Developing Secure Software, assignment 1

During development of software, faults and flaws are introduced either from the implementation or from the design of the software. During runtime these faults and flaws can propagate into failures that can result in vulnerabilities if the right conditions are present. Failures and especially vulnerabilities increase the cost for the developers and require them to spend time on maintenance instead of new features. Many telecom developers rely on testing to reduce their maintenance cost and achieve software with high availability. Unfortunately most of the testing is done to verify functionality and not to find vulnerabilities.

Figure 1

Figure 1 illustrates the problem with most common development process, testing and early fault detection is focused on bugs and verifying requirements and spend little time on any “extra” functionality that might have been added, i.e. vulnerabilities. To improve the software’s security many developers use penetration testing on release products. While the testing does find vulnerabilities it has to be performed late in development after all the functions have been verified as no more functionality is permitted to be added after the penetration testing has passed. Therefore it is expensive to do security verification testing and it adds lead time for the product. Instead the solution might be to improve already present development process to include security. Some attempts like Microsoft’s, Security Development Lifecycle, have been tried with encouraging results. Figure 2 shows a layout of the Security Development Lifecycle.

Figure 2. SDL Overview

I’m focusing in introducing security in a telecom development process, these are often test driven because telecom graded systems require a high uptime and function verification. I plan on using many case studies to examine the effects and benefits of different methods and tools that can be incorporated into an already existing process. Because case studies are hard to quantify I will try and use the waste source of resources in Ericsson and try and acquire data from several products and different sites. Survey studies will also help answering question about the developer’s interaction with the development process. With penetration testing the developers are all security
experts that have at least an education in security and very often several years of programming experience. In an integrated security process it is not possible to guarantee that the developers understand the issues with security problems, it will therefore also be important to examine how easily developers understand what the meaning with security tasks and tools output is. The preferred final goal of my research would be a improved development cycle that is security aware and while it will not catch all security vulnerabilities there is at least a chance or the knowledge of what type of vulnerabilities are detected and witch are not, thus guiding any penetration testing and shortening lead time.
While the goal of the research is to better understands of security during development the research hampered by the ability to measure success. Because security vulnerabilities are by nature unknown, there will never be a conclusive result where any method or tool will provide an exact benefit to the process. Instead the after effects, such as less TR’s, might be used to determine the effectiveness of a security improvement.

Background, lightweight software security, assignment 2

Software security touchpoints are based on good software engineering and involve explicitly pondering security throughout the software lifecycle. This means knowing and understanding common risks (including language-based implementation bugs and architectural flaws), designing for security and subjecting all software artifacts to thorough, objective risk analyses and testing. "Software Security Touchpoints" specifies one set of touchpoints and shows how software practitioners can apply them to the various software artifacts produced during software development. This means understanding how to work security engineering into requirements, architecture, design, coding, testing, validation, measurement and maintenance.

1. All software projects produce at least one artifact: source code. At the code level, the focus is on implementation bugs, especially those that static analysis tools that scan source code for common vulnerabilities can discover. Code review is a necessary practice, but not sufficient for
achieving secure software. Security bugs (especially in C and C++) are a real problem, but architectural flaws wreak just as much havoc. Just as you can't test quality into software, you can't bolt security features onto code and expect it to become hack-proof. Security must be built in throughout the application development lifecycle.

Static code analysis articles:
Improving security using extensible lightweight static analysis, D Evans, D Larochelle

2. At the design and architecture level, a system must be coherent and present a unified security front. Designers, architects and analysts should clearly document assumptions and identify possible attacks. At both the specifications-based architecture stage and at the class-hierarchy design stage, risk analysis is a necessity. At this point, security analysts uncover and rank architectural flaws so that mitigation can begin. Disregarding risk analysis at this level leads to costly problems down the road. Note that risks crop up during all stages of the software lifecycle, so a constant risk analysis thread, with recurring risk tracking and monitoring activities, is highly recommended.

Risk analysis articles:
Attack Trees, B Schneier
Attack Modeling for Information Security and Survivability, AP Moore, RJ Ellison, RC Linger
Assessment and control of software risks, C Jones

3. Penetration testing is also useful, especially if an architectural risk analysis is driving the tests. It provides a good understanding of fielded software in its real environment, but any such testing that doesn't take the software architecture into account probably won't uncover anything interesting about software risk. Software that fails during the kind of canned black-box testing practiced by prefab application security testing tools is truly bad. Thus, passing a low-octane penetration test reveals little about your actual security posture, but failing a canned penetration test indicates that you're in very deep trouble indeed.

Testing security articles:
Exploiting Software: How to Break Code, G Hoglund, G McGraw

4. Security testing must encompass two strategies: testing security functionality with standard functional testing techniques, and risk-based security testing based on attack patterns. A good security test plan does both. Security problems aren't always apparent, even when you probe a system directly, so standard-issue quality assurance is unlikely to uncover all critical security issues.

Risk base testing articles:
Risk-based testing: Risk analysis fundamentals and metrics for software testing including a financial application case study, S Amland

5. Building abuse cases is a great way to get into the mind of the attacker. Similar to use cases, abuse cases describe the system's behavior under attack; building abuse cases requires explicit coverage of what should be protected, from whom, and for how long.

Abuse cases articles:
Using abuse case models for security requirements analysis, J McDermott, C Fox
6. Security must reside explicitly at the requirements level. Good security requirements cover both overt functional security (say, the use of applied cryptography) and emergent characteristics (best captured by abuse cases and attack patterns).

Requirements and abuse case articles:
Eliciting security requirements with misuse cases, G Sindre, AL Opdahl
Security Requirements Engineering: When Anti-requirements Hit the Fan, R Crook, D Ince, L Lin, B Nuseibeh

7. Battle-scarred operations personnel carefully monitor fielded systems during use for security attacks. Attacks do occur, regardless of the strength of design and implementation, so monitoring software behavior is an essential defensive technique. Knowledge gained by understanding attacks and exploits should be cycled back into software development.

Taxonomy articles:
Use of A Taxonomy of Security Faults, T Aslam, I Krsul, E Spafford
Seven pernicious kingdoms: a taxonomy of software security errors, K Tsipenyuk, B Chess, G McGraw

Literatures
This book provide an analysis of the major problems with all software, and give a collection of techniques with which to address the recurring problems, such as buffer overflows, access control exposures, randomness flaws and other security-related defects.

Secure Programming with Static Analysis, by Brian Chess, Jacob West
This book shows the reader how to effectively use static analysis tools as a part of the code review process to automate finding security bugs. Because most programs are to large for manual line by line analysis this book presents a good automated solution.

This book is about the process that designs and implements strong programs. It starts with architecture and design documents, then follows through to design and maintenance.

This book emphasizes the differences between bugs (coding errors) and flaws (deeper architectural problems). It shows that automated code inspection tools can be applied more or less successfully to the first problem set, but human investigation is required to address the second. The book clarifies the need for an entire development process and not a bolt on solution.

Conferences
These are three conferences that have the highest prestige and are still relevant in the lightweight secure process field. I calculated prestige based on several different criteria. The number of publications and citations of the articles published at the conferences, but also abstract measurements.

CSF : IEEE Computer Security Foundations
Many new techniques and methods within security have been presented at Security Foundations Symposium and therefore there publications often become seed papers that many refer to.  
**IEEE Computer Society's Technical Committee on Security and Privacy**  
In this conference real implementations and actual projects results are often published and therefore have more contact with industry.  
**European Systems & Software Process Improvement and Innovation**  
While the two first conferences focus on security, my research enters the realm of software process, but there are no security software process conferences. This conference focuses heavily on case studies, the same as my research.

### Groups and organizations

**Center for Internet Security** - CIS members develop and encourage the widespread use of security configuration benchmarks through a global consensus process involving participants from the public and private sectors. - [http://www.cisecurity.org/](http://www.cisecurity.org/)

**Computer Emergency Response Team** - CERT has developed a methodology to help organizations build security into the early stages of the production life cycle. The Security Quality Requirements Engineering (SQUARE) methodology consists of nine steps that generate a final deliverable of categorized and prioritized security requirements. Although the SQUARE methodology could likely be generalized to any large-scale design project, it was designed for use with information technology systems. - [http://www.cert.org/cert/](http://www.cert.org/cert/)


### Important researchers

G McGraw – Produced numerous article and books about secure development processes. Lead researcher in the field and the seed writer of lightweight security, touchpoints.

D Viega – Code security researcher that has done much collaborator work with G Mcgraw.

D Evans, D Larochelle – Several seed papers in 2000 – 2002 that are today standards in automated vulnerability research. An, according to me, important part in development of secure software.

J Jürjens – Research and seed paper on designing security with the aid of UML.

B Schneier – Lead developer in cryptology but also seed paper on attack trees and security design.

### My research

While my research is dependent on development process as the SDL or G McGraw’s books, my focus is not to add new methods and tools to the process. Instead I’m investigating the process claim that they can be integrated into existing development cycles, to be able to do that I have to know if any improvement has been done to the software after a security method has been added. This presents the biggest problem I have right now and it is not currently addressed by any of the articles or books that have been mentioned, how to measure the improvement.