Interventions for Software Security

Creating a Lightweight Program of Assurance Techniques for Developers

Charles Weir
Security Lancaster
Lancaster University
United Kingdom
c.weir1@lancaster.ac.uk

Ingolf Becker
Security and Crime Science
University College London
United Kingdom
i.becker@ucl.ac.uk

James Noble
Engineering and Computer Science
Victoria University of Wellington,
New Zealand
kjx@ecs.vuw.ac.nz

Lynne Blair
Computing and Communications
Lancaster University
United Kingdom
l.blair@lancaster.ac.uk

M. Angela Sasse
Computer Science
University College London
United Kingdom
a.sasse@ucl.ac.uk

Awais Rashid
Bristol Cyber Security Group
Bristol University
United Kingdom
awais.rashid@bristol.ac.uk

***Abstract*— Though some software development teams are highly effective at delivering security, others either do not care or do not have access to security experts to teach them how. Unfortunately, these latter teams are still responsible for the security of the systems they build: systems that are ever more important to ever more people. We propose that a series of lightweight interventions, six hours of facilitated workshops delivered over three months, can improve a team's motivation to consider security and awareness of assurance techniques, changing its security culture even when no security experts are involved. The interventions were developed after an Appreciative Inquiry and Grounded Theory survey of security professionals to find out what approaches work best. They were then validated in fieldwork with a Participatory Action Research study that delivered the workshops to three development organizations. This approach has the potential to be applied by many development teams, improving the security of software worldwide.**

Keywords— Developer centered security; software security; software developer; intervention; action research

I. INTRODUCTION

Software security and privacy are becoming major issues: almost every week we hear that yet another organization's software systems have been compromised [1]. While there are many aspects to security and privacy, the security of an organization's software clearly has a large impact on whether such breaches happen. Therefore, the effectiveness of development teams at creating secure software is vital[1].

Many if not most developers, unfortunately, consider software security to be 'not their problem' [2]. Developers may expect security to be handled by a different team; or consider it too expensive to incorporate without a significant drive from product management. In the past, many organizations have addressed the issue with prescriptive instructions for development teams to follow or specifications of tools for developers to use; this approach of giving instructions to 'passive' developers has not been widely adopted [3].

Existing research has identified a range of well-understood assurance techniques [4] used by security professionals to help improve the security of a system. Yet if we are to improve software security in a wide range of teams, we need approaches that work where resources may be limited and security expertise unavailable. So, in this paper we explore:

1. What are inexpensive ways to introduce such techniques within a software development team?
2. How can the techniques be introduced in situations where there are no security experts directly involved?

This paper presents research into these questions: a survey of security professionals who work with software developers to address the first question; and based on the results, the subsequent creation and trials of a package, 'Developer Security Essentials', to address the second.

The contribution of this paper is:

- Industry evidence that motivating developers to introduce security changes is a primary way of introducing security improvements in development teams
- Identification of eight primary techniques currently in use, including two types of motivational workshop not previously identified.
- Proof that a package based on these techniques can improve the security of code delivered by a development team that has no access to security experts.

The structure of this paper is as follows: Section II establishes the existing literature on the subject; Section III explains the research and analysis methods; Section IV describes the results and conclusions from the survey; Section V describes the Developer Security Essentials package, plus the companies and developer teams who participated in the trials; Section VI gives the results of the trials; and Section VII summarizes and identifies future work.

[1] Throughout this paper we use 'secure' and 'security' to refer to the security and privacy aspects of software development; and 'developers' to refer to all those involved with creating software: programmers, analysts, designers, testers, and managers.

II. BACKGROUND

Recent research related to assurance techniques to improve developer security has taken a variety of approaches. This section examines the existing academic literature and related publications on the subject of helping and encouraging developers to improve their software security. We explore several areas in turn, examining the existing research available in each, and moving from technical to more sociological approaches.

A. *Code Analysis Tools*

There has been considerable recent security research into solving software security problems with code analysis tools. Five research groups [5]–[9] have created security defect detection tools to help developers improve code, using feedback via IDEs or elsewhere. Nguyen et al. [8] explored the impact of their tool on Android developers, concluding a high value for '*quick fixes'*: changes requiring little effort on the part of the programmer. Xie et al. [10] explored the impact of their IDE-based security analysis tool for web applications on a sample of 21 students and 6 professionals and found two interesting conclusions:

> *Users do not mind real-time warnings, but do not seem to want them to persist, even if they choose to ignore them. And even when creating secure code is relatively easy … users still need to be motivated to make the needed changes.*

B. *Adoption of Security-Enhancing Activities*

A different approach to improving software quality has been via changes to development processes, and there has been significant research into applying such changes to software security improvement. Indeed, prior to about 2010 the way of improving software security was a 'Secure Development Lifecycle' (SDL), a prescriptive set of instructions to managers, developers and stakeholders on how to add security activities to the development process. De Win et al. [11] compare the three major SDLs of that time, OWASP's CLASP, Microsoft's SDL and McGraw's Touchpoints, contrasting their features in the context of a simple project. Since 2010, SDLs have been replaced by 'Security Capability Maturity Models' [12], which measure the effectiveness of corporate security enhancements rather than mandating how they are achieved.

Meanwhile, however, other research suggests resistance from development teams to adopting a prescriptive methodology. For example Conradi and Dybå found in a survey that developers are skeptical about adopting the formal routines found in traditional quality systems [13]; others came to the same conclusion [14]–[16]. Indeed Geer's online survey of 46 developers recruited from those already specializing in secure software development found only 30% of them using SDLs [17]; Xiao et al.'s later survey of 40 developers [3], found only 2 using them. While these sample sizes were fairly small, their findings provide a plausible explanation for the abandonment of SDLs.

Taking a different approach, Such et al. investigated the economics of software security, surveying 150 security specialists to analyze the economics of different assurance techniques [4]. The survey generated a taxonomy of twenty assurance techniques and found wide variations in the perceived cost-effectiveness of each.

C. *Consultancy and Training Interventions*

Several research teams have explored the impact of training and external involvement on teams' delivery of secure software. Türpe et al. [18] explored the effect of a single penetration testing session and workshop on 37 members of a large geographically-dispersed project. The results were not encouraging; the main reason was that the workshop consultant highlighted problems without offering much in the way of solutions.

Poller et al.'s later study [19] followed an unsuccessful attempt to improve long term security practices in an agile development team of about 15 people. The study investigated the effect of security consultants whose task '*was not to advise the product group on how to change their organizational routines, but to challenge and teach them about security issues of their product'*. This proved insufficient, for two reasons. First, pressure to add functionality meant that attention was not given to security issues. Second, developers had trouble 'improving security' because their normal work procedures and ways of structuring their work did not support that kind of quality goal. The authors concluded that successful interventions would need "*to investigate the potential business value of security, thus making it a more tangible development goal*"; and that security is best promoted as a team, not individual, effort.

D. *Improving Security Experts' Interactions with Developers*

Recent work by Ashenden and Lawrence [20] took a different approach. They used an Action Research method to investigate and improve the relationships between security professionals and business people in a single company, and found the approach effective in improving communication, though no evidence is yet available of longer-term impact.

Other work has also investigated the impact of different kinds of relationship on software security: Werlinger et al.'s ethnographic study and survey [21] explored the relationships of security practitioners (mainly operations staff) on the effectiveness of security, and proposed several tool enhancements to improve this, particularly in the control of information being communicated to other stakeholders. Haney and Lutters found from a survey of security practitioners [22] that the role is service-oriented and involves both customer service and advocacy skills.

E. *Limitations of Existing Literature*

The code analysis tools required other interventions to get them adopted; and the other approaches required both security professionals and interventions that were costly in terms of effort involved. We are aware of no academic literature investigating lightweight ways to encourage developers to adopt successful security practices. In this work we offer one possible such approach.

III. METHODOLOGY

This section introduces the methodology used in the two phases of the research.

A. *Survey Methodology*

The open nature of the question '*What are effective ways to introduce such techniques within a software development team?*' required an inductive approach. We therefore interviewed a range of professional software security practitioners to ask how they achieved successful security-enhancing interventions to software development teams. Interviewees were chosen opportunistically; our connections in industry provided introductions to a number of successful, and mostly senior, practitioners with considerable experience of helping teams achieve software security. We interviewed 15 different experts from 14 different organizations. 12 were based in the UK, 2 in Germany, and one in the USA. They are described briefly in Table 1, which assigns an identifier, P1 to P15, to each.

The successfulness of their interventions was self-reported; all the organizations involved, however, have strong track records in achieving secure software. Specifically, P3, P4, P5, P6, P8 and P10 are from well-known organisations associated with effective security; P1, P2, P7, P9, P11, P12, P14, P15 and P16 are in businesses successfully selling secure services; and P13 is respected in the security research community. The survey was not looking for certainty that the interventions led to secure software; instead it sought the most promising techniques.

We wanted a firm basis on which to postulate theories, and therefore adopted Grounded Theory [23], which provides an academically rigorous basis, and has been widely used to investigate software development practice [24].

To achieve as good communication as possible, all the interviews were face-to-face, usually at the interviewee's workplace. Our questions aimed to draw out what participants had found most effective, and what they had seen to be most effective in other teams. To emphasize the positive, we used open questions about successful techniques known to the interviewees, avoiding asking questions about perceived problems or unsuccessful approaches. This approach relates to the Appreciative Inquiry school of Action Research [25], and indeed we used questions based on Appreciative Inquiry's 'Discovery' of best past practice, and 'Dream' of ideal future practice.

B. *Survey Analysis*

Grounded Theory involves line by line textual analysis of research data, in this case of transcriptions of the interviews along with notes and comments made by the interviewer. We used guidelines from a survey of previous software engineering Grounded Theory studies [24] to guide the research. As is typical in such work, we recorded the interviews and transcribed them manually. The lead author did the coding, categorizing and sorting operations on the data using the commercial tool NVivo. The final code book consisted of 4 families of codes and a total of 132 codes, applied to 1125 quotations in total.

In our coding, we were looking for the assurance techniques used by our interviewees and their ways of introducing them; and for comments and strategy related to them and their effectiveness. We also wanted a 'core category' to cover the widest possible scope of concepts discussed by the interviewees.

Section V.A describes how we then used the learning, from this 'core category' and the identified techniques, as a basis to construct an 'intervention package' to be delivered by the researchers to development teams, based on the best practice described by the interviewees.

C. *Package Trials Methodology*

Given that the researchers themselves directly influence the behavior of the research participants—the researchers *provide* the intervention—an ethnographic research approach was deemed inappropriate. Instead an accepted methodology, used in many forms of academic social research, including software engineering [26], [27], is Action Research [28]. This is an approach to research in communities that emphasizes participation and action; Action Research aims at understanding a situation in a practical context and aims at improving it by changing the situation.

Specifically, we used Participatory Action Research [29], with the lead author working as 'intervener', directly with the participants[2]. We had a Pragmatic approach, since the intention was to primarily to trial the impact of the interventions. This stage of the project involved only a single feedback cycle [30].

The key research question was: '*What security effects did the intervention package have?*' To measure an effect, we needed a baseline with no intervention. A-B testing, requiring a different team working in parallel, was not practical. Instead, we used a longitudinal approach, deducing a baseline ('no intervention situation') from the initial situation plus a knowledge of the original plans by the team leaders to improve security over the same timescale.

First, we interviewed a selection of the future participants to establish a baseline in terms of their current understanding, practice and plans related to secure software development. We then carried out a series of intervention workshops with members of the development teams, led by the intervener. Finally, a suitable time after the final intervention workshop, we re-interviewed the same participants as before.

D. *Package Trials Analysis*

The recordings of the interviews and most of the workshops—a total of 19 hours of audio—were transcribed and qualitatively analyzed. In an iterative process, two of the authors coded all transcripts. Initially both authors used open coding [23] on the first two hours of material, then agreed on a coding scheme based on that and the research questions. Differences in coding were discussed and resolved between us. This final code book consisted of 5 families of codes, making a total of 41 codes, applied to 1405 quotations in total.

In coding, we were looking for aspects of security improvement—including in learning and attitude—implied by statements from the speakers. We sought signs of new knowledge in

[2] The Action Research is 'Participatory' in that research subjects worked with researchers to create security outcomes; the subjects did not influence the intervention design.

the team, new activities related to security, and evidence of improvements in the security of developed software; we also recorded evidence of security activities and awareness before the start of the interventions.

Both studies were approved by the Lancaster University Faculty of Science and Technology Research Ethics committee.

IV. Survey Results

From the survey we derived an overview theme ('core category'), and also the experts' successful intervention practices.

A. *Overview Theme*

In our analysis, the key theme we found was the perception of developers themselves as the drivers of security adoption. This is different from the perception of developers as agents to be controlled by a 'Secure Development Process'.

Figure 1: Developer as Driver of Adoption

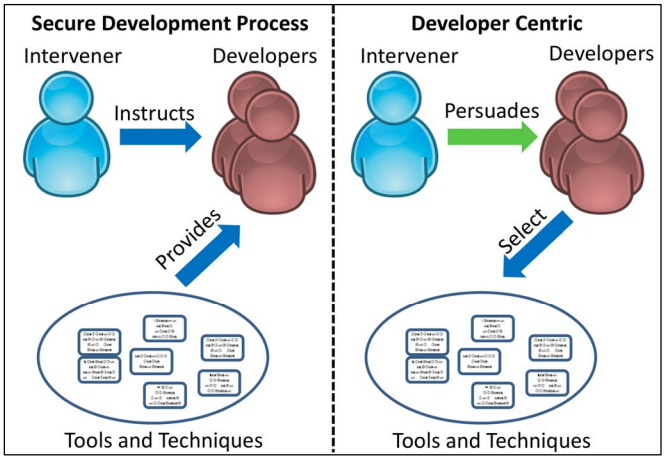

Figure 1 illustrates the difference. In the Secure Development Process approach, the role of the intervener is to tell the developers what to do, and to provide the techniques and tools that the developers are required to use [11]. Instead we found our interviewees promoted a 'Developer Centric' approach, with role of the intervener as sensitizing the developers to their security needs, allowing them to choose for themselves which tools and techniques to use.

One interviewee described the difference as follows:

> *It's not just about educating the developers, well, I guess it was, but we had to get the developers on side, the developers had to understand why we were doing this, as well as what it was that we needed them to do, so it was a kind of two pronged thing. (P2)*

Most implied security motivation as a fundamental requirement:

> *I think the learning component is a very strong thing, and the second thing is they are proud people, and they want to produce code they are proud of, and also producing security needs to be part of that. (P6)*

They were clear that even those with power in the organization still have to work by persuasion, not coercion. For example, even though P8 is Head of Security for a large multinational company, he told us:

> *So, working with Dev Teams ... you can't go in and say, "you must do it this way", it would never work… What you have got to do is go in there, and you have to convince them that it is to their advantage to do it that way. (P8)*

B. *Intervention Practices*

The coding process generated a set of intervention techniques, as shown in Table 1. The majority of these are already well known and documented; for these we have used the terminology defined by Such et al. [4]. However, three of the practices—**On-the-job Training**, **Incentivization Session**, and **Product Negotiation**—are new in the context of security assurance techniques. These, shaded diagonally in the table, are specific to motivating and empowering developers to make their own security decisions. These techniques are not novel, since they are in use in industry; they are however seldom discussed in developer security literature.

Table 1 also provides an indication of the share of the interviewees' discussion taken by each technique. The numbers indicate the percentage of identified quotations for each interviewee that discussed each intervention technique; cells are highlighted based on their values.

All interviewees but one discussed **Automated Static Analysis** tools, many in some depth, though few described them as particularly valuable. Several warned about the risk of developers believing that using a tool on its own would achieve security.

> *To a degree, yes, they are useful. ... But the danger with them is that you think that is making your code secure, and it is not. It is just finding a certain class of problems in it. (P2)*

Many interviewees stressed the importance of **Penetration Testing**. As discussed in Section II.C, this can be part of an **Incentivization Session**; however, it is more often used as a supporting technique.

> *If the team has developed something new… and it is a significant change, we might get it externally pen tested, if we think that we can't test it ourselves. (P12)*

Unfortunately, all forms of Penetration Testing require expertise; this expertise is in short supply currently, which makes the technique unsuitable for teams on a budget.

> *I fairly often get rolled out to persuade clients that [a pen test] is necessary. It is quite expensive. (P1)*

Code Review, scheduled meetings or pair programming to analyze code for security defects, was also popular. It requires a particular culture in the programming team:

> *It is in the culture. We do reviews; we always have to do reviews. We set things up, and it is not regarded as a second class. (P6)*

On-the-Job Training was widely used. This could be informal training (only P11 provided formal training),

> *[Our security specialist] will ... provide a show and tell ... a few times a year. (P1)*

mentoring,

> *We send people on site, and we embed them into other teams. Our processes ... [are] then taken up by the customer [developer] teams. (P3)*

Table 1: Percentage Discussion Share for Each Interviewee about Each Intervention

ID	Role	Organisation	Automated Static Analysis	Penetration Testing	Code Review	On-the-job training	Incentivization Session	Product negotiation	Threat Assessment	Configuration Review
P1	CEO	Outsourced software developers	4	23	7	2	3			6
P2	Consultant	Security consultancy	7		2	2	7		2	
P3	Team leader	Security and military supplier	14	2	13	10		1		
P4	Researcher	Research organisation	4	4		4			2	
P5	Security team leader	Operating System Supplier	7	5	3	7	8	5		2
P6	Security expert	Security and military supplier	2	4	4	1	1			
P7	CEO	Software security tool supplier	33		2		4			2
P8	Security expert	Telecommunications provider	12	1	1		3		2	
P9	Consultant	Security consultancy	5	6	1	1	3	3	2	3
P10	Security expert	Software package supplier	7		1	8	4		4	1
P11	Trainer and consultant	Software security service supplier	17	8	5	8	8	2	2	
P12	Security team lead	Telecomms service provider	4	3	5	4	4	8	1	
P13	Researcher	Research organisation	12		6		6	2	6	
P14	Principal engineer	Outsourced software developers				8	2	2		
P15	Security team manager	Outsourced software developers	25		20	2			2	

Key: deeper shades of blue are higher percentage values; diagonal shading shows items not normally considered assurance techniques.

or via a 'security champion' selected from the team:

> *Security Champions, … one person in the team more interested in security. … You need that person in a team, you actually do. (P11)*

The **Incentivization Session**, to motivate developers might be a one-to-one talk,

> *Everyone who joins [this company] gets a security talk… And it includes examples of things that have gone wrong and why, and how badly these things can go wrong, and how easy it is to screw it up, and some pointers on things to read about, to learn about. (P1)*

or a training course or workshop.

> *So, we run a very large-scale education program … where we … tell developers exactly what happens in the real world, how TalkTalk was hacked, how Sony was hacked… Then we also show them all the stuff that our red teams do—our internal hackers—which really scares them! (P5)*

Surprising for us was the importance of the technique of **Product Negotiation**, empowering product management to make security decisions.

> *[A security enhancement] will go into a planning cycle. You can't just … say 'everyone has to do this tomorrow' because people are already maxed. It has to be planned. (P5)*

This requires developers to be able to explain security risks and mitigation costs:

> *We don't normally struggle with getting developers to do things; we do struggle with management to understand that security has an implementation cost, and if you want security features you need different sprints to be allocated. (P11)*

Threat Assessment, or 'threat modelling' was seen as important not just for its innate value:

> *Your answer to any kind of security question anywhere should almost always start with a threat model. (P9)*

but also for the learning the modelling process provides to developers:

> *Threat modelling: what I see as the big benefit here is… putting the team into the perspective, to think about the functionality from a different aspect, from a different point of view. (P10)*

Configuration Review, choosing secure components and frameworks, and keeping them up-to-date, was the last of the techniques described:

> *So, from an attacker's point of view, you look at whatever the system is, you don't need to look at the code at all,*

what [components] would they have used to produce this... And you'll find code exploits! You'll find the OWASP top 10 [types of vulnerability] in one [component] alone! (P9)

V. Creating and Trialling the Interventions Package

As our next step, we wanted to translate these findings into practical support for development teams, including those where resources may be limited and security expertise unavailable. This section explores how we developed a package of activities, 'Developer Security Essentials', to provide a low-cost intervention for such teams, and describes the commercial development teams who trialed it.

To trial the package, we arranged to work with three different commercial development teams, to find out the impact of the package on each.

A. *Creating the Package*

Each intervention identified from the survey had a variety of forms, suitable for different development budgets, team sizes, and team cultures. Looking at the list of interventions, we observed that three—**Incentivization Session**, **Threat Assessment**, and **On-the-Job Training**—are 'process interventions' and can be implemented for limited cost by an external facilitator. Three more—**Automatic Static Analysis**, **Configuration Review**, and **Product Negotiation**—require commitment by the developers to go ahead. And the last two—**Penetration Testing** and **Code Review**—are often considered relatively expensive [31] and outside the range of activities affordable for some development teams; the authors' experience as consultants suggested that persuading teams to spend several thousand pounds on commercial **Penetration Testing**, or to change culture to support **Code Review,** would both be difficult. We, therefore, concentrated on the first three, and used opportunities within the consultancy to consider the remaining interventions. Note that the researchers carrying out the interventions were not themselves 'security experts', and lacked specific knowledge of the issues and solutions for each of the domains and applications involved; we needed approaches that drew on the knowledge and expertise of the developers themselves.

Our biggest challenge was to find a suitable way to provide the **Incentivization Session**. The versions described by our interviewees were not suitable for a lightweight intervention. Instead we used the 'Agile Security Game' [32], invented by the lead author. This was based on the 'Mumba' role-playing game, invented by Frey [33] to help elicit participants' prior experience of real-life security attacks.

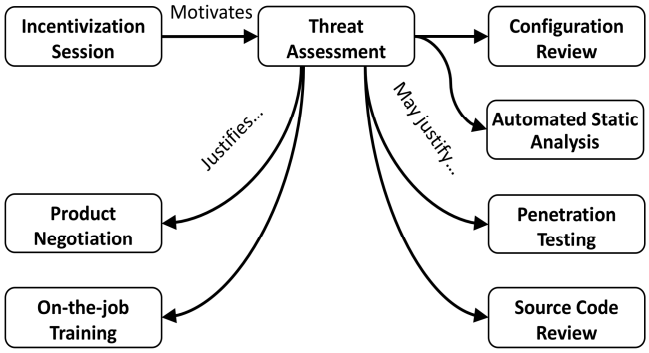

Figure 2: Structure of the Interventions

Threat Assessment, too, was also challenging to implement. Much of the literature [34], [35] describes a heavyweight process taking a while to set up and requiring considerable knowledge of possible technical threats, preferably with support from a professional with a detailed understanding of both the industry sector and current cyber threats to it. Neither knowledge nor expert were available. However, as technical lead for a major mobile money project the lead author had faced this problem and developed a lightweight brainstorming process to identify threats and mitigations. This had delivered a banking product with successful security [36]. Accordingly, we decided to trial the same approach here.

From the authors' own consultancy experience, and the experience of Türpe et al. and Poller et al. [18], [19], we knew that a single intervention was unlikely to be successful on its own. Therefore, we agreed to a monthly meeting, as **On-the-Job Training**; its main purpose was to act as a regular 'nudge' of the importance of security.

To introduce the remaining interventions, we used an ad-hoc approach, as shown in Figure 2. The facilitator mentioned and discussed each of these interventions with the developers during the **Threat Assessment**, the mitigation discussions, and the subsequent sessions, using comments from the developers as cues.

B. *The Development Teams Involved*

We trialed the package with three development teams in three different companies, selected opportunistically; the following were the teams involved. The individual members we interviewed are identified using the team letter and a number: 'A1'.

1) Team A

Team A works for a company employing around 50 people in the UK. Their software product manages sensitive management data, and is used by some very large organizations, including several that are household names.

The company development teams show some of the enthusiasm and characteristics of a start-up. We observed a culture of technological improvement, and a willingness to embrace change. We worked with some dozen developers, including team lead A1 and programmers with a wide range of experience (such as A2, A3, A4).

2) Team B

Team B is a tiny non-profit start-up, run on a part-time basis by two professionals: an educationalist and a software project manager. Their purpose is to provide work experience for promising young people who would otherwise be unable to get initial jobs in Information Technology. They undertake pro-bono software development projects for charities.

The development team constituted the educationalist as team lead (B1), the project manager (B2), and two student developers (B3, B4).

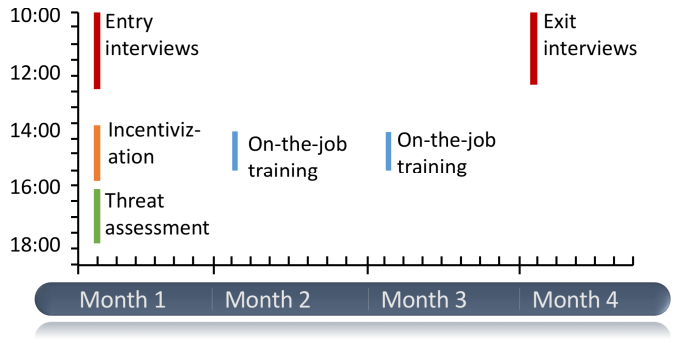

Figure 3: Intervention Timescales

3) Team C

Team C work for a well-known and long-established multi-national company, providing services via the Internet to a range of companies and individuals. The product is mature software, with a policy of continuous improvement.

We worked with a dozen team members including Quality Assurance (C1, C4), managers (C2) and programmers (C3, C5). All the team were competent and experienced professionals; in contrast to Team A we noted more emphasis on inter-departmental politics.

During the interventions, Team C's company changed policy on testing and three of our participants took redundancy. The changes meant that we managed only one 'continuous reminder' session after two months, though we achieved exit interviews with all of the participants except C4.

VI. Trial Results

This section describes the results of the Action Research-based trials in terms of cost, impact on the development process, and learning by the participants.

A. *Intervention Time Requirements*

Figure 3 shows the timeline for the program. It will be seen that, despite three months elapsed time, the total effort required from the intervener was only a total of two days, of which four hours were research interviews and not part of the intervention itself. So, including preparation, the total time spent by the intervener was less than two working days for each company. In terms of team effort involved, the approximate participant numbers and times involved were as shown in Table 2.

Table 2: Participant Time Cost in Person Hours

	Elapsed	Team A	Team B	Team C
Incentivization session	1.5	18	6	18
Threat modelling workshop	1.5	18	6	18
Follow-up 1	1	6	4	
Follow-up 2	1	6	4	8
Exit workshop	1	10	4	
Total		58	24	44

There were no significant financial costs apart from travel costs for the researcher.

We can therefore summarize the cost of this set of interventions as follows:

Intervention facilitator:	**15 man hours**
Development team:	**20 - 70 man hours**
Financial cost:	**minimal**

The cost of the interventions, therefore, is within the scope of a wide range of organizations.

B. *Security Outcomes for Each Team*

This section identifies the concrete outcomes attributable to the interventions. Quotations are attributed to the speaker where identifiable, or else to the appropriate session.

1) Outcomes for Team A

There were at least two significant improvements in Team A's product and process security as a result of the interventions. Beforehand, the developers had been thinking of security improvements as line by line improvements in the code they themselves had written. Afterwards, they understood that their most effective security improvements were likely to be elsewhere:

> *I find it a little concerning that there are so many attacks that we traditionally haven't mitigated against. (A Workshop)*

Specifically, they made two changes. First, they introduced a component security checker to their build cycle, and embarked on a program of updating and replacing components according to their security vulnerabilities.

> *We [have built] the OWASP dependency checker into our build process, … and established a process for how we deal with new vulnerabilities in existing libraries, or adding new libraries or upgrading libraries. (A1)*

Second, they identified their own existing customers as competitors with each other, and therefore potential 'attackers', and identified that the permissions functionality was therefore a major privacy issue; making fixes in this area was likely to give security wins:

> *I have a … task to check user permissions, and check that a user has access to that specific entity or a set of those entities (A2)*

They also introduced team sessions studying the OWASP 'Top Ten' vulnerabilities These had been mooted prior to the interventions but were only carried out after the initial workshops. And in later **On-the-job Training** sessions they established that the prioritization of security features required product management, not development, decisions.

> *That is where the priority call would come from. I think [Product Management] do understand it, … but there is always going to be that element of weighing up (Group Session)*

2) Outcomes for Team B

Team B, with very little prior security experience, had more potential improvements in process and in product security. As a result of the first **Threat Assessment** process they made several changes.

First, they introduced improved security for development workstations and code repositories, against the threat of malicious code modifications or access to personal data:

> *[We did] an audit on our computer systems: on our laptops… and the laptops that the students are bringing. We do scans, and make sure that the antivirus and anti-malware protection is all up-to-date. (B1)*

Next, they changed the planned design of the website not to use local databases for storing form data:

> *We said about the form, that it would send an email to the applicant. (B1)*

Plus, the student developers improved their security hygiene:

> *I also update my data a lot more, I back it up, not just to a file server but with a USB. (B2)*
>
> *The laptop I've been using, I've been making sure that the anti-virus is up to date. (B4)*

The **Threat Assessment** for Team B's later project led to caution with web service access keys:

> *We need to make sure that we are absolutely totally aware of how we make sure that those [API] keys don't become public, and that all students know that we have to do that. (B1)*

Finally, **Threat Assessment** is now being used for later projects:

> *We developed a threat model at the start of our current project and it is used in the code reviews and testing. (B1)*

Given that Company B's purpose is to help introduce students to IT roles, we note also that one student developer, B4, showed aptitude for identifying security threats in the workshops, and expressed an interest in a security-related career.

3) Outcomes for Team C

There were no identifiable improvements to Team C's process or product as a result of the interventions. The primary reason for this is that their security knowledge and practice as a team were already good: better than they may have realized.

> *I'm not sure too many changes were made. (C1)*

Thus, while some security improvements were made,

> *I'm much happier because we started working with Two Factor Authentication… for our client… admins… (C5)*

we believe these were the result of a wider awareness of security needs within the organization rather than because of the interventions (authentication issues were not discussed in the workshops). Indeed, the two main issues highlighted from the

Table 3: Summary of Techniques Adopted

	Team A	Team B	Team C
Incentivization Session	Provided by Intervener	Provided by Intervener	Provided by Intervener
Threat Assessment	Led by Intervener	Introduced for subsequent projects	Led by Intervener
On-the-Job Training	Instigated study of OWASP T10	Introduced due to intervention	Already in place
Product Negotiation	Introduced due to Intervention		Already in place
Configuration Review	Introduced due to Intervention	Introduced due to intervention	Already in place
Automated S. Analysis			Already in place
Pen. Testing	Already in place		Already in place
Code Review	Already in place		Already in place

Table 4: Evidence of Learning

ID	*Role*	*Experience (years)*	Automated Static Analysis	Product Negotiation	Configuration Review	On-the-job Training	Threat Assessment
A1	Architect	17	1	4	3	3	3
A2	Programmer	2				2	2
A3	Programmer	14		1	3	2	
A4	Programmer	3			2		1
B1	Manager	25		1			2
B2	Manager	13					
B3	Programmer	<1					
B4	Programmer	<1					
C1	QA	7	1		1	1	1
C2	Manager	13		1			
C3	Programmer	3					
C5	Programmer	10		1			3
A	Team discussion		6	1	11	6	2
B	Team discussion			2	3		4
C	Team discussion			4			1

Key: deeper shades of blue indicate higher counts

Threat Assessment—control of physical access to developer workstations and the relationship with the company's security department—were outside the control of the participants in the workshops.

The participants did, though, identify improved communication and understanding resulting from the interventions:

> *I think it got everyone talking about security a bit more, especially within our team... There was a lot of security things going on that I didn't know about. (C1)*

C. *Summary of Techniques Adopted*

Table 3 summarizes the above outcomes: shaded cells indicate new assurance techniques adopted as a result of the intervention process.

D. *Security Learning as a Result of the Interventions*

The outcomes in the previous section are valuable, but even more important for long term impact is the ability of the teams and individuals within them to implement secure software in future. To assess this, the exit interviews included an open question to elicit whether the participants appreciated the need for **Threat Assessment** and other interventions. We coded statements that showed evidence of an appreciation and internalization of the various techniques.

Table 4 shows the results of that analysis, along with brief descriptions of each participant. The top lines (A1–C5) consider the exit interviews for each participant and identify how many statements indicated internalized understanding of each assurance technique. The bottom three lines consider group discussions towards the end of the process and show the number of participant statements that showed similar understanding.

There was little discussion of **Penetration Testing** and **Code Review**, and only A1 showed appreciation of the **Incentivization Session**; these are not shown in the table.

As Table 4 shows, though both teams B and C implemented many of the assurance techniques, many of the individuals we interviewed did not evince a strong understanding of the reasons and approach to do so for future projects. Note however that since there were no explicit interview questions about each technique, the omission may not reflect the true understanding of the participants involved.

However, members of the Company A gained a good understanding of the techniques; we can conclude they did not implement **Automatic Static Analysis** as a positive decision based on discussion. The leaders of teams A and B indicated they had learned aspects of future **Product Negotiation**:

> *I guess, one challenge, as always, is playing what we, as architects, believe are the most pressing security concerns, against what customers are asking for in terms of dealing with security concerns. (A1)*
>
> *I would …feel confident to be able to talk to people about our security policies and how we manage security (B1)*

And of **Threat Assessment**:

> *[If I was advising a team on security] I think brainstorming threats and vulnerabilities and assets is really helpful. (A1)*
>
> *And one of the things that I think we probably are doing, as a result of being part of this process, is that auditing, that thinking things through first, what are our security issues, what are our risks, and how we are going to deal with those, in terms of the design. (B1)*

VII. Discussion and Future Work

This research shows that an affordable program of interventions, costing limited effort on the part of a facilitator and a development team—and not requiring the involvement of a security expert—can measurably improve the ability of that team to deliver secure software. Specifically, we conclude from Table 3 that such a program can be effective with teams with limited or no security experience.

The impact of the interventions differed hugely between teams: not only in the nature of the security issues addressed; but also in the teams' responses to the interventions and in how they benefitted. Team A introduced better development processes; Team B gained an awareness of several specific security improvements and the need for Threat Assessment; and for Team C the interventions prompted better communication and understanding (Section VI).

The successes identified came through the developers' choices. As the expert survey concluded (Section IV.A), to be effective a program needs to motivate rather than simply direct the teams involved. And, indeed, the interventions were successful to the extent that they could change the developers' thinking, understanding and motivation. The interventions involved, predominantly, conversations between developers, allowing them to learn mainly from each other, and to motivate themselves rather than respond to outside pressures. Table 3 and Table 4 suggest that this was an effective motivation and learning approach.

A. *Future Work*

We identified two key areas for future work on the interventions. First, the participant-driven nature of the workshops meant that not every technique was covered for every team: Team B did not discuss **Automated Static Analysis, Penetration Testing,** nor **Code Review**, for example. One participant suggested a checklist or take-away sheet after the first day's presentation:

> *I think maybe some sort of tick sheet in terms of "have you got these things in place?" to take away, that might be a good addition (A1).*

Second, for the program to scale to a wider number of participant teams, we need program leaders who appreciate the aims of the different sessions, such as the importance of motivation to achieve team empowerment hence **Incentivization Session**. Yet Table 4 suggests that this knowledge was not successfully conveyed to many of the participants. Nor did the participants learn how to use the program themselves. Also, to use the techniques, participants will need to facilitate some of the sessions; it is unclear yet what this will require.

In future we plan to address this problem in a second Action Research cycle, by providing the interventions program materials in book form, and by coaching 'interveners' to provide the training workshops and techniques in their own development teams, merely supported by the researchers.

Considering the design of the trials, the limited number of development teams involved, and the relatively short three-month term of each trial, both offer scope for improvement. Furthermore, the changes resulting from the interventions were self-reported: the trials do not provide evidence that the techniques were indeed implemented; nor that they improved the security of the resulting code.

To address the first issue, our next Action Research cycle will involve a larger number of teams; to address the second we plan follow-up interviews after one year where possible for each team; to address the last, we plan to request details of improvements made and vulnerabilities removed.

The data set from this larger set of teams may also permit analysis of which assurance methods can be applied successfully in which kinds of software engineering practice.

VIII. Conclusion

Lightweight, facilitation-based, interventions of the kind reported here offer the potential to help software development teams with limited current security skills to improve their software security. Widescale adoption of programs of this kind will empower developers, and play a much-needed role in improving software security for all end users.

References

[1] Forbes, "Top 2016 Cybersecurity Reports Out From AT&T, Cisco, Dell, Google, IBM, McAfee, Symantec And Verizon," *Forbes*, 2017. [Online]. Available: https://www.forbes.com/sites/stevemorgan/2016/05/09/top-2016-cybersecurity-reports-out-from-att-cisco-dell-google-ibm-mcafee-symantec-and-verizon/. [Accessed: 25-Sep-2017].

[2] J. Xie, H. R. Lipford, and B. Chu, "Why Do Programmers Make Security Errors?," in *IEEE Symposium on Visual Languages and Human Centric Computing*, 2011, pp. 161–164.

[3] S. Xiao, J. Witschey, and E. Murphy-Hill, “Social Influences on Secure Development Tool Adoption: Why Security Tools Spread,” in *ACM Conference on Computer Supported Cooperative Work*, 2014, pp. 1095–1106.

[4] J. M. Such, A. Gouglidis, W. Knowles, G. Misra, and A. Rashid, “Information Assurance Techniques: Perceived Cost Effectiveness,” *Comput. Secur.*, vol. 60, pp. 117–133, 2016.

[5] Y. R. Smeets, “Improving the Adoption of Dynamic Web Security Vulnerability Scanners,” Radboud University, NL, 2015.

[6] J. Xie, B. Chu, H. R. Lipford, and J. T. Melton, “ASIDE: IDE Support for Web Application Security,” in *27th Annual Computer Security Applications Conference*, 2011, p. 267.

[7] I. Pribik and A. Felfernig, “Towards Persuasive Technology for Software Development Environments: An Empirical Study,” in *International Conference on Persuasive Technology*, 2012, pp. 227–238.

[8] D. C. Nguyen, D. Wermke, M. Backes, C. Weir, and S. Fahl, “A Stitch in Time: Supporting Android Developers in Writing Secure Code,” in *ACM SIGSAC Conference on Computer & Communications Security*, 2017.

[9] L. N. Q. Do, K. Ali, B. Livshits, E. Bodden, J. Smith, and E. Murphy-Hill, “Just-in-time Static Analysis,” in *26th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2017, pp. 307–317.

[10] J. Xie, H. R. Lipford, and B. B.-T. Chu, “Evaluating Interactive Support for Secure Programming,” in *SIGCHI Conference on Human Factors in Computing Systems*, 2012, pp. 2707–2716.

[11] B. De Win, R. Scandariato, K. Buyens, J. Grégoire, and W. Joosen, “On the Secure Software Development Process: CLASP, SDL and Touchpoints Compared,” *Inf. Softw. Technol.*, vol. 51, no. 7, pp. 1152–1171, Jul. 2009.

[12] G. Mcgraw, S. Migues, and J. West, “Building Security in Maturity Model 8.” 2017.

[13] R. Conradi and T. Dybå, “An Empirical Study on the Utility of Formal Routines to Transfer Knowledge and Experience,” *ACM SIGSOFT Softw. Eng. Notes*, vol. 26, no. 5, pp. 268–276, 2001.

[14] B. Hardgrave, F. Davis, and C. Riemenschneider, “Investigating Determinants of Software Developers’ Intentions to Follow Methodologies,” *J. Manag. Inf. Syst.*, vol. 20, no. 1, pp. 123–151, Jul. 2003.

[15] M. Lavallee and P. N. Robillard, “The Impacts of Software Process Improvement on Developers: A Systematic Review,” in *34th International Conference on Software Engineering, ICSE 2012*, 2012, pp. 113–122.

[16] C. K. Riemenschneider, B. C. Hardgrave, and F. D. Davis, “Explaining Software Developer Acceptance of Methodologies: A Comparison of Five Theoretical Models,” *IEEE Trans. Softw. Eng.*, vol. 28, no. 12, pp. 1135–1145, Dec. 2002.

[17] D. Geer, “Are Companies Actually Using Secure Development Life Cycles?,” *IEEE Computer*, vol. June, pp. 12–16, 2010.

[18] S. Türpe, L. Kocksch, and A. Poller, “Penetration Tests a Turning Point in Security Practices? Organizational Challenges and Implications in a Software Development Team,” in *Second Workshop on Security Information Workers*, 2016.

[19] A. Poller, L. Kocksch, S. Türpe, F. A. Epp, and K. Kinder-Kurlanda, “Can Security Become a Routine? A Study of Organizational Change in an Agile Software Development Group,” in *ACM Conference on Computer Supported Cooperative Work*, 2017, pp. 2489–2503.

[20] D. Ashenden and D. Lawrence, “Security Dialogues : Building Better Relationships,” *IEEE Secur. Priv. Mag.*, vol. 14, no. 3, pp. 82–87, 2016.

[21] R. Werlinger, K. Hawkey, D. Botta, and K. Beznosov, “Security Practitioners in Context: Their Activities and Interactions with Other Stakeholders within Organizations,” *Int. J. Hum. Comput. Stud.*, vol. 67, no. 7, pp. 584–606, 2009.

[22] J. M. Haney and W. G. Lutters, “Skills and Characteristics of Successful Cybersecurity Advocates,” in *Third Workshop on Security Information Workers*, 2017.

[23] B. G. Glaser and A. L. Strauss, *The Discovery of Grounded Theory: Strategies for Qualitative Research*. Chicago: Aldine Transaction, 1973.

[24] K. Stol, P. Ralph, and B. Fitzgerald, “Grounded Theory in Software Engineering Research: A Critical Review and Guidelines,” in *38th International Conference on Software Engineering*, 2015, pp. 120–131.

[25] D. L. Cooperrider and D. Whitney, “Appreciative Inquiry: A Positive Revolution in Change,” *Appreciative Inq.*, p. 30, 2005.

[26] H. Sharp, Y. Dittrich, and C. R. B. De Souza, “The Role of Ethnographic Studies in Empirical Software Engineering,” *IEEE Trans. Softw. Eng.*, vol. 42, no. 8, pp. 786–804, 2016.

[27] Y. Dittrich, K. Rönkkö, J. Eriksson, C. Hansson, and O. Lindeberg, “Cooperative Method Development: Combining Qualitative Empirical Research With Method, Technique and Process Improvement,” *Empir. Softw. Eng.*, vol. 13, no. 3, pp. 231–260, 2008.

[28] W. F. Whyte, *Participatory Action Research*. Sage Publications, Inc, 1991.

[29] R. L. Baskerville, “Investigating Information Systems with Action Research,” *Commun. AIS*, vol. 2, no. 3es, p. 4, 1999.

[30] K. Petersen, C. Gencel, N. Asghari, D. Baca, and S. Betz, “Action Research as a Model for Industry-Academia Collaboration in the Software Engineering Context,” in *International Workshop on Long-Term Industrial Collaboration on Software Engineering*, 2014, pp. 55–62.

[31] J. M. Such, A. Gouglidis, W. Knowles, G. Misra, and A. Rashid, “The Economics of Assurance Activities,” 2015.

[32] C. Weir, “The Agile App Security Game – Leader’s Instructions,” 2017. [Online]. Available: https://www.securedevelopment.org/app/download/11233441072/TheAgileAppSecurityGame.zip. [Accessed: 28-Mar-2018].

[33] S. Frey, “Mumba Role Playing Game: The Rulebook.” 2016.

[34] A. Shostack, *Threat Modeling: Designing for Security*. John Wiley & Sons, 2014.

[35] Microsoft, “Microsoft Secure Development Lifecycle.” [Online]. Available: https://www.microsoft.com/en-us/sdl/. [Accessed: 29-Mar-2018].

[36] EE, “Cash On Tap from EE,” *YouTube*, 2014. [Online]. Available: https://www.youtube.com/watch?v=51CJNfRDuiI. [Accessed: 19-Sep-2018].