How to improve the security skills of mobile app developers? Comparing and contrasting expert views

Charles Weir
Security Lancaster
Lancaster University, UK
+44-7876-027350
c.weir1@lancaster.ac.uk

Awais Rashid
Security Lancaster
Lancaster University, UK
+44-1524-510316
a.rashid@lancaster.ac.uk

James Noble
Victoria University
Wellington, NZ
+64-4-4635233
kjx@ecs.vuw.ac.nz

ABSTRACT
Programmers' lack of knowledge and ability in secure development threatens everyone who uses mobile apps. There's no consensus on how to empower app programmers to get that knowledge. Based on interviews with twelve industry experts we argue that the discipline of secure app development is still at an early stage. Only once industry and academia have produced effective app developer motivation and training approaches shall we begin to see the kinds of secure apps we need to combat crime and privacy invasions.

CCS Concepts
• **Security and privacy~Software security engineering** • *Security and privacy~Social aspects of security and privacy* • *Software and its engineering~Programming teams*

Keywords
secure development, software security, app security, secure app development, app development, app programmer, app developer, mobile app, whole system security, penetration testing, continued learning, application security, secure app, security issue

1. INTRODUCTION
The past ten years has seen a massive growth in the creation and usage of mobile phone and tablet apps. Increasingly those apps are handling sensitive information about us: controlling our financial transactions, enabling our personal communication and social networking and holding the intimate details of our lives. So the security of those apps is becoming increasingly vital.

In this context, it's disturbing to find that some 73% of US app development professionals, interviewed in a recent IBM-sponsored survey [16], believe that developer lack of knowledge of secure coding practice is a major concern. Analysis of existing apps also gives us reason for disquiet. Enck et al [7] analyzed some 1100 commercial Android apps in 2011 and found privacy issues in a majority of them . Bluebox, an app security solution provider, analyzed the top five payment apps in 2015 [2] and found both vulnerabilities permitting financial theft and privacy issues in all of them.

Virtually all of the vulnerabilities reported in these papers were due to choices by the programmers developing the apps; they could have made choices that didn't lead to the issues. Thus the security of users and data depends vitally on programmers' security practices.

So it's important to improve the effectiveness of developers at producing secure apps. There are three research questions that address this effectiveness, all worthwhile:
(1) What kinds of security errors do programmers make?
(2) How do we improve the systems and compilers that support the developers in their work; and
(3) How can we improve the security skills of the app developers themselves?

There has been a good deal of work on the first question such as the previously-mentioned work by Enck and Bluebox, or work by Xie et al [20] exploring the reasons why programmers make security errors. Various projects – including Xie et al's IDE enhancements [19], compiler improvements and libFuzzer's testing support [13] – address the second question. Taking the third question, however, there is little understanding how and why app programmers learn security and what approaches are likely to work best. This work starts to address that gap.

This paper draws on an ongoing project exploring how app programmers learn security. We suggest a simple model of the motivations of an app programmer, and explore how different experts' approaches relate to that model. We explore the aspects of app security that were accepted by all the experts; and highlight some of the differences.

Based on this exploration, we argue that app security is still at an early stage. This has significant implications for potential implementers. In particular they have choices to make about what aspects of app security and secure processes are appropriate to their projects, and these decisions are not yet codified in an industrywide shared understanding.

2. BACKGROUND
We found little existing work about how programmers learn application security. There is a selection of books and papers aiming to provide the information the programmers need to know. Examples include the 'security patterns' movement of the early 2000's such as 'Security Patterns…' by Schumacher+ [17], which provides a range of information from abstract process to detailed implementation. More recent books are Gary McGraw's 'Software Security' [14], which provides a process-based approach; or Howard, LeBlanc and Viega's '24 Deadly Sins of Software Security' [11], which concentrates on classic programmer security errors. More popular with programmers[1] are

[1] Based on Amazon.com book rankings at March 2016.

books targeted specifically at particular platforms, such as Application Security for the Android Platform [18] or Learning iOS Security [1], both of which restrict themselves to exploring the security features of each respective platform.

Other effective learning resources include Microsoft's classic application developer website on application security [15], which has everything from the Microsoft Security Development Lifecycle to details about the security use of specific Microsoft tools and environments; and the OWASP community-written 'Developer Guide' [21], providing a more general app-specific guide to security issues.

However we found relatively little literature on *how programmers learn* and nothing specific to app programmers. Johnson and Senges [12] studied how programmers learned to function in a complicated organization, Google. They concluded that the majority of programmer learning there was peer learning, facilitated by strong corporate standards and culture. Other studies have incorporated the concepts of programmer learning into the wider term of Software Process Improvement (SPI). So for example a study by Dyba [6] examines learning as one aspect of SPI, differentiating Exploitation, the dissemination of existing knowledge, from Exploration, the gaining of new knowledge; it concludes that both have a positive effect on productivity but doesn't explore mechanisms.

A little-known work by Enes [8] used interviews to discover how professionals, including programmers, acquire their expert knowledge. It concludes that the preferred learning mechanisms are all informal ones: especially on-the job training and personal interaction. It also highlights, as an important factor, professional pride in having 'expert areas' of competence.

In the context of learning about software security, a particularly important finding is that of Conradi and Dyba [4]. This identifies that programmers had difficulty with, and resisted, learning from the output of process improvers, and particularly from formal written routines. This suggests an 'impedance mismatch' between those who write instructions and processes for developers, and the developers themselves who are expected to carry them out. We can speculate that security experts tend to think in terms of complete lists of issues and ways to break software; developers think in terms of simplest and quickest ways to create desired functionality.

3. OUR RESEARCH

The purpose of our research was to provide ways to improve app software development, motivated by personal observation that app security was difficult to learn. So our approach to this research was pragmatic. We used aspects of different methodological approaches to get the most effective results.

Our method was to interview experts in app security. We used semi-structured interviews with app development specialists, and analyzed them using Grounded Theory [3,9].

We had observed that much of the research and thinking in security appears negative from the point of view of an app developer: criticisms of existing software; penetration testing; analysis of exploits. In the context of the 'impedance mismatch', we were looking instead for positive approaches to security learning. Thus the nature of the questions and the thrust of the analysis were guided by a further research method, Appreciative Inquiry [5]. This led our emphasis in the interviews to be the positive techniques the experts had discovered or used, and where and why those were particularly successful.

We chose our interviewees opportunistically, mainly through introductions from former colleagues. Time and practicality limited the number of interviews to a dozen. Guest [10] suggests that further interviews would be unlikely to generate much in the way of further new theory. In practice we believe that we have probably reached this 'theoretical saturation point' with respect to app security techniques (Section 7), but not with respect to the contrasts among experts (Sections 5 and 6)

Table 1: The experts interviewed

ID	Organisation type	Typical role
P1	Bespoke app developer	Developing apps for business clients; author on app security
P2	Mobile phone manufacturer	Leader of large team specialising in security
P3	Operating system supplier	Developer of user-facing web services
P4	Smart card specialists	Design and implementation of smart card software
P5	Security-related SaS supplier	Architecting and promoting a secure service
P6	Promoting industry	App security consultancy
P7	Mobile phone manufacturer	Developer and software architect for OS services
P8	Telecoms service provider	Architecting mobile phone services
P9	Bank	Analysis, design and implementing changes to web-based services
P10	Secure app technology provider	Architecting and promoting app technologies
P11	Operating system supplier	Designing and promoting security enhancements
P12	Bespoke app developer	Developing apps for business clients

Table 1 gives an overview of the experts interviewed. For each, we have given an indication of the nature of the companies they are currently involved with and their typical role. Some were contractors working for more than one organization; for them this shows the organization they work with most. All had more than 20 years' experience in software development. All were currently working in some way with secure app development; all but P1 had at least 5 years' experience working with secure software development; and all but P5 and P8 had backgrounds as a

software developer. Probably typically of their roles in this industry, all were male.

Some of the interview questions related to the experts' analysis of how to achieve secure app development; others to their own history and ways of learning about secure app development. Thus we can distinguish two forms of information from the interviews: information about how the app security experts had achieved their expertise and kept themselves updated, and information they had about best learning approaches for those working with them.

In this paper we quote extensively from the interviews. To convey correctly the context and protect the confidentiality of the interviewees, we've amended the quotations appropriately; square brackets show additions and replacements; ellipses show removals.

4. RESEARCH MODEL

We observed that the experts differed widely in their original reasons for learning about software security. And there was correspondingly little agreement on how best to motivate app programmers generally to produce good secure apps.

Our analysis of the interviews highlighted four forces motivating a programmer to learn and act on software security, as illustrated in Figure 1.

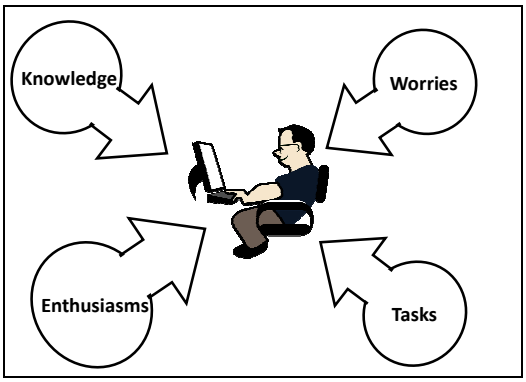

Figure 1: Motivation forces on a programmer

These forces are:

Knowledge the knowledge and skills that the programmers have learned in the past or gained through experience on how to deal with software security issues.

Tasks the formal and informal assignments of code to write, changes to make, training, and related work that the programmer has as their overt job.

Worries the concerns and fears the programmer has about what they are doing.

Enthusiasms the positive inspirations that motivate the programmer to make specific choices.

We found a tension between these as two pairs of alternatives: those who saw knowledge as a motivation didn't feel the need for tasks and vice versa; those who felt worries were a motivation didn't consider enthusiasm and vice versa.

So where an expert's interview expressed a position on these forces, we express that position as a location on each scale. For example, an expert who expressed strong views that security should be part of every relevant activity in software development would be represented at the 'knowledge' end of the scale; an expert who mildly suggested that security could be included as the tasks of penetration testing and app hardening would place towards the 'tasks' end of the scale.

We found similar tensions between approaches to different views on implementing security and on the role of teamwork. These tensions we also represented on appropriate scales.

Thus in diagrams in the following sections 5 and 6 we position the views or information expressed by experts on specific topics against axes representing two related scales. Each diagram shows only the experts who expressed a clear opinion, and shows clusters where several shared roughly the same position. The resulting pattern highlights the range of views expressed.

5. REASONS FOR LEARNING

5.1 Experts' reasons for learning

Thus Figure 2 expresses the original motivations for the experts themselves for learning about software security.

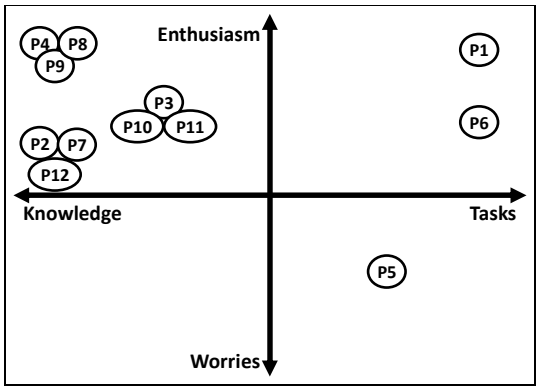

Figure 2: Reasons for original learning

As it shows, these varied significantly. Most had learned from day-to-day experience, given enthusiasm for the security aspects of that experience; some had started as hackers:

"Actually when I was a kid – fortunately, I never released any of this stuff – I did actually take copy protection off games for the intellectual challenge of this" (P3)

Others had started on projects which required security:

"[While at college] I had three very fun summers working on top secret projects and things like that, which had a fair amount of security in it." (P12)

"I did a lot of firmware work on a magnetic stripe card reader ... that had a number of security features… I definitely got the [security] bug there". (P4)

Only P1 had decided to learn about software security as a career decision – to build experience and credibility in a new area.

5.2 Experts' reasons for continued learning

We also analyzed experts' reasons for continued learning. Here we have more consistency; for most it's an out-of-band task in addition to their normal day job, and they do it on an ad-hoc basis. Only P3 and P11, who work for a global, security aware, company, receive security-related training; and P1, in his role as author, assigns app security learning as part of his normal work" Most kept up to date through a background task of following appropriate internet media – Bruce Schneier's update email was the most commonly mentioned medium (P3, P7, P8), or:

"My work screen has a Twitter feed just running up the right hand side. Whenever I get to enough of a break that I can glance over I'll take a look at whatever is currently up there." (P7)

"I listen to a few podcasts… Security Now… with Steve Gibson on the TWiT Network" (P12)

Figure 3 shows the experts' reasons for their continued learning, where these were discussed.

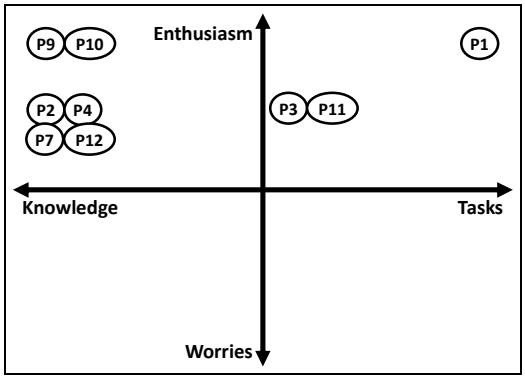

Figure 3: Motivation for continued learning

5.3 Motivating programmers to learn

All the interviewees who discussed the point stressed that programmers had a tendency to avoid security issues and concentrate on delivering functionality. Some highlighted that few undergraduate level computing courses incorporate security into normal examples and practice.

"So for the majority of people who are currently going through various computer science degrees, security doesn't really come into it at all, in any real context". (P10)

Many correlated general life experience, software development experience, and especially formal software development experience with ability at software security. Those who discussed it stressed the difficulty in motivating inexperienced developers:

"When I'm talking to 22 year old phenomenally brilliant mathematician software developer who has got almost no life experience at all – how do I make him care about things that seem unimportant to him?" (P5)

However the interviewees showed little consistency in their approaches to solving this problem and motivating programmers to work on security, as follows.

5.3.1 Enthusiasm or worry?

Some wanted security as an enthusiasm, wanting programmers to be passionate about doing a good job on security:

"trying to talk to my developers about this and trying to come up with techniques that make them think about it in a way that makes them care about it" (P5)

Others felt it should be a worry, where the impact of poor security is a threat to the programmers:

"We'll need a mass security event [caused by a mobile app] to get programmers to take app security seriously" (P1)

5.3.2 Knowledge or task-based?

Some represented making systems secure as part of a process, where developers do the right thing because they are expert and knowledgeable:

"So you are going to have to get developers to understand computer science, and the consequences of the code they are writing" (P6)

Others saw the adding of and planning of security as part of the functionality requirements and thus as a specific task.

"Mine is much more practical. I need it to work, I'll put something together that actually does the job, and I will learn whatever I need to learn to do that. And then move on, if necessary." (P4)

We observe that these external motivators for programmers naturally follow the same axes as the motivators the experts had had for their own learning. Figure 4 shows how the experts who expressed views are positioned on the same axes.

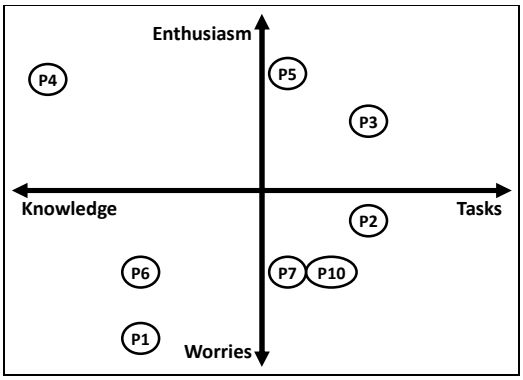

Figure 4: Opinions on how to motivate app programmers

6. IMPLEMENTING SECURITY

We also saw differences in how the experts suggested achieving app security.

6.1 Approach to teamwork on security

There were differences in how they viewed team interaction and their roles.

6.1.1 Teamwork vs individual rigor

Some stress communication between and within teams:

"And I think one thing that we were incredibly good at with [a specific project], is bringing the entire project team together probably with the aid of, as well as the formal meetings, some of the more casual discussions over a beer. And so everybody fully understood the scope of what everyone was bringing to the table and there was never any of the artificial formalities that sometimes you can get around these projects where it feels uncomfortable to pick the phone up to somebody." (P8)

Others stressed individual rigor, as their primary tool. For example:

"I tend to look at things in a stepwise way. Certainly when you're evolving software, you don't necessarily have formal proof but you can go in sufficiently simple steps that you can see that it's obviously correct." (P12)

We saw the latter view expressed usually related to single developer situations where there weren't others with whom to discuss security.

6.1.2 Influencing vs directing

Another distinction that emerged is that some saw their best means for they themselves to influence the team members as directive, exerting authority:

"I had success [by] whacking them over the head with a wet fish" (P7, speaking metaphorically).

Whilst others saw their role as influencing, questioning and encouraging:

"[I] throw out a few 'what ifs' you know, what if I did that, and get somebody who is aware and will have an understanding of what you are suggesting, and they will counter with a sensible response." (P8).

Figure 5 shows these two contrasts.

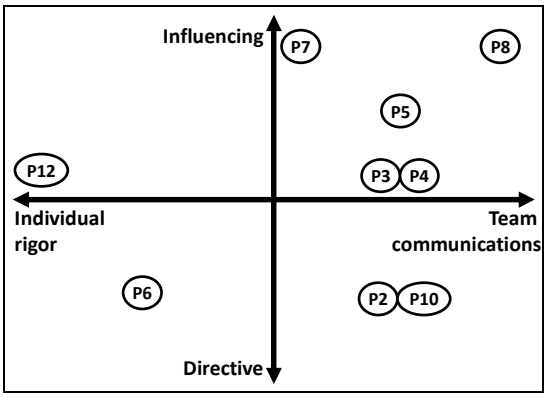

Figure 5: Expectation of team interaction

Though one might expect the choice of influencing vs directing to reflect the expert's authority in the organization, in fact this was not necessarily the case. For example P5's role gave him authority and P7 was referring to peers.

6.2 Approach to security

In terms of knowledge transfer and implementing app security there were also differences between experts. These were more nuanced, reflecting differences in emphasis.

6.2.1 Checklists or whole system security?

Some experts preferred a checklist, excellent-coding attitude to security:

"Checklists I think are wonderful things. And if they are Why, How, What, Where, When, not just 'does it' – it's not just a 'yes / no'. It's a checklist that goes, in what way have you done this?" (P5)

Others stressed the importance of various aspects of Whole System security:

"I would just wish that education was better and that developers understood about separation of code and data and Saltzer and Schroeder's 8 principles of computer security, and understood the background more and focussed less on the top 10 vulnerabilities – what they happen to be this year." (P6)

6.2.2 Concentrate on attacker or stakeholder?

There was an interesting distinction as to whether the emphasis was more on attackers, or on stakeholders such as product managers. Some emphasized the importance of understanding and reacting to different kinds of attackers:

"You also try and understand why someone is coming to your service in the first place. And try to give them what they want up front, so they lose interest and go away." (P9)

"I think it is actually very important to understand the motivations behind why somebody is hacking the system. We try to address the motivations of the attackers, versus the technical aspects - just locking it down for the sake of locking it down." (P11)

Others emphasized the importance of negotiation with stakeholders on what security was put in the product:

[When I started] a project I'd go back and ask [my customer]…'You do realize this [information] can be seen'. It goes from there: 'how secure do you want it to be?' You have to show that there's a problem first I think" (P1)

Figure 6 shows these differences in emphasis.

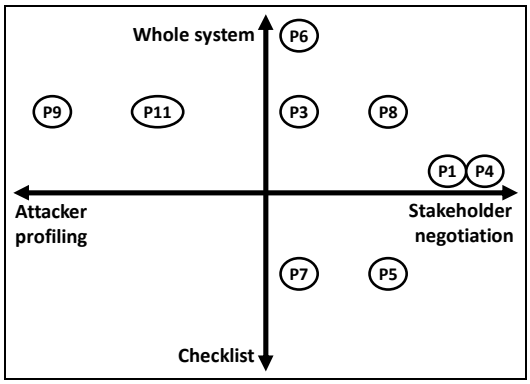

Figure 6: Preferred approach to app security

7. WHAT TO LEARN

Our purpose in the research was to work towards helping app developers in future. To work towards a practical positive output we looked therefore for development techniques that lead to good app security. There was no clear consensus, but we did identify a range of approaches each championed by at least one of the interviewees, and each rejected by none. Thus we synthesized agreement on important things for app programmers to learn.

Many of our interviewees were clear that app programming techniques and choices alone wouldn't be sufficient. So they take as a given approaches such as the OWASP lists and penetration testing, and they build on that.

"We said 'right – these checklist things don't work, because products are too different, and so on, actually what we are looking at is more…' – and to be fair, these days, you would call them architecture patterns rather than design patterns because they are at a higher level than typical design patterns." (P6)

This section explores some of the 'best practice points' raised in the interviews. They're grouped under four categories: analysis, dialectic development, continuous feedback, and continuous enhancement.

7.1 Analysis

Analysis covers the programmer involvement in work outside the main design and coding of their apps. In some projects this will be upfront work; in agile projects it is likely to continue throughout the development life-cycle.

Important features of analysis include agreement on the level of security and the processes involved:

"It depends on the client, how much money they've got, how much time they want, how much time they've got to get to market. And you have to compromise all the processes, how you do it, and the degree to which you do it based on them. There is no One Way. It's the same with security." (P1)

To that, they added security-aware choices of libraries and environments; ideation sessions working with stakeholders and penetration testing experts of different possible exploits on the system; and formal or informal risk assessments of the likelihood of each exploit and its possible impact.

"One of the things I like to do with the [penetration testing] guys is to, if you sit down and say 'what are all the different ways you could subvert this system'. It is quite common to come up with 20, 30, 40, 50 in five or ten minutes of brainstorming. I bet you, you wouldn't think of half of them." (P2)

7.2 Dialectic development

'Dialectic' means the finding out of knowledge through one person questioning another. Because of the adversarial nature of security, many of the most effective techniques for finding security issues are dialectic. Some of the techniques recommended were penetration testing, code reviews, pair programming, and a variety of code analysis tools.

"I think the one [approach] that has been, arguably, most useful has been using specialist external consultancy around security. Not for training, but 'can you just come in and penetration test this device'" (P2)

"Nothing gets submitted without it being reviewed by at least another engineer. And there are strong processes to protect that fact. … The most successful technique has to be review by [a security] expert – you can't really beat that – an actual conversational review by an expert, because someone who is an expert in security might not be an expert in the domain." (P3)

"[The most successful technique I have found is] to use various types of Lint checkers" (P7)

7.3 Continuous feedback

Many interviewees stressed the interactive nature of defending apps. In order to deal with new exploits and analyze existing ones, developers need access to information about what is happening to the deployed apps.

This is more difficult with apps than with in-house or cloud-based systems, since apps are deployed remotely and may not have continuous communication with any central point. App developers typically need to put in extra functionality and have support processes to ensure they receive feedback. These range from delayed logging back to a central server:

I've built quite a bit into the apps where they have their own debug logs because I don't trust the likes of Google because they have to sanitize what they give you because they've got privacy issues on their side of things. (P12)

to offering bounty for people who report possible weaknesses:

"We pay people to report bugs to us" (P11)

Based on this feedback programmers can analyze new security issues and plan fixes into the development stream for the future.

7.4 Continuous enhancement

The interviewees also stated that software needs to continue to change throughout its lifetime in order to remain sufficiently secure. New exploits, improved processing power, and wider publication of existing exploits all mean that what might have been secure a year ago may not be now.

"Security is a process and update rates are an important part of that process" (P3)

Two aspects make this particularly difficult for apps. First, the process of upgrading an app's code is difficult. It usually requires a new release via an 'App Store', whether for a mobile app or a new OS version; but many users may choose not to upgrade.

"The moment you release something to [a mobile phone OS], you will, in general, never get a 100% update rate, because loads of people [install] software once and never update." (P3)

So developers of mobile apps need to consider whether functionality is required to work around this.

Second is the nature of app development 'contracts', whether internal to a company or commercial external contracts. In many cases app development is seen as 'fire and forget'; on completion of the initial app development phase, the team is allocated to different projects.

"Like many things that get delivered in a project, the project ends and interest dies with it. Unfortunately. And I think you lead into a significant challenge in securing things on an operational basis". (P8)

Secure app development therefore requires a different, continued development, approach to support sufficient security maintenance.

8. WHAT NEXT?

We observed in section 7 a lack of consistent emphasis on different secure app development techniques, and we observed in section 5 notable differences of opinion on how to motivate programmers to security, as highlighted by the spread of the points in Figure 4. Section 6 showed even stronger contrasts in experts' approaches to teamwork in Figure 5, and their approaches to app security in Figure 6.

The authors had experienced a similar lack of consistency in the early days of both the object oriented design paradigm (OOD) and the Agile development paradigm, each of which in due course converged into well accepted approaches: around UML and Scrum respectively. In the early days of each there were many good ideas and many experts championing different aspects of those ideas; the current situation in secure app development has a similar character. This suggests that the discipline of app development security is still at an early stage.

The convergence around UML and Scrum led to greatly increased programmer acceptance and knowledge of OOD and Agile development respectively. We suggest that similar convergence in app development security will lead to greatly improved programmer knowledge in that area. Looking at the history of object oriented design and agile development we believe two steps are likely to lead to this convergence. First is the codification of the main principles by well-respected experts in a popular form: a book, online resource, or even video. Second is the championing of that codification by one or more large commercial organizations. Microsoft and Google are likely contenders, but both are tainted by their commitments to specific mobile platforms so it remains to be seen which organization may champion a global approach.

9. CONCLUSION

In this paper we have highlighted the risks associated with app security, and identified the importance of ensuring programmer

motivation to program apps securely and of improving their skills at doing so.

From our research we showed in Section 5 that there is little similarity in people's motivation to learn software security, nor consensus on how to motivate app developers to do so. Section 6 showed diverging opinions on the use of teamwork and on the best approach to implementing security. And Section 7 highlighted valuable 'whole system security' approaches to apply to app development despite a lack of industry-wide consensus on them.

The lack of agreement is a significant challenge to improving the skills of app programmers, and Section 8 highlighted the importance of working towards an industry-wide consensus based on research.

Thus future research is needed to explore objectively the most effective ways to motivate app developers in different contexts to learn and use secure development practices. And research is required to clarify the best security practices in different app development contexts and to discover the most effective ways for programmers to learn them.

10. REFERENCES

[1] Banks, A. and Edge, C.S. *Learning iOS Security*. Packt Publishing, 2015.

[2] Bluebox Security. 'Tis the Season to Risk Mobile App Payments - An Evaluation of Top Payment Apps. (2015).

[3] Charmaz, K. *Constructing grounded theory*. Sage, London, 2014.

[4] Conradi, R. and Dybå, T. An empirical study on the utility of formal routines to transfer knowledge and experience. *ACM SIGSOFT Software Engineering Notes 26*, 5 (2001), 268–276.

[5] Cooperrider, D.L. and Whitney, D. Appreciative inquiry: a positive revolution in change. *Appreciative Inquiry*, (2005), 30.

[6] Dybå, T. An empirical investigation of the key factors for success in software process improvement. *IEEE Transactions on Software Engineering 31*, 5 (2005), 410–424.

[7] Enck, W., Octeau, D., McDaniel, P., and Chaudhuri, S. A Study of Android Application Security. *USENIX security symposium*, 2011. http://www.usenix.org/event/sec11/tech/slides/enck.pdf.

[8] Enes, P. and Conradi, R. Acquiring and Sharing Expert Knowledge. 2005. http://www.idi.ntnu.no/grupper/su/fordypningsprosjekt-2005/aanes-fordyp05.pdf.

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

[10] Guest, G., Bunce, A., and Johnson, L. How many interviews are enough? An experiment with data saturation and variability. *Field methods 18*, 1 (2006), 59–82.

[11] Howard, M., LeBlanc, D., and Viega, J. *24 Deadly Sins of Software Security: Programming Flaws and How to Fix Them*. McGraw-Hill, Inc., 2009.

[12] Johnson, M. and Senges, M. Learning to be a programmer in a complex organization. *Journal of Workplace Learning 22*, 3 (2010), 180–194.

[13] LLVM Project. libFuzzer. http://llvm.org/docs/LibFuzzer.html.

[14] McGraw, G. *Software security: building security in*. Addison-Wesley Professional, 2006.

[15] Microsoft. Learning Security - MSDN. https://msdn.microsoft.com/en-us/security/aa570420.aspx.

[16] Ponemon Institute. *The State of Mobile Application Insecurity*. 2015.

[17] Schumacher, M., Fernandez-buglioni, E., Hybertson, D., Buschmann, F., and Sommerlad, P. *Security Patterns: Integrating Security and Systems Engineering*. John Wiley & Sons, 2005.

[18] Six, J. *Application Security for the Android Platform*. O'Reilly, Sebastapol, CA, 2011.

[19] Xie, J., Chu, B., Lipford, H.R., and Melton, J.T. ASIDE: IDE support for web application security. *Proceedings of the 27th Annual Computer Security Applications Conference on - ACSAC '11*, (2011), 267.

[20] Xie, J., Lipford, H.R., and Chu, B. Why do programmers make security errors? *Proceedings - 2011 IEEE Symposium on Visual Languages and Human Centric Computing, VL/HCC 2011*, (2011), 161–164.

[21] OWASP Developer Guide. https://github.com/OWASP/DevGuide.