Putting the Sec in DevSecOps: Using Social Practice Theory to Improve Secure Software Development

Debi Ashenden
debi.ashenden@port.ac.uk
School of Computing, University of Portsmouth
Portsmouth, UK

Gail Ollis
gail.ollis@port.ac.uk
School of Computing, University of Portsmouth
Portsmouth, UK

ABSTRACT

Practices such as open source development, agile, DevOps and DevSecOps mean that cyber security professionals need to find ways to blend cyber security with software development practices. One way of approaching this is as an awareness, education and training problem and many organisations are focusing on training software developers in cyber security. In this paper, however, we make the case for looking more broadly at group rather than individual behaviours, by examining the social practices of software developers. Changing software development practices are shaping the lived experience of software developers and we argue that understanding these practices will enable us to improve secure software development. We use social practice theory as a framework to develop recommendations for aligning and blending cyber security and software development. To achieve this, we carried out a rapid review of research on software development practices and supplemented this with data from ten key informant interviews to ascertain what we need to consider when developing an intervention for secure software development. Finally, we outline how our research could be used to develop a workshop that would facilitate the co-creation of security practices for software development. We conclude with suggestions for future research.

CCS CONCEPTS

• **Security and privacy → Social aspects of security and privacy**; **Software security engineering**; **Human and societal aspects of security and privacy**.

KEYWORDS

Secure Software Development, Social Practice Theory, Cyber Security, DevSecOps

ACM Reference Format:
Debi Ashenden and Gail Ollis. 2020. Putting the Sec in DevSecOps: Using Social Practice Theory to Improve Secure Software Development. In *New Security Paradigms Workshop 2020 (NSPW '20), October 26–29, 2020, Online, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3442167.3442178

1 INTRODUCTION

In the move to agile development and continuous integration or continuous delivery, software development cycles are shorter and faster. It has always been difficult to ensure that security is considered up front in the software development process. Indeed research has shown that in established open source projects, vulnerabilities can exist for many years and through many versions of the software [48]. With software development trends now requiring an increase in the tempo of security processes the relationship between cyber security professionals and software developers has never been more important. Changing software development practices shape the daily lives of software developers in new ways and cyber security professionals have to be able to negotiate, align and blend the way they implement security with these new ways of working. This puts pressure on a relationship that has not always been easy [7]. The success of new software trends is as much about close collaboration and trusted relationships as it is about the tools, processes and strategies that an organisation has in place. In this paper we examine the social aspects of software development practices and explore what this means for cyber security.

New approaches to software development emphasise collaboration to bring a focus to security [14] with the goal of ensuring the inclusion of security from the beginning of the development lifecycle and increasing awareness of obstacles to engaging with security [46]. This also raises the question of where security happens in the software development process. Does it sit within the individual skill sets of software developers, or the tools they use to test the security of their code; does it reside within the process that governs software development or is it the task of one security expert within a project team? It is likely that some combination of these is required but getting the balance right needs to be negotiated and, as our research demonstrates, incorporating these elements effectively into everyday working practices requires a change of culture as well as productive dialogues between software developers and cyber security professionals.

Cyber security is a complex challenge in its own right, affected by both social and technical factors [13] and cyber security professionals have consistently advocated that security should be 'shifted left' in the software development process so that it is considered early on in a project. The problems around secure coding and software developers is not a new topic as Apvrille and Pourzandi [6] point out. The cyber security research community has long since drawn attention to the importance of the usability of security in the software design process [19, 70] and on tool development to support developers [52, 69]. As Pieczul et al. [51] point out though it is 'not feasible' to expect developers to become security experts and yet in spite of this, as Wurster and Van Oorschot [69] state, 'apparently many in the security community continue to believe that developer education will solve the problem'.

As a subset of cyber security, secure software development is also receiving attention from organisations, Government bodies in the UK and in the US, as well as software development researchers [3, 5, 11, 45]. At their best, white papers, guidance documents and research papers speak to both cyber security professionals and software developers but there is often an implicit suggestion that security is something software developers now have to add to their responsibilities. Given that the relationship between cyber security professionals and software developers is not always synergistic, and may in some instances be adversarial [8], the suggestion that software developers need to step up and do more is unlikely to set the right culture for collaboration and change.

Attempts to ensure that software developers engage with security processes, secure coding practices and the use of security tools, usually focus on raising awareness of security risks and secure coding training programmes. Behaviour change is a complex topic and such initiatives often have limited success, not least because raising awareness does not necessarily lead to behaviour change [7]. As Ashenden and Lawrence [7] point out, the first task in changing behaviour is to generate insight into the community whose behaviour you seek to change and in this paper we use social practice theory (SPT) as a way of generating this insight. The benefit of using SPT is that it addresses behaviour at a group level 'rather than focusing solely on the individual' [62]. This means that responsibility for change is shared and collective. It is also a way of understanding the contextual factors that play a key role in the success of behaviour change interventions by acting either as a motivator or a barrier. As Beautement et al. [9] found in their research there is a point beyond which individuals will choose work arounds rather than following security processes and SPT allows us to start to understand the factors that might impact on that tipping point. Our use of SPT builds on the research by Kocksch et al. [31] that seeks to, 're-frame IT security as a social phenomenon in the making' by acknowledging that security takes place in the 'in-between' – between people and things – and is 'a matter of forging relations'.

The study of social practices stems from the work of sociologists such as Giddens [21] and Reckwitz [53] and in more recent years has increasingly been advocated for use in behaviour change research [59, 62]. Practices are 'ways of understanding, knowing how and desiring' [53] and SPT offers a way of 'understanding the complex dynamics that that make up a practice' [62].

We use SPT to generate insight into the behaviour of software developers that will help cyber security practitioners understand how to co-create behaviour change initiatives that are more likely to be successful. As such we offer the first step in a behaviour change programme by using SPT to discover how software developers understand their current behaviours through their experience of the practice of software developer. SPT allows us to focus on the developers' point of view, attempting to make explicit the reasons for their practices. This can explain the 'why' of developer choices in context, highlighting how they choose their methods and tools in everyday use [16] and ensures that, 'meaningful and sustainable improvements can be researched, devised and introduced to practice' [57]. In terms of ameliorating behaviour, understanding the developer's logic behind current methods is likely to offer an easier transition to more secure practices. Through SPT individuals become the agents of practice rather than the focus, allowing the actions and interactions to become more distinct; this enables a clearer view of where there are strengths and weaknesses with regard to the inclusion of security.

2 THE PRACTICE OF SOFTWARE DEVELOPMENT

In recent years software development research has begun to explore ways of theorising about software development practices [16, 49]. Päivärinta and Smolandar [49] point out though that when they reviewed research on software development in professional organisations between 2010 and 2013, out of 271 articles they only found four that included research with actual software developers. They note that most research focuses on 'technical rationality', where software developers and the context in which they work are abstracted to provide input to formal models, and they call for a move to 'reflection-in-action' research that seeks to understand the lived experience of software developers.

This accords with the views of Sharp et al. [57], who make the case for more ethnographic studies as a way of understanding the, 'socio-technological realities surrounding everyday software development practices'. The use of the term 'practice' [16, 49, 57] is a way to describe the bundles of activities that make up the software development process in the real world. Practice theory is a sociological theory that Dittrich [16] defines for the context of software development as a 'commonly agreed upon way of acting that is acknowledged by the team'. Dittrich acknowledges that there is a difference between software development 'as described' versus software development 'in use' because while there will be existing processes and formal rules that support the integration of security with software development there is often a gap between a documented process and what happens in reality. Software developers themselves acknowledge this in their use of terms like 'ScrumBut' [1]. As Ashenden and Lawrence [8] point out this is particularly the case with security where the gap is between 'an organisation's mandated formal security processes and what actually happens'. They attribute this to organisational pressures in other areas that mean that 'projects have to finish, systems have to run and the business has to move forward' [8].

More recent research has examined the motivations of the software developer community [10, 56, 66]. There is a distinct culture and community surrounding software developers however, which Sharp et al. [58] argue transcends borders and organisations and has been identified as central to combatting social and technical problems regarding software and security [11].

Of course, cyber security too is a practice enacted in various ways in different organisations and when we think about integrating cyber security with software development we can see that we are actually trying to blend and merge two, often very different social practices. Williams [68] writes about this in a short summary of three annual Continuous Deployment Summits held by Facebook, Netflix and Google between 2015 and 2017. She makes references to practices but without explicitly grounding her review

[1] ScrumBut is where an organisation or team uses Scrum but knows they fall short in practice – hence the term 'we use Scrum but ...'

in practice theory and highlights three themes that are key to the successful integration of cyber security and software development practices; communication, culture and technical aspects. She points to areas where security and software development practices have been blended successfully – where there is 'close communication', 'shameless retrospectives' that include security and privacy failures, and check-in points for testing. More recent work by Kocksch et al. [31] lays the foundations for the study of the practice of security. In their work they use ethnographic vignettes to explore the bundle of practices that make up security and conclude that security is an 'ongoing collective endeavor' that requires both empathy and care if it is to achieve its goals. This research provides a useful starting point for understanding the social practice of software developers.

3 SOCIAL PRACTICE THEORY

Social Practice Theory is increasingly used within behaviour change programmes in fields as diverse as environmental issues and human consumption [18, 25, 29]. Part of the success of SPT in these areas is that the individual is no longer the focal point of change. This supports a trend of encouraging agency in individuals so that they direct their own behaviour change, rather than an outside force, such as cyber security professionals in this instance, imposing modification through regulation [38]. This seems especially pertinent when we increasingly need cyber security professionals and software developers to work together as peers. It is an approach that should encourage a move towards negotiating the inclusion of security in software development so that both communities are satisfied.

SPT is the examination of preferences, habits and behaviours of people which develop in a social context [26], but instead of individuals being the unit of analysis, it is the 'practices' that are examined [42]. A practice is the core component of SPT and is made up of three elements; materials, meanings and competences [60]:

- Practices are defined as a way of understanding and behaving around a specific act that has been normalised by routine. A practice may appear at different times and places, 'carried out by different body/minds' [53].
- Materials are the physical objects, tools and infrastructure used in the practice.
- Competence refers to the knowledge, skills and understanding needed to carry out the practice.
- Meanings are the cultural understanding, expectations and shared meanings that circulate through the practice.

Shove [59] advocates SPT as an approach to help institutions structure their actions by 'making some [actions] very much more likely than others' by providing a focus on the specifics of common social processes, which create observable patterns [29].

We can see an example of how the social practice of cyber security in the workplace has changed over time. Two decades ago an employee's technological materials might have been a desk top computer on an organisation-wide network. It would have had basic word processing/spreadsheet functionality. The employee would have had little to no technical security competence beyond perhaps a password for access that may well have been shared. The meaning given to the social practice of technology in the workplace may well have included innovation, forward thinking and business success, it is unlikely it would have included confidentiality, integrity or availability. Today the same employee would have a range of technological materials that could include a desktop computer, a laptop, a tablet and a smart phone. They would be able to access their company information on all devices. They will have an array of apps to help them communicate and work collaboratively alongside traditional office applications. An organisation will now ensure the employee has some competence in, and takes responsibility for, both using and securing the devices and the information on them. The competences have now extended from being able to construct more and often increasingly complex passwords to managing two-factor authentication as well as noticing and acting to prevent phishing attacks.

Employees also need to be able to manage the security of the interaction between the technology and the physical environment which may change as they work while on the move, switching between devices. Employees now have to have competence in managing context switching so they don't inadvertently reveal company information while on a train for example. They have to protect their devices from being stolen as they move between physical locations. The meanings applied to cyber security as a workplace practice have also changed. Previously it was hidden, someone else's responsibility and not something to dwell on, whereas now the employee is constantly told it is their responsibility. Cyber security is now a serious topic for organisations. Employees know that if they fail at practicing cyber security there may be a regulatory fine, they could be disciplined or lose their jobs, there are standards to comply with. The social practice of workplace cyber security for employees has changed significantly over recent years. To begin to understand the social practices of software development and how it might impact on how we integrate cyber security we carried out two activities. Firstly, we conducted a rapid review of literature on software development to see what we could learn about social practices. Secondly, we supplemented the literature review with ten interviews with key informants. These were people across the software development industry who could give us further insight into the social practices that might cause barriers to, or constraints on, blending cyber security with software development.

What follows is by no means a comprehensive study of the social practices of software developers but it does offer a proof of concept for how SPT could be a useful lens to support the co-creation of practices for secure software development. Our analysis of the literature and the interviews can be split into two key practices. The first is a meta-practice that is determined by the philosophy of software development activities and distinguishes between proprietary software development and open source software development. The second practice focuses more specifically on the bundle of practices that make up software development itself.

4 TWO REALMS: PROPRIETARY OR OPEN SOURCE

The literature on the practice of software development typically considers either the proprietary software realm or the open source realm, not both together. These are illustrated in Figure 1, each realm including its own characteristic social spheres. Proprietary

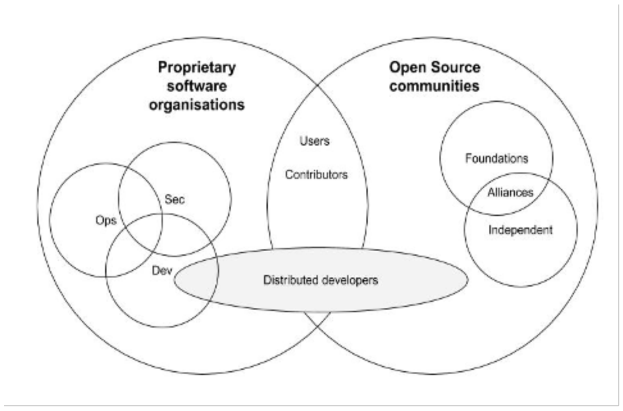

Figure 1: Realms of Software Development

software is the realm of commercial companies and is 'closed source'. Customers typically enjoy the right only to run the software and do not have access to the source code. In the OSS realm, development effort is voluntary, source code publicly available and permission to use or edit it constrained only by the terms of free licences.

However, proprietary software makes extensive use of Open Source Software (OSS) so software security depends on the practices of both realms. There are commonalities between proprietary and OSS development practices in meanings, materials and competence elements of SPT, suggesting potential for behaviour change initiatives which will work for both realms.

4.1 Two Realms: Meanings

Historically many companies perceived OSS to mean insecure code, posing a risk to the security triad of confidentiality, integrity and availability [61]. This has changed as OSS has matured. A commercial product's need for trust in OSS code, confidence in its legal and reputational status, and access to shared knowledge about it can now be met by projects under the governance of formal affiliation and governance structures that have helped to establish OSS as trustworthy [17].

There are still independent projects led informally by core developers [55], a risk that should be considered when choosing libraries for secure software development. However, this does not mean that OSS is developed by enthusiastic amateurs. Many software developers operate in both realms and apply the same professional standards throughout. It is their obligations and motivations for the work that are realm-dependent.

4.2 Two Realms: Materials

The change in companies' perceptions of OSS has created a greater overlap between the two realms than just sharing of some personnel. Companies are now OSS users, routinely using OSS tools [1, 35] and embedding OSS libraries in their products. Given this vested interest in OSS, some become contributors to its development by hiring OSS community leaders or endorsing the use of company time for work on open source projects [17]. Companies may also 'open source' (make publicly available) source code which they initially developed in-house. Google, for example, developed the TensorFlow [23] platform (machine learning computation, based on patterns and inference rather than a sequence of instructions) for internal use and later released it under the Apache licence.

Whether for a company project or an OSS one, the day-to-day software development workflow is similar. Coordination of many individuals' contribution to a code base is normally implemented using a Distributed Version Control System (DVCS) such as Git [22] to support independent work that can later be combined in a well-managed fashion. GitHub offers the same workflow features with an added layer of infrastructure to host a public Git repository. It also provides a gatekeeping mechanism for code changes to be reviewed by a privileged user before they are accepted, known as a 'pull request'.

The "pull request" mechanism is used by some projects in both realms. The key differences in the use of DVCS lie merely in the extent of the distribution (local employees, employees across multiple sites or individual contributors anywhere in the world) and where the repository is hosted (in a company server or online). GitHub, for example, is popular with OSS projects, but some commercial projects too find it useful to outsource hosting infrastructure instead of providing their own server.

Other tools of software development (programming languages, development environments, testing methods and so on) are also common to the two realms. The similarities are evident in the IEEE Spectrum ranking [28]; the top six languages on open source hubs (Python, Java, C, C++, JavaScript and C#) are exactly the same as those most in demand by employers. It is symptomatic of the change in industry use of OSS that none of these is a proprietary programming language or reliant on commercial licences; free compilers/interpreters are available for mainstream platform architectures. Fully proprietary languages such as MATLAB [40] do not make the top ten for either open source use or industry demand.

4.3 Two Realms: Competences

Some of the overlap in tool use can be accounted for by the number of paid developers also contributing their professional skills to OSS. Necessary competences across OSS and proprietary software development are very similar. There is a logistical difference in the barriers to participation - professional developers are selectively recruited, whereas anyone can volunteer work to OSS projects – but in either case a code review may determine whether or not the code is incorporated.

Distributed teams require some different competences from co-located ones. Licorish and MacDonell [33] cite two challenges: time zone differences and, between Asia and the West, cultural mismatch. Overcoming these is possible [12] but requires effort. OSS project developers may be co-located only when sponsored by employers or participating in a 'sprint' or 'hackathon' (a physical gathering to work together on a project, whether independently organised or supported by a tech conference [2]) and therefore face the challenges of a very distributed team. Conversely, while proprietary software projects may have teams distributed across company sites it is commonplace to have one or more teams of co-located developers.

4.4 Implications for Security

With the meaning of OSS now much more positive in companies' perceptions, cyber security professionals need to know where the organisation or individual projects sit on the proprietary to OSS spectrum as this could impact on their risk exposure. They also need to understand the affiliation and governance structures that are being relied on to provide trust and to work with them to assure the same security standards whilst also understanding any vulnerabilities that such structures may cause.

The goal for working with the OSS realm must be 'DevSec' rather than 'DevSecOps' since OSS typically has no 'Ops' (see Figure 1) but otherwise the materials used are likely to be the same. This offers a potential locus for security processes that integrate well with developers' existing practices across both realms. The role of DVCS as a fundamental material element in both makes it a particularly important element to consider.

There are two competence factors with security implications. One concerns the authority to mandate security knowledge. Voluntary participants have rather different obligations to those of paid employees, although the overlap in personnel means that security knowledge among professional developers should also influence OSS development. The second factor is geographically distributed teams and the impact of associated cultural differences. These have implications for the take up and adherence to security policies and processes, whichever authority sets them.

5 COLLABORATIVE PRACTICES: LEARNING FROM DEVOPS AND DEVSECOPS

Our search terms explicitly combined DevOps or DevSecOps with Social and Practice to explore the nature of the enhanced collaboration that is sought by these approaches. Since OSS typically has no 'Ops' much of the material in this section comes from the proprietary realm. However, the OSS realm is by definition a community effort so it, too, contributes to the discussion of collaborative practices. It is a social context with different meaning but many of the same competences and materials.

DevOps is the combination of software development and operations, a practice designed to address the coordination problems that can arise when a live software system is installed and supported by an operations team largely separate from the developer team writing the software. In our goal of blending cyber security (Sec) with software development (Dev) practices we can learn from the DevOps experience of blending the 'two cultures' of Dev and Ops and from the further DevSecOps experience of also including security in the blended activity.

5.1 Collaborative Practices: Meanings

Iden and Bygstad [27] highlight the importance of trust between the 'two cultures' of Dev and Ops. For Luz et al. [36] a collaborative culture is at the heart of the DevOps model, along with straightforward communication and blameless, shared responsibilities. Although their account centres on a single collaborative culture rather than two cultures with mutual trust, they make it clear that the collaborative culture is not something that a team simply has as an endogenous quality. It is enabled and maintained by the practices used and it needs to be kept alive. In the words of a participant in the study by Luz et al, this 'is still a challenge for us… The idea is not to let the culture die.' [36].

These principles also apply to DevSecOps, where communication of security requirements is added to those of development and ops. The intention is the same: to enable decisions about each of three aspects to be set with an understanding of the consequences for the other aspects and those who are responsible for them. Tomas, Li and Huang [64] describe four key elements for successful DevSecOps practice: culture, automation, sharing and measurement. Of these, culture is the bedrock, consistent with the findings of Iden and Bygstad [27] and Luz et al. [36]. Successful adoption of the new, collaborative way of working can be supported by tools and processes, but depends on having and fostering a collaborative culture. Together these papers show that bringing Dev and Sec together (with Ops or, as will be the case in the OSS realm, without) is not a one-off intervention but an ongoing process of maintaining that collaborative culture, a process in which the right tools can be helpful.

The emotional landscape of developers also has an impact on the success of new structures and processes [41] and it is logical that this would include any structural or process changes to accommodate cyber security. For example, how software developers perceive a transition to agile will depend on issues around collaboration, communication, commitment or resistance to change, trust, and organisational culture [20]. A study by Graziotin et al [24] suggests that while happiness may only benefit the software developers themselves, conversely software developers believe that their individual unhappiness is detrimental to the team.

A large-scale survey conducted by Meyer et al. [44] highlights the importance of agency for software developers and strongly recommends the need to 'empower developers to choose their own tools and tasks' if they are to have 'a good day'. In their study they recognised that developer activities can be split into two groups – coding tasks and collaborative activities. Satisfaction for developers depends on balancing these two groups of tasks. Their description of what makes a good day is when they are 'working in code', having 'constructive discussions', learning and helping co-workers. Their acceptance of certain collaborative activities, such as meetings, also varies through the project lifecycle.

One aspect of developer practice that can be particularly fraught with emotion and that is very relevant to security is code review. There are different types of code review ranging from pair programming, informal walk throughs and mandatory approvals. In the research that we reviewed, problems were identified as arising from scheduling and timing issues [37] for example, code review taking a long time or not happening at the right time, or through rejection of code [4]. In both cases, however, good communication was a way to overcome these problems [37].

Alami et al. [4] suggest that the 'emotionally loaded practice' of code review works in the open source community because of what they term the 'hacker ethic' which blends an 'ethic of passion' with an 'ethic of caring', meaning a strong intrinsic motivation to 'learn, to grow reputation and to improve one's positioning in the job market', We note the overlap in personnel between OSS and proprietary software and suggest that the desire to produce good quality code does not disappear when the same developer turns to their professional work, but may be constrained by their work environment.

5.2 Collaborative Practices: Materials

The issue of communication and the role it plays in successful collaboration comes up repeatedly. Luz et al. [36] note that face-to-face communication was preferred by their participants, but where this was not possible they wanted to use tools rather than a formal ticketing system. Licorish and MacDonell [34] analyse communication artefacts by a project team distributed across the USA, Canada and Europe. The 474 different contributors included team leads, project managers and admins as well as programmers. The research focuses on quantitative aspects of communication such as how much time developers spend on different kinds of tasks, how much communication there is for different task types, and how long tasks take.

Communication between software developers is important and more complex than current research suggests. A key tool around which collaborative behaviour is typically coordinated is the Distributed Version Control System (DVCS), an essential repository of a project's files which tracks changes to them and allows any version to be recovered. Examples include Git [22] and Mercurial [43]. They allow changes to be worked on in parallel and then individuals' work to be merged together when ready. A history of the changes to files is kept as they are 'checked in'. Kalliamvakou et al. [30] examine collaborative practice via GitHub. The trace such tools create has great potential to show metrics such as how long developers work independently before merging their work into the whole, which files are edited by multiple developers, and which files experience a large number of changes. Data mining is no doubt an attractive approach to yield detailed metrics, but without ethnographic study it can only tell us about some of the mechanics and not about the meaning or the experience of being involved in a collaborative project.

5.3 Collaborative Practices: Competences

Knowledge sharing is an important aspect of collaboration. Licorish and MacDonell [33] suggest that knowledge sharing in a project centres around a subset of the team that they call 'core developers', also referred to as 'software gems'. Whether or not their formal role is a leadership one, these people play a co-ordinating role in knowledge sharing. While some papers identify such people by the frequency of their contributions to the impersonal communication artefacts of a project, Licorish and MacDonell [33] report that knowledge sharing is a social process. They recognise that a team's behavioural norms are an important influence on individual knowledge sharing behaviour; social motivation is a factor, and closely linked to trust. Whether or not the overall pattern of a team's communications follows that found in its online artefacts is unclear.

5.4 Implications for Security

Aligning security with quality and emphasising that security is part of the ethic of care could be a useful way of framing security messages and communications. Our review suggests that developers may already have the ethical mindset to be able to include security as part of their quality ethic. In the proprietary realm, this needs companies to understand the relationship and be supportive of developers' inclination to exercise their existing ethic of care. Our review did not identify any papers which explored the meanings employers ascribe to caring about code.

Cyber security professionals should, where possible, use the software developers' preferred method of communication and methods by which they already share knowledge as part of their workflow. The metrics which demonstrate communication patterns suggest a promising place to start, but these would be complemented by an understanding of the meanings ascribed to communication methods to avoid proposals that fit the overt artefacts but conflict with developers' experience of using them.

To facilitate knowledge sharing of security issues cyber security professionals need to identify the core developers who have influence in a project. These individuals would make useful Security Champions. One way to identify them would be by examining the communications across a team but at the moment there is a lack of research on how influence is built and exhibited in developer communication.

Ensuring software developers have agency and are empowered in their work, that they can still balance coding and collaboration, are going to be important considerations in any security intervention. Cyber security professionals should aim to negotiate processes and tools use with software developers as this would both engage them and ensure they have agency. Ideally the tempo of security activities should align with the tempo of software development activities so that meetings focusing on security happen at the optimum time in a project lifecycle where possible. This could be facilitated by integrating security activities at the natural 'meeting points' that a DVCS creates as part of the collaborative workflow in both OSS and proprietary realms.

6 INTERVIEWS

Our literature review demonstrated that while we found useful information about the materials, meaning and competences we needed to dig deeper to start to understand some of the nuances of the lifeworld of software developers. As noted previously there are very few ethnographic studies of software developers in general. While we would advocate further ethnographic studies, in the meantime we carried out a pilot study of interviews with key informants to demonstrate the potential value of SPT. Key informants are individuals selected because they have a particularly informed perspective on a topic. They are hand-picked to provide insight into, and the identification of, relevant issues. Qualitative interviews are a recognized data collection method for SPT and key informants are often 'important gatekeepers for insights on practices' [39].

The interviews were between 45 minutes to an hour in length. They were semi-structured around the topics of interviewees' experience of developing software, their working day, their views of what it meant to be a software developer, their understanding and perceptions of security in the software development process, the materials that they use in their day-to-day work and the skills they value. Our interviews shed light on some aspects of materials and competences but also focus on the meanings software developers assign.

Our interviewees included five open source software developers from a global open source software company. These interviews were conducted during a week where the researcher worked alongside the developers in their office. This allowed for informal conversations as well as more formal interviews. In addition, we interviewed one technical pre-sales consultant with experience in both software development and security from the same organisation. We also interviewed one agile coach who is responsible for advising and coaching software developers in the finance sector, one solution engineer from a small open source software company, one software security architect and a senior manager for trustworthy computing from a global proprietary software company. Between them they worked across five different organisations and two different countries (the UK and the US). Five of them had experience with cyber security as well as software development.

6.1 Revisiting the Two Realms

The overlap between OSS and proprietary systems for software development was borne out in our interviews. One interviewee asserted, however, that sometimes code can be open source in name, in that it is placed on GitHub but the, 'developer mentality behind it is closed source' so it can be downloaded, changed and used but cannot be updated or developed further via GitHub. Alternatively, it could be a blend of closed source code plugged into an open source core. Another interviewee said that nearly 'every software developer is using open source' but that what they produce is not necessarily open source. The notion of a developer's mentality being either open source or closed source is interesting but is not addressed in the research literature. The idea of software development practices being as much about a state of mind as about tools and processes is one that needs to be explored further.

Ethics and affiliation become entwined for OSS developers. Some organisations have ethics processes to determine how developers should act in the communities they work in. There was reference to giving back to the community and one said that 'Making a contribution is important' which is what we found in the literature around an ethics of care. Some developers commented on the difficulty of balancing affiliations where they are affiliated to their own organisation but also to organisations such as Apache.

With our interviewees, distributed teams were the norm covering a wide geographical distribution (one team had members in the UK, US, Ireland, Serbia and China). One interviewee said that one of the developers in his team sometimes worked out of New York – he didn't know why but it didn't matter as long as she told him which time zone she was in for meetings.

6.2 The Lifeworld of a Developer

The idea of the 'lifeworld' is about understanding the lived experience of others, focusing on their perceptions. Understanding the lifeworld of software developers means that cyber security practitioners can start to develop empathy with their day-to-day experiences [31]. This is the first step to understanding why people do what they do [7] and allows cyber security practitioners to understand the context in which software developers experience security from their perspective which can then inform how they implement security practices.

One of our key informant interviewees said that open source development tends to attract individuals who are 'passionate' about what they do and will often work longer hours than contracted. They have 'passion' and 'attention to detail'. Another said that their organisation may be 'sponsoring the project' but developers will feel that 'it's theirs' and may 'forget why they're here'. Developers also enjoy the creativity of their role with one interviewee saying, 'I'm not a creative person but it's the one thing where I can write something down and see something happen'.

Some developers need a 'darkened quiet room' to work whereas other need a 'social, community, environment'. Pair programming can work for some but a lot of use is made of internal, virtual chat facilities and this is especially important when there is widespread geographical distribution. For others, the 'teddy bear experience' (talking it through out loud as if to a soft toy) works just as well. When it came to collaboration one interviewee said that a high performing team is one where there is 'implicit trust'. Software developers need agency as we saw in the review of the research but one way this was described in the interviews was that they, 'don't want to have to jump through hoops to get what they need' and they expect access to GitHub now because they use it as a 'portfolio'. For communication they need to be able to talk to other team members easily and quickly but using tools. They want chat but 'without having to get up'. They also want chat linked to their other collaboration tools.

The importance of influence and knowledge sharing came through in a richer way in our interviews. The value of interpersonal skills was highlighted as an important aspect of team work. One participant said that many software developers are good at team work but do so via virtual communication so they are team players 'but not in a traditional sense'. The ability to influence and persuade was also seen as an important skill and this is why role and profile outside their organisations is important. They become accepted by 'evangelising about a product', providing code, documentation and tutorials or running special interest groups. One software developer said that the qualities of a good software developer are being an 'awesome communicator', 'articulate', having a 'broad knowledge of many subjects' while being 'assertive' and able to get 'people's attention'.

Our interviewees had a range of perspectives on code reviews and acknowledged that the experience depends on the team dynamics and the project. Some are very strict and some 'require two code reviewers'. They can be 'frustrating' because some take 'months with 20-30 iterations'. A software developer's experience of code review can depend on reputation – if you already have a good reputation then your code might not be so closely scrutinised.

One interviewee suggested that you can see group dynamics through the process. For example, spending less or more time reviewing someone's code either because you think it's likely to be good or because you think it's likely to be bad. Bearing out what the literature says about code review one software developer highlighted that it could be 'demoralising if it takes five days'. For others the process was not 'necessarily an ego issue' but was supportive and egalitarian where junior developers would review code for those more senior.

6.3 Implications for Security

Our interviews shed light on the lifeworld of software developers in a way that added more richness to the meaning of social practices than we could extract from the literature. This is an important first step and further research needs to be carried out here as it is by understanding the lifeworld in this way that we will be able to effectively blend the social practice of software development with that of cyber security. The interviews have started to reveal what is important to software developers and in turn these elements need to be taken into account in any security intervention. They help us to understand how software developers see themselves, their identity, affiliations and motivations. Their focus on ethics and their role within their community are particularly important and both could be invaluable in promoting security. Their perception of their affiliation could potentially increase security risk. The process of code review needs to be examined to ensure it is not leading to security vulnerabilities merely because of the way it is carried out. Finally, the range of requirements for their work environment, especially with regard with communication, breaks through stereotypes of software developers and offers lessons for cyber security professionals who need to engage with them.

7 A CULTURE OF SECURE SOFTWARE DEVELOPMENT

Iterative software development initiatives such as DevSecOps are sometimes seen as synonymous with a rejection of the 'additional development efforts' [47] required by security and are seen as an overhead [65]. Rindell et al. [54] highlight what they see as an incompatibility between iterative models and the more traditional sequential models of security engineering, noting that the idea of security requirements as non-functional requirements is still a convenient argument. Oyetoyan et al. [47] do note, however, that some agile organisations have managed to incorporate security 'in a way that fits their processes and practices'. Both Oyetoyan et al. [47] and Rindell et al. [54] propose mitigations which call for aligning security with the software development process. What is missing from the picture is some of the detail of what is actually involved in the lifeworld of software developers and the social practice of software development.

The role of culture is an influential aspect of the software development practices [50] and to be able to improve security within a software development lifecycle there must be acknowledgement of social as well as technical aspects [63, 64]. To create and maintain a supportive and healthy security culture within the organisational structure, security must be included as a natural aspect of developer routine and the end product [67], rather than being seen as a separate feature or add-on.

The study by Tomas et al. [64] sets out the need for a security culture among developers. Trust plays a part here too, in the wider relationships between software developers and cyber security professionals. Study participants reported that they 'feel attacked' by security engineers if they fail to measure up. As Ashenden and Lawrence [8] point out, software developers are sometimes reluctant to engage with cyber security professionals for fear that they will 'shoot the baby' and as one interviewee said, they 'don't want security to say no'. Invoking the 'lens of care' that Kocksch et al. [31] bring to their study of security and software developers, this is the point where we can clearly see the need for empathy in order to bridge the divide between software developers and cyber security professionals. Both groups need to find a way to understand each other.

Passos et al. [50] have studied the belief systems in software organisations and find that both past experiences and organisational culture influence practices. When organisational culture is compatible with developers' beliefs, it serves to strengthen those beliefs. When the two conflict, the cultural norms of the developers may be weakened by disagreement. However, strong developer beliefs associated with positive results can spread to other developer teams and become part of the organisational culture.

8 CO-CREATING SECURITY WITH SOFTWARE DEVS

As we have demonstrated, using SPT to understand the lifeworld of developers offers us a glimpse of what it feels like to be in their position. From this perspective we can start to appreciate how security processes, tools and technologies are experienced by them. The next step would be to develop this empathy further and one way could be to bring them closer into the process of designing security interventions. Co-creation has been identified as an effective way of designing for behaviour change, particularly when building relationships is important [15]. In this instance this could take the form of co-creating education and training initiatives and designing an environment that supports both software developers and cyber security practitioners.

Ashenden and Lawrence [7] used co-creation in their action research project to develop better security dialogues. Their design of a two-day workshop with cyber security practitioners acted both as a training programme to help cyber security practitioners facilitate generative dialogues and as a way of gathering rich, qualitative data about the practices that inhibited engagement with security. They note that 'security dialogues can make up the shortfall between the security process and the real-world context in which it operates'.

We would seek to build on this idea and develop workshops that bring together cyber security practitioners and software developers to work together on activities that would address the following issues in a specific organisational context:

- Clarifying the affiliation structures within a project, articulating whether the project is open source or proprietary or a blend. Understanding the implications for developers to build their portfolios. Discussing the risks of the chosen affiliation structure both on the project and the potential risk of insider threat.

- Understanding the physical collaboration and communication mechanisms and constraints – for example how distributed is the team geographically, what time zone they are working in, what the balance will be between home working and office working. Ensuring that physical communication and collaboration mechanisms are secure.
- A negotiation session between cyber security professionals and software developers to identify tools that balance security with usability for communication and collaboration in negotiation with software developers.
- Jointly identifying any barriers to collaboration and communication that might exist physically, socially, or technically arising from security requirements.
- Co-creating security processes that allow software developers to optimise the division of their work between coding and collaboration for their productivity.
- Designing a code review process that aligns cybersecurity professionals' need for security with software developers' desire for quality code.

By using SPT as the foundation for designing the workshop we focus on group practices rather than individual behaviours. In this way we are more likely to avoid an adversarial environment that attributes blame. Instead, by structuring the workshop around the exploration and shaping of practices we believe we will facilitate constructive and generative dialogue. By co-creating security practices we would start to ensure that responsibility for security is shared by cybersecurity professionals and software developers in practice as well as in the process.

9 CONCLUSIONS AND FUTURE RESEARCH

With a move to agile, continuous integration/continuous delivery, DevOps and DevSecOps, software development cycles are shorter and faster. Blending cyber security processes with those of software development is now more important than ever but is proving difficult to achieve. There is increased interest at the moment in how to ensure secure software development but initiatives tend to focus on tools, technology and processes without considering the social practice of software development or the relationship between cyber security professionals and software developers.

We have made a case for using SPT to understand the lifeworld of software developers in order to be able to better blend cyber security with software development. Our rationale for taking SPT as our approach stemmed from existing research which suggests that to change behaviour we need to understand why people do what they do from their own perspective – in this case the perspective of software developers. SPT also enabled us to build on the current turn towards examining practices in software development research.

SPT allows us to start to understand security as it happens in the 'in-between' [31]; those liminal spaces between documented processes and real-world actions, between the cyber security practitioner and the software developer. We need to start building the type of bridges between the two communities that C.P. Snow envisaged between art and science; it should not be one or the other that wins but it should be about learning from each other [32]. Using SPT we have shown that we have a way to consider the 'interplay of structure and situated practice' [52] and the impact this has on secure software development. Through the framework of materials, meanings and competences we have a way of helping cyber security practitioners to understand why software developers do what they do and a starting point for understanding how to co-create solutions. By moving the focus away from individual responsibility, we avoid a zero-sum game where all the potential blame lies with the software developer, and encourage a shared responsibility where both the software developer and the cyber security practitioner can win.

This is in no way a comprehensive study of the social practices of software developers as they relate to security; the next step is to carry out a much larger scale study. We believe though that we've provided a proof of concept that demonstrates the potential benefits of using SPT to develop practices that support secure software development effectively. Our next step will be to develop and trial a workshop that brings software developers and cyber security practitioners together to co-create solutions. This would build on other successful initiatives that take this approach. The aim is to facilitate generative dialogue between the two communities.

Future research will focus on studies that cross cyber security and software development communities. There is a need for ethnographic studies not just in software development but in DevSecOps teams specifically. Two topics that warrant more in-depth focus from cyber security researchers because of the impact that they could have on risk profiles are the code review process and the affiliation and identity of software developers.

ACKNOWLEDGMENTS

This work was funded by the Centre for Research and Evidence on Security Threats (ESRC award ES/N009614/1) and EPSRC award EP/P01166711. The authors would like to thank Dr Helen Thackray for her help with the literature review.

REFERENCES

[1] 2020. Enabling Open Innovation & Collaboration | The Eclipse Foundation. https://www.eclipse.org/ [Online; accessed 14-Dec-2020].

[2] 2020. EuroPython 2019 Sprints & EuroPython 2019, Basel, Switzerland, 8-14 July 2019. https://ep2019.europython.eu/events/sprints/ [Online; accessed 14-Dec-2020].

[3] Yasemin Acar, Christian Stransky, Dominik Wermke, Charles Weir, Michelle L Mazurek, and Sascha Fahl. 2017. Developers need support, too: A survey of security advice for software developers. In *2017 IEEE Cybersecurity Development (SecDev)*. IEEE, 22–26.

[4] Adam Alami, Marisa Leavitt Cohn, and Andrzej Wąsowski. 2019. Why does code review work for open source software communities?. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 1073–1083.

[5] Edward Amoroso. 2018. Recent progress in software security. *IEEE Software* 35, 2 (2018), 11–13.

[6] Axelle Apvrille and Makan Pourzandi. 2005. Secure software development by example. *IEEE Security & Privacy* 3, 4 (2005), 10–17.

[7] Debi Ashenden and Darren Lawrence. 2013. Can we sell security like soap? A new approach to behaviour change. In *Proceedings of the 2013 New Security Paradigms Workshop*. 87–94.

[8] Debi Ashenden and Darren Lawrence. 2016. Security dialogues: Building better relationships between security and business. *IEEE Security & Privacy* 14, 3 (2016), 82–87.

[9] Adam Beautement, M Angela Sasse, and Mike Wonham. 2008. The compliance budget: managing security behaviour in organisations. In *Proceedings of the 2008 New Security Paradigms Workshop*. 47–58.

[10] Sarah Beecham, Nathan Baddoo, Tracy Hall, Hugh Robinson, and Helen Sharp. 2008. Motivation in Software Engineering: A systematic literature review. *Information and software technology* 50, 9-10 (2008), 860–878.
[11] Gerry Gerard Claps, Richard Berntsson Svensson, and Aybüke Aurum. 2015. On the journey to continuous deployment: Technical and social challenges along the way. *Information and Software technology* 57 (2015), 21–31.
[12] Robert Davison, France Bélanger, Manju Ahuja, Mary Beth Watson-Manheim, J Alberto Espinosa, William DeLone, and Gwanhoo Lee. 2006. Global boundaries, task processes and IS project success: a field study. *Information Technology & People* (2006).
[13] Hans de Bruijn and Marijn Janssen. 2017. Building cybersecurity awareness: The need for evidence-based framing strategies. *Government Information Quarterly* 34, 1 (2017), 1–7.
[14] Breno B Nicolau de França, Helvio Jeronimo, and Guilherme Horta Travassos. 2016. Characterizing DevOps by hearing multiple voices. In *Proceedings of the 30th Brazilian symposium on software engineering*. 53–62.
[15] Darshan Desai. 2009. Role of relationship management and value co-creation in social marketing. *Social Marketing Quarterly* 15, 4 (2009), 112–125.
[16] Yvonne Dittrich. 2016. What does it mean to use a method? Towards a practice theory for software engineering. *Information and Software Technology* 70 (2016), 220–231.
[17] Remo Eckert, Matthias Stuermer, and Thomas Myrach. 2019. Alone or Together? Inter-organizational affiliations of open source communities. *Journal of systems and software* 149 (2019), 250–262.
[18] David Evans. 2011. Consuming conventions: sustainable consumption, ecological citizenship and the worlds of worth. *Journal of Rural Studies* 27, 2 (2011), 109–115.
[19] Ivan Flechais, M Angela Sasse, and Stephen MV Hailes. 2003. Bringing security home: a process for developing secure and usable systems. In *Proceedings of the 2003 workshop on New security paradigms*. 49–57.
[20] Taghi Javdani Gandomani and Mina Ziaei Nafchi. 2016. Agile transition and adoption human-related challenges and issues: A Grounded Theory approach. *Computers in Human Behavior* 62 (2016), 257–266.
[21] Anthony Giddens. 1984. *The constitution of society: Outline of the theory of structuration*. Univ of California Press.
[22] Git. 2020. –fast-version-control. https://git-scm.com/ [Online; accessed 14-Dec-2020].
[23] GitHub. 2020. GitHub - tensorflow/tensorflow: An Open Source Machine Learning Framework for Everyone. https://github.com/tensorflow/tensorflow [Online; accessed 14-Dec-2020].
[24] Daniel Graziotin, Fabian Fagerholm, Xiaofeng Wang, and Pekka Abrahamsson. 2018. What happens when software developers are (un) happy. *Journal of Systems and Software* 140 (2018), 32–47.
[25] Bente Halkier. 2001. Risk and food: environmental concerns and consumer practices. *International journal of food science & technology* 36, 8 (2001), 801–812.
[26] Georg Holtz. 2014. Generating social practices. *Journal of Artificial Societies and Social Simulation* 17, 1 (2014), 17.
[27] Jon Iden and Bendik Bygstad. 2018. The social interaction of developers and IT operations staff in software development projects. *International Journal of Project Management* 36, 3 (2018), 485–497.
[28] IEEE Spectrum. 2020. Interactive: The Top Programming Languages 2019 - IEEE Spectrum. https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2019 [Online; accessed 14-Dec-2020].
[29] Jack Ingram, Elizabeth Shove, and Matthew Watson. 2007. Products and practices: Selected concepts from science and technology studies and from social theories of consumption and practice. *Design issues* 23, 2 (2007), 3–16.
[30] Eirini Kalliamvakou, Daniela Damian, Kelly Blincoe, Leif Singer, and Daniel M German. 2015. Open source-style collaborative development practices in commercial projects using GitHub. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 1. IEEE, 574–585.
[31] Laura Kocksch, Matthias Korn, Andreas Poller, and Susann Wagenknecht. 2018. Caring for IT Security: Accountabilities, Moralities, and Oscillations in IT Security Practices. *Proceedings of the ACM on Human-Computer Interaction* 2, CSCW (2018), 1–20.
[32] Lawrence M Krauss. 2009. CP Snow in New York. *Scientific American* 301, 3 (2009), 32–34.
[33] Sherlock A Licorish and Stephen G MacDonell. 2014. Understanding the attitudes, knowledge sharing behaviors and task performance of core developers: A longitudinal study. *Information and Software Technology* 56, 12 (2014), 1578–1596.
[34] Sherlock A Licorish and Stephen G MacDonell. 2017. Exploring software developers' work practices: Task differences, participation, engagement, and speed of task resolution. *Information & Management* 54, 3 (2017), 364–382.
[35] Linux Foundation. 2020. The Linux Foundation – Supporting Open Source Ecosystems. https://www.linuxfoundation.org/ [Online; accessed 14-Dec-2020].
[36] Welder Pinheiro Luz, Gustavo Pinto, and Rodrigo Bonifácio. 2019. Adopting DevOps in the real world: A theory, a model, and a case study. *Journal of Systems and Software* 157 (2019), 110384.
[37] Laura MacLeod, Michaela Greiler, Margaret-Anne Storey, Christian Bird, and Jacek Czerwonka. 2017. Code reviewing in the trenches: Challenges and best practices. *IEEE Software* 35, 4 (2017), 34–42.
[38] Greg Marsden, Caroline Mullen, Ian Bache, Ian Bartle, and Matt Flinders. 2014. Carbon reduction and travel behaviour: Discourses, disputes and contradictions in governance. *Transport Policy* 35 (2014), 71–78.
[39] Lydia Martens. 2012. Practice 'in talk'and talk 'as practice': Dish washing and the reach of language. *Sociological Research Online* 17, 3 (2012), 103–113.
[40] Matlab. 2020. MathWorks. https://uk.mathworks.com/products/matlab.html [Online; accessed 14-Dec-2020].
[41] Christoph Matthies, Johannes Huegle, Tobias Dürschmid, and Ralf Teusner. 2019. Attitudes, beliefs, and development data concerning agile software development practices. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*. IEEE, 158–169.
[42] Petra Sylvia Meier, Alan Warde, and John Holmes. 2018. All drinking is not equal: how a social practice theory lens could enhance public health research on alcohol and other health behaviours. *Addiction* 113, 2 (2018), 206–213.
[43] Mercurial SCM. 2020. Work easier, Work faster. https://www.mercurial-scm.org [Online; accessed 14-Dec-2020].
[44] Andre Meyer, Earl T Barr, Christian Bird, and Thomas Zimmermann. 2019. Today was a good day: The daily life of software developers. *IEEE Transactions on Software Engineering* (2019).
[45] Nabil M Mohammed, Mahmood Niazi, Mohammad Alshayeb, and Sajjad Mahmood. 2017. Exploring software security approaches in software development lifecycle: A systematic mapping study. *Computer Standards & Interfaces* 50 (2017), 107–115.
[46] Håvard Myrbakken and Ricardo Colomo-Palacios. 2017. DevSecOps: a multivocal literature review. In *International Conference on Software Process Improvement and Capability Determination*. Springer, 17–29.
[47] Tosin Daniel Oyetoyan, Daniela Soares Cruzes, and Martin Gilje Jaatun. 2016. An empirical study on the relationship between software security skills, usage and training needs in agile settings. In *2016 11th International Conference on Availability, Reliability and Security (ARES)*. IEEE, 548–555.
[48] Andy Ozment and Stuart E Schechter. 2006. Milk or wine: does software security improve with age?. In *USENIX Security Symposium*, Vol. 6.
[49] Tero Päivärinta and Kari Smolander. 2015. Theorizing about software development practices. *Science of Computer Programming* 101 (2015), 124–135.
[50] Carol Passos, Manoel Mendonça, and Daniela S Cruzes. 2014. The role of organizational culture in software development practices: a cross-case analysis of four software companies. In *2014 Brazilian Symposium on Software Engineering*. IEEE, 121–130.
[51] Olgierd Pieczul, Simon Foley, and Mary Ellen Zurko. 2017. Developer-centered Security and the Symmetry of Ignorance. In *Proceedings of the 2017 New Security Paradigms Workshop*. 46–56.
[52] Andreas Poller, Laura Kocksch, Sven Türpe, Felix Anand Epp, and Katharina Kinder-Kurlanda. 2017. Can security become a routine? A study of organizational change in an agile software development group. In *Proceedings of the 2017 ACM conference on computer supported cooperative work and social computing*. 2489–2503.
[53] Andreas Reckwitz. 2002. Toward a theory of social practices: A development in culturalist theorizing. *European journal of social theory* 5, 2 (2002), 243–263.
[54] Kalle Rindell, Sami Hyrynsalmi, and Ville Leppänen. 2018. Aligning security objectives with agile software development. In *Proceedings of the 19th International Conference on Agile Software Development: Companion*. 1–9.
[55] Scikit-Learn. 2020. scikit-learn: machine learning in Python. https://scikit-learn.org/stable/index.html. [Online; accessed 14-Dec-2020].
[56] Helen Sharp, Nathan Baddoo, Sarah Beecham, Tracy Hall, and Hugh Robinson. 2009. Models of motivation in software engineering. *Information and software technology* 51, 1 (2009), 219–233.
[57] Helen Sharp, Yvonne Dittrich, and Cleidson RB De Souza. 2016. The role of ethnographic studies in empirical software engineering. *IEEE Transactions on Software Engineering* 42, 8 (2016), 786–804.
[58] Helen Sharp, Hugh Robinson, and Mark Woodman. 2000. Software engineering: community and culture. *IEEE Software* 17, 1 (2000), 40–47.
[59] Elizabeth Shove. 2010. Beyond the ABC: climate change policy and theories of social change. *Environment and planning A* 42, 6 (2010), 1273–1285.
[60] Elizabeth Shove, Mika Pantzar, and Matt Watson. 2012. *The dynamics of social practice: Everyday life and how it changes*. Sage.
[61] Mario Silic and Andrea Back. 2016. The influence of risk factors in decision-making process for open source software adoption. *International Journal of Information Technology & Decision Making* 15, 01 (2016), 151–185.
[62] Fiona Spotswood, Tim Chatterton, Alan Tapp, and David Williams. 2015. Analysing cycling as a social practice: An empirical grounding for behaviour change. *Transportation research part F: traffic psychology and behaviour* 29 (2015), 22–33.
[63] Damian Andrew Andrew Tamburri, Fabio Palomba, and Rick Kazman. 2019. Exploring community smells in open-source: An automated approach. *IEEE Transactions on software Engineering* (2019).

[64] Nora Tomas, Jingyue Li, and Huang Huang. 2019. An empirical study on culture, automation, measurement, and sharing of DevSecOps. In *2019 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*. IEEE, 1–8.

[65] Amber van der Heijden, Cosmin Broasca, and Alexander Serebrenik. 2018. An empirical perspective on security challenges in large-scale agile software development. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. 1–4.

[66] Georg Von Krogh, Stefan Haefliger, Sebastian Spaeth, and Martin W Wallin. 2012. Carrots and rainbows: Motivation and social practice in open source software development. *MIS quarterly* (2012), 649–676.

[67] Shao-Fang Wen, Mazaher Kianpour, and Stewart Kowalski. 2019. An empirical study of security culture in open source software communities. In *2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. IEEE, 863–870.

[68] Laurie Williams. 2018. Continuously integrating security. In *Proceedings of the 1st International Workshop on Security Awareness from Design to Deployment*. 1–2.

[69] Glenn Wurster and Paul C Van Oorschot. 2008. The developer is the enemy. In *Proceedings of the 2008 New Security Paradigms Workshop*. 89–97.

[70] Mary Ellen Zurko and Richard T Simon. 1996. User-centered security. In *Proceedings of the 1996 workshop on New security paradigms*. 27–33.