Secure Agile

How to make secure applications using Agile Methods

Thomas Stiehm, CTO

tom.stiehm@coveros.com

About Coveros

- Coveros helps organizations accelerate the delivery of business value through secure, reliable software

Symantec

UnitedHealth Group

IFC International Finance Corporation World Bank Group

U.S.ARMY

AKO ARMY KNOWLEDGE ONLINE

Schlumberger

RSA The Security Division of EMC

UNOS DONATE LIFE

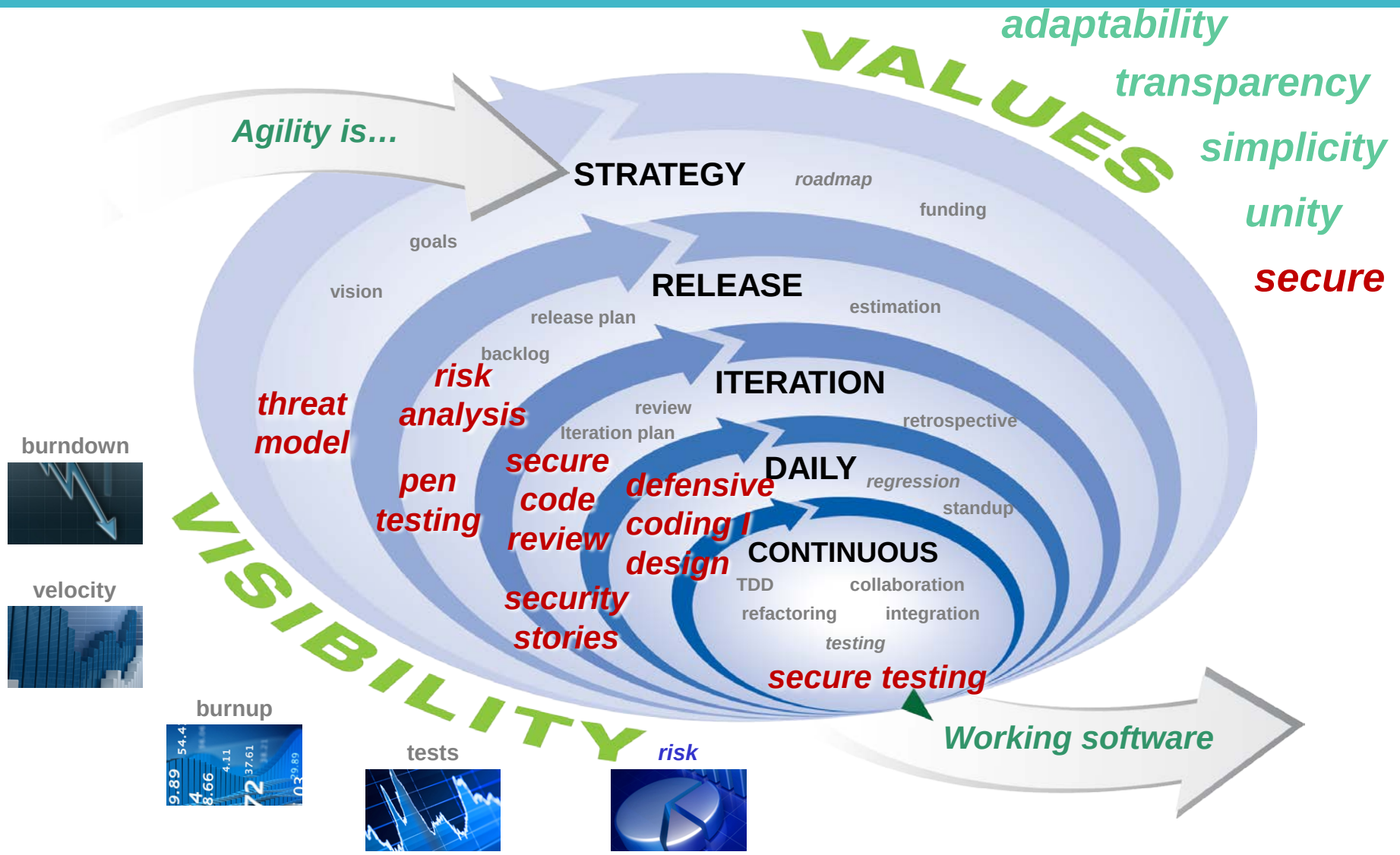
EMPLOYERS

Forge.mil

BNP PARIBAS

FannieMae

SecureAgile™ Development Process



Assures time-to-market while achieving security objectives

SecureAgile™ Security Practices

- Threat Modeling
- Risk Analysis
- Pen Testing
- Security Stories
- Secure Code Review
- Defensive Coding and Design
- Secure Testing
  - Static Code Analysis
  - Automated Security Testing

Threat Modeling

- Threat modeling is the process of defining a system's attack surface to support application risk assessments and to determine appropriate security controls. This includes assets that may be compromised and vulnerabilities that can be used to attack the system.

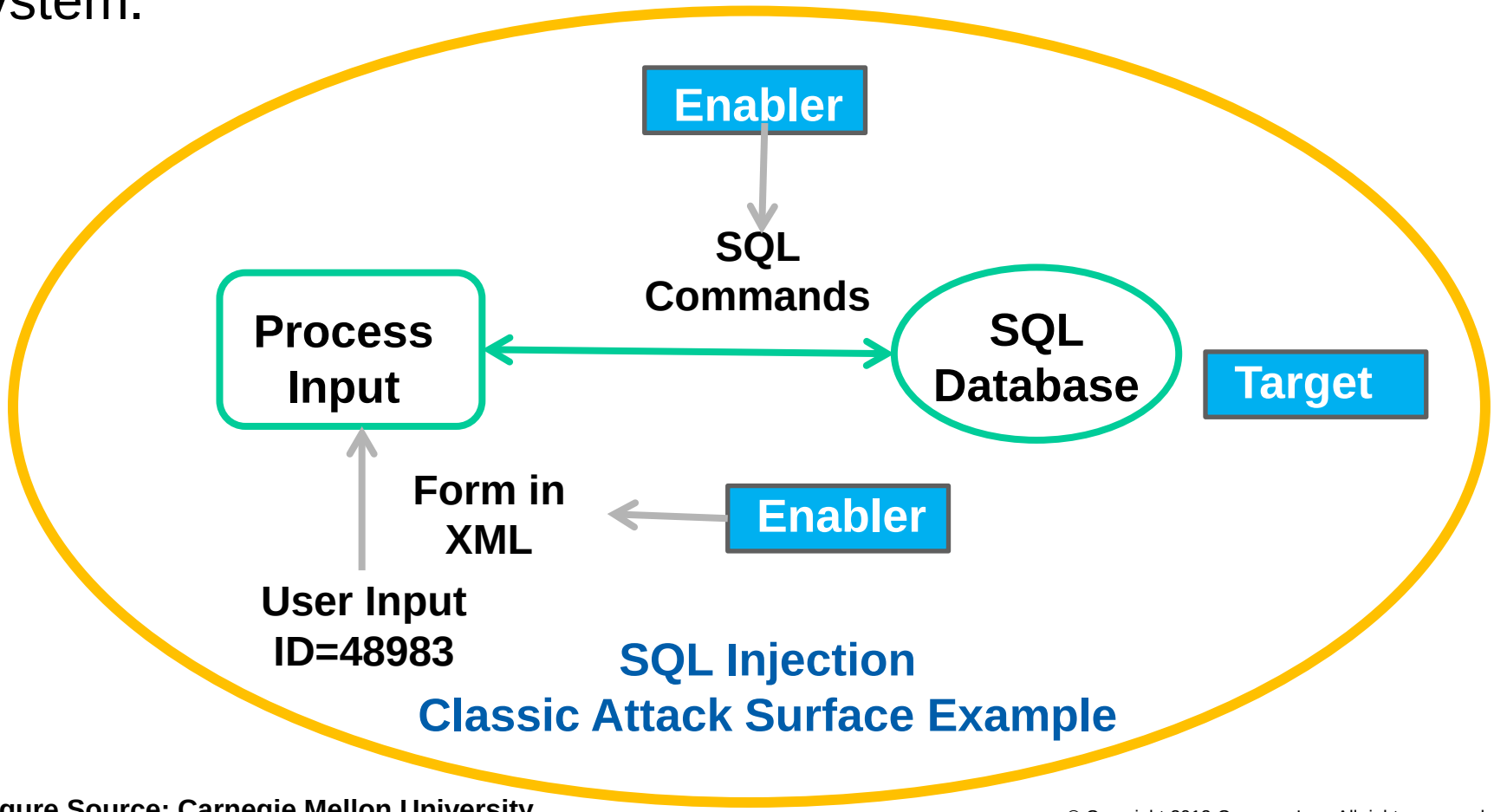


Figure Source: Carnegie Mellon University

Risk Analysis

- Identify areas of risk in the system, including:
  - Requirements
  - Design
  - Architecture
- Use abuse cases to drive risk based testing
- Build scenarios based on identified risks
- Use risk scenarios to drive security requirements
- Test risk conditions explicitly

Pen Testing

- Penetration Testing or Pen Testing, is the process of attacking a system like a malicious outsider in order to evaluate the security of the system
- Perform penetration testing for risks uncovered throughout the lifecycle
- Penetration testing is not a substitute for automated secure code review

Security Stories

- Why write Security Stories?
  - To make sure all explicit security requirements, both functional and non-functional, are documented and can be used to guide secure development and testing activities
- Develop misuse and abuse cases that capture non-normative behavior (attacks) according to your threat model
- Think like a potential attacker and use your knowledge of the system architecture and risks
- Drive test plans from the abuse cases
- Also write functional security stories

Misuse / Abuse Case Development

- Purpose: Define the possible mechanisms an adversary might exploit to compromise your system

- Approach:
  - "User shall not …" pattern
  - Misuse cases are extensions to stories that highlight ways in which the system might be misused accidentally
  - Abuse cases are extensions to stories that highlight ways in which the system might be abused on purpose

- Results:
  - Insight into potential abuses that can be avoided and tested

Secure Code Review

- Start with automated secure code review tools to find known issues and pinpoint areas in the code to review manually
- Review sections of the code manually, focus on areas that the automated tools found to contain a lot of issues, bugs cluster
- Real-time secure code review can be done as part of pair programming
- Train developers how to do secure code reviews
- Automated security analyzers should be run as part of a continuous integration process to identify known coding weaknesses during all builds

Defensive Design

- Software is designed to be secure through:
  - Identification and integration of security controls based upon the threat model
  - Use of security protection mechanisms for software startup, reboot, and shutdown procedures
  - Appropriate and comprehensive error and exception handling of all critical functions
  - Use of code libraries that have been vetted for security
  - Use of off-the-shelf components for encryption, random number generation, and other complex mathematical calculations

Defensive Coding

- Secure coding is done through:
  - Avoiding known dangerous coding constructs, system calls and programming short cuts
  - Continued security scans of new code at each check-in
  - Proper integration and testing of secure design features

Secure Testing

- There are a variety of testing types that must be performed during agile development iterations to assure application security
  - Functional security testing – testing the capabilities and integration of security controls into the application
  - Non-functional security testing – testing against the misuse and abuse cases developed during story creation
  - Risk-based testing – testing the application against the identified threats within the threat model
- Automation is required for continuous security testing
- Leverage security testing tools, either Open Source or Commercial tools

Implementing Agile Application Security

- Adopt and use an application security process from the beginning of the project
- Create application security requirements with the functional application requirements
- Lead the security requirements process, sell the value of good security practices to the business
- Development teams need software security training, early
- Security practices needs to be burned-in and made part of how the team works
- Security work should be done by experienced, technically strong developers
- Create application security standards and practices, monitor compliance with the standards
- Put security controls into your base software architecture

Implementing Agile Application Security

- Use security tools such as static code analysis and web scanners to verify security controls
- Conducting manual security verification like code reviews and penetration testing
- Use outside security testers to break the system and look for holes
- Problems found in security testing need to be added the team's backlog
- Security tests don't always fit in time boxes so, if needed, run them as parallel engagements
- Consider a "hardening sprint" to focus on fixing the security problems found through security testing

Thank You

Supplemental Material

Vulnerabilities

- OWASP Top Ten:
  - https://www.owasp.org/index.php/Top_10_2010

- **2011 CWE/SANS Top 25 Most Dangerous Software Errors**
  - http://cwe.mitre.org/top25/

- There is a lot of overlap as there are major categories that generate a lot of vulnerabilities
- For Example:
  - Injection Attacks and
  - Misconfigurations