# Abstract

Software development teams are increasingly faced with security concerns regarding the software they develop. While many software development security practices have been proposed, published empirical evidence for their suitability and effectiveness is limited. The goal of this research is to support theory construction through empirical evidence collection for security practice use in software development by building a measurement framework for software development security practices and their correlations with security-related outcomes.

To orient the framework, we set two practice-oriented sub-goals:

Reduce post-release vulnerabilities for a project through security practice selection and use.

Identify security practices most likely to reduce post-release vulnerabilities from the pool of practices not currently applied on a project.

To meet our goals, we define and evaluate the “Security Practices Evaluation Framework” (SP-EF).

This document describes how to collect the data required for SP-EF. Examining patterns in the aggregated data supports making security-focused improvements to the development process.

# Introduction

Vulnerability prevention and removal are becoming increasingly important in software development. A number of security practices have been developed, advocated and implemented, but there is not yet empirical evidence for which practices work best in a given development environment.

In this document, we describe the data elements of the Security Practices Evaluation Framework (SP-EF), and provide guidance on how to collect the data elements. SP-EF contains three categories of data elements; practice adherence metrics, outcome measures, and context factors. The practice adherence metrics are a set of attributes and values that are used to describe security practices in use on a project, and the degree to which each practice is adhered to by the project team. Outcome measures are a set of attributes and values that are used to describe the security-related outcomes of the project. Context factors are a set of attributes and values that are used to provide a basis of comparison between projects measured using SP-EF.

*The goal of the SP-EF is to provide a repeatable set of measures and measurement instructions structuring case studies of security practice use so that the case studies can be combined, compared, and analyzed to form a family of evidence on security practice use.[[1]](#footnote-1)*

We have adopted the design of the Extreme Programming Evaluation Framework (XP-EF) [1], where possible. Goals for the framework include that the metrics be:

* Simple enough for a small team to measure without a metrics specialist and with minimal burden
* Concrete and unambiguous
* Comprehensive and complete enough to cover vital factors

The framework is designed for use throughout development, as well as for annotation of projects that have been completed.

The primary data sources required are the project’s documentation, particularly all process-related documentation, the version control system, and the bug tracker.

We collected practice and artifact use by searching for references to keywords in both the bug repository and commit logs for each practice and artifact.

# Project Demographics

To identify the collected data, record the following items:

* Organization name
* Project name
* Date(s) measurements were taken
* Start date of measurement period
* End date of measurement period
* Version control system
* Bug tracking system

# Practice Adherence Metrics

SP-EF’s practice adherence metrics are designed to measure how closely a project adheres to a set of security practices. For each practice, a set of four metrics, Usage frequency, Ease of use, Assistance, and Training, measures the practice’s usage. Where practices depend on information sources, ‘artifacts’, we also collect artifact metrics measuring the usage of each artifact identified. We collect data on both practice and artifact adherence to distinguish, for example, between thorough application of a practice based on an ad hoc collection of notes, and casual application of a practice based on a thorough, formal specification.

A set of metrics for 28 security practices and 15 artifacts represent SP-EF’s notion of practice adherence. The metrics characterize adherence for each practice and artifact. The practices have been abstracted from lists of software development security practices [BSIMM, OWASP, SafeCode, MS SDL]. The artifacts are security-related information sources that are referenced or created in the course of applying the practices.

At each application of SP-EF, for each of the 28 SP-EF security practices, we collect the data elements listed in the following table:

|  |  |  |  |
| --- | --- | --- | --- |
| Context Factor | Definition | Data Collection Rule | Citation |
| Practice Name | Security Practice | Choose from SP-EF list of Security Practices | SP-EF |
| Role | Role of person applying the practice. | Select from list of SP-EF project roles, enumerated in the Role Name Context Factor. | SP-EF |
| Phase | Project phase during which the practice is performed. | Select from list of project phases/activities: Initiation, Requirements, Design, Implementation, Unit Testing, Testing, Release, Operations | ODC |
| Usage Frequency | Periodicity of the use of this practice by project staff | Select from list: Not Used, Daily, Weekly, Monthly, Quarterly, Annually, Less than Annually,  Per Commit, Per Build, Per Release |  |
| Ease of Use | Project staff perception of ease with which this practice is performed | Select from list: Very Low, Low, Nominal, High, Very High. | COCOMO |
| Assistance | Project staff perception of assistance this practice provides for security. | Select from list: Very Low, Low, Nominal, High, Very High | UTAUT |
| Training | Project staff level of training in this practice | Select from list: Very Low, Low, Nominal, High, Very High |  |
| Artifacts Used | List of artifacts used in the course of applying this practice | Choose from SP-EF list of Artifacts |  |
| Practice Application Date | Date of practice use in the project |  |  |
| Evidence Source | Source of information given in this observation. | If a source for the data collected is available, provide link or cite in this field |  |
| Comments | Any additional information believed to be useful by the researcher and/or project staff. | Free-form text, at observer’s discretion |  |

For <Project Name>, as observed <Observation Date> [to <Most Recent Date>], we collected the following data:

### P1 Examine and document likely threats to the organization and application type

Description: Effort to discover and document potential threats to the organization and its systems, and documentation of the threats identified.

Data: (Example data from Firefox 3.6 series of releases)

* Role(s): Management
* Phase(s): Requirements, Design
* Usage Frequency:
* Ease Of Use: Nominal
* Assistance: Nominal
* Training: Nominal
* Artifacts Affected: Assets, Threats.
* Artifacts Referenced: Assets, Threats, Bug Reports, News, Attacks
* Practice Application Date: 2/1/2007
* Evidence Source: <https://wiki.mozilla.org/Firefox3/Firefox_Requirements_Meetings/Security_and_Privacy>
* Comments:

#### P2 - Establish Data Classification Scheme

Description: Description of types of data and their importance to the organization, e.g. Personally-Identifiable Information (PII), financial information, confidentiality requirements.

Data:

* Role(s):
* Phase(s):
* Usage Frequency:
* Ease Of Use:
* Assistance:
* Training:
* Artifacts Affected:
* Artifacts Referenced:
* Practice Application Date:
* Evidence Source:
* Comments:

#### P3 - Specify Security Requirements

Description: Description of expectations of the software’s handling of data with respect to confidentiality, availability, integrity, repudiation, authentication, and authorization.

Data:

* Role(s):
* Phase(s):
* Usage Frequency:
* Ease Of Use:
* Assistance:
* Training:
* Artifacts Affected:
* Artifacts Referenced:
* Practice Application Date:
* Evidence Source:
* Comments:

#### P4 - Create Technical Stack

Description: Description of all technical components (e.g. Operating systems, languages, libraries, tools, services, etc.) required, allowed, and/or banned in the production and use of the software being developed.

Data:

* Role(s):
* Phase(s):
* Usage Frequency:
* Ease Of Use:
* Assistance:
* Training:
* Artifacts Affected:
* Artifacts Referenced:
* Practice Application Date:
* Evidence Source:
* Comments:

#### P5 - Ensure Technical Stack is Current

Description: Tracking of patches and new releases of all components within the technical stack, and the application of patches and new releases to the technical stack.

Data:

* Role(s):
* Phase(s):
* Usage Frequency:
* Ease Of Use:
* Assistance:
* Training:
* Artifacts Affected:
* Artifacts Referenced:
* Practice Application Date:
* Evidence Source:
* Comments:

#### P6 - Build Threat Model

Description: Analyze the software’s dataflow for security vulnerabilities, and document potential threats, and the goals and means of attackers.

Data:

* Role(s):
* Phase(s):
* Usage Frequency:
* Ease Of Use:
* Assistance:
* Training:
* Artifacts Affected:
* Artifacts Referenced:
* Practice Application Date:
* Evidence Source:
* Comments:

#### P7 - Address Security Requirements

Description: Consider existing Security Requirements and apply them while implementing the software.

Data:

* Role(s):
* Phase(s):
* Usage Frequency:
* Ease Of Use:
* Assistance:
* Training:
* Artifacts Affected:
* Artifacts Referenced:
* Practice Application Date:
* Evidence Source:
* Comments:

#### P8 - Apply Data Classification Scheme

Description: Consider the Data Classification Scheme and apply them while implementing the software.

Data:

* Role(s):
* Phase(s):
* Usage Frequency:
* Ease Of Use:
* Assistance:
* Training:
* Artifacts Affected:
* Artifacts Referenced:
* Practice Application Date:
* Evidence Source:
* Comments:

#### P9 - Apply Secure Coding Standard

Description: Description of security rules applying to each technical component used in the software.

Data:

* Role(s):
* Phase(s):
* Usage Frequency:
* Ease Of Use:
* Assistance:
* Training:
* Artifacts Affected:
* Artifacts Referenced:
* Practice Application Date:
* Evidence Source:
* Comments:

#### P10 - Apply Automated Code Analysis Tools

Description:

Data:

* Role(s):
* Phase(s):
* Usage Frequency:
* Ease Of Use:
* Assistance:
* Training:
* Artifacts Affected:
* Artifacts Referenced:
* Practice Application Date:
* Evidence Source:
* Comments:

#### P11 - Leverage coverage analysis

Description:

Data:

* Role(s):
* Phase(s):
* Usage Frequency:
* Ease Of Use:
* Assistance:
* Training:
* Artifacts Affected:
* Artifacts Referenced:
* Practice Application Date:
* Evidence Source:
* Comments:

#### P12 - Validate input and output against common vulnerabilities

Description:

Data:

* Role(s):
* Phase(s):
* Usage Frequency:
* Ease Of Use:
* Assistance:
* Training:
* Artifacts Affected:
* Artifacts Referenced:
* Practice Application Date:
* Evidence Source:
* Comments:

P13 - Drive tests with security requirements and security features

Description:

Data:

* Role(s):
* Phase(s):
* Usage Frequency:
* Ease Of Use:
* Assistance:
* Training:
* Artifacts Affected:
* Artifacts Referenced:
* Practice Application Date:
* Evidence Source:
* Comments:

#### P14 - Use threat model when creating test plan

Description:

Data:

* Role(s):
* Phase(s):
* Usage Frequency:
* Ease Of Use:
* Assistance:
* Training:
* Artifacts Affected:
* Artifacts Referenced:
* Practice Application Date:
* Evidence Source:
* Comments:

#### P15 - Ensure edge/boundary testing

Description:

Data:

* Role(s):
* Phase(s):
* Usage Frequency:
* Ease Of Use:
* Assistance:
* Training:
* Artifacts Affected:
* Artifacts Referenced:
* Practice Application Date:
* Evidence Source:
* Comments:

#### P16 - Fuzz test

Description:

Data:

* Role(s):
* Phase(s):
* Usage Frequency:
* Ease Of Use:
* Assistance:
* Training:
* Artifacts Affected:
* Artifacts Referenced:
* Practice Application Date:
* Evidence Source:
* Comments:

#### P17 - Use appropriate testing tools

Description:

Data:

* Role(s):
* Phase(s):
* Usage Frequency:
* Ease Of Use:
* Assistance:
* Training:
* Artifacts Affected:
* Artifacts Referenced:
* Practice Application Date:
* Evidence Source:
* Comments:

#### P18 - Create and use automation to do what attackers will do

Description:

Data:

* Role(s):
* Phase(s):
* Usage Frequency:
* Ease Of Use:
* Assistance:
* Training:
* Artifacts Affected:
* Artifacts Referenced:
* Practice Application Date:
* Evidence Source:
* Comments:

#### P19 - Provide guidance on handling security-related error messages

Description:

Data:

* Role(s):
* Phase(s):
* Usage Frequency:
* Ease Of Use:
* Assistance:
* Training:
* Artifacts Affected:
* Artifacts Referenced:
* Practice Application Date:
* Evidence Source:
* Comments:

#### P20 - Use penetration testing tools internally

Description:

Data:

* Role(s):
* Phase(s):
* Usage Frequency:
* Ease Of Use:
* Assistance:
* Training:
* Artifacts Affected:
* Artifacts Referenced:
* Practice Application Date:
* Evidence Source:
* Comments:

#### P21 - Review test results and correct, or formally accept the risks of releasing with failed checks

Description:

Data:

* Role(s):
* Phase(s):
* Usage Frequency:
* Ease Of Use:
* Assistance:
* Training:
* Artifacts Affected:
* Artifacts Referenced:
* Practice Application Date:
* Evidence Source:
* Comments:

#### P22 - Provide all available information to Penetration Testers

Description:

Data:

* Role(s):
* Phase(s):
* Usage Frequency:
* Ease Of Use:
* Assistance:
* Training:
* Artifacts Affected:
* Artifacts Referenced:
* Practice Application Date:
* Evidence Source:
* Comments:

#### P23 - Use external penetration testers to find problems

Description:

Data:

* Role(s):
* Phase(s):
* Usage Frequency:
* Ease Of Use:
* Assistance:
* Training:
* Artifacts Affected:
* Artifacts Referenced:
* Practice Application Date:
* Evidence Source:
* Comments:

#### P24 - Record important security-specific knowledge affecting deployed app security

Description:

Data:

* Role(s):
* Phase(s):
* Usage Frequency:
* Ease Of Use:
* Assistance:
* Training:
* Artifacts Affected:
* Artifacts Referenced:
* Practice Application Date:
* Evidence Source:
* Comments:

#### P25 - Fix all bugs found in operations

Description:

Data:

* Role(s):
* Phase(s):
* Usage Frequency:
* Ease Of Use:
* Assistance:
* Training:
* Artifacts Affected:
* Artifacts Referenced:
* Practice Application Date:
* Evidence Source:
* Comments:

#### P26 - Create a Top N Bugs List

Description:

Data:

* Role(s):
* Phase(s):
* Usage Frequency:
* Ease Of Use:
* Assistance:
* Training:
* Artifacts Affected:
* Artifacts Referenced:
* Practice Application Date:
* Evidence Source:
* Comments:

#### P27 - Enhance SSDL to prevent bugs found in operations

Description:

Data:

* Role(s):
* Phase(s):
* Usage Frequency:
* Ease Of Use:
* Assistance:
* Training:
* Artifacts Affected:
* Artifacts Referenced:
* Practice Application Date:
* Evidence Source:
* Comments:

#### P28 – Provide Training

Description:

Data:

* Role(s):
* Phase(s):
* Usage Frequency:
* Ease Of Use:
* Assistance:
* Training:
* Artifacts Affected:
* Artifacts Referenced:
* Practice Application Date:
* Evidence Source:
* Comments:

## Artifact Adherence Metrics

For each artifact referenced in the practices used, we record the attributes listed in the table below:

|  |  |  |  |
| --- | --- | --- | --- |
| Data Element | Definition | Counting Rule | Citation |
| Artifact Name | Artifact name | Choose from SP-EF list of Artifacts |  |
| Artifact Channel | Means, ‘channel’, by which the artifact is communicated. | Select from list: Whiteboard, Email, Online (e.g. forum, website), Paper |  |
| Artifact Source | Organization which generated the artifact | Select from list: project, organization, external |  |
| Artifact Type | Level of rigor and effort invested in the artifact | Select from list: Absent, Ad hoc, Guideline/Standard, Structured Notation, Formal Model |  |
| Artifact Assurance | Means by which the use of the artifact is checked in the course of the project | Select from list: Absent, Ad hoc, Review, Test(s), Tool |  |
| Artifact Revision Frequency | Frequency with which the artifact is updated. | Select from list: Not Applicable, Daily, Weekly, Monthly, Quarterly, Annually, Less than Annually |  |
| Artifact Example | Example of the artifact | If possible, provide a citation, link, or other description of the artifact. |  |
| Earliest Artifact Application Date | Earliest observable date of artifact use in the project |  |  |
| Evidence Source | Source of information given in this observation. | If a source for the data collected is available, provide link or cite in this field |  |
| Comments | Any additional information believed to be useful by the researcher and/or project staff. | Free-form text, at observer’s discretion |  |

### A1 – Security Principles

Description: The Security Principles artifact represents the project’s notion of security principles to be applied in the course of development.

Data:

* Channel:
* Source:
* Type:
* Assurance:
* Revision Frequency:
* Example:
* Earliest Artifact Application Date:
* Evidence Source:
* Comments:

### A1 - Assets

Description: The Assets artifact represents the project’s (and/or the organization’s) notion of what should be protected.

### A2 - Bug Reports

Description: The Bug Reports artifact represents the project’s notion of how issues with the software are tracked and reported.

### A3 - Data Classification Scheme

Description: The Data Classification Scheme artifact represents the project’s notion of how data managed by the software is classified.

### A4 - Secure Coding Standard

Description: The Secure Coding Standard artifact represents the project’s notion of security-specific rules for the software’s source code.

A5 - Secure Software Development Lifecycle (SSDL**)**

Description: The SSDL artifact represents the project’s notion of its own methodology (ground rules) for secure software development.

### A6 - Security Features

Description: The Security Features artifact represents the project’s notion of availability of and documentation for platform and software features that support security.

### A7 - Security Principles

Description: The Security Principles artifact represents the project’s notion of security principles to be applied in the course of development.

### A8 - Security Requirements

Description: The Security Requirements artifact represents the project’s notion of security-specific requirements to be implemented.

### A9 - Technical Stack

Description: The Technical Stack artifact represents the project’s notion of tool and platform requirements.

### A10 - Test Plan

Description: The Test Plan artifact represents the project’s notion of how the software is to be tested.

### A11 - Test Results

Description: The Test Results artifact represents the project’s notion of results from testing the software.

### A12 - Test Results

Description: The Test Results artifact represents the project’s notion of results from testing the software.

### A13 - Threat Model

Description: The Threat Model artifact represents the project’s notion of the software’s expected attackers and their means of attack, and software weaknesses and risks.

### A14 - Threats

Description: The Threats artifact represents the project’s notion of risks to the project Assets.

### A15 - Top N Bugs List

Description: The Top N Bugs List artifact represents the project’s notion of the most common security bugs affecting the project’s platform, language, and source code.

# Outcome Measures

Counting, and recording, defects, and vulnerabilities.

|  |  |  |  |
| --- | --- | --- | --- |
| Context Factor | Definition | Data Collection Rule | Citation |
| Pre-release Defects | Defects found in new and changed code before software is released |  |  |
| Post-release Defects | Defects found in new and changed code after software is released |  |  |
| Pre-release Vulnerabilities | Vulnerabilities found in new and changed code before software is released |  |  |
| Post-release Vulnerabilities | Vulnerabilities found in new and changed code after software is released |  |  |
|  |  |  |  |
|  |  |  |  |

# Context Factors

Drawing general conclusions from empirical studies in software engineering is difficult because the results of any process largely depend upon the specifics of the study and relevant context factors. We cannot assume a priori that a study’s results generalize beyond the specific environment in which it was conducted [3]. Therefore, recording an experiment’s context factors is essential for comparison purposes and for fully understanding the similarities and differences between the case study and one’s own environment.

## Team Metrics

SP-EF identifies four roles involved in applying security practices during software development. Management sets standards, generates requirements, monitors processes. Development writes code and related artifacts to produce a software release. Testing writes test plans and related artifacts to assure a software release is ready. Operations configures and monitors software releases in the context of software systems.

|  |  |  |  |
| --- | --- | --- | --- |
| Context Factor | Definition | Counting Rule | Citation |
| Role Name | Role of person conducting activity on behalf of the project | Identify individuals engaged full-time or part-time in the role’s activities. The roles defined for SP-EF are: Management (Project Management, Requirements Engineer, Documentation, Build Administrator, Security), Development (Designer, Developer), Testing (Quality Assurance, Penetration Tester, External Penetration Tester), and Operations (User, Systems Administrator, Database Administrator), Other |  |
| Role Size | Number of people engaged in this role on the project during the time period measured | Count individuals engaged full-time or part-time in the role’s activities. |  |
| Role Experience | Number of years of experience by team members performing this role for the project Alternative: use COCOMO-style ‘Nominal’ Likert scale | Ask individuals in role how much experience they have with the role, as measured in years, and sum the collected numbers. Fractions (1/2), decimal (.5) parts of years are acceptable. | Jones, Kan, Boehm (COCOMO II) |
| Role Capability | Evaluation of role’s capability for this project | Select from list: Very Low (15th percentile), Low (35th), Nominal (55th), High (75th), Very High (90th) |  |
| Team Size | Number of distinct people across all roles on the project during the time period measured | Count individuals engaged full-time or part-time on the project. On single person projects, Team Size and Role Size are both 1, while on larger projects, Team Size reflects the total number of people while Role Size reflects the distribution of effort. |  |
|  |  |  |  |

## Project Metrics

|  |  |  |  |
| --- | --- | --- | --- |
| Context Factor | Definition | Counting Rule | Citation |
| Software Release | Source Code Metrics, Test Metrics for software produced during the time period measured |  |  |
| Management Role | Team Metrics data for Management role |  |  |
| Development Role | Team Metrics data for Development role |  |  |
| Testing Role | Team Metrics data for Testing role |  |  |
| Development Methodology | Project approach to the software development lifecycle | Text description from project staff (e.g., XP, Scrum, Waterfall, Spiral, …) |  |
| Person Months | Amount of effort spent on project | Total person-months spent on project, summing all individual efforts |  |

## Software Release Metrics

|  |  |  |  |
| --- | --- | --- | --- |
| Context Factor | Definition | Counting Rule | Citation |
| Software Release | Source Code Metrics, Test Metrics for software produced during the time period measured |  |  |
| Management Role | Team Metrics data for Management role |  |  |
| Development Role | Team Metrics data for Development role |  |  |
| Testing Role | Team Metrics data for Testing role |  |  |
| Development Methodology | Project approach to the software development lifecycle | Text description from project staff (e.g., XP, Scrum, Waterfall, Spiral, …) |  |
| Person Months | Amount of effort spent on project | Total person-months spent on project, summing all individual efforts |  |
| Source Lines of Code (SLOC) | Number of non-blank, non-comment lines present in the release of the software being working on during the current project. | Use cloc where possible |  |
| Churn | Number of non-blank, non-comment lines changed, added, or deleted in the release of the software being working on during the current project. | Use cloc where possible, applied to the previous version of the software and to the current version, and taking the difference. |  |
| Components | List of components used by project but not written by project (e.g. libraries, frameworks from organization, third-parties, open-source) | Work with project team and/or source code to determine dependencies on components. Record the vendor/source/OSS project name, license, and whether the source is available for each component. |  |

## Software System Metrics

|  |  |  |  |
| --- | --- | --- | --- |
| Context Factor | Definition | Counting Rule | Citation |
| Domain |  |  |  |
| Age | Number of years software since software was first written |  |  |
| Identities Managed | Average number of personal identities managed by system |  |  |
| Confidentiality Requirement | Importance of confidentiality to the system’s operation | Select from Low, Medium, High, Not Defined | CVSS (https://www.first.org/cvss/cvss-guide#i2.3.3) |
| Integrity Requirement | Importance of integrity to the system’s operation | Select from Low, Medium, High, Not Defined | CVSS |
| Availability Requirement | Importance of availability to the system’s operation | Select from Low, Medium, High, Not Defined | CVSS |
| Installations | Number of installations of the system |  |  |
| Users | Number of users that access system |  |  |
| Usage profile | Conditions under which the system runs | Select from: default service, optional service, default executable, optional executable |  |

## Source Code Metrics

|  |  |  |  |
| --- | --- | --- | --- |
| Context Factor | Definition | Counting Rule | Citation |
| Source Lines of Code (SLOC) | Number of non-blank, non-comment lines present in the release of the software being working on during the current project. | Use cloc where possible | http://cloc.sourceforge.net/ |
| Churn | Number of non-blank, non-comment lines changed, added, or deleted in the release of the software being working on during the current project. | Use cloc where possible, applied to the previous version of the software and to the current version, and taking the difference. |  |
| Language | Computer language(s) in which the software release is written in. | Record the language(s) being used by the team during the release (available from cloc). |  |
| Platform | Software environment on which the software release/system is run when it is released. | Record the platform(s) on which the released software runs (e.g. Linux, Windows, iOS, Android, …). |  |

## Test Metrics

|  |  |  |  |
| --- | --- | --- | --- |
| Context Factor | Definition | Counting Rule | Citation |
|  | Test Cases Total |  |  |
|  | Security Test Cases Total |  |  |

|  |  |  |
| --- | --- | --- |
| Metric | Start | End |
| Organization Name | Mozilla | Mozilla |
| Project Name | Firefox | Firefox |
| Measurement Date | 10/30/2009 | 3/13/2012 |
| Core Development Team Size |  |  |
| Total Development Team Size |  |  |
| Development Team Capability |  |  |
| Core Testing Team Size |  |  |
| Total Testing Team Size |  |  |
| Testing Team Capability |  |  |
| Domain |  |  |
| Age |  |  |
| Confidentiality Requirement |  |  |
| Integrity Requirement |  |  |
| Availability Requirement |  |  |
| Installs |  |  |
| Machines |  |  |
| Identities per Install |  |  |
| Source Lines of Code (SLOC) |  |  |
| Churn |  |  |
| Language |  |  |
| Platform |  |  |
| Pre-release Defects |  |  |
| Pre-release Vulnerabilities |  |  |
| Post-release Defects |  |  |
| Post-release Vulnerabilities |  |  |

# Scratchpad for full measurement definitions

## Demographics

Demographic data must be recorded for each data element or set of data elements collected, including Date data recorded, who recorded, whether estimation or aggregation were used, confidence in #.

## Organization Name

Security practices, personnel policies, media and public attention, and many other factors will vary from organization to organization. We record the organization name to permit controlling for the organization.

## Project Name

Development platforms, schedules, staffing Security practices, personnel policies, and many other factors will vary from project to project. We record the project name to permit controlling for the project.

## Confidentiality, Integrity, and Availability Requirements

These values are taken directly from CVSS, and this section paraphrases the description in the CVSS Guide [ref]. These metrics measure the security requirements of the software under development. Each security requirement has three possible values: “low”, “medium”, and “high”.

To choose a value for each context factor, consider the most sensitive data that passes through, or is kept by, the software being evaluated. For example, a web browser may access highly confidential personal information such as bank account or medical record data, to which a “High” Confidentiality Requirement would apply.

|  |  |
| --- | --- |
| Metric Value | Description |
| Low (L) | Loss of [confidentiality | integrity | availability] is likely to have only a limited  adverse effect on the organization or individuals associated with the organization  (e.g., employees, customers). |
| Medium (M) | Loss of [confidentiality | integrity | availability] is likely to have a serious adverse  effect on the organization or individuals associated with the organization (e.g.,  employees, customers). |
| High (H) | Loss of [confidentiality | integrity | availability] is likely to have a catastrophic  adverse effect on the organization or individuals associated with the organization  (e.g., employees, customers). |
| Not Defined (ND) | Assigning this value to the metric will not influence the score. It is a signal to the  equation to skip this metric. |

# 1) Core Team Size: Team sizes are tracked for development and testing.

|  |  |
| --- | --- |
| Count | When working with a project in progress, count full time people currently engaged in development roles. Separately count full time people engaged in testing roles. When working with historical project data, sort participants by their number of commits (or bug reports) and count participants contributing the first 80% of commits (bug reports) to estimate development team size and testing team size. [4]PM. |
| Exclude | Do not count specialists who aid the team *part time* (such as performance, user interface design, management, project management, or localization specialists, technical writers). These will be recorded in another section. |
| Demographics | Date data recorded, who recorded, whether estimation or aggregation were used, confidence in # |

# 1) Total Team Size: Team sizes are tracked for development and testing.

|  |  |
| --- | --- |
| Count | When working with a project in progress, count all people currently or previously engaged in development roles. Separately count all people engaged or previously engaged in testing roles. When working with historical project data, count all participants with one or more commits (or bug reports). |
| Demographics | Date data recorded, who recorded, whether estimation or aggregation were used, confidence in # |

## Age

Product age relates to both the availability of product knowledge as well as product refinement. An older product might be considered more stable with fewer defects, but there may be a lack of personnel or technology to support the system. Furthermore, making significant changes to a legacy system may be an extremely complex and laborious task. Working with a newer product may involve instituting complex elements of architectural design that may influence subsequent development, and may be prone to more defects since the product has not received extensive field use.

|  |  |
| --- | --- |
| Count | Determine the date of the first commit/first lines of code written. Record the number of months elapsed since that date. The Age may be recorded and expressed in other units (days, years). |

## 

## Number of Machines

The rise of botnets, networks of computers that can be centrally directed, has created a black market for their services. In 2013, an hour of machine time on a botnet ranged from 2.5 – 12 US cents[[2]](#footnote-2). Infesting machines with malware enabling central control creates Botnets, and so the number of machines a piece of software runs on is a risk factor.

|  |  |
| --- | --- |
| Count | Count (estimate) the machines (physical or virtual) on which the software runs. |
| Demographics | Date data recorded, who recorded, whether estimation or aggregation were used, confidence in # |

## Number of Installs

Some software (e.g. database-backed web applications) may require more than one machine (e.g. web server, database server, either physical or virtual) for a single install. A vulnerability in any part of the installed software may enable access to all of the underlying machines, making mulit-machine installs a risk factor for security purposes.

|  |  |
| --- | --- |
| Count | Count (estimate) the number of installations of the software. |
| Demographics | Date data recorded, who recorded, whether estimation or aggregation were used, confidence in # |

## Number of Personal Identities Managed

Along with the market in botnets, a black market for personal identities, names, addresses, credit card numbers, bank account numbers, has developed. In 2011, a personal identity could be bought (in groups of 1000) for 16 US cents[[3]](#footnote-3). One component of software security risk is the presence and use of personal information, represented by the number of identities accessible to the software.

|  |  |
| --- | --- |
| Count | Count (estimate) the number of personal identities managed by the software. A browser might manage one or two identities, while a database system might manage millions. |
| Demographics | Date data recorded, who recorded, whether estimation or aggregation were used, confidence in # |

## Source Lines of Code (SLOC)

|  |  |
| --- | --- |
| Count |  |

## Churn

|  |  |
| --- | --- |
| Count | The total added, modified, and deleted lines of code of the software release during the date range evaluated. |

## Language

|  |  |
| --- | --- |
| Count |  |

## Platform

|  |  |
| --- | --- |
| Count |  |

Project

Summary: The SP-cf’s project-specific factors are designed to help quantify projects in terms of size, cost, and schedule. Gauging the size of the project can be done in many ways.

Effort will be focused in different project areas, such as quality assurance or requirement satisfaction, depending upon the nature of a software project. An enhancement project may focus on a specific piece of the product, thus narrowing the project’s scope and possibly simplifying the overall process. A maintenance project is concerned with correcting bugs, but must also prevent further defect injection while fixing the system. The presence of constraints significantly increases the amount of risk associated with a project. A fixed-delivery date may force delivery of a product before it has been thoroughly tested. Conversely, stringent reliability constraints may influence the amount of new functionality that can be introduced into the system.

Tracking the amount of effort spent on a project is difficult but is necessary for comparisons to be drawn about quality and productivity. Both person months and elapsed months are recorded because some people work part time on other projects, thus necessitating a time-independent metric. Additionally, both of these measures are used to calibrate productivity in the SP-om. Recording the amount of new and changed lines of code can help understand the effort expended on a project.

Person Months: This metric provides basic documentation of the amount of effort spent on the project. Both Person Months and Elapsed Months are included as context factors because some people work part time on other projects.

|  |  |
| --- | --- |
| Count | Do count the number of person months spent by each person on the project, including partial months and part-time work. |

Elapsed Months: This metric provides basic documentation of the overall schedule of the project in terms of calendar days.

|  |  |
| --- | --- |
| Count | Do count calendar days from when the initial requirements are selected for the release. Include design time, unit test time, and time for test phases until the product is considered final and ready for release. |
| Exclude | Do not count time to manufacture media after code is final (time for pressing CDs, for example). |

Nature of the Project: Effort will be focused in different areas depending upon the nature of a software project. An enhancement project may focus on a specific piece of the product, thus narrowing the project’s scope and possibly simplifying the overall process. A maintenance project is concerned with correcting bugs, but must also prevent further defect injection while fixing the system. We also include migration to mark a conversion to a new platform, which may involve understand a significant amount of new technology. Finally, we use reengineering to denote the reimplementation of a legacy system, for instance, converting a 16-bit system to a 32-bit application.

|  |  |
| --- | --- |
| Record | Whether the project is an enhancement of a previous release, a new product, a maintenance effort, a migration, or a reengineering effort. |

**Constraints:** The presence of constraints significantly increases the amount of risk associated with a project. This factor describes if the project is time boxed, or if the list of requirements is fixed and the date moves to accommodate the content. A fixed-delivery date may force a product out the door before it has been thoroughly tested. Conversely, severe reliability constraints may influence the amount of new functionality that can be introduced into the system. XP’s Planning Game practice states release dates are firm and features are added, adjusted, or removed the release date, so many classic XP projects would generally be “date constrained.”

|  |  |
| --- | --- |
| Count | Any constraints by which the project is bound, e.g. fixed delivery dates, fixed-price contracts, team size limitations, reliability constraints, etc. Examples would be Date Constrained, Scope Constrained, Resource Constrained and any combinations and degrees of these. |

Geographic factors

Summary: Distributed development has become more commonplace in industry and may influence XP practices such as pair programming, continuous integration, and collective ownership. Teams that communicate via the Internet may suffer increased communication and feedback times when compared to co-located teams

Table 6: Example geographic factors

|  |  |
| --- | --- |
| Team location | Collocated |

1) Team Location: Distributed teams that communicate via the Internet are becoming more commonplace, and it is possible that team location and accessibility may influence an XP project. A distributed team faces more challenges than a co-located team during development. Communication and feedback times are typically increased when the team is distributed over many sites.

|  |  |
| --- | --- |
| Record | Record whether the team is collocated or distributed. A collocated team is found in the same building and area, such that personal interaction is easily facilitated. If the team is distributed, record whether the distribution is across several buildings, cities, countries, or time zones. |

|  |  |  |
| --- | --- | --- |
| Metric Name | Start | End |
| Mozilla |  |  |
| Firefox | Firefox 3.6.x | Firefox 3.6.x |
|  | 10/30/09 | 3/13/12 |
| Core Team Size | 87 | 93 |
| Total Team Size | 680 | 740 |
| Team Capability | High | High |
| Core Team Size | 1320 | 1135 |
| Total Team Size | 10029 | 12229 |
| Team Capability | Nominal | Nominal |
| Domain |  |  |
| Age | 7.5 years | 10 years |
| Confidentiality Requirement | High | High |
| Integrity Requirement | High | High |
| Availability Requirement | Medium | Medium |
| Installs | 365000000 | 60000000 |
| Machines | 365000000 | 60000000 |
| Identities per install | 1-2.5 | 1-2.5 |
| SLOC | 4645135 | 6770243 |
| Churn | 0 | 2125108 |
| Language | C++/C/Javascript | C++/C/Javascript |
| Platform | OS, multiple | OS, multiple |
| Pre-release Defects |  |  |
| Pre-release Vulnerabilities |  |  |
| Post-release Defects | 0 | 22418 |
| Post-release Vulnerabilities | 0 | 317 |

# Appendix

## Security Practices List

In this section, we list the 28 SP-EF practices in their template form, annotated with the expected values for Roles and Artifacts.

We build our notion of practices and artifacts according to the following template:

[An Agent (Person, Role, Process) ]

As a <Role>

In order to achieve <SecurityGoal>,

Applies: <Practice>

To: some <Artifact > - from entity set, fill in blanks for specific one used in this instance (Where)

[guided by some <Artifact>]

[guided by some <SecurityPrinciple> ]

[producing some resulting <Artifact>]

Logging: <Results>

[Demographics: <When>, <Agent>, <Mode>,]

* Key:
  + [] – optional data elements
  + <Role> - ‘a defined function to be performed by a project team member, such as testing, filing, inspecting, coding*.’ [IEEE Glossary]*.
  + <SecurityGoals> - Security practices are intended to maintain security properties of the software being developed. Riaz et al. [2] analyzed ten descriptions of security objectives and identified six central objectives (properties): Confidentiality, Integrity, Availability, Identification & Authentication, Accountability, and Privacy (CIAIAP). In this iteration of SP-EF, we make the simplifying assumption that each practice addresses each security property equally, but we include the goal in the template as a place to start when refining this assumption.
  + <Practice> - ‘a specific type of professional or management activity that contributes to the execution of a process and that may employ one or more techniques and tools*.’* [IEEE Glossary]. The list of 27 SP-EF practices is below.
  + <Artifact> - In SP-EF, artifact is defined as an information source created or referenced in the course of applying security practices in software development, adapting IEEE’s definition of documentation, ‘any written or pictorial information describing, defining, specifying, reporting, or certifying activities, requirements, procedures, or results’ to the information sources used to manage security knowledge during practice application.
  + [<SecurityPrinciple>] – if one or more specific security principles are considered during application of a practice, their names can optionally be recorded here.
  + [<Results>] – An optional log entry containing details of the changes achieved by applying the practice (Mode=Log).
  + <When> - Date, time of practice application (Mode=Log)
  + <Agent> - the (optional, anonymized) identity of the person applying the practice, the filled-in template data. (Mode=Log)
  + <Mode> - Mode indicates the level of detail at which practice adherence is recorded. Mode=‘Plan’ indicates that the practice adherence record is a project level summary. Mode=‘Log’ indicates that the practice adherence record is a log entry. Once data has been collected at the ‘Log’ level, a set of log entries can be compared against a ‘Plan’-level record to assess adherence. Currently, only ‘Plan’ level records are collected.

## Practice templates

The following templates represent the collected advice of the security practice lists evaluated while building the framework.

### Address Security Requirements

As [Development, Testing, Operations],

In order to achieve: <SecurityGoals>,

Applies: ‘Address Security Requirements’

To: Design, Source Code, Software Release, Unit Tests, Test Plan, Test Suite, Operator Documentation User Documentation

Guided by: Security Requirements, Requirements, Threat Model, Data Classification Scheme

Producing: Source Code, Software Release, Unit Tests, Test Plan, Test Suite, Operator Documentation, User Documentation

Logging: <Activity>, <Results>

### Apply Data Classification Scheme

As [Development, Testing],

In order to achieve: <SecurityGoals>,

Applies: ‘Apply Data Classification Scheme’

To: Design, Source Code, Software Release, Unit Tests, Test Plan, Test Suite, Operator Documentation, User Documentation

Guided by: Data Classification Scheme

Producing: Source Code, Software Release, Unit Tests, Test Plan, Test Suite, Operator Documentation, User Documentation

Logging: <Activity>, <Results>

### Apply Secure Coding Standard

As [Development],

In order to achieve: <SecurityGoals>,

Applies: ‘Apply Secure Coding Standard’

To: Source Code, Software Release, Unit Tests

Guided by: Secure Coding Standard

Producing: Source Code, Software Release, Unit Tests

Logging: <Activity>, <Results>

### Build Threat Model

As [Management, Development, Testing]

In order to achieve <SecurityGoals>,

Applies ‘Build Threat Model’

To: Threat Model

Guided by: Assets, Threats, Bug Reports, Goals, Requirements, Technical Stack

Guided by some <SecurityPrinciple>

Producing: Threat Model

Logging: <Activity>, <Results>

<Phase?>

### Apply Automated Code Analysis Tools

As [Development],

In order to achieve: <SecurityGoals>,

Applies: ‘Apply Automated Code Analysis Tools’

To: Source Code, Software Release, Unit Tests

Guided by: Code Analysis test results (true positives), local tool customizations/tailoring, Technical Stack

Producing: Source Code, Software Release, Unit Tests

Logging: <Activity>, <Results>

### Create a Top N Bugs List

As [Management, Development, Testing, Operations],

In order to achieve: <SecurityGoals>,

Applies: ‘Create a Top N Bugs List’

To: Software System, Source Code, Software Release, Unit Tests, Test Plan, Test Suite, Technical Stack

Guided by: Bug Reports

Producing: Top N Bug List

Logging: <Activity>, <Results>

### Create technical stack recommendations

As [Management],

In order to achieve: <SecurityGoals>,

Applies: ‘Create Technical Stack’

To: Technical Stack

Guided by: Assets, Threats, Threat Model, Bug Reports

Producing: Technical Stack

Logging: <Activity>, <Results>

### Create and use automation to do what attackers will do

As [Testing],

In order to achieve: <SecurityGoals>,

Applies: ‘Create and use automation to do what attackers will do’

To: Test Suite

Guided by: Assets, Threats, Threat Model, Bug Reports

Producing: Test Suite

Logging: <Activity>, <Results>

### Drive tests with security requirements and security features

As [Testing],

In order to achieve: <SecurityGoals>,

Applies: ‘Drive tests with security requirements and security features’

To: Test Plan, Test Suite

Guided by: Security Requirements, Security Features

Producing: Test Plan, Test Suite

Logging: <Activity>, <Results>

### Enhance SSDL to prevent bugs found in operations

As [Management],

In order to achieve: <SecurityGoals>,

Applies: ‘Enhance SSDL to prevent bugs found in operations’

To: SSDL

Guided by: Bug Reports, <Activity>, <Results>

Producing: SSDL updates

Logging: <Activity>, <Results>

### Ensure edge/boundary value testing

As [Management, Testing],

In order to achieve: <SecurityGoals>,

Applies: ‘Ensure edge/boundary testing’

To: Software Release, Software System

Guided by: Security Requirements, Threat Model, Software Release, Software System

Producing: Test Plan, Test Suite

Logging: <Activity>, <Results>

### Ensure Technical Stack is Current

As [Management],

In order to achieve: <SecurityGoals>,

Applies: ‘Ensure Technical Stack is Current’

To: Technical Stack

Guided by: Patches, Vendor Updates, Release Notes

Producing: Technical Stack

Logging: <Activity>, <Results>

### Establish Data Classification Scheme

As [Management],

In order to achieve: <SecurityGoals>,

Applies: ‘Establish Data Classification Scheme’

To: Data Classification Scheme

Guided by: Assets, Threats, Requirements, Bug Reports

Producing: Data Classification Scheme

Logging: <Activity>, <Results>

### Examine and document likely threats to the organization and application type

As [Management],

In order to achieve: <SecurityGoals>,

Applies: ‘Examine and document likely threats to the organization and application type

To: Assets, Threats

Guided by: Assets, Threats, Bug Reports, News, Attacks

Producing: Assets, Threats

Logging: <Activity>, <Results>

### Fix all bugs found in operations

As [Development, Operations],

In order to achieve: <SecurityGoals>,

Applies: ‘Fix all bugs found in operations’

To: Source Code, Unit Tests, Software Release

Guided by: Bug Reports

Producing: Source Code, Unit Tests, Software Release

Logging: <Activity>, <Results>

### Fuzz test

As [Testing],

In order to achieve: <SecurityGoals>,

Applies: ‘Fuzz test’

To: Software Release, Software System

Guided by: Security Requirements, Threat Model, Software Release, Software System

Producing: Test Plan, Test Suite

Logging: <Activity>, <Results>

### Leverage coverage analysis

As [Development, Testing],

In order to achieve: <SecurityGoals>,

Applies: ‘Leverage coverage analysis’

To: Source Code, Software Release

Guided by: Technical Stack::Coverage Analysis Tools

Producing: Source Code, Unit Tests, Test Plan, Test Suite

Logging: <Activity>, <Results>

### Provide all available information to Penetration Testers

As [Management],

In order to achieve: <SecurityGoals>,

Applies: ‘Provide all available information to Penetration Testers’

To: Penetration Testers

Guided by: All Artifacts

Producing: Bug Reports

Logging: <Activity>, <Results>

### Provide guidance on handling security-related error messages

As [Management, Development, Testing],

In order to achieve: <SecurityGoals>,

Applies: ‘Provide guidance on handling security-related error messages’

To: Software Release, Software System

Guided by: All Artifacts

Producing: Operator Documentation, User Documentation

Logging: <Activity>, <Results>

### Record important security-specific knowledge affecting deployed app security

As [Management, Development, Testing],

In order to achieve: <SecurityGoals>,

Applies: ‘Record important security-specific knowledge affecting deployed app security’

To: Software Release, Software System

Guided by: All Artifacts

Producing: Operator Documentation, User Documentation

Logging: <Activity>, <Results>

### Review test results and correct, or formally accept the risks of releasing with failed checks

As [Management],

In order to achieve: <SecurityGoals>,

Applies: ‘Review test results and correct, or formally accept the risks of releasing with failed checks’

To: Software Release

Guided by: Test Plan, Test Suite, Bug Reports

Producing: Software Release

Logging: <Activity>, <Results>

### Specify Security Requirements

As [Management],

In order to achieve: <SecurityGoals>,

Applies: ‘Specify Security Requirements’

To: Security Requirements

Guided by: Assets, Threats, Requirements, Threat Model, Data Classification Scheme

Producing: Security Requirements

Logging: <Activity>, <Results>

### Use appropriate testing tools

As [Testing],

In order to achieve: <SecurityGoals>,

Applies: ‘Use appropriate testing tools’

To: Software Release, Software System

Guided by: Security Requirements, Threat Model, Software Release, Software System, Technical Stack::Test Tools

Producing: Test Plan, Test Suite

Logging: <Activity>, <Results>

### Use external penetration testers to find problems

As [Management, Testing],

In order to achieve: <SecurityGoals>,

Applies: ‘Use external penetration testers to find problems’

To: Software Release, Software System

Guided by: All Artifacts

Producing: Bug Reports

Logging: <Activity>, <Results>

### Use penetration testing tools internally

As [Testing],

In order to achieve: <SecurityGoals>,

Applies: ‘Use penetration testing tools internally’

To: Software Release, Software System

Guided by: All Artifacts

Producing: Bug Reports

Logging: <Activity>, <Results>

### Use threat model when creating test plan

As [Testing],

In order to achieve: <SecurityGoals>,

Applies: ‘Use threat model when creating test plan’

To: Test Plan, Test Suite

Guided by: Threat Model

Producing: Test Plan, Test Suite

Logging: <Activity>, <Results>

### Validate input and output against common vulnerabilities

As [Development, Testing],

In order to achieve: <SecurityGoals>,

Applies: ‘Validate input and output against common vulnerabilities’

To: Software Release, Software System

Guided by: Design, Software Release, Threat Model

Producing: Test Plan, Test Suite

Logging: <Activity>, <Results>

1. We encourage collaboration in the development of the SP-EF. If you wish to extend or adapt the SP-EF for use in your own studies, please fork the github repository. [↑](#footnote-ref-1)
2. http://www.webroot.com/blog/2013/02/28/how-much-does-it-cost-to-buy-10000-u-s-based-malware-infected-hosts/ [↑](#footnote-ref-2)
3. <http://krebsonsecurity.com/2011/11/how-much-is-your-identity-worth/> [↑](#footnote-ref-3)