# Accounting for socio-technical resilience in software engineering

Tamara Lopez
*School of Computing&Communications*
*The Open University*
Milton Keynes, UK
tamara.lopez@open.ac.uk

Helen Sharp
*School of Computing&Communications*
*The Open University*
Milton Keynes, UK
helen.sharp@open.ac.uk

Michel Wermelinger
*School of Computing & Communications*
*The Open University*
Milton Keynes, UK
michel.wermelinger@open.ac.uk

Melanie Langer
*Department of Psychology*
*Lancaster University*
Lancaster, UK
m.langer@lancaster.ac.uk

Mark Levine
*Department of Psychology*
*Lancaster University*
Lancaster, UK
m.levine@lancaster.ac.uk

Caroline Jay
*School of Engineering*
*Manchester University*
Manchester, UK
caroline.jay@manchester.ac.uk

Yijun Yu
*School of Computing&Communications*
*The Open University*
Milton Keynes, UK
yijun.yu@open.ac.uk

Bashar Nuseibeh
*The Open University*, Milton Keynes, UK &
*Lero*, Limerick, Ireland
bashar.nuseibeh@open.ac.uk

***Abstract*— Resilience engineering (RE) is most commonly applied at the organisational level, and has historically been associated with safety-critical industries such as nuclear, medical or aviation. This paper explores the application of RE frameworks within software engineering, and investigates resilient performance of the socio-technical system that supports the creation of software. We present a preliminary study based on a secondary analysis of data from previous ethnographic studies of commercial software practice. This analysis uses an RE framework devised for small team practice in safety critical settings. We present and discuss three salient episodes of software practice that illustrate the application of RE principles to software engineering, and suggest how this kind of analysis may benefit software engineering. We present challenges and opportunities based on our experience and propose future research directions.**

***Keywords— resilience engineering, change, ethnography***

## I. INTRODUCTION

Software development operates within an uncertain business environment, one that has been characterised as VUCA (volatile, uncertain, complex and ambiguous [11]). In the face of endemic change, resilience is key [39], and although determining the impact of change on software teams is critical for business it is also vital to support the well-being, and hence productivity, of software engineers [28].

The term "resilience" is often used in the context of software engineering but usually it refers to a technical quality. In contrast, our work takes a socio-technical perspective and focuses on resilient practices of individuals and small teams tasked with creating software, an activity where social factors are particularly key [4]. To do this we draw on methods and techniques associated with resilience engineering [15] (RE), a field that regards resilient performance as inherently socio-technical. RE principles have been applied in some areas of software engineering, such as error handling [22] and outages in internet-facing systems [10], but not on software creation. The paper has two aims:

(1) To illustrate whether and how concepts from RE may be applied to data from software development practice to characterise resilient performance. This is achieved through a secondary analysis of ethnographic data sets using an RE framework designed for use in a small team context. Three salient episodes from this data that have the potential to demonstrate resilient performance are presented and discussed.

(2) To explore the potential benefits and challenges of performing such an analysis. This is achieved through a set of reflections on the experience of applying RE to ethnographic data on software practice. We conclude that applying RE principles to software creation presents opportunities but is not straightforward.

Section II outlines literature that defines resilience within software engineering and introduces resilience engineering in safety science. Section III describes the preliminary study conducted to meet our first aim. Section IV addresses the paper's second aim by presenting a series of reflections and future directions. Section V concludes the paper.

## II. BACKGROUND

### A. Resilience in Software Engineering

Resilience is often characterised within software engineering in terms of fault-tolerance, i.e. the ability for technical systems to continue to operate in the event of component failure. This focus on managing an undesirable outcome means that the phenomena contributing to the failure of technical systems are often ignored [20]. However, with the growth in scale and complexity of systems, Laprie argued that resilience is increasingly aligned with the expectation that software systems will remain dependable in the face of continuous changes [20]. These changes have both social and technical aspects and are dynamic: they can be functional, environmental, involve hardware and software, be foreseen or unforeseen, and may manifest in the short term, for example, during adaptations made within software as it runs, or in the long-term, as in the combination and recombination of existing systems of hardware and software [20].

The growing recognition in software engineering that dependability must account for the interplay between social and technical aspects raises questions about the role of human activity in keeping systems resilient. This has been explored within software engineering in terms of how developers detect

This research was supported by UKRI/EPSRC EP/T017465/1, EP/R013144/1), and SFI (13/RC/2094_P2).

and recover from errors [23]. Findings in this work expand the conceptual framework for error handling to include situated problem solving [23] and establish a connection between human error and professional growth [22].

Human activity has also been explored in the context of business-critical systems, where events that threaten operational status are conceptualized as incidents [6] or anomalies [10]. Cook [6] describes interventions taken to resolve undesirable system behaviour in these contexts as happening "above the line" of code and other technical artefacts. Problem solving hinges on inferences or hypotheses [10] based on representations that appear on screens and displays, and draws on mental models formed over time of how underlying parts of the system work [6]. The professionals that intervene in such problems include engineers who design and write code, people who build and deploy the code, and operations teams that monitor technical systems in operation. The argument for resilience made within this area of software engineering is that it is the adaptive capacity of people, their cognitive activity, and their ability to coordinate with one another during incidents that resolves operational problems before they cause service outages [39].

Although software operations teams and error handling have been viewed through a socio-technical lens, the activities of software design and creation have not been examined for evidence of resilient performance.

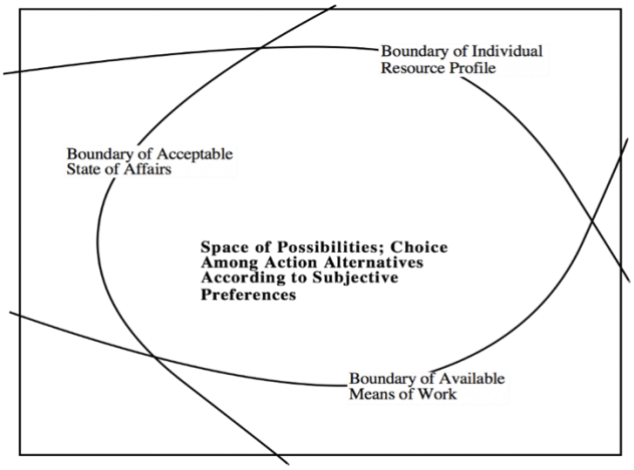


Fig. 1. Rasmussen's space of possibilities [29].

### B. Resilience Engineering in Safety Science

Within RE, resilience is regarded as socio-technical; it works on three levels: operationally in the individuals and teams that work on tasks; in organisational efforts to coordinate, support and manage operations; and within the industrial system which designs and produces technologies that are used in work [25]. RE, which emerged out of studies in safety science, provides techniques and frameworks for documenting and understanding how organisations and individuals learn about, monitor and respond to changes, disturbances, or opportunities in everyday situations. According to RE, the potential for resilience [9, 13] lies within the recognition and promotion of these capabilities, rather than in managing risk.

At the task level, workers are said to contribute to resilient performance through adaptations that meet immediate situational demands [29], and compensation mechanisms that address imperfect circumstances [9]. Within the space of possibilities (Fig. 1), these adjustments are self-directed, but are constrained by individual capacity, the plans, policies, and cultural norms for doing things within organisations and professions, and available resources (including technologies).

Resilience manifests through the combination of plans and processes put into place to make systems work, plus the performance of workers in the moment [40]. These two are in tension; some actions taken by workers to meet situational demands result in errors or failure, or break policies or procedures for how work "should" be done. However, such adjustments are recognised within resilience engineering to be acceptable or even necessary as a part of meeting commitments to safety or other goals for resilient performance [36]. In software engineering, examples of adaptation and compensation could include the way agile development practices are implemented within an organisation or reflect covert [40] or shadow tactics [17] that come into use when organizational security policies intersect with demands for engineering productivity [21].

### C. Applying Resilience Engineering

The elements of a RE analysis [13] include understanding how work is done, identifying indicators of resilient performance within that practice, knowing the goals for the future status of the system, and determining how resilient performance can be maintained when changes are made. Analyses centre around events that include instances of adaptation or compensation, and are guided by an understanding of what "resilient performance" means in this work context. For example resilient performance is often defined in relation to performance goals [39].

RE frameworks have been developed to model or represent aspects of resilience at the functional level [14] and can be used to assess organisational resilience [13]. Furniss et al. [9] created a framework for systematically identifying socio-technical resilient performance at the small team level. Applying this framework involves identifying and describing details of episodes observed within team-based situations that exhibit the potential for resilient performance, and comparing instances of the strategies used with findings in RE and safety studies in other domains. To gain insight into how these strategies are leveraged, the descriptions are structured to capture adaptations or compensations, and contextual elements of the socio-technical system that influence the actions taken: the resources and enabling conditions; whether the instance presents a systemic vulnerability, threat or opportunity for improvement of the system; and the mode or state of the system at the time the instance occurred. The outputs of analysis represent work as it is done and along with strategies, form repertoires that indicate the potential for resilient performance in the wider socio-technical system [9].

## III. A PRELIMINARY STUDY INVESTIGATING SOCIO-TECHNICAL RESILIENCE IN SOFTWARE ENGINEERING PRACTICE

This preliminary study examines software creation and focuses on individuals and small teams by applying the small team framework described above [9]. For this study, the goal we adopt for resilient performance is to resolve problems so that software development can progress. Although productivity and efficiency are commonly identified as key performance areas for software development, "making progress" is an aspect of work that developers value, as widely reflected in trade [1] and research literatures examining motivation [5] and satisfaction [8] and perceptions of success

[37]. It has also been recognised more widely as important to knowledge workers[2].

### A. Secondary analysis of ethnographic data

To perform a secondary analysis, we examined previously collected data from three sets of ethnographic studies performed by the authors. Set 1 focused on five commercial agile software development teams, were conducted between 2003 and 2019 and were analysed using distributed cognition to understand collaboration and information flow in software teams, e.g. [7, 33]. Set 2 was collected between 2010 and 2013 and used thematic analysis to identify how developers handle errors that arise during software development [22, 23]. Set 3 was a multi-sited ethnographic study that used a range of analysis approaches to examine security in software development; it was conducted between 2017 and 2019 [21]. Although not collected with RE analysis in mind, the nature of the data collected, and prior analysis undertaken was compatible with performing an RE analysis. For example, the studies were ethnographic and hence aimed to surface the participants' perspectives (referred to as "informants" in ethnographic studies); they focused on day-to-day practice of individuals and teams; and the data was available for further analysis.

Our approach to analysis consisted of three steps. In each, the ethnographic stance was maintained in order to reflect the informant's perspective on activities and conditions:

1. Using principles from error handling [22], identify episodes where our informants experienced an interruption or change in their workflow that required problem solving. These principles originally focused on individuals; to broaden the scope to include teams, episodes were sought in which interruptions originated inside an individual or team's own space of activity or within the wider organisation.
2. Analyse and document activity within each episode. The first two authors re-read transcripts, fieldnotes and prior reflections from the studies listed above. Virtual boards were used to group data from each episode into categories according to the structured descriptions from Furniss' framework [9].
3. Using our definition of resilient performance, we assessed how each episode demonstrated principles of resilience engineering or signified an example of resilient performance. First, strategies employed by our informants were identified and compared with previous literature [12, 19, 24, 26]. Next, we considered the mechanisms underlying the strategies [9], including threats and opportunities, alternative actions, and possible consequences for each episode.

The outcome from this analysis was a set of episode descriptions that followed Furniss' formulation and reflected resilience engineering principles. The episodes were identified and analysed by the first two authors. Findings and conclusions were discussed with the wider team.

### B. Findings

The analysis aimed to characterize episodes that illustrate the application of the RE framework rather than to be comprehensive. We identified several episodes from the ethnographic data but here we report one episode from each data set (Table 1). The following episode descriptions use RE terminology and are presented according to the structure from Furniss' framework [9]: RE terms are highlighted in bold, and each description starts with a narrative of the episode itself, then discusses its potential for resilient performance and the strategies used within the adaptation or compensation.

TABLE I. THREE ILLUSTRATIVE EPISODES

| ***Episode*** | ***Brief Description*** | ***Adaptation/Compensation (A)/(C)*** |
|---|---|---|
| Breaking the estimate (set 1) | During a sprint the developer recognises that the estimate is inaccurate and interrupts his workflow to investigate. He discovers that he is implementing a different design than the one estimated | C: accept new estimate,<br>A: new design |
| Rolling back code (set 2) | After manually copying a changed file to a testing server, a service fails. The developer realises that he copied the file to the wrong folder, overwriting working code. | C: walk the code back, leaving the change until a later time |
| Creating a new definition of done (set 3) | The team fails to complete agreed tasks within sprints due to unplanned work requests as clients call them directly with bugs or feature requests | A: create a different definition of done |

#### 1) Episode 1: Breaking the estimate

The planning game involves estimating the effort required for each user story, which in turn requires decisions on how to implement a solution. However, in our episode, the design used as the basis for estimating was not documented and a different design was being implemented, which compromised the estimate. In this case, we **observed** that a story card was a **resource** that helped the developer realise the story was taking significantly longer to implement than the estimate given on the story card. The estimate on the card was an early indicator or hint of a potential problem [38]. We relate this recognition of a problem to the **strategy** "planning-based detection" [19]. We also **observed** the developer speaking with the teammate who wrote the story card to understand why this may have happened, suggesting the use of the **strategy** "anticipates weaknesses in plans and identifies information need" [19]. The **vulnerability** was lack of documentation for the intended design and the **threats** were that a different design might weaken the code or an extended estimate may inhibit the team's progress. Exploring a different design and sharing an understanding about the software and its requirements presented an **opportunity**, and the willingness of the teammates to engage in discussion with one another suggest the "flexible approach to planning" [19] strategy. After discussing the situation with the teammate, the developer decided to carry on implementing their own design [34].

This episode characterizes resilient performance that is **enabled** through top-down plans and bottom-up efforts [40], environmental resources and conditions [9], and demonstrates that situations within the **mode** of normal operations can involve both adaptation and compensation. Within this environment it was acceptable for the developer to question the estimate given on the story card and to speak with the teammate. By following the signal [10] on the story card, the developers were able to align perspective with one another and to augment their own experience [22].

*2) Episode 2: Rolling back code*

The developer's team maintains a product name and an ID service for their department. The services were implemented in the same file, but each has its own directory on the server. The developer was given the task to change a service so that it would reference a different database. To deploy the changed ID service, the developer had to manually copy the file into the proper folder on a testing server. We **observed** that after doing this the developer realised that the changed file had been copied into the wrong folder, overwriting the product naming service and causing it to fail. We relate this to the **strategy** "outcome based-detection" [19]. This broke an internal commitment made between the development teams to always keep services running, indicating a **vulnerability** and marking a transition in the **mode** from normal to critical practice. The developer tried several things to fix the broken service. First, the broken service could be restored from a back-up, however all backups had been deleted through an automated process. Second, the code could be redeployed for both services. Unfortunately, changes to the ID service impacted the product naming service and refactoring the naming service was out of scope for the user story. Ultimately the developer tried to alter the build to deploy the older version of the broken product naming service. However, the developer did not have permission to alter builds, and the service was not operational overnight. In the end, the developer's team decided to **compensate** by abandoning the task for the sprint, rolling back all changes and redeploying both services. This restored access to the product naming service for the department [23].

This resilience episode characterises the cyclical nature of error handling within software development [22], that includes the interplay within episodes of awareness, planning, and outcome-based **strategies** [19]. In this episode, the problem was detected based on outcomes, in which the developer used the **strategy** of "examining relational and temporal patterns of changes" [19]. However, the fuller data set [23] indicates that while attempting to bring the service back up, the developer also used "loose plans to gain flexibility" [19], and used awareness of the system to "detect missing cues" and find "hidden assumptions" [19]. The episode illustrates **brittleness** within a system [40], showing how constraints on resources and enabling conditions [9] can negatively influence the strategies an individual is able to use.

*3) Episode 3: Creating a new definition of done*

We **observed** that one team in a large workforce management software company has created its own definition of "done" and pinned the definition above the team Kanban board in a public space. The team tailors a reporting system that clients of the company's software use to develop intelligence about workflow. The team is also responsible for managing bugs within the broader software suite—to help ensure that clients can maintain "business as usual". The nature of the work in the reporting system and the commitment to handle bugs results in many calls or requests for changes that come directly to individual team members. This is a **threat** to making progress for the team, as they cannot always complete tasks for sprints. This causes **stress**, and the lack of progress against agreed tasks is perceived by team members to impact standing within the department. Tailoring the definition of done to respond to this **vulnerability** is a strategy that reorients the view of "making progress" to account for circumstances particular to the team. This relates to "managing workload" (individual strategy) or "workload distribution management" (joint strategy) [24] and "willingness to relax efficiency temporarily" [12].

Within this resilience episode, the team is **enabled** by recognised agile practices and their local adoption to create an individual definition of done. This exemplifies **coordination** within resilient performance, that is, **joint activity** that includes establishing and maintaining **common ground** with one another and the **negotiation and commitment** to undertaking a joint task [18]. The public space provides an additional **resource** in which to communicate the nature of the difference in the work performed by the team. The enactment of this strategy and the department's recognition that it was acceptable are an example of the co-production of resilience through plans for work that come "down" into the space of practice and bottom-up efforts to keep a system working [40].

### C. Study limitations

This was an exploratory study designed to apply concepts from RE to data from software development practice. It is not a comprehensive study and there are other episodes that we identified in the data. It therefore has inherent limitations as a standalone study. Accepting this, we consider the work in terms of trustworthiness in flexible design research [31].

The original studies employed various techniques to support trustworthiness including: member checking; seeking confirming and disconfirming evidence; triangulation of data, method and observer; and providing audit trails. These measures are described in earlier study reports [7, 21-23, 33, 34]; their qualities carry forward into this secondary analysis.

Two limitations relating specifically to this study are interpretation and bias. Regarding interpretation, the original studies were not focused on resilience, but they were focused on everyday practices in individual and team activities, and were designed to capture the informants' perspectives. Hence they provide a solid basis on which to identify resilient episodes. Regarding bias, the ethnographic stance requires a reflexive, nuanced mindset. As with all research involving people, bias is possible [31]. But in this secondary analysis, having two analysts who were variably acquainted with the data sets helped to keep the analysis independent. RE studies are often performed on historic data and by using previously-collected data we are continuing that tradition, but by adding in the ethnographic mindset we are countering, to some degree, the bias inherent in retrospective accounts.

## IV. Reflections and Future Directions

This study demonstrates that RE principles and concepts can be applied to illustrate elements of resilient performance in software development practice at the individual and small team levels. Furthermore, three salient episodes have been presented with elements that indicate the potential for resilient performance, based on a realistic characterisation of resilience in a software development context. In this section we reflect on our experience and suggest future research directions.

Performing an RE analysis of software practice is not straightforward. Firstly, although identifying potential episodes was straightforward because the data was well-known to the authors, agreeing on the focus of the episode and extracting their characteristics required considerable discussion and reflection. Secondly, while RE takes a socio-technical view of resilience, some affective and cultural factors, which prior work in software engineering has shown to be important to practitioners, are not accounted for in RE

frameworks, such as peer reputation [3] and shadow tactics [17]. These factors and how to capture them through an RE analysis deserve further investigation. Thirdly, determining whether an aspect of practice is resilient or not requires an understanding of specific resilience goals within a domain. In this study we used an illustrative goal that focuses on making progress, but other resilience goals for software development might focus on team well-being or customer value, for example. The goal for any one study would need to be decided with practitioners and for the specific context. Fourthly, identifying competencies in practice and linking them to strategies in other RE literature requires detailed knowledge and understanding of the activity and the participants' point of view. This requires detailed fieldwork and analysis. Finally, RE concepts and vocabulary need to be interpreted within a software engineering context. For example, a strategy identified for *Episode 3* was "willingness to relax efficiency temporarily". In this context "temporarily" might mean for a sprint, or a release cycle, while in another context, such as a hospital, "temporarily" might mean for only a few hours.

However, an RE analysis also presents opportunities. Causal analyses within safety science have been found to be biased, reflecting analysts' aims rather than practitioner rationalities [16]. Avoiding such bias relies on fine-grained reflexive qualitative analysis, that maintains the practitioners' point of view. This resonates with the aims of ethnographic studies [32], and RE therefore provides a lens through which ethnographic data may be analysed. Sharp et al [32] identified four potential roles in empirical software engineering for ethnographic studies: to strengthen investigations into the social and human aspects of software engineering; to inform the design of software engineering tools; to improve process development; and to articulate research questions and complement other research methods. RE analyses may support any of these roles, but here we consider two: designing new tools and improving processes.

One value of understanding current practice is to make informed decisions about the potential impact of changes within software engineering environments such as introducing new tools and languages, or process improvements. Identifying activity in current practice that contributes to resilient performance allows its presence or subsequent absence to be tracked, and for the impact of the change to be assessed after time has passed. For example, questions framed using terminology from [9] might be: will the change remove observed interruptions to practice? will the actions taken remain relevant? do the threats to progress remain and will opportunities still present themselves in the changed environment? Furthermore, will the enabling conditions and resources within the environment that supported the activity remain or lose relevance?

Another value in characterising current practice in these terms is to capture patterns of behaviour that may be disseminated. Professional learning is widely recognized to be a component of resilient performance [29, 13]. Performing an RE analysis may, for example, expose useful strategies that can be shared through formal or informal learning channels.

Finally, the framework used here [9] was helpful in working with ethnographic data. It provided both structure and vocabulary to support an RE analysis and clarified how to focus on small teams and individuals. Performing an ethnographic study can be challenging [27], and structuring frameworks provide focus for fieldwork and analysis. One gap in prior RE studies identified by [9] is that they focus on finding evidence for resilience at different levels of abstraction, such as within an entire industry like aviation, or operations within an organisation, as within nuclear plants. At these higher levels of analysis, the nuanced details of practice within a profession are lost and this kind of analysis could not support the assessment of some key changes that affect software development teams. A second gap is methodological: because the RE discipline lacks shared criteria or common approaches for undertaking analyses, it is difficult for studies to build upon one another's findings. The framework used provides a foundation for producing a traceable hierarchy that links RE theory to empirical evidence and for building generalised categories (referred to as "markers" in [9]) that are applicable across industries. Although we have found commonalities between our observations and those in other domains, our analysis has not yet identified examples of practice within software engineering that can be generalised as transferable "markers" of resilience.

This set of reflections points to several future research directions. The study performed here was based on historic data to illustrate application. New studies that focus on collecting data specifically for an RE analysis would allow the technique to be explored in more depth, allow further investigation of social and cultural factors that are relevant to software engineering, and to define goals for resilient performance that are relevant within the domain. Salient episodes of practice identified through these new studies could then be used to fully evaluate their usefulness in determining the impact of changes to practice brought about by the growing number of automation projects in software engineering, e.g. towards the automatic generation of code such as chatGPT and GitHub's CoPilot.

Assuming that this evaluation is positive and following efforts in Air Traffic Control (ATC), we are inspired by Stroeve et al [35] to suggest the development of a catalogue of episodes for software development. Stroeve et al's work used the same small teams framework, and the resultant catalogue captures over 400 episodes related to ATC. A similar undertaking within software engineering would be considerable, but could be a useful resource for querying changes. One of the potential challenges in using Stroeve's catalogue is its size: finding relevant entries in the resource to use in analytic exercises is difficult. However, the concepts used to characterise the episodes are in themselves a useful way in which to structure and index the work.

Looking further to the future, a set of episodes that capture the potential for resilient performance may be useful in determining socio-technical requirements for new tools and other automation endeavours, as episodes capture the nuanced socio-technical context within which work happens.

## V. Conclusions

This paper aimed to illustrate whether and how concepts from RE can be used to characterise resilient performance in software design and creation at the individual and small team levels of activity. Findings from the preliminary study demonstrated that characteristics of socio-technical resilience are recognisable within adaptations made by developers and teams. The study also indicated how this may be done by utilising one framework to analyse ethnographic data of software practice. Further, being able to relate developers' strategies with examples in external literatures indicates that

agile and other software engineering practices reflect aspects of resilient performance observed in other domains. Applying this lens to software practice therefore has the potential to inform understanding about how changes to the system that supports software development impact professional practice, to provide input to specification and design activities, and in the longer term, to assess the impact of changes on resilient performance.

Several benefits and challenges of performing an RE analysis with software development data have also been identified. Applying the RE lens is promising, but there are some aspects that require adaptation before its full potential may be assessed, and further investigation of resilience goals within software practice is required.

## ACKNOWLEDGMENTS

We thank all study participants who informed this work.

## REFERENCES

[1] The Agile Manifesto. Agile Alliance. 2001. http://agilemanifesto.org/

[2] T.M. Amabile & S. Kramer. 2011. *The progress principle*. Harvard Business Review Press.

[3] S. Amreen, A. Karnauch, and A. Mockus. 2020. Developer Reputation Estimator (DRE). In Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering (ASE'19), IEEE Press, 1082-1085.

[4] A. Bäckevik, E. Tholén and L. Gren. 2019 "Social Identity in Software Development," *IEEE/ACM 12th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, pp. 107-114.

[5] S. Beecham, N. Baddoo, T. Hall, H. Robinson, and H. Sharp. 2008. 'Motivation in Software Engineering: A Systematic Literature Review', *Information and Software Technology*, **50**, 860-878

[6] R.I. Cook. 2020. Above the line, below the line. Communications of the ACM, 63(3), 43-46

[7] A. Deshpande, H. Sharp, L. Barroca and A.J. Gregory. 2016. Remote Working and Collaboration in Agile Teams. In *International Conference on Information* Systems. 11-14 Dec 2016, Dublin, Ireland.

[8] C. França F. QB Da Silva, and H. Sharp. 2020. Motivation and satisfaction of software engineers. *IEEE Transactions on Software Engineering* 46, 2, 118–140.

[9] D. Furniss, J. Back, A. Blandford, M. Hildebrandt, and H. Broberg. 2011. A resilience markers framework for small teams. *Reliability Engineering & System Safety* 96, 1, 2–10.

[10] M.R. Grayson. 2020. Cognitive work of hypothesis exploration during anomaly response. Communications of the ACM, 63(4), 97-103

[11] B. Johansen. 2009. *Leaders make the future*. San Francisco, CA: Berrett-Koehler Publishers.

[12] B. Johansson, and M. Lindgren. 2008 A quick and dirty evaluation of resilience enhancing properties in safety critical systems. In: Proceedings of the Third Symposium on Resilience Engineering, Juan-les-Pins, France, October 28- 30.

[13] E. Hollnagel. 2018. Safety-II in practice: developing the resilience potentials. Routledge. Chapter 2. What does resilience mean

[14] E. Hollnagel. 2012. *FRAM, the Functional Resonance Analysis Method: Modelling Complex Socio-technical Systems*. Ashgate Publishing, Ltd.

[15] E. Hollnagel, D.D. Woods and N. Leveson (eds). 2006 *Resilience Engineering: Concepts and Precepts.* CRC Press.

[16] E. Hollnagel and R. Amalberti. 2001. The emperor's new clothes: Or whatever happened to "human error". In *Proceedings of the 4th international workshop on human error, safety and systems development*. Linköping University, 1–18.

[17] I. Kirlappos, S. Parkin, and M.A. Sasse. 2014. Learning from Shadow Security: Why understanding non-compliance provides the basis for effective security. In *Workshop on Usable Security (USEC)*. 27 April.

[18] G. Klein, P.J. Feltovich, J.M. Bradshaw, and D.D. Woods. 2005 Common ground and coordination in joint activity. *Organizational simulation* 53, 139–184.

[19] T. Kontogiannis and S. Malakis. 2009. A proactive approach to human error detection and identification in aviation in air traffic control. Safety Science, 47, 693-707.

[20] J.-C. Laprie. 2008. From dependability to resilience. In *38th IEEE/IFIP Int. Conf. On dependable systems and networks*. G8–G9.

[21] T. Lopez, H. Sharp, T. Tun, A. Bandara, M. Levine, and B. Nuseibeh. 2022. Security Responses in Software Development. *ACM Transactions on Software Engineering and Methodology*. Early access, https://dl.acm.org/doi/abs/10.1145/3563211.

[22] T. Lopez, H. Sharp, M. Petre, and B. Nuseibeh. 2021. Bumps in the Code: Error Handling During Software Development. *IEEE Software* 38, 3, 26–34.

[23] Lopez, T., Petre, M., & Nuseibeh, B. 2016. Examining active error in software development. In 2016 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC) (pp. 152-156).

[24] S. Malakis, T. Kontogiannis. 2008. Cognitive Strategies in Emergency and Abnormal Training: Implications for Resilience in Air Traffic Control. In Proceedings of the Third Symposium on Resilience Engineering, Juan-les- Pins, France, October 28-30.

[25] N. McDonald. 2006. Chapter 11 Organizational Resilience and Industrial Risk. In *Resilience Engineering: Concepts and Precepts.* Edited by Hollnagel, Woods and Leveson, pp 155-179, CRC Press.

[26] R. Mumaw, E. Roth, K. Vicente, C. Burns. 2000. There is more to monitoring a Nuclear Power Plant than meets the eye. Human Factors, 42(1), 36-55.

[27] C. Passos, D. S. Cruzes, T. Dyba, and M. Mendonça, 2012. Challenges of applying ethnography to study software practices. In Proc. ACM-IEEE Int. Symp. Empirical Softw. Eng. Meas., pp. 9–18

[28] P. Ralph, S. Baltes, G. Adisaputri, R. Torkar, V. Kovalenko, M. Kalinowski, ... & M. Zhou. 2020. Pandemic Programming: How COVID-19 affects software developers and how their organizations can help. arXiv preprint arXiv:2005.01127

[29] J. Rasmussen. 1990. The role of error in organizing behaviour. *Ergonomics* 33, 10-11, 1185–1199

[30] J. Reason. 1990. *Human error*. Cambridge university press.

[31] C. Robson & K. McCartan. 2016. *Real world research: a resource for users of social research methods in applied settings*, 4th edition, Wiley.

[32] H. Sharp, Y. Dittrich and C. deSouza. 2016. The Role of Ethnographic Studies in Empirical Software Engineering. *IEEE Transactions on Software Engineering*, 48, 8, 786-804.

[33] H. Sharp, and H. Robinson. 2008. Collaboration and co-ordination in mature eXtreme programming teams. *International Journal of Human-Computer Studies*, 66(7), 506–518

[34] H. Sharp, H. Robinson, J. Segal, and D. Furniss. 2006. The Role of Story Cards and the Wall in XP teams: a distributed cognition perspective. In *AGILE 2006 (AGILE'06)*. IEEE.

[35] S.H. Stroeve, B.A. van Doorn, & M.H.C. Everdij. 2013. The human contribution - Analysis of the human role in resilience in ATM. Report number: Deliverable D1.2, EU FP7 Resilience 2050, DOI: 10.13140/2.1.3527.3287.

[36] B. Tjørhom and K. Aase. 2012. The art of balance: using upward resilience traits to deal with conflicting goals. In *Resilience engineering in practice: A Guidebook*. 2012. CRC Press, 157–170.

[37] B. Trinkenreich, M. Guizani, I. Wiese, T. Conte, M. Gerosa, A. Sarma, & I. Steinmacher. 2021. Pots of Gold at the End of the Rainbow: What is Success for Open Source Contributors?. IEEE Transactions on Software Engineering, 48(10), 3940-3953

[38] R. Westrum. 2006. Chapter 5 A Typology of Resilience Situations. In *Resilience Engineering: Concepts and Precepts.* Edited by Hollnagel, Woods and Leveson, pp 55-65, CRC Press.

[39] D.D. Woods and J. Allspaw. 2020. Revealing the Critical Role of Human Performance in Software, *Communications of the ACM*, May, 63, 5, pp 64-67, doi:10.1145/3380468

[40] D.D. Woods. 2006. Chapter 2 Essential Characteristics of Resilience. In *Resilience Engineering: Concepts and Precepts.* Edited by Hollnagel, Woods and Leveson, pp 21-33. CRC Press.