Lecture 14, April 29

ECS 235B, Foundations of Computer and Information Security

Spring Quarter 2026

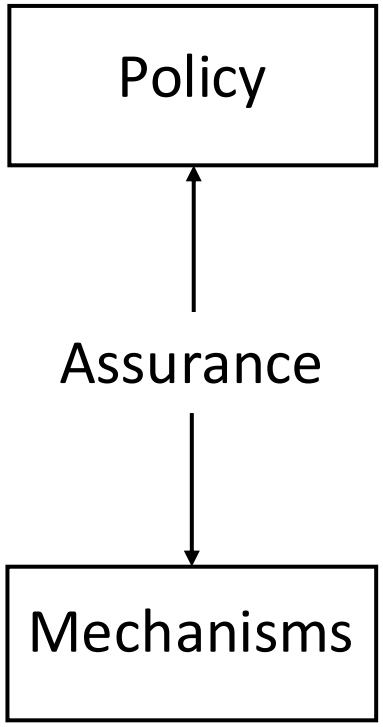
Assurance: Overview

- Trust
- Problems from lack of assurance
- Types of assurance
- Life cycle and assurance
- Waterfall life cycle model
- Other life cycle models

Trust

- *Trustworthy* entity has sufficient credible evidence leading one to believe that the system will meet a set of requirements
- *Trust* is a measure of trustworthiness relying on the evidence
- *Assurance* is confidence that an entity meets its security requirements based on evidence provided by applying assurance techniques

Relationships



Statement of requirements that explicitly define the security expectations of the mechanism(s)

Provides justification that the mechanism meets policy through assurance evidence and approvals based on evidence

Executable entities that are designed and implemented to meet the requirements of the policy

Trusted System

- System that has been shown to meet well-defined requirements under an evaluation by a credible body of experts who are certified to assign trust ratings or assurance levels to evaluated products and systems
  - Use specific methodologies to gather assurance evidence
  - These methodologies typically have increasing ”levels of trust”

Problem Sources

1. Requirements definitions, omissions, and mistakes
2. System design flaws
3. Hardware implementation flaws, such as wiring and chip flaws
4. Software implementation errors, program bugs, and compiler bugs
5. System use and operation errors and inadvertent mistakes
6. Willful system misuse
7. Hardware, communication, or other equipment malfunction
8. Environmental problems, natural causes, and acts of God
9. Evolution, maintenance, faulty upgrades, and decommissions

Examples

- Challenger explosion
  - Sensors removed from booster rockets to meet accelerated launch schedule
- Deaths from faulty radiation therapy system
  - Hardware safety interlock removed
  - Flaws in software design
- Bell V22 Osprey crashes
  - Failure to correct for malfunctioning components; two faulty ones could outvote a third
- Intel 486 chip
  - Bug in trigonometric functions

Role of Requirements

- *Requirements* are statements of goals that must be satisfied
  - Vary from high-level, generic issues to low-level, concrete issues
- *Security objectives* are high-level security issues
- *Security requirements* are specific, concrete issues
- *Security policy* is set of specific statements that, when enforced, result in a secure system
  - Alternatively, a statement that partitions states of system into a set of authorized states and a set of unauthorized states
- *Security model* describes a family of policies, systems, or entities and is more abstract than a policy
  - A policy is specific to particular entities

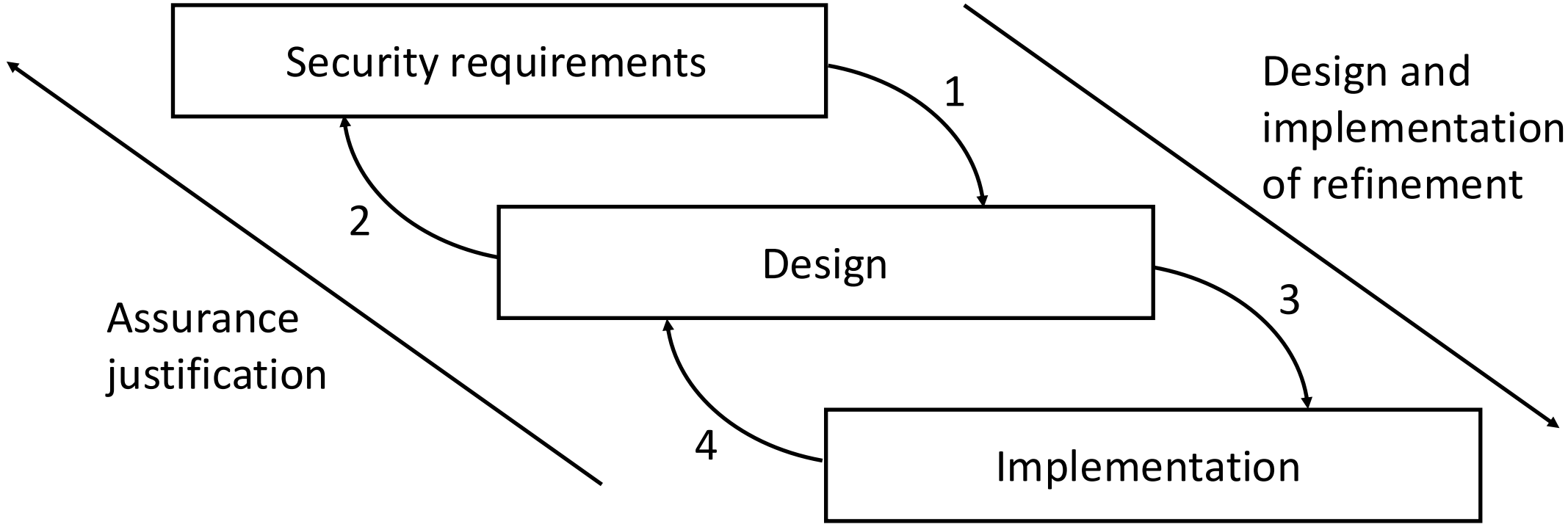
Types of Assurance

- *Policy assurance* is evidence establishing security requirements in policy is complete, consistent, technically sound
- *Design assurance* is evidence establishing design sufficient to meet requirements of security policy
- *Implementation assurance* is evidence establishing implementation consistent with security requirements of security policy

Types of Assurance

- *Operational assurance* is evidence establishing system sustains the security policy requirements during installation, configuration, and day-to-day operation
  - Also called *administrative assurance*

Life Cycle


Security requirements
1
Design and
implementation
of refinement
2
Design
3
Assurance
justification
4
Implementation

Life Cycle for Building Secure, Trusted Systems

- Life cycle process establish discipline, control in the building of a product or system
  - This provides confidence in consistency, quality of resulting system
- Assurance *requires* life cycle model end engineering process in *every* situation
  - Size and complexity will vary
- Life cycle defined in stages

Generic Life Cycle Model

These are present in all models, but the emphasis and focus is different for each project, and will be more detailed than what is presented here

- Conception
- Manufacture
- Deployment
- Fielded Product Life

Conception

- Idea
  - Decisions to pursue it
- Proof of concept
  - See if idea has merit
- High-level requirements analysis
  - What does “secure” mean for this concept?
  - Is it possible for this concept to meet this meaning of security?
  - Is the organization willing to support the additional resources required to make this concept meet this meaning of security?
- Identify threats, assumptions

Manufacture

- Develop detailed plans for each group involved
  - May depend on use; internal product requires no sales
- Implement the plans to create entity
  - Includes decisions whether to proceed, for example due to market needs
  - Software development, engineering process is in this stage

Deployment

- Delivery
  - Assure that correct masters are delivered to production and protected
  - Assure integrity of what is delivered to customers, sales organizations
- Installation and configuration
  - Ensure product works appropriately for specific environment into which it is installed
  - Service people know security procedures
- Example of configuration failure
  - 2013: Target breached via a third party vendor, as network architected with improper security controls

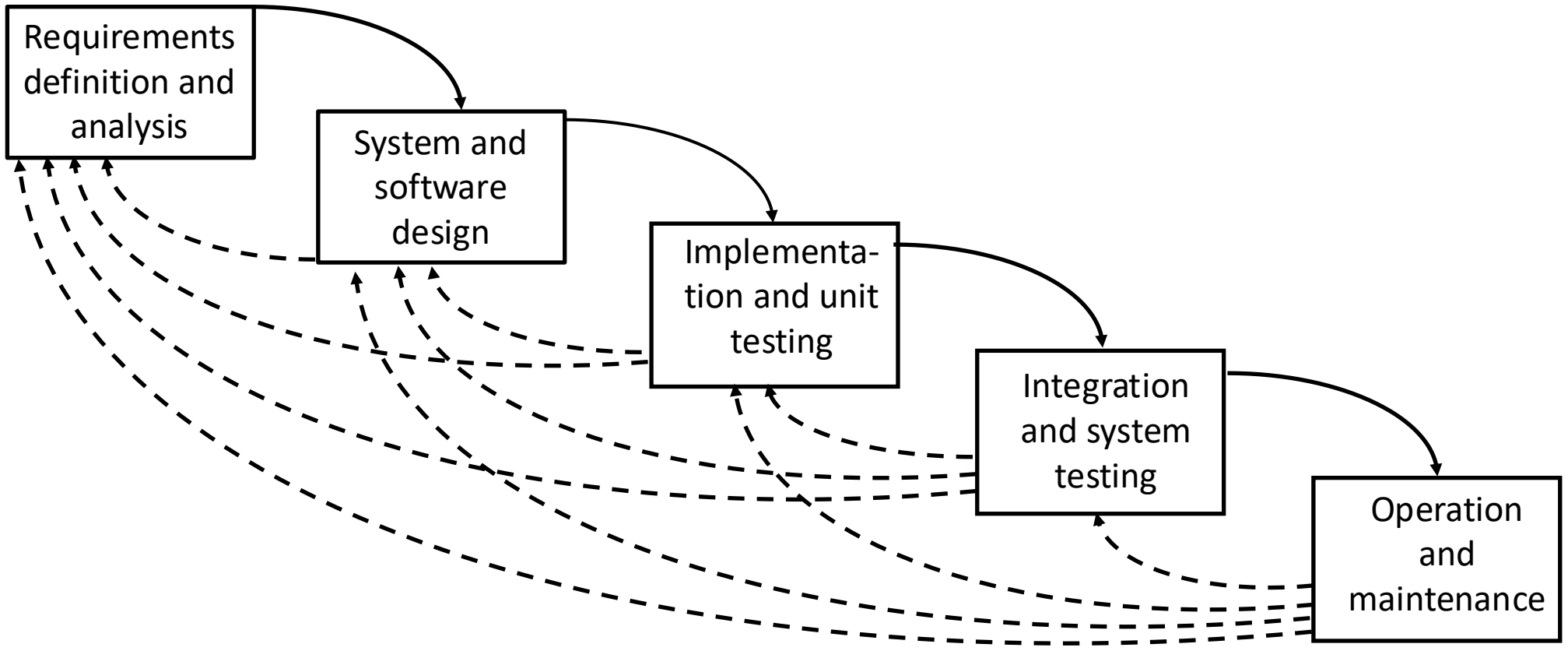
Fielded Product Life

- Routine maintenance, patching
  - Responsibility of engineering in small organizations
  - Responsibility may be in different group than one that manufactures product
  - Example of failure: 2017 Equifax breach believed due to failing to install an important system patch, resulting in breach of financial information for hundreds of millions of people
- Customer service, support organizations
- Retirement or decommission of product

Waterfall Life Cycle Model

- Requirements definition and analysis
  - Functional and non-functional
  - General (for customer), specifications
- System and software design
- Implementation and unit testing
- Integration and system testing
- Operation and maintenance

Relationship of Stages



Agile Software Development

- Software development is creative process, always changing, never really completed
- Leads to agile methodologies
  - Focuses on working together
  - Agile team efficiently works together in their environment
  - Team engages customer as a member of the team, developing requirements and scoping of the project
  - Accept, adapt to rapidly changing requirements
    - Allows for continuous improvement

Agile Methodologies

Term “Agile software development” used to describe several Agile methodologies

- Scrum
- Kanban
- Extreme Programming (XP)
- Others
  - Feature-Driven Development (FDD), Dynamic Systems Development Method (DSDM), Pragmatic Programming

In all, evidence of trustworthiness for assurance adduced *after* development

Scrum

- Split project into small parts that can be done in a short timeframe (called a *sprint*)
  - This *product backlog* created by product owner, who represents customer, product stakeholders
- Scrum team agrees on a small subset from top of backlog, decides how to design, implement it
  - Goal: complete this within the sprint

Scrum

- Every day, team meets to evaluate progress, adjust as needed to get a workable solution within each sprint
  - At the end, work completed should be ready to ship, demo, or put back into backlog if not complete
- Iterate until product complete

Kanban

- Identify lanes of work: to be done, in progress, completed, deployed
- Each lane except the last has limit on how many items can be in that lane
  - Based on staff available to perform the work
- Teams take item off to be done lane, work on it until completed
  - When implemented correctly, team is completing work on top item in lane when another item arrives
- Goal: deliver product to customer within expected timeline
  - Methodology originated at Toyota

Extreme Programming

- Rapid prototyping and “best practices”
- Project driven by business decisions
- Requirements open until project complete
- Programmers work in teams
- Components tested, integrated several times a day
- Objective is to get system into production as quickly as possible, then enhance it

Models

- Exploratory programming
  - Develop working system quickly
  - Used when detailed requirements specification cannot be formulated in advance, and adequacy is goal
  - No requirements or design specification, so low assurance
- Prototyping
  - Objective is to establish system requirements
  - Future iterations (after first) allow assurance techniques

Models

- Formal transformation
  - Create formal specification
  - Translate it into program using correctness-preserving transformations
  - Very conducive to assurance methods
- System assembly from reusable components
  - Depends on whether components are trusted
  - Must assure connections, composition as well
  - Very complex, difficult to assure

Threats and Goals

- *Threat* is a danger that can lead to undesirable consequences
- *Vulnerability* is a weakness allowing a threat to occur
- Each identified threat requires countermeasure
  - Unauthorized people using system mitigated by requiring identification and authentication
- Often single countermeasure addresses multiple threats

Architecture

- Where do security enforcement mechanisms go?
  - Focus of control on operations or data?
    - Operating system: typically on data
    - Applications: typically on operations
  - Centralized or distributed enforcement mechanisms?
    - Centralized: called by routines
    - Distributed: spread across several routines

Layered Architecture

- Security mechanisms at any layer
  - Example: 4 layers in architecture
    - *Application layer*: user tasks
    - *Services layer*: services in support of applications
    - *Operating system layer*: the kernel
    - *Hardware layer*: firmware and hardware proper
- Where to put security services?
  - Early decision: which layer to put security service in

Security Services in Layers

- Choose best layer
  - User actions: probably at applications layer
  - Erasing data in freed disk blocks: OS layer
- Determine supporting services at lower layers
  - Security mechanism at application layer needs support in all 3 lower layers
- May not be possible
  - Application may require new service at OS layer; but OS layer services may be set up and no new ones can be added

Security: Built In or Add On?

- Think of security as you do performance
  - You don’t build a system, then add in performance later
    - Can “tweak” system to improve performance a little
    - Much more effective to change fundamental algorithms, design
- You need to design it in
  - Otherwise, system lacks fundamental and structural concepts for high assurance

Reference Validation Mechanism

- *Reference monitor* is access control concept of an abstract machine that mediates all accesses to objects by subjects
- *Reference validation mechanism* (RVM) is an implementation of the reference monitor concept.
  - Tamperproof
  - Complete (always invoked and can never be bypassed)
  - Simple (small enough to be subject to analysis and testing, the completeness of which can be assured)
    - Last engenders trust by providing evidence of correctness

Examples

- *Security kernel* combines hardware and software to implement reference monitor
- *Trusted computing base* (*TCB*) consists of all protection mechanisms within a system responsible for enforcing security policy
  - Includes hardware and software
  - Generalizes notion of security kernel