Interventions for Long Term 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. We tested the interventions in a Participatory Action Research field study where we delivered the workshops to three software development organizations, and evaluated their effectiveness through interviews beforehand, immediately afterwards, and after twelve months. We found that the interventions can be effective with teams with limited or no security experience, and that improvement is long lasting. This approach and the learning points arising from the work here have the potential to be applied in many development teams, improving the security of software worldwide.**

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

1 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[*].

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 assurance techniques within a software development team?
2. How can these 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. It expands on an earlier paper by the authors[5], incorporating new 'longitudinal' data from interviews one year after the intervention (section 6.5), a comparison of the activities of different teams (section 5.3), a more extensive literature survey (section 2), and a discussion of 'Blockers and Motivators' encountered by the different teams (section 6.5).

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 assurance techniques widely used in interventions, including two types of motivational workshop not previously identified; and
- Proof that a package based on these techniques can provide long term improvements in 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 2 establishes the existing literature on the subject; Section 3 explains the research and analysis methods; Section 4 describes the results and conclusions from the survey; Section 5 describes the Developer Security Essentials package, plus the companies and developer teams who participated in the trials; Section 6 gives the results of the trials; and Section 7 summarizes and identifies future work.

---

[*] 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.

2 BACKGROUND

This section examines the existing academic literature and related publications on the subject of helping and encouraging developers to improve their software security: 'Developer-centered Security'.

Research on assurance techniques to improve developer security has taken a variety of approaches. In this section, we explore several areas in turn: first research to support developers, moving from technical to more sociological approaches; and then support for researchers.

2.1 Exploring the Problem

Olivera et al.[6] used psychological manipulation in a study on 47 developer volunteers to explore what caused them to include vulnerabilities in software, concluding that main causes were developers' focus on 'normal cases' and a lack of priority for security. They concluded that, more than security training, the solution is targeting security reminders to the points in development when they were most needed.

Several researchers have investigated the difficulties around security fixes. Derr et al.[7] investigated the extent to which Android app developers keep library versions up to date, with a survey of app developers and analysis of app binaries. They found huge scope for solving vulnerabilities by library updating without changes in code, but that frequent backward incompatible changes and incorrect Semantic Versioning in libraries currently make such updates problematic. Others looked at the necessity for fixes: Nayak et al.[8] found that less than 15% of known vulnerabilities were actually used in attacks, suggesting an opportunity for a more nuanced approach than just fixing everything. Vaniea and Rashidi[9] used a survey to analyze user thinking around the update procedure, deriving advice for developers planning such mechanisms including a recommendation for a 'recovery path'.

Some researchers have investigated security requirements, especially related to privacy. A literature survey by Turpe[10] found a range of research related to security requirements, mainly exploring Threat Modelling techniques, but no agreement on terminology or approach. Senarath and Arachchilage[11] used a programming task given to 35 developers to explore issues related to user privacy, finding it to be difficult to understand such requirements and translate them into engineering techniques, and recommending solutions in the specification of privacy requirements. Similar research by the same authors[12] found that developers use their own privacy expectations to guide software privacy decisions; these differ from the expectations of non-developer users, though the authors point out there is no easy solution for this problem.

2.2 Code Analysis Tools

There has been considerable recent research into solving software security problems with code analysis tools. Christakis and Bird[13] surveyed Microsoft developers' opinions about such tools, finding that they consider security defects the most important for a code analysis tool to find, and that the key features they need are relevant results, speed, and the ability to suppress earlier warnings in incremental changes. Witschey et al.[14] surveyed around 50 developers to quantify the factors that caused them to adopt automated security tools, finding that the most important factor was seeing peers using them.

Many research groups[15–19] have created security defect detection tools to help developers improve code, using feedback via IDEs or elsewhere. Tabassum et al.[20] used a student developer study to compare the learning from using their tool with that from a short code review session by an expert; they found the expert more effective. Nguyen et al.[18] 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.[21] 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:

> *[Developers] 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 … [developers] still need to be motivated to make the needed changes.*

2.3 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 accepted 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. A paper by De Win et al. compares 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[22]. However, other research from that time 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[23]; others came to the same conclusion[24–26]. Indeed Geer's online survey of 46 developers recruited from those already specializing in secure software development found only 30% of them using SDLs[27]; Xiao et al.'s later survey of 40 developers[3], found only 2 using them. While these sample sizes were fairly small, the findings provide a plausible explanation for the abandonment of SDLs.

Since 2010, SDLs have been replaced by 'Security Capability Maturity Models', such as BSIMM[28], which measure the effectiveness of corporate security enhancements rather than mandating how they are achieved.

More recently, Caputo et al.[29] used three case studies to explore several theories about what changes in software development might lead to more usable security, concluding a need for the alignment of security goals with business goals.

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.

In terms of practical support for developers, a recent book 'Agile Application Security' by Bell et al.[30] provides guidance, a discussion of tools and detail on a range of assurance techniques.

2.4 Motivators and Blockers of Secure Software Development

In many organizations security has been an afterthought to an otherwise successful business model. This security is bolted on and is often treated as a compliance exercise[31]. This leads to suboptimal resource allocation: the security tasks are not integrated in the primary productive task of individuals, and professionals are often faced with a choice between following the rules or getting the job done[32].

Looking for an alternative to enforcing security compliance through policies, security researchers have been working together with behavioral scientists to change security behaviors. Pfleeger et al. observe that the key to enabling good security behavior is good motivators[33].

There is a wider literature around such motivators and their negative counterpart, blockers. Fogg found that individuals also need to have the ability to perform the required behavior and be triggered to perform it[34]. And Myers and Titgjen report that positive aspects to behavior change do not cancel out existing negative blockers; blockers to secure behavior need to be tackled at the same time to achieve lasting behavior change[35].

2.5 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.[36] 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[37] 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.

2.6 Improving Security Experts' Interactions with Developers

Other work has investigated the impact of different kinds of relationship on software security: Werlinger et al.'s ethnographic study and survey[38] 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[39] that the role is service-oriented and involves both customer service and advocacy skills.

Ur Rahman and Williams[40] surveyed web-based information and nine teams of developers to investigate how DevOps* incorporated security into projects, finding increased collaboration between developers and security specialists, and security benefits in the automation of testing, configuration and deployment.

Ashenden and Lawrence[41] took a proactive approach, using 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.

2.7 Supporting Developer-Centered Security Research

One constraint on research in this area has been the difficulty of experimental trials with professional developers. To address this, Stransky et al.[42] have trialed a 'Developer Observatory', to allow experimental coding engagements with large numbers of developers. Acar et al.[43] used a programming study with Python open source developers to explore the extent to which findings

* The integration of operations procedures into code, facilitating short development-release cycles.

from such studies might depend on the background of the developers chosen, finding the determinant to be participants' experience using the language, rather than status as a student or professional, or security experience. This supports future experiments with student participants, to the extent that results will be similar to those that would be obtained with professionals.

Another constraint has been the wide range of venues and journals covering the topic making research topics difficult to identify; a recent systematic literature survey by Tahaei and Vaniea[44] provides an introduction for researchers.

2.8 Limitations of Existing Literature

There are several notable absences from this literature.

There is little work addressing the—admittedly intractable—problem of measuring aspects of software security. We also found no work on the security blockers and motivators encountered by software developers.

There is, too, relatively little research discussing successful security interventions to support development teams, even though the security track records of many large companies suggest that such interventions must exist. In the research discussed above, even code analysis tools required other interventions to get them adopted; and 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.

3 METHODOLOGY

The research described in this paper was in two phases: an interview survey of security professionals who work with software developers; and based on the results, the creation and trials of a package, 'Developer Security Essentials'. This section introduces the methodology used in each phase.

3.1 Interview Survey Methodology

The open nature of the first research question '*What are inexpensive ways to introduce assurance 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 organizations 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[45], which provides an academically rigorous basis, and has been widely used to investigate software development practice[46].

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[47], and indeed we used questions based on Appreciative Inquiry's 'Discovery' of best past practice, and 'Dream' of ideal future practice. Appendix A lists the questions.

3.2 Interview Survey Analysis

Our Grounded Theory analysis involved 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[46] 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 5.1 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.

3.3 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[48,49], is Action Research[50]. 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[51], with the lead author working as 'intervener', directly with the participants[*]. We had a Pragmatic approach, since the intention was primarily to trial the impact of the interventions. This stage of the project involved only a single feedback cycle[52].

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

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

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. Then, a month after the final intervention workshop, we carried out 'Exit Interviews' with the same participants as before.

Finally, about a year after the initial workshops, we carried out 'One Year Interviews' by telephone with the leaders of each team, to find out to what extent the security effects of the package were long-lasting.

3.4 Package Trials Analysis

The recordings of all the interviews and most of the workshops—a total of 20 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[45] on the first two hours of material, then agreed on a coding scheme based on that and the research questions. Then both authors independently coded all the remaining material and compared the results. Differences in coding were discussed and resolved between us.

Following the one-year interviews, both authors again independently coded the transcribed interviews and compared the results. The final code book consisted of 5 families of codes, making a total of 41 codes, applied to 1465 quotations in total.

In all this coding we were looking for aspects of security improvement—including in learning and attitude—implied by statements from the speakers. We analyzed the kinds of interaction involved in the workshops, and looked for 'Motivators and Blockers': aspects that helped and hindered such security improvement. We particularly sought signs of new knowledge in 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 the survey and the intervention trials research were approved by the Lancaster University Faculty of Science and Technology Research Ethics committee.

4 SURVEY RESULTS

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

4.1 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

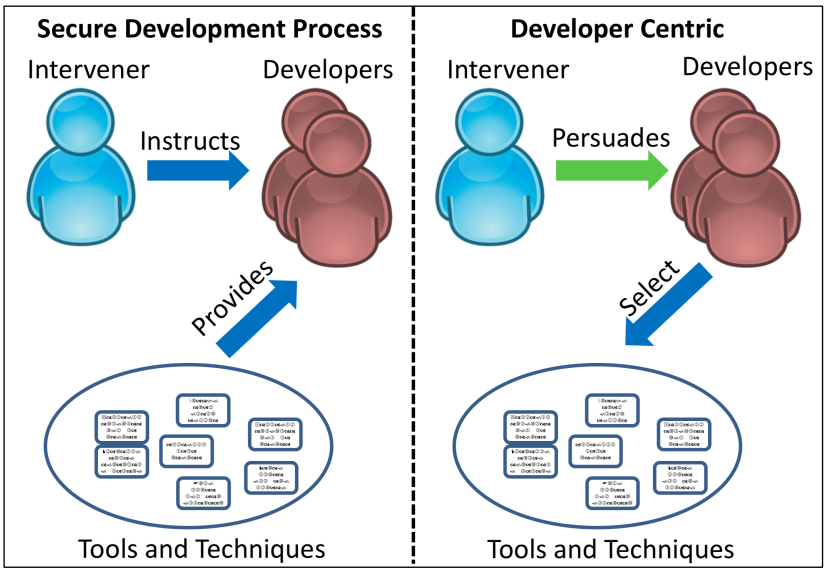


Figure 1 illustrates the difference. In the Secure Development Process approach, the role of the intervener is to tell the developers what to do, and to provide the techniques and tools that the developers are required to use[22]. Instead we found our interviewees promoted a 'Developer Centric' approach, 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)*

> *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)*

4.2 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 2.5, 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)*

> *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)*

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

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

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

ID	Role	Organization	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 organization	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	Telecoms service provider	4	3	5	4	4	8	1	
P13	Researcher	Research organization	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; shading shows items not normally considered assurance techniques.

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 one-to-many informational session:

> *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)*
>
> *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)*
>
> *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 ten [types of vulnerability] in one [component] alone! (P9)*

5 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.

5.1 Creating the Package

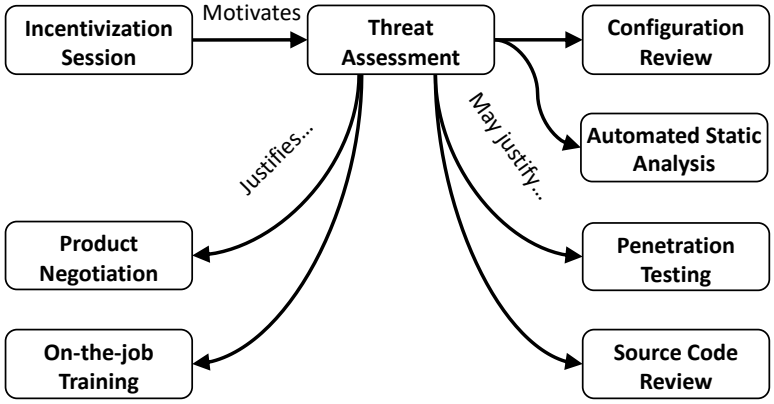
Each intervention identified from the survey had a variety of forms, suitable for different development budgets, team sizes, and team cultures. Looking at the list of interventions, we observed that three—**Incentivization Session**, **Threat Assessment**, and **On-the-Job Training**—are 'process interventions' 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. The last two—**Penetration Testing** and **Code Review**—are often considered relatively expensive[4] 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', invented by the lead author[53]. This was based on the 'Mumba' role-playing game, invented by Frey et al.[54] to help elicit participants' prior experience of real-life security attacks.

Threat Assessment, too, was also challenging to implement. Much of the literature[55,56] 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[57]. Accordingly, we decided to trial the same approach here.

Figure 2: Structure of the Interventions



From the authors' own consultancy experience, and the experience of Türpe et al. and Poller et al.[36,37], 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.

5.2 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'.

5.2.1 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).

5.2.2 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).

5.2.3 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.

5.3 Using the Package

Figure 3 shows how the nature of the workshops differed with different teams. It charts the proportion of words from each set of intervention workshops coded as corresponding to different activities.

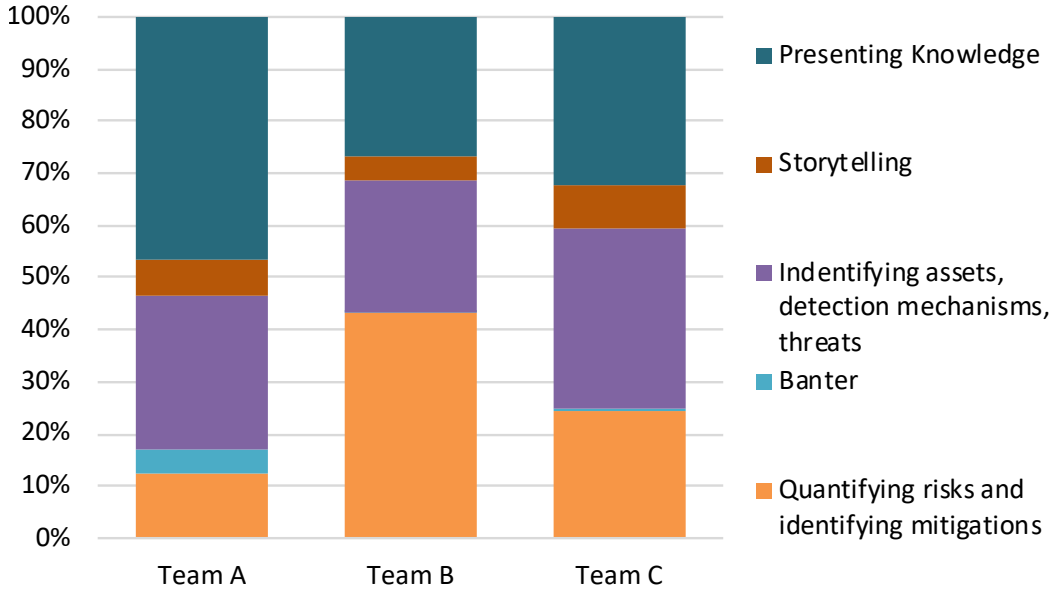


Figure 3: Discussion in the Intervention Workshops

A substantial portion of each session was Presenting Knowledge: one or more participants describing systems or known security issues. In particular, we observed many examples of an effective way of presenting knowledge, 'Storytelling' (shown separately in the diagram), narrating how a participant addressed or was affected by security issues[58].

The workshops varied considerably in the proportions of time devoted to the main activities: knowledge presentation, to finding issues and vulnerabilities, and to addressing the issues discovered. This reflects differences in culture, structure and projects between the teams.

A particular revealing measurement in terms of culture is the amount of Banter, friendly joshing and jokes, involved: Team A's high performing and relaxed culture had a good deal; Team C's more formal culture evinced little, and Team B, with a large disparity in status between participants, had none.

The differing proportions reflect different emphasis in the workshops. For Team A, the novelty was discovering the true nature of their security threats, while addressing them would be business as usual and so required less discussion. For Team B, starting from virtually no security knowledge and working on less security-critical projects, it was more important to find ways to deal with the smaller set of risks they did identify. And for Team C, with good security expertise but poor communication between teams, most of the benefit was in pooling knowledge fragmented among the participants, and hence discussion was fairly evenly spread between the three main activities.

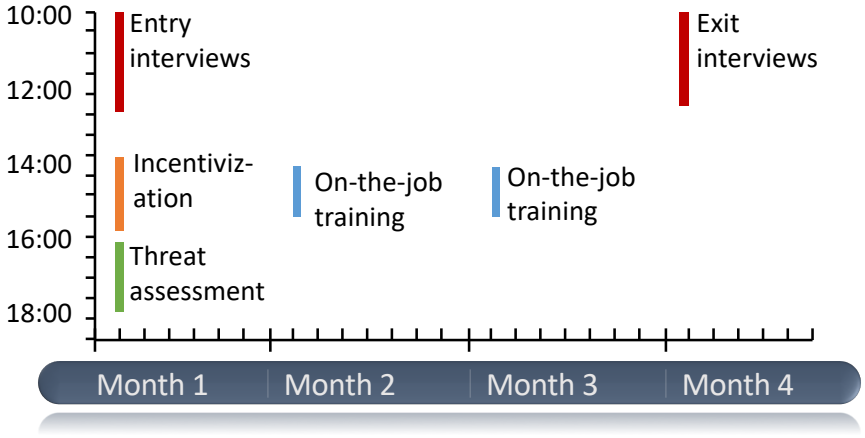


Figure 4: Intervention Timescales

6 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.

6.1 Intervention Time Requirements

Figure 4 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 person hours**
Development team: **20 - 70 person hours**
Financial cost: **minimal**

6.2 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.

6.2.1 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 three 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)*

Thirdly, they introduced a monthly focus on the OWASP 'Top Ten' vulnerabilities, one at a time. This approach had been mooted prior to the interventions but was only carried out after the initial workshops.

> *[A team architect] puts out a 'we're working through this one this week', and he puts up a link and it has got everybody's name next to it, and you read through it, and then there is more information if you want. You can ask questions, and we have got a good internal issue tracking*

board. Any kind of potential thing, big or small, goes on there, and it can get prioritized into our work properly. (A4)

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)

An unusual and intriguing approach they also tried was having one of the team be a covert 'saboteur', occasionally introducing security defects to see if the review process would find it; in practice, though, they found it problematic:

One team did the saboteur exercise … It was a bit mixed. The saboteur didn't enjoy being a saboteur... (Group Session)

6.2.2 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)

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

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)

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)

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 Team 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.

6.2.3 Outcomes for Team C

There were no identifiable improvements to Team C's process or product directly attributable to 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)

While some changes were made as a result on ongoing security improvements,

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

the participants did identify improved communication and understanding as 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)*

6.3 Summary of Techniques Adopted

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

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 Top 10	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 Static Analysis			Already in place
Penetration Testing	Already in place		Already in place
Code Review	Already in place		Already in place

6.4 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: 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

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 Team 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)*

[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)

6.5 Blockers and Motivators Encountered by the Teams

Using the same open coding as before, we analyzed the interviews and workshops to identify 'blockers', problems that threatened to prevent adoption of the practices; and 'motivators', incentives for practicing secure software development. In total, we identified 44 mentions of blockers and 27 mentions of motivators. They fit broadly into the following categories: organizational aspects, supporting tools and the product/business themselves. Figure 5 shows to what extent each was referenced by participants from each company.

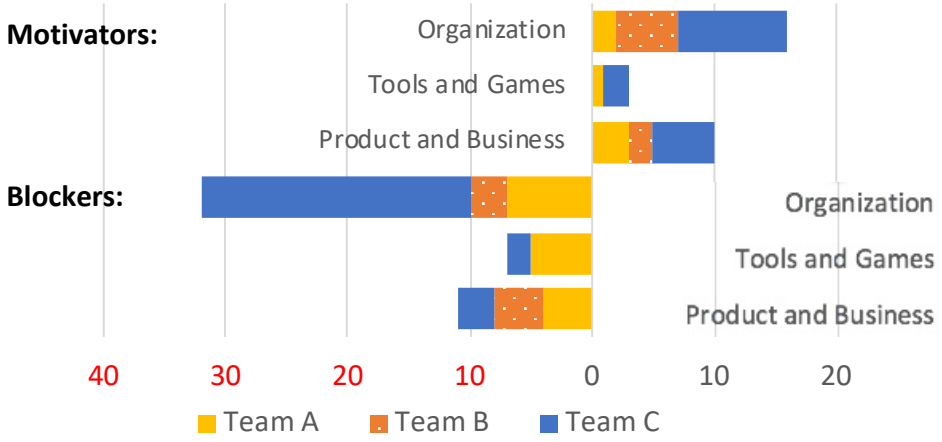


Figure 5: Mentions of Blockers and Motivators, by Company

6.5.1 Organizational Blokers and Motivators

Under organizational aspects we found blockers in management issues such as no clear ownership of security in the organization, and time and workload management. This is essentially the key scarce resource in organizations, and poor management of employees' time and workload will override any personal, positive factors[59]. Participants from Team C described a dysfunctional relationship with a security team that was required to sign off on products but gave no guidance to developers and was not approachable for help. Their security team apparently practiced an internal 'security through obscurity' approach, which makes learning from security issues difficult for developers.

It was almost as if this information was kept confidential, on a need to know basis, and unfortunately it means that [development] teams will find it difficult to learn from the event. (C2)

This is reflected in the large number of organization blockers identified by Team C in Figure 5. Two participants from Team A also noted that while the security education received in the organization was interesting and helpful, it was frustrating that there was no space for reflecting on it or practicing it when developing.

At the same time, management can also provide a motivator. In Team B security aspects were integrated into the development processes and acknowledged in planning:

> *"We needed to put it into our procedures, not just into our thoughts, but into our ... you know, 'this is the way we work, this is what we do'. This is what I have got out of it." (B1).*

One participant in Team C reiterated this point by considering holistic thinking about the product to generate security understanding and motivation. Another participant suggested that security targets should be part of performance indicators for employees in order to motivate work on security.

6.5.2 Supporting Tools as Blockers and Motivators

In terms of supporting tools, Teams A and C used a large number of tools, games and procedures to support their secure software development processes. A few of these approaches were abandoned due to their poor design: one game in Team A caused embarrassment to individuals, and made employees feel uncomfortable (see Section 6.2.1). Two participants noted the complexity of their infrastructures and the difficulty to integrate and configure off-the-shelf security solutions, especially when legacy software is involved. In particular, encryption, key management and cloud computing platforms were mentioned as aspects where achieving security was unreasonably difficult.

Yet at the same time, outside influences were also perceived as motivators for security. In Team C, compliance checks, certification and recent relevant legislation have all caused an increased interest in security in the organization, and this has driven security improvements. The participants also mentioned changes in architectures as motivators for security improvement, as in the participant's opinion improved features and improved security often co-occur. The developers are also keen to release their code to the public, motivating a greater focus on security:

> *Our problem, I think, here is that we have a tendency to want to make our code repositories public, as a means of helping the wider world. The problem with that is that you automatically put yourself in a vulnerable position. (C2)*

6.5.3 Business Function Blockers and Motivators

The third category of issues center on the team's business function. Team A noted that customer's security policies and requests for customization of the product are significant barriers to maintaining secure code and good policies. In Team C security was reported to be difficult to sell. Yet in Team A security customers were actively requesting security, making them a motivator for secure software development. The recent increase in news coverage on security is also seen as a motivator in both companies:

> *Some of it could be good old-fashioned scaremongering due to what has happened in the press, but if that is what works, then fine, we'll take that. Because the reality is, it was the stuff that needed to be done." (C2)*

6.5.4 Tension Between Blockers and Motivators

All three organizations had motivators in the categories where blockers were present. But these motivators where not created in response to the blockers, but rather as independent encouragements for secure software development.

The implication for development teams is the need both to encourage the motivators, and to resolve the blockers. Since most were outside the immediate control of the developers, this is an organizational, rather than a developer, opportunity for improvement.

6.6 Security Outcomes after One Year

Academic feedback from an early report on this research[60] questioned whether the security impact of the intervention was long-term rather than just over the three months. Accordingly, we requested further participant interviews after one year to find out.

Only Teams A and B responded to our requests for a further interview. In the case of Team C, only one of the initial interviewees was still with the company after one year and we received no response to interview requests.

6.6.1 Team A after One Year

As shown in Table 3 the intervention workshops had led Team A to instigate the techniques of Product Negotiation and Configuration Review. One year on, secure development has become increasingly important to sales (Product Negotiation):

> *With every sale we will get stringent questions around security. … I think, increasingly, there are more questions around development processes, and application security. And clearly, without being able to answer those questions satisfactorily, we wouldn't be able to sell. (A1)*

Configuration Review is now part of their development process:

> *The OWASP dependency checker is very much embedded in our process. We have never yet got it to the point where all the dependencies are green! But we do now appear to be at the point where it is a regular part of our process to check for new vulnerabilities that have been found, and to add upgrades for those libraries that contain known vulnerabilities, within, either, the next release, or the release after that, depending on how much other pressure there is on our road map. (A1)*

Disappointingly, the two innovative forms of On-the-job Training instigated by the teams independently of the workshops had not continued:

> *Sprint by sprint [we were] picking up one of the OWASP Top Ten, and getting all the developers to review it, and identify issues where we weren't meeting those things. That, sadly, has fallen by the wayside … [because] we didn't have the bandwidth on our road map to deal with the things that people were highlighting. (A1)*
>
> *[The secret saboteur] carried on for a few sprints, I think it didn't work out quite so well, when somebody got accused of being the saboteur, when actually it was just a genuine mistake they made! It then became very embarrassing for that person. I think that fell by the wayside, slightly! (A1)*

They were, however, considering using an Automated Static Analysis tool:

> *One of the things on our backlog is bringing in SonarQube which might potentially identify security issues in the code. (A1)*

6.6.2 Team B after One Year

Team B had introduced Threat Assessment, On-the-job Training and Configuration Review (Table 3) in the months after their initial workshops. Since the emphasis for Team B is on training, it was encouraging to find that they had continued improving their development security work.

Specifically, they now have Incentivization Sessions, and they teach Threat Assessment:

> *We are just starting a new project, so part of the induction, and part of the on-boarding for all the students, is that we do a little bit of security training, and we do a threat-modelling exercise. (B1)*

Their Code Reviews include security:

> *And then, as far as our code reviews are concerned, [we are] actually looking at security aspects, at every stage. So, each time we are doing a code review, security is one of the things on the form to tick. (B1)*

And they have various forms of On-the-job Training:

> *So, [we have] training; there is the thinking about it, and planning it into the design at the beginning, and then consciously monitoring as we go through. (B1)*
>
> *We had a few other students who come in at a later stage, and [B4] did a nice 'Brown Bag' talk on security. And we are passing that on. (B1)*

Particularly gratifying for us was that the intervention helped identify an aptitude in one participant for security work, and to inspire a choice of career:

> *And [B4] who was going to be a struggle because of his Maths and English, his options going forward are quite limited, but he is… about to start a Level 2 Traineeship in Cybersecurity! Something that came out of your research was really how interested he is in it. (B1)*

6.6.3 One Year Summary

Table 5 summarizes the techniques in use after one year. As in Table 3, the shading indicates assurance techniques introduced as a result of the interventions.

Table 5: Techniques in Use after One Year

	Team A	Team B
Incentivization Session		In regular use
Threat Assessment		In regular use
On-the-Job Training		In regular use
Product Negotiation	Introduced due to intervention	
Configuration Review	Introduced due to intervention	Introduced due to intervention
Automated Static Analysis	Planning introduction.	
Penetration Testing	Already in place	
Code Review	Already in place	In regular use

7 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 substantially 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, and from Table 5 that the improvement is long lasting.

The impact of the interventions differed 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. Sections 5.3 and 6.5 explored some of the differences in the ways the teams responded to these interventions.

The successes identified came through the developers' choices. As the expert survey concluded (Section 4.1), 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, Table 4 and Table 5 suggest that this was an effective motivation and learning approach.

Indeed, by contrast to the results of earlier studies based on interventions using Penetration Testing as a motivator[36,37], in this study Table 5 shows that for both Teams A and B the long term impact after one year was still important.

7.1 Learning for Software Development Teams

We highlight several learning points for software developers: the effectiveness of team activities, key assurance techniques, and the importance of organizational issues.

7.1.1 Apply Team Activities to Security

All the workshops derive their effectiveness more from discussions between participants than from any information provided by the intervener, and as Section 5.3 shows the nature of these discussions was different for each team. So, the success of these interventions can be attributed to the team nature of the activities, and on the participants bringing their own unique range of expertise and knowledge to them.

Whether or not a given team uses the specific workshops described here, we conclude that there is benefit in regarding software security as a team, as much as an individual developer, process.

7.1.2 Focus on Key Assurance Techniques

In an example of the Pareto Principle, that 80% of the benefit often derives from 20% of the input[61], section 5.1 shows that the three assurance techniques that are within the scope of most development teams, out of out of twenty in use by industry, are together capable of delivering a large impact.

We conclude that teams will benefit most from concentrating first on these techniques, namely **Automated Static Analysis**, **Threat Assessment** and **Configuration Review**.

7.1.3 Address Organizational Issues

As Figure 5 shows, some 40% of mentions of issues by participants, both of Motivators supporting security improvement, and Blockers discouraging it, are ascribable to organizational issues. Whilst

this finding will not surprise any security professional, it emphasizes the need to regard the promotion of software development security as a systemic, rather than purely a development team, matter.

7.2 Future Work

This research opens up possibilities for future work in two directions: improvement of the interventions themselves, and support for the adoption of the key assurance techniques.

7.2.1 Future Work on Interventions

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 intervention 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.

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. As a longer-term goal, we plan to make the interventions self-sufficient, by providing materials so that intervenors can teach themselves how to carry out the interventions.

Considering the design of the trials, the limited number of development teams involved offers scope for improvement. Furthermore, the changes resulting from the interventions were self-reported: the trials do not provide certainty to what extent 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 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.

7.2.2 Future Work on Assurance Techniques

The importance suggested by Section 4.2 of a small number of assurance techniques provides an incentive for further research on those techniques. While **Automated Static Analysis** and **Penetration Testing** have received a good deal of research attention, participant comments, especially Blockers and Motivators, suggest areas for inquiry for others:

Threat Assessment Participants requested example expert assessments for different domains and types of software, to act as a basis for their own assessments.

Configuration Review	Several Blockers suggest a need for improvements to tools and to vulnerability databases to support more fine-grained component analysis.
Code Review	Traditional line-by-line code review may not be optimal for security issues: one participant, for example, described instead asking developers to show in their code how they addressed specific security issues. There is a need for experimentation investigating the merits of different approaches.
Product Negotiation	Participants requested methods to express specific security improvements as organization benefits; and ways to identify the probability of different security breaches.
Incentivization Session	Alternatives to the Agile Security Game include Capture-the-flag games, Penetration Test-based sessions, and case study-based training. While this work proves the success of the first, research would be valuable to compare other approaches in differing situations.
On-the-job Training	The interventions provide only a one-off security improvement. Games such as the "covert saboteur" in Team A offer opportunities for developers to develop their skills further. However, as we saw, the effectiveness of such approaches depends on personal aspects and team dynamics (Section 6.2.1). Research is needed to provide low-time-cost ways to continue the team security improvement process.

8 CONCLUSION

In this paper we have studied the effectiveness of a series of lightweight interventions to promote secure software development. We facilitated six hours of workshops with each of three teams, and conducted interviews beforehand, immediately afterwards and after twelve months.

This paper proves that low-cost interventions by facilitators who are not security experts can provide long-term security improvements within development teams. It identifies a shortlist of effective assurance techniques that such interventions may aim to introduce, and highlights aspects of the intervention that support their introduction. We found ample motivators for secure development for organisations to provide, and a range of blockers for them to address.

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 to 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. https://www.forbes.com/sites/stevemorgan/2016/05/09/top-2016-cybersecurity-reports-out-from-att-cisco-dell-google-ibm-mcafee-symantec-and-verizon/. Published 2017. Accessed September 25, 2017.

2. Xie J, Lipford HR, Chu B. Why Do Programmers Make Security Errors? In: *IEEE Symposium on Visual Languages and Human Centric Computing*. ; 2011:161-164. doi:10.1109/VLHCC.2011.6070393

3. Xiao S, Witschey J, Murphy-Hill E. Social Influences on Secure Development Tool Adoption: Why Security Tools Spread. In: *ACM Conference on Computer Supported Cooperative Work*. ; 2014:1095-1106. doi:10.1145/2531602.2531722

4. Such JM, Gouglidis A, Knowles W, Misra G, Rashid A. Information Assurance Techniques: Perceived Cost Effectiveness. *Comput Secur*. 2016;60:117-133. doi:10.1016/j.cose.2016.03.009

5. Weir C, Becker I, Noble J, Blair L, Sasse MA, Rashid A. Interventions for Software Security: Creating a Lightweight Program of Assurance Techniques for Developers. In: *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice*. IEEE; 2019.

6. Oliveira D, Rosenthal M, Morin N, Yeh K-C, Cappos J, Zhuang Y. It's the Psychology Stupid: How Heuristics Explain Software Vulnerabilities and How Priming Can Illuminate Developer's Blind Spots. In: *Proceedings of the 30th Annual Computer Security Applications Conference (ACSAC14)*. ; 2014. doi:10.1145/2664243.2664254

7. Derr E, Bugiel S, Fahl S, Acar Y, Backes M. Keep Me Updated: An Empirical Study of Third-Party Library Updatability on Android. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security - CCS '17*. New York, New York, USA: ACM Press; 2017:2187-2200. doi:10.1145/3133956.3134059

8. Nayak K, Marino D, Efstathopoulos P, Dumitraş T. Some Vulnerabilities Are Different Than Others: Studying Vulnerabilities and Attack Surfaces in the Wild. In: *International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*. ; 2014. doi:10.1007/978-3-319-11379-1_21

9. Vaniea K, Rashidi Y. Tales of Software Updates: The Process of Updating Software. In: *Proceedings for Computer Human Interaction (CHI) 2016*. ; 2016:3215-3226. doi:10.1145/2858036.2858303

10. Turpe S. The Trouble with Security Requirements. In: *Proceedings - 2017 IEEE 25th International Requirements Engineering Conference, RE 2017*. ; 2017:122-133. doi:10.1109/RE.2017.13

11. Senarath A, Arachchilage NAG. Why Developers Cannot Embed Privacy into Software Systems? In: *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering (EASE18)*. ; 2018:211-216. doi:10.1145/3210459.3210484

12. Senarath AR, Arachchilage NAG. Understanding User Privacy Expectations: A Software Developer's Perspective. *Telemat Informatics*. 2018;35(7):1845-1862. doi:10.1016/j.tele.2018.05.012

13. Christakis M, Bird C. What Developers Want and Need From Program Analysis: An Empirical Study. In: *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering - ASE 2016*. New York, New York, USA: ACM Press; 2016:332-343. doi:10.1145/2970276.2970347

14. Witschey J, Zielinska O, Welk A, Murphy-Hill E, Mayhorn C, Zimmermann T. Quantifying Developers' Adoption of Security Tools. In: *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2015*. New York, New York, USA: ACM Press; 2015:260-271. doi:10.1145/2786805.2786816

15. Smeets YR. Improving the Adoption of Dynamic Web Security Vulnerability Scanners. 2015.

16. Xie J, Chu B, Lipford HR, Melton JT. ASIDE: IDE Support for Web Application Security. In: *27th Annual Computer Security Applications Conference*. ; 2011:267. doi:10.1145/2076732.2076770

17. Pribik I, Felfernig A. Towards Persuasive Technology for Software Development Environments: An Empirical Study. In: *International Conference on Persuasive Technology*. Springer; 2012:227-238. doi:10.1007/978-3-642-31037-9_20

18. Nguyen DC, Wermke D, Backes M, Weir C, Fahl S. A Stitch in Time: Supporting Android Developers in Writing Secure Code. In: *ACM SIGSAC Conference on Computer & Communications Security*. ACM; 2017. doi:0.1145/3133956.3133977

19. Do LNQ, Ali K, Livshits B, Bodden E, Smith J, Murphy-Hill E. Just-in-time Static Analysis. In: *26th ACM SIGSOFT International Symposium on Software Testing and Analysis*. ; 2017:307-317. doi:10.1145/3092703.3092705

20. Tabassum M, Watson S, Lipford HR. Comparing Educational Approaches to Secure Programming : Tool vs. TA. In: *Thirteenth Symposium on Usable Privacy and Security (SOUPS17)*. ; 2017. https://www.usenix.org/system/files/conference/soups2017/wsiw2017-tabassum.pdf.

21. Xie J, Lipford HR, Chu BB-T. Evaluating Interactive Support for Secure Programming. In: *SIGCHI Conference on Human Factors in Computing Systems*. CHI '12. New York, NY, USA: ACM; 2012:2707-2716. doi:10.1145/2207676.2208665

22. De Win B, Scandariato R, Buyens K, Grégoire J, Joosen W. On the Secure Software Development Process: CLASP, SDL and Touchpoints Compared. *Inf Softw Technol*. 2009;51(7):1152-1171. doi:10.1016/j.infsof.2008.01.010

23. Conradi R, Dybå T. An Empirical Study on the Utility of Formal Routines to Transfer Knowledge and Experience. *ACM SIGSOFT Softw Eng Notes*. 2001;26(5):268-276. doi:10.1145/503271.503246

24. Hardgrave B, Davis F, Riemenschneider C. Investigating Determinants of Software Developers' Intentions to Follow Methodologies. *J Manag Inf Syst*. 2003;20(1):123-151. doi:10.1080/07421222.2003.11045751

25. Lavallee M, Robillard PN. The Impacts of Software Process Improvement on Developers: A Systematic Review. In: *34th International Conference on Software Engineering, ICSE 2012*. ; 2012:113-122. doi:10.1109/ICSE.2012.6227201

26. Riemenschneider CK, Hardgrave BC, Davis FD. Explaining Software Developer Acceptance of Methodologies: A Comparison of Five Theoretical Models. *IEEE Trans Softw Eng*. 2002;28(12):1135-1145. doi:10.1109/TSE.2002.1158287

27. Geer D. Are Companies Actually Using Secure Development Life Cycles? *IEEE Comput*. 2010;June:12-16.

28. Building Security In Maturity Model | BSIMM. https://www.bsimm.com/. Accessed April 14, 2019.

29. Caputo DD, Pfleeger SL, Sasse MA, Ammann P, Offutt J, Deng L. Barriers to Usable Security? Three Organizational Case Studies. *IEEE Secur Priv*. 2016;14(5):22-32. doi:10.1109/MSP.2016.95

30. Bell L, Brunton-Spall M, Smith R, Bird J. *Agile Application Security: Enabling Security in a Continuous Delivery Pipeline*. Sebastopol, CA: O'Reilly; 2017.

31. Kirlappos I, Beautement A, Sasse MA. "Comply or Die" Is Dead: Long Live Security-Aware Principal Agents. In: *Financial Cryptography and Data Security*. Heidelberg: Springer Berlin; 2013:70-82.

32. Kirlappos I, Parkin S, Sasse MA. Shadow Security as a Tool for the Learning Organization. *ACM SIGCAS Comput Soc*. 2015;45(1):29-37. doi:10.1145/2738210.2738216

33. Pfleeger SL, Sasse MA, Furnham A. From Weakest Link to Security Hero: Transforming Staff Security Behavior. *J Homel Secur Emerg Manag*. 2014;11(4):489-510. doi:10.1515/jhsem-2014-0035

34. Fogg BJ. A Behavior Model for Persuasive Design. In: *Proceedings of the 4th International Conference on Persuasive Technology*. Persuasive '09. ACM; 2009:40:1--40:7. doi:10.1145/1541948.1541999

35. Myers RM, Tietjen MA, Myers RM. Motivation and Job Satisfaction. *Manag Decis*. 1998;36(4):226-231. doi:10.1108/00251749810211027

36. Türpe S, Kocksch L, Poller A. Penetration Tests a Turning Point in Security Practices? Organizational Challenges and Implications in a Software Development Team. In: *Second Workshop on Security Information Workers*. USENIX Association; 2016. https://www.usenix.org/conference/soups2016/workshop-program/wsiw16/presentation/turpe.

37. Poller A, Kocksch L, Türpe S, Epp FA, Kinder-Kurlanda K. Can Security Become a Routine? A Study of Organizational Change in an Agile Software Development Group. In: *ACM Conference on Computer Supported Cooperative Work*. ; 2017:2489-2503. doi:10.1145/2998181.2998191

38. Werlinger R, Hawkey K, Botta D, Beznosov K. Security Practitioners in Context: Their Activities and Interactions with Other Stakeholders within Organizations. *Int J Hum Comput Stud*. 2009;67(7):584-606. doi:10.1016/j.ijhcs.2009.03.002

39. Haney JM, Lutters WG. Skills and Characteristics of Successful Cybersecurity Advocates. In: *Third Workshop on Security Information Workers*. USENIX Association; 2017. https://www.usenix.org/system/files/conference/soups2017/wsiw2017-haney.pdf.

40. Ur Rahman AA, Williams L. Software Security in DevOps: Synthesizing Practitioners' Perceptions and Practices. In: *Proceedings of the International Workshop on Continuous Software Evolution and Delivery - CSED '16*. New York, New York, USA: ACM Press; 2016:70-76. doi:10.1145/2896941.2896946

41. Ashenden D, Lawrence D. Security Dialogues : Building Better Relationships. *IEEE Secur Priv Mag*. 2016;14(3):82-87.

42. Stransky C, Acar Y, Nguyen DC, et al. Lessons Learned from Using an Online Platform to Conduct Large-Scale , Online Controlled Security Experiments with Software Developers. *CSET '17 (USENIX Work Cyber Secur Exp Test)*. 2017.

43. Acar Y, Stransky C, Wermke D, Mazurek ML, Fahl S. Security Developer Studies with GitHub Users: Exploring a Convenience Sample. In: *Proceedings of the Thirteenth Symposium on Usable Privacy and Security (SOUPS17)*. ; 2017. https://www.usenix.org/conference/soups2017/technical-sessions/presentation/acar.

44. Tahaei M, Vaniea K. A Survey on Developer-Centred Security. In: *Under Review*. ; 2019:14.

45. Glaser BG, Strauss AL. *The Discovery of Grounded Theory: Strategies for Qualitative Research*. Chicago: Aldine Transaction; 1973.

46. Stol K, Ralph P, Fitzgerald B. Grounded Theory in Software Engineering Research: A Critical Review and Guidelines. In: *38th International Conference on Software Engineering*. ACM; 2015:120-131. doi:http://dx.doi.org/10.1145/2884781.2884833

47. Cooperrider DL, Whitney D. *Appreciative Inquiry: A Positive Revolution in Change*. ReadHowYouWant; 2005.

48. Sharp H, Dittrich Y, De Souza CRB. The Role of Ethnographic Studies in Empirical Software Engineering. *IEEE Trans Softw Eng*. 2016;42(8):786-804. doi:10.1109/TSE.2016.2519887

49. Dittrich Y, Rönkkö K, Eriksson J, Hansson C, Lindeberg O. Cooperative Method Development: Combining Qualitative Empirical Research With Method, Technique and Process Improvement. *Empir Softw Eng*. 2008;13(3):231-260. doi:10.1007/s10664-007-9057-1

50. Whyte WF. *Participatory Action Research*. Sage Publications, Inc; 1991.

51. Baskerville RL. Investigating Information Systems with Action Research. *Commun AIS*. 1999;2(3es):4. doi:10.17705/1cais.00219

52. Petersen K, Gencel C, Asghari N, Baca D, Betz S. 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*. ACM; 2014:55-62. doi:10.1145/2647648.2647656

53. Weir C. The Agile App Security Game – Leader's Instructions. https://www.securedevelopment.org/resources/agile-security-game/. Accessed April 14, 2019.

54. Frey S, Rashid A, Anthonysamy P, Pinto-Albuquerque M, Naqvi SA. The Good, the Bad and the Ugly: A Study of Security Decisions in a Cyber-Physical Systems Game. *IEEE Trans Softw Eng*. 2017:1-16. doi:10.1109/TSE.2017.2782813

55. Shostack A. *Threat Modeling: Designing for Security*. John Wiley & Sons; 2014.

56. Microsoft. Microsoft Secure Development Lifecycle. https://www.microsoft.com/en-us/sdl/. Accessed March 29, 2018.

57. EE. Cash On Tap from EE. YouTube. https://www.youtube.com/watch?v=51CJNfRDuiI. Published 2014. Accessed September 19, 2018.

58. Haney JM, Lutters WG. It's Scary… It's Confusing… It's Dull: How Cybersecurity Advocates Overcome Negative Perceptions of Security. In: *Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018)*. ; 2018:411-425. https://www.usenix.org/conference/soups2018/presentation/haney-perceptions.

59. Shah AK, Mullainathan S, Shafir E. Some Consequences of Having Too Little. *Science (80-)*. 2013;338(6107):682-685. doi:10.1126/science.1222426

60. Weir C, Blair L, Becker I, Sasse MA, Noble J. Light-touch Interventions to Improve Software Development Security. In: Yao D, Chong S, eds. *IEEE Cybersecurity Development Conference*. Boston, MA, USA: IEEE Computer Society; 2018:12. doi:10.1109/SecDev.2018.00019

61. Cheng R. *Genetic Algorithms and Engineering Optimization*. Wiley-Interscience; 2000.

APPENDIX A: SURVEY INTERVIEW QUESTIONS

Introduction – establish context

- What is your current role, and what do you find yourself doing day-to-day? Tell me about a typical day at work?
- Briefly, how did you first get involved with secure software development?

Exploration

- What's your interest in security? What do you do about it, and how do you deal with it day-to-day?
- What do you want to achieve when you're helping a team improve software security? How do you define and measure success?
- What is the most successful intervention technique you've found? Where do you concentrate your efforts?
- Can you think of a particular triumph in your work – where you've worked with a team that has improved their security? How did you achieve that?
- Have any of your teams used code checking tools? How happy were you with their effectiveness at finding problems; and their ease of use?
- What do you find effective as motivation for secure development?
- How do you frighten developers into security, or emphasize the positive aspects?
- To what extent are laws and standards helpful in getting teams to be effective at software security? How do you find out about them and keep up to date?
- When new people join an existing team, how do you motivate them and how do they learn what's required? Do you encourage double checking of contributions from new people or treat them "as usual"
- What are the best ways you've found to get teams to tackle specific things:
    - Security coordination with other teams;
    - Reviews and penetration testing;
    - Designing to get feedback from the users?
    - What else?
- Have you had a nightmare scenario? Or consider this nightmare scenario. You're working with a team that's just learned they have a security flaw in a website that's very heavily used. Have you even had a situation like that (no details required)? What did or would you do to help the team tackle it?

Vision

Let's imaging we're a few years in the future, and the problem of getting teams up to speed with app security has been licked; it's now a part of everyday software development life. How was it done? What were the first small steps?

Clarification (as appropriate)

- And how did you achieve that?
- Oh I see. Could you give an example?