Scuola universitaria professionale della Svizzera italiana
Dipartimento tecnologie innovative

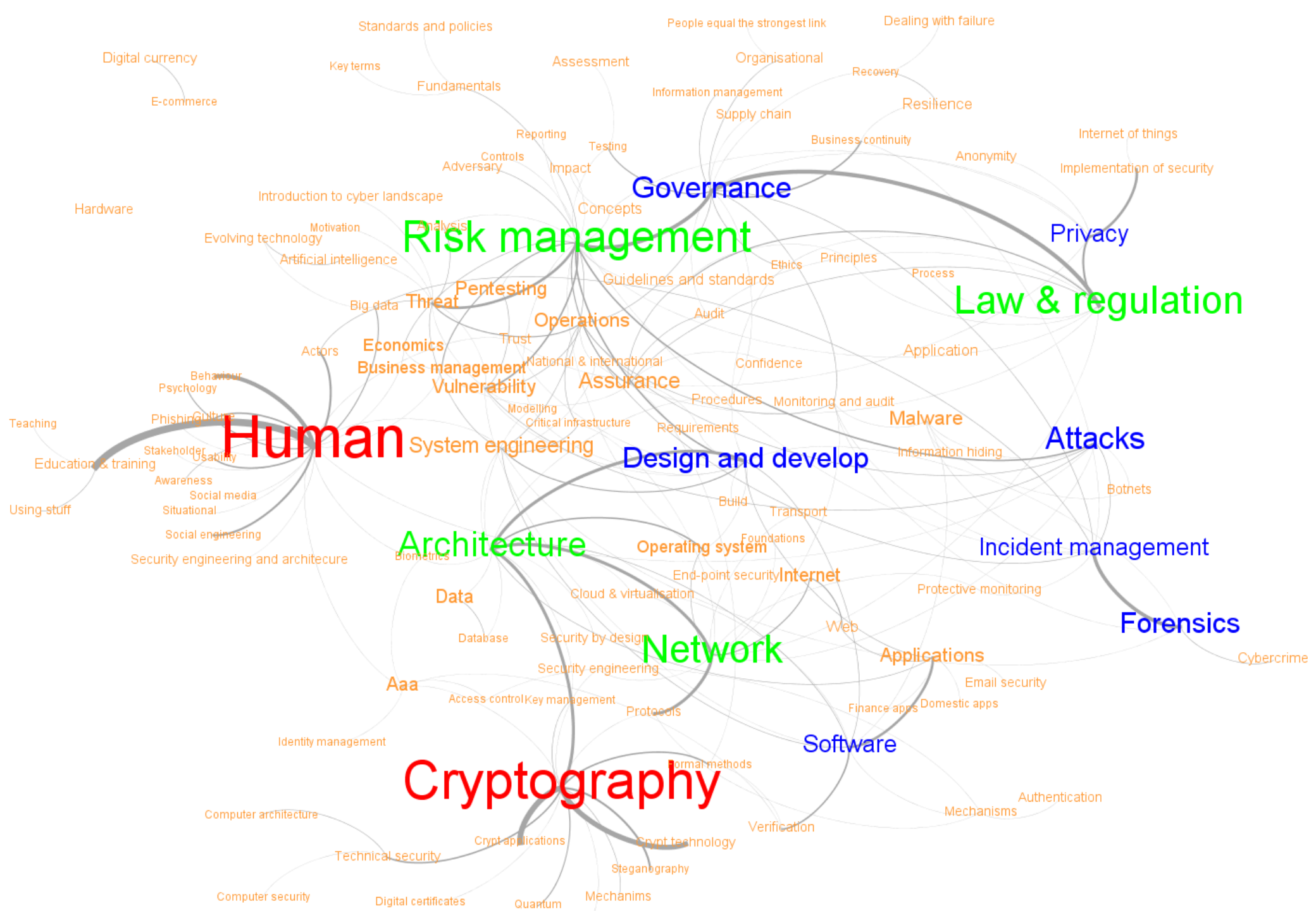
SUPSI

Security by Design

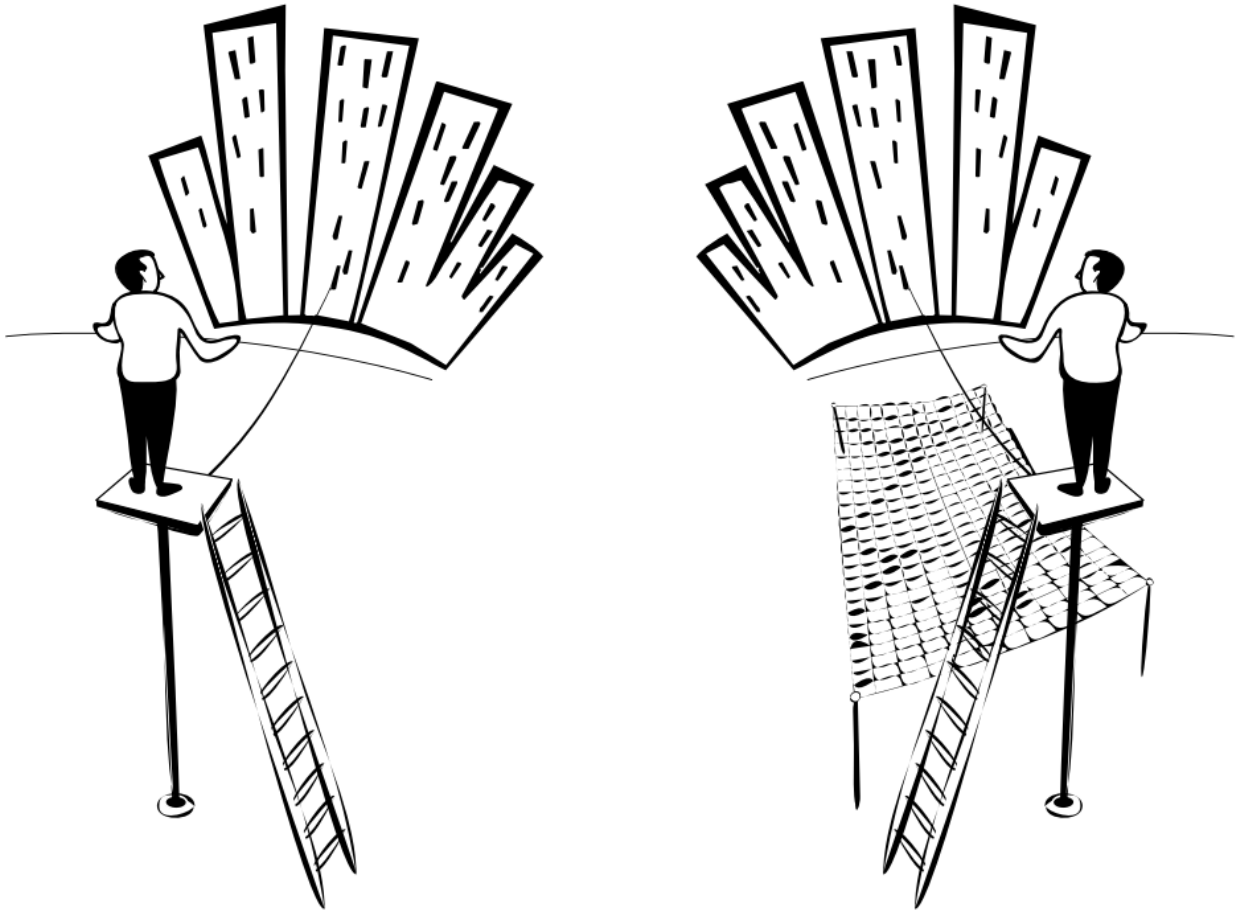
Introduction

SUPSI DTI
2025 - 2026

Angelo Consoli


Standards and policies
People equal the strongest link
Dealing with failure
Digital currency
Key terms
Assessment
Organisational
Recovery
Fundamentals
Information management
E-commerce
Resilience
Supply chain
Reporting
Testing
Business continuity
Internet of things
Controls
Adversary
Impact
Anonymity
Implementation of security
Governance
Introduction to cyber landscape
Hardware
Concepts
Motivation
Analysis
Risk management
Privacy
Evolving technology
Artificial intelligence
Ethics
Principles
Process
Guidelines and standards
Pentesting
Law & regulation
Big data
Threat
Audit
Operations
Trust
Economics
Application
Actors
National & international
Confidence
Business management
Behaviour
Psychology
Vulnerability
Assurance
Procedures
Monitoring and audit
Modelling
Malware
Phishing
Critical infrastructure
Teaching
Requirements
Human
System engineering
Attacks
Stakeholder
Usability
Information hiding
Education & training
Design and develop
Awareness
Botnets
Social media
Build
Using stuff
Situational
Transport
Social engineering
Foundations
Architecture
Operating system
Incident management
Security engineering and architecure
Biometrics
End-point security
Internet
Protective monitoring
Data
Cloud & virtualization
Forensics
Web
Database
Security by design
Network
Applications
Cybercrime
Security engineering
Email security
Aaa
Access control
Key management
Domestic apps
Protocols
Finance apps
Identity management
Software
Formal methods
Cryptography
Authentication
Mechanisms
Computer architecture
Verification
Crypt applications
Crypt technology
Technical security
Steganography
Computer security
Digital certificates
Quantum
Mechanims

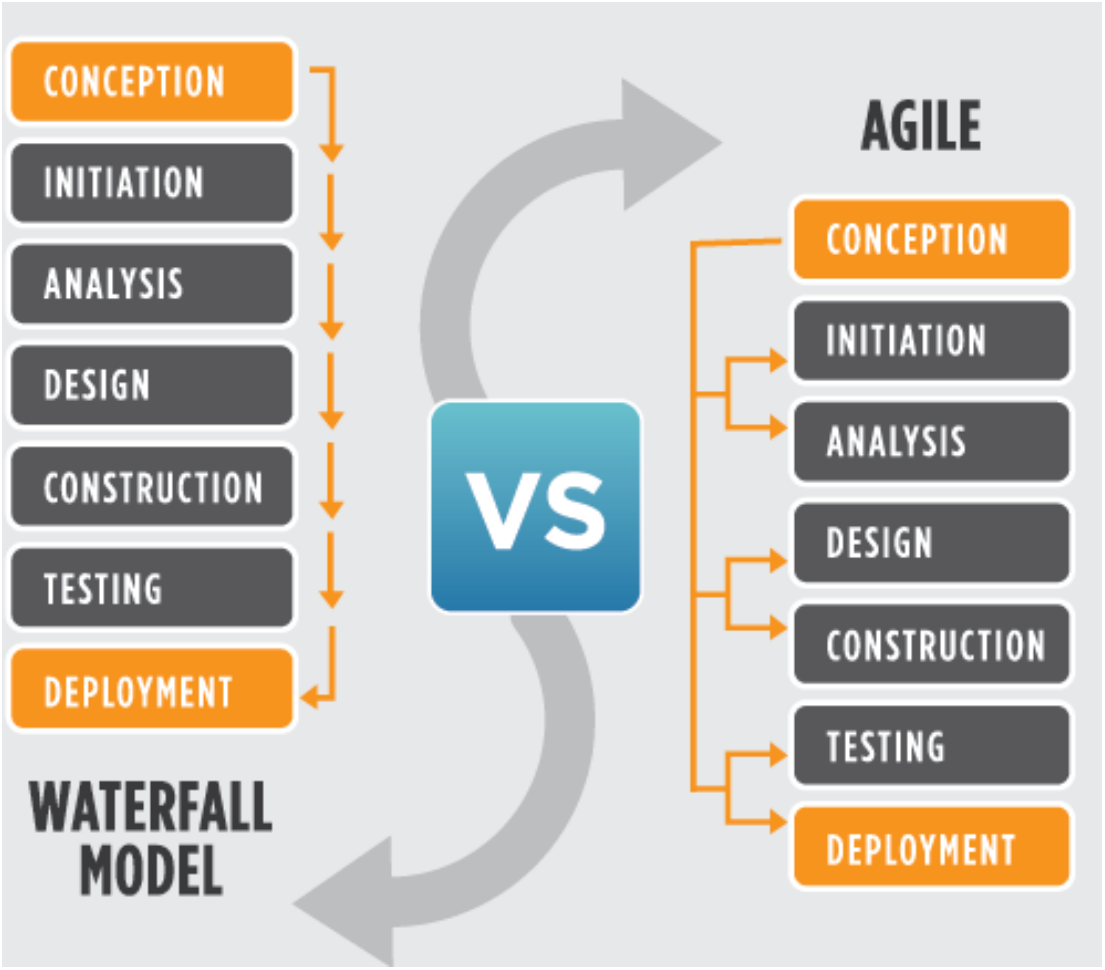
Security-by-Design is an approach to software and hardware development that seeks to **minimise systems vulnerabilities** and **reduce** the **attack surface** through designing and building security in every phase of the **SDLC** (Systems Development Lifecycle).



Developing software without security in mind is like walking a high wire without a net

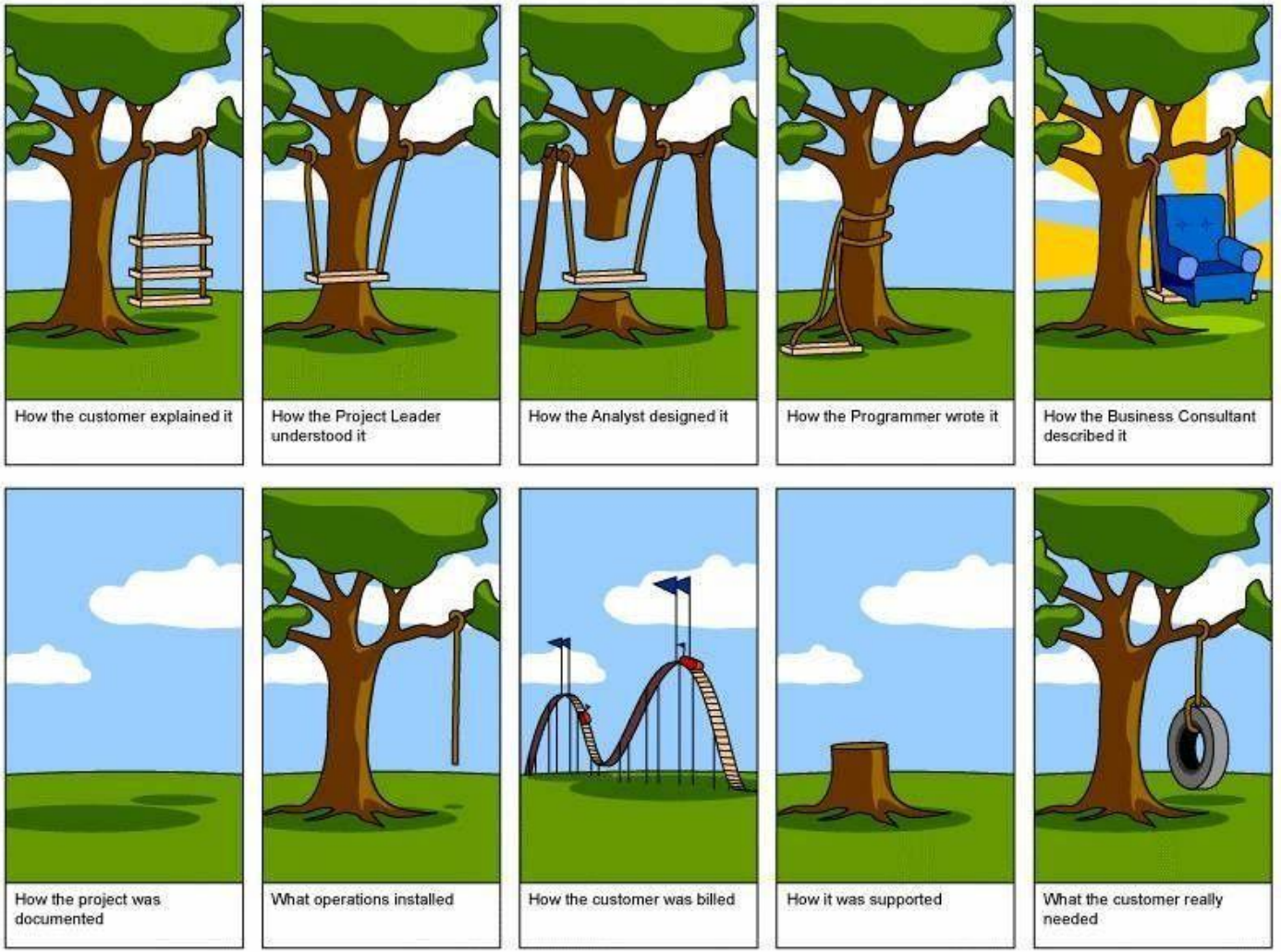
SDLC
Systems Development Lifecycle

www.synotive.com

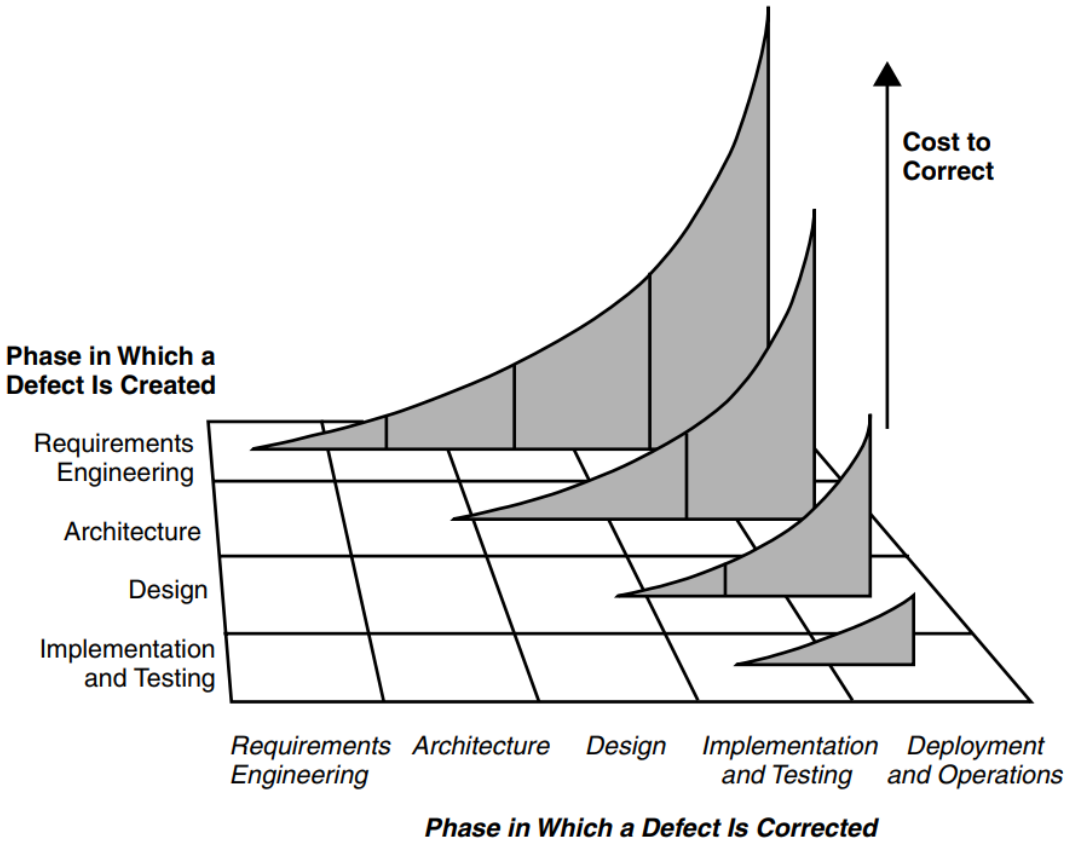


www.seguetech.com

SDLC
Systems Development Lifecycle



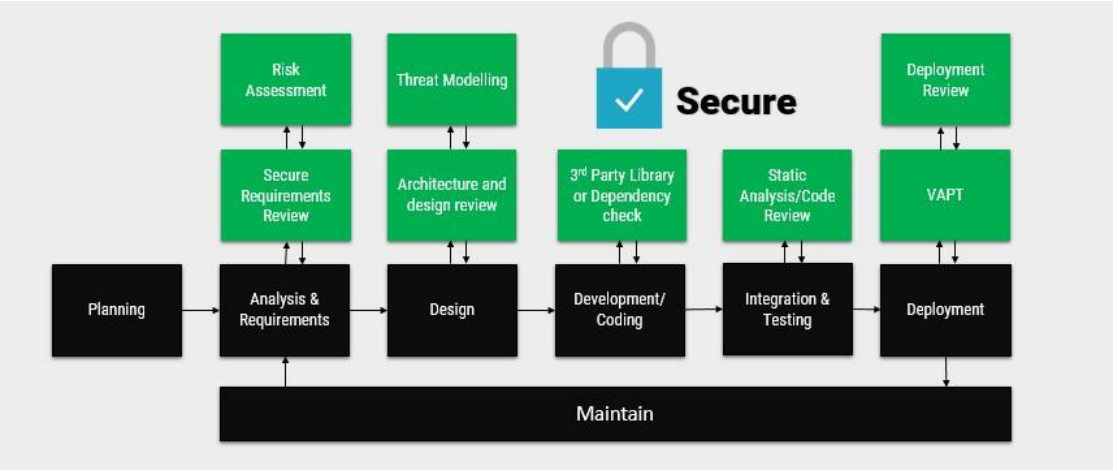
Cost of Error correction



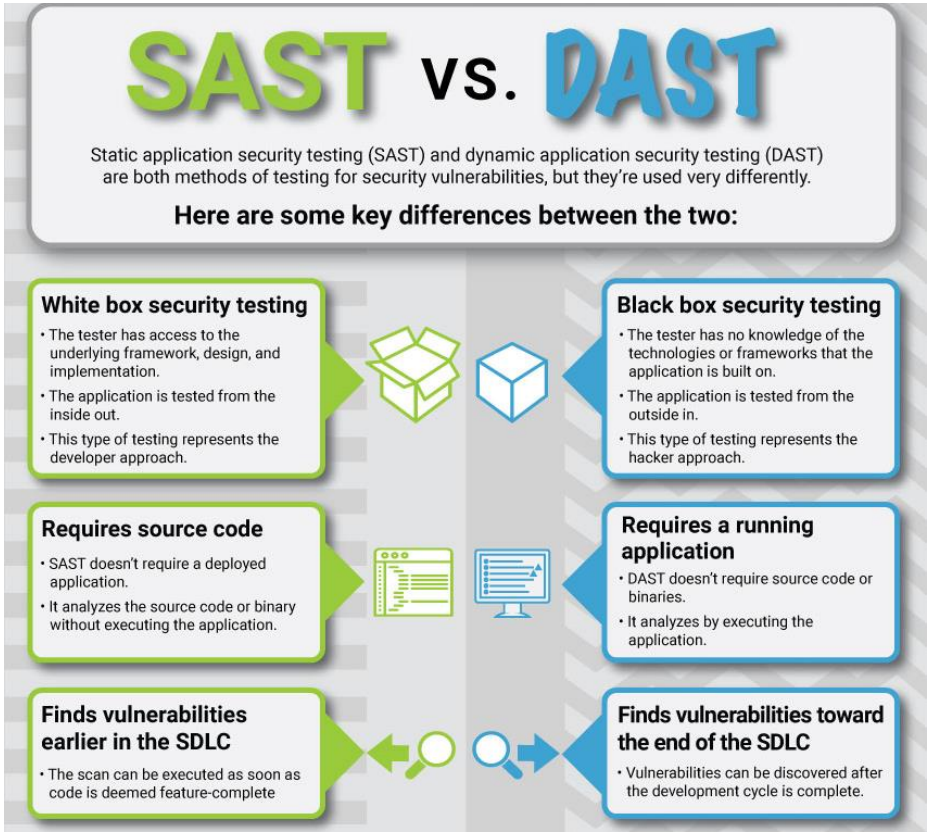
Cost of correcting defects by life-cycle phase

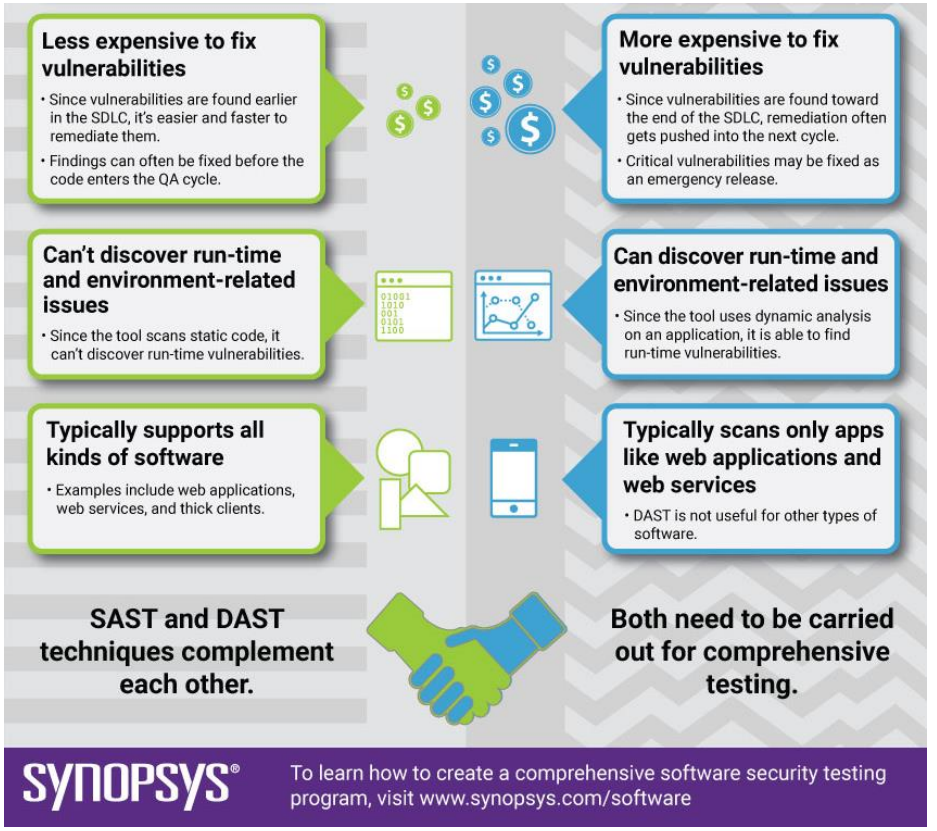
SAST & DAST

- SAST Static Application Security Testing
- DAST Dynamic Application Security Testing
- Technique used for securing software by reviewing the source code
- Not enough …



SAST & DAST


SAST vs. DAST
Static application security testing (SAST) and dynamic application security testing (DAST) are both methods of testing for security vulnerabilities, but they're used very differently.
Here are some key differences between the two:
White box security testing
• The tester has access to the underlying framework, design, and implementation.
• The application is tested from the inside out.
• This type of testing represents the developer approach.
Black box security testing
• The tester has no knowledge of the technologies or frameworks that the application is built on.
• The application is tested from the outside in.
• This type of testing represents the hacker approach.
Requires source code
• SAST doesn't require a deployed application.
• It analyzes the source code or binary without executing the application.
Requires a running application
• DAST doesn't require source code or binaries.
• It analyzes by executing the application.
Finds vulnerabilities earlier in the SDLC
• The scan can be executed as soon as code is deemed feature-complete
Finds vulnerabilities toward the end of the SDLC
• Vulnerabilities can be discovered after the development cycle is complete.


Less expensive to fix vulnerabilities
• Since vulnerabilities are found earlier in the SDLC, it's easier and faster to remediate them.
• Findings can often be fixed before the code enters the QA cycle.
More expensive to fix vulnerabilities
• Since vulnerabilities are found toward the end of the SDLC, remediation often gets pushed into the next cycle.
• Critical vulnerabilities may be fixed as an emergency release.
Can't discover run-time and environment-related issues
• Since the tool scans static code, it can't discover run-time vulnerabilities.
Can discover run-time and environment-related issues
• Since the tool uses dynamic analysis on an application, it is able to find run-time vulnerabilities.
Typically supports all kinds of software
• Examples include web applications, web services, and thick clients.
Typically scans only apps like web applications and web services
• DAST is not useful for other types of software.
SAST and DAST techniques complement each other.
Both need to be carried out for comprehensive testing.
SYNOPSYS®
To learn how to create a comprehensive software security testing program, visit www.synopsys.com/software

ISO/IEC/IEEE 12207 Software life cycle processes

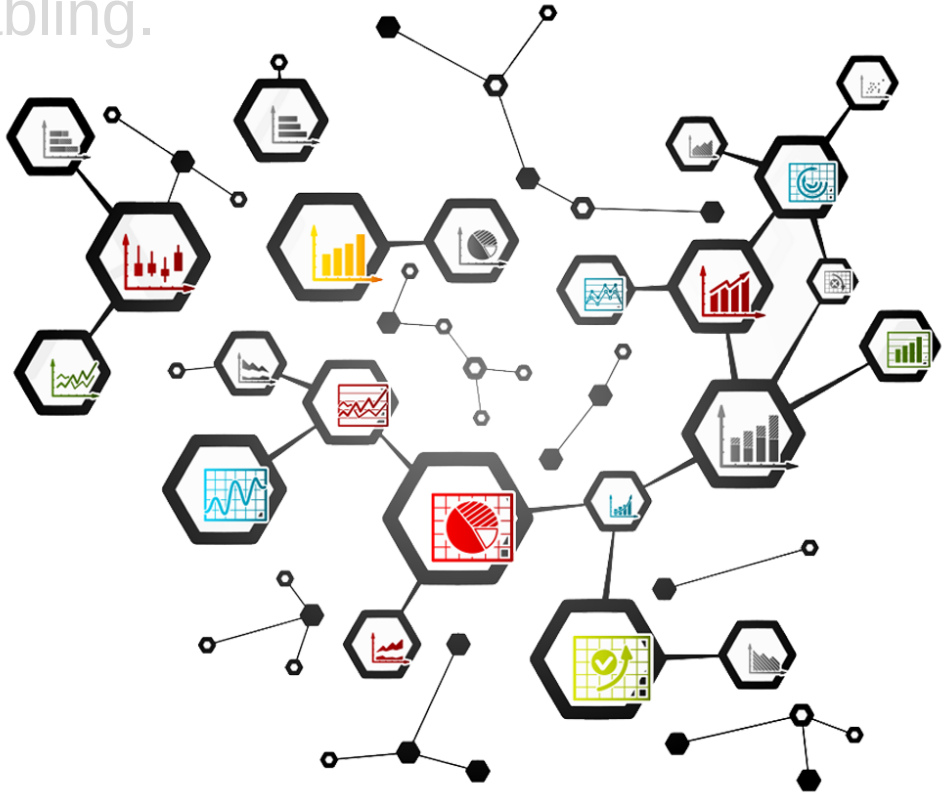
International standard for software lifecycle processes.
Latest version ISO/IEC/IEEE 12207:**2017**

Four main process groups:

- Agreement processes.
- Organizational project-enabling.
- Technical management.
- Technical processes.

ISO/IEC/IEEE 12207
Software life cycle processes

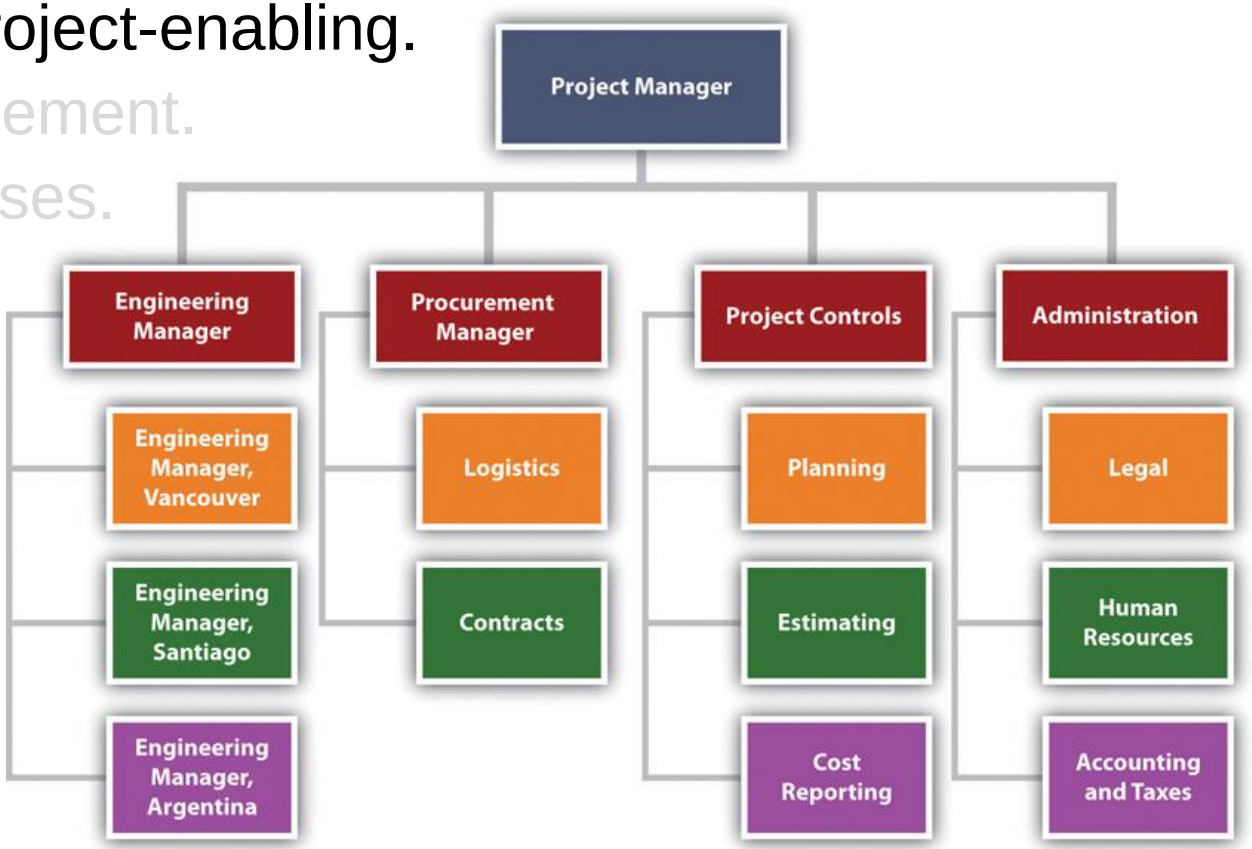
- Agreement processes.
- Organizational project-enabling.
- Technical management.
- Technical processes.



https://www.pngfuel.com

ISO/IEC/IEEE 12207
Software life cycle processes

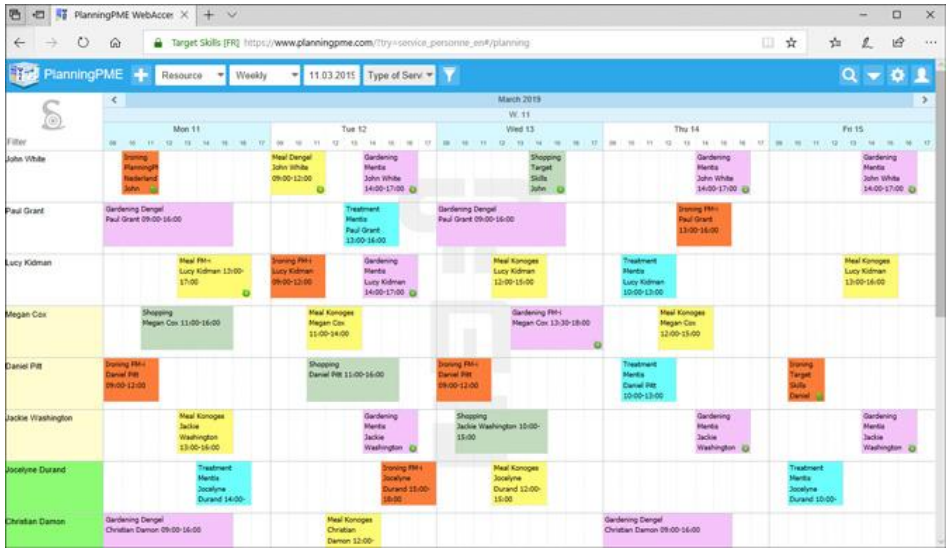
- Agreement processes.
- Organizational project-enabling.
- Technical management.
- Technical processes.



https://saylordotorg.github.io/text_project-management-from-simple-to-complex-v1.1/s05-02-project-organization.html

ISO/IEC/IEEE 12207
Software life cycle processes

- Agreement processes.
- Organizational project-enabling.
- Technical management.
- Technical processes.



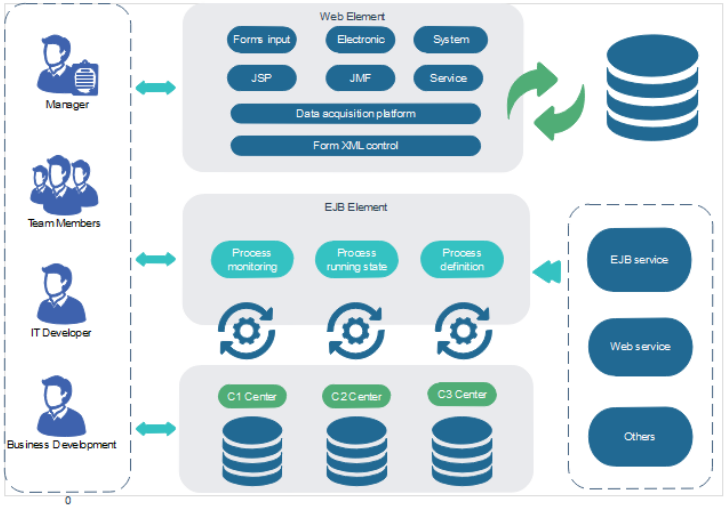
www.planningpme.com

www.thriveglobal.com

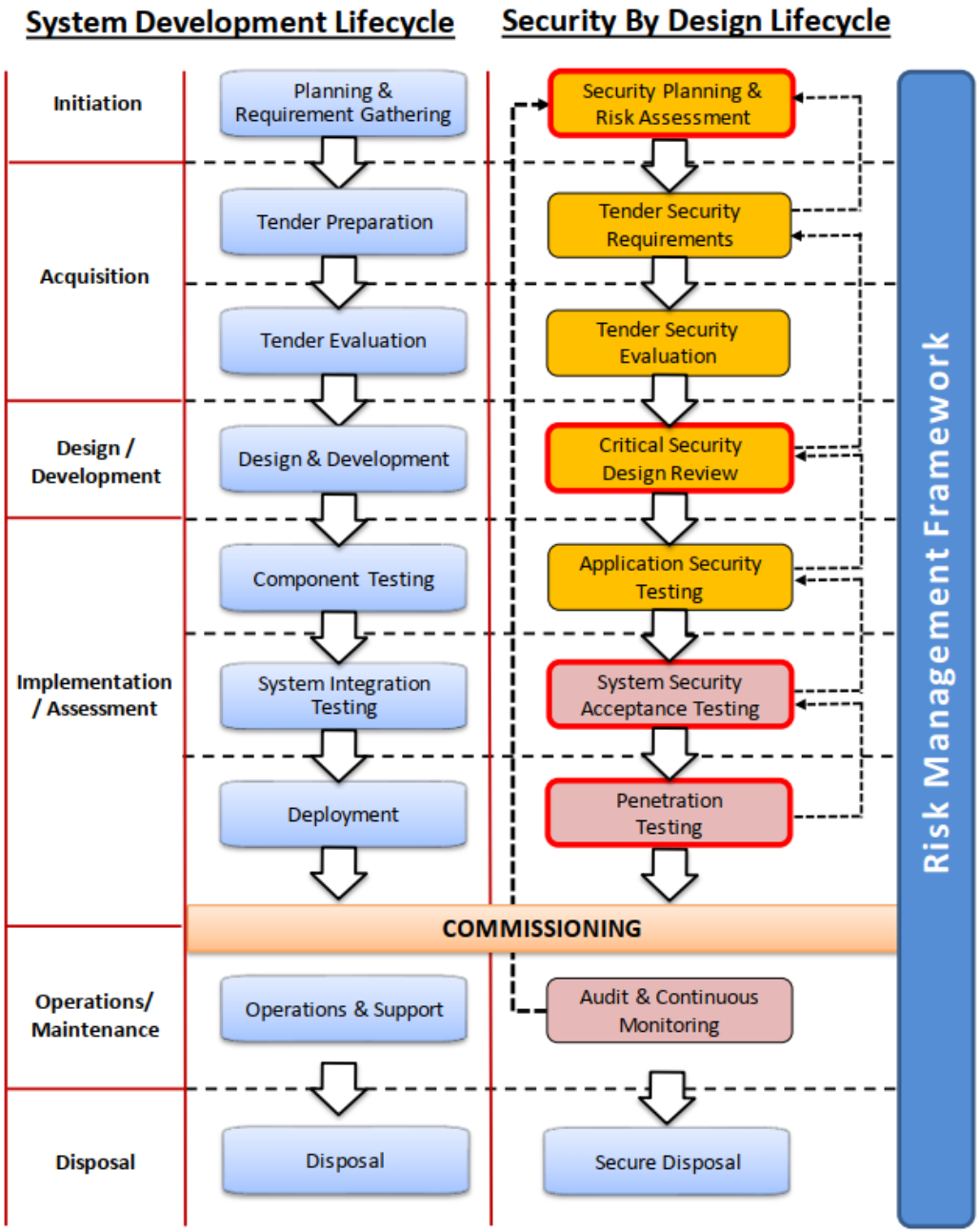
ISO/IEC/IEEE 12207
Software life cycle processes

- Agreement processes.
- Organizational project-enabling.
- Technical management.
- Technical processes.

www.visual-paradigm.com



www.edrawsoft.com

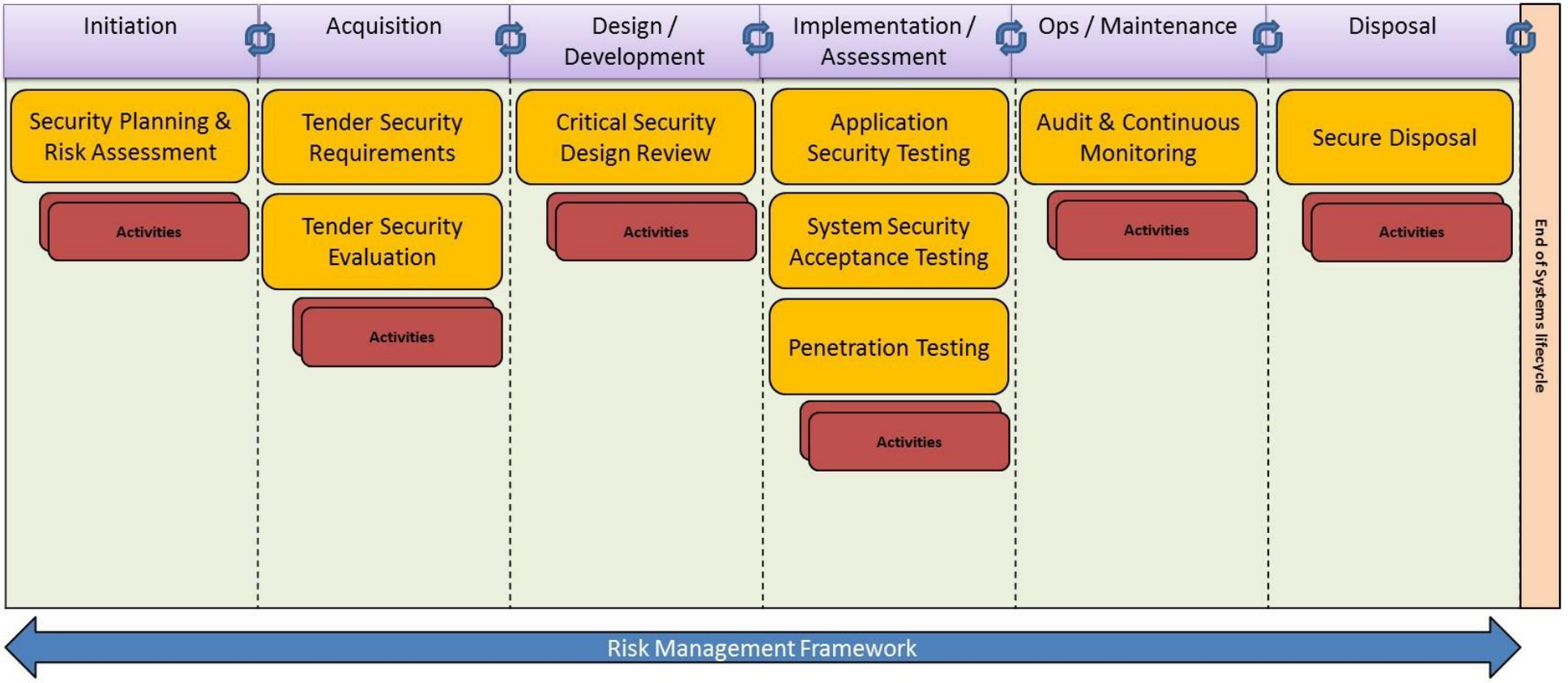


Security-by-Design Framework Pag. 10, CSA Singapore, Version 1.0, 2017

SDLC and Security by Design

- Performed by Project Team
- Performed by Security Officers
- Performed by Independent Third-Party Assessor
- Milestones / Deliverables

Security by Design Framework

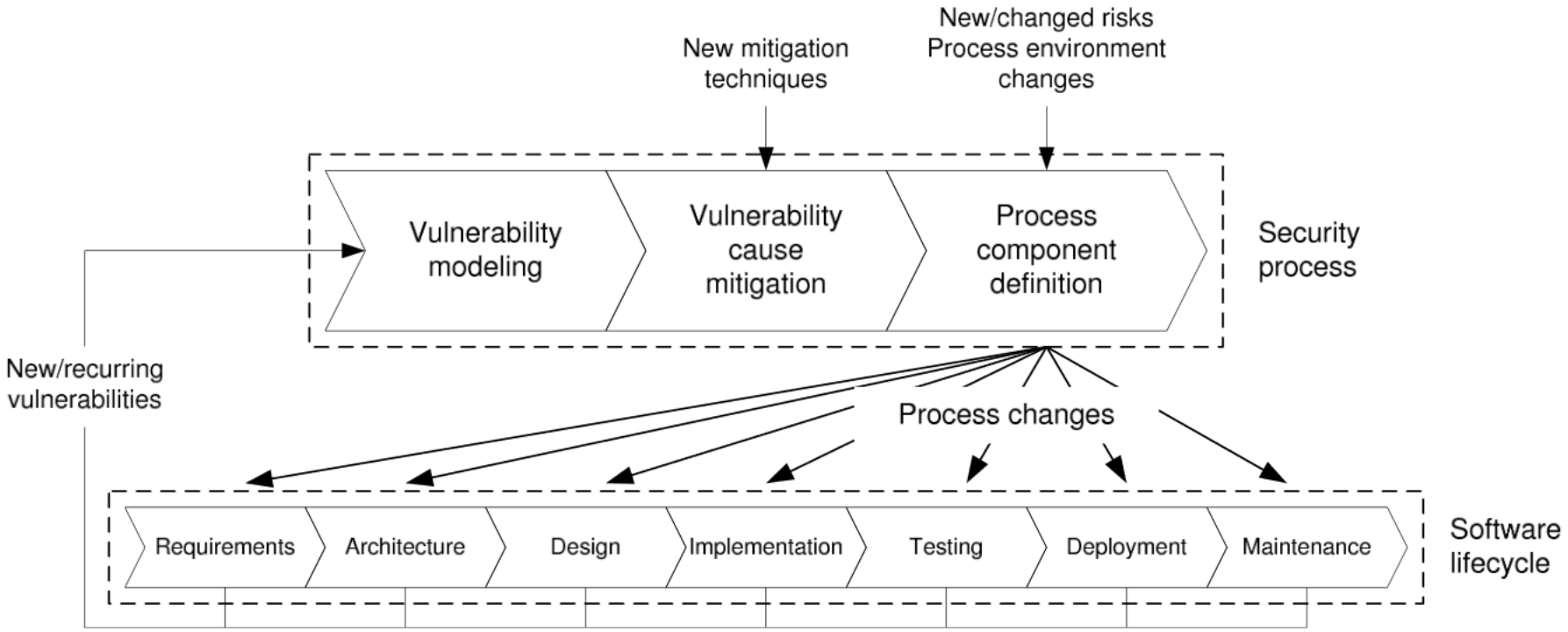


Security-by-Design Framework Pag. 12, CSA Singapore, Version 1.0, 2017

Security by Design Frameworks

- International Organization for Standardization (**ISO**)
- National Institute for Standards & Technology (**NIST**)
- US Government (**HIPAA & FedRAMP**)
- Information Systems Audit and Control Association (**ISACA**)
- Cloud Security Alliance (**CSA**)
- Center for Internet Security (**CIS**)
- Open Web Application Security Project (**OWASP**)

Security by Design Framework



"Design of a Process for Software Security", 2007

Software Security Flaws

Software **security flaws** can be introduced at any stage of the software development lifecycle:

- Not identifying security requirements up front.
- Creating **conceptual designs** that have **logic errors**.
- Using **poor coding practices** that introduce technical vulnerabilities.
- **Deploying** the software improperly.
- Introducing flaws during **maintenance** or **updating**.

Critical Software Components

Critical components of software include:

- The software and its associated information.
- The operating systems of the associated servers.
- The backend database.
- Other applications in a shared environment.
- The user's system.
- Other software that the user interacts with.

Critical Software Components

- Important: Keep the software (also development software) updated.

15 CVE-2019-9025	119	Overflow	2019-02-22	2019-04-17	7.5	None	Remote	Low	Not required	Partial	Partial	Partial

An issue was discovered in PHP 7.3.x before 7.3.1. An invalid multibyte string supplied as an argument to the mb_split() function in ext/mbstring/php_mbregex.c can cause PHP to execute memcpy() with a negative argument, which could read and write past buffers allocated for the data.

16 CVE-2019-9024	125		2019-02-22	2019-06-18	5.0	None	Remote	Low	Not required	Partial	None	None

An issue was discovered in PHP before 5.6.40, 7.x before 7.1.26, 7.2.x before 7.2.14, and 7.3.x before 7.3.1. xmlrpc_decode() can allow a hostile XMLRPC server to cause PHP to read memory outside of allocated areas in base64_decode_xmlrpc in ext/xmlrpc/libxmlrpc/base64.c.

17 CVE-2019-9023	125		2019-02-22	2019-06-18	7.5	None	Remote	Low	Not required	Partial	Partial	Partial

An issue was discovered in PHP before 5.6.40, 7.x before 7.1.26, 7.2.x before 7.2.14, and 7.3.x before 7.3.1. A number of heap-based buffer over-read instances are present in mbstring regular expression functions when supplied with invalid multibyte data. These occur in ext/mbstring/oniguruma/regcomp.c, ext/mbstring/oniguruma/regexec.c, ext/mbstring/oniguruma /regparse.c, ext/mbstring/oniguruma/enc/unicode.c, and ext/mbstring/oniguruma/src/utf32_be.c when a multibyte regular expression pattern contains invalid multibyte sequences.

Software Security Flaws

What to do?

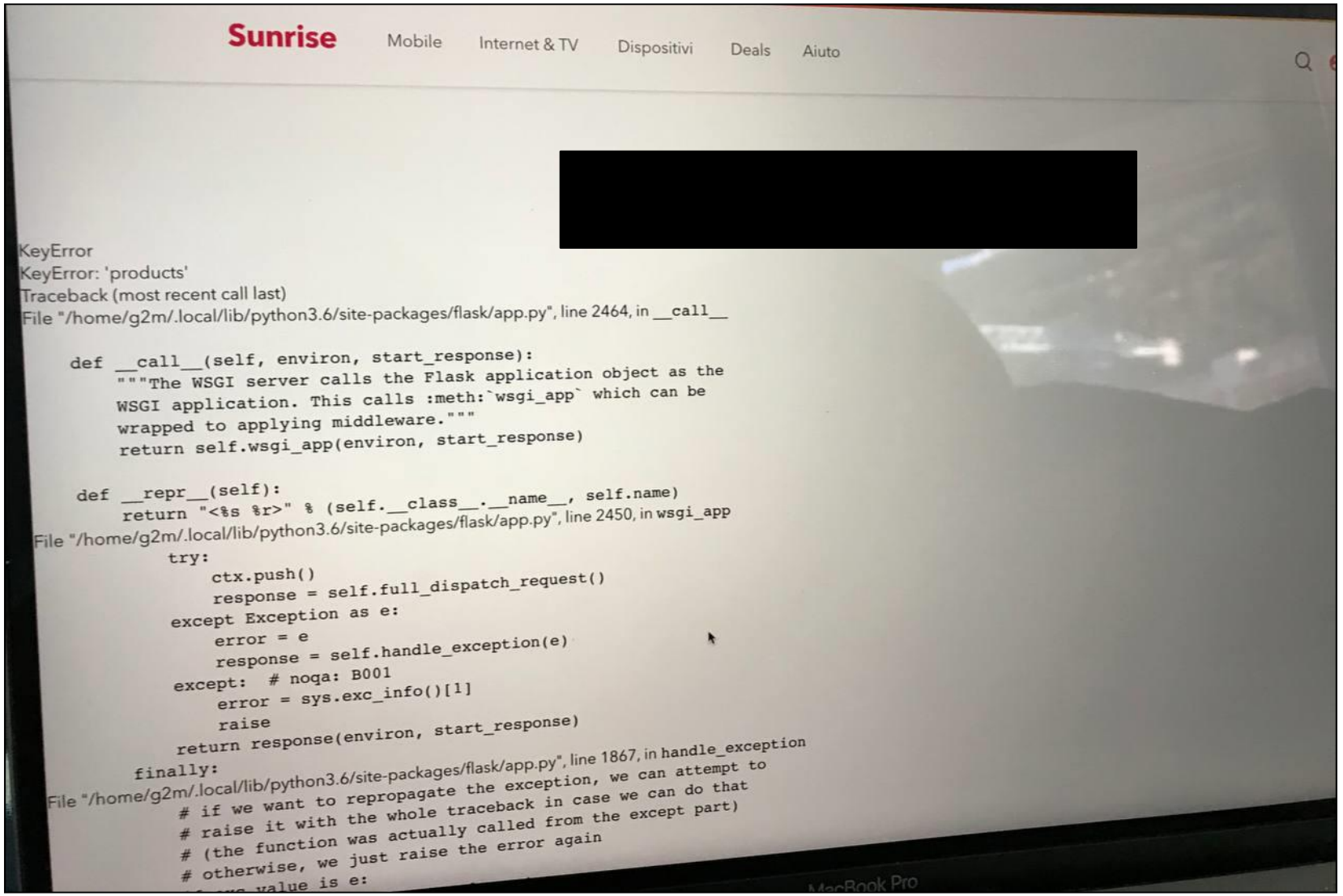
- Software and Hardware hardening.
- Follow strict procedures principles (OWASP, NIST derivate, …).
- … Use logic.

```java
public boolean isNegative(int number) {
    String numberString = number + "";
    if (numberString.charAt(0) == '-') {
        return true;
    }
    else {
        return false;
    }
```

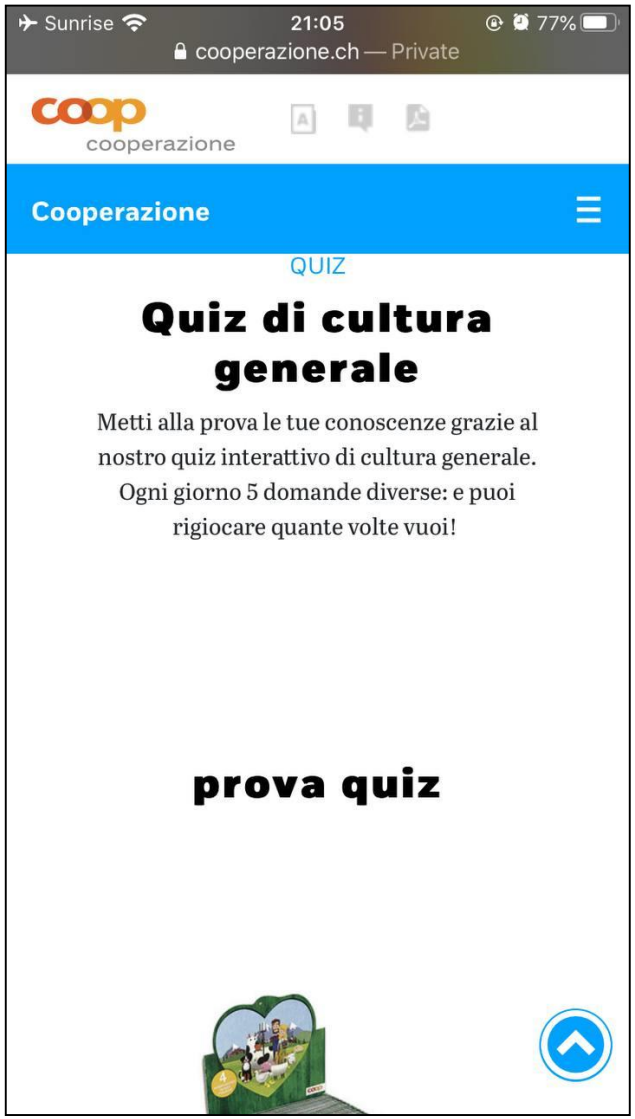
What not to do: Write bad code …

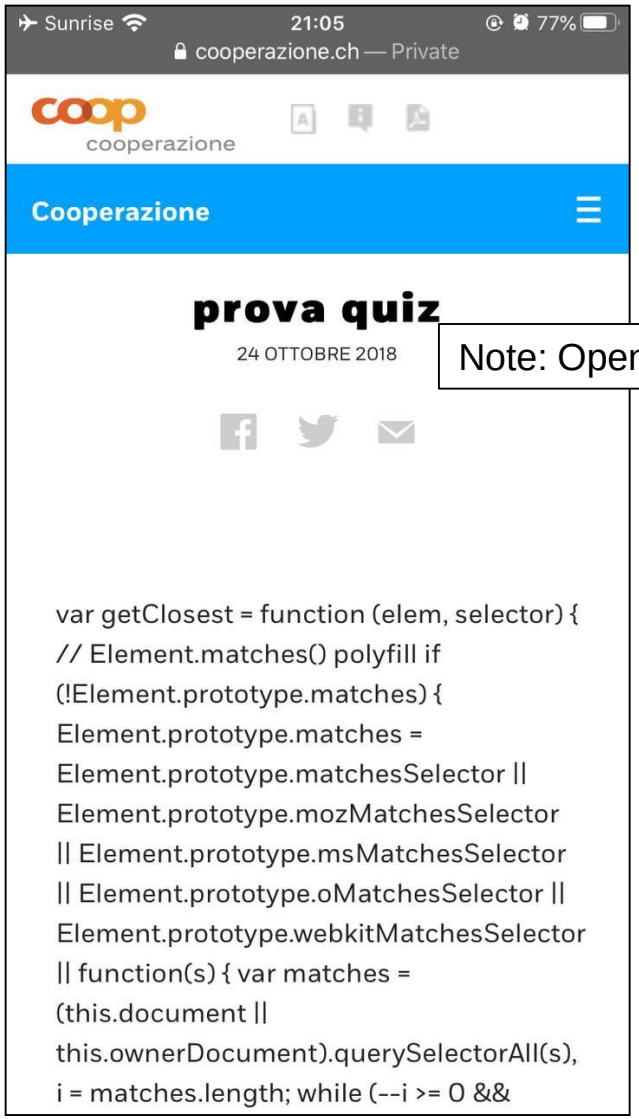
```
optimisedRandomNumber()
{
    return 7; // Chosen at random after a vote by the dev team.
}
```

Software Security Flaws



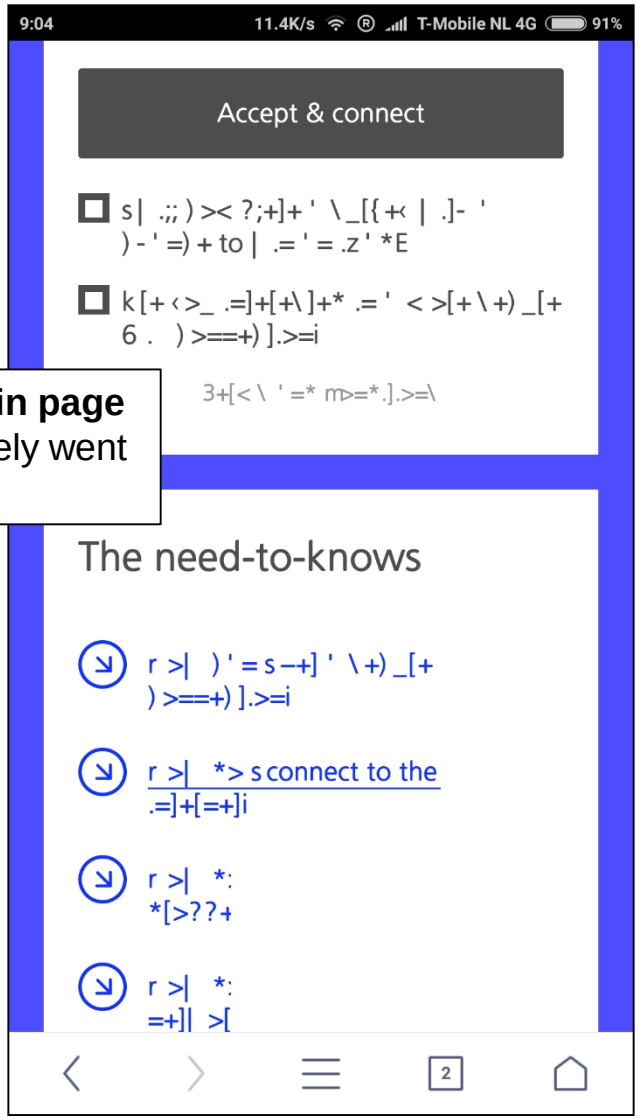
Software Security Flaws



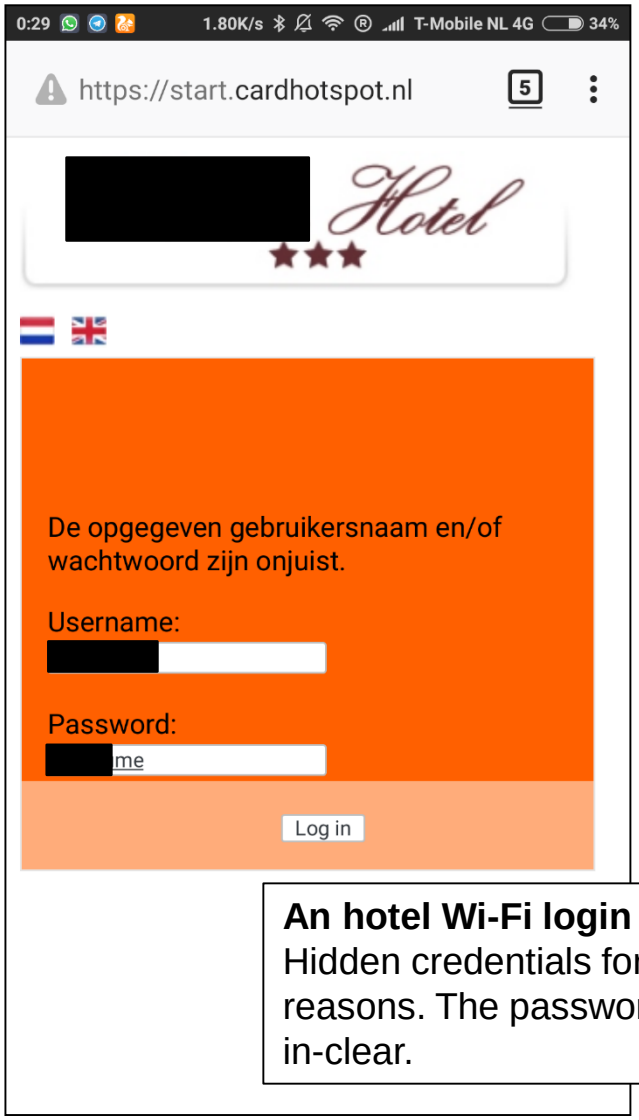


Note: Opened August 2021 ….

Software Security Flaws

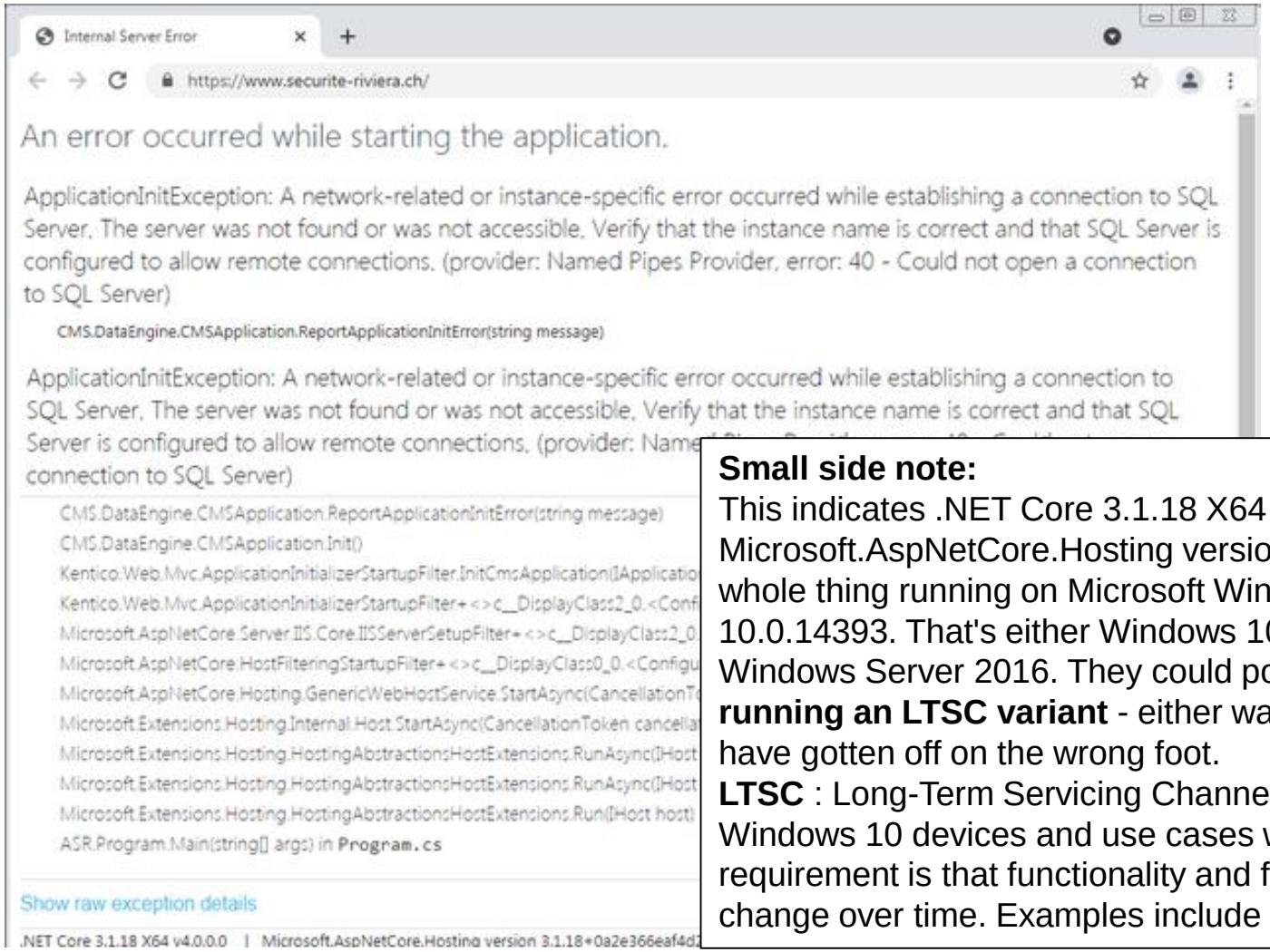


Airport Wi-Fi login page
Something definitely went wrong…



An hotel Wi-Fi login page
Hidden credentials for obvious reasons. The password field is totally in-clear.

2021 Cyberattack Montreaux



Small side note:
This indicates .NET Core 3.1.18 X64 v4.0.0.0, Microsoft.AspNetCore.Hosting version 3.1.18, with the whole thing running on Microsoft Windows 10 build 10.0.14393. That's either Windows 10 version 1607 or Windows Server 2016. They could potentially be **running an LTSC variant** - either way they seems to have gotten off on the wrong foot.
LTSC : Long-Term Servicing Channel is designed for Windows 10 devices and use cases where the key requirement is that functionality and features don't change over time. Examples include medical systems.

Source: https://borncity.com/win/2021/10/11/schweizer-gemeinde-montreux-opfer-eines-cyberangriffs

IEEE Software Life Cycle

- **SQA** - Software quality assurance IEEE 730
- **SCM** - Software configuration management IEEE 828
- **STD** - Software test documentation IEEE 829
- **SRS** - Software requirements specification IEEE 830
- **V&V** - Software verification and validation IEEE 1012
- **SDD** - Software design description IEEE 1016
- **SPM** - Software project management IEEE 1058
- **SUD** - Software user documentation IEEE 1063

https://en.wikipedia.org/wiki/Software_verification_and_validation

IEEE Software Life Cycle

- **SQA** - Software quality assurance IEEE 730
- **SCM** - Software configuration management IEEE 828
- **STD** - Software test documentation IEEE 829
- **SRS** - Software requirements specification IEEE 830
- **V&V** - Software verification and validation IEEE 1012
- **SDD** - Software design description IEEE 1016
- **SPM** - Software project management IEEE 1058
- **SUD** - Software user documentation IEEE 1063

IEEE Software Life Cycle

Software verification and validation IEEE 1012

- Verification: Are we building the product right?
- Validation: Are we building the right product?

- High-level checking.

OWASP Secure Design Principles

OWASP : Open Web Application Security Project

Publications:

- OWASP Top Ten (Latest versions 2013 and 2017).
- OWASP Software Assurance Maturity Model.
- OWASP Development Guide.
- OWASP Web Security & Mobile Testing Guides.
- OWASP Secure Design Principles.
- …

https://blog.threatpress.com/security-design-principles-owasp/

OWASP Secure Design Principles

Some aspects to consider:

- Asset clarification.
- Understand attackers.
- Core pillars of information security.
- Security architecture.

OWASP Secure Design Principles

Asset clarification:

Identify and classify the data that the application will handle. OWASP suggests that programmers create security controls that are appropriate for the value of the data being managed. For example, an application processing financial information must have much tighter restrictions than a blog or web forum.

OWASP Secure Design Principles

Understanding attackers:

Both insider and outsider:

- Disgruntled staff members and programmers.
- Drive-by attacks that release viruses or Trojan attacks onto the system.
- Motivated cybercriminals.
- Criminal organisations with malicious intent.
- Script kiddies.

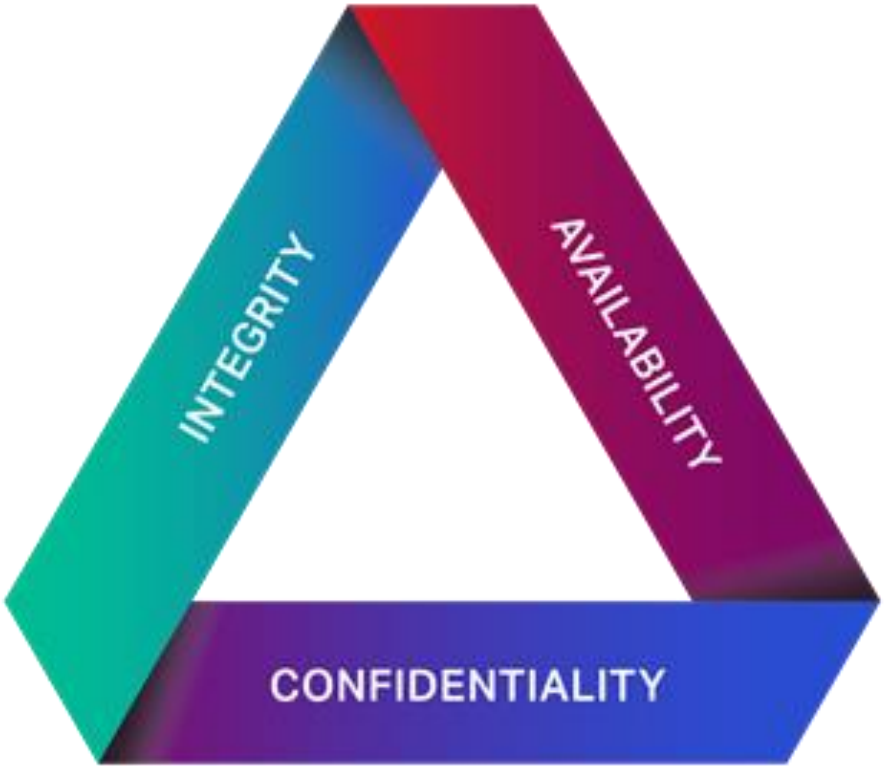
OWASP Secure Design Principles

Core pillars of information security:

OWASP recommends that all security controls should be designed with the core pillars of information security in mind:

- **Confidentiality**: only allow access to data for which the user is permitted
- **Integrity**: ensure data is not tampered or altered by unauthorised users
- **Availability**: ensure systems and data are available to authorised users when they need it

OWASP Secure Design Principles



https://www.ciena.com/insights/articles/Following-the-3-pillar-approach-to-effective-security-strategy.html

OWASP Secure Design Principles

Security architecture:

Every application should have security measures designed to cover all kinds of risks, ranging from typical usage risks (accidental data erasure) through to extreme attacks (brute force attacks, injection attacks etc.).

Developers should consider each feature on the application they are designing and ask the following questions:

- Is the process surrounding this feature as safe as possible? In other words, is this a flawed process?
- If I were evil, how would I abuse this feature?
- Is the feature required to be on by default? If so, are there limits or options that could help reduce the risk from this feature?

By “thinking evil” developers can identify the ways that cybercriminals and malicious individuals might seek to attack a web application.

OWASP Secure Design Principles

- Defense in Depth
- Least Privilege
- Separation of Duties
- Economy of Mechanism
- Complete Mediation
- Open Design
- Least Common Mechanism
- Psychological acceptability
- Weakest Link
- Leveraging Existing Components

OWASP Secure Design Principles

- **Minimise attack surface** area: reduce potential vulnerabilities, restrict the function a user can access.
- Establish **secure defaults**: strong registration processes, stand to high-security levels.
- The principle of **least privilege**: a user should have the minimum set of privileges required to perform a specific task.
- The principle of **defence in depth**: implement multiple layers of validation, additional security auditing tools, and logging tools (Captcha systems, IP checking, login attempts logging, brute force detection, …).

OWASP Secure Design Principles

- **Fail securely**: failing should not change the user privileges and it should not show user sensitive information, database queries or logs.
- **Don’t trust services**: pay close attention to third-party services.
- Separation of duties: assign right authorization and permissions to correct users, pay close attention to superusers.
- **Avoid security by obscurity**: there should be sufficient security controls in place to keep your application safe without hiding core functionality or source code.
- **Keep security simple**: Do not use sophisticated and complex software, it will likely increase the risk of errors.
- **Fix security issues correctly**: follow strict procedure to identify the root of security related problems and repair them accordingly.

Other Common Secure-Design Principles

According to "The Protection of Information in Computer Systems"

- Economy of mechanism
- Fail-safe defaults
- Complete mediation
- Open design Open design
- Separation of privilege
- Least privilege Operate
- Least common mechanism
- Psychological acceptability Is your secured product easy to use?

Code Validation

1. Create a Validation Plan.
2. Define System Requirements.
3. Establish Validation Protocol and Test Specifications.
4. Complete Testing Campaigns.
5. Develop Procedures/Reports.