EPIC2020
Ethnographic Praxis in Industry Conference

CASE STUDY

Software Quality and Its Entanglements in Practice

JULIA PRIOR, *University of Technology Sydney*
JOHN LEANEY, *University of Technology Sydney*

Effective software quality assurance in large-scale, complex software systems is one of the most vexed issues in software engineering, and, it is becoming ever more challenging. Software quality and its assurance is part of software development practice, a messy, complicated and constantly shifting human endeavor.

What emerged from our immersive study in a large Australian software development company is that software quality in practice is inextricably entangled with the phenomena of productivity, time, infrastructure and human practice. This ethnographic insight --- made visible to the organization and its developers via the rich picture and the concept of entanglements--- built their trust in our work and expertise. This led to us being invited to work with the software development teams on areas for change and improvement and moving to a participatory and leading role in organizational change.

Keywords: ethnography, entanglements, rich_pictures, software_development

INTRODUCTION AND CONTEXT

Effective software quality assurance in large-scale, complex software systems is one of the most vexed issues in software engineering. Today's software systems provide sophisticated functionality that was not even imaginable a couple of decades ago – and assuring quality in these increasingly capabable, adaptive and connected systems is becoming ever more challenging (Mistrik et al. 2016). Software quality and its assurance is part of software development practice – a messy, complicated and constantly shifting human endeavor.

We were drawn to the participant organization – a software development company – as a fascinating and promising place to explore challenges in quality assurance through the lived experience of professional software developers. Its flagship software product, Connect (a pseudonym), is an extraordinarily large and complex software system used by thousands of customers in dozens of countries across the globe. The organization's profound knowledge and experience in the industry it has served for over two decades, and a continual development approach, ensures that Connect's functionality becomes more and more advanced every year.

At the time of the fieldwork, there were just over two hundred software developers working collaboratively in a dozen software product development teams. They were based primarily in the head office in Australia, with most responsible for different modules of functionality in Connect, and a couple of smaller teams developing separate products that interacted directly with Connect.

Keeping Connect performing reliably for its tens of thousands of users necessitates robust software development, quality assurance and work management processes. The company has developed a high quality ethic around its development of software over many years. This has come about by the continuing discussions around quality and productivity that permeate all teams. The quality assurance processes it has in place make for an extremely robust product that is recognised as such by the industry that it serves (Prior 2011). Nonetheless, challenges to the reliability of these quality processes were posed by the consistent growth of both Connect and the number of new developers unfamiliar with the organization and its complex systems, processes and practices.

This case study is based on the ethnographic work we carried out when the first author spent a six-month sabbatical working full-time in the organization.

RICH PICTURES

As this is ethnographic work, there is of course thick, rich data. We needed to make friends with all of this data, to manage and analyse it without becoming overwhelmed. It became apparent as data was collected that some diagrammatic means of representing the relationships discovered from analysis was important. For a large system, or complex environment, diagrams "encourage holistic rather than reductionist thinking about a situation" (Checkland 2000).

Rich pictures are compilations of drawings, pictures, symbols and text, that show relationships, connections, influence, processes, as well as characters and characteristics, points of view, prejudices and preferences.

We chose Checkland's rich pictures as they are not hierarchical, can be used to extend analysis via the Soft Systems Methodology (SSM) and both authors were familiar with the SSM and had used it in the past. SSM shares similar theoretical underpinnings to an ethnographic approach. The most notable one is the lack of belief in a universal theory, or driving system, for an organization.

The rich pictures we created with developers proved particularly useful for:

- Exploring and identifying aspects and perspectives to include in mapping a system or situation
- Capturing structure and process of what is happening in a situation, as well as people's feelings, values and perspectives
- Fostering communication with others about a situation
- Developing a shared understanding of a situation or initiative as a group
- Motivating further discussion, learning and/or action
- The unanticipated effect of the rich picture was the deeper engagement of the developers with our work, which helped build their trust in us, and appreciation of our research.

BUILDING A RICH PICTURE OF THE ORGANISATION

A Rich Picture

The first author started the rich picture by using Post-Its, as she could place and move them around easily.

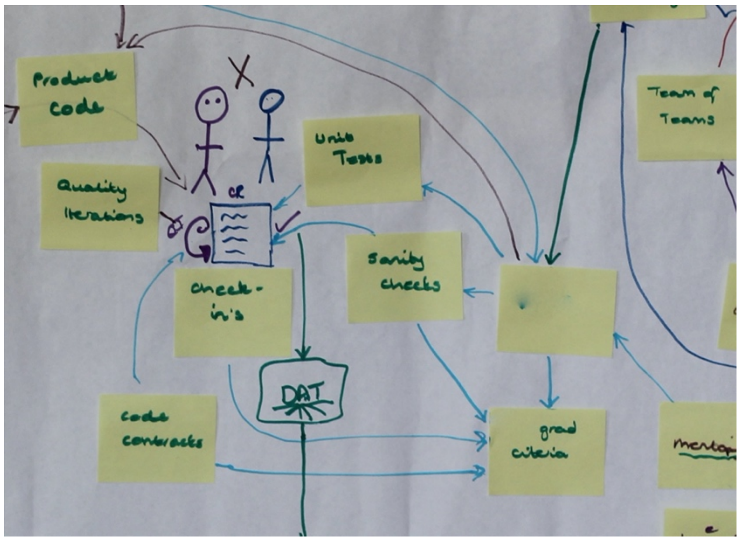

Figure 1. First rich picture composed of Post-its.

This piece of the first, rather rough, rich picture represents activity around the testing process: we see a couple of new developers writing some unit tests and using the automated test system, DAT. We also see their code going through a couple of iterations to improve quality. These iterations are about the code being refined, and eventually the code being checked into the code database. In their development process, TestFirst is a fundamental approach in which the tests are written before the functional code. It directs thinking towards outcomes, and how they will be tested to demonstrate correctness.

Our observations were that the more experienced developers will talk about the essential use of TestFirst – as a design approach, but also for investigating and fixing defects:

> "Let's write the test first, and then see if we need to change the others [unit tests]."

In the picture, we see the developers performing tests, driven by quality needs. In tension, they are also driven by the need for progress, as expressed by the Post-It labelled 'check-in's'. A check-in occurs when a developer uploads their final tested and peer-reviewed new or revised code to the main codebase. One can also see the interactions with new and trainee developers in the company, shown by the Post-Its, 'sanity checks', 'grad criteria' and the redacted Post-It. A 'sanity check' is a brief run through of the functionality of the code to establish that it works more or less as expected; 'grad criteria' refers to the set of measures that a new developer must meet before they can graduate from, or complete, their probationary training – these include a minimum number of check-in's and sanity checks, for example.

For many developers, these training interactions cause tension in achieving productivity, as demonstrated by the following quote:

> "All the senior devs. are already busy doing what they are working on at the moment. It's kind of like, they've got their work and they have to teach other people at the same time. So the priority for senior dev. is, of course, their current work."

In summary, what we were seeing more clearly via the Rich Picture was an understanding that was broader (more of the interactions within and between teams and the influences on developers' behavior) and deeper (more subtle interactions).

A Richer Rich Picture

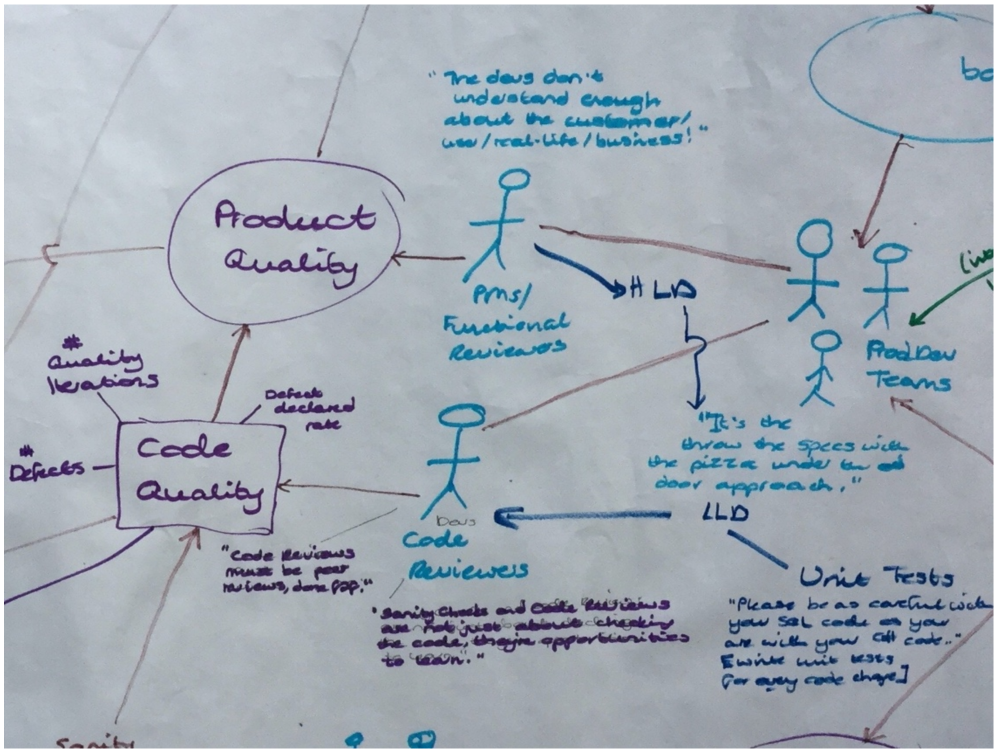
"The devs don't understand enough about the customer/ user/real-life/business!"
Product Quality
PMs/ Functional Reviewers
HLD
ProdDev Teams
"It's the throw the specs with the pizza under the door approach."
LLD
Unit Tests
"Please be as careful with your SQL code as you are with your C# code.." [write unit tests for every code change]
# Quality Iterations
Defects declared rate
# Defects
Code Quality
"Code Reviews must be peer reviews, done f2f."
Devs
Code Reviewers
"Sanity Checks and Code Reviews are not just about checking the code, they're opportunities to learn."

Figure 2. Richer picture excerpt, hand drawn.

A couple of weeks later, the authors re-drew the rich picture as a solely hand-drawn diagram. Even the small Post-It notes proved to be too large, and they didn't allow for as much flexibility or creativity as we wanted. Adding more connecting lines, colors, some drawings, free-form shapes and labels helped us to build richness into the diagram.

In this version of the rich picture, there are more characters: developers who review code for correctness, product managers who manage the requirements of the product being developed and development teams. High level and low-level design processes are now included, using the acronyms HLD and LLD. These acronyms are commonly used within the company and also save space on the picture. We have been able to group items into larger umbrella items, including Product Quality and Code Quality.We have added quotes to the rich picture, representing the sort of attitudes and beliefs that are held by people in various roles.

The tension around quality and productivity can be seen in the following quotes.

"Code Reviews must be peer reviews done face to face" is commenting on the tension between the effectiveness of code reviews as learning experiences for improving code quality in tension with the time taken to do reviews.

> "Sanity Checks and Code Reviews are not just about checking the code, they are opportunities to learn."

This especially relates to new developers.

Near the bottom of this diagram, and the stick figure labelled Code Reviewers, and linked to Unit Tests there is something that the first author heard one developer say to another during a code review; explaining that TestFirst should be applied to every sort of code change, they went on, "Please be as careful with your SQL code as you are with your C# code!" (SQL code is used for accessing databases, whil C# code is used for implementing the function of the system).

For a business based largely around very analytical software developers, spending most of their days writing code, they rely on talking to each other. This is especially true around the issues of quality and time.

An Even Richer Rich Picture

In an extensive open-plan environment dominated by large monitors and powerful desktops, there was very little paper around.

Our rich picture was on a large piece of paper, about A2- size. Because she wanted to keep it in sight and in mind, the first author left it laid out on the empty desk next to hers, for several weeks.

This provoked developers who came over to her desk to talk to her, and developers who were just passing, to comment on it and ask questions about it. It gave her unexpected opportunities to discuss what it represented and meant with the developers. Our understanding and interpretation of their work was made obvious to them, in a way that written text in a report, that they probably wouldn't read, would not have. It gave them a way to directly engage with and contribute to our fieldwork. Further, it gave us a unique way to validate our understanding of their situation with them, while the first author was there full-time.

This excerpt is the same section of the rich picture as that in the previous slide, but it is from several weeks later. It is a richer picture, in that it has had a lot of extra things added to it: more quotes, more interconnecting lines, more processes and text. Notably, the interconnections with design, side effects and product quality.

"It's the throw the specs with the pizza under the door approach", commenting on the concerns of the relationship between product managers and developers in what happens between the HLD (for which the product managers are primarily responsible) and the LLD (for which the developers are primarily responsible).

> "The devs don't understand enough about the customer/user/real-life business!",

relating to the concerns of the product managers.

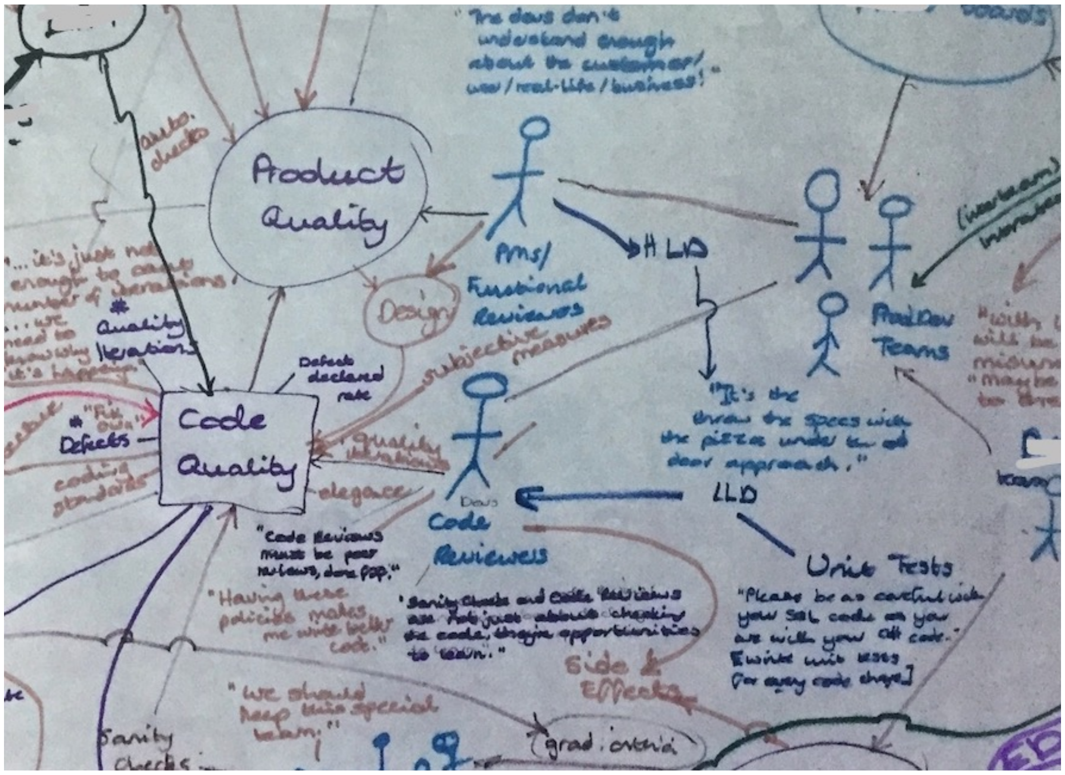
"The devs don't understand enough about the customer/user/real-life/business!"
Product Quality
auto. checks
PMs/ Functional Reviewers
HLD
ProdDev Teams
(inter-team) interfaces
"...it's just not enough to count number of iterations ... we need to know why it's happening."
# Quality Iterations
Design
subjective measures
Defect declared rate
"It's the throw the specs with the pizzas under the door approach."
# "Fix own" Defects
Code Quality
quality iterations
coding standards
elegance
Devs
Code Reviewers
LLD
Unit Tests
"Code Reviews must be peer reviews, done f2f."
"Having these policies makes me write better code."
"Sanity Checks and Code Reviews are not just about checking the code, they're opportunities to learn."
"Please be as careful with your SQL code as you are with your C# code." [write unit tests for every code change]
Side Effects
"We should keep this special team."
Sanity checks
grad. criteria

Figure 3. Even richer picture excerpt.

Code quality is now embellished, and the associated processes are acknowledged by comments such as, "Having these policies makes me write better code".

On Quality Iterations, "It's just not enough to count the number of iterations … we need to know why it is happening." How many iterations (loops, occurrences) it takes to improve quality to an acceptable standard is not useful without understanding why it is happening. The quality iteration count is fast, but not necessarily useful to improving quality.

Over the next month or so, as we kept adding to the rich picture, our ethnographic understanding of the situation and the developers' software quality practices continued to deepen.

The Complete Rich Picture

Below is the whole rich picture as it was at the end of the fieldwork period.

The phenomena of Software Quality, Productivity, People, Processes and Practices that emerged from the fieldwork are highlighted. as well as the overarching layers of communications, education, and Time.

What emerged was the components and people of the company demonstrably in rich, dynamic relationships.

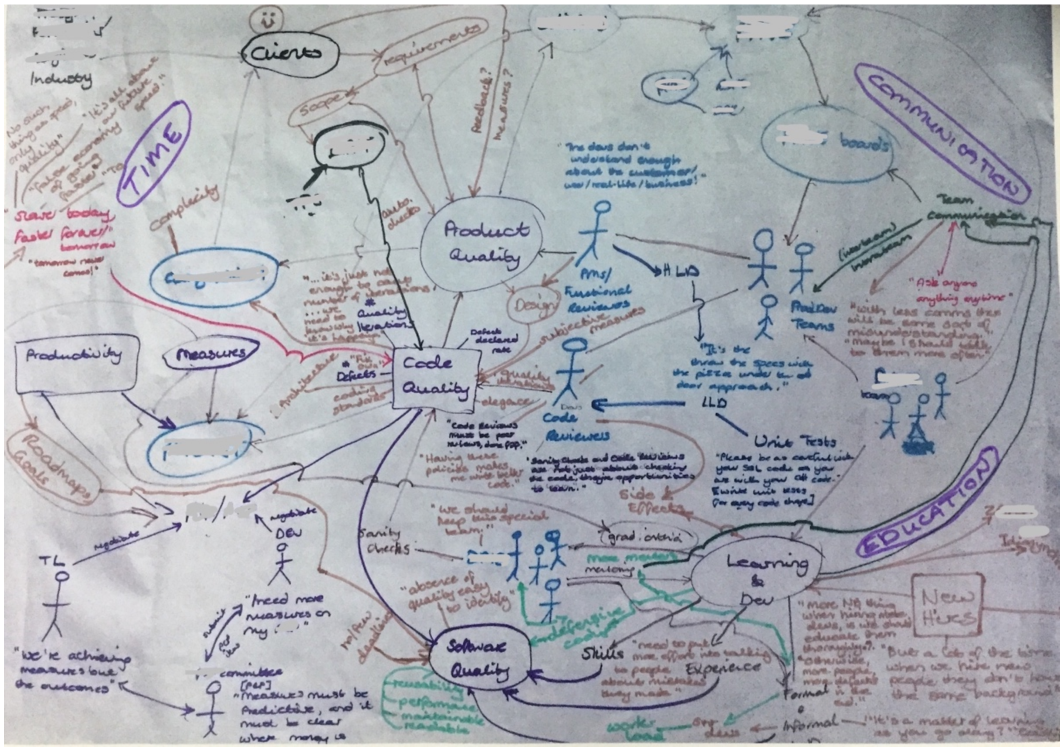
Industry
Clients
requirements
Scope
Feedback?
measures?
TIME
"No such thing as speed, only quality"
"It's all about our future speed!"
"False economy of going faster"
"Save today, faster forever"
complexity
Product Quality
Design
"The devs don't understand enough about the customer/ user/real-life/business!"
PMs/ Functional Reviewers
HLD
COMMUNICATION
boards
Team communication
"Ask anyone anything anytime"
Position Teams
Productivity
Measures
Code Quality
Defects
coding standards
elegance
Code Reviewers
LLD
Unit Tests
"It's the throw the specs with the pizza under the door approach."
"Code Reviews must be peer reviews, done F2F."
"Having these policies makes me write better code."
Side Effects
Roadmaps
Goals
negotiate
DEV
TL
"We're achieving measures but not the outcomes"
"Need more measures on my ..."
Sanity checks
"We should keep this special team!"
grad. criteria
more meetings
"absence of quality easy to identify"
Learning & Dev
EDUCATION
New Hires
Software Quality
Skills
Experience
Formal
Informal
workload
reusability
performance
maintainable
readable
"It's a matter of learning as you go along?"

Figure 4. Complete rich picture.

TOWARDS ENTANGLEMENTS

A number of patterns, themes and connections emerged from analysis of the rich picture and the thick data that it represented. We would be writing about productivity and find that we were also talking about quality, and also people and time. It seemed impossible to talk about these elements separately. Looking for terms and ideas to express the strong bonds represented in the rich picture led to discovering previous work on entanglement, in particular Scott & Orlikowski (2014) and their use of Barad's (2007) notion of entanglement.

Scott and Orlikowski (2014)'s approach, which is based on Barad's agential realism theory of knowledge and being, gave a legitimacy to, as well as a way of articulating, the entanglements that emerged from our study. Scott and Orlikowski (2014) define entanglement as ``the inseparability of meaning and matter." These authors cite Barad (2007, p.ix), who explains,

> ``To be entangled is not simply to be intertwined with another, as in the joining of separate entities, but *to lack an independent, self-contained existence…* " (our emphasis).

Barad (2007, p.ix) continues, "... Existence is not an individual affair. Individuals do not pre-exist their interactions; rather, individuals emerge through and as part of their entangled intra-relating." These individuals are not necessarily humans, but include non-humans, objects or phenomena involved in the situation we are trying to understand. Each of these is not a discrete factor in reality, an independent object with "independently determinate boundaries and properties" (Barad 2007, p.33).

Barad (2007) describes these things as "phenomena" and sees them as relational, with their agency residing in that relating, rather than agency as something that resides in an individual thing. This is similar to Suchman (2007)'s understanding of agency. Barad goes further: when defining phenomena, firstly, in referring to phenomena as agencies, and secondly, and most significantly, that their existence and properties arise through their intra-acting with one another.

Intra-action differs from the notion of 'interaction'. 'Interaction' assumes that there are independent objects, or phenomena, each with their own agency, that precede or pre-exist their interaction or relating. Intra-action, however, is the mutual constitution of entangled phenomena: these phenomena come into being through their intra-actions.

Barad (2007) considers phenomena and their continual intra-actions to be constitutive of reality. Entanglements are dynamic, they are already made, as well as always in the making (Suchman 2012).

We realized that entanglement meant, in the first instance, that any attempt to understand the company in terms less than the whole rich picture, its elements and interactions, would lead to the understanding of a different company. And, in fact, a fictitious company.

A shift in our interpretation and representation of the local software development endeavor occurred in the move from initially exploring software development as a human endeavor, and as situated action (Suchman 2007), to a post-human perspective of entanglements in the local context. In the latter, humans and non-humans, their intra-actions and agencies, are seen as being equal participants, active in the ongoing, dynamic entanglements from which phenomena such as quality, productivity and practice come into recognizable being.

Viewing local software development as relating phenomena, and exploring the nuances of their intra-actions, makes entanglement a meaningful way of discussing the reality of software development practice. The entanglements of people's actions with phenomena such as quality, productivity and time, is a characteristic of the perpetually generated context in which the design and development of complex software is accomplished.

"Slower today, faster tomorrow"

"Slower today, faster tomorrow" is one of the company's software development mantras. Experienced developers talk frequently about what this mantra means: if developers spend time and effort on assuring quality in their original code, then all of the developers will be more productive in the longer term. In other words, they will spend most of their time adding new functionality to the codebase, rather than spending time fixing defects that have been discovered in previously released (deployed) code.

> "So previously I would quite often talk about quality in the context of the speed quality trade-off… Because having quality gives you speed. So slower today and faster forever. So I've really toned back on my attempt to be fast and I've really just thought about how we can have quality instead. Because I don't even need to think about speed, I just get it automatically. So, for me quality is the ability for what we do now to have long lasting positive outcomes on the goals that we're trying to

> achieve. So if we produce something that may take a little bit of time but in the long run saves us a lot of time then that was the right thing to do that, it's good quality."

Code that is not written well, that does not adhere to the company's coding standards, for example, is difficult to maintain and change later, and this in turn may lead to further defects and decreased productivity:

> "If you don't write code in a good way, developers will spend more time reading and changing it, which will result in more waste at the end. It's all about our future speed."
> "Particularly, I'm a software developer, so the quality for developers means we should write very elegant code. So, probably, for example, if we write, if I write, very dodgy code, there's a high possibility that my code would break something of the software or [worse] result in an unhappy client. Then they will lodge another incident and more repetitive work. So yeah, that quality [coughs] means, for me, is more work, more time—yeah, less productivity as well."

Increased defects in the code means that at some stage, the software will not work as expected, or worse, will crash while the customer is using it. Developer time will then need to be spent on fixing those defects, rather than spending that time on developing – and delivering – new features in the software.

> "I mean, when we say we should deliver good qualities, there's always another thing called time frame. To deliver the good quality software, definitely we need more time. But normally people at [the company] got overloaded easily because if we got too much work, unfortunately we got too much defect as well."

The above quote highlights the tension between a stated value of spending time on quality, and the experience of time being scarce. However, spending substantial time taking action to improve the quality of the code is potentially detrimental to throughput and thus productivity.

After a developer at a daily team stand-up meeting said, in an ironic tone,

> "Slower today, faster tomorrow!"

one comedian from the Productivity team responded, "but tomorrow never comes!"

They were reminding the team that one can spend forever getting something closer to perfect, or 'high quality', but, taken to the extreme, the work will never be delivered. This concern about not delivering 'enough' is not often explicitly articulated, but it is alluded to frequently and underlies much of the developers' everyday practices, behavior and decisions.

The issues of software quality and productivity in practice are about people's practices in time and over time. Decisions that the developers continually have to make include: what should we spend time on? how much time should we spend on what kind of work? should we spend more time on this work for better quality? if our throughput is higher in the short-term are we more productive in the long-term? how and where are people spending their time? and so on.

> "Because I find a lot of the time when something goes wrong it's because - not that someone just did something silly, it's often that we didn't consider something. That if we thought about it for maybe half an hour longer, we could have."

This apparently simple, short phrase "Slower today, faster tomorrow!", frequently quoted by developers in discussions about quality, is really about the ongoing entanglements of the phenomena of quality, productivity, people's (developers') practices and time. This is illustrated by experienced developers' quotes above from their discussions about this mantra and what 'quality' means at the company. Moreover, it signifies how these phenomena are mutually constitutive: dynamically forming and shaping each other through their continual intra-actions.

Developers Becoming

A "fully-fledged developer" a (human) developer comes into being through ongoing intra-actions with quality, productivity and technical development principles, processes and tools, and with the other developers, over a considerable time. These continual intra-actions generate entanglements within the local development environment and over months, the novice becomes a developer, and over years, they become a fluent, proficient developer. But they are not simply skillful developers; these developers are experts in the entanglements that are particular to the local environment in the participant company.

Producing high-quality enterprise software requires fluent, expert software developers, who have excellent programming skills, as well as the high-level technical skills to work with the automated testing system, sophisticated technology stack and other technological infrastructure used to continually build a complex, but robust, software product such as Connect. A reasonable amount of domain understanding of the logistics industry is also necessary in order to be able to work as an effective developer in this company.

The production of high-quality software requires new hires (developers) to gain both technical competence and fluency in the local codebase; both of these take time. The problem is not simply a concern that is regularly raised by more experienced developers, i.e., that new hires lack the necessary technical skills and expertise to be productive and produce quality code, i.e., code that is maintainable, efficient and thoroughly tested. It is also about the continual trade-off for senior, experienced developers between mentoring, or coaching, of new developers, which takes considerable time, and getting their own development work done in a timely manner:

> "All the senior devs. are already busy doing what they are working on at the moment. It's kind of like, they've got their work and they have to teach other people at the same time. So the priority for senior devs. is, of course, their current work."
>
> "Yeah, and it takes a lot of time as well. Sometimes my manager asks me to be a mentor to the new developers, but I'm already overloaded and then this new people come and ask me, 'How can I do this? How can I do that?'. Sometimes it's really annoying. If I didn't have enough work to do, I'd be more than happy to help them, but the reality is not like that."

These quotes from senior developers in two different teams make the point that the senior developers' most important focus is their 'current' , i.e.,technical work, their design and development work, and this is what they 'should' make their priority, in order to be productive. Mentoring newer developers is an extra, 'really annoying' impost on their time and effort. They do not view mentoring – getting newer developers au fait with the company's development systems, processes and tools – as being as valuable a use of their time and expertise as producing software themselves.

The next quote from a technical team lead refers in part to the assumption that it takes a certain amount of time for any developer to become expert enough to both produce quality code themselves, but also to make assessment of the level of quality of another developer's code:

> "A typical situation, when some developer jumps from junior level to let's say senior level his complexity of work rises, it's natural that number of defects can grow as well but it's kind of natural at first… but I'm going to introduce it and what I'm going to do, I'm going to assign that task to junior developer capability. They need to learn how to do code review because it's - a typical situation… I'm not ready to give them proper final code reviews but at least I think if I give them these intermediate code reviews maybe they can improve their code in quality as well. Because typically it's, I don't know, sometimes it's as long as two years for a developer to gain my trust, so I progress the developer to a capability which allows code reviews."

The aim for a developer's performance is that they become fluent in producing complex code in a collaborative development environment. The more fluent a developer is, the faster they will produce code. And, crucially, they will not only code faster (than a developer who is not as fluent), the code that they will produce will be a higher quality code, without requiring as much iteration or revision. They are therefore more productive as individual developers. Further, they will be able to do code reviews of other developers' code more effectively, which will improve that code's quality. And, if this developer is mentoring a new developer, the less experienced developer will be coached to write higher quality code. So, a secondary effect that is hoped for is that the reviewed developer's approach will change, or at least shift, so that the code they write in future improves also. This impacts productivity in two ways: firstly, it ensures that the particular piece of code in question that is checked in to the code base and eventually released to customers is higher quality, and secondly, fluency of the newer developer improves which, in turn, will greatly improve the code that they produce in daily practice. This will then reduce the need for iterative code reviews at the development end and/or defect fixing at the production end.

Insights from entanglements

The entanglements that are central to our understanding of the local software development situation are those arising from the intra-actions of quality, productivity, people, practices and time. They are not the only ones in the local situation, of course, but these are the ones that emerged most persuasively from our fieldwork and analysis. A researcher's observations in any situation are always limited in various ways, and we can therefore only ever have partial knowledge of it (Haraway 2001).

Perceived software quality and productivity levels unfold as a result of the ongoing intra-actions over time of the developers, their everyday practices, company software quality and productivity principles and processes, development infrastructure and other undefined (in this fieldwork and study) phenomena. Ultimately, levels of quality and productivity in the company depend entirely on the developers' everyday actions that make up their practices. In the end, it is only what the developers do – the actions they perform day-after-day, over long periods of time – that matters. It is the intra-actions of practices (actions), quality, productivity and time as developers continually attempt to balance the demands of quality and productivity, and the efforts given to achieve one or the other, or both, over time that give rise to ongoing entanglements. These entanglements mutually and simultaneously form these phenomena. The phenomena that we identify as practices, quality and productivity are becoming; they continually come into recognizable being through their dynamic entanglements with each other, time and the developers themselves.

These entanglements give us some insight into the subtle complexities of this kind of software development work and the expertise and technical fluency required to carry it out effectively. They also give us a way to describe the continually generated context in which the collaborative design and development of complex software is accomplished.

Somewhat ironically, taking a human-centric stance led us to conclude that quality and productivity in software development requires more than simply focusing on the humans (software developers, in this case). Applying Scott and Orlikowski (2014)'s Baradian approach to reality as ongoing intra-actions of phenomena gave a legitimacy to, and a way of articulating, the dynamic entanglements that emerged from our study. Recognition of these entanglements shifted our perspective from a humanist one, focused on collaborative software development as essentially a human endeavor, to a post-human appreciation of the setting's complexities and the mutual constitution of the phenomena central to our research focus, i.e., the developers and their practices, software quality, productivity and time.

LESSONS LEARNT AND ORGANIZATIONAL IMPACT

The concept of entanglement provides an explanation of the local situation as dynamic, multiple and emergent. Together with the rich picture, it

- presents the nuances of the developers' everyday work practices as they are constituted within the local situation; and
- builds trust with participants, as they see an attempt to capture and express the complexities of software development and their lived experience of it.

This research had a significant impact on the organization and our continuing relationship with it and the developers.

By making our ethnographic work visible through the rich picture, and encouraging participant developers to make suggestions or additions, there is a sense in which they jointly own this work.

The rich picture continues to evolve, and is now explicitly owned and edited by the organization, and used to explore software quality concerns, with our oversight.

The ethnographic insights that we shared with the participants helped us to secure support for, and engagement with, subsequent experiments in mentoring and measurement. The aim is to help them develop practices that will sustain, even increase, software quality, in the face of particular challenges. These are the continual growth in the size, complexity and customer reach of the Connect codebase, and the ongoing hiring of new developers unfamiliar with the organization's quality principles and practices. The work will be characterized by participatory methods and deep collaboration with the developers, enhancing the potential future organizational impacts.

Julia Prior is an Associate Professor in software engineering at UTS. She is a software developer, an ethnographer and a teacher. Her research focuses on understanding the lived experience of professionals developing large, complex software systems and the mechanisms that enable effective collaboration and quality assurance. You can contact her on <julia.prior@uts.edu.au>

John Leaney is an Adjunct Professor in software engineering at UTS. Over the last fifteen years, he has developed expertise in combining qualitative techniques, such as action research and ethnography, with quantitative approaches to provide effective methods for understanding and designing architecture-focussed, complex software systems.

NOTES

Acknowledgements. The authors thank the participant company and its software developers – we greatly appreciate their enthusiasm and generosity in working with us. We thank the Faculty of Engineering and Information Technology at UTS for granting the first author a six-month Professional Experience Program (sabbatical) that enabled us to carry out this work. And we are most grateful to Scott Matter, our EPIC curator, who was very generous with his time and expertise in providing feedback and advice that substantially improved this article.

REFERENCES

Barad, K. 2007. Meeting the universe halfway: Quantum physics and the entanglement of matter and meaning. Duke University Press.

Checkland, Peter. 2000. "Soft systems methodology: a thirty year retrospective." Systems Research and Behavioral Science: 17, S11– S58.

Haraway, Donna. 1988. "Situated Knowledges: The Science Question in Feminism and the Privilege of Partial Perspective." Feminist Studies, 14(3), 575-599.

Mistrik, Ivan; Richard M. Soley; Nour Ali; John Grundy; Bedir Tekinerdogan, eds. 2016. Software quality assurance: in large scale and complex software-intensive systems. Morgan Kaufmann.

Prior, Julia. 2011. Everyday practices of agile software developers. PhD dissertation, University of Technology Sydney.

Scott, S and Orlikowski, W. 2014. "Entanglements in practice: Performing anonymity through social media." MIS Quarterly, 38, 873–893.

Suchman, Lucy. 2007. “Human-machine reconfigurations: Plans and situated actions.” Cambridge University Press.

Suchman, Lucy. 2012. "Configuration." Inventive methods, pp. 62-74. Routledge.