guttal, A. R. Ives, et al., "Methods for detecting early warnings of critical transitions in time series illustrated using simulated ecological data," *PLoS ONE*, e41010, 2012. DOI: https://doi.org/10.1371/journal.pone.0041010.

[22] J. Textor, J. Hardt, and S. Knüppel, "DAGitty: A graphical tool for analyzing causal diagrams," *Epidemiology (Cambridge, Mass.)*, vol. 22, no. 5, p. 745, 2011. DOI: 10.1097/EDE.0b013e318225c2be.

[23] M. Kothari, "Early warning signs in software projects," M.S. thesis, Auckland University of Technology, 2010.

[24] R. Biggs, S. R. Carpenter, and W. A. Brock, "Turning back from the brink: Detecting an impending regime shift in time to avert it," *Proceedings of the National Academy of Sciences*, pp. 826–831, 2009. DOI: https://doi.org/10.1073/pnas.0811729106.

[25] V. Guttal and C. Jayaprakash, "Spatial variance and spatial skewness: Leading indicators of regime shifts in spatial ecological systems," *Theoretical Ecology*, pp. 3–12, 2009.

[26] M. Scheffer et al., "Early-warning signals for critical transitions," *Nature*, pp. 53–59, 2009.

[27] V. Guttal and C. Jayaprakash, "Changing skewness: An early warning signal of regime shifts in ecosystems," *Ecology Letters*, pp. 450–460, 2008.

[28] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," 2007.

[29] A. L. Mayer, C. W. Pawlowski, and H. Cabezas, "Fisher information and dynamic regime changes in ecological systems," *Ecological Modelling*, pp. 72–82, 2006.

[30] I. O. Nikander, "Early warnings: A phenomenon in project management," Ph.D. dissertation, 2002.

[31] S. McConnell, *Software project survival guide: How to be sure your first important project isn't your last*. 1998.

A

Systematic Literature Review References

Referenced SLR Studies

[S1] G. Annunziata, S. Lambiase, F. Palomba, G. Catolino, and F. Ferrucci, "How do communities of ml-enabled systems smell? a cross-sectional study on the prevalence of community smells," in *Proceedings of the 29th International Conference on Evaluation and Assessment in Software Engineering*, 2025, pp. 272–282.

[S2] M. Fan, Y. Zhang, K.-J. Stol, and H. Liu, "Core developer turnover in the rust package ecosystem: Prevalence, impact, and awareness," *Proc. ACM Softw. Eng.*, vol. 2, no. FSE, Jun. 2025. DOI: 10.1145/3729392.

[S3] M. Shehata, S. Makhkamjonoov, M. Syed, and E. Parra, "Cascading effects: Analyzing project failure impact in the maven central ecosystem," in *2025 IEEE/ACM 22nd International Conference on Mining Software Repositories (MSR)*, 2025, pp. 309–313.

[S4] O. Gordieiev, D. Gordieieva, A. Rainer, and O. Pishchukhina, "Relationship between factors influencing the software development process and software defects," in *2023 13th International Conference on Dependable Systems, Services and Technologies (DESSERT)*, 2023, pp. 1–7.

[S5] A. Ait, J. L. C. Izquierdo, and J. Cabot, "An empirical study on the survival rate of github projects," in *2022 IEEE/ACM 19th International Conference on Mining Software Repositories (MSR)*, 2022, pp. 365–375. DOI: 10.1145/3524842.3527941.

[S6] B. Komal et al., "The impact of scope creep on project success: An empirical investigation," *IEEE Access*, vol. 8, pp. 125 755–125 775, 2020. DOI: 10.1109/ACCESS.2020.3007098.

[S7] G. Avelino, E. Constantinou, M. T. Valente, and A. Serebrenik, *On the abandonment and survival of open source projects: An empirical investigation*, 2019. DOI: https://doi.org/10.48550/arXiv.1906.08058. arXiv: 1906.08058 [cs.SE].

[S8] A. A. Zafar et al., "Taxonomy of factors causing integration failure during global software development," *IEEE Access*, vol. 6, pp. 22 228–22 239, 2018. DOI: 10.1109/ACCESS.2017.2782843.

[S9] J. Coelho and M. T. Valente, "Why modern open source projects fail," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2017, Paderborn, Germany: Association for Computing Machinery, 2017, pp. 186–196, ISBN: 9781450351058. [Online]. Available: https://doi.org/10.1145/3106237.3106246.

[S10] A. R. Adam and M. Danaparamita, "Understanding the influence of poor scope management affecting the successful of an it project," in *2016 International Conference on Information Management and Technology (ICIMTech)*, 2016, pp. 124–129.

[S11] A. Alami, "Why do information technology projects fail?" *Procedia Computer Science*, pp. 62–71, 2016.

[S12] M. Ansar and T. A. Khan, "Non-technical issues in software designing phase," in *2016 Sixth International Conference on Innovative Computing Technology (INTECH)*, 2016, pp. 179–184.

[S13] P. Mafra, M. Kalinowski, D. Méndez Fernández, M. Felderer, and S. Wagner, "Towards guidelines for preventing critical requirements engineering problems," in *2016 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 2016, pp. 25–29.

[S14] N. Abdullah, M. A. Jabar, F. Sidi, and Y. Yah, "Early prediction model to manage risks from earliest stages to increase project success," *Journal of Theoretical and Applied Information Technology*, pp. 413–419, 2015.

[S15] M. Foucault, M. Palyart, X. Blanc, G. C. Murphy, and J.-R. Falleri, "Impact of developer turnover on quality in open-source software," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, 2015, pp. 829–841.

[S16] J. L. King and C. P. Simon, "Complications with complexity in requirements," *ACM Trans. Manage. Inf. Syst.*, 2015.

Additional Included SLR Studies

Z. Feng, K. Kimura, B. Trinkenreich, I. Steinmacher, M. Gerosa, and A. Sarma, "Addressing oss community managers' challenges in contributor retention," *ACM Trans. Softw. Eng. Methodol.*, 2025.

Z. Liu, Y. Shen, Z. Zhang, Y. Guo, and H. Sun, "On the prediction of open source software ecosystem health based on time series analysis," in *Computer Supported Cooperative Work and Social Computing*, 2025, pp. 138–149.

A. F. Rashid, M. A. Hagal, and F. M. Naser El-Firjani, "A study to investigate software failure factors in libyan organizations," in *2024 International Symposium of Systems, Advanced Technologies and Knowledge (ISSATK)*, 2024, pp. 1–6.

J. Schmidt, "An it project postmortem: Identifying root causes and eliminating rival explanations," *Journal of Information Technology Case and Application Research*, pp. 318–364, 2024.

S. Shafiq, C. Mayr-Dorn, A. Mashkoor, and A. Egyed, "Balanced knowledge distribution among software development teams—observations from open- and closed-source software development," *Journal of Software: Evolution and Process*, 2024.

M. Y. Kotowaroo and R. K. Sungkur, "Success and failure factors affecting software development projects from it professionals' perspective," in *Soft Computing for Security Applications*, 2023, pp. 757–772.

J. Schmidt, "Mitigating risk of failure in information technology projects: Causes and mechanisms," *Project Leadership and Society*, p. 100 097, 2023.

M. Joblin and S. Apel, "How do successful and failed projects differ? a socio-technical analysis," *ACM Trans. Softw. Eng. Methodol.*, 2022.

A. Nizam, "Software project failure process definition," *IEEE Access*, pp. 34 428–34 441, 2022.

L. Bao, X. Xia, D. Lo, and G. C. Murphy, "A large scale study of long-time contributor prediction for github projects," *IEEE Transactions on Software Engineering*, pp. 1277–1298, 2021.

S. H. Kaisler, W. H. Money, and S. Cohen, "Forensic analysis of failing software projects issues and challenges," 2021, pp. 960–969.

S. Lauesen, "It project failures, causes and cures," *IEEE Access*, pp. 72 059–72 067, 2020.

H. A. H. Gondal, S. M. U. Din, S. Fayyaz, M. D. Zeb, and B. Nadeem, "Preeminent risk factor affecting software development," in *2018 International Conference on Advancements in Computational Sciences (ICACS)*, 2018, pp. 1–7.

S. Linßen, D. Basten, and J. Richter, "Antecedents and consequences of time pressure in scrum projects: Insights from a qualitative study," 2018.

Y. Yan, P. Liao, and Z. Zhang, "An ontology framework of software requirements change management process based on causality," in *Proceedings of the 2nd International Conference on Information System and Data Mining*, 2018, pp. 107–111.

D. Ehls, "Open source project collapse - sources and patterns of failure," 2017.

C. W. Chesterman, K. Castelle, and J. J. Shauger, "Interpreting barriers to success in software development and deployment using systems theory," *International Journal of System of Systems Engineering*, pp. 143–158, 2016.

D. L. Hughes, Y. K. Dwivedi, N. P. Rana, and A. C. Simintiras, "Information systems project failure – analysis of causal links using interpretive structural modelling," *Production Planning and Control*, pp. 1313–1333, 2016.

D. Méndez Fernández et al., "Naming the pain in requirements engineering: Contemporary problems, causes, and effects in practice," *Empirical Software Engineering*, 2016.

G. Guillaume-Joseph and J. S. Wasek, "Improving software project outcomes through predictive analytics: Part 1," *IEEE Engineering Management Review*, pp. 26–38, 2015.

G. Guillaume-Joseph and J. S. Wasek, "Improving software project outcomes through predictive analytics: Part 2," *IEEE Engineering Management Review*, pp. 39–49, 2015.

R. Jiang, "A novel risk metric for staff turnover in a software project based on information entropy," *Entropy*, pp. 2834–2852, 2015.

A. Shimoda, "Causal analysis of system failures from it project trouble reports," in *2015 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*, 2015, pp. 1302–1306.

B

Appendix

B.1 Reclassification of EWS

The EWS and their classification are based on the paper and supplement paper by Dakos et al. [8], [9].

- **Autocorrelation:** autocorrelation, spatial_correlation, correlations
- **Variance:** variance, spatial_variance, detrended_fluctuation_analysis
- **Skewness:** skewness, spatial_skewness

B.2 Individual project plots of RQ3

B.2.1 Core Developer Turnover in the Rust Package Ecosystem Commit Metric Plots

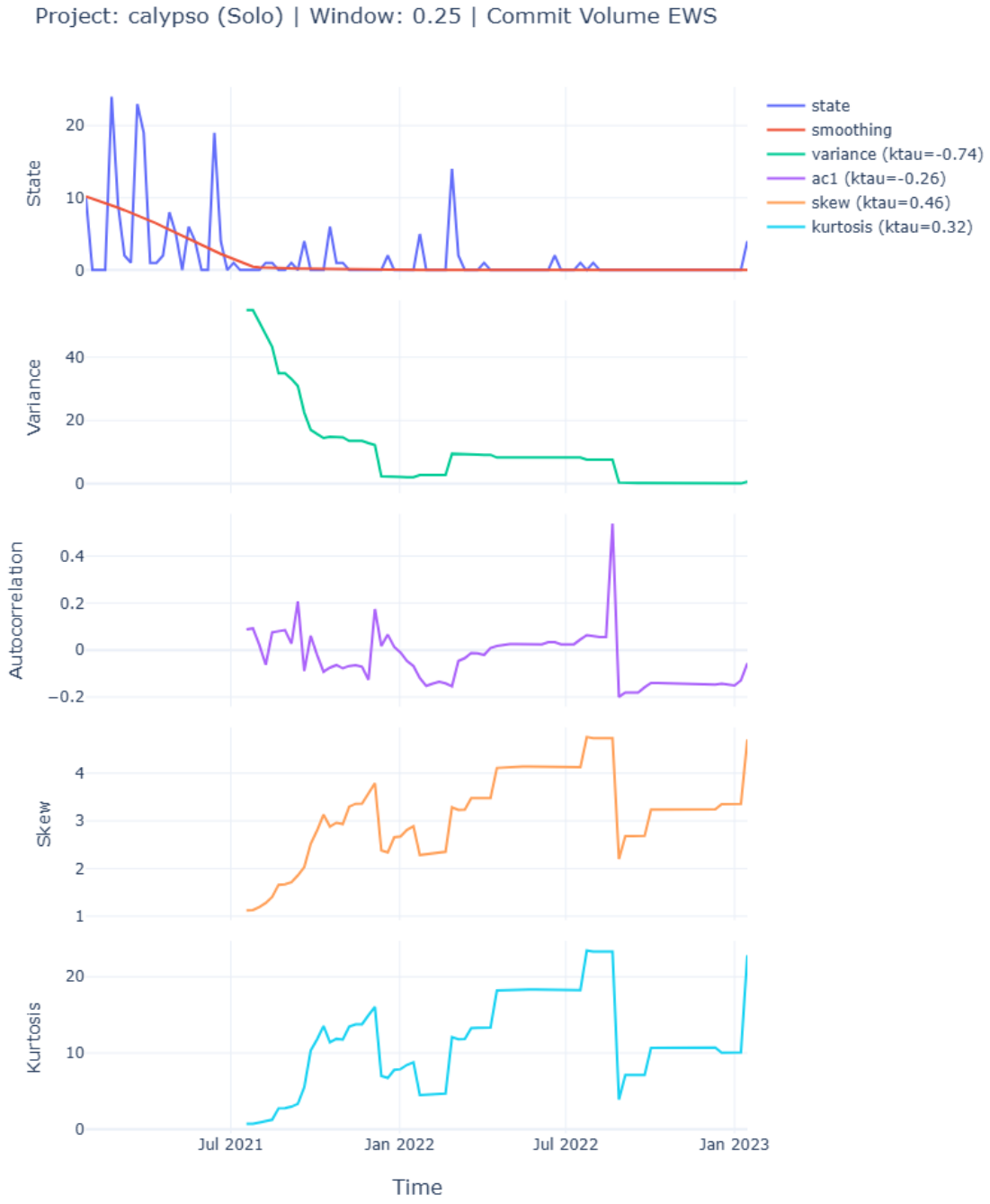

Figure B.1: EWS Graphs for Project with ID RP1 from the Rust Package Ecosystem Dataset (Commit Metric)

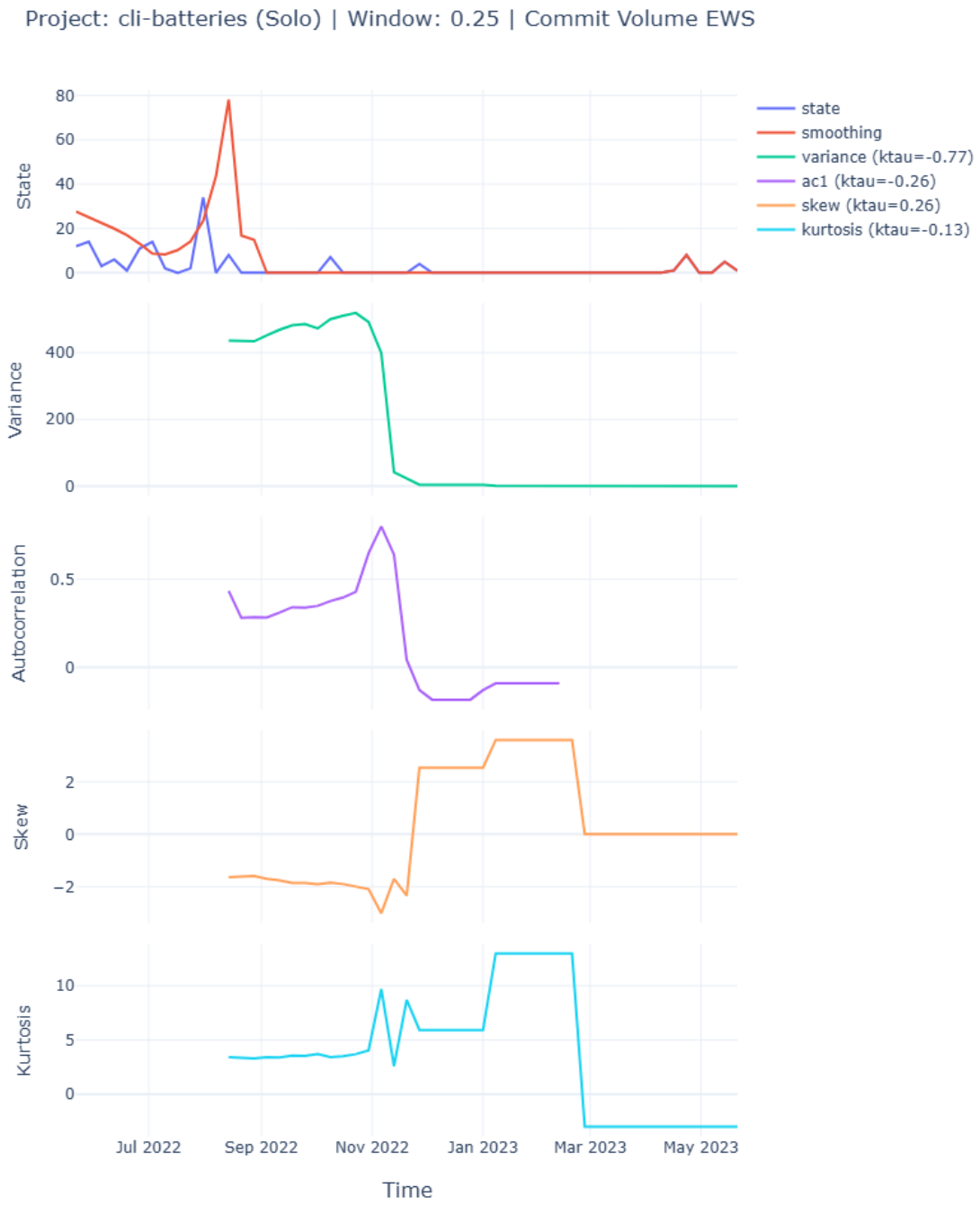

Figure B.2: EWS Graphs for Project with ID RP2 from the Rust Package Ecosystem Dataset (Commit Metric)

Figure B.3: EWS Graphs for Project with ID RP3 from the Rust Package Ecosystem Dataset (Commit Metric)

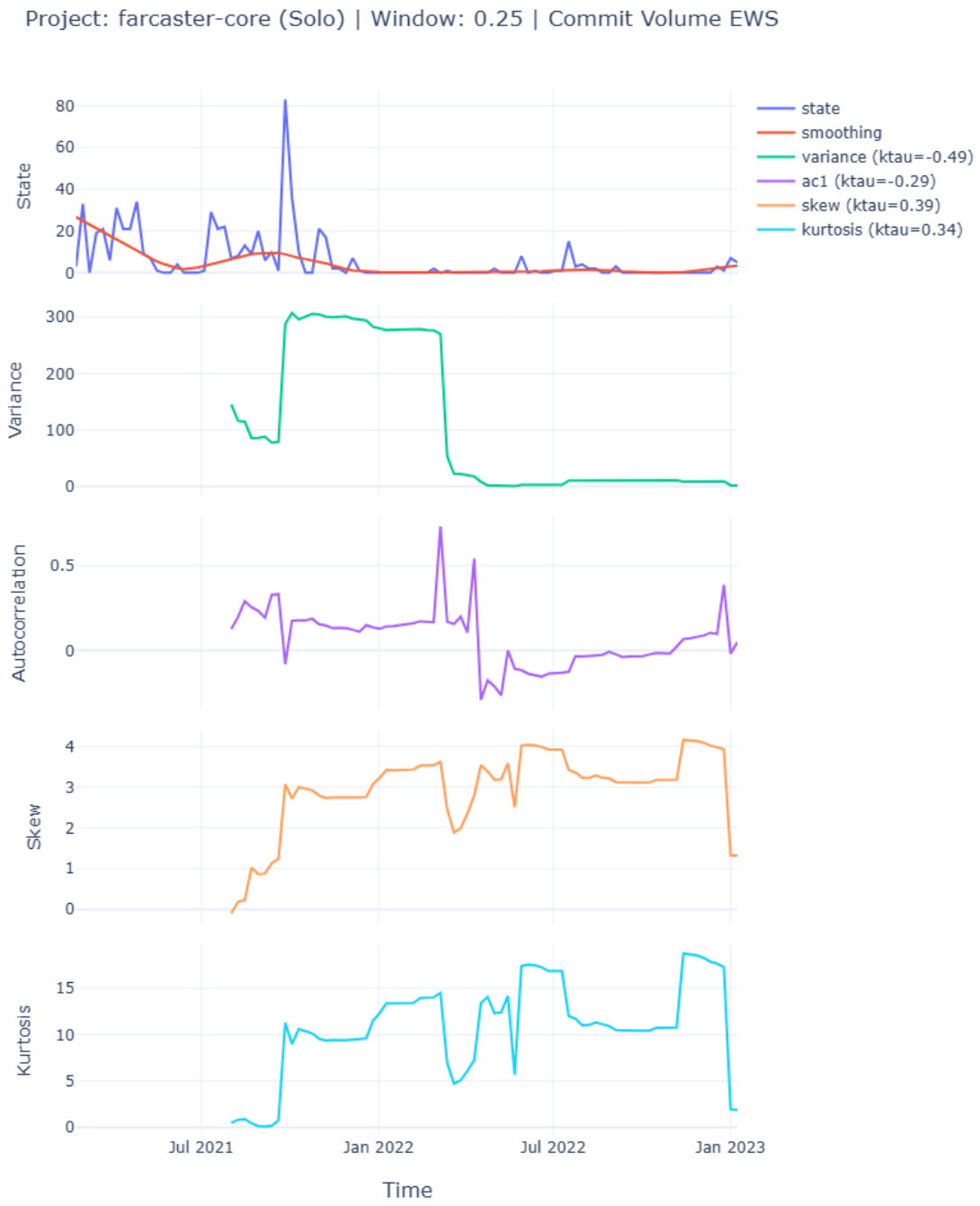

Figure B.4: EWS Graphs for Project with ID RP4 from the Rust Package Ecosystem Dataset (Commit Metric)

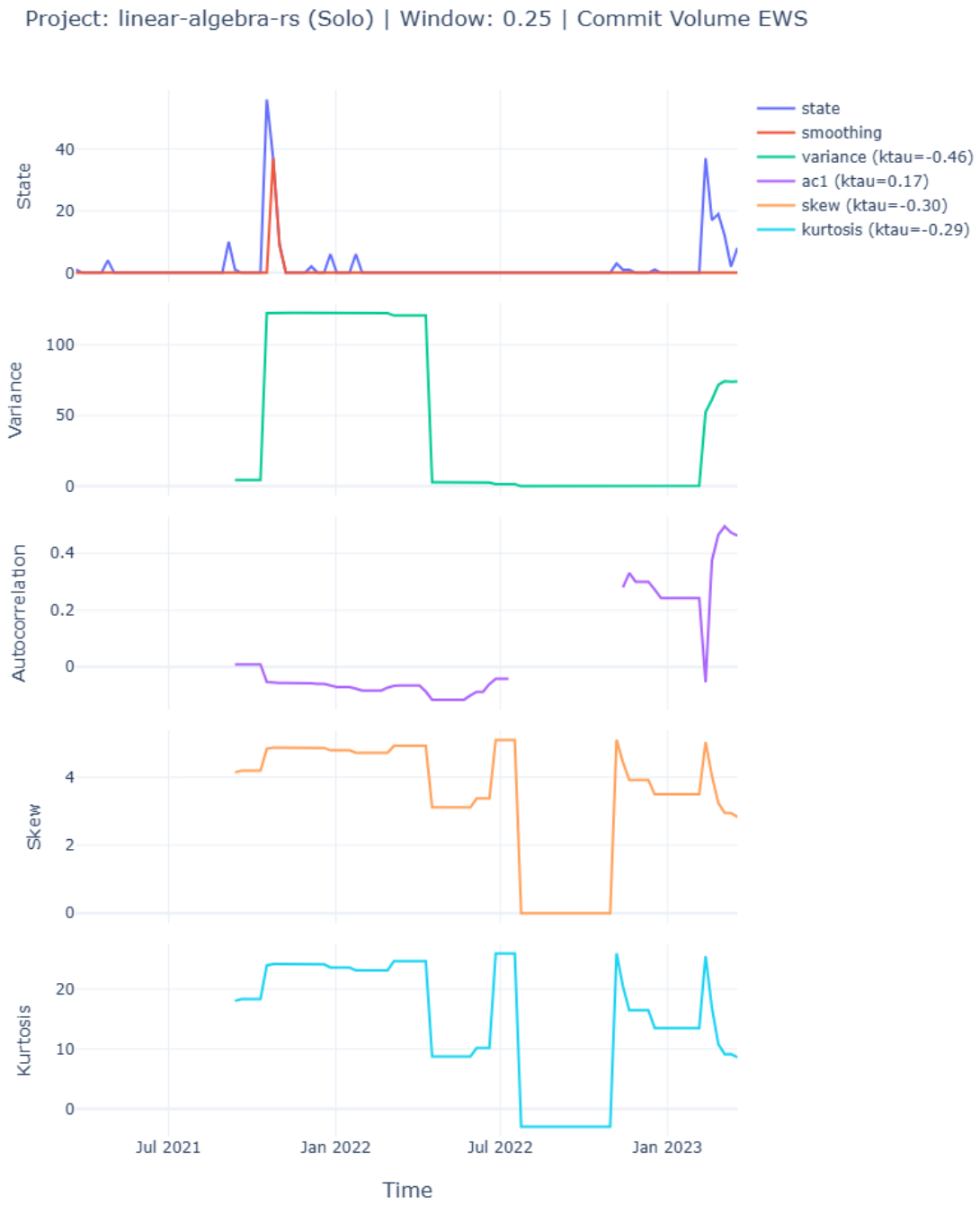

Figure B.5: EWS Graphs for Project with ID RP5 from the Rust Package Ecosystem Dataset (Commit Metric)

Figure B.6: EWS Graphs for Project with ID RP6 from the Rust Package Ecosystem Dataset (Commit Metric)

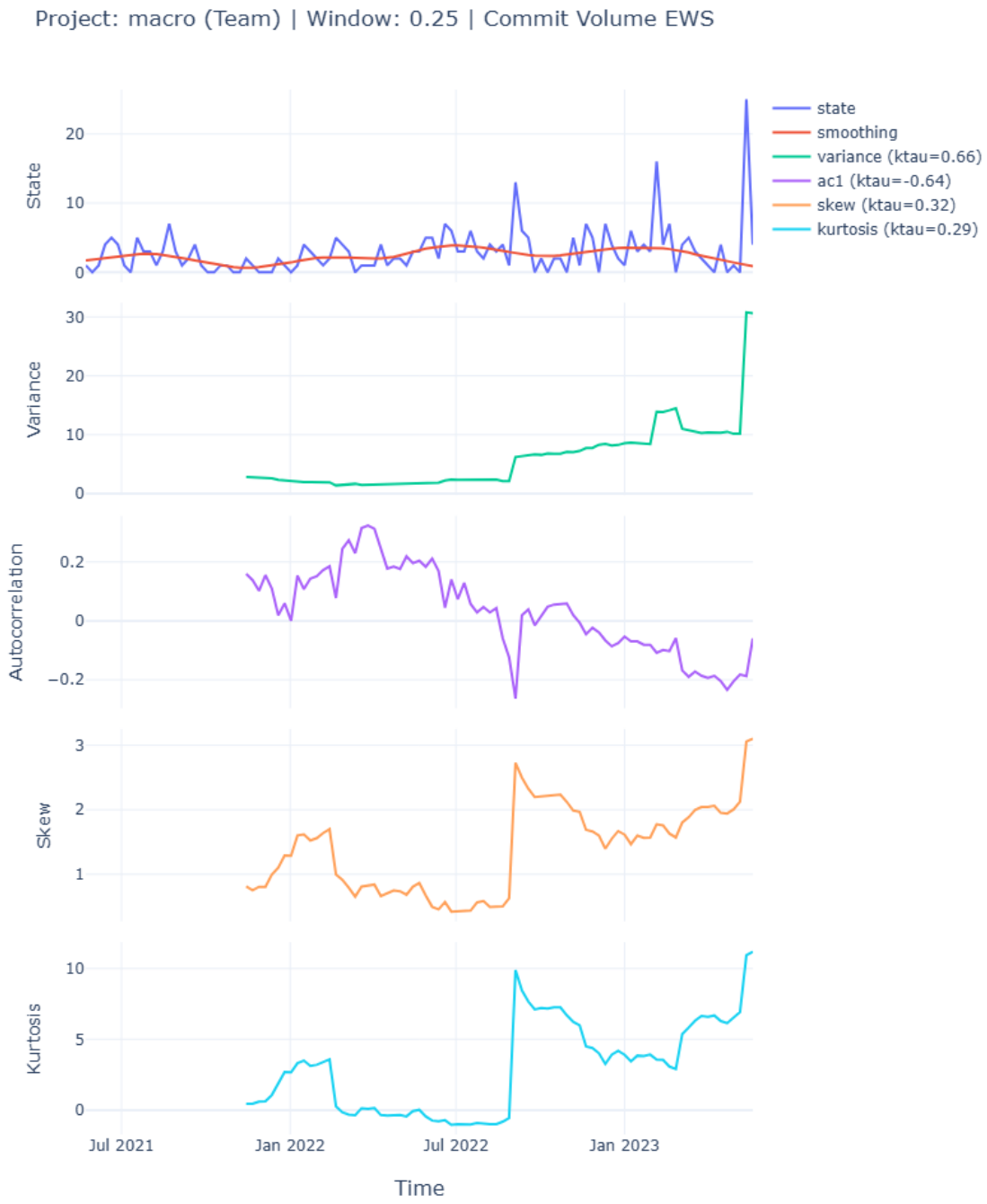

Figure B.7: EWS Graphs for Project with ID RP7 from the Rust Package Ecosystem Dataset (Commit Metric)

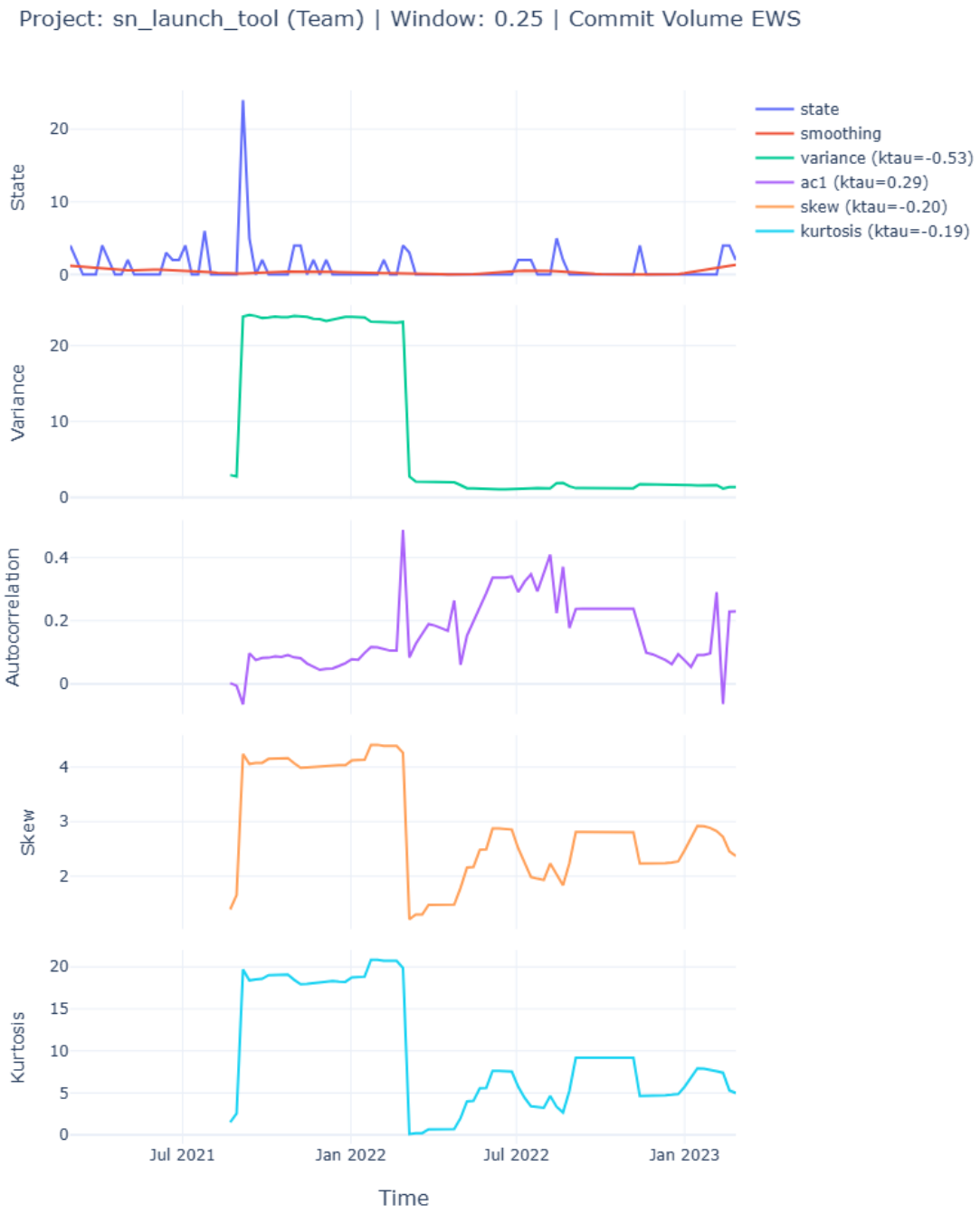

Figure B.8: EWS Graphs for Project with ID RP8 from the Rust Package Ecosystem Dataset (Commit Metric)

Figure B.9: EWS Graphs for Project with ID RP9 from the Rust Package Ecosystem Dataset (Commit Metric)

Figure B.10: EWS Graphs for Project with ID RP10 from the Rust Package Ecosystem Dataset (Commit Metric)

B.2.2 Core Developer Turnover in the Rust Package Ecosystem Latency Metric Plots

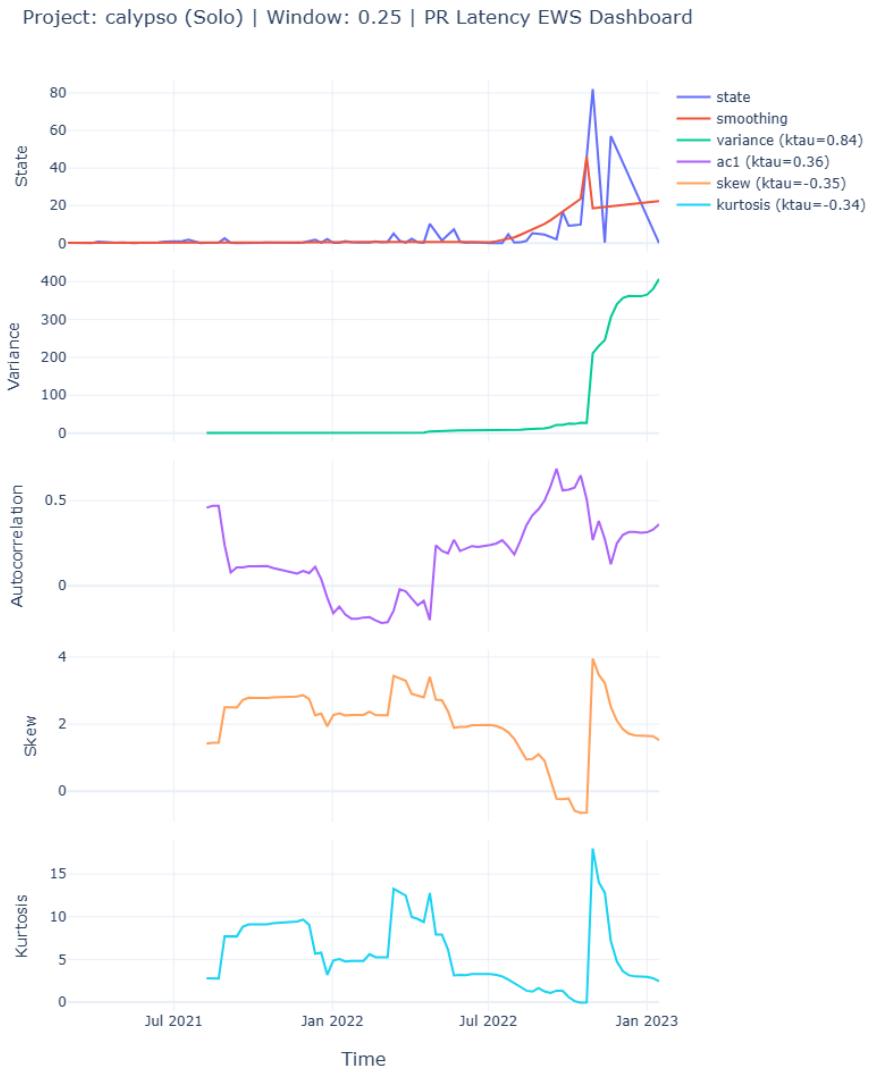

Figure B.11: EWS Graphs for Project with ID RP1 from the Rust Package Ecosystem Dataset (Latency Metric)

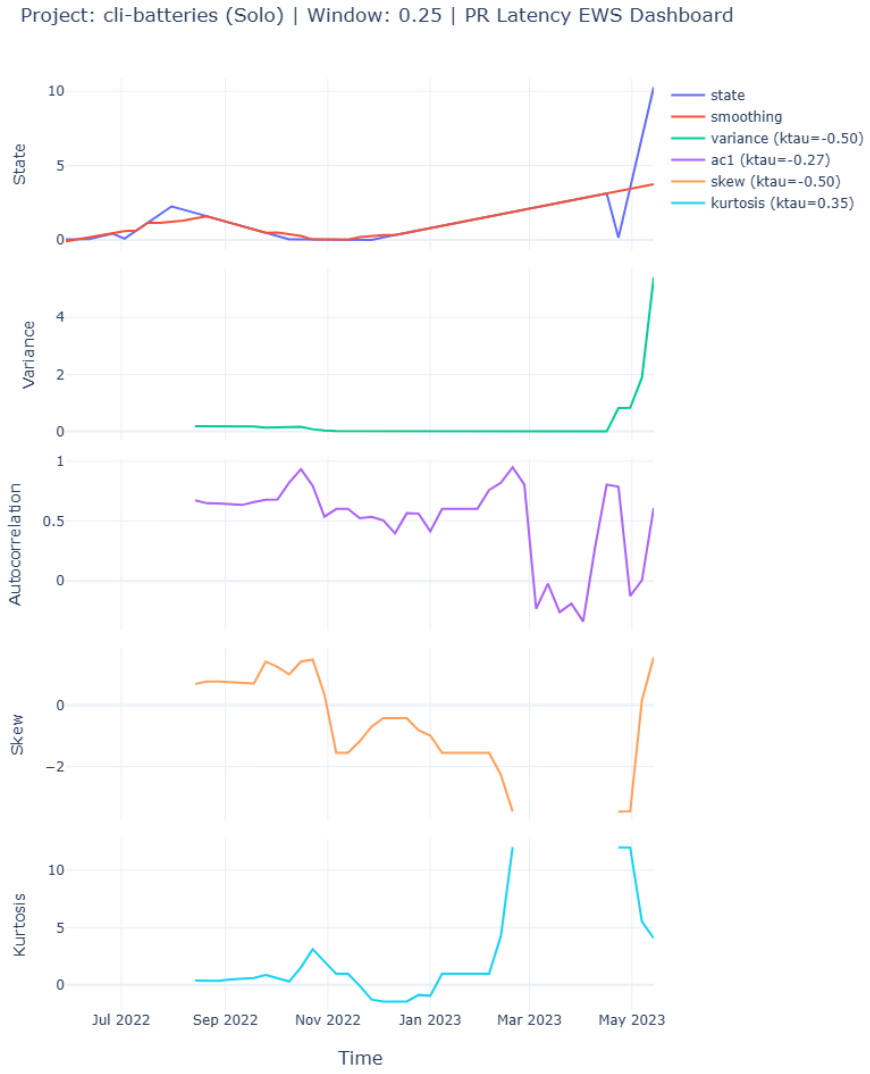

Figure B.12: EWS Graphs for Project with ID RP2 from the Rust Package Ecosystem Dataset (Latency Metric)

Figure B.13: EWS Graphs for Project with ID RP3 from the Rust Package Ecosystem Dataset (Latency Metric)

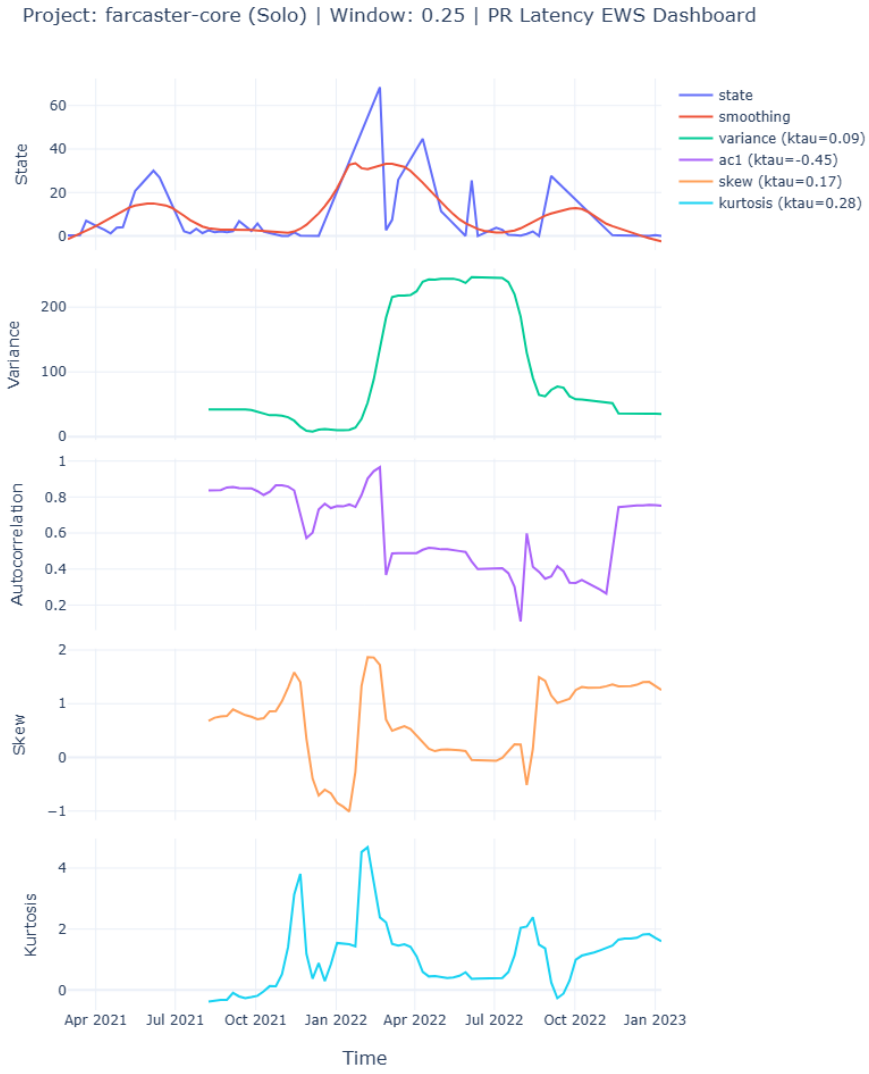

Figure B.14: EWS Graphs for Project with ID RP4 from the Rust Package Ecosystem Dataset (Latency Metric)

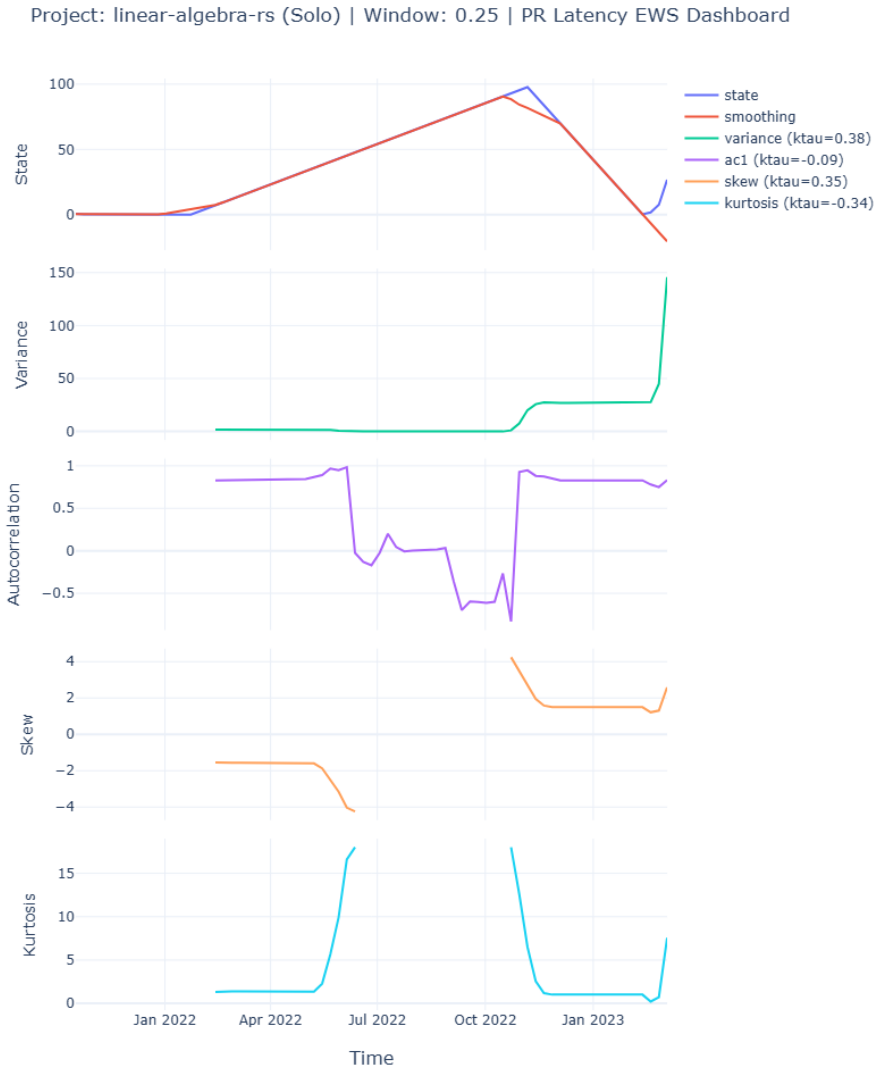

Figure B.15: EWS Graphs for Project with ID RP5 from the Rust Package Ecosystem Dataset (Latency Metric)

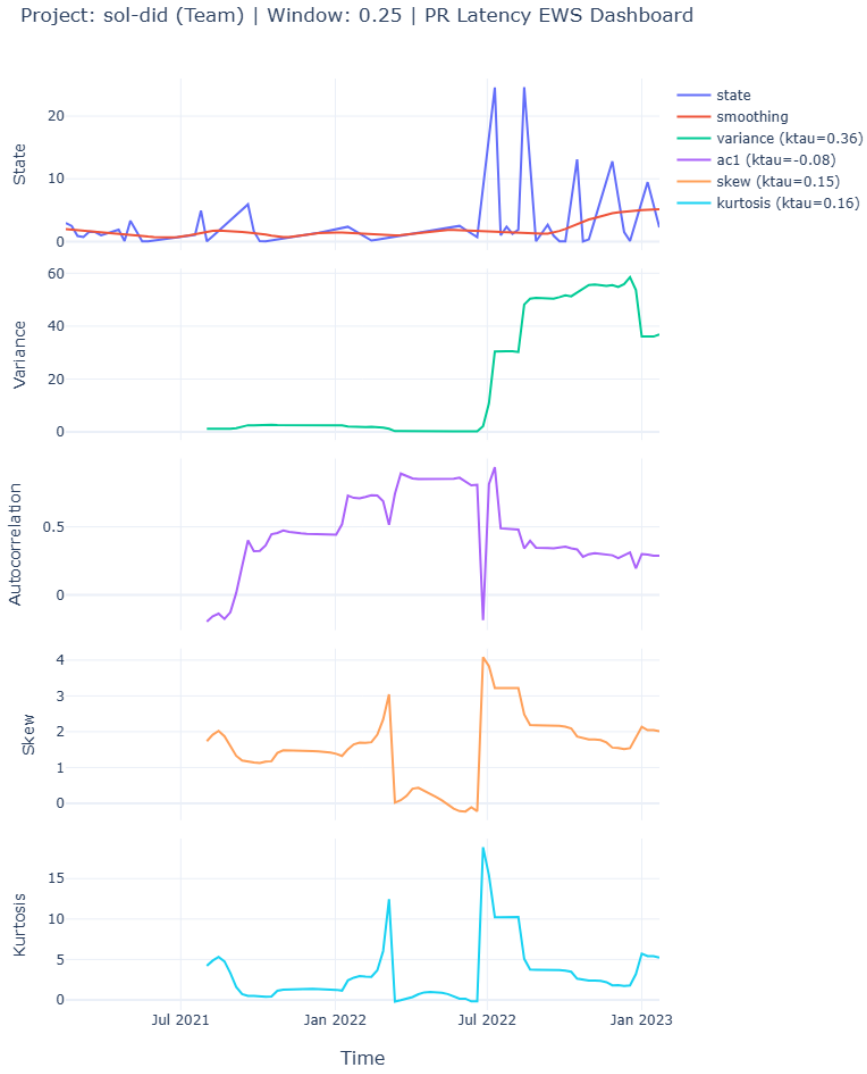

Figure B.16: EWS Graphs for Project with ID RP6 from the Rust Package Ecosystem Dataset (Latency Metric)

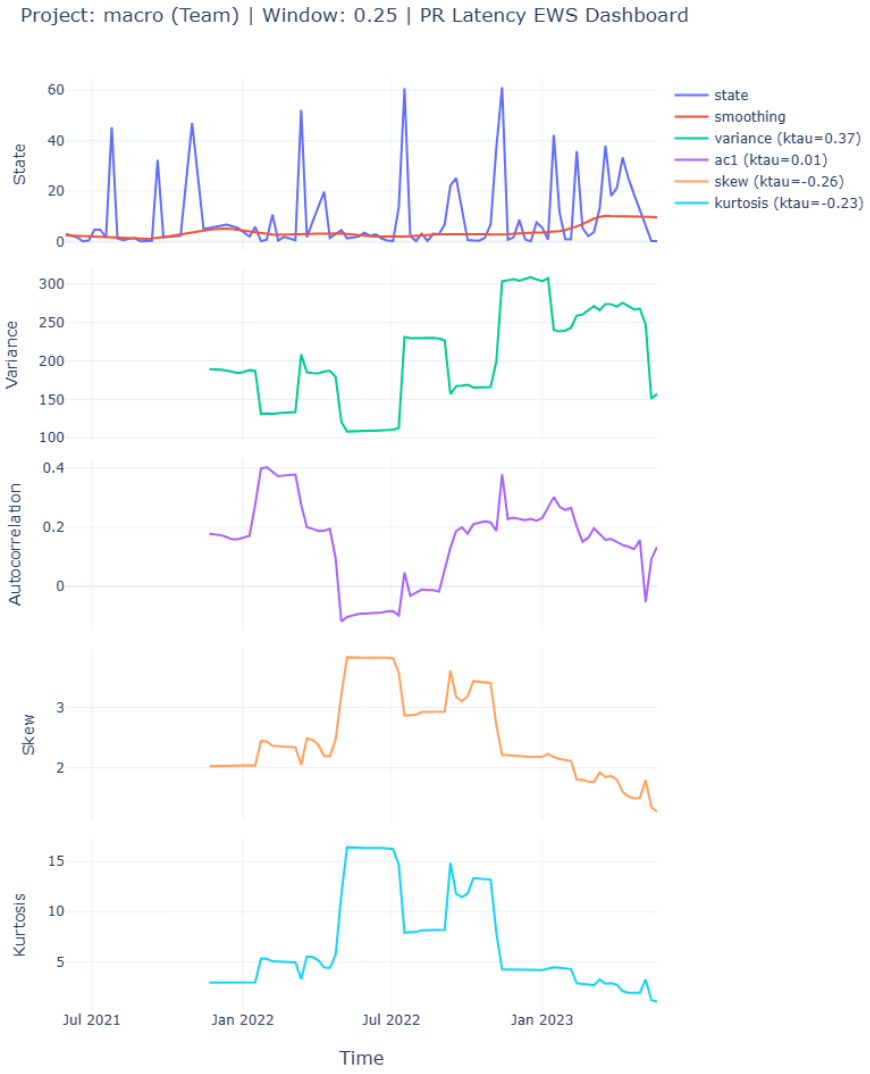

Figure B.17: EWS Graphs for Project with ID RP7 from the Rust Package Ecosystem Dataset (Latency Metric)

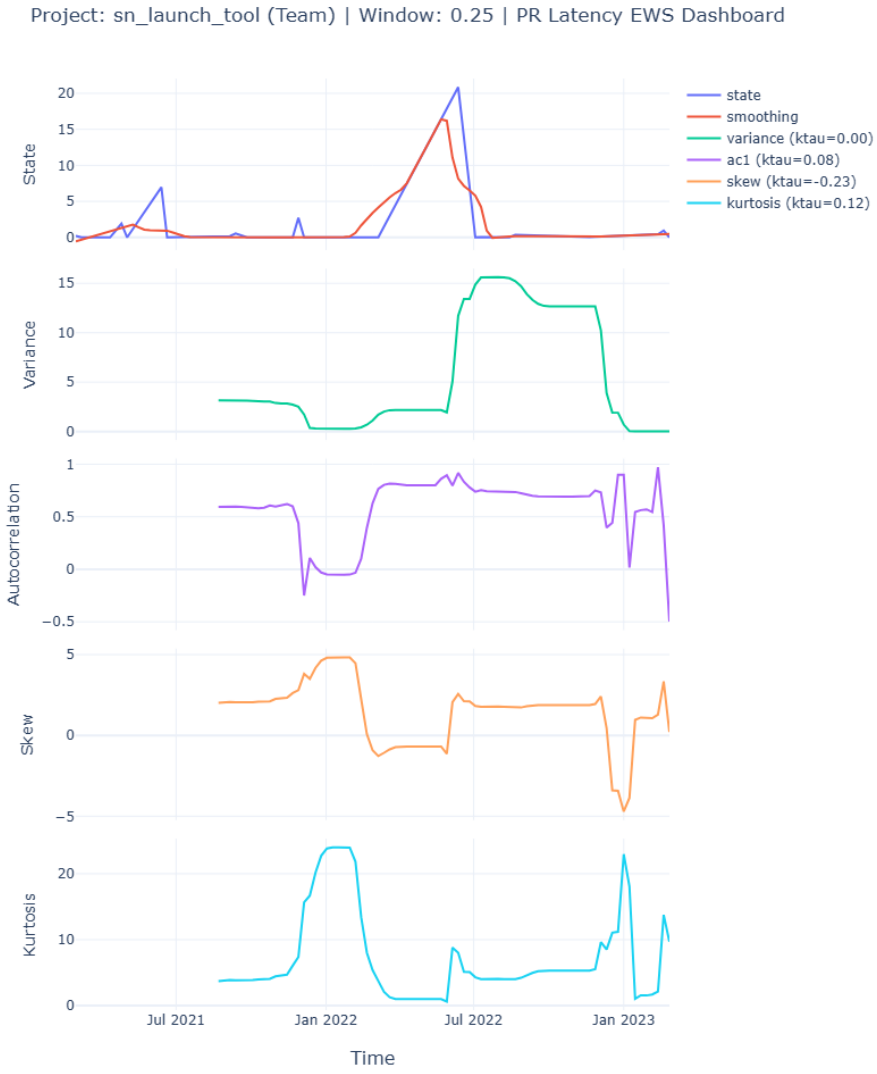

Figure B.18: EWS Graphs for Project with ID RP8 from the Rust Package Ecosystem Dataset (Latency Metric)

Figure B.19: EWS Graphs for Project with ID RP9 from the Rust Package Ecosystem Dataset (Latency Metric)

Figure B.20: EWS Graphs for Project with ID RP10 from the Rust Package Ecosystem Dataset (Latency Metric)

B.2.3 Survival Rate of GitHub Projects Activity Metric Plots

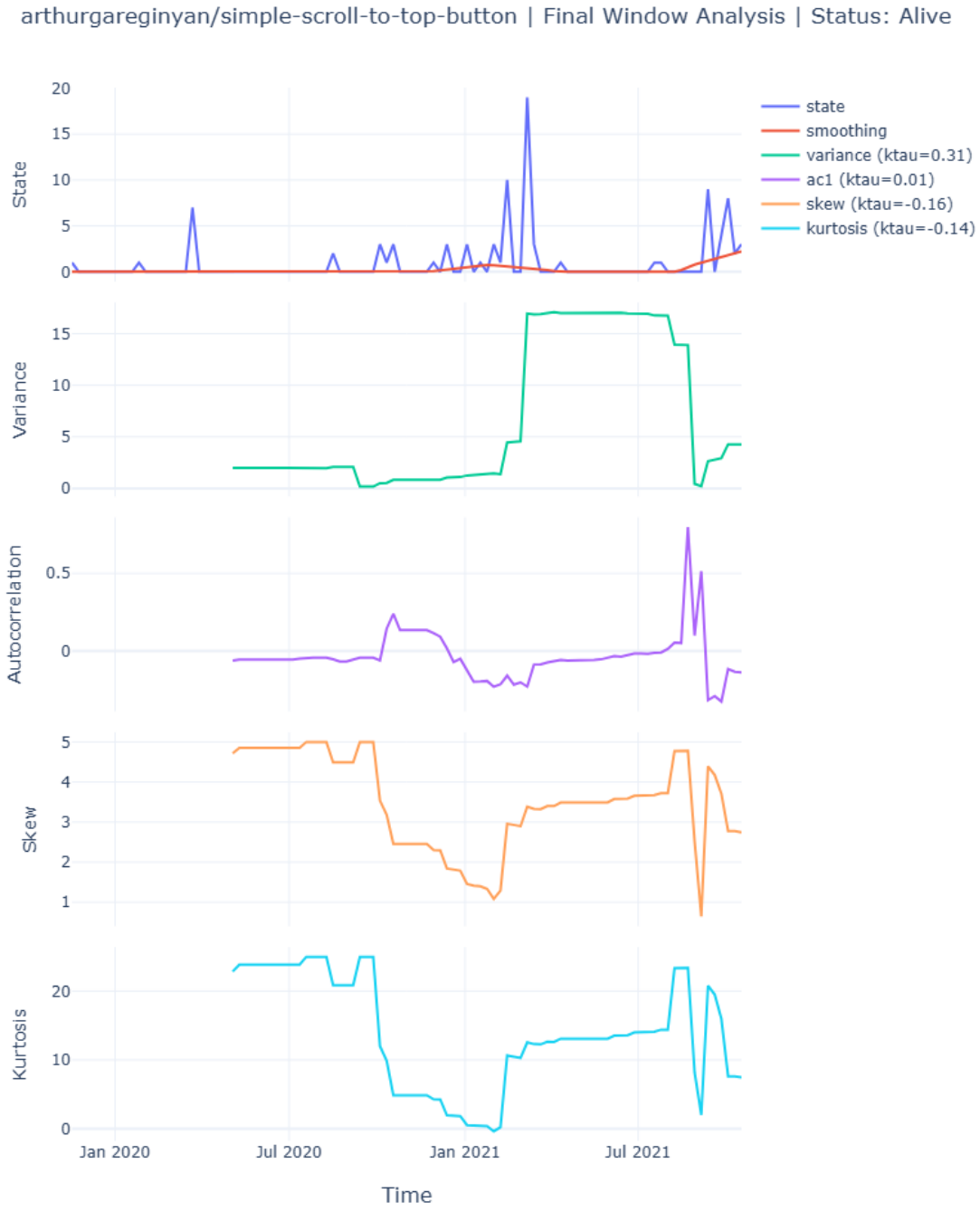

Figure B.21: EWS Graphs for Project with ID SP1 from the Survival Rate of GitHub Projects Dataset (Activity Metric)

Figure B.22: EWS Graphs for Project with ID SP2 from the Survival Rate of GitHub Projects Dataset (Activity Metric)

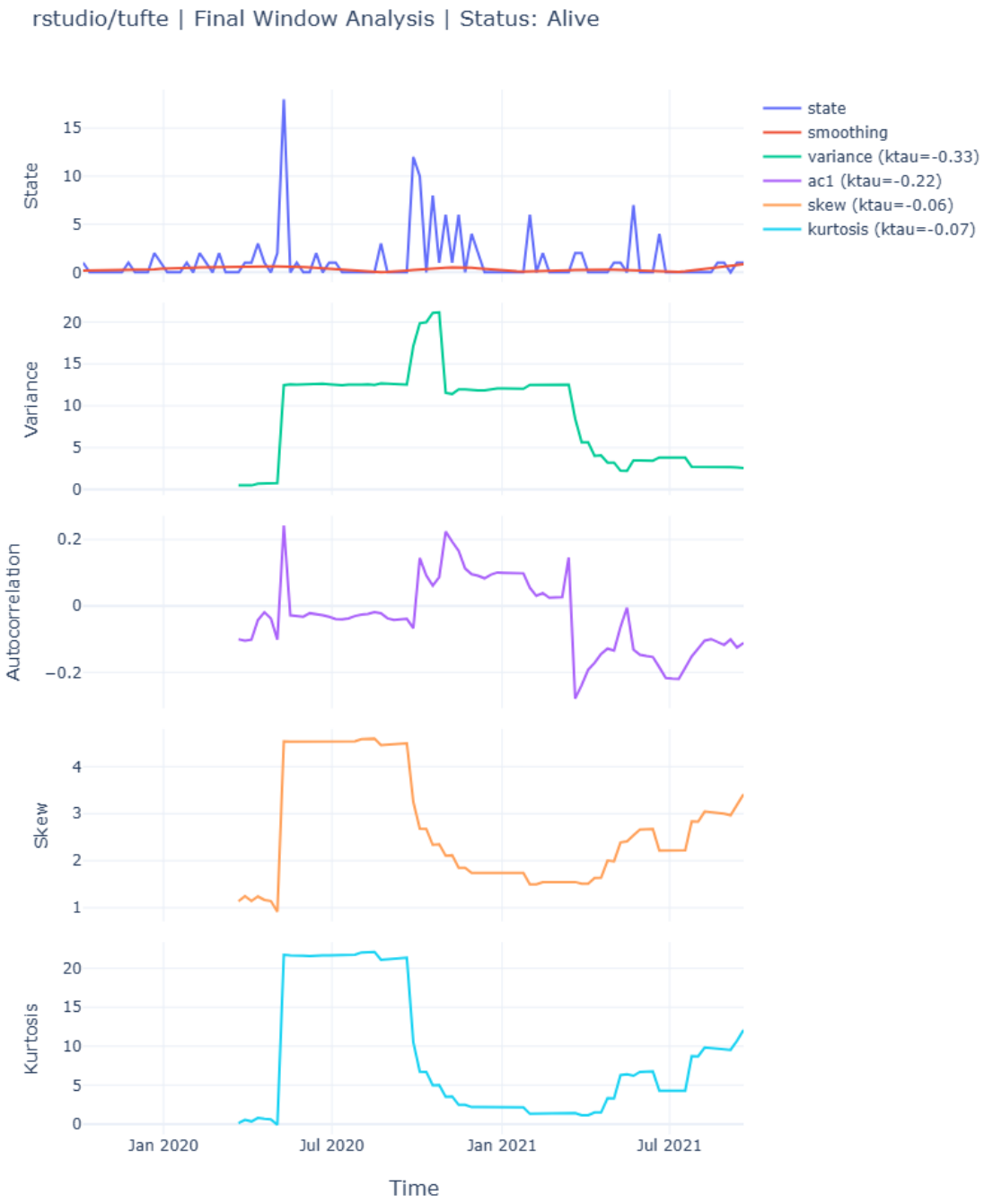

Figure B.23: EWS Graphs for Project with ID SP3 from the Survival Rate of GitHub Projects Dataset (Activity Metric)

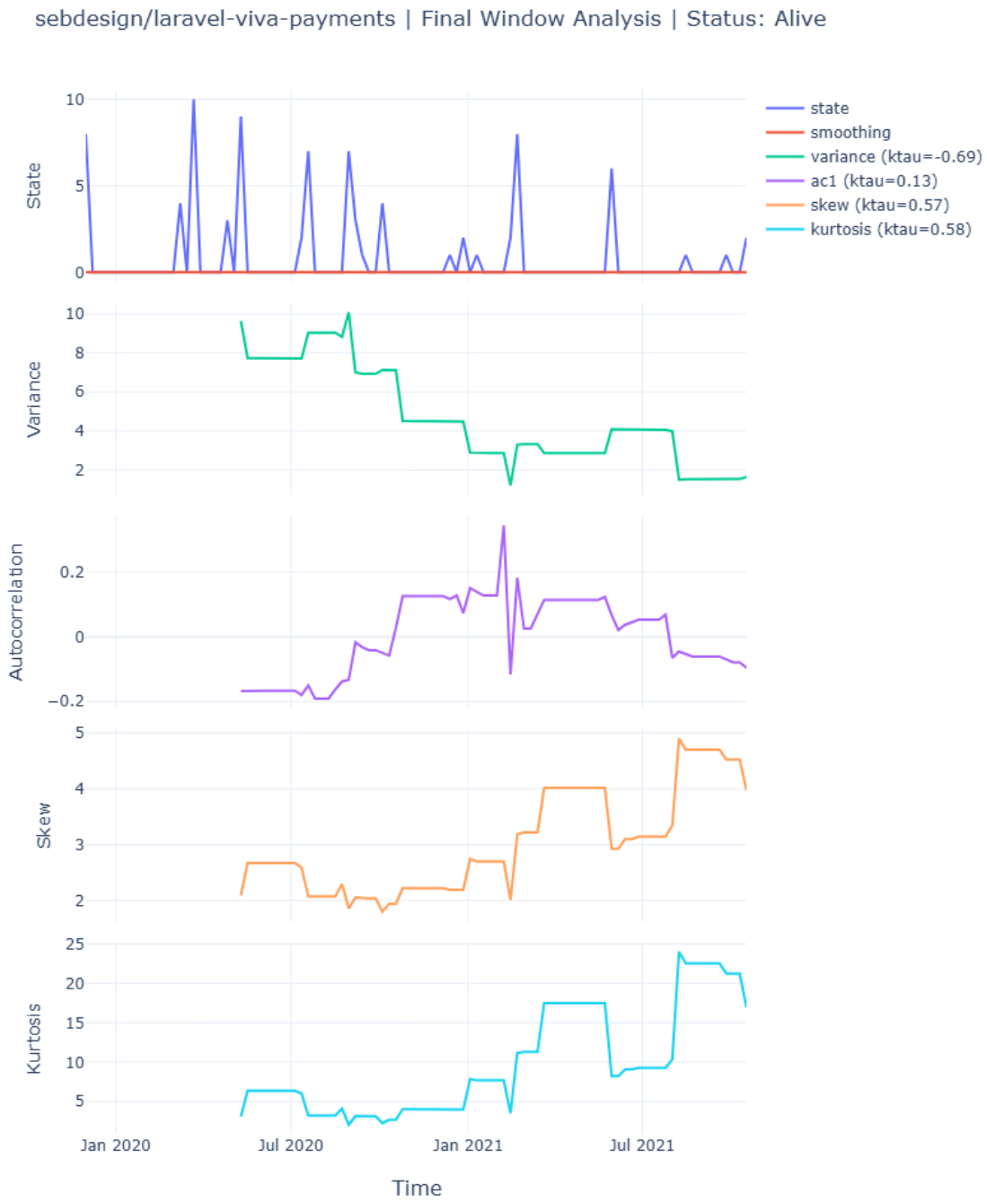

Figure B.24: EWS Graphs for Project with ID SP4 from the Survival Rate of GitHub Projects Dataset (Activity Metric)

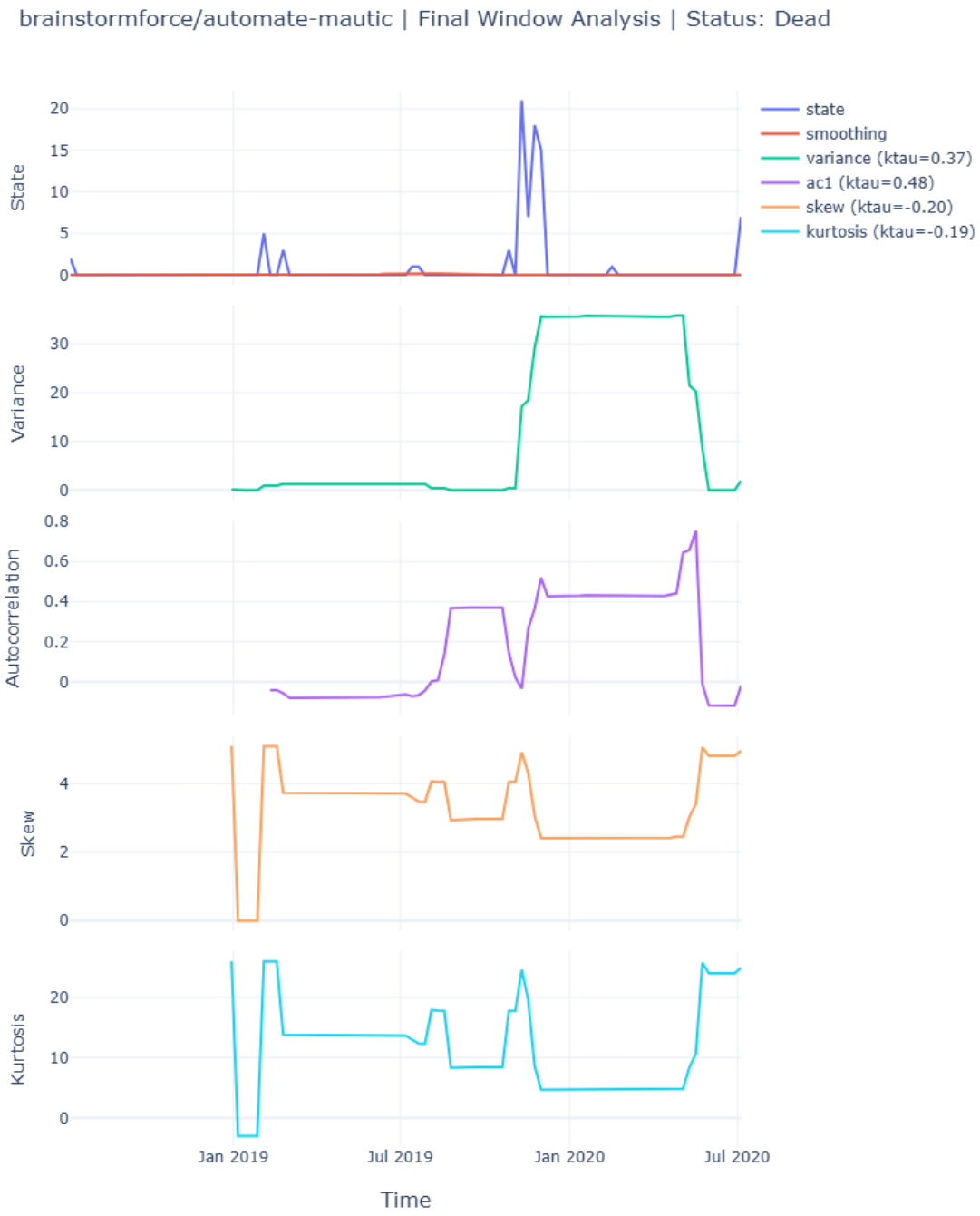

Figure B.25: EWS Graphs for Project with ID SP5 from the Survival Rate of GitHub Projects Dataset (Activity Metric)

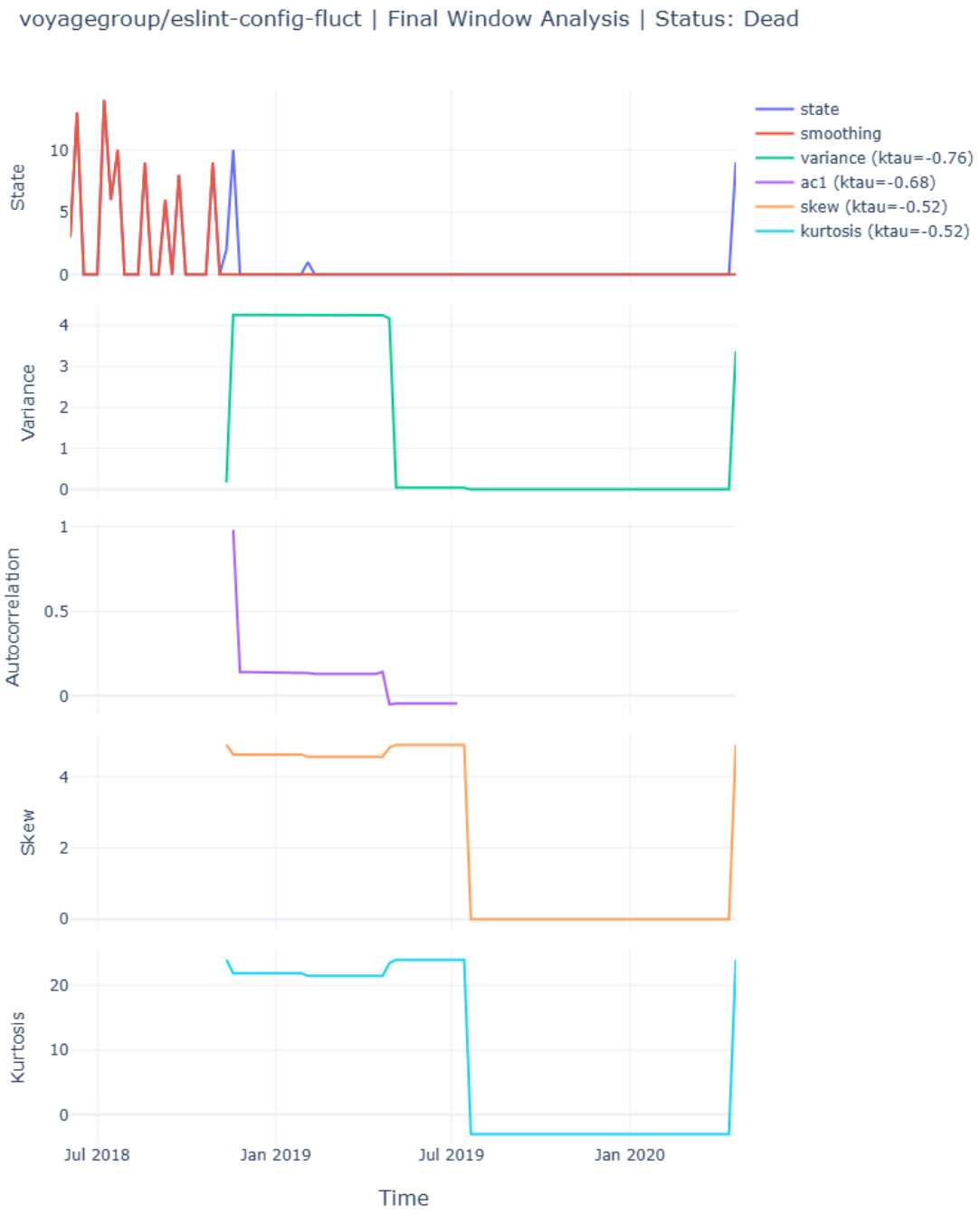

Figure B.26: EWS Graphs for Project with ID SP6 from the Survival Rate of GitHub Projects Dataset (Activity Metric)

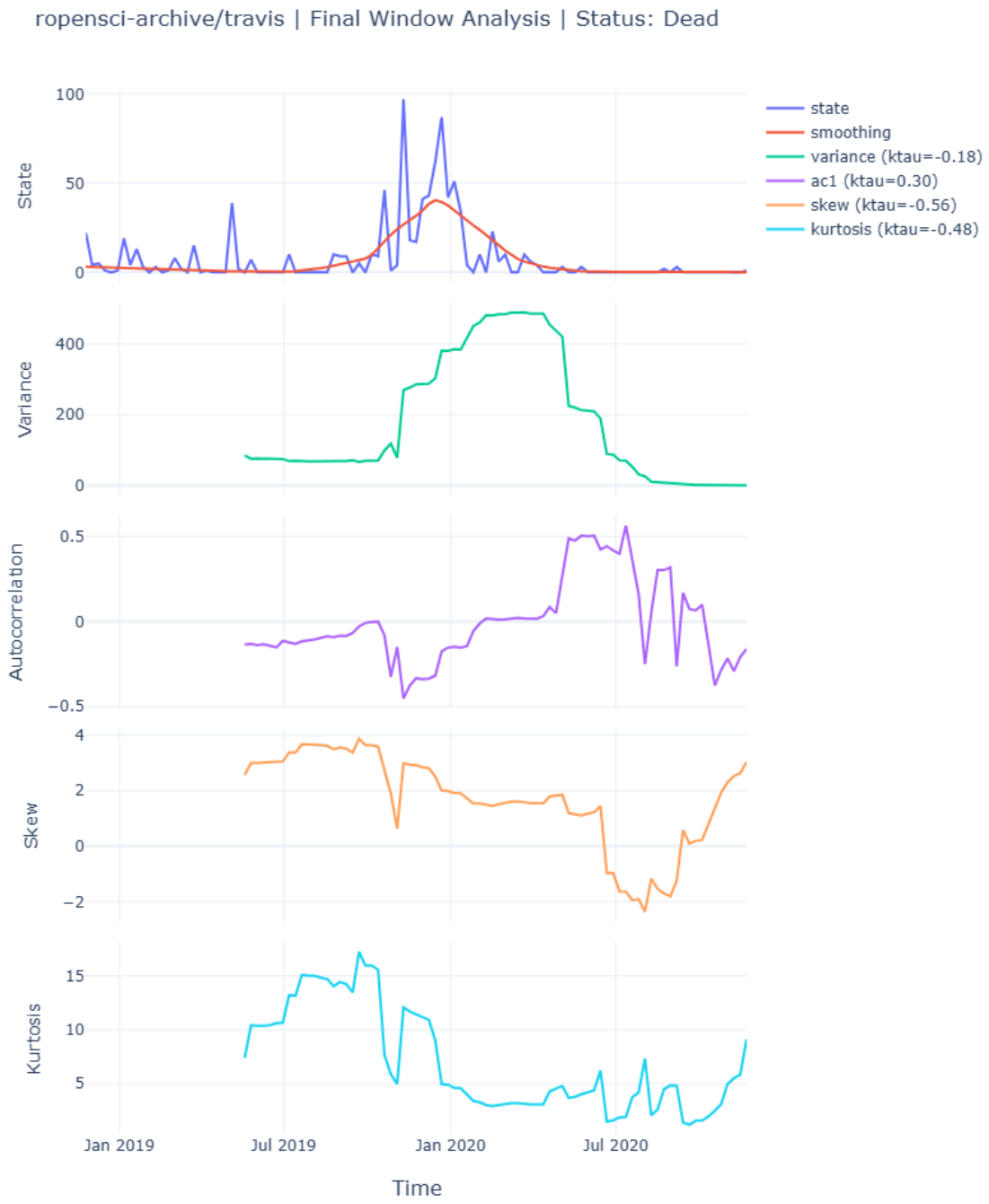

Figure B.27: EWS Graphs for Project with ID SP7 from the Survival Rate of GitHub Projects Dataset (Activity Metric)

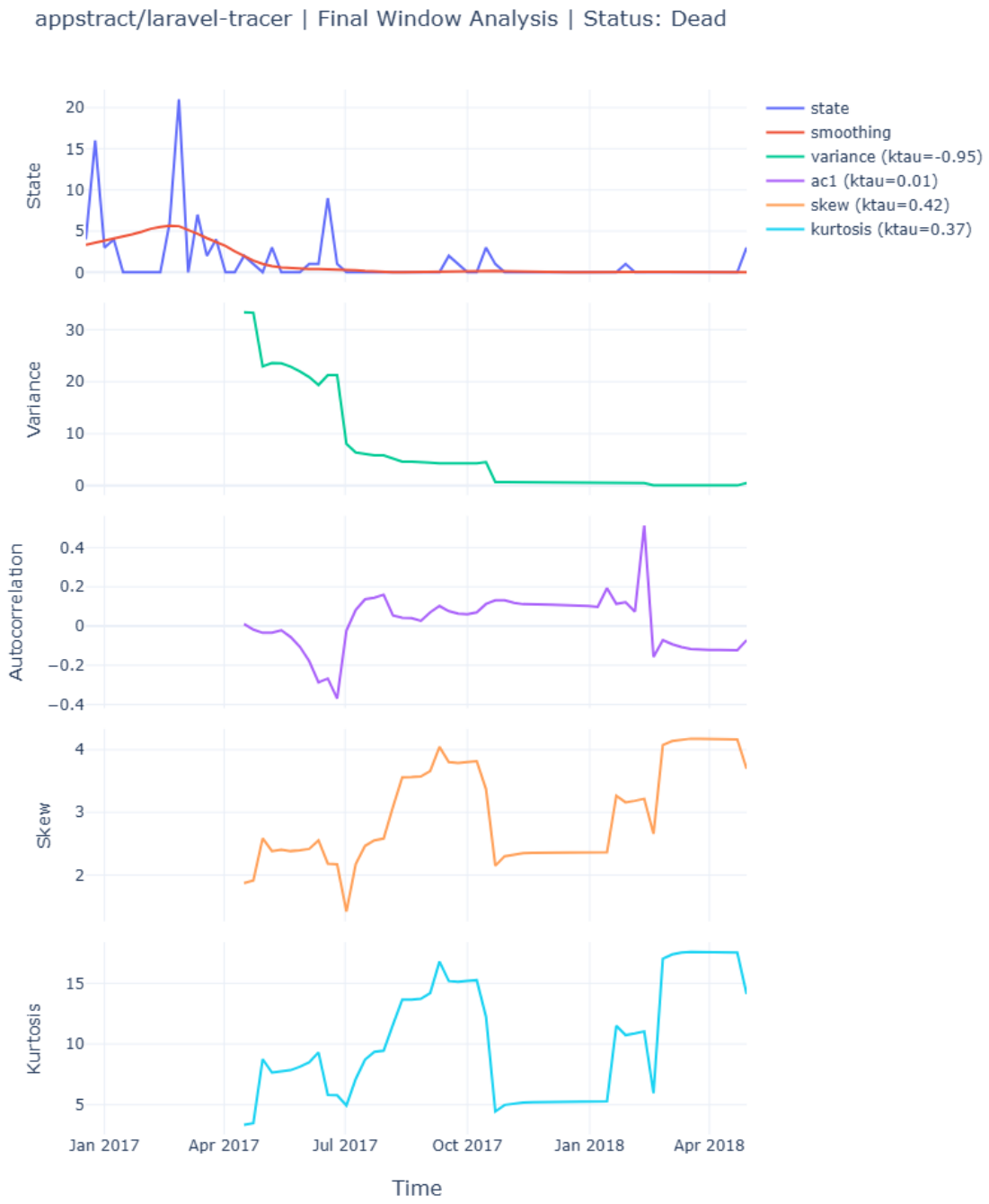

Figure B.28: EWS Graphs for Project with ID SP8 from the Survival Rate of GitHub Projects Dataset (Activity Metric)

B.2.4 On the Abandonment and Survival of Open Source Projects Commit Metric Plots

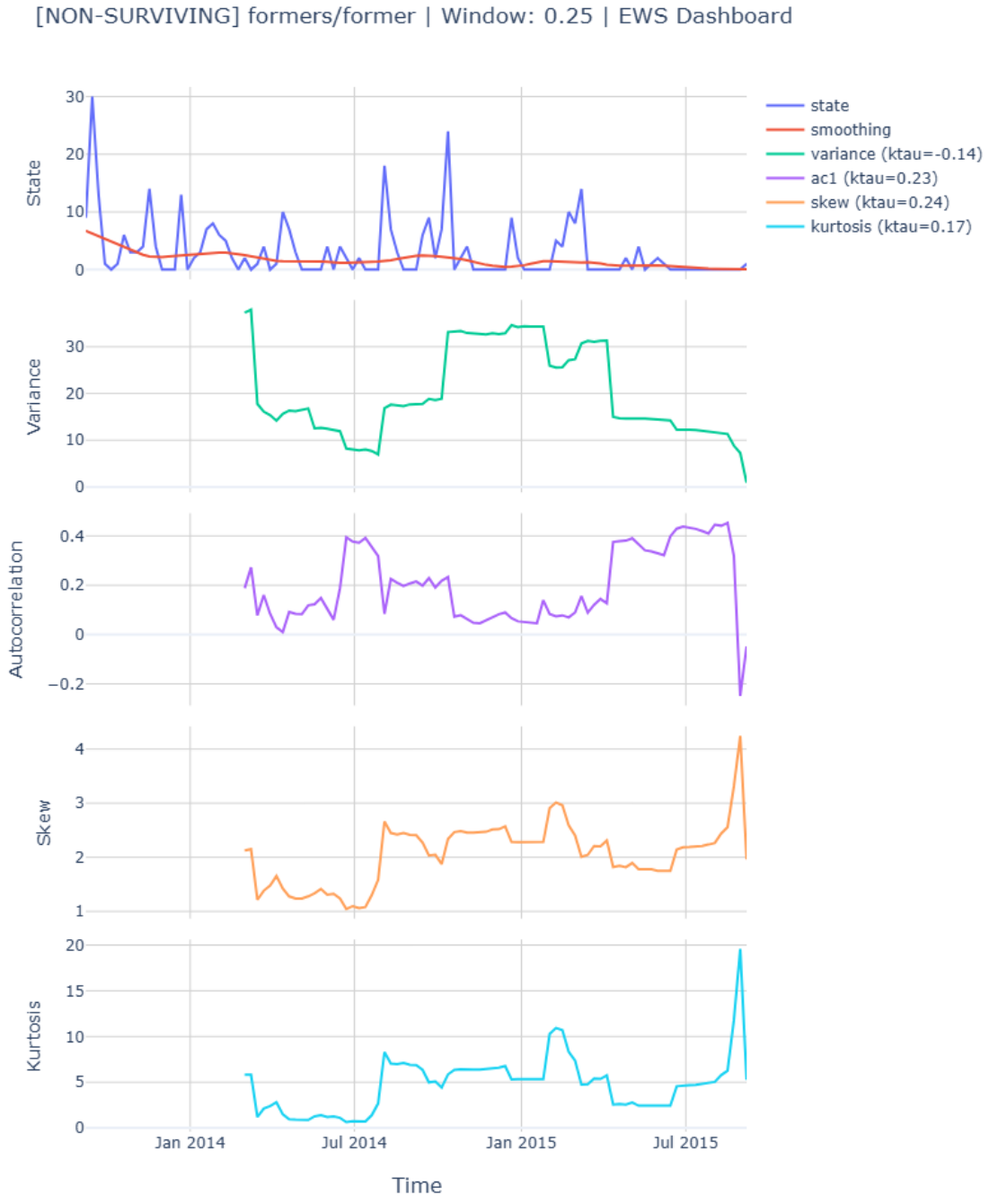

Figure B.29: EWS Graphs for Project with ID AP1 from On the abandonment and survival of open source project Dataset (Commit Metric)

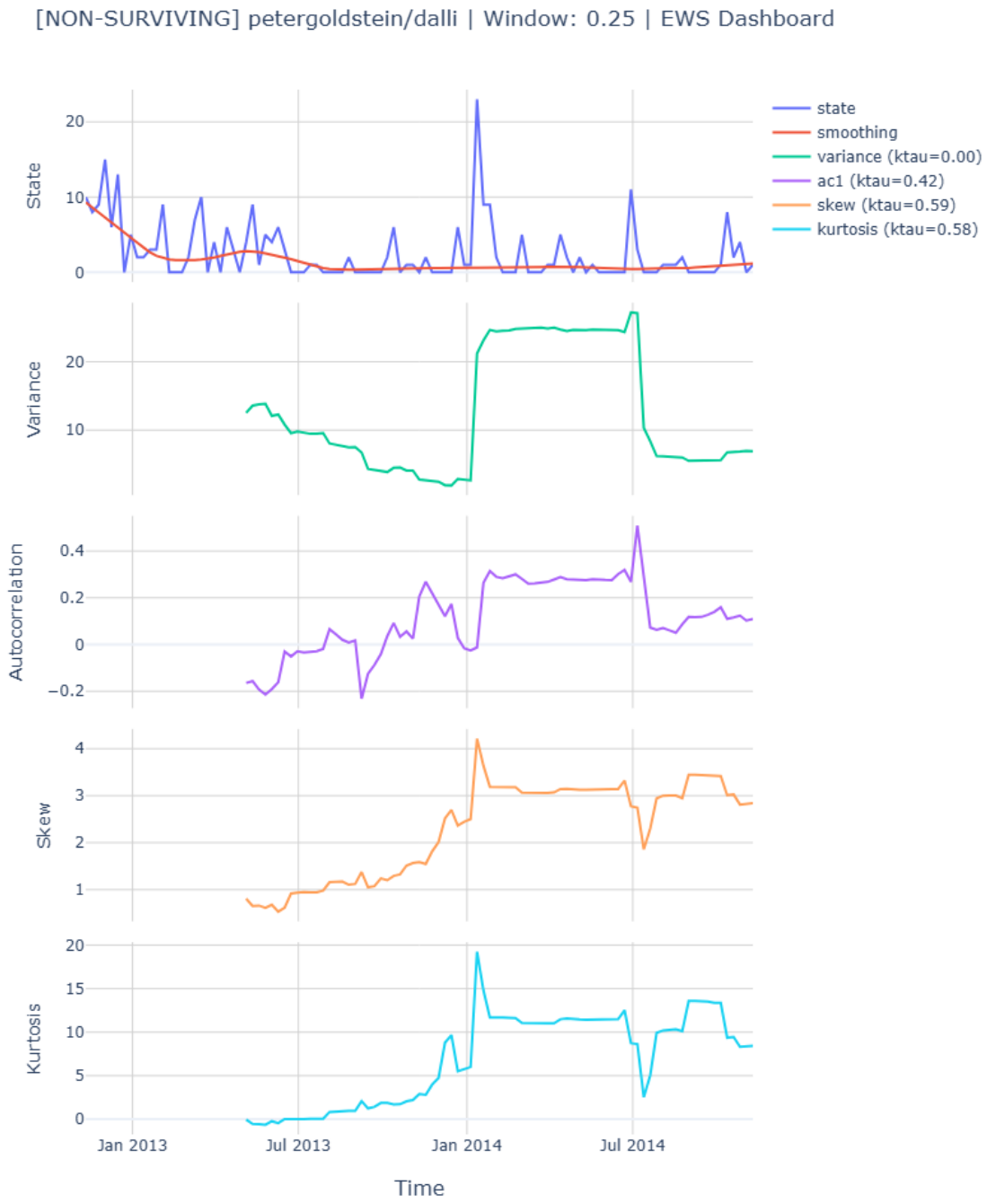

Figure B.30: EWS Graphs for Project with ID AP2 from On the abandonment and survival of open source project Dataset (Commit Metric)

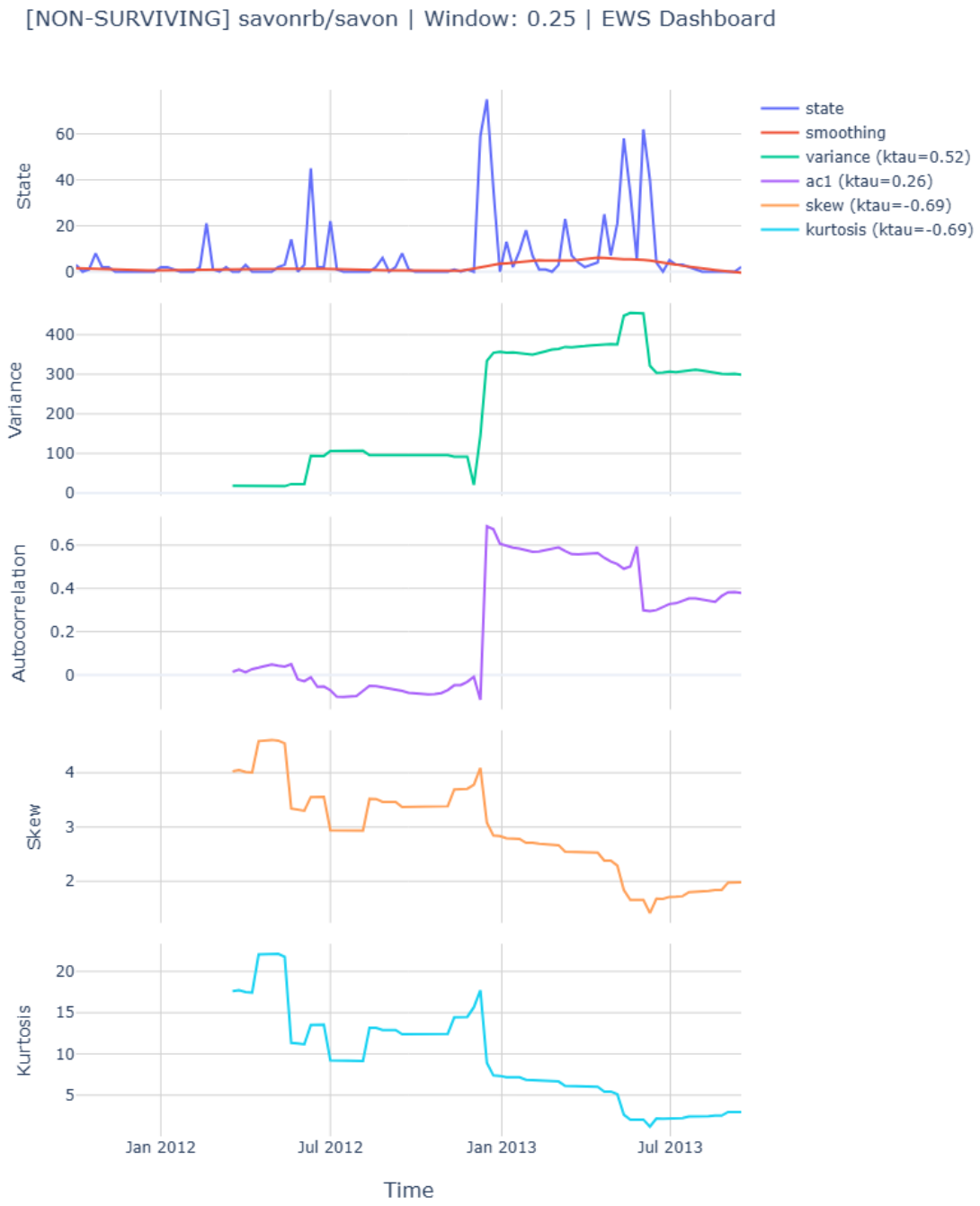

Figure B.31: EWS Graphs for Project with ID AP3 from On the abandonment and survival of open source project Dataset (Commit Metric)

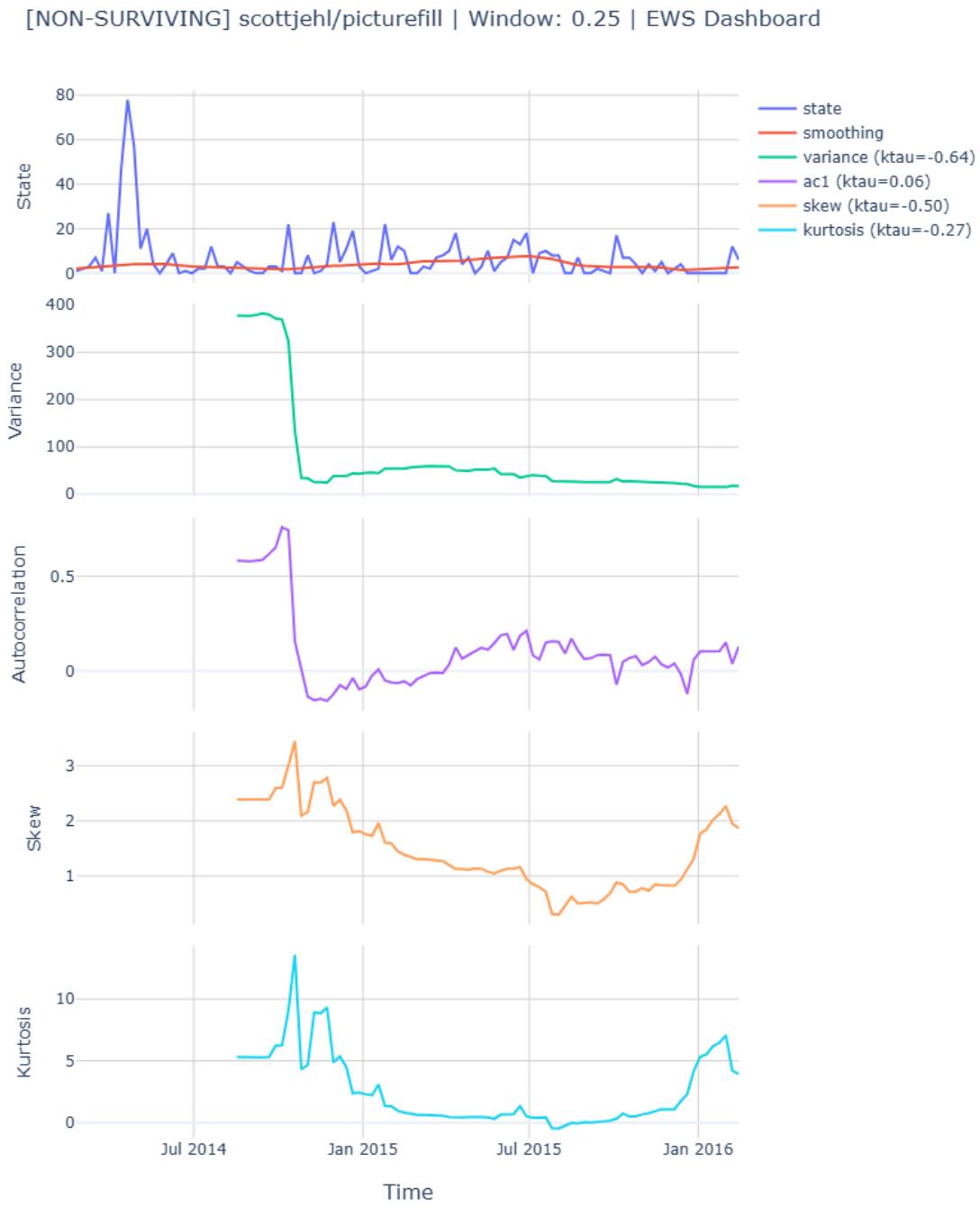

Figure B.32: EWS Graphs for Project with ID AP4 from On the abandonment and survival of open source project Dataset (Commit Metric)

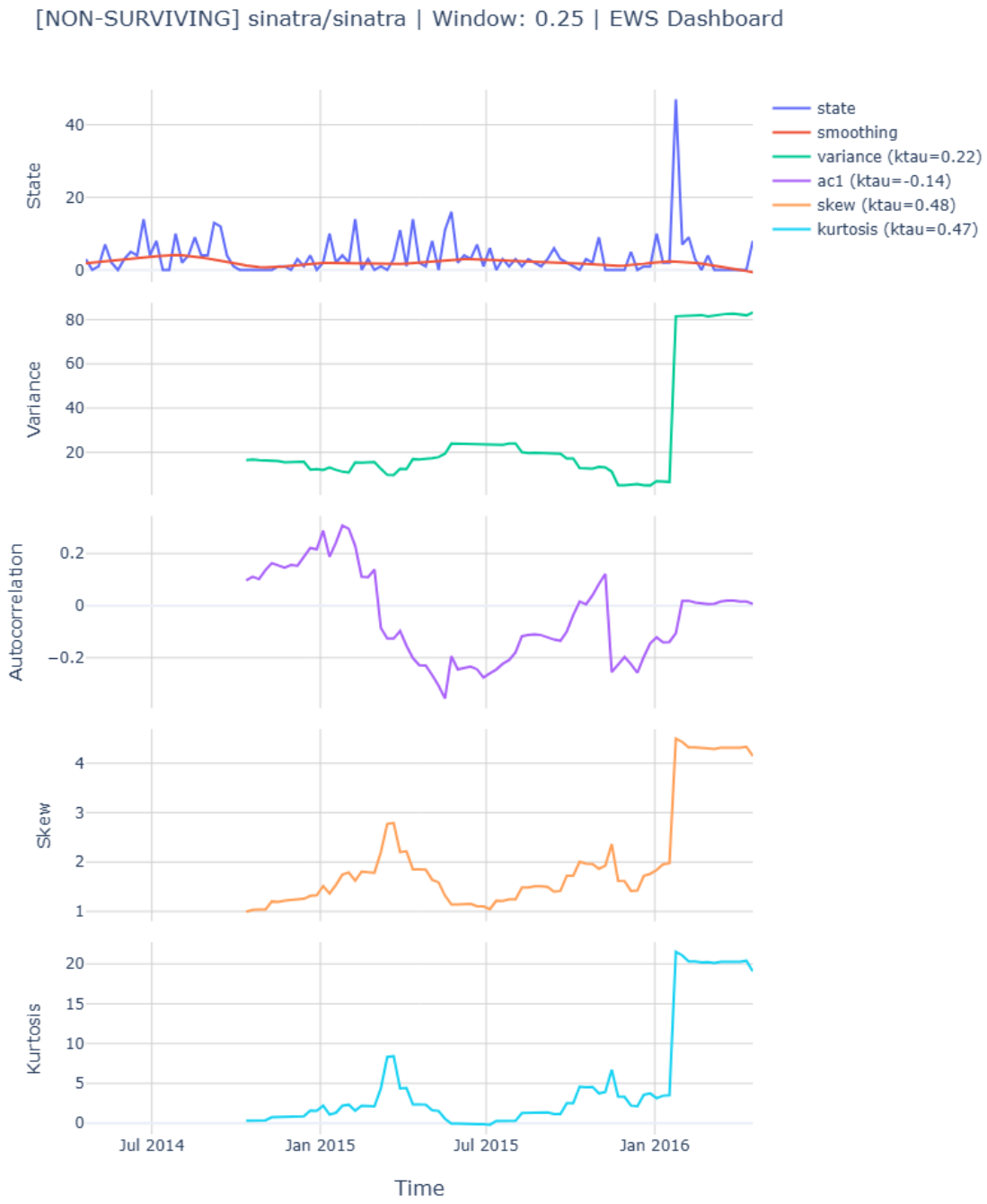

Figure B.33: EWS Graphs for Project with ID AP5 from On the abandonment and survival of open source project Dataset (Commit Metric)

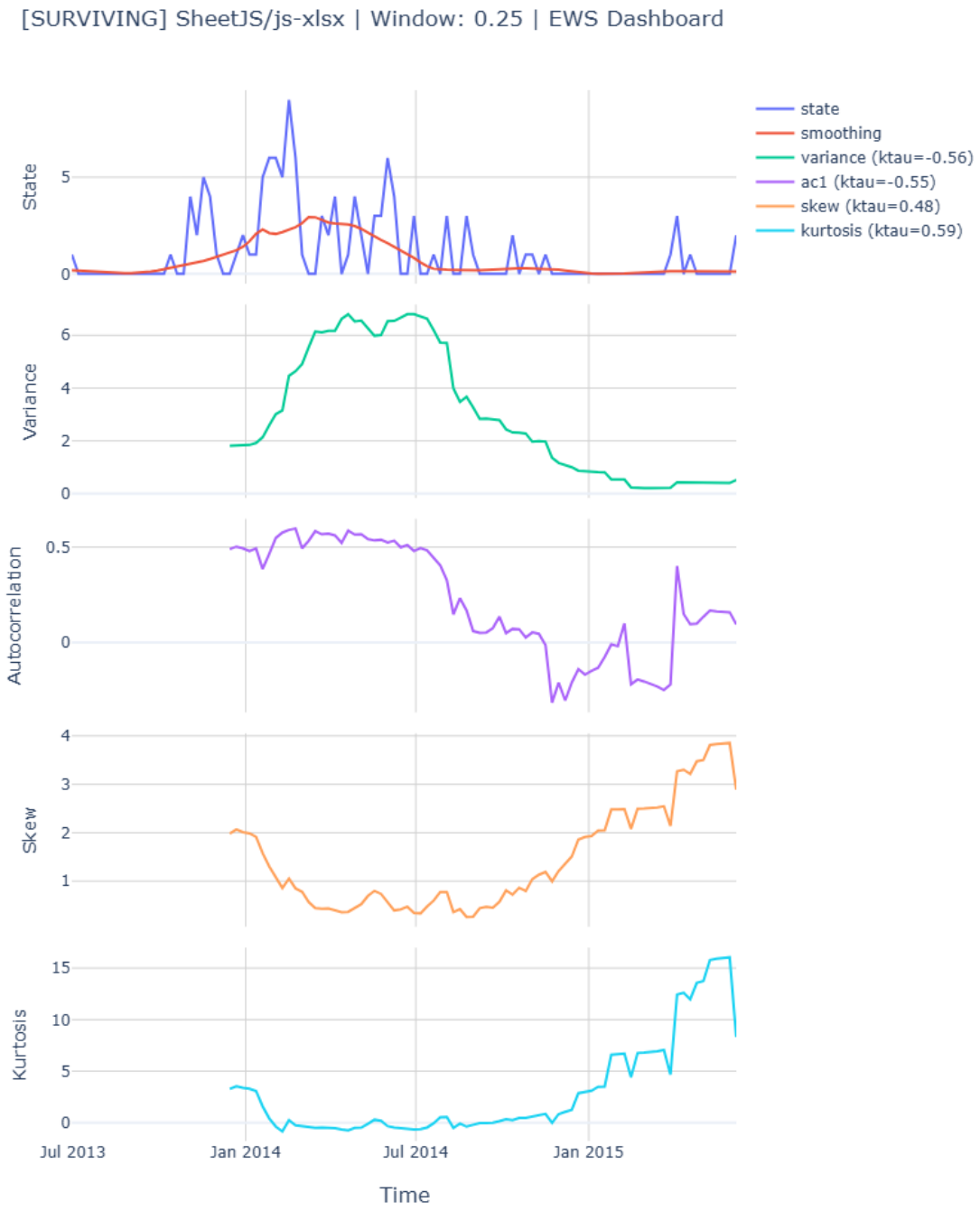

Figure B.34: EWS Graphs for Project with ID AP6 from On the abandonment and survival of open source project Dataset (Commit Metric)

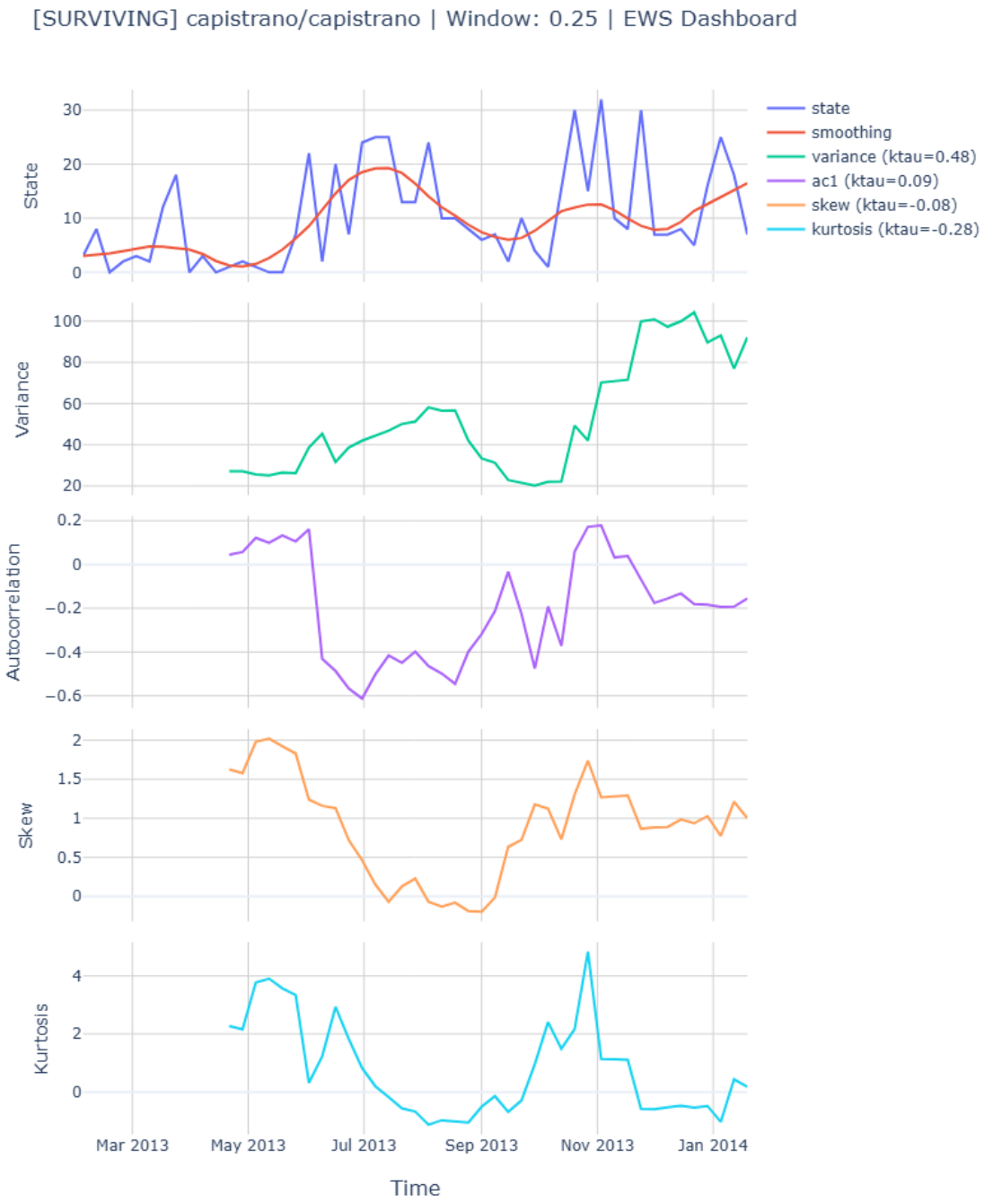

Figure B.35: EWS Graphs for Project with ID AP7 from On the abandonment and survival of open source project Dataset (Commit Metric)

Figure B.36: EWS Graphs for Project with ID AP8 from On the abandonment and survival of open source project Dataset (Commit Metric)

Figure B.37: EWS Graphs for Project with ID AP9 from On the abandonment and survival of open source project Dataset (Commit Metric)

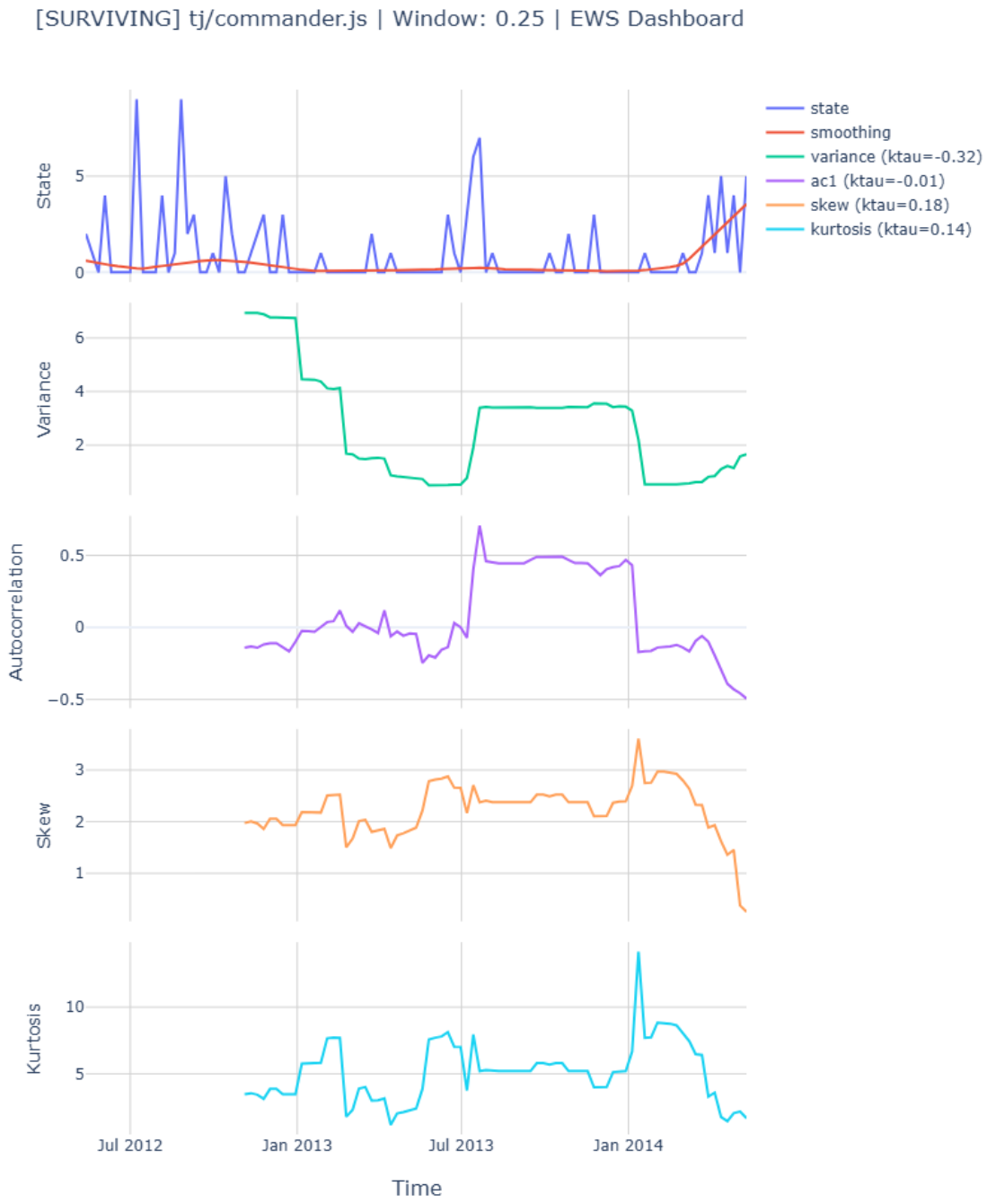

Figure B.38: EWS Graphs for Project with ID AP10 from On the abandonment and survival of open source project Dataset (Commit Metric)