Article

Security-focused Prototyping: A Natural Precursor to Secure Development

Sam Attwood* [1], Nana Onumah [2], Katie Paxton-Fear [2] and Rupak Kharel [3]

[1] Department of Computing and Mathematics, Faculty of Science and Engineering, Manchester Metropolitan University, Manchester, United Kingdom; S.Attwood@mmu.ac.uk; Nana-Kwesi.A.Onumah@stu.mmu.ac.uk; K.Paxton-Fear@mmu.ac.uk; R.Kharel@mmu.ac.uk

* Correspondence: S.Attwood@mmu.ac.uk

Version March 15, 2021 submitted to J. Cybersecur. Priv.

Abstract: Secure development is a proactive approach to cyber security. Rather than building a technological solution and then securing it in retrospect, secure development strives to embed good security practices throughout the development process and thereby reduces risk. Unfortunately, evidence suggests secure development is complex, costly, and limited in practice. This article therefore introduces security-focused prototyping as a natural precursor to secure development that embeds security at the beginning of the development process, can be used to discover domain specific security requirements, and can help organisations navigate the complexity of secure development such that the resources and commitment it requires are better understood. Two case studies–one considering the creation of a bespoke web platform and the other considering the application layer of an Internet of Things system–verify the potential of the approach and its ability to discover domain specific security requirements in particular. Future work could build on this work by conducting case studies to further verify the potential of security-focused prototyping and even investigate its capacity to be used as a tool capable of reducing a broader, socio-technical, kind of risk.

Keywords: cyber security; secure development; prototyping; web security; internet of things; software security; digitalization; socio-technical security

1. Introduction

Technological advances create both opportunities and challenges. New technologies have the potential to enable new solutions that change human lives for the better. However, the same new technologies also have the potential to create new challenges. Often, these challenges are related to security, as new technologies inevitably lead to new vulnerabilities for malicious actors to exploit. The emergence of the Internet of Things (IoT) in recent times serves as an example: industrial applications of IoT technologies are synonymous with the idea of a fourth industrial revolution (Industry 4.0) and optimistic predictions of the value created by industrial IoT range as high as $15 trillion of global GDP by 2030 [1], however, IoT devices can also be leveraged by malicious actors as part of Distributed Denial-of-Service attacks without the end user's knowledge [2].

Secure development models provide a means of minimizing risk when creating a technological solution and are therefore especially useful tools when working with emerging technologies. Unfortunately, secure development is complex, costly, and limited in practice [3]. There are numerous secure development models and no universally accepted answer as to which is the best. Furthermore, many of the prominent secure development models are not always applicable. Geer surveyed 46 organisations and found that only the most technically sophisticated–approximately 10% of respondents–were adopting a secure development model [3]. Many of the respondents reported that they had not adopted a secure development model because it was either too expensive, required

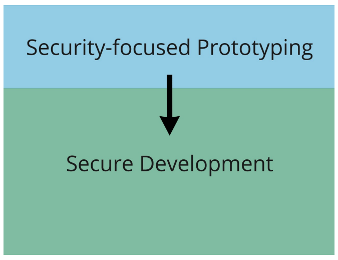

Figure 1. A visualization of how security-focused prototyping promotes the adoption of secure development practices.

too many resources, or was too time consuming [3], and numerous studies present a similar narrative [4–7]. This is despite the simple idea of prevention being better than cure arguably encapsulating the entire concept.

The COVID-19 pandemic has highlighted the need for secure development models that are flexible and viable when time is in short supply. It has seen governments around the globe race to launch contact-tracing smartphone applications whilst citizens, partially motivated by the apparently hurried development, simultaneously raise security and privacy concerns [8]. In short, the pandemic created a situation in which rapid, yet secure development was required; a situation that required a development model with the facets of rapid development approaches–such as Boehm's spiral model [9] and subsequent agile models [10]–but was nonetheless capable of producing a secure solution that addressed privacy concerns.

The challenges faced by small and medium sized organisations (SMEs) further highlight the need to make secure development more easily accessible. In general, SMEs are perceived as being less equipped when it comes to cyber security than larger organisations [11] and this is true of secure development as well; Assal and Chiasson surveyed 123 developers and found that SMEs were more likely to have 'competing priorities and no plan' and be 'unequipped for security' than larger enterprises [7]. The COVID-19 pandemic has only exaggerated these challenges and placed a greater pressure on SMEs to digitalize and to do it quickly [12]. Unfortunately, the various barriers–including the requirement for security–to SMEs looking to digitalize are well documented [13–15]. Furthermore, when it comes to security these barriers must be overcome repeatedly; the threat landscape is forever changing and organisations with a digital presence (of all sizes) must change with it.

The Greater Manchester Cyber Foundry project further evidences the need for a more accessible model of secure development that is viable when time is in short supply. Broadly speaking, the project consists of 2 phases, the second of which requires the creation of proof-of-concept demonstrators over relatively short timescales. These proof-of-concept demonstrators serve a dual purpose and are intended to increase innovation (and thereby economic growth), but also the adoption of security practices, across SMEs in the Greater Manchester (UK) region. The challenge faced by the project is that the short timescales over which the proof-of-concept demonstrators need to be developed make it difficult to adopt a well-established secure development model. So how can the proof-of-concept demonstrators that are created as a part of the project help increase the adoption of security practices?

To address all these problems we present a novel technique–security-focused prototyping–that acts as a precursor to secure development (see figure 1) by embedding security at the beginning of the development process, discovering domain specific security requirements, and providing a means of understanding the level of resources and commitment that are required for secure development to continue. We also present 5 aspects of secure development that are intended to act as a streamlined yet informative definition of secure development (and are the foundations upon which security-focused prototyping is built). We start by describing the largely disparate worlds of rapid and secure development in section 2, then synthesize the two by introducing security-focused prototyping in section 3, before validating the approach by describing its application to two case studies in section 4, and ultimately concluding by discussing the potential of the technique in section 5.

2. Background

2.1. Rapid Development & Prototyping

The drive towards rapid development and lightweight software development models arguably began when Boehm introduced the spiral model in 1988 [9]. The model was motivated by perceived shortcomings in the waterfall model of software development, as well as its prevalence [9]. It was differentiated from prior approaches by the fact that it was risk-driven; the prior approaches were more document-driven or code-driven [9].

In a later work, Boehm states that spiral development is a family of software development processes that are characterized by repeatedly iterating a set of elemental development processes and managing risk such that it is actively being reduced [16]. Importantly, the risk that is being managed here is the risk that the delivery of the project will be delayed or fail [16]; the spiral model does not manage security risks to the extent of the models discussed in section 2.2. Boehm also goes on to describe 6 characteristics, or invariants, that those following a spiral model should observe [16]:

- Concurrent determination of key artifacts. Sequential determination often leads to mistakes. For example, a premature commitment to an inappropriate platform or service provider.
- Consideration in each spiral of the main spiral elements. A failure to revisit the key objectives and risks before undertaking activities could cause time to be wasted on activities that are unacceptable to key stakeholders.
- Level of effort driven by risk considerations. Any development activity is only beneficial up to a point. For example, prototyping can minimise risk exposure, but excessive prototyping can delay a project.
- Degree of detail driven by risk considerations. Artifacts are only beneficial up to a point. For example, a detailed specification of a graphical user interface risks an awkward design being embedded into the development process.
- Use of anchor point milestones. The original spiral model lacked intermediary milestones that could serve as progress checks. This led to to issues such as requirements creep and unrealistic expectations.
- Emphasis on system and lifecycle activities and artifacts. Software construction activities should not overshadow other activities as this can lead to the concerns and objectives of stakeholders being lost.

Shortly after Boehm had described the invariants that those following a spiral model should observe, seventeen software developers gathered to discuss similarly lightweight software development models [10]. The result of this gathering was the agile manifesto [10]. Today, many software development models can be considered agile and agile models are prevalent across the software engineering discipline [17]. The twelve principles that underpin these models are as follows (and outlined fully in the agile manifesto [10]):

- Satisfy the customer. Deliver valuable software early on and on a regular basis to achieve this.
- Welcoming changing requirements. Even late on in the development process.
- Deliver working software frequently. The greater the frequency the better.
- Stakeholders and developers should work together. Ideally, this will be on a daily basis.
- Build projects around motivated individuals. Give these individuals the support they need and trust them to get the job done.
- Promote efficient and effective communication. Put value in face-to-face communication to achieve this.
- Working software is the primary measure of progress.
- Promote sustainable development. Stakeholders and developers should be able to maintain a constant pace.
- Continuous attention to technical excellence and good design enhance agility.

- Simplicity is essential. Try and maximise the amount of work not done.
- Self organizing teams typically produce the best results. The best architectures, designs, and requirements normally emerge from these teams.
- Reflect at regular intervals and identify ways to improve. Then tune and adjust behaviours accordingly.

Most of the agile development models (and Boehm's spiral model in particular) place considerable importance on prototyping as an activity and use it as a kind of insurance. Prototyping allows for something tangible–the prototype itself–to be created early on in the development process and therefore provides a crucial basis for discussion and ideation. Moreover, if any problems stemming for the underlying assumptions and requirements of a project exist, prototyping allows for them to identified early on before they become too costly. The question *What is prototyping?* is answered more comprehensively by Budde et al. who start by putting forward the following four points to serve as a preliminary characterisation of the term [18]:

- Prototyping is an approach based on evolutionary view of software development. It strives to have an impact on the development process as a whole.
- Prototyping involves producing early working versions of a system. So that these early versions can be experimented with.
- Prototyping provides a basis for discussion among all the groups involved in the development process. Users and developers in particular.
- Prototyping informs further development. Experience gained via prototyping and experimentation is fed into further development.

Budde et al. discuss several different aspects of prototyping in greater detail [18]. One of these aspects–the distinction between horizontal and vertical prototyping–is of particular significance with regards to this work. According to Budde et al., in horizontal prototyping specific individual layers of a system are built, and in vertical prototyping a selected part of the target system is implemented completely (down through all layers) [18]. If we consider a web application as an example, the act of prototyping the user interface layer could be described as horizontal (see left of figure 2), whereas the act of prototyping a new feature–across the user interface layer but other layers as well–could be described as vertical (see right of figure 2).

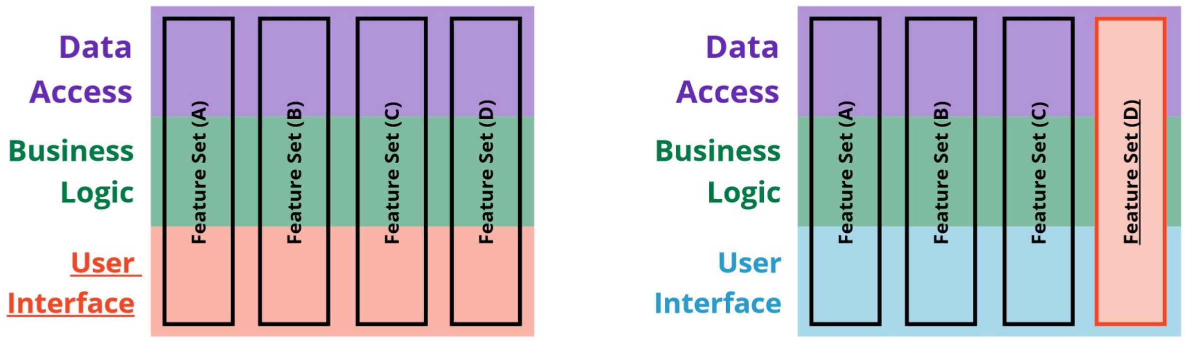

Figure 2. A visualization of horizontal prototyping (left) and vertical prototyping (right). The focus of the prototyping is highlighted by underlined text and red background.

2.2. Secure Development

In 2002, the Trustworthy computing memo was sent to all Microsoft employees [19]. The memo sought to lessen customer concerns and bad press caused by security concerns [19,20]. It defined trustworthy computing as *computing that is available, reliable and secure as electricity, water services and telephony* [19]. It triggered a rethink within Microsoft and ultimately led to the Microsoft Security Development Lifecycle (MSDL) [20]. Today, 12 practices make up the MSDL [21]. Table 1 presents

Table 1. The Microsoft Security Development Lifecycle (●), software security touchpoints (▲), and SAFE Code practices (■) mapped to the six phases of development put forward by De Win et al: education and awareness (4.1); project inception (4.2); analysis and requirements (4.3); architectural and detailed design (4.4); implementation and testing (4.5); release, deployment, and testing (4.6) [22]. Based on a similar table in the Secure Software Lifecycle Knowledge Area [20].

Practice	**4.1**	**4.2**	**4.3**	**4.4**	**4.5**	**4.6**
Provide training	●					
Plan the implementation of secure development	■	■				
Manage security findings	■	■	■	■	■	
Define metrics and compliance reporting		●				
Define and use cryptography standards		●				
Use approved tools		●				
Abuse cases			▲			
Define security requirements			● ▲ ■			
Perform threat modelling			●			
Design			■	■		
Establish design requirements				●		
Architectural risk analysis			▲	▲	▲	
Code reviews					▲	
Perform static analysis security testing					●	
Perform dynamic analysis security testing					●	
Testing and validation					■	
Manage risk of third-party components					● ■	
Follow secure coding practices					■	
Risk-based security testing					▲	
Perform penetration testing					● ▲	● ▲
Establish a standard incident response						●
Vulnerability response and disclosure						■
Security operations						▲

these practices in a summarized form (for brevity) and maps them to six common development phases forward by De Win et al. [22]. Considered as a whole the MSDL is a comprehensive process. Table 1 shows that it specifies at-least one practice for each of the development phases. However, the MSDL does have a significant shortcoming–the practices it contains explain *what* should be done, but not *how* it should be done.

In 2004, McGraw proposed seven software security touchpoints (a set of best practices) [23]. These touchpoints are presented and mapped across the six development phases first put forward by De Win et al [22] in table 1. From table 1 a shortcoming of the touchpoints can be identified, none of them cover the 'education and awareness' and the 'project inception' phases. This lack of coverage at the earliest development phases is problematic and could result in teams being under-prepared. Furthermore, much like the practices in the MSDL, many of the touchpoints specify *what* should be done but fail to go into significant detail as to *how* it should be done. However, in both cases this lack of detail is most likely deliberate and in a sense is what makes the models valuable. By failing to specify exactly *how* practices should be implemented both models remain technology and process agnostic, which means both can be considered generally applicable.

Several years after the touchpoints were introduced, in 2007, the Software Assurance Forum for Excellence in Code (SAFE Code) was founded [24]. SAFE Code is an industry-led non-profit organization with the goal of promoting and facilitating the adoption of effective secure development practices [24]. In pursuit of this goal, the organisation publishes guidance that is centred around 8 fundamental practices [25]. Table 1 shows that these practices are comprehensive and together span the six phases of development. However, the point about practices specifying *what* to do and not *how* to do it applies here as well. Like the MSDL and the touchpoints, the SAFE Code practices are largely technology and process agnostic, which is often perceived as a strength but is also a weakness.

Even though the practices are sound, the lack a systematic method for realizing them is a barrier organizations looking to achieve secure development must overcome.

Table 1 demonstrates that there is some overlap between the MSDL, the touchpoints, and the SAFE Code guidance. It shows that all three recommend defining security requirements and that the MSDL and touchpoints both recommend penetration testing be performed. Furthermore, there are similarities between the three approaches that are not immediately apparent from table 1. For example, all three recommend code reviews but do so in subtly different ways. The MSDL draws a distinction between static and dynamic analysis security testing [21], the touchpoints simply recommend code reviews and note that these can be manual or automated [23], and SAFE Code advises code reviews be performed along with other activities like penetration testing by recommending that testing and validation be performed [25].

However, table 1 also demonstrates that there are significant differences between the three secure development models. For example, the touchpoints differ from the other two models in offering no guidance related to the planning and implementation of secure development. Furthermore, there are differences in the detail of similar recommendations that are not demonstrated by table 1. Both the MSDL and SAFE Code provide guidance with regards to the management of third-party components, but the detail of this guidance is subtly different in a number of ways. SAFE Code maps mitigate, monitor, and assess activities onto a third-party component management lifecycle [25], whereas the MSDL simply lists 4 practices that can be adopted [21].

Table 1 therefore evidences and demonstrates the complexity of secure development (as a whole, not any model in particular). Complexity which has led to research and the formulation of further models that provide a means of assessing and evaluating other secure development models [22,26,27]. The Software Assurance Maturity Model (SAMM) [26], and an early fork of it known as the Building Security In Maturity Model (BSIMM) [27], being particularly notable examples. The SAMM and BSIMM models are both comprehensive and provide a means for organisations to improve via the incremental adoption of security practices. Nonetheless, both of the models fail to provide domain specific recommendations and the narrative of secure development being too costly, requiring too many resources, and being too time consuming has continued since their formulation [3–7], which suggests further work is still needed to make secure development more accessible.

The complexity of secure development is further exaggerated by the variety of technologies that can be developed today. In general, secure development refers to the secure development of software (and this is true of all the models described so far) but software itself is always changing. Software can be developed for mobiles, it can be delivered via cloud computing, and it can form a part of an Internet of Things (IoT) system. Each of these scenarios has unique challenges and corresponding guidance associated with it [28–30]. Together, they therefore not only exaggerate the complexity of secure development, they also demonstrate that it is highly domain-dependent. Different practices, tools, and threat actors need to be considered when building solutions with different technologies and for different purposes. The prominent models discussed in this section, and most secure development models in general, are technology agnostic and therefore fail to offer to domain specific guidance.

3. Security-focused Prototyping: Streamlining Secure Development

In this section, we attempt to synthesize the worlds of prototyping and secure development, and propose a new technique termed security-focused prototyping before explaining its potential. However, before doing this we briefly review a body of related works, such that this work is clearly distinguished from them. Broadly speaking, this body of related works considers the synthesis of agile development with secure development [31–37]. For example, SAFE Code provides security guidance for agile practitioners [32], Microsoft lists 8 practices as a part of a Secure DevOps model [33], and a recent study proposes and examines a new model that is centered around 23 principles with the aim of ensuring both agility and security [34]. Put briefly, our work is differentiated from this body of related works due to our in-depth focus on prototyping.

Prior models that synthesize agile and secure development have failed to tap into the potential of prototyping. Security-focused prototyping taps into this potential and thereby provides a means of: embedding security at the beginning of the development process, discovering domain specific requirements, and navigating the complexity of secure development such that the resources and commitment it requires are better understood. The prior models do a good job when it comes to the first point–embedding security at the beginning of (and throughout) the development process–but they evidently fail to address the latter two. Even when secure development is agile organisations can struggle with regards to resources and commitment [38,39]. Furthermore, because the guidance and recommendations in agile secure development models are typically technology agnostic [32–34], these models (like non-agile secure development models) can fall short of delivering the domain specific guidance that secure development requires.

Synthesizing the worlds of prototyping and secure development can address these issues and as a first step towards this goal we propose 5 aspects of secure development. Each of these aspects was arrived at by examining well-established secure development models–the Microsoft Secure Development Lifecycle, the seven software security touchpoints, and the various practices that are recommended by SAFE Code–and considering what their practices hope to achieve. The aspects are as follows (listed in order of eminence):

1. *Dualistic*. Secure development consists of both constructive (building) and destructive (breaking) activities such that risk is minimised but delivery is not hampered.
2. *Pessimistic*. Secure development assumes that the solution being developed will come under attack. As such, even constructive activities are done with malicious actors in mind.
3. *Holistic*. Secure development promotes security throughout the entire development process, the entire development team, and even throughout the entirety of the organisation that is undertaking the project.
4. *Auditable*. Secure development puts value in transparent reporting and honest communication. The security requirements that must be gathered in many models are perhaps the best demonstration of this but any approach that recommends clean and clear comments has an auditable aspect.
5. *Cyclical*. Secure development encourages repeated reviews in an attempt to counter a threat landscape that is always evolving. Many models encourage continuous, or at least regular, training and testing.

The primary purpose of these aspects of secure development is to act as a guide for security-focused prototyping; if a prototyping activity captures these aspects then it is on its way to being security-focused. That said, these aspects may be of use beyond security-focused prototyping as well. The aspects are an attempt to streamline secure development and distill it down to its essence. Any organisation that is looking to achieve secure development may therefore benefit from these aspects and use them as a lightweight form of guidance. Furthermore, because the aspects strive to be very general (more so than the technology agnostic guidance of the models discussed in section 2.2) they may be of use beyond software development and could act as a vessel through which the lessons of secure software development are transferred across to other technological developments.

The second (and final) step towards synthesising the worlds of prototyping and secure development is to build upon the preexisting idea of vertical prototyping. As is described in section 2.1, vertical prototyping involves implementing a selected part of a target system down through all of its constituent layers. This vertical kind of prototyping, combined with the aspects of secure development, can act as a powerful preparatory process that paves the way for secure development. By implementing features across all the layers we get an idea of what needs to be done to secure each layer individually, but also how the layers interact with one another, and therefore what needs to be done to secure the system as a whole. Different tools and activities will be needed for different layers and it is only by implementing features across them all, while simultaneously attempting to capture the 5 aspects of secure development, that we can achieve a complete understanding.

4. Case Studies

To demonstrate the potential of security-focused prototyping this section considers two case studies. Both of the case studies consider the application of security-focused prototyping within the context of technical assistance delivered as a part of The Greater Manchester Cyber Foundry project. As is noted in section 1, The Greater Manchester Cyber Foundry project requires the creation of proof-of-concept demonstrators over relatively short timescales. These proof-of-concept demonstrators serve a dual purpose and are intended to drive further innovation while simultaneously promoting the adoption of good security practices. The project is therefore an ideal test-bed to verify the potential of security-focused prototyping.

4.1. Case Study: The Security-focused Prototyping of a Bespoke Web Platform

Digital Oracles Ltd are building a web platform that connects technology start-ups, industry experts and early stage investors. The Greater Manchester Cyber Foundry provided technical assistance towards this goal in the form of security-focused prototyping. This security-focused prototyping involved the implementation of a subset of desirable features and helped establish an architecture that could be built on in further work. It also sought to mitigate a number of vulnerabilities that are intrinsic to products and services that are delivered using the Web. If exploited, these vulnerabilities could inflict reputational or financial harm on Digital Oracles Ltd. For example, if a malicious actor was able to exploit a SQL injection vulnerability this could result in data being corrupted or exposed to unauthorised parties, which would of course damage the reputation of Digital Oracles Ltd.

The prototyping that was done to assist Digital Oracles Ltd was security-focused and therefore captured the 5 aspects of secure development described in section 3:

- The dualistic aspect was captured via a practice of periodically switching between constructive and destructive activities. So, after being constructive and implementing an authorisation guard we switched to being destructive and tried to bypass said guard.
- The pessimistic aspect was captured via the usage of a code scanning tool–named Security Code Scan–and automated checks for vulnerabilities in third-party dependencies.
- The holistic aspect was captured via regular meetings with stakeholders that helped us consider financial/organisational constraints and think about how these constraints impacted the technical implementation.
- The auditable aspect was captured via the use of the Git version control system [40], architecture decision records [41], and minutes of meetings/interactions with stakeholders.
- The cyclical aspect was captured via the implementation of sprints (a common practice in agile approaches to software development). Work was divided into sprints and at the end of each sprint a meeting/interaction with stakeholders was held. This allowed for everyone to reflect on the previous sprint and think about what needed to be done, and could be done better, in the next sprint.

The feature-set, or vertical, that was the focus of the security-focused prototyping was centered around the start-ups that would be using the platform. Functionality that allowed a start-up to register/login, take a self-impact assessment, and track their progress over time were all implemented as a part of the security-focused prototyping process. All of this functionality was implemented through all of the layers of the system. Notably, our understanding of the layers improved as the process of security-focused prototyping was performed. At the start of the process, we thought of these layers much as they are seen in figure 2 (so user interface, business logic, and data access layers). However, as the security-focused prototyping process was performed and our understanding of the platform (and how it could be secured) improved, a clearer picture began to emerge. Figure 3 illustrates how layers were thought of towards the end of the process and is based on both the SPA + API and Clean architectures [42–44].

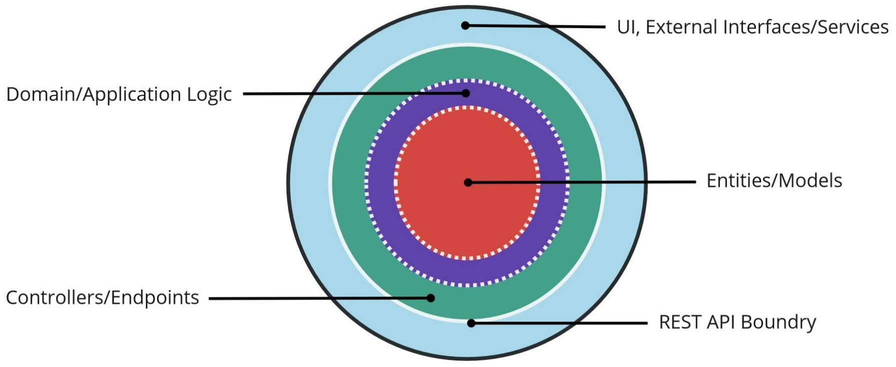

Figure 3. A visualization of how layers were thought of towards the end of the security-focused prototyping process.

As a result of the security-focused prototyping a proof-of-concept demonstrator was produced. This proof-of-concept demonstrator was comprised of a handover document and a repository that contained the source code for the prototype web platform. The handover document communicated the findings of the prototyping and made highly relevant recommendations with regards to further secure development, thereby providing the kind of domain specific guidance many secure development models lack. Table 2 presents a summarized list of the security findings that resulted from the security-focused prototyping process. Importantly, because security-focused prototyping is a form of vertical prototyping, the findings help contribute towards the security of the entire system–throughout all the layers shown in figure 3.

Table 2. Security findings that resulted from the security-focused prototyping done to assist Digital Oracles Ltd.

#	**Security finding**
1	An object relational mapper, such as Entity Framework Core, should be used to reduce the risk posed by SQL injection. This applies to the Domain and Entities layers seen in figure 3.
2	Even if an object relational mapper is used checks (code reviews) should be made to confirm that no concatenated strings are executed against the database. This applies to the Domain layer seen in figure 3.
3	A security code scanner should be used to highlight any potential injection vulnerabilities and cross-site scripting vulnerabilities. Security Code Scan being a good example of a code scanner for the C# programming language. This applies to the Controller, Domain, and Entities layers seen in figure 3.
4	Third party services, such as Auth0, should be used to reduce the burden and risk that come with implementing complex authentication and authorisation functionality. Precise details as to how to configure and integrate Auth0 with specific technologies (and across all the layers seen in figure 3) were included in the handover document.
5	Even if a third party service is used to implement authentication and authorisation functionality care should still be taken to ensure that functions, or endpoints, and individual objects are guarded to an appropriate level (and code reviews can help with this). This mainly applied to the Controller and UI layers seen in figure 3.
6	Error messages returned to end users should be generic and not contain any sensitive information. Samuela Rescsa [45] provides a good explanation of how this can be achieved when using the .NET Core framework.
7	All responses sent from the API component should include security headers. The `X-Content-Type-Options: nosniff` and the `X-Frame-Options: DENY` headers should be sent to mitigate MIME sniffing and click jacking.
8	Audit logs should be written after input validation failures, output validation failures, and application errors.

4.2. Case Study: The Security-focused Prototyping of the Application Layer of an IoT System

Pure O2 Ltd received technical assistance, in the form of security-focused prototyping, towards their ultimate goal of a cyber-physical system that monitors the well-being of patients, alerts medical practitioners or carers when a dangerous situation arises, and (potentially) collects data from which insights can be drawn. The security-focused prototyping focused on the application layer of this planned system. The cyber security challenge the assist faced was therefore similar to the one mentioned in the previous case study (section 4.1); products and services delivered via the Web have intrinsic security vulnerabilities that need to be mitigated. However, the requirement for real-time functionality (to alert medical practitioners or carers) meant that additional technologies and therefore vulnerabilities had to be considered, and that the challenge the prototyping helped overcome was ultimately quite different.

The prototyping that was done to support Pure O2 Ltd was security-focused and therefore captured the 5 aspects of secure development described in section 3. Each of the aspects was captured in much the same way as it was during the prototyping that was done to support Digital Oracles Ltd (see section 4.1). Importantly, this similarity in how the prototyping efforts sought to be dualistic, pessimistic, holistic, auditable, and cyclical, did not stop the security-focused prototyping from uncovering security findings specific to the domain and technologies Pure O2 Ltd was interested in.

Two feature-sets were the focus of the security-focused prototyping. The first of these feature-sets led to a prototype REST API that sends and receives mock data being developed. The second feature-set led to an early version of an alert system, a Hub, built using SignalR [46] and Web Sockets [47] being developed. Together, these two prototypes form an early working version of an application layer, which is illustrated in figure 4. The two feature-sets were chosen as the focus of the prototyping as it was identified early on that these different sets would need to be built across different layers (the REST API and the Hub); by prototyping the chosen sets we were able to hit all of the layers and got a better understanding of how to secure the application layer as a whole.

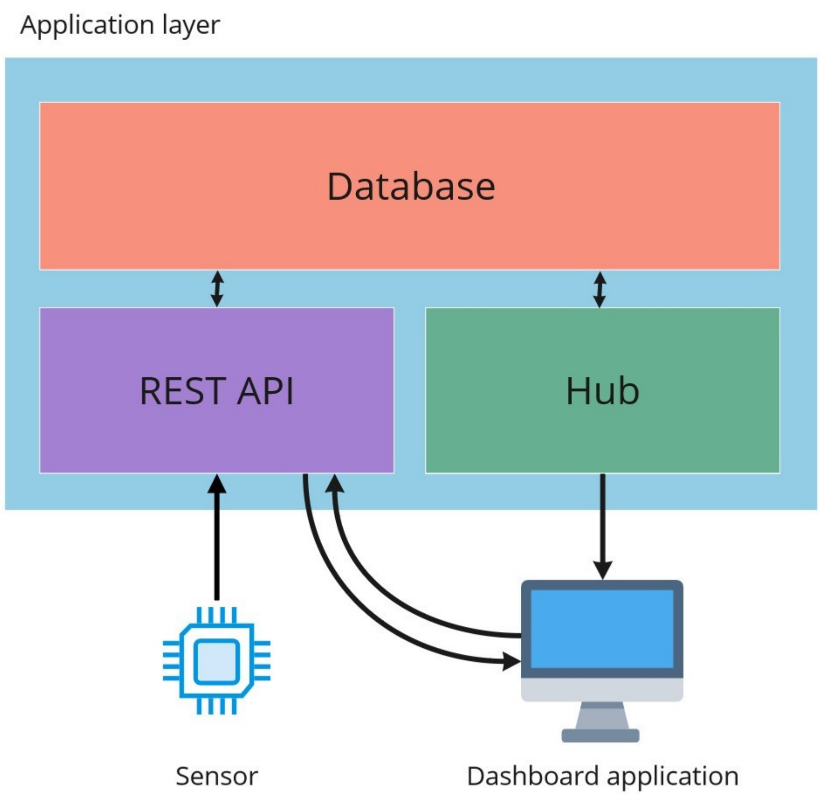

Figure 4. A visualization of the application layer created via security-focused prototyping.

As a result of the security-focused prototyping a proof-of-concept demonstrator was produced. This proof-of-concept demonstrator was comprised of a handover document and several repositories that contained the source code for the application layer of an Internet of Things system. The handover document communicated the findings of the prototyping and made highly relevant recommendations with regards to further secure development, thereby providing the kind of domain specific guidance many secure development models lack (just as it did in the previous case study). Table 3 presents a summarized list of findings that resulted from the security-focused prototyping process.

Table 3. Security findings from the security-focused prototyping done to assist Pure O2 Ltd.

#	**Security finding**
1	An object relational mapper, such as Entity Framework Core, should be used to reduce the risk posed by SQL injection. This applies to the REST API layer seen in figure 4.
2	Even if an object relational mapper is used checks (code reviews) should be made to confirm that no concatenated strings are executed against the database. This applies to the REST API layer seen in figure 4.
3	A security code scanner should be used to highlight any potential injection vulnerabilities and cross-site scripting vulnerabilities. Security Code Scan being a good example of a code scanner for the C# programming language. This applies to the REST API layer seen in figure 4.
4	Error messages returned to end users should be generic and not contain any sensitive information. Samuela Rescsa [45] provides a good explanation of how this can be achieved when using the .NET Core framework. This applies to the REST API layer seen in figure 4. SignalR does not expose sensitive error messages by default and the Hub layer is therefore secure by default in this regard.
5	All responses sent from the REST API component should include security headers. The `X-Content-Type-Options: nosniff` and the `X-Frame-Options: DENY` headers should be sent to mitigate MIME sniffing and click jacking.
6	Audit logs should be written after input validation failures, output validation failures, and application errors. Furthermore, because Signal R sends the access token in a query string when using WebSockets and Server-Sent Events as a transport method, care should be taken to configure the logger such that it only logs messages annotated Warning or higher. Alternatively, logging middle ware that filters out the access token could be written.
7	Cross-origin resource sharing (CORS) can be configured to allow cross-origin SignalR connections in the browser. Care should be taken when defining this configuration and the principle of least privilege should be applied.
8	The protections provided by CORS do not apply to WebSockets. Browsers do not send pre-flight requests when issuing WebSocket requests, but they do send an Origin header. Care should be taken to make sure that these headers are validated.
9	SiginalR uses per-connection buffers to manage messages and limits the size of messages by default. However, this default configuration can be overriden if required. Care should be taken when overriding the default configuration as doing so could allow a malicious actor to send large messages that significantly reduce the number of concurrent connections and impact the availability of the service Pure O2 Ltd plans to provide, which could be fatal and in the worst case result in lives being lost.
10	Third party services, such as Auth0, should be used to reduce the burden and risk that come with implementing complex authentication and authorisation functionality. Auth0 allows for communications between the IoT device and the control system to be secured using the Client Credentials Grant Flow and for communications between the browser-based application and the control system to be secured using the Authorisation Code Flow with PKCE. Further details as to how to configure and integrate Auth0 with the relevant technologies were included in the handover document.
11	Even if a third party service is used to implement authentication and authorisation functionality care should still be taken to ensure that functions, or endpoints, and individual objects are guarded to an appropriate level (and code reviews can help with this).

5. Conclusions

This article introduces security-focused prototyping, a process that can act as a valuable precursor to secure development. Motivated by the complexity of secure development as a topic and the inability of prominent secure development models to provide domain specific guidance, security-focused prototyping streamlines secure development and harnesses the power of prototyping as an activity that reduces risk. Two case studies have been described–one considering the creation of a bespoke web platform and the other considering the application layer of an Internet of Things system. Together these case studies demonstrate the ability of security-focused prototyping to: embed security at the very beginning of the development process, discover domain specific security requirements, and allow for the resources and commitment secure development will need to be better understood.

The potential of security-focused prototyping to industry is therefore clear; it is a process that can be used to better reduce risk. Future work could build on this working hypothesis and further evidence the potential of security-focused prototyping by applying it in different domains. Furthermore, the 5 aspects security-focused prototyping is built around (identified in section 3) have their own value, independent of security-focused prototyping. Future work could use these aspects as a framework for creating new models of secure software development, or even as a vessel through which the principles of secure software development could be transferred across to other technological developments. Ultimately, security-focused prototyping even has potential as a process capable of mitigating a broader, socio-technical, kind of risk–something many prominent secure development models fail to provide.

In summary, prototyping and secure development are analogous processes. By combining the two, security-focused prototyping makes secure development more accessible and thereby contributes towards a future where the benefits new technologies bring can be realised with less risk.

Author Contributions: Conceptualization, S.A and N.O; methodology and investigation, S.A; writing–original draft preparation, S.A; writing–review and editing, S.A, N.O, K.P.F, and R.K; supervision, R.K and K.P.F; funding acquisition, R.K. All authors have read and agreed to the published version of the manuscript.

Funding: This is a direct result of working on the ERDF Greater Manchester Cyber Foundry which is part funded by the European Regional Development Fund.

Acknowledgments: We gratefully acknowledge the support of Digital Oracles Limited and Pure O2 Limited.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

IoT	Internet of Things
SME	Small to medium sized organisation
MSDL	Microsoft Security Development Lifecycle
SAFE Code	Software Assurance Forum for Excellence in Code
SAMM	Software Assurance Maturity Model
BSIMM	Building Security in Maturity Model
SPA	Single-page application
REST	Representational State Transfer
API	Application programming interface
CORS	Cross-origin Resource Sharing

References

1. Daugherty, P.; Banerjee, P.; Negm, W.; Alter, A.E. Driving Unconventional Growth through the Industrial Internet of Things. Technical report, Accenture, 2015.

2. Zhou, W.; Jia, Y.; Peng, A.; Zhang, Y.; Liu, P. The Effect of IoT New Features on Security and Privacy: New Threats, Existing Solutions, and Challenges Yet to Be Solved. *IEEE Internet of Things Journal* **2019**, *6*, 1606–1616. doi:10.1109/JIOT.2018.2847733.
3. Geer, D. Are Companies Actually Using Secure Development Life Cycles? *Computer* **2010**, *43*, 12–16. doi:10.1109/MC.2010.159.
4. Assal, H.; Chiasson, S. Security in the software development lifecycle. Symposium on Usable Privacy and Security. USENIX Association, pp. 281–296.
5. Tondel, I.A.; Jaatun, M.G.; Meland, P.H. Security Requirements for the Rest of Us: A Survey. *IEEE Software* **2008**, *25*, 20–27. doi:10.1109/MS.2008.19.
6. Mohammad, A.; Alqatawna, J.; Abushariah, M. Secure software engineering: Evaluation of emerging trends. 8th International Conference on Information Technology (ICIT), 2017, pp. 814–818. doi:10.1109/ICITECH.2017.8079952.
7. Assal, H.; Chiasson, S. 'Think Secure from the Beginning': A Survey with Software Developers. Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems; Association for Computing Machinery: New York, NY, USA, 2019; CHI '19, p. 1–13. doi:10.1145/3290605.3300519.
8. Criddle, C.; Kelion, L. Coronavirus contact-tracing: World split between two types of app, 2020. Available online: https://web.archive.org/web/20210310122748/https://www.bbc.com/news/technology-52355028 (accessed on 10-03-21).
9. Boehm, B.W. A spiral model of software development and enhancement. *Computer* **1988**, *21*, 61–72. doi:10.1109/2.59.
10. Beck, K.; Beedle, M.; Van Bennekum, A.; others. Manifesto for agile software development. Technical report, 2001.
11. Osborn, E. Business versus technology: Sources of the perceived lack of cyber security in SMEs [Working Paper]. Technical report, 2014.
12. Finn, B. Hundreds of businesses move online as Covid-19 shutters 'bricks and mortar' stores, 2020. Available online: https://web.archive.org/web/20210310134008/https://www.rte.ie/news/business/2020/0407/1129150-covid-19-prompts-hundreds-of-businesses-to-move-online/ (accessed on 10-03-21).
13. Abolhassan, F. Security: The real challenge for digitalization. In *Cyber Security. Simply. Make it Happen.*; Springer, 2017; pp. 1–11.
14. Peillon, S.; Dubruc, N. Barriers to digital servitization in French manufacturing SMEs. *Conference on Industrial Product-Service Systems* **2019**, *83*, 146–150.
15. Kabanda, S.; Tanner, M.; Kent, C. Exploring SME cybersecurity practices in developing countries. *Journal of Organizational Computing and Electronic Commerce* **2018**, *28*, 269–282.
16. Boehm, B.; Hansen, W.J. Spiral development: Experience, principles, and refinements. Technical report, Carnegie Mellon University, 2000.
17. Hoda, R.; Salleh, N.; Grundy, J. The rise and evolution of agile software development. *IEEE software* **2018**, *35*, 58–63.
18. Budde, R.; Kautz, K.; Kuhlenkamp, K.; Züllighoven, H. What is prototyping? *Information Technology & People* **1992**.
19. Bill Gates: Trustworthy Computing, 2002. Available online: https://web.archive.org/web/20210310122257/https://www.wired.com/2002/01/bill-gates-trustworthy-computing/ (accessed on 10-03-21).
20. Williams, L., Secure Software Lifecycle Knowledge Area. In *Cyber Security Body of Knowledge*; University of Bristol, 2019. Version 1.0.
21. What are the Micrsoft SDL practices?, 2021. Available online: https://web.archive.org/web/20210310121322/https://www.microsoft.com/en-us/securityengineering/sdl/practices (accessed on 10-03-21).
22. De Win, B.; Scandariato, R.; Buyens, K.; Grégoire, J.; Joosen, W. On the secure software development process: CLASP, SDL and Touchpoints compared. *Information and software technology* **2009**, *51*, 1152–1171.
23. McGraw, G. Software security. *IEEE Security Privacy* **2004**, *2*, 80–83. doi:10.1109/MSECP.2004.1281254.
24. Our History. Available online: https://web.archive.org/web/20210310140744/https://safecode.org/our-history/ (accessed 10-03-21).

25. Fundamental practices for secure software development: Essential elements of a secure development lifecycle program. Technical report, Software Assurance Forum for Excellence in Code, 2018.
26. SAMM Model Overview, 2020. Available online: https://web.archive.org/web/20210226052500/https://owaspsamm.org/model/ (accessed on 12-03-21).
27. Williams, L.; McGraw, G.; Migues, S. Engineering security vulnerability prevention, detection, and response. *IEEE Software* **2018**, *35*, 76–80.
28. OWASP Mobile Security, 2020. Available online: https://web.archive.org/web/20210312161251/https://owasp.org/www-project-mobile-security/ (accessed on 12-03-21).
29. Takabi, H.; Joshi, J.B.; Ahn, G.J. Security and privacy challenges in cloud computing environments. *IEEE Security & Privacy* **2010**, *8*, 24–31.
30. Hassan, W.H.; others. Current research on Internet of Things (IoT) security: A survey. *Computer networks* **2019**, *148*, 283–294.
31. Azham, Z.; Ghani, I.; Ithnin, N. Security backlog in Scrum security practices. *Malaysian Conference in Software Engineering* **2011**, pp. 414–417.
32. Asthana, V.; Tarandach, I.; O'Donoghue, N.; Sullivan, B.; Saario, M. Practical security stories and security tasks for agile development environments. Technical report, Software Assurance Forum for Excellence in Code, 2012.
33. Secure DevOps, 2021. Available online: https://web.archive.org/web/20210312163014/https://www.microsoft.com/en-us/securityengineering/devsecops (accessed 12-03-21).
34. de Vicente Mohino, J.; Bermejo Higuera, J.; Bermejo Higuera, J.R.; Sicilia Montalvo, J.A. The Application of a New Secure Software Development Life Cycle (S-SDLC) with Agile Methodologies. *Electronics* **2019**, *8*. doi:10.3390/electronics8111218.
35. Pohl, C.; Hof, H. Secure Scrum: Development of Secure Software with Scrum. *ArXiv* **2015**, *abs/1507.02992*.
36. Oueslati, H.; Rahman, M.M.; Othmane, L.B. Literature Review of the Challenges of Developing Secure Software Using the Agile Approach. *10th International Conference on Availability, Reliability and Security* **2015**, pp. 540–547.
37. Bishop, D.; Rowland, P. Agile and Secure Software Development: An Unfinished Story. *Issues in Information Systems*.
38. Oyetoyan, T.D.; Cruzes, D.S.; Jaatun, M.G. An empirical study on the relationship between software security skills, usage and training needs in agile settings. 2016 11th International Conference on Availability, Reliability and Security (ARES). IEEE, 2016, pp. 548–555.
39. Morrison, P.; Smith, B.H.; Williams, L. Surveying security practice adherence in software development. Proceedings of the Hot Topics in Science of Security: Symposium and Bootcamp, 2017, pp. 85–94.
40. Chacon, S.; Straub, B. *Pro Git*; Apress, 2014.
41. Architectural decision record (ADR), 2021. Available online: https://web.archive.org/web/20210311025433/https://github.com/joelparkerhenderson/architecture_decision_record (accessed on 12-03-21).
42. Martin, R. The Clean Architecture, 2012. Available online: https://web.archive.org/web/20210312162015/https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html (accessed on 12-03-21).
43. Fielding, R.T.; Taylor, R.N. Principled design of the modern web architecture. *ACM Transactions on Internet Technology (TOIT)* **2002**, *2*, 115–150.
44. What's the difference between single-page application and multi-page application?, 2017. Available online: https://web.archive.org/web/20210312162539/https://www.adcisolutions.com/knowledge/whats-difference-between-single-page-application-and-multi-page-application (accessed on 12-03-21).
45. Resca, S. *Hands-On RESTful Web Services with ASP.NET Core 3*; Packt, 2019.
46. Introduction to ASP.NET Core SignalR, 2019. Available online: https://web.archive.org/web/20210313155700/https://docs.microsoft.com/en-gb/aspnet/core/signalr/introduction?view=aspnetcore-5.0 (accessed on 13-03-21).
47. The WebSocket Protocol, 2011. Available online: https://web.archive.org/web/20210308025822/https://tools.ietf.org/html/rfc6455/ (accessed on 12-03-21).