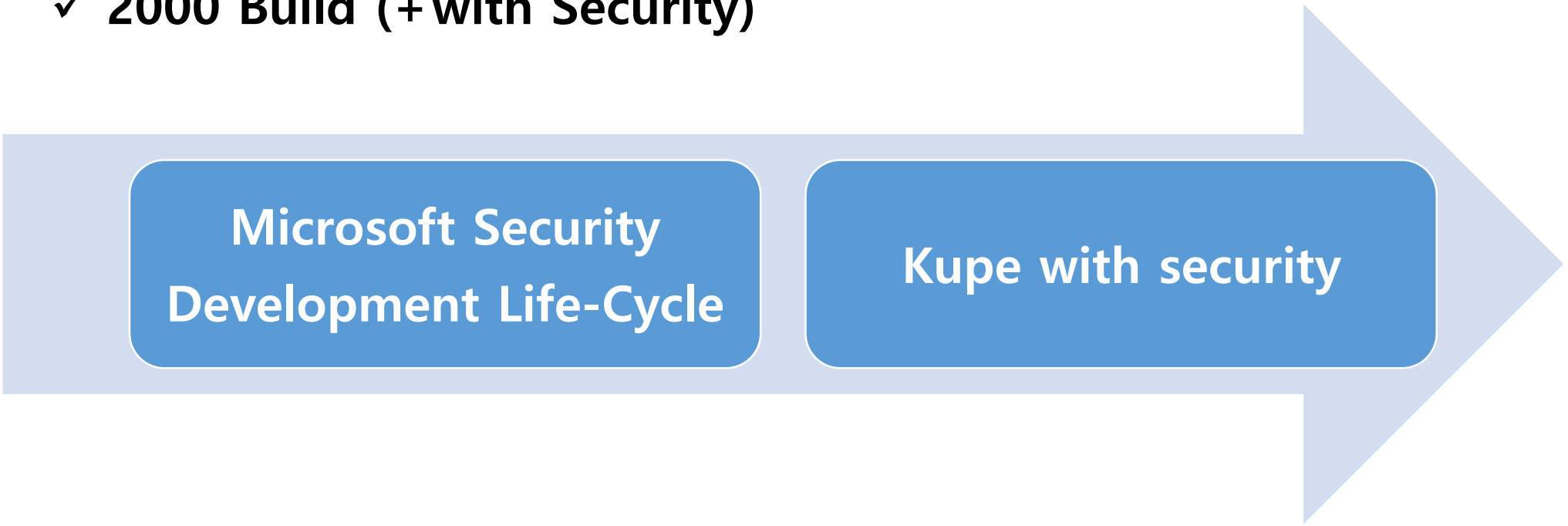
IT융합정보보호학과
신민용 원찬희 이상진

Agenda

- 1. Microsoft Security Development Life-Cycle
- 2. Kupe with security
  - ✓ Overview
  - ✓ 1000 Plan and Elaboration (+Security)
  - ✓ 2000 Build (+with Security)



KU 건국대학교 KONKUK UNIV.

KUPE

Konkuk Unified Process for Education

+

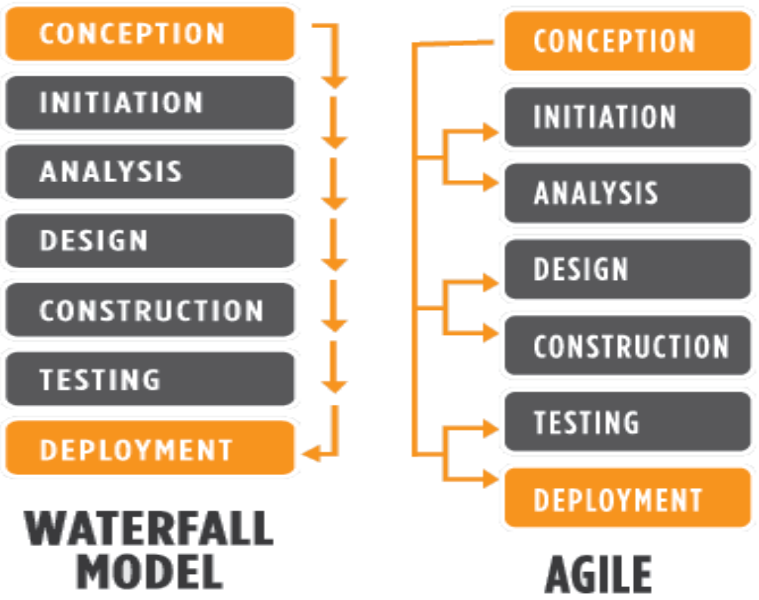
Microsoft
Security Development Life-Cycle

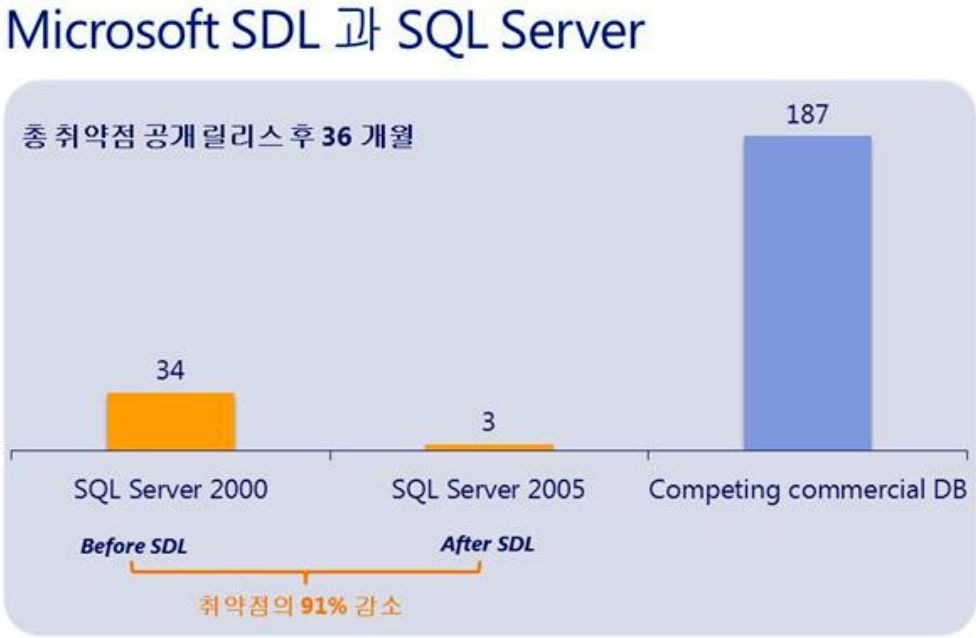
KUPE with security enforce

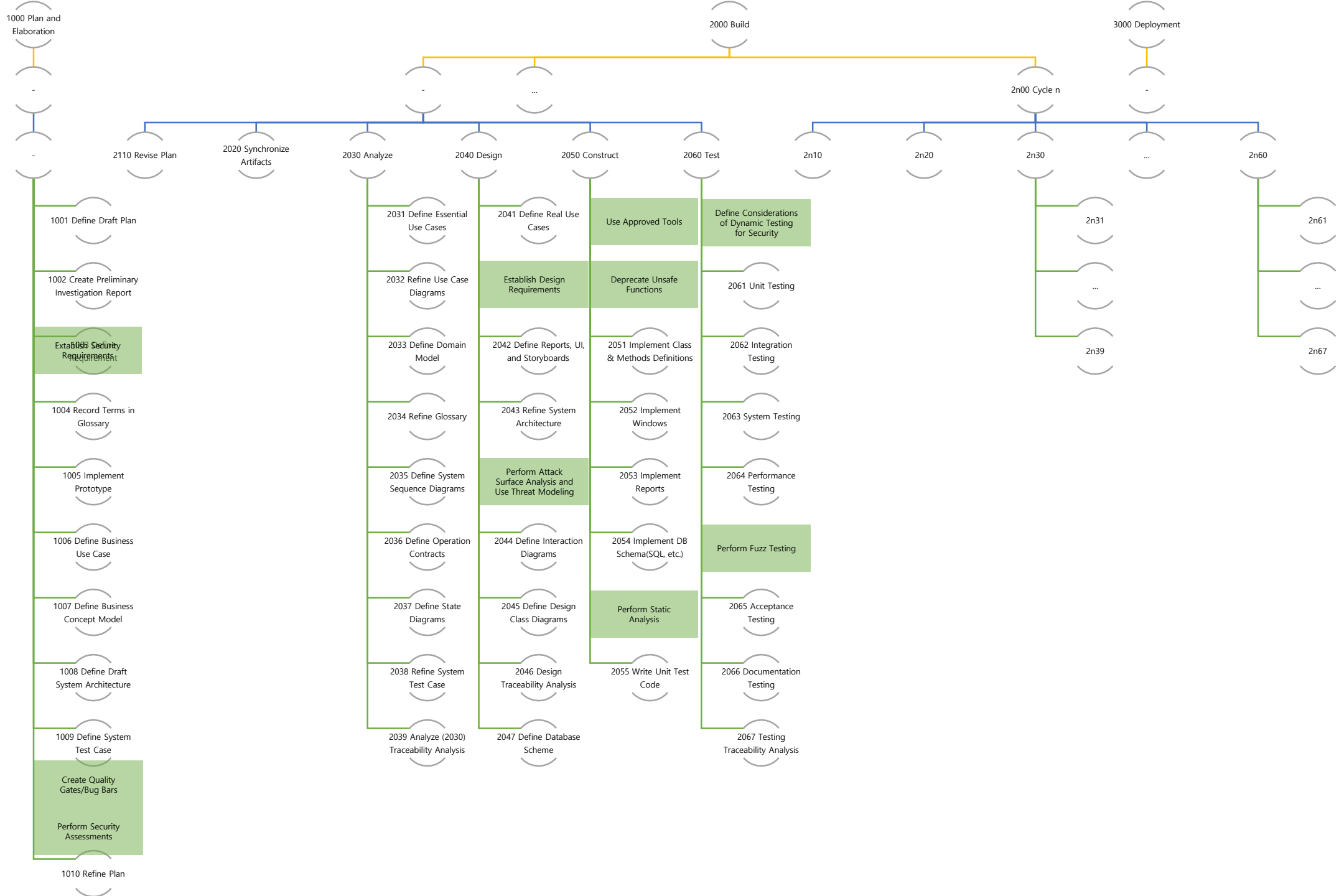
Microsoft Security Development Life-Cycle

- **Description**
  - 마이크로소프트의 보안 개발 방법에 대한 투명한 제공을 위해 제작된 보안 프로세스
  - 연간 교육을 통해 매우 정교하며 실제로 MS R&D에서 개발 적용 중
  - 모든 종류의 소프트웨어 개발 프로젝트 적용 가능
  - 다양한 방법론 적용 가능

    ex) WaterFall, Spiral, agile Development



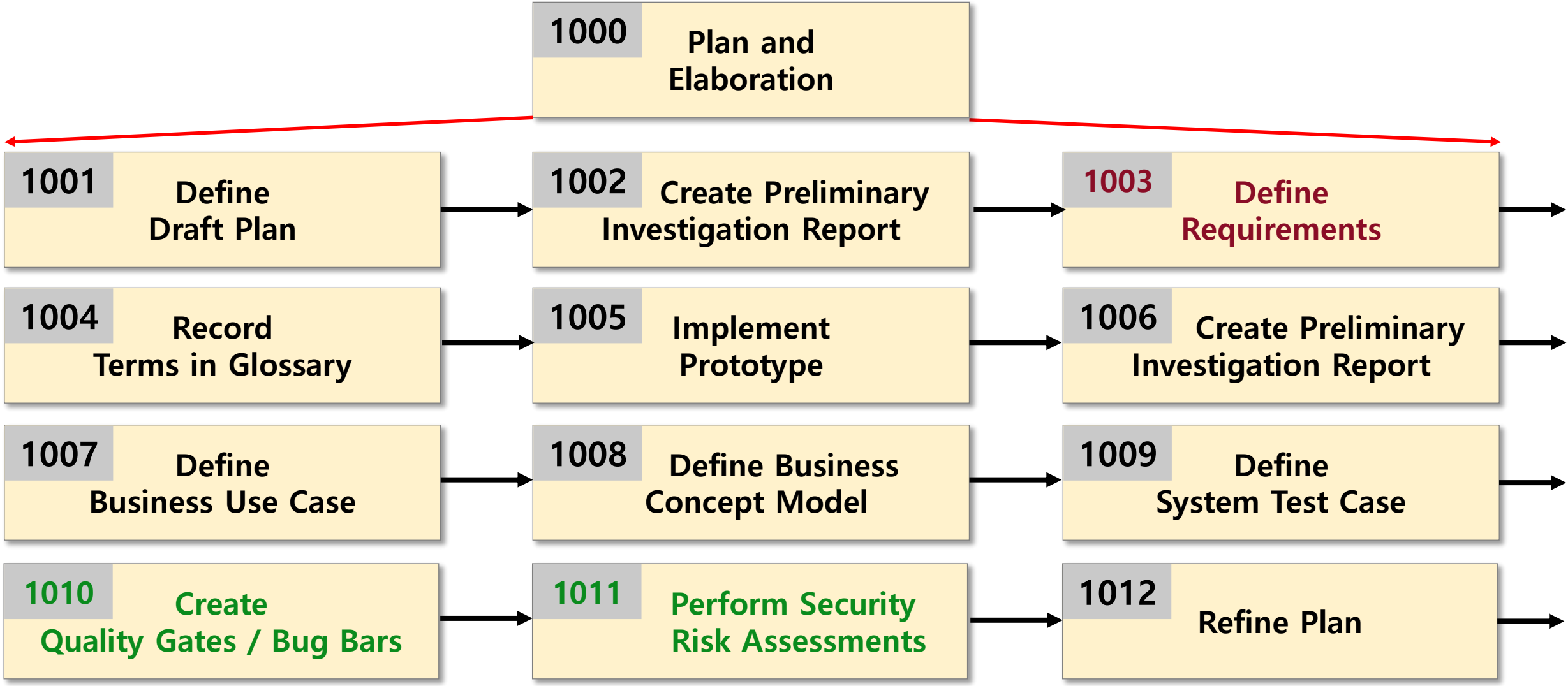



1000 Plan and Elaboration
2000 Build
3000 Deployment
-
-
...
2n00 Cycle n
-
-
2110 Revise Plan
2020 Synchronize Artifacts
2030 Analyze
2040 Design
2050 Construct
2060 Test
2n10
2n20
2n30
...
2n60
1001 Define Draft Plan
1002 Create Preliminary Investigation Report
1003 Establish Security Requirements
1004 Record Terms in Glossary
1005 Implement Prototype
1006 Define Business Use Case
1007 Define Business Concept Model
1008 Define Draft System Architecture
1009 Define System Test Case
Create Quality Gates/Bug Bars
Perform Security Assessments
1010 Refine Plan
2031 Define Essential Use Cases
2032 Refine Use Case Diagrams
2033 Define Domain Model
2034 Refine Glossary
2035 Define System Sequence Diagrams
2036 Define Operation Contracts
2037 Define State Diagrams
2038 Refine System Test Case
2039 Analyze (2030) Traceability Analysis
2041 Define Real Use Cases
Establish Design Requirements
2042 Define Reports, UI, and Storyboards
2043 Refine System Architecture
Perform Attack Surface Analysis and Use Threat Modeling
2044 Define Interaction Diagrams
2045 Define Design Class Diagrams
2046 Design Traceability Analysis
2047 Define Database Scheme
Use Approved Tools
Deprecate Unsafe Functions
2051 Implement Class & Methods Definitions
2052 Implement Windows
2053 Implement Reports
2054 Implement DB Schema(SQL, etc.)
Perform Static Analysis
2055 Write Unit Test Code
Define Considerations of Dynamic Testing for Security
2061 Unit Testing
2062 Integration Testing
2063 System Testing
2064 Performance Testing
Perform Fuzz Testing
2065 Acceptance Testing
2066 Documentation Testing
2067 Testing Traceability Analysis
2n31
...
2n39
2n61
...
2n67

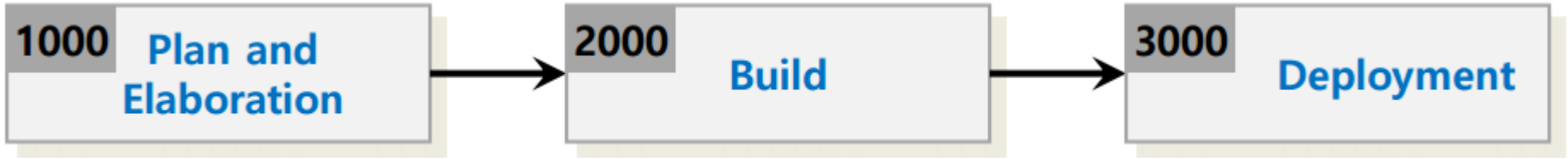
Stage 1000. Plan and Elaboration

- Sec+ Stage 1000 Activities

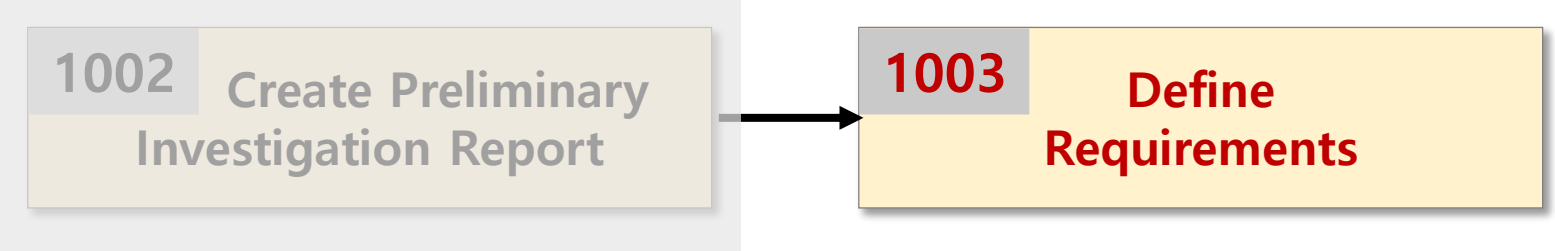
- Addition of activities
- Addition of contents



수정



Activity 1003.
Define Requirements



- **Description**
  - Write a requirement specification for a product
  - Input : draft project plan, investigation report
  - Output : a requirement specification

- **What is a requirement? (IEEE Std610.12-1990) “**

Activity 1003. Define Requirements

- **Functional requirements “**

- **Non-Functional requirements**
  - Constraints on the services or functions offered by the system as timing constraints, constraints on the development process, standards, etc.
  - Portability, Reliability, Usability, Efficiency(Space, Performance)
  - Delivery, Implementation, Standards
  - Ethical, Interoperability, Legislative(Safety, Privacy)
  - Establish security requirements that consider best integrate security into the development process and identify key security objectives

- **Recommended reference : IEEE Std. 830-1998**

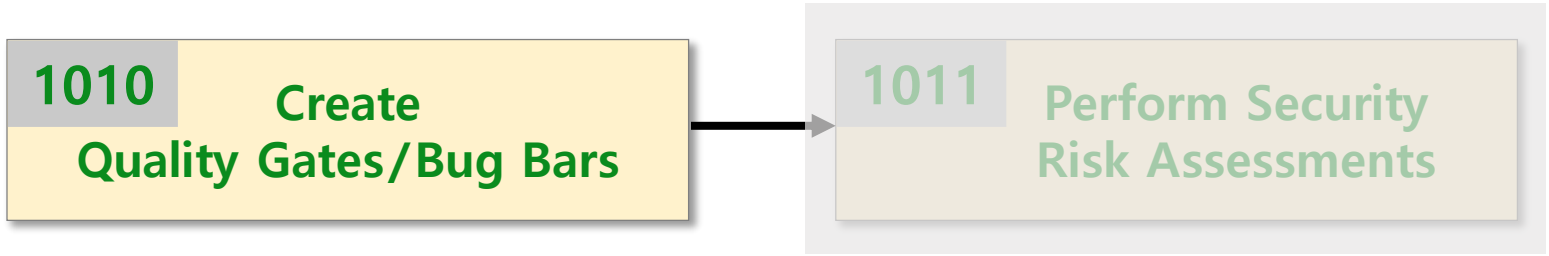
Activity 1003. Define Requirements

- **Steps**
  1. Gather all kinds of useful documents
  2. Write an overview statement (objective and name of the system, etc.)
  3. Determine customers who use the product
  4. Write goals of the project
  5. Identify system functions
     - Functional requirements
     - Add function references(such as R1.1, …) into the identified functions
     - Categorize identified functions into Event, Hidden, and Frill

Activity 1003. Define Requirements

6. Identify system attributes
    - Non-functional requirements
        - Especially, security requirements
            1. Identify the team or individual that tracks and manages security for the product
            2. Define minimum security criteria
            3. Identify security requirements
            4. Specify bug/work tracking tool
7. Identify other requirements (Optional)
    - Assumptions, Risks, Glossary, etc.

Activity 1010.
Create Quality Gates/Bug Bars



- **Description**
  - Define criteria of quality gates and bug bars to establish minimum acceptable levels of security quality
  - Improves the understanding of risks associated with security issues and enables teams to identify and fix security bugs during development
  - Input : investigation report, requirement specification, prototype, business use case model
  - Output : a quality gates/bug bars criteria

Activity 1010. Create Quality Gates/Bug Bars

- **What is Quality Gates?**
  - Check quality for approaching levels of objective
  - Depend on functions and approaches
  - The term 'Quality Gate' is not used consistently and can be applied to:
    - project management
    - quality management
    - risk management
    - any combination of these three management disciplines
  - Can be found in different software process models
    - the V-Modell XT of the German federal administration
    - Cooper's Stage-gate model

Activity 1010. Create Quality Gates/Bug Bars

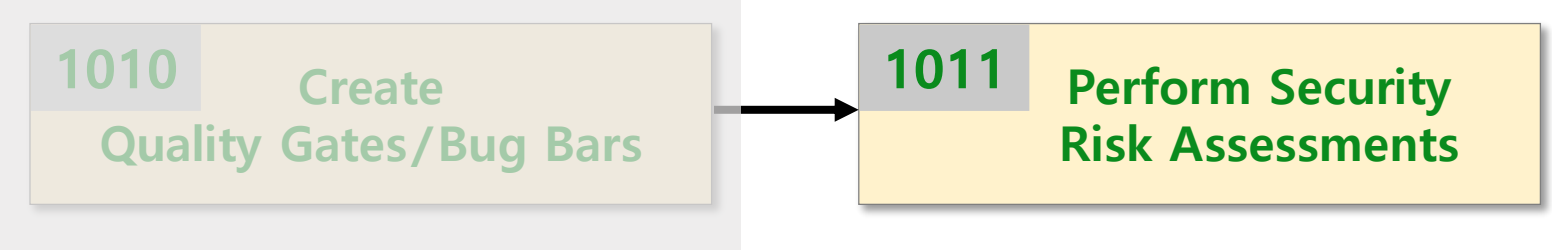
- **What are bug bars?**
  - A bug bar is a quality gate which is used to define the severity thresholds of security vulnerabilities
  - Used to define the severity thresholds of security vulnerabilities
  - Vulnerabilities (security bugs values)
    - ✓ Spoofing
    - ✓ Tampering
    - ✓ Repudiation
    - ✓ Information Disclosure
    - ✓ Denial of Service (DoS)

Activity 1010. Create Quality Gates/Bug Bars

- example of criteria for bug bars:

Server	
Critical	Elevation of privilege: The ability to either execute arbitrary code or obtain more privilege than authorized ● Remote anonymous user ▪ Examples: ✓ Unauthorized file system access: arbitrary writing to the file system ✓ Execution of arbitrary code ✓ SQL injection (that allows code execution) ● All write access violations (AV), exploitable read AVs, or integer overflows in remote anonymously callable code
Important	Non-default critical scenarios or cases where mitigations exist that can help prevent critical scenarios. ● Denial of service: Must be "easy to exploit" by sending a small amount of data or be otherwise quickly induced Anonymous ▪ Persistent DoS ▪ Examples: ✓ Sending a single malicious TCP packet results in a Blue Screen of Death (BSoD) ✓ Sending a small number of packets that causes a service failure ▪ Temporary DoS with amplification ▪ Examples: ✓ Sending a small number of packets that causes the system to be unusable for a period of time ✓ A web server (like IIS) being down for a minute or longer ✓ A single remote client consuming all available resources (sessions, memory) on a server by establishing sessions and keeping them open
Moderate	● Denial of serviceAnonymous ▪ Temporary DoS without amplification in a default/common install. ▪ Example: ✓ **Multiple remote clients** consuming all available resources (sessions, memory) on a server by establishing sessions and keeping them open
Low	● Information disclosure (untargeted) ▪ Runtime information ▪ Example: ✓ Leak of random heap memory

Activity 1011.
Perform Security Risk Assessments



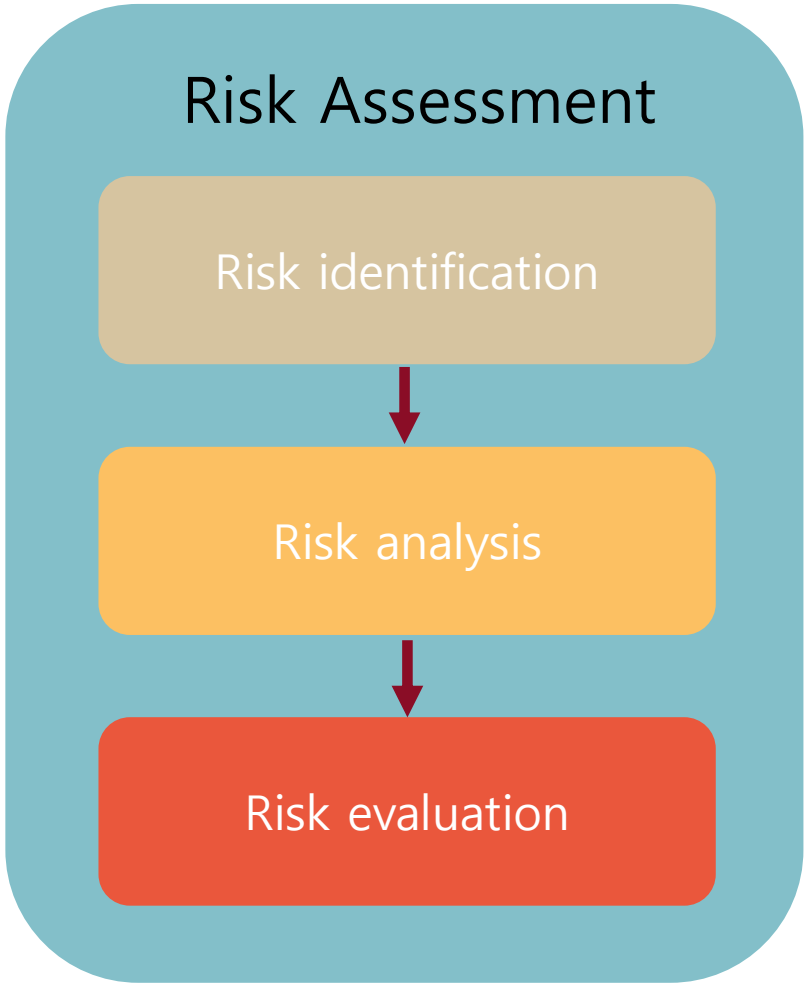
- **Description**
  - identify functional aspects of the software that require deep review.
  - include the following information:
    - What portions of the project will require threat models before release.
    - What portions of the project will require security design reviews before release.
    - What portions of the project will require penetration testing (pen testing) by a mutually agreed-upon group that is external to the project team. Any portion of the project that requires pen testing must resolve issues identified during pen testing before it is approved for release.
    - Any additional testing or analysis requirements the security advisor deems necessary to mitigate security risks.
    - Clarification of the specific scope of fuzz testing requirements.

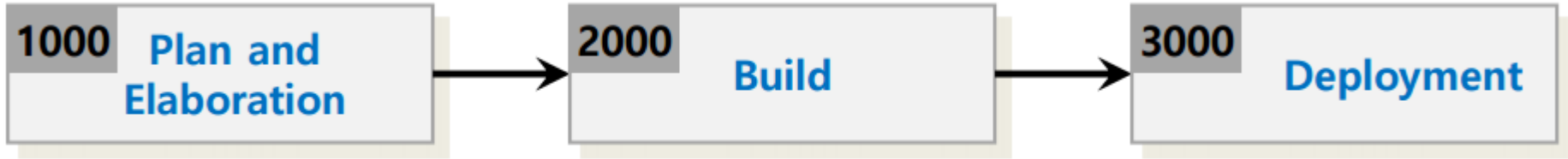
Activity 1011. Perform Security Risk Assessments

- Input : investigation report, requirement specification, business use case model, draft system architecture, quality gates/bug bars criteria
- Output : a security risk assessments report

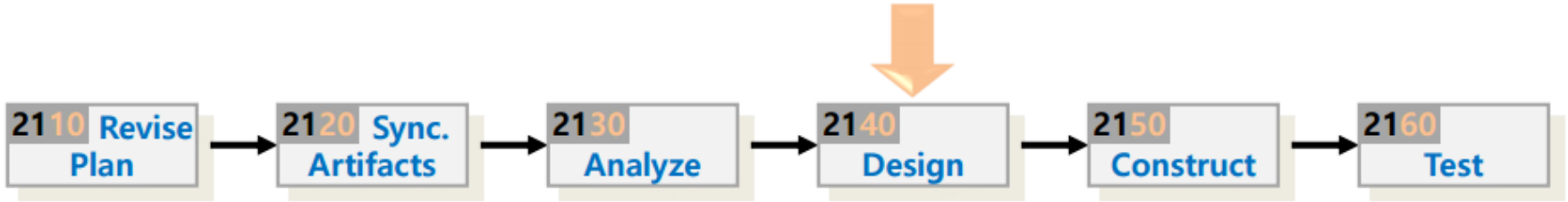
- **Steps**
  1. Risk identification
  2. Risk analysis
  3. Risk evaluation

- **Recommended reference : ISO/IEC 31000, 27005**





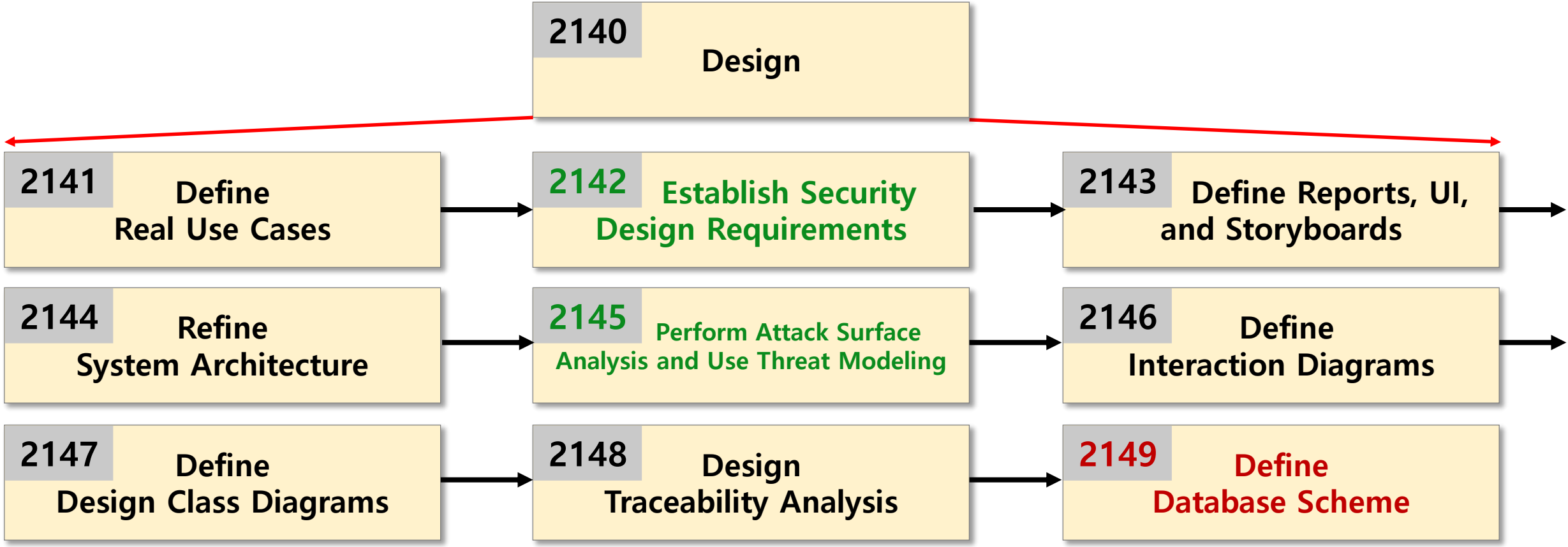
Phase 2040. Design



Phase 2040. Design

- Sec+ Phase 2040 Activities

- Addition of activities
- Addition of contents



Activity 2042.
Establish Security Design Requirements

- **Description**
  - Consider security design requirements
  - Input : intermediate deliverables
  - Output : security design requirements specifications

Activity 2042.
Establish Security Design Requirements

- **What are design requirements?**
  - Describe security features that will be directly exposed to users
  - Describe how to securely implement all functionality provided by a given feature or function

Activity 2042.
Establish Security Design Requirements

- **Steps**
  1. Creation of security design specifications
  2. Review specification
     - Validate design specifications against the application's functional specification
     - Functional specification
       - Accurately and completely describe the intended use of a feature or function
       - Describe how to deploy the feature or function in a secure fashion
  3. Specify of minimal cryptographic design requirements
     - SDL cryptographic requirements (at a high-level)
       - Use AES for symmetric encryption/decryption
       - Use RSA for asymmetric encryption/decryption and signatures
       - Use 1024-bit or better RSA keys
       - Use SHA-256 or better for hashing and message authentication codes

Activity 2045.
perform Attack Surface Analysis and Threat Modeling

- **Description**
  - Reducing the opportunities for attackers to exploit a potential weak spot or vulnerability
    - Analyze overall attack surface
    - Includes disabling or restricting access to system services, applying the principle of least privilege, and employing layered defenses wherever possible
  - Input : intermediate deliverables
  - Output : attack surface analysis, threat model

Activity 2045. perform Attack Surface Analysis, Threat Modeling

- **What is attack surface?**
  - Risk by giving attackers opportunity to exploit a potential vulnerability
    - Attack surface reduction encompasses the following:
      - shutting off or restricting access to system services
      - employing layered defenses wherever possible
      - applying the principle of least privilege

- **What is Threat Modeling?**
  - Systematic process used to identify threats and vulnerabilities in software
  - Consider security issues at the component or application level
  - Team exercise encompassing program/project managers, developers, and testers
  - Primary security analysis task

Activity 2045. perform Attack Surface Analysis, Threat Modeling

- **Steps**
  1. Analysis attack surface
     - Use Code Access Security (CAS) correctly
       - When developing with managed code, use strong-named assemblies and request minimal permission.
       - When using strong-named assemblies, do not use APTCA (Allow Partially Trusted Caller Attribute) unless the assembly was approved by a security review.
     - Manage firewall exceptions carefully
       - Be logical and consistent when you make firewall exceptions
     - Ensure your application runs correctly as a non-administrator
       - following the requirement will enable teams to design and develop their applications with a standard user in mind
       - This will result in reducing attack surface exposed by applications, increasing the security of the user and system

Activity 2045. perform Attack Surface Analysis, Threat Modeling

- **Steps**
  2. Threat Modeling
     1) Draw a diagram using DFDs(Data Flow Diagrams)
        - Include processes, data stores, data flows, and trust boundaries
        - Diagrams per scenario may be helpful
     2) Identify threats
        - Use STRIDE
        - Get specific about threat manifestation
        - Apply STRIDE threats to each element

Activity 2045. perform Attack Surface Analysis, Threat Modeling

3) Mitigation

- address or alleviate a problem
- Mitigation is the point of threat modeling (Design secure software)
- Ways to address threats
  - Redesign to eliminate
  - Apply standard mitigations
  - Invent new mitigations (riskier)
  - Accept vulnerability in design

3) Validation

- Validate threat models, quality of threats and mitigations

• Microsoft Threat Modeling Tool 2016

- address or alleviate

Activity 2045. perform Attack Surface Analysis, Threat Modeling

- **Steps**
  2. Threat Modeling

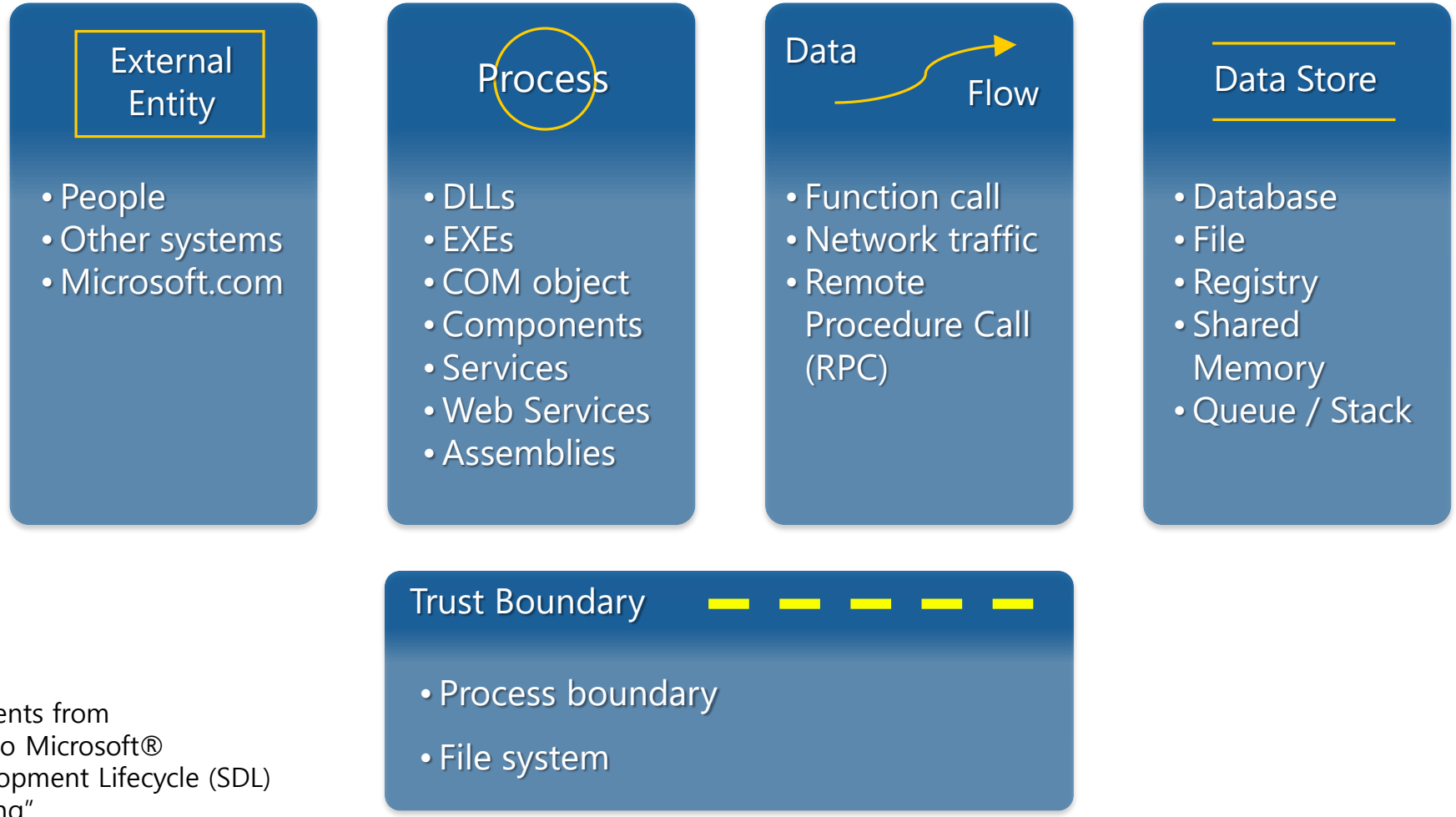


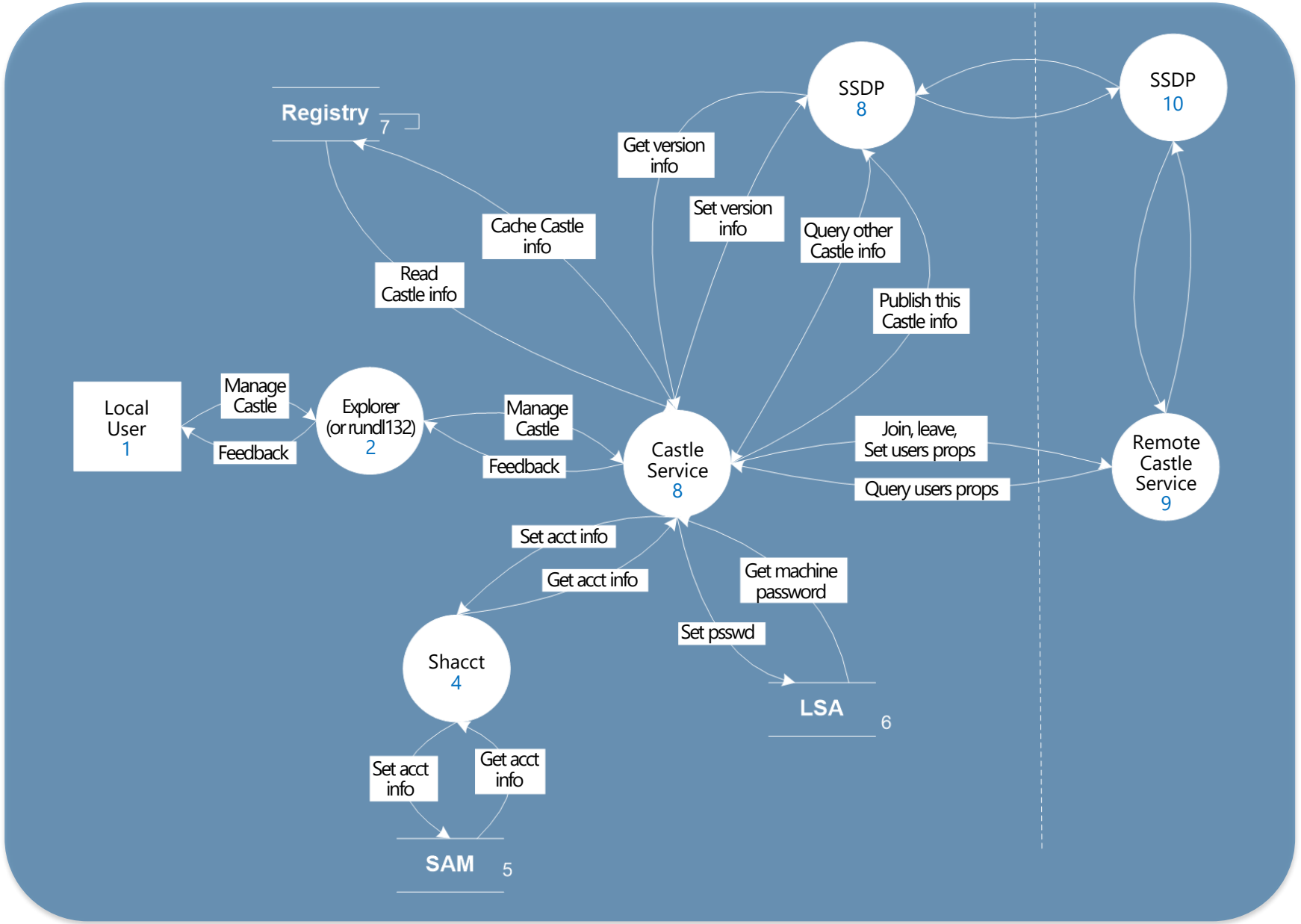
Diagram Elements from "Introduction to Microsoft® Security Development Lifecycle (SDL) Threat Modeling"

Activity 2045. perform Attack Surface Analysis, Threat Modeling

- **Steps**

  2. Threat Modeling

Level 1 Diagram from "Introduction to Microsoft® Security Development Lifecycle (SDL) Threat Modeling"



Activity 2045. perform Attack Surface Analysis, Threat Modeling

- **Steps**
  2. Threat Modeling

Threat	Spoofing
Property	Authentication
Definition	Impersonating something or someone else
Example	Pretending to be any of billg, microsoft.com, or ntdll.dll

S T R I D E

Threat	Property we want
Spoofing	Authentication
Tampering	Integrity
Repudiation	Nonrepudiation
Information Disclosure	Confidentiality
Denial of Service	Availability
Elevation of Privilege	Authorization

Extracts from "Introduction to Microsoft® Security Development Lifecycle (SDL) Threat Modeling"

Activity 2045. perform Attack Surface Analysis, Threat Modeling

- **Steps**
  2. Threat Modeling

Spoofing	Authentication	To authenticate principals: • Cookie authentication • Kerberos authentication • PKI systems such as SSL/TLS and certificates To authenticate code or data: • Digital signatures
Tampering	Integrity	• Windows Vista Mandatory Integrity Controls • ACLs • Digital signatures
Repudiation	Non Repudiation	• Secure logging and auditing • Digital Signatures
Information Disclosure	Confidentiality	• Encryption • ACLS
Denial of Service	Availability	• ACLs • Filtering • Quotas
Elevation of Privilege	Authorization	• ACLs • Group or role membership • Privilege ownership • Input validation

Standard Mitigations from "Introduction to Microsoft® Security Development Lifecycle (SDL) Threat Modeling"

Activity 2045. perform Attack Surface Analysis, Threat Modeling

- **Steps**

  2. Threat Modeling

  - Mitigation #54, Rasterization Service performs the following mitigation strategies:
    1. OM is validated and checked by (component) before being handed over to Rasterization Service
    2. The resources are decoded and validated by interacting subsystems, such as [foo], [bar], and [boop]
    3. Rasterization ensures that if there are any resource problems while loading and converting OM to raster data, it returns a proper error code
    4. Rasterization Service will be thoroughly fuzz tested

  (Comment: Fuzzing isn't a mitigation, but it's a great thing to plan as part 4)

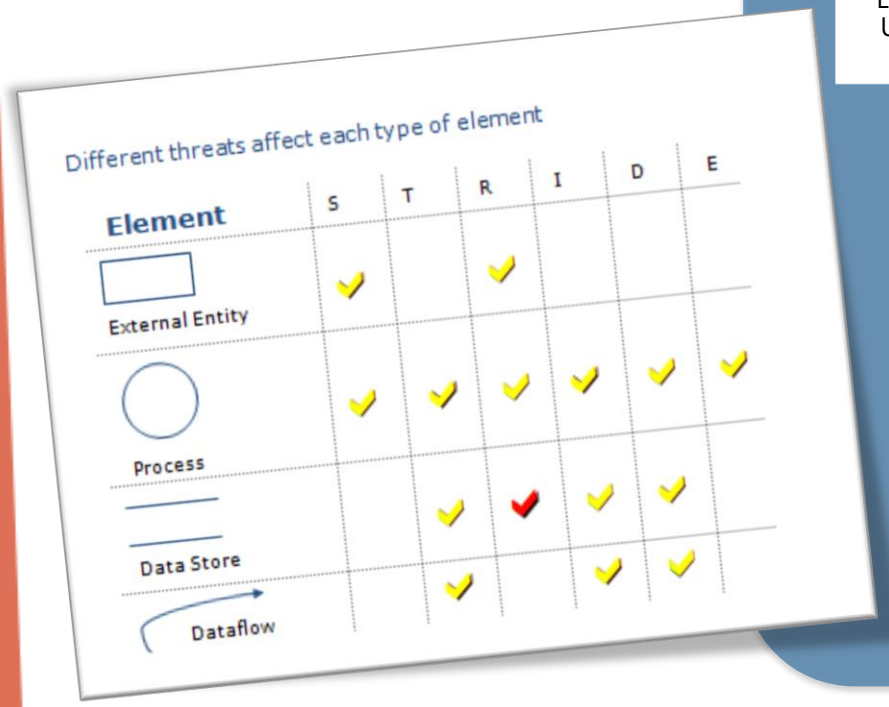
Sample Mitigation from "Introduction to Microsoft® Security Development Lifecycle (SDL) Threat Modeling"

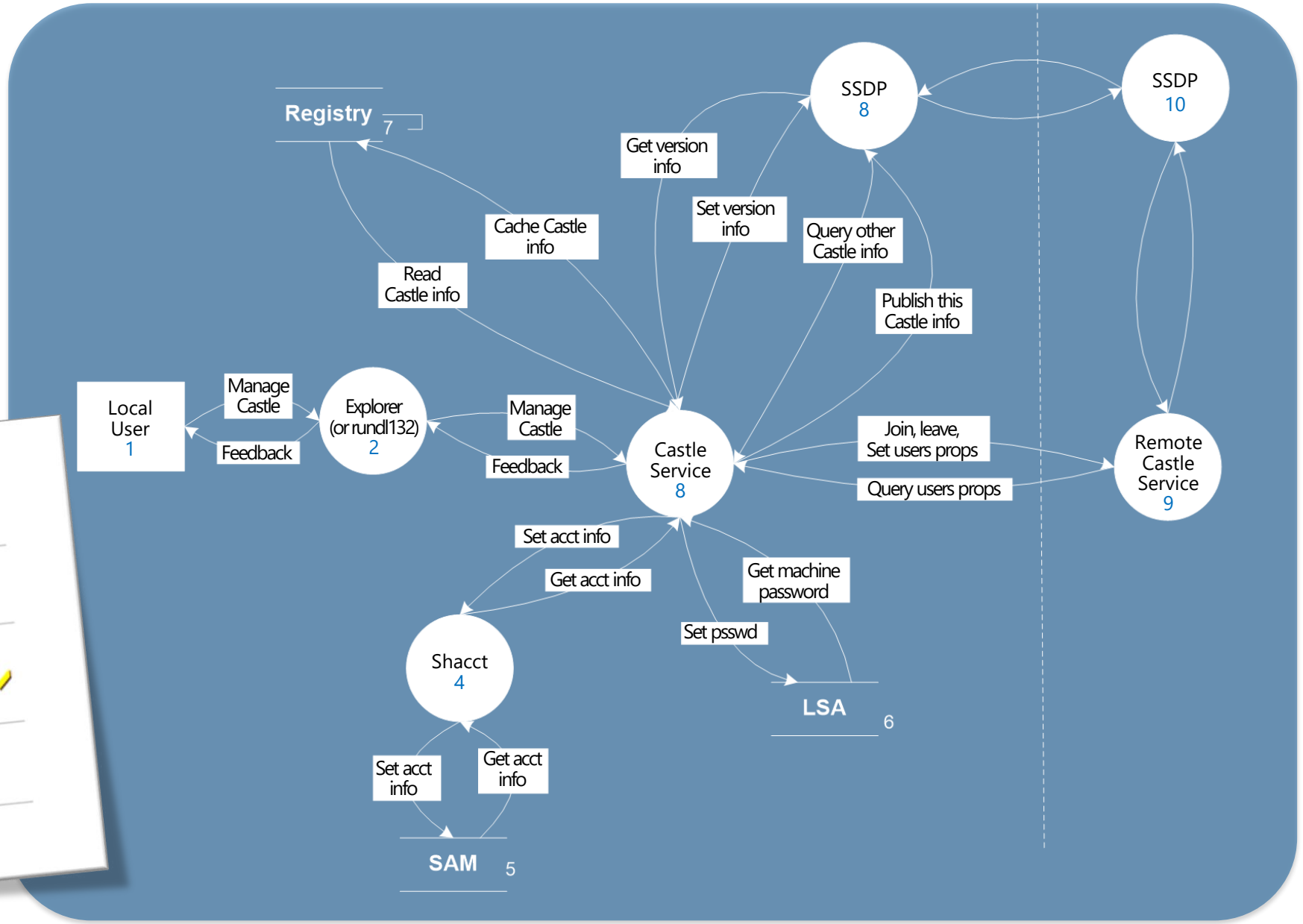
Activity 2045. perform Attack Surface Analysis, Threat Modeling

- **Steps**

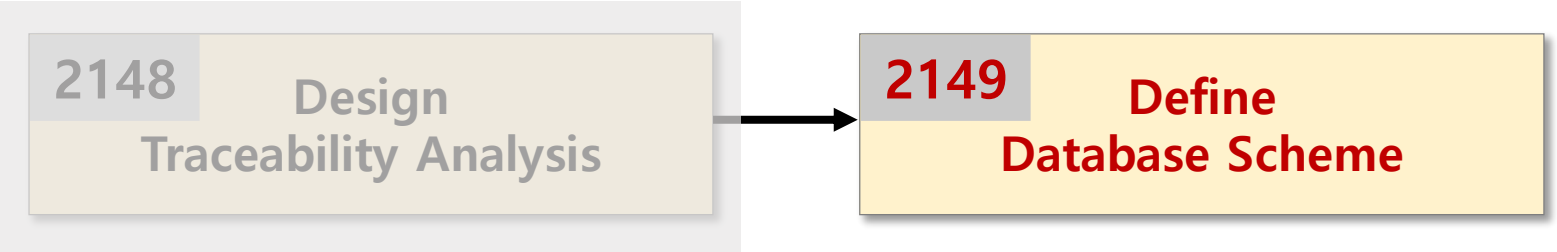
  2. Threat Modeling

Level 1 Diagram from
"Introduction to Microsoft® Security Development Lifecycle (SDL) Threat Modeling"





Activity 2049.
Define Database Scheme

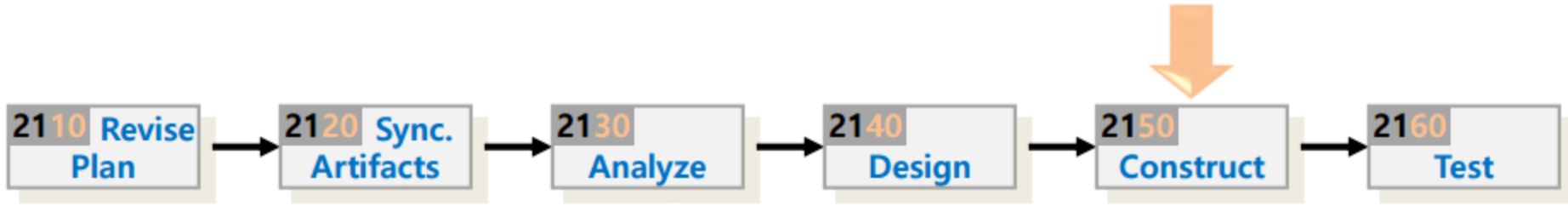


- **Description**
  - Design database, table, and records
  - Map classes into tables
  - Take account of security requirements
  - Input : Design Class Diagram
  - Output : A Database Schema

Activity 2049. Define Database Scheme

- **Steps**
  1. Map classes into tables
  2. Map relationships between classes into relations between tables
  3. Map attributes into fields of tables
  4. Design Schema

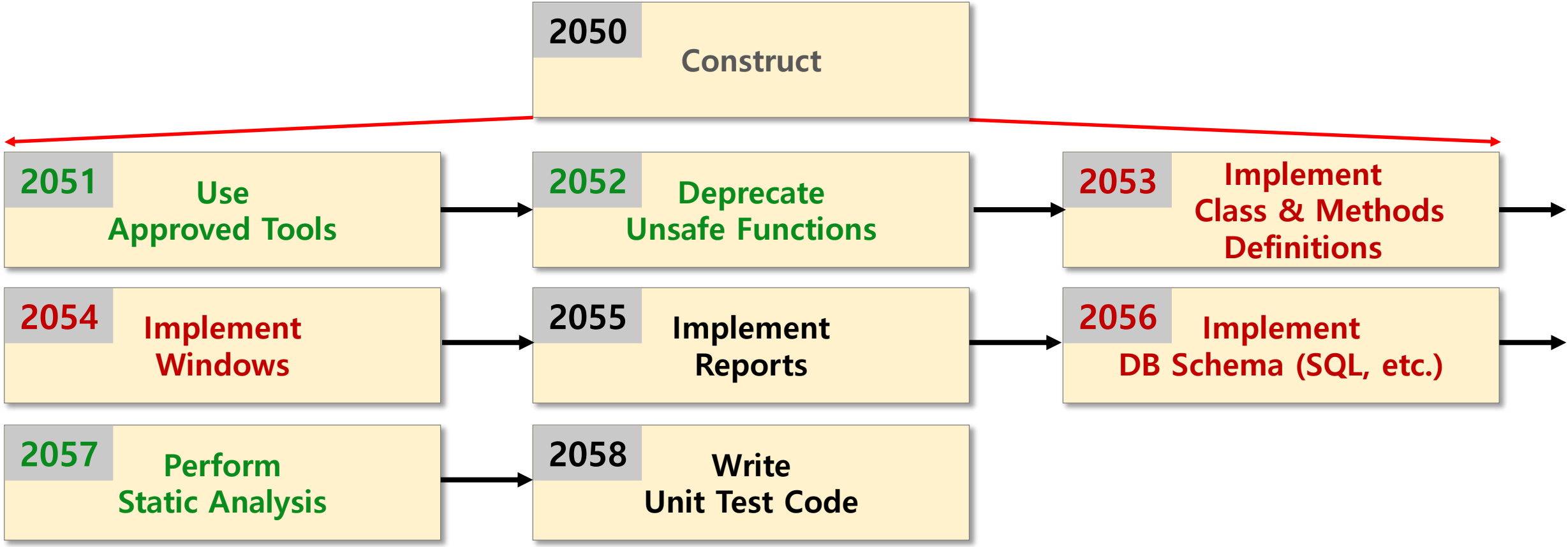
Phase 2050.
Construct



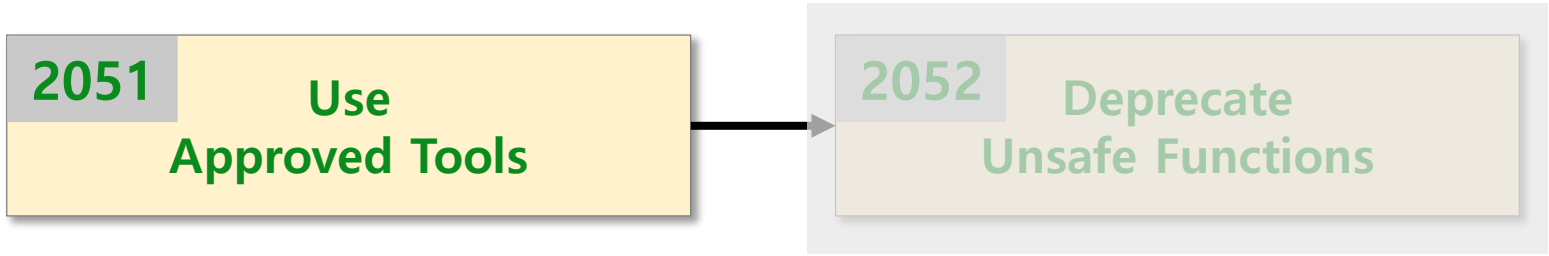
Phase 2050. Construct

- Sec+ Phase 2050 Activities

- Addition of activities
- Addition of contents



Activity 2051.
Use Approved Tools



- **Description**
  - Prepare to use approved tools and define their associated security checks

- **Steps**
  1. Define a list of approved tools and their associated security checks
     - Such as compiler/linker options and warnings
  2. Publish the list
  3. Approve the list

Activity 2052.
Deprecate Unsafe Functions

- **Description**
  - Analyze all functions and APIs that will be used in conjunction with a software development project and prohibit those that are determined to be unsafe.
  - Input : design class diagram, real use cases, interaction diagram
  - Output : a list of unsafe functions and APIs, safer alternatives

Activity 2052. Deprecate Unsafe Functions

- Examples of unsafe functions (overrun)

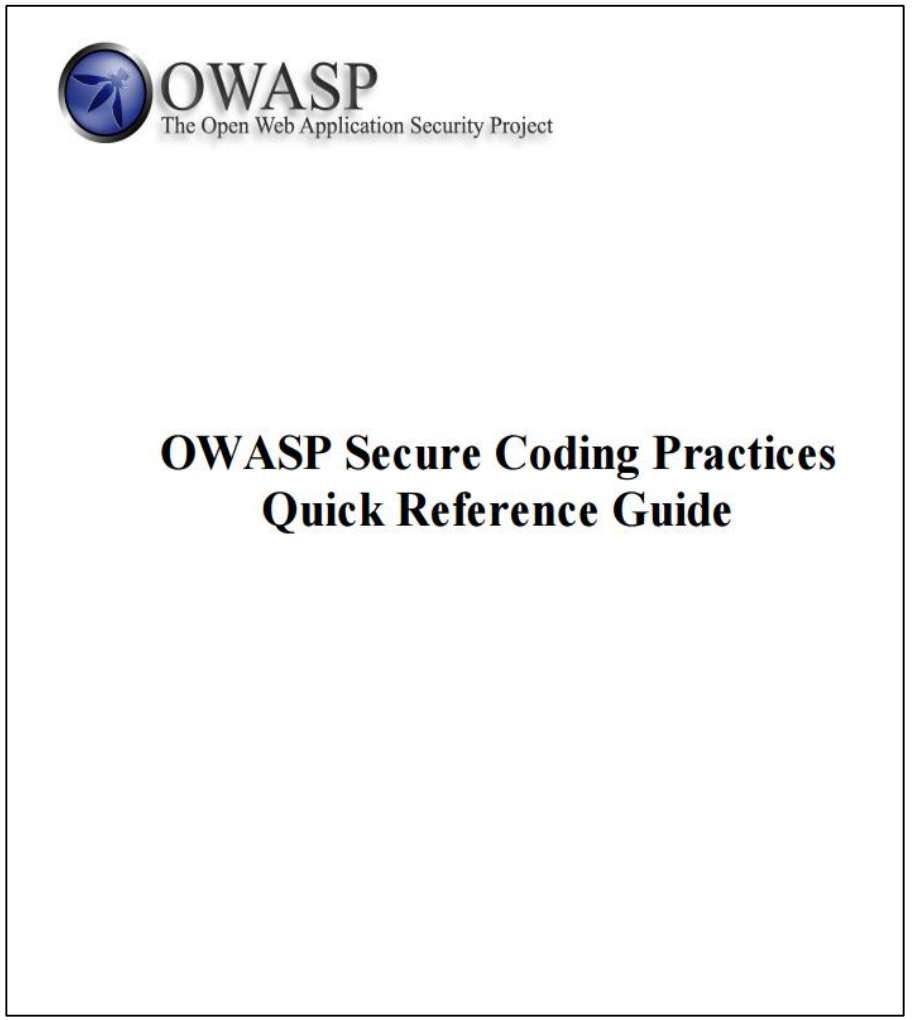
```
TCHAR   *src = TEXT("123456789");
TCHAR   dest[5];
_tcscpy(dest, src);
```

Function	Replaces
strcpy	StringCbCopy, StringCbCopyEx, StringCchCopy, StringCchCopyEx
strncpy	StringCbCopyN, StringCbCopyNEx, StringCchCopyN, StringCchCopyNEx
strcat	StringCbCat, StringCbCatEx, StringCchCat, StringCchCatEx
strncat	StringCbCatN, StringCbCatNEx, StringCchCatN, StringCchCatNEx
gets	StringCbGets, StringCbGetsEx, StringCchGets, StringCchGetsEx

Activity 2052. Deprecate Unsafe Functions

- **Information related to secure coding :**
  - OWASP Top 10 & Guideline of Secure coding(Ref)

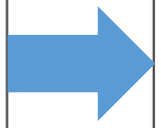
OWASP Top
A1 – Injection
A2 – Broken Authentication and Session Management
A3 – Cross-Site Scripting (XSS)
A4 – Insecure Direct Object References
A5 – Security Misconfiguration
A6 – Sensitive Data Exposure
A7 – Missing Function Level Access Control
A8 – Cross-Site Request Forgery (CSRF)
A9 – Using Known Vulnerable Components
A10 – Unvalidated Redirects and Forwards



Activity 2052. Deprecate Unsafe Functions

- Example of secure coding (Use of Hard-coded Password)

```
1:  public class connectDB
2:  {
3:      private Connection conn;
4:
5:      public Connection DBConnect(String url, String id)
6:      {
7:          try
8:          {
9:              // password가 하드-코드 되어있다.
10:             conn =    DriverManager.getConnection(url, id, "tiger");
11:         }
12:         catch (SQLException e)
13:         {
14:             System.err.println("...");
15:         }
16:         return conn;
17:     }
18: }
```



```
1:  public class connectDB
2:  {
3:    public Connection connect(Properties props) throws NoSuchAlgorithmException,
    NoSuchPaddingException, InvalidKeyException, IllegalBlockSizeException, BadPaddingException
4:    {
5:      try
6:      {
7:        String url = props.getProperty("url");
8:        String id = props.getProperty("id");
9:        String pwd = props.getProperty("passwd");
10:
11:       //외부 설정 파일에서 패스워드를 가져오며, 패스워드가 값이 있는지 체크하고 있음
12:       if (url != null && !"".equals(url) && id != null &&    !"".equals(id) && pwd != null
    &&    !"".equals(pwd))
13:       {
14:         KeyGenerator kgen = KeyGenerator.getInstance("Blowfish");
15:         SecretKey skey = kgen.generateKey();
16:         byte[] raw = skey.getEncoded();
17:         SecretKeySpec skeySpec = new SecretKeySpec(raw, "Blowfish");
18:
19:         Cipher cipher = Cipher.getInstance("Blowfish");
20:         cipher.init(Cipher.DECRYPT_MODE, skeySpec);
21:         byte[] decrypted_pwd = cipher.doFinal(pwd.getBytes());
22:         pwd = new String(decrypted_pwd);
23:         conn = DriverManager.getConnection(url, id, pwd);
24:       }
25:     }
26:     catch (SQLException e)
27:     {
28:       ......
```

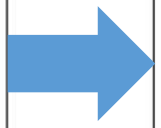
[Measure]

- Passwords hassing
- Strong user passwords

Activity 2052. Deprecate Unsafe Functions

- Example of secure coding (Use of Hard-coded Password)

```
1:  public class connectDB
2:  {
3:      private Connection conn;
4:
5:      public Connection DBConnect(String url, String id)
6:      {
7:          try
8:          {
9:              // password가 하드-코드 되어있다.
10:             conn =    DriverManager.getConnection(url, id, "tiger");
11:         }
12:         catch (SQLException e)
13:         {
14:             System.err.println("...");
15:         }
16:         return conn;
17:     }
18: }
```



```
1:  public class connectDB
2:  {
3:    public Connection connect(Properties props) throws NoSuchAlgorithmException,
    NoSuchPaddingException, InvalidKeyException, IllegalBlockSizeException, BadPaddingException
4:    {
5:      try
6:      {
7:        String url = props.getProperty("url");
8:        String id = props.getProperty("id");
9:        String pwd = props.getProperty("passwd");
10:
11:       //외부 설정 파일에서 패스워드를 가져오며, 패스워드가 값이 있는지 체크하고 있음
12:       if (url != null && !"".equals(url) && id != null &&    !"".equals(id) && pwd != null
    &&    !"".equals(pwd))
13:       {
14:         KeyGenerator kgen = KeyGenerator.getInstance("Blowfish");
15:         SecretKey skey = kgen.generateKey();
16:         byte[] raw = skey.getEncoded();
17:         SecretKeySpec skeySpec = new SecretKeySpec(raw, "Blowfish");
18:
19:         Cipher cipher = Cipher.getInstance("Blowfish");
20:         cipher.init(Cipher.DECRYPT_MODE, skeySpec);
21:         byte[] decrypted_pwd = cipher.doFinal(pwd.getBytes());
22:         pwd = new String(decrypted_pwd);
23:         conn = DriverManager.getConnection(url, id, pwd);
24:       }
25:     }
26:     catch (SQLException e)
27:     {
28:       ......
```

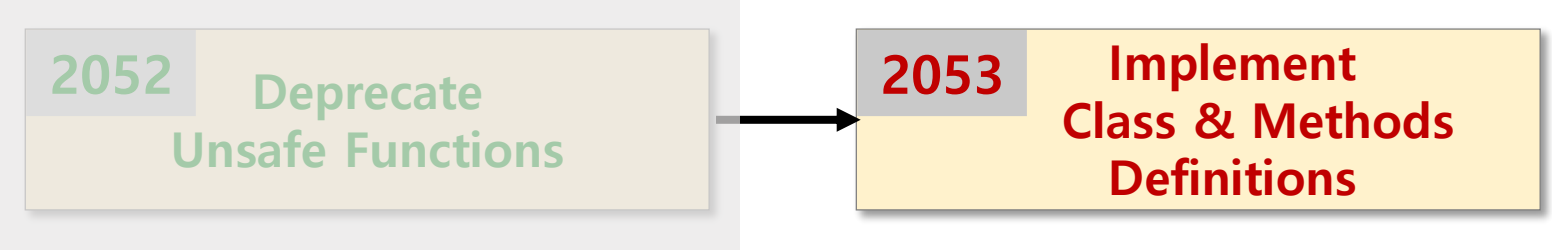
[Measure]

- Passwords hassing
- Strong user passwords

Activity 2052. Deprecate Unsafe Functions

- **Steps**
  1. Analyze functions and APIs
  2. List and prohibit unsafe functions and APIs
  3. Use header files, newer compilers, or code scanning tools to check code (including legacy code where appropriate) for the existence of banned functions
  4. Replace those banned functions with safer alternatives:
     - New native C and C ++ code should not use banned versions of string buffer handling functions.
       (Check the "Setup check-in policies" task for information on how to ensure this.)
     - Sections marked as shared in shipping binaries represent a security threat.
       (Use properly secured dynamically created shared memory objects instead.)
     - Ensure that the application domain group is granted only execute permissions on your stored procedures.
       (∴Do not grant any other permission on your database to any other user or group.)
     - All web applications accessing databases should always use stored procedures.
       (Do not use "exec @ sql" construct in your stored procedures.)
  5. And perform secure coding

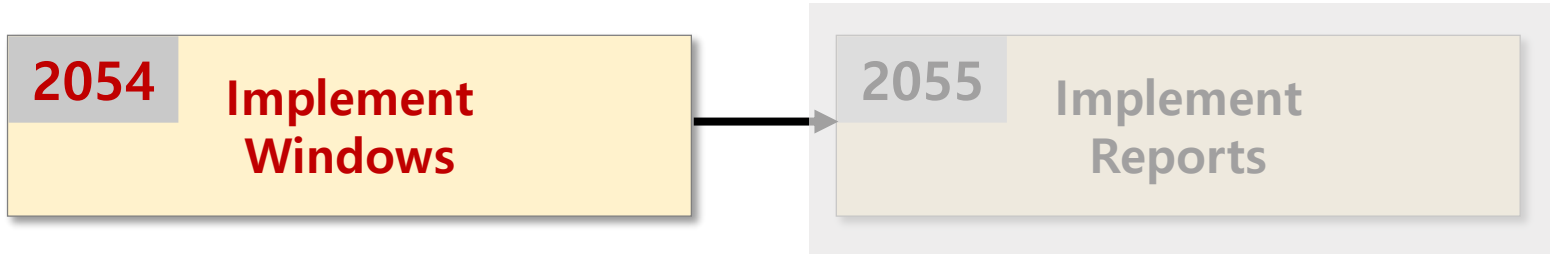
Activity 2053.
Implement Class & Methods Definitions



- “

- **Description**
  - Input : safer alternatives

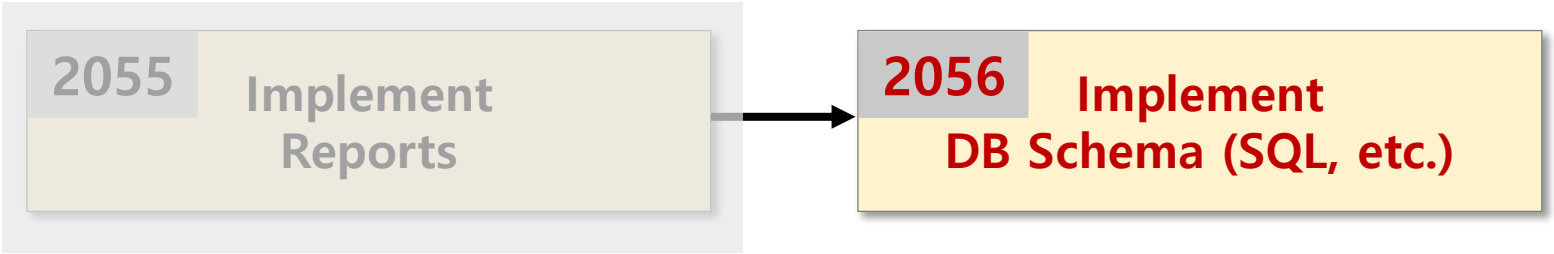
Activity 2054.
Implement Windows



- “

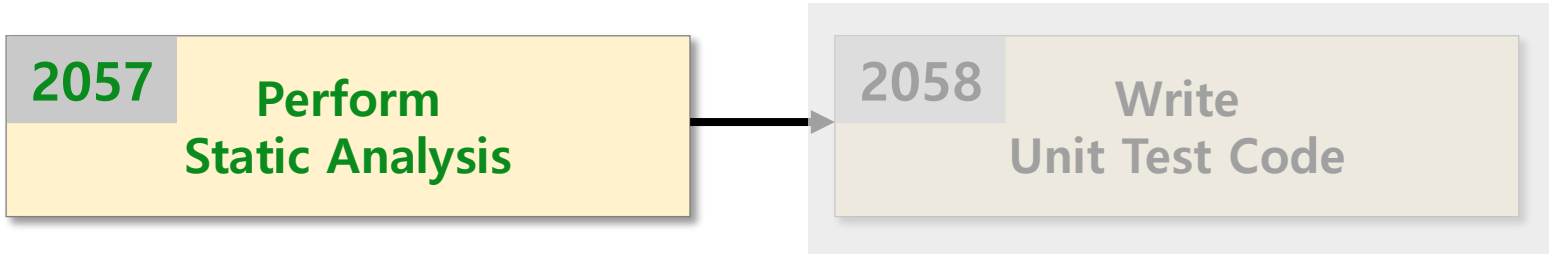
- **Description**
  - Input : safer alternatives

Activity 2056.
Implement DB Schema (SQL, etc.)



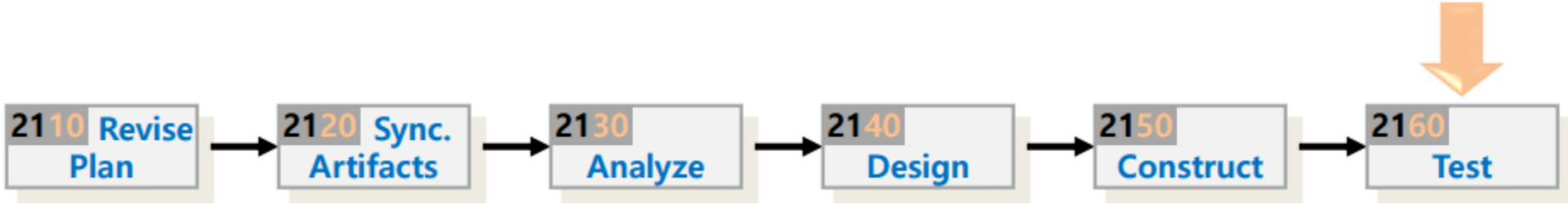
- **Description**
  - Consider security risk such as SQL injection (cf. 2052)
  - Input : safer alternatives

Activity 2057.
Perform Static Code Analysis for Security



- **Description**
  - Static analysis of source code provides a scalable capability for security code review and can help ensure that secure coding policies are being followed
  - With human review or static code analysis tools
  - Input : implements results
  - Output : static code analysis results

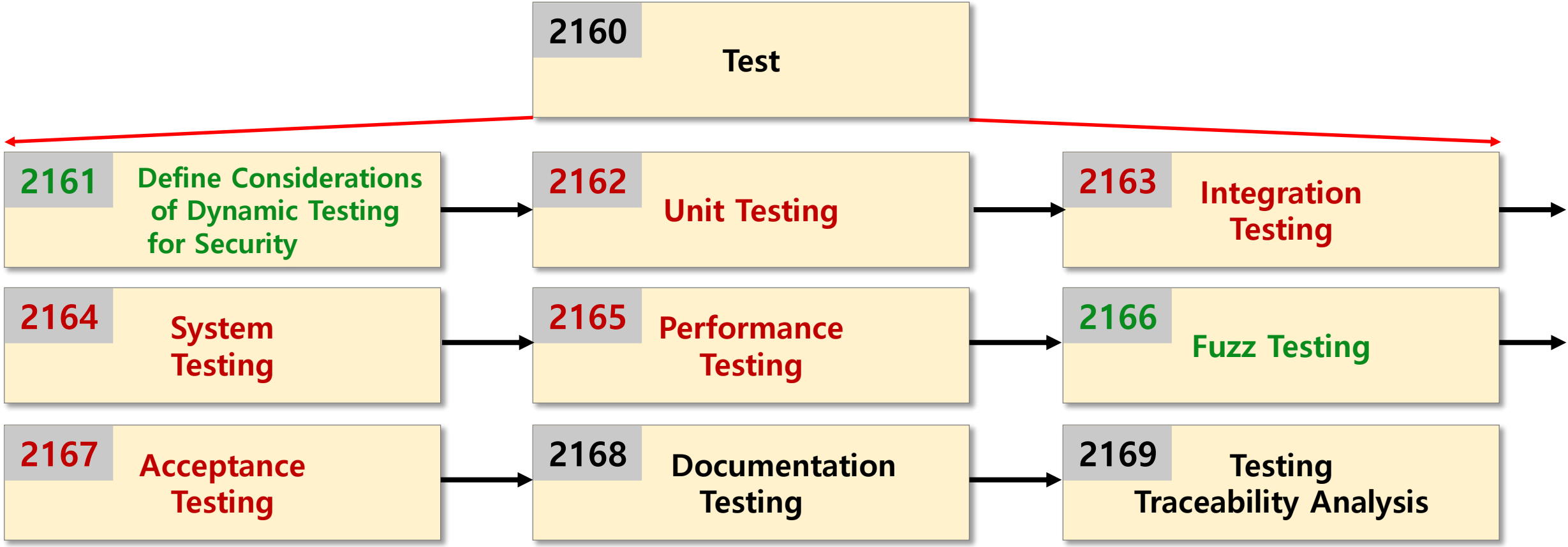
Phase 2060.
Test



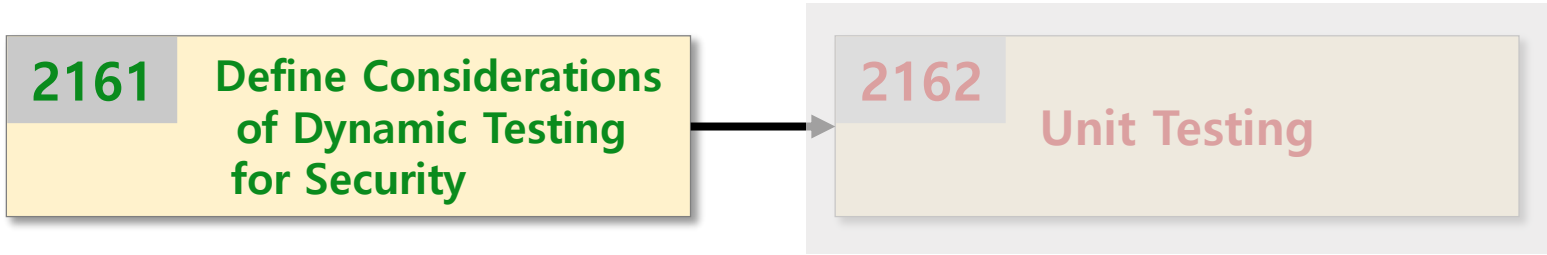
Phase 2060. Test

- Sec+ Phase 2060 Activities

- Addition of activities
- Addition of contents

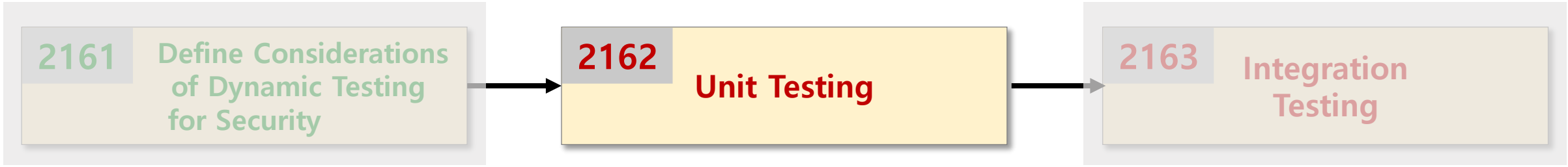


Activity 2061.
Define Considerations of Dynamic Testing for Security



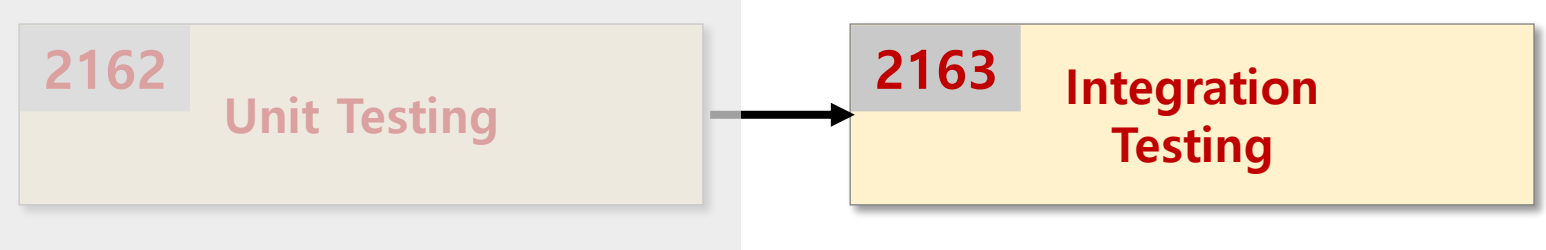
- **Description**
  - Define considerations of the following activities for testing
  - With dynamic testing tools
  - Implementation tools can be found at:
    - ✓Bounds checker : Memory error detection for Windows based applications.
    - ✓Cenzic : publishes a line of dynamic application security tools that scans web applications for security vulnerabilities.
    - ✓ClearSQL : is a review and quality control and a code illustration tool for PL/SQL.
    - ✓Domalloc : library for checking memory allocation and leaks. Software must be recompiled, and all files must include the special C header file dmalloc.h.
  - Input : implements results
  - Output : considerations of dynamic testing

Activity 2062.
Unit Testing



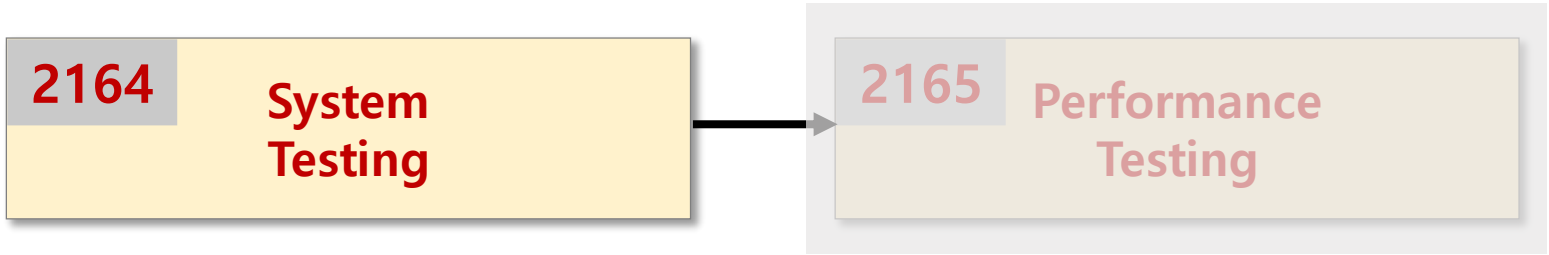
- **Description**
  - unit testing is a software testing method by which individual units of source code, sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures, are tested to determine whether they are fit for use.
  - Perform unit testing and identify the results of testing
  - Perform with considerations of 2061
  - Input : unit test code, implement results, considerations of dynamic testing
  - Output : Unit testing results, reports

Activity 2063.
Integration Testing



- **Description**
  - Integration testing is the phase in software testing in which individual software modules are combined and tested as a group
  - Perform with considerations of 2061
  - Input : class & method definitions, considerations of dynamic testing
  - Output : Integration testing results, reports

Activity 2064.
System Testing

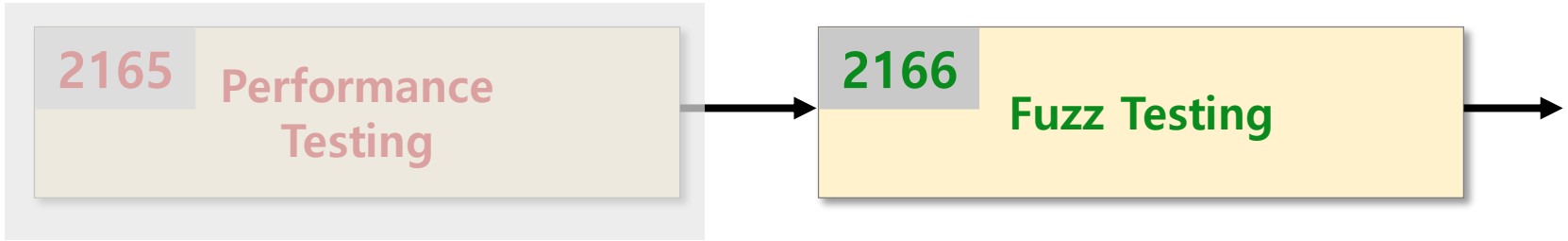


- **Description**
  - System testing of software or hardware is testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements
  - Perform with considerations of 2061
  - Input : implements results, system test plan and cases, considerations of dynamic testing
  - Output : System testing results, reports

Activity 2065.
Performance Testing

- **Description**
  - Performance testing is in general, a testing practice performed to determine how a system performs in terms of responsiveness and stability under a particular workload.
  - Perform with considerations of 2061
  - Input : Implements result (program), considerations of dynamic testing
  - Output : Performance testing results, reports

Activity 2066.
Fuzz Testing



- **Description**
  - A specialized form of dynamic testing used to induce program failure by deliberately introducing malformed or random data to an application
  - Perform with considerations of 2061
  - Input : Implements result (program), considerations of dynamic testing
  - Output : fuzz testing results, reports

Activity 2066. Fuzz Testing

- **What is a fuzz testing?**
  - By entering random data into the software, it means to detect the security weakness of the software by inducing the systematic failure of the software.

  - Useful tools:
    - SDL Regex Fuzzer: A verification tool to help test regular expressions for potential denial of service vulnerabilities.
    - MiniFuzz: A basic testing tool designed to help detect code flaws that may expose security vulnerabilities in file-handling code.

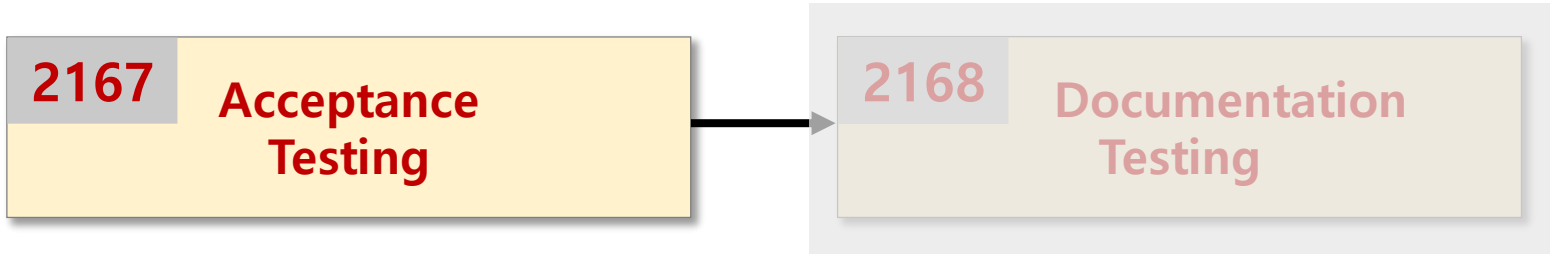
Activity 2066. Fuzz Testing

- Vulnerabilities can be detected :
  - ✓ Buffer overflow
  - ✓ Integer overflow
  - ✓ Format string bug
  - ✓ Race condition
  - ✓ SQL injection
  - ✓ XSS
  - ✓ Remote command execution

- Example of Fuzz testing :

The following code can occur overflow
int size = red_ccr_size(packet); buffer = (char*) malloc(size + 1);

Activity 2067.
Acceptance Testing



- **Description**
  - Acceptance testing is a test conducted to determine if the requirements of a specification or contract are met
  - It is used to determine the final acceptance
  - Especially, review requirements for attack surface
    - Re-review threat models and attack surface measurement of a given application
    - All security bugs identified in your project should be reviewed against the security bug bar/quality criteria established for your project to:
      - ✓ ensure you have met the criteria or
      - ✓ understand the potential attack surface associated with any bugs granted exceptions
  - Perform with considerations of 2061

Activity 2067. Acceptance Testing

- Input : requirements specification, system, considerations of dynamic testing
- Output : Acceptance testing results, reports

- Steps
  1. Review that any design or implementation changes to the system have been accounted for
  2. Review any new attack vectors created as a result of the changes
  3. Mitigate those attack vectors

References

- ISO/IEC 31000, 27005, Risk management
- SDL Process Guidance Version 5.2
- SDL Security Bug Bar (Sample), https://msdn.microsoft.com/en-us/library/cc307404.aspx
- Microsoft SDL Threat Modeling, https://www.microsoft.com/en-us/SDL/process/design.aspx
- Basic of Secure Design, Development and Test, https://www.microsoft.com/en-us/SDL/process/verification.aspx
- 소프트웨어 개발보안 가이드 2013.11, https://www.kisa.or.kr/public/laws/laws3_View.jsp?mode=view&p_No=259&b_No=259&d_No=56&ST=T&SV=

References

- IEEE Standard Glossary of Software Engineering Terminology, IEEE Std610.12-1990
- IEEE Recommended Practice for Software Requirements Specifications, IEEE Std. 830-1998
- Microsoft : Security development lifecycle, https://www.microsoft.com/en-us/sdl/
- Wiki, https://www.wikipedia.org/
- OWASP top 10 & Secure coding, https://www.owasp.org
- Secure Coding Practices Quick Reference Guide, https://www.owasp.org/images/0/08/OWASP_SCP_Quick_Reference_Guide_v2.pdf

Thank you ☺