



BITS Pilani

Pilani | Dubai | Goa | Hyderabad

SEZG566/SSZG566

Secure Software Engineering

T V Rao



- *The slides presented here are obtained from the authors of the books, product documentations, and from various other contributors. I hereby acknowledge all the contributors for their material and inputs.*
- *I have added and modified slides to suit the requirements of the course.*



Phases in software development

Software Engineering - Definitions

- (1969 – Fritz Bauer) Software engineering is the establishment and use of *sound engineering principles* in order to obtain *economically* software that is *reliable* and works *efficiently* on *real machines*
- (IEEE) The application of a *systematic, disciplined, quantifiable* approach to the *development, operation, and maintenance* of software; that is, the application of engineering to software

Knowledge Areas in v3(2014) of SWEBOK

Requirements	Configuration Management
Design	Quality
Construction	Processes
Testing	Models & Methods
Maintenance	Engineering Management
	Project Management
	Economics

Some Authors refer two columns as Primary & Supporting Activities
Pressman calls them Framework and Umbrella Activities

Software Engineering Process KA

A Process defines who is doing what, when, and how to reach a certain goal

-Ivar Jacobson, Grady Booch, and James Rumbaugh

A software process is a set of interrelated activities and tasks that transform input work products into output work products. At minimum, the description of a software process includes required inputs, transforming work activities, and outputs generated.

- SWEBOK

A software process infrastructure can provide process definitions, policies for interpreting and applying the processes, and descriptions of the procedures to be used to implement the processes.

A software development life cycle (SDLC) includes the software processes used to specify and transform software requirements into a deliverable software product.

How Process Models Differ?

While all Process Models take same primary and supporting activities, they differ with regard to

- Overall flow of activities, actions, and tasks and the interdependencies among them
- Degree to which actions and tasks are defined within each framework activity
- Degree to which work products are identified and required
- Manner in which quality assurance activities are applied
- Manner in which project tracking and control activities are applied
- Overall degree of detail and rigor with which the process is described
- Degree to which customer and other stakeholders are involved in the project
- Level of autonomy given to the software team
- Degree to which team organization and roles are prescribed

Prescriptive and agile processes

Prescriptive processes are processes where all of the process activities are planned in advance and progress is measured against this plan.

In agile processes, planning is incremental and it is easier to change the process to reflect changing customer requirements.

In practice, most practical processes may include elements of both plan-driven and agile approaches.

There are NO right or wrong software processes.

Software Requirements

The Software Requirements knowledge area (KA) is concerned with the elicitation, analysis, specification, and validation of software requirements as well as the management of requirements during the whole life cycle.

Software projects are critically vulnerable when the requirements related activities are poorly performed

Functional & Nonfunctional Requirements

Functional requirements describe the functions that the software is to execute; for example, formatting some text or modulating a signal. They are sometimes known as capabilities or features. A functional requirement can also be described as one for which a finite set of test steps can be written to validate its behavior.

Nonfunctional requirements are the ones that act to constrain the solution. Nonfunctional requirements are aka constraints or quality requirements. They can be classified according to whether they are performance requirements, maintainability requirements, safety requirements, reliability requirements, security requirements, interoperability requirements

Emergent properties of software

Some requirements represent *emergent properties* of software—that is, requirements that cannot be addressed by a single component but that depend on how all the software components interoperate.

- The throughput requirement for a call center would, for example, depend on how the telephone system, information system, and the operators all interacted under actual operating conditions.

Emergent properties are crucially dependent on the system architecture.

Experts consider ***Security*** as an ***emergent property*** of the software.

Software Design

Software design consists of two activities that fit between software requirements analysis and software construction:

- Software architectural design (sometimes called high-level design): develops top-level structure and organization of the software and identifies the various components.
- Software detailed design: specifies each component in sufficient detail to facilitate its construction.

Software Design

- Software design principles include abstraction; coupling and cohesion; decomposition and modularization; encapsulation/information hiding; separation of interface and implementation; sufficiency, completeness, and primitiveness; and separation of concerns.
- Design for security is concerned with
 - How to prevent unauthorized disclosure, creation, change, deletion, or denial of access to information and other resources.
 - It is also concerned with how to tolerate security-related attacks or violations by limiting damage, continuing service, speeding repair and recovery, and failing and recovering securely.

Software Construction

Software construction refers to the detailed creation of working software through a combination of coding, verification, unit testing, integration testing, and debugging.

Throughout construction, software engineers both unit test and integration test their work. Thus, the Software Construction KA is closely linked to the Software Testing KA.

Code is the ultimate deliverable of a software project, and thus the Software Quality KA is closely linked to the Software Construction KA.

Software Testing

Software testing consists of the *dynamic* verification that a program provides *expected* behaviors on a *finite* set of test cases, suitably *selected* from the usually infinite execution domain

- *Dynamic*: The input value alone is not always sufficient to specify a test, since a system might react to the same input with different behaviors, depending on the system state.
- *Finite*: Even in simple programs, so many test cases are theoretically possible that exhaustive testing is infeasible.
- *Selected*: How to identify the most suitable test set under given conditions is a complex problem; in practice, risk analysis techniques and software engineering expertise are applied.
- *Expected*: It must be possible, although not always easy, to decide whether the observed outcomes of program testing are acceptable or not.



BITS Pilani

Pilani | Dubai | Goa | Hyderabad

Software development – Supporting Activities

Software quality

Software quality may refer:

- to desirable characteristics of software products,
- to the extent to which a particular software product possess those characteristics, and
- to processes, tools, and techniques used to achieve those characteristics

Experts defined variously

“conformance to requirements” - Phil Crosby

“achieving excellent levels of fitness for use” - Watt Humphrey

“market-driven quality” where the “customer is the final arbiter” - IBM

Software quality

A healthy software engineering culture includes the understanding that tradeoffs among cost, schedule, and quality are a basic tenant of the engineering of any product. The tradeoff is best decided by understanding four cost of quality categories: prevention, appraisal, internal failure, and external failure.

Prevention costs include investments in software process improvement efforts, quality infrastructure, quality tools, training, audits, and management reviews

Appraisal costs arise from project activities that find defects.

Costs of internal failures are those that are incurred to fix defects found during appraisal activities and discovered prior to delivery of the software product to the customer

External failure costs include activities to respond to software problems discovered after delivery to the customer.

Configuration management

Configuration management (CM) is the discipline of identifying the configuration of a system at distinct points in time for the purpose of systematically controlling changes to the configuration and maintaining the integrity and traceability of the configuration throughout the system life cycle.

A system can be defined as the combination of interacting elements organized to achieve one or more stated purposes

Configuration of a software system is collection of specific versions of hardware, firmware, or software items combined according to specific build procedures for a purpose.

Software Maintenance

A software product must change or evolve over time. Once a software is in operation, defects are uncovered, operating environments change, and new user requirements surface.

Software maintenance is an integral part of a software life cycle. However, software development used to receive more importance than software maintenance in most organizations. It is changing

- Due to large capital spending needed for software development, organizations are trying to maximize lifespan of existing software
- Due to large scale availability of open source components, organizations increased focus on maintenance

Maintainer's activities

Five key characteristics comprise the maintainer's activities:

- maintaining control over the software's day-to-day functions;
- maintaining control over software modification;
- perfecting existing functions;
- identifying security threats and fixing security vulnerabilities; and
- preventing software performance from degrading to unacceptable levels.

Software Engineering Management

Software engineering management can be defined as the application of management activities — planning, coordinating, measuring, monitoring, controlling, and reporting — to achieve quality etc.

There are aspects specific to software projects and software life cycle that complicate effective management, including

- Clients often don't know what is needed or what is feasible
- As a result of changing requirements, software is often built using an iterative process
- The degree of novelty and complexity is often high
- There is often a rapid rate of change in the underlying technology

Software Engineering Economics

Software engineering economics is about relating the attributes of software and software processes to economic measures

- involves balancing risk and profitability, while maximizing benefits and wealth of the organization.
- identify organizational goals, time horizons, risk factors, and financial constraints
- identify and implement the appropriate portfolio and investment decisions to manage cash flow, and funding;
- measure financial performance, such as cash flow and ROI

Software Engineering Process

software process is a set of interrelated activities and tasks that transform input work products into output work products

a software process includes required inputs, transforming work activities, and outputs generated

a software process may also include its entry and exit criteria and decomposition of the work activities into tasks, which are the smallest units of work

Value individuals & interactions over processes & tools – Agile manifesto

Agile models are designed to facilitate evolution of the software requirements during the project

Software Engineering Professional Practice

Software Engineering Professional Practice includes knowledge, skills, and attitudes that software engineers must possess to practice software engineering in a professional, responsible, and ethical manner

The concept of professional practice becomes applicable within the professions that have a generally accepted body of knowledge

A code of ethics and professional conduct for software engineering was approved by the ACM Council and the IEEE CS Board of Governors in 1999

–Software engineers shall commit themselves to making the analysis, specification, design, development, testing and maintenance of software a beneficial and respected profession. In accordance with their commitment to the health, safety and welfare of the public, software engineers shall adhere to the eight principles concerning the public, client and employer, product, judgment, management, profession, colleagues, and self, respectively



Work products during SDLC

Requirement Models

Scenario-based modeling – represents the system from the user's point of view

Flow-oriented modeling – provides an indication of how data objects are transformed by a set of processing functions

Class-based modeling – defines objects, attributes, and relationships

Behavioral modeling – depicts the states of the classes and the impact of events on these states

SafeHome (from Pressman)

SafeHome: The home security function would protect against and/or recognize a variety of undesirable “situations” such as illegal entry, fire, flooding, carbon monoxide levels, and others. It’ll use our wireless sensors to detect each situation. It can be programmed by the homeowner, and will automatically telephone a monitoring agency when a situation is detected.

Objects described for SafeHome might include the control panel, smoke detectors, window and door sensors, motion detectors, an alarm, an event (a sensor has been activated), a display, a PC, telephone numbers, a telephone call, and so on. The list of services might include configuring the system, setting the alarm, monitoring the sensors, dialing the phone, programming the control panel, and reading the display (note that services act on objects).

Use-Cases

A collection of user scenarios that describe the thread of usage of a system

Each scenario is described from the point-of-view of an “actor”—a person or device that interacts with the software in some way

Each scenario answers the following questions:

- Who is the primary actor, the secondary actor (s)?
- What are the actor’s goals?
- What preconditions should exist before the story begins?
- What main tasks or functions are performed by the actor?
- What extensions might be considered as the story is described?
- What variations in the actor’s interaction are possible?
- What system information will the actor acquire, produce, or change?
- Will the actor have to inform the system about changes in the external environment?
- What information does the actor desire from the system?
- Does the actor wish to be informed about unexpected changes?

Use case—detailed example (Pressman)

Example: “SAFEHOME” system (Pressman)

Use case name: *InitiateMonitoring*

Participating actors: homeowner, technicians, sensors

Flow of events (homeowner):

- Homeowner wants to set the system when the homeowner leaves house or remains in house
- Homeowner observes control panel
- Homeowner enters password
- Homeowner selects “stay” or “away”
- Homeowner observes that red alarm light has come on, indicating the system is armed

Use case—detailed example (Pressman)

Pre condition(s)

Homeowner decides to set control panel

Post condition(s)

- Control panel is not ready; homeowner must check all sensors and reset them if necessary
- Control panel indicates incorrect password (one beep)—homeowner enters correct password
- Password not recognized—must contact monitoring and response subsystem to reprogram password
- *Stay* selected: control panel beeps twice and lights *stay* light; perimeter sensors are activated
- *Away* selected: control panel beeps three times and lights *away* light; all sensors are activated

Use case—detailed example (Pressman)

Quality requirements:

- Control panel may display additional text messages
- time the homeowner has to enter the password from the time the first key is pressed
- Ability to activate the system without the use of a password or with an abbreviated password
- Ability to deactivate the system before it actually activates

Misuse case (searchsoftwarequality.techtarget.com)

Name: Attack on "forgot password" functionality

Summary: A malicious user tries to attack the "forgot password" functionality in order to gain access to the Web application or guess a valid e-mail address

Author: Anurag Agarwal

Date: April 15, 2006

Possible Attacks:

- SQL injection attack
- Brute force attack to guess a valid user
- Sniffing attack on e-mail sent with password on an insecure transmission channel

Trigger Point: Can happen anytime

Preconditions: None

Assumptions:

- The attacker can perform this attack remotely over the Internet
- The attacker can be an anonymous user

Misuse case (searchsoftwarequality.techtarget.com)

Worst case threat (post condition) :

- Attacker gains entry into the company database and steals sensitive information
- Attacker is able to modify an existing e-mail address to its own e-mail address and mails the password to himself to gain unauthorized entry into the system

Related business rule:

- The system should e-mail the password to a valid e-mail address entered

Capture guarantee (post condition) :

- Attacker cannot gain access to the database to steal or modify information
- Attacker cannot identify the e-mail address of a valid user
- Attacker cannot view the password sent in an e-mail to an e-mail address of a valid user

Misuse case (searchsoftwarequality.techtarget.com)

Potential misuser profile:

- Script kiddie
- Skilled attacker

Threat level: High

Mitigation steps:

- SQL injection attack
 - List all the mitigation steps to avoid a SQL injection attack.
- Brute force attack
 - Accept first name, last name along with e-mail address
 - Have proper error handling so as not to reveal information to the attacker
 - Delay 3 to 5 seconds before re-entering the e-mail address
 - Lock out page for that IP address after 10 attempts
- Sniffing attack
 - Send password e-mail on a secure transmission channel with strong encryption

Data Flow Modeling

Data Flow Diagram

- Depicts how input is transformed into output as data objects move through a system

Process Specification

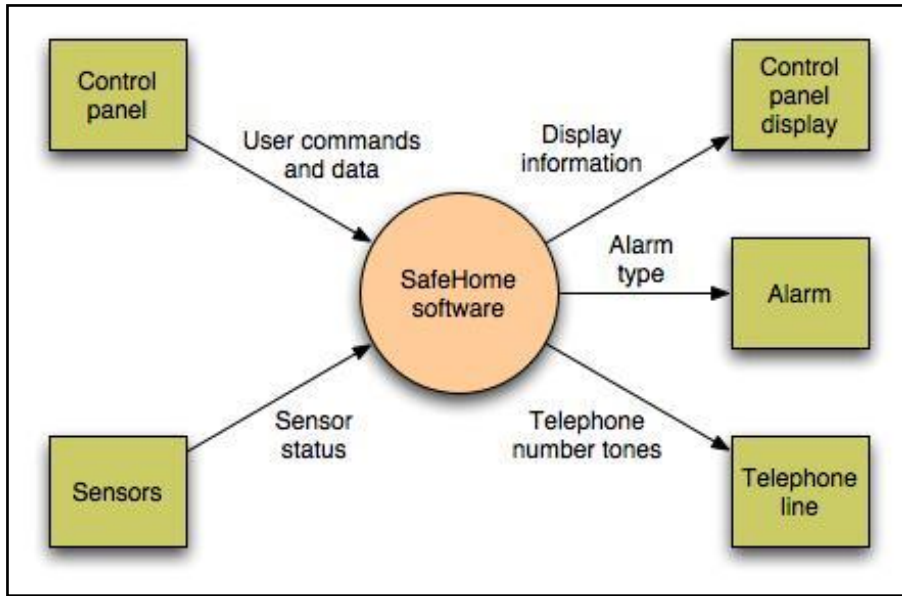
- Describes data flow processing at the lowest level of refinement in the data flow diagrams

Data Flow Modeling

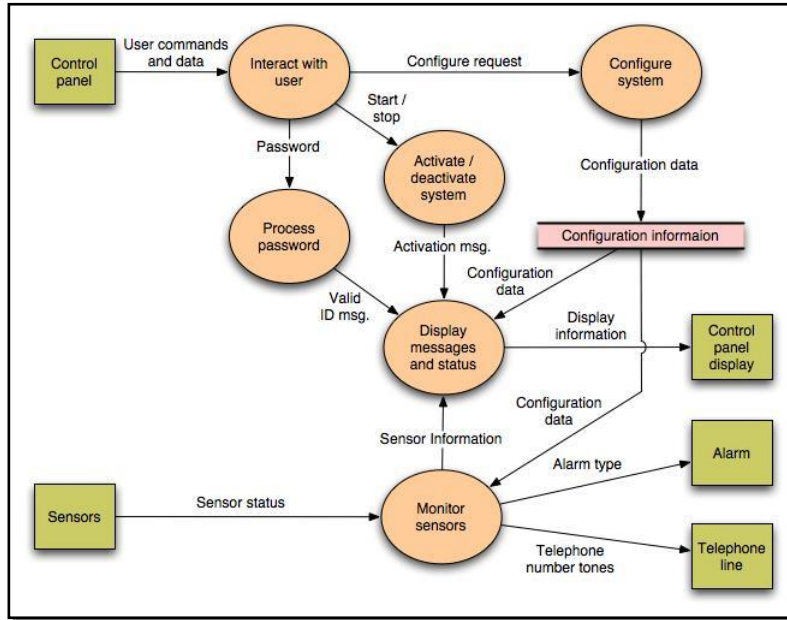
Guidelines

- Depict the system as single bubble in level 0.
- Carefully note primary input and output.
- Refine by isolating candidate processes and their associated data objects and data stores.
- Label all elements with meaningful names.
- Maintain information conformity between levels.
- Refine one bubble at a time.

Data Flow Diagram



Data Flow Diagram (Next Level)





BITS Pilani

Pilani | Dubai | Goa | Hyderabad

Security implications to SDLC

What is OWASP SAMM?

SAMM stands for Software Assurance Maturity Model.

“Our mission is to provide an effective and measurable way for all types of organizations to analyze and improve their software security posture. We want to raise awareness and educate organizations on how to design, develop, and deploy secure software through our self-assessment model. SAMM supports the complete software lifecycle and is technology and process agnostic. We built SAMM to be evolvable and risk-driven in nature, as there is no single recipe that works for all organizations.”

OWASP SAMM

SAMM (Software Assurance Maturity Model) is the OWASP framework to help organizations assess, formulate, and implement a strategy for software security, that can be integrated into their existing Software Development Lifecycle (SDLC)

SAMM is based around a set of 15 security practices, which are grouped into five business functions

Every security practice contains a set of activities, structured into three maturity levels (1-3).

OWASP SAMM Overview



SAMM Checklist

OWASP SAMM Toolkit.xlsx
(<https://owaspsamm.org/assessment/>)

OWASP SAMM Practices

SAMM prescribes methodology for practice that includes

- Assessment
- Results
- Success metrics
- Costs
- Personnel (Roles)
- Levels



BITS Pilani

Pilani | Dubai | Goa | Hyderabad

Security implications to SDLC

Security Development Lifecycle

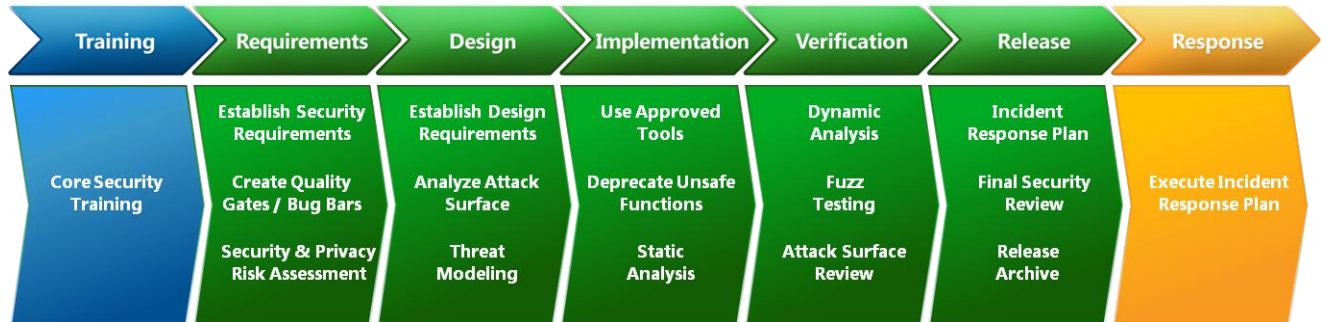
Security Development Lifecycle

The Security Development Lifecycle (SDL) is a security assurance process developed by Microsoft as a company-wide initiative and a mandatory policy to reduce the number and severity of vulnerabilities in software products.

The Microsoft SDL is based on three core concepts —

- education,*
- continuous process improvement, and*
- accountability*

Security Development Lifecycle



SDL - Core Security Training

Basic software security training should cover foundational concepts such as:

- Secure design, including the following topics: Attack surface reduction, Defense in depth, principle of least privilege, Secure defaults
- Threat modeling, including the following topics: Overview of threat modeling, Design implications of a threat model, Coding constraints based on a threat model
- Secure coding, including the following topics: Buffer overruns (for applications using C and C++), Integer arithmetic errors (for applications using C and C++), Cross-site scripting (for managed code and Web applications), SQL injection (for managed code and Web applications), Weak cryptography
- Security testing, including the following topics: Differences between security testing and functional testing, Risk assessment, Security testing methods
- Privacy, including the following topics: Types of privacy-sensitive data, Privacy design best practices, Risk assessment, Privacy development best practices, Privacy testing best practices

SDL Roles

SDL roles are designed to provide project security and privacy oversight and have the authority to accept or reject security and privacy plans from a project team.

Security Advisor/Privacy Advisor. This role is filled by subject-matter experts (SMEs) from outside the project team. The person chosen for this task must fill two sub-roles:

- Auditor: monitors each phase of the software development process and attest to successful completion of each security requirement
- Expert: must possess verifiable subject-matter expertise in security.

Team Champion. should be filled by SMEs from the project team. Responsible for the negotiation, acceptance, and tracking of minimum security and privacy requirements

Some SDL Practices

Threat Modeling

- used in environments where there is meaningful security risk
- allows development teams to consider, document, and discuss the security implications of designs in the context of their planned operational environment
- Threat modeling is a team exercise, encompassing program/project managers, developers, and testers, performed during the software design stage

Attack Surface Reduction

- a means of reducing risk by giving attackers less opportunity to exploit a potential weak spot or vulnerability
- encompasses shutting off or restricting access to system services, applying the principle of least privilege, and employing layered defenses wherever possible

Some SDL Practices

Static Analysis

- The team should be aware of the strengths and weaknesses of static analysis tools and be prepared to augment static analysis tools with other tools or human review as appropriate

Dynamic Program Analysis

- specify tools that monitor application behavior for memory corruption, user privilege issues, and other critical security problems

Fuzz Testing

- a specialized form of dynamic analysis used to induce program failure by deliberately introducing malformed or random data to an application

Some SDL Practices

Final Security Review

- a deliberate examination of all the security activities performed on a software application prior to release
- performed by the security advisor with assistance from the regular development staff and the security and privacy team leads
- includes an examination of threat models, exception requests, tool output, and performance against the previously determined quality gates or bug bars
- If a team does not meet all SDL requirements and the security advisor cannot approve the project, the project cannot be released
- Teams must either address whatever SDL requirements that they can prior to launch or escalate to executive management for a decision



BITS Pilani

Pilani | Dubai | Goa | Hyderabad

Security in Agile Development



The Manifesto for Agile Software Development

“We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.”

Kent Beck et al



What is “Agility”?

Effective (rapid and adaptive) response to change

Effective communication among all stakeholders

Drawing the customer onto the team

Organizing a team so that it is in control of the work performed

Yielding ...

Rapid, incremental delivery of software

Security in Agile Development

1. Utilize the hacker in that developer

A developer with the right tools, training and mindset can uncover the most common security pitfalls in code.

—Developers must be empowered through training and the provision of the right tools that will enable them to analyze code before submitting it to quality assurance or testing teams.

All development team members must be aware of the most common vulnerabilities in each system they develop.

—With developers' ratio to security specialists standing at about 100:1, it is only plausible to pass the responsibility of securing systems to the developers. They can then perform routine scans of their code and fix any issues that may arise during the core development process.

Team leaders can decide to incorporate the security team members into individual development teams to discover vulnerabilities early in development.

2. Always consider the “evil” user interacting with the system

Many developers envision a perfect software without taking into account the possibility of an “evil” user interaction

Team players should be made aware of the possible ways an attacker may interact with the system.

This is a continuous process starting with the statement of security-based user stories.

Only a few clients will talk about security in user stories. Security team has to identify critical points and valuable assets of the software that may be targeted by an attacker. They can enumerate possible attack goals and advise the development team in advance.

Security in Agile Development

3. Uphold continuous integration practices, tools and platforms

There are tools to integrate with development platforms and give insightful information about code.

These include code integration tools, code scanners and shared repositories.

Code analysis tools are available to help debug and refactor buggy code, thereby improving the system's overall safety.

Upholding continuous integration best practices work and help improve the quality of the software, thereby assuring the security of users, user data and vendors.

4. Review user stories with every iteration and adapt as necessary

Agile methodologies have short iterations, resulting in the release of new or improved software system features.

—Therefore, it is possible to review all user stories that necessitate iteration and how they relate to the system's general security before releasing it for public use.

The team leader must also organize peer review of code and how frequently this occurs. Two pairs of eyes are often better than one during code review.

The software must be adapted to better its reliability and security if vulnerabilities are discovered in the review process.

Security in Agile Development

5. Innovate with security

Agile teams neglect security due to absence of robust, agile security practices and testing tools.

Progressive teams have to innovate workable security policies and developing tools based on such policies.

Most CI tools are open to customization.

Therefore, an organization or team can choose a tool that works with their development practices and customize it to serve the most common security concerns affecting their code.

6. Cultivate the culture of security

Organizational culture influences how things are generally done.

If team leaders have high regard for secure systems, the juniors are likely to copy the same and practice it when writing code.

Educating the team and organizing seminars for security and agile development can also help members adopt the security needs of software quickly.

Every new team member should be taken through the necessary software security procedures and guidelines of organization.

Furthermore, combining all the other security aspects of agile development with a supportive and dynamic security culture will boost the general feel of your software.



BITS Pilani

Pilani | Dubai | Goa | Hyderabad

DevOps and DevSecOps

DevOps

A set of practices that combines software development (Dev) and IT operations (Ops)



Why DevOps? Iterative and Rapid Software release cycles mean software moves between development team to IT operations team repeatedly. It means strong binding is needed between two teams.

DevOps is complementary with Agile software development; several DevOps aspects came from the Agile methodology

DevOps Practices

Continuous integration and continuous delivery (CI/CD)

- CI - merge code changes frequently into the main code. CD - frequent, automated deployment of new versions into a production environment

Version Control

- tracking revisions and change history to make code easy to review and recover

Agile software development

- emphasizes team collaboration, customer and user feedback, and rapid change through short release cycles

Infrastructure as code

- defines system resources and topologies in a descriptive manner that allows teams to manage those resources as they would code

Configuration management

- managing the state of resources in a system including servers, virtual machines, and databases

Continuous monitoring

- having real-time visibility into the performance and health of the entire stack, from the underlying infrastructure to higher-level software components

What is DevSecOps?

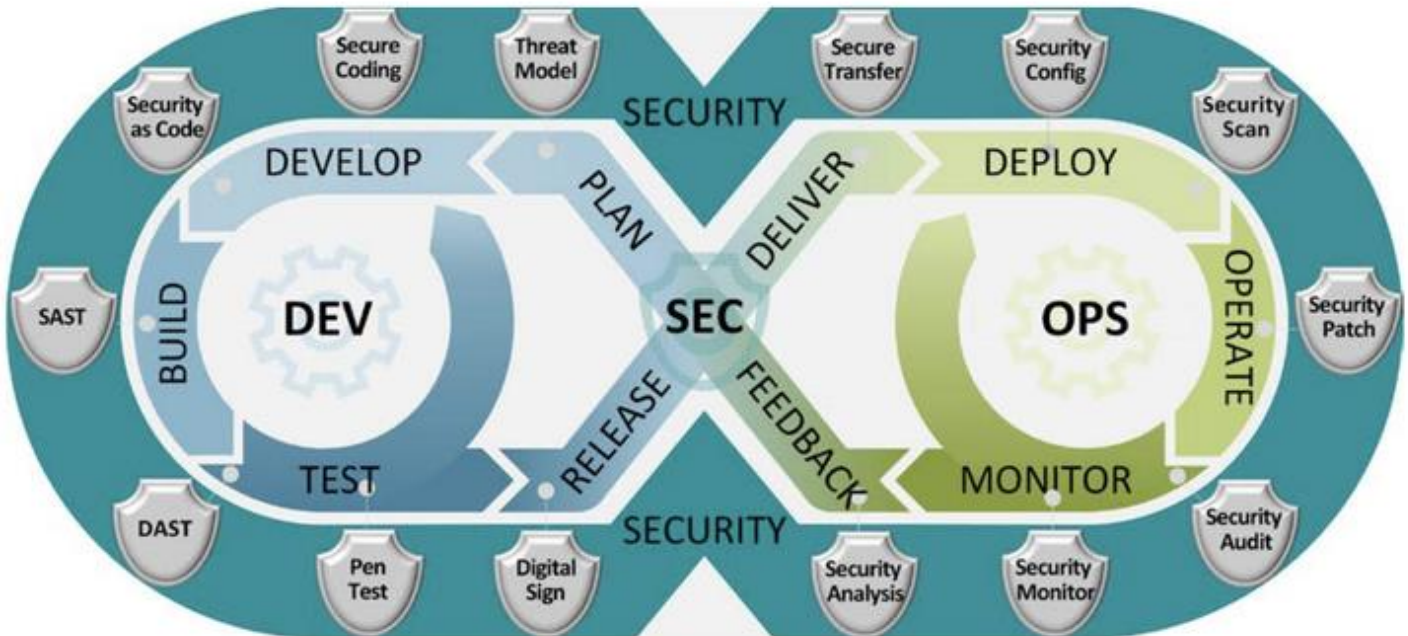
DevSecOps—short for *development, security, and operations*—automates the integration of security at every phase of the software development lifecycle, from initial design through integration, testing, deployment, and software delivery.

- In the past, security was incorporated to software at the end of the development cycle by a separate security team and was tested by a separate QA team

DevSecOps integrates application and infrastructure security seamlessly into Agile and DevOps processes and tools.

DevSecOps makes application and infrastructure security a shared responsibility of development, security, and IT operations teams, rather than the sole responsibility of a security silo

DevSecOps Software Lifecycle



Best Practices for DevSecOps

Shift left


- It encourages software engineers to move security from the right (end) to the left (beginning) of the DevOps (delivery) process.
- Shifting left allows the DevSecOps team to identify security risks and exposures early and ensures that these security threats are addressed immediately

Security education

- development engineers, operations teams, and compliance teams should be familiar with the basic principles of application security and other security engineering practices

Traceability, auditability, and visibility

- Traceability helps to track configuration items across the development cycle to where requirements are implemented in the code
- Auditability is for ensuring compliance with security controls
- Visibility means the organization has a measure the heartbeat of the operation, send alerts, has awareness of cyberattacks as they occur, and provide accountability



Software Engineering: A Practitioner's Approach, 7/e Roger Pressman

Software Engineering, Pearson Education, 9th Ed., 2010. Ian Sommerville

SWEBOK by IEEE/ACM

www.owasp.com

sei.cmu.edu

www.microsoft.com

www.devopedia.org/devops

www.ibm.com



Thank You!