
# Module 1: Introduction and Software Lifecycle Models

## Introduction to Assured Software Engineering

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

# Notices



This material is based upon work funded and supported by the Independent Agency under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center sponsored by the United States Department of Defense.

# Outline

About this course

Software assurance challenges

Foundations for software assurance

Software assurance guiding principles

# Learning Outcomes

After completing this course, students will be able to

- Understand lifecycle models and processes for newly developed software systems
- Understand software engineering and security lifecycle models and processes for the development of a software system
- Understand assurance methods and techniques for typical lifecycle phases
- Elicit and analyze requirements for assured software
- Apply UML, analyze software behaviors
- Perform verification and validation

# Course Topics

Security models and methods in the areas of

- lifecycle process models
- risk management
- requirements engineering
- architecture and design
- verification and validation

# Prerequisites and Co-requisite

Prerequisite: Computer Science II

Co-requisite: Computer Science III

# Educational Activities

Class will be lecture and discussion

Readings from textbook, papers, reports

Homework assignments

Project including selected software development activities

# Text(s) and Key References

Allen, Julia H., Barnum, Sean, Ellison, Robert J., McGraw, Gary, & Mead, Nancy R. *Software Security Engineering: A Guide for Project Managers*. Addison Wesley Professional, 2008. (Available from InformIT and Amazon.com)

Mead, Nancy R., Woody, Carol C., *Cyber Security Engineering: A Practical Approach for Systems and Software Assurance*. Addison Wesley Professional, 2017. (Available from InformIT and Amazon.com)

U.S. Department of Homeland Security. Build Security In Website

Additional readings and videos, etc. as needed

# Grading Criteria

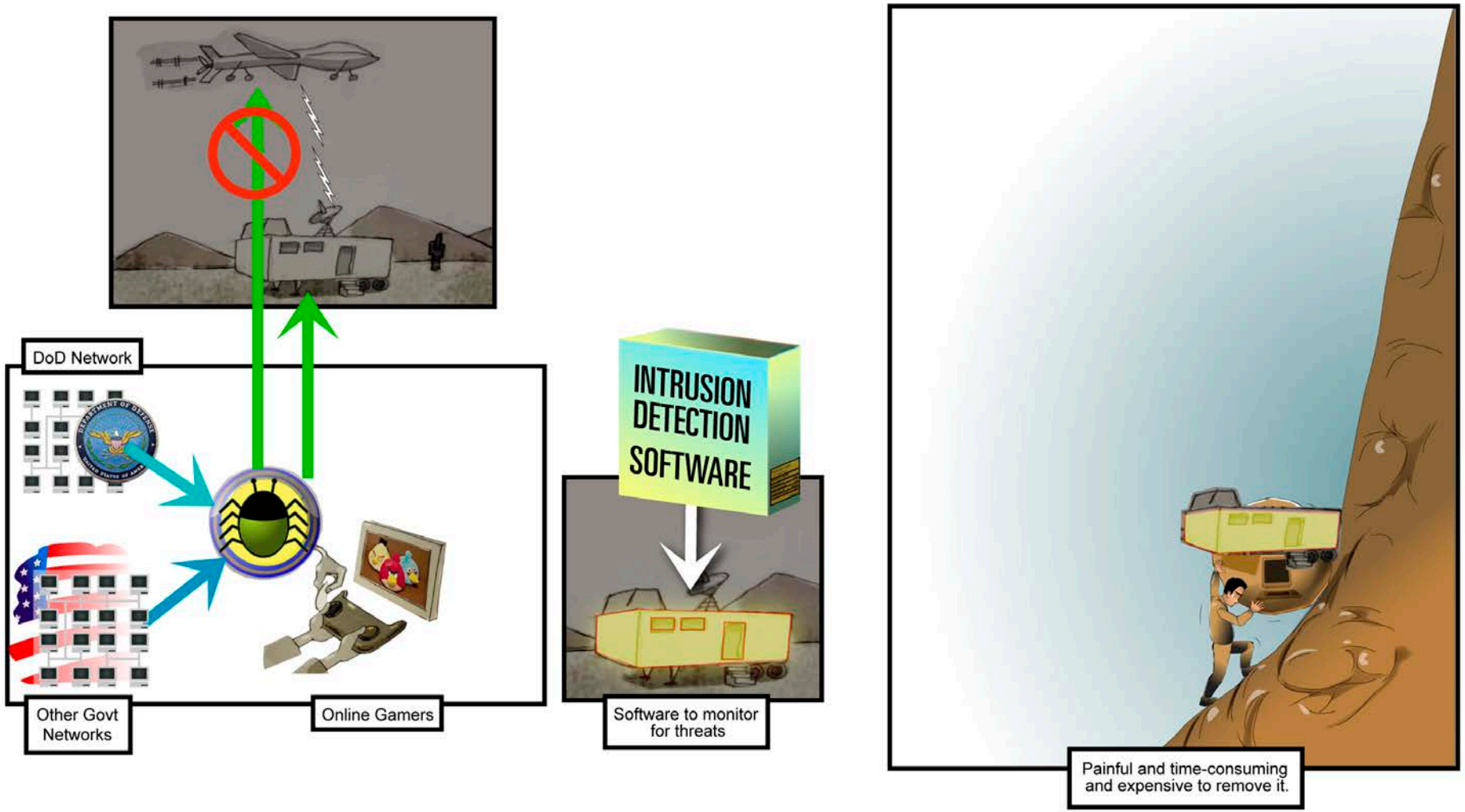
- 45% individual assignments
- 5% class participation
- 50% team project

Grading will take into consideration completeness, creativity, deep insights, and thinking outside the box. Sources must be cited. Material lifted from another source must be in quotes.

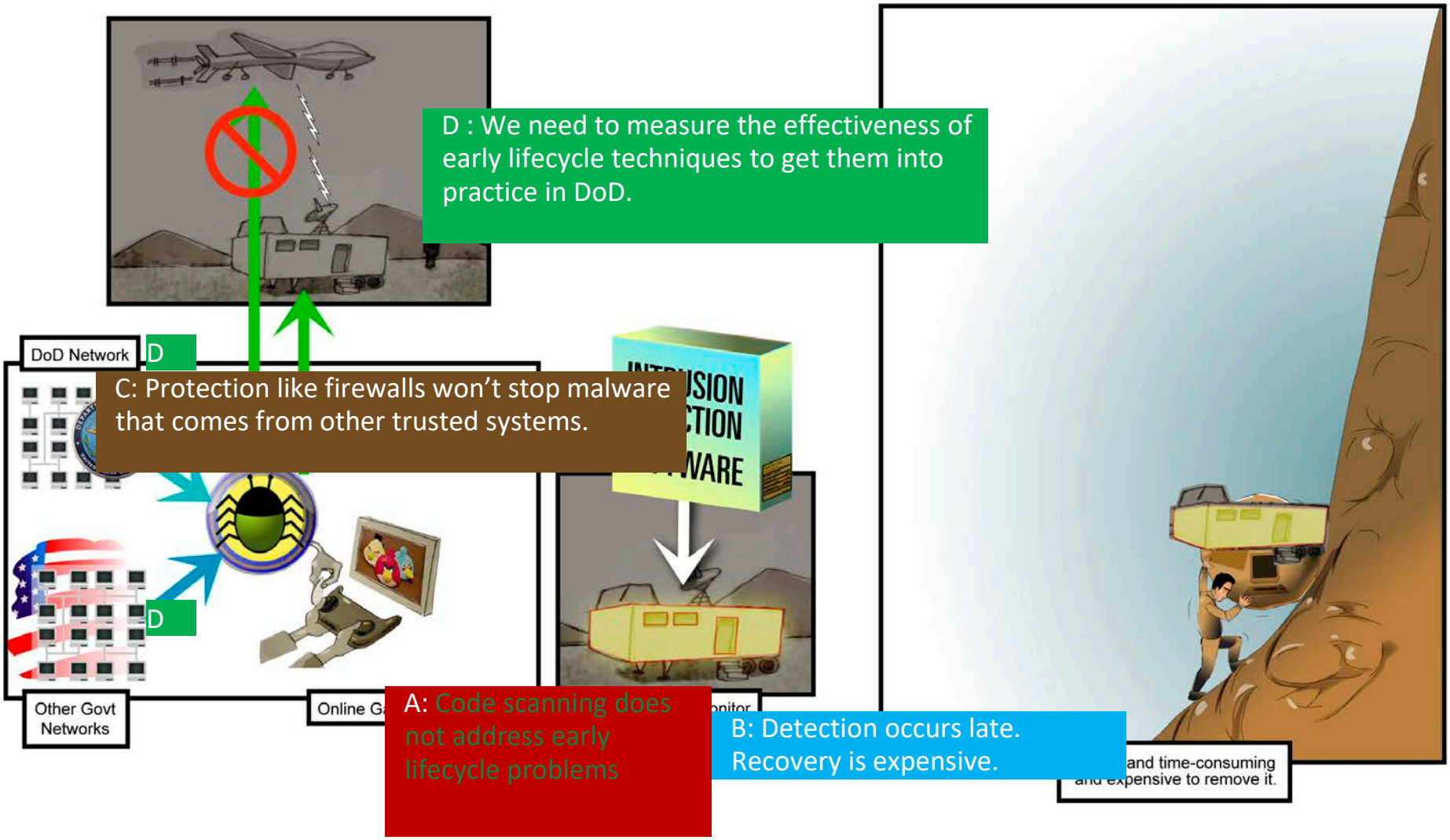
Assignments are to be turned in BEFORE class on the day they are due. Assignments not turned in on time will lose 10% for each day late.

# Software Assurance Challenges

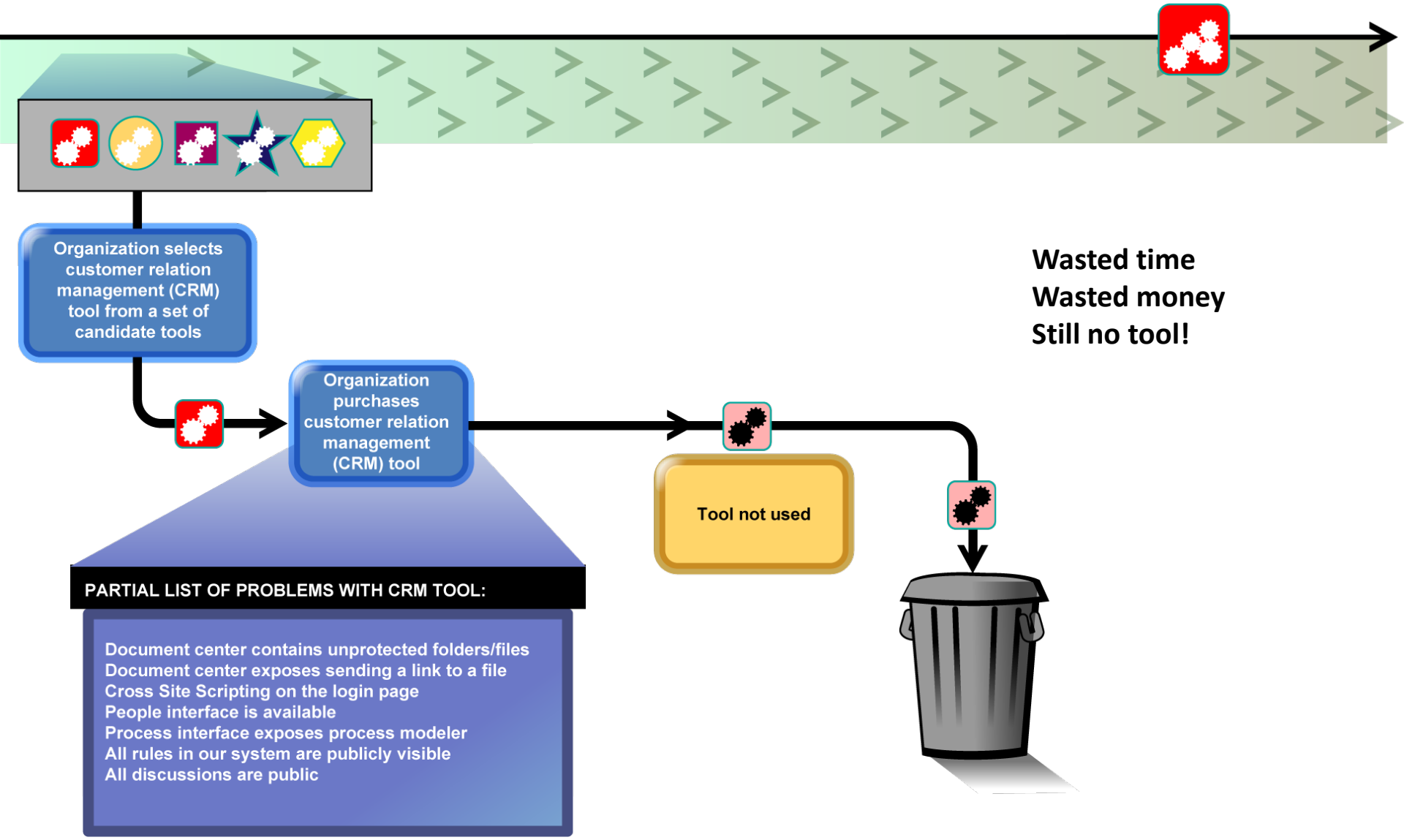
# Scenario – Drone Virus Attack



# Drone Scenario – Key Challenges



# Is There Really a COTS Security Problem?

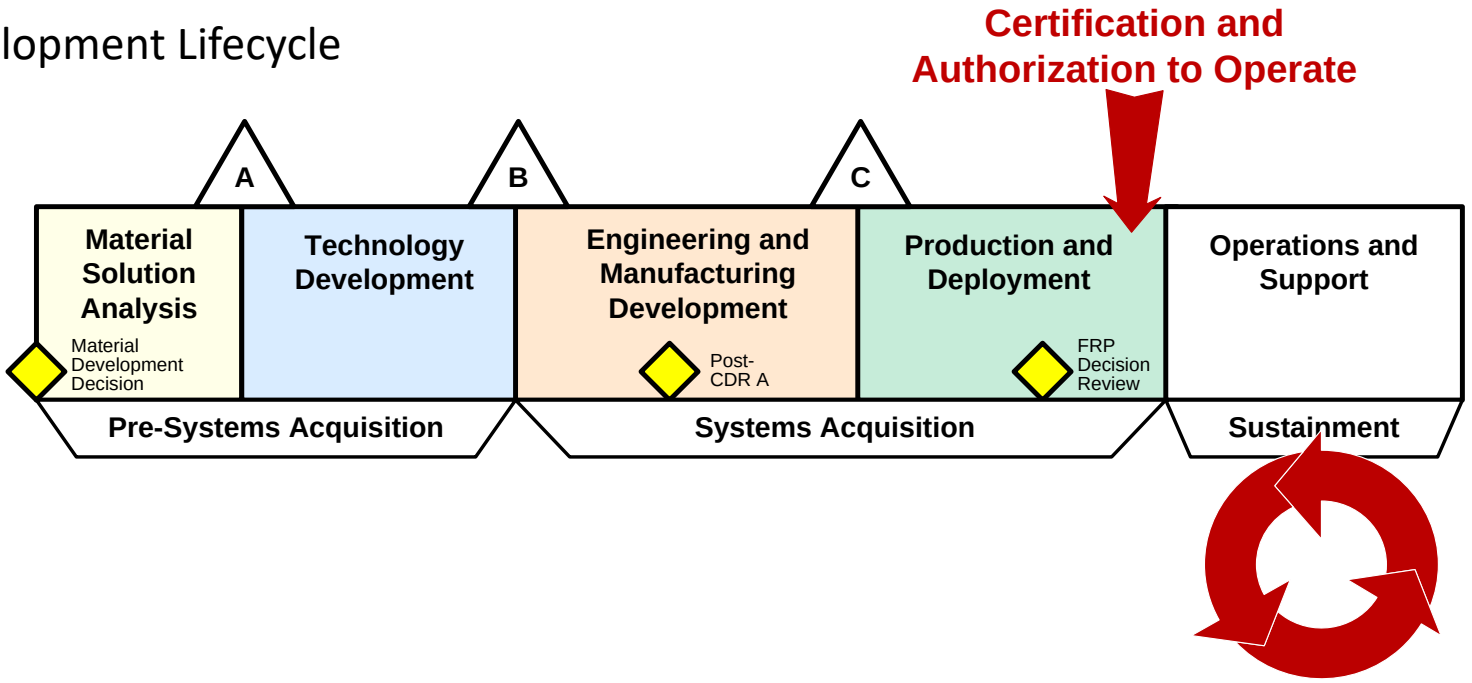


# Discussion

What examples of software security problems have you heard of lately?

# Current Challenge for Software Assurance

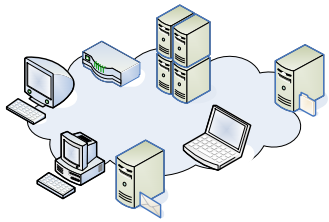
Development Lifecycle






**Software Patch Cycle**

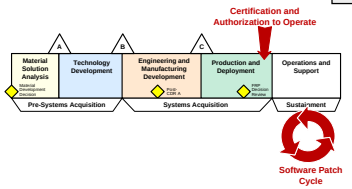
Patch & Pray

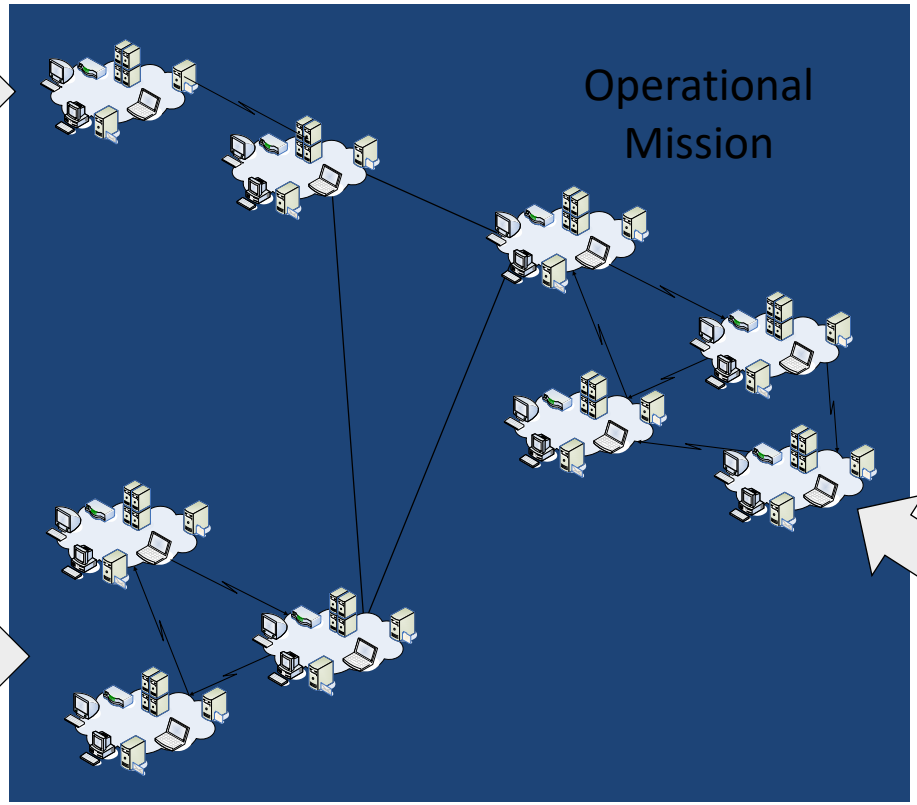


47,202 known vulnerabilities as of 9/17/11

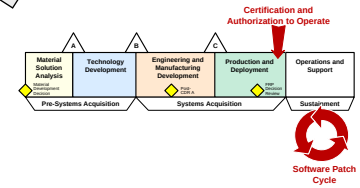
# Operational Mission Reality – Systems of Systems

Development 1

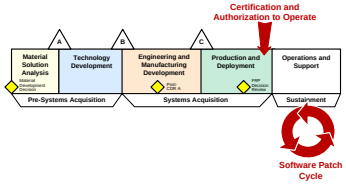



Assure and Verify Mission Security




Development 3




Development 2

# Discussion

What is Software Assurance?

# Definition: Software Assurance

Software assurance (Software Assurance Curriculum Project)

> An application of technologies and processes to achieve a required level of confidence that software systems and services function in the intended manner, are free from accidental or intentional vulnerabilities, provide security capabilities appropriate to the threat environment, and recover from intrusions and failures.

# Foundations for Software Assurance

# Information/IT Security Point of View

Typically dealing with an organization's infrastructure provider, the management chain, and the CIO

End objective is to provide a functional, available, secure operational infrastructure and applications for all users

Information protection and privacy are demanding increasing attention (regulatory, marketplace pressure)

Software/application security may or may not be on the radar screen

# Software Security Point of View -1

Dealing primarily with software/application developers and their management chain

- in-house, service provider, purchased software

End objective is to produce working systems and applications, on schedule, on budget

# Software Security Point of View -2

Security typically addressed (if at all):

- During coding and testing
- During operations/production as an “after the fact” add-on; reactive
- For COTS, open source, or third party software, as a provider/vendor responsibility
- What’s wrong with the above approach?

COTS: Commercial Off The Shelf

# Why Software Security? -1

Developed nations' economies and defense depend, in large part, on the reliable execution of software.

Software is ubiquitous, affecting all aspects of our personal and professional lives.

Software vulnerabilities are equally ubiquitous, jeopardizing

- Personal identities
- Intellectual property
- Consumer trust
- Business services, operations, and continuity
- Critical infrastructures and government

# Why Software Security? -2

Most successful attacks result from

- Targeting and exploiting known, non-patched software vulnerabilities
- Insecure software configurations

Many of these are introduced during software design and development.

Increasing trend of assembling systems from purchased parts means getting software acquisition right with respect to security.

Refer to Polydys and Wisseman. “Software Assurance in Acquisition: Mitigating Risks to the Enterprise.” 2007.

# What Is Software Security?

Software security is NOT

- Firewalls, intrusion detection, encryption, or tools that protect the environment in which the software operates

Software security IS

- Engineering software so that it continues to function under attack
- The ability of software to recognize, resist, tolerate, and recover from events that threaten it

The goal: Better, defect-free software that can function more robustly in its operational production environment

# Security Perspectives

https://www.securecoding.cert.org/confluence/display/seccode/Top+10+Secure+Coding+Practices

# Software Needs to Be Trusted

Exploitation of software defects is estimated to cost the U.S. economy $60 billion annually.

Software development and sustainment activities must follow proper practices, but there is no authoritative point of reference.

The U.S. Department of Homeland Security (DHS) created a group to define a common body of knowledge (CBK) for secure software assurance.

# Definition: Software Assurance (recap)

Software assurance (Software Assurance Curriculum Project)

- Application of technologies and processes to achieve a required level of confidence that software systems and services function in the intended manner, are free from accidental or intentional vulnerabilities, provide security capabilities appropriate to the threat environment, and recover from intrusions and failures.

# Addressing the Gaps

DHS enlisted the SEI's CERT Division to coordinate the development of a curriculum for a Master of Software Assurance (MSwA) degree program. (what and how)
http://www.cert.org/mswa/

- Developed a curriculum body of knowledge and associated outcomes
- Identified the need for a coherent set of guiding principles for secure software assurance

The SEI's CERT Division and the Software Engineering Program at Oxford University, UK collaborated to build a set of principles. (why)

# Software Assurance Guiding Principles

# Security Principles -1

Saltzer and Schroeder* defined security as “techniques that control who may use or modify the computer or the information contained in it”.

They described the three main categories of concern:

- Confidentiality
- Integrity
- Availability

* Reference: Saltzer and Schroeder, “The Protection of Information in Computer Systems.” *Communications of the ACM*, 1974.

# Security Principles -2

Economy of mechanism: Keep the design as simple and small as possible.

Fail-safe defaults: Base access decisions on permission rather than exclusion.

Complete mediation: Every access to every object must be checked for authority.

Open design: The design should not be secret. The mechanisms should not depend on the ignorance of potential attackers, but rather on the possession of specific, and more easily protected, keys or passwords.

# Security Principles -3

Separation of privilege: Where feasible, a protection mechanism that requires two keys to unlock it is more robust and flexible than one that allows access to the presenter of only a single key.

Least privilege: Every program and every user of the system should operate using the least set of privileges necessary to complete the job.

Least common mechanism: Minimize the amount of mechanism common to more than one user and depended on by all users.

Psychological acceptability: It is essential that the human interface be designed for ease of use, so that users routinely and automatically apply the protection mechanisms correctly.

# Technology Environment in 1974

S360 in use from 1964-1978

S370 came on the market in 1972

COBOL and BAL programming languages

MVS operating system released in March 1974

Patches were carefully tested to minimize operational disruption

# Changes Since 1974

Internet

Morris worm – November 2, 1988

50,000+ software vulnerabilities and exposures (CVE)

Java, C++, C#

Mobile computing

Cloud

Etc.

# Principles of Software Assurance

A set of principles to guide learners in understanding the WHY of software assurance

# Risk

**The perception of risk drives assurance decisions.**

- Assurance implementation choices (policies, practices, tools, restrictions) are based on the perception of threat and the impact should that threat be realized.
- Perceptions are based on successful attacks.
  - The current state of assurance is largely reactive.
  - More successful organizations react and recover faster, learn from the reactive responses or others, and are more vigilant in anticipating and detecting attacks.
- Misperceptions are failures to recognize threats and impacts – “how could it happen to us?” or “it could not happen here!”

# Interactions

**Highly connected systems require alignment of risk across all stakeholders and systems** otherwise critical threats will be unaddressed (missed, ignored) at different points in the interactions.

- There are costs to addressing assurance which must be balanced against the impact of the risk.
- Risk must also be balanced with other opportunities/needs (performance, reliability, usability, etc.).
- Interactions occur at many technology levels (network, security appliances, architecture, applications, data storage, etc.) and are supported by a wide range of roles.

# Trusted Dependencies

**Your assurance depends on other people's assurance decisions and the level of trust you place on these dependencies**.

- Each dependency represents a risk.
- Dependency decisions should be based on a realistic assessment of the threats, impacts, and opportunities represented by an interaction.
- Dependencies are not static and trust relationships should be reviewed to identify changes that warrant reconsideration.
- Using many standardized pieces to build technology applications and infrastructure increases the dependency on other's assurance decisions.

# Attacker

There are **no perfect protections against attacks**.

There exists a broad community of attackers with growing technology capabilities able to compromise the confidentiality, integrity, and availability of any and all of your technology assets and the attacker profile is constantly changing.

- The attacker uses technology, processes, standards, and practices to craft a compromise (socio-technical responses).
- Attacks are crafted to take advantage of the ways we normally use technology or designed to contrive exceptional situations where defenses are circumvented

# Coordination and Education

Assurance requires **knowledge of what can go wrong and effective risk coordination among all technology participants** and their governing bodies.

- Protection must be applied broadly across the people, processes, and technology because the attacker will take advantage of all possible entry points.
- Authority and responsibility must be clearly established at an appropriate level in the organization to ensure effective participation and coverage.
- All participants must have appropriate competencies for software assurance.

# Well-Planned and Dynamic

T**he capabilities to respond to a threat must be designed into the system and the threat is always changing**.

Assurance implementation must represent a balance among governance, construction, and operation and is highly sensitive to changes in each of these areas.

- Engineering challenge: Assurance cannot be added later; you must plan and build to the level of acceptable assurance that you need**.**
- Continuous monitoring must be part of the planned response.
- No one has resources to redesign systems every time the threat changes.

# Measurable

**A means to measure and monitor assurance must be built in**.

If you cannot measure it, you cannot manage it.

- All elements of the socio-technical environment must tie together (practices, processes, procedures, products, etc.) and measurements must be consistent.
- Effective measurement is well supported by sound engineering and organizational principles.
  - Well formed and consistently applied processes are critical to ensure an appropriate measurable response.
- Measurement must be multi-faceted.

# Class Assignment One

Surf the web and find four different actual examples of successful intrusion:

- One that resulted from human error (e.g., such as giving out a password or downloading a virus)
- One that resulted from a system configuration error
- One that resulted from software providing an intrusion opportunity because of a flawed development process
- One that resulted from a vulnerability in a COTS product

Describe how each of these attacks could have been avoided.

- Consider changes in policy, configuration management, software development practice, and COTS acquisition practices.

# Questions?


# Module 2:
# Software Development Lifecycles
## (Developed by David Root)

Introduction to Assured Software Engineering

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

# Notices



This material is based upon work funded and supported by the Independent Agency under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center sponsored by the United States Department of Defense.

# Topics

Software development lifecycles (SDLCs)

- Defined
- Difference from “process”
- Compare to development variables
- Common lifecycles

# What Is a Software Lifecycle?

# What Is a Lifecycle?

Webster's Dictionary (1892):

"The series of stages in form and functional activity through which an organism passes between successive recurrences of a specified primary stage."

Reifer (1997): (product)

"Period of time that begins when a software product is conceived and ends when the product is retired from use."

# What Is a Software Lifecycle?

The *software lifecycle* is the *cradle to grave* existence of a software product or software intensive system.

- includes initial development, repairs, and enhancement, and decommission

Management of the entire lifecycle of a software intensive system requires a deeper knowledge than basic in-the-small development intuition and experience.

developed by Tony Lattanze

# More on Lifecycles

Lifecycle models attempt to generalize the software development process into steps with associated activities and/or artifacts.

- They model how a project is planned, controlled, and monitored from inception to completion.

Lifecycle models provide a starting point for defining what we will do.

But, what is the end point of a project?

# So…What Is a Process?

A process is a sequence of steps performed for a given purpose.

Webster's:

"a series of actions or operations conducing to an end"

a series of actions that produce something or that lead to a particular result

# Process ≠ Lifecycle

Software process is not the same as lifecycle models.

- process refers to the specific steps used in a specific organization to build systems
- indicates the specific activities that must be undertaken and artifacts that must be produced
- *process definitions* include more detail than provided lifecycle models

Software processes are sometimes defined in the context of a lifecycle model.

# What Is Important?

What you call “it” isn’t important.

What stakeholders understand *is* important.

# Sample Lifecycles

Ad Hoc

Classic (waterfall)

Prototype

RAD

Incremental

Spiral

WinWin

V model

Chaos

Concurrent COTS 4th Gen

What about Agile?

# Be Very Careful Here

Is this just semantics?

Are there standard definitions?

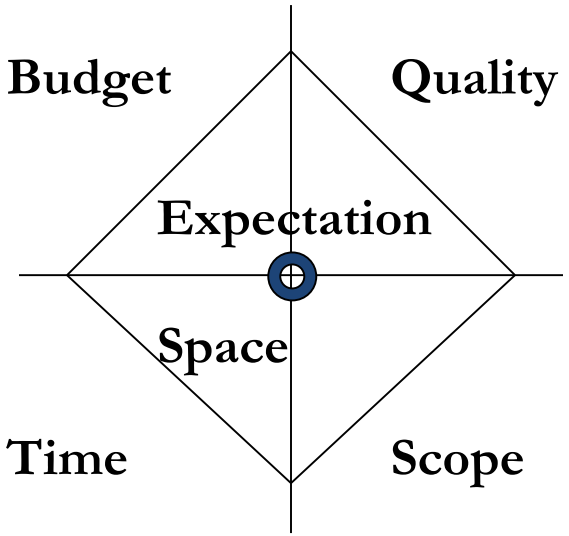
How should one approach this with a new project?

Remember, we tend to think linearly, sequentially. Is this a problem?

*Define, communicate, define, communicate...*

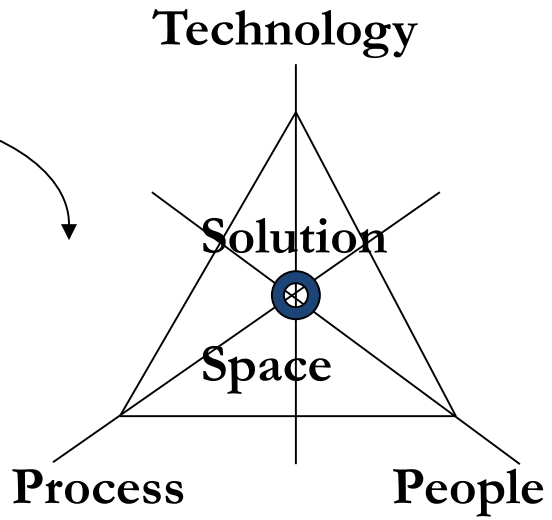
# Remember This When Looking at SDLCs

## Customer's View




f(x)

## Developer's View




**f(x) = f(Planning, Process, People, Product, ?…..)**

# When Looking At Projects

You need to ask,

"What SDLC would *define* my project best?"

(The project drives the lifecycle, not the other way around.)

What criteria are important for the project?

# Criteria You Need to Consider

Stakeholders

- Who?
- Backgrounds, domain expertise
- Commitment to project

Environments

- Business / market
- Cultures

Moral, legal constraints

# Ad Hoc “Hobbyist”

Legacy

Code – Test – Code – Test……

- Becomes a mess, chuck it, start over

Design (high-level) – Code – Test – Code – Test…..

- (Reality was Code - Test – Code – Test – Document the resulting design)

Lack of defined, formalized processes

*Is this the same as “no process?”*

*Is this still viable for a project?*

# Waterfall Model -1

First proposed in 1970 by W.W. Royce

Development flows steadily through

- requirements analysis, design implementation, testing, integration, and maintenance.

Royce advocated *iterations* of waterfalls adapting the results of the precedent waterfall.

# Waterfall Model -2

Technology had some influence on the viability of the waterfall model.

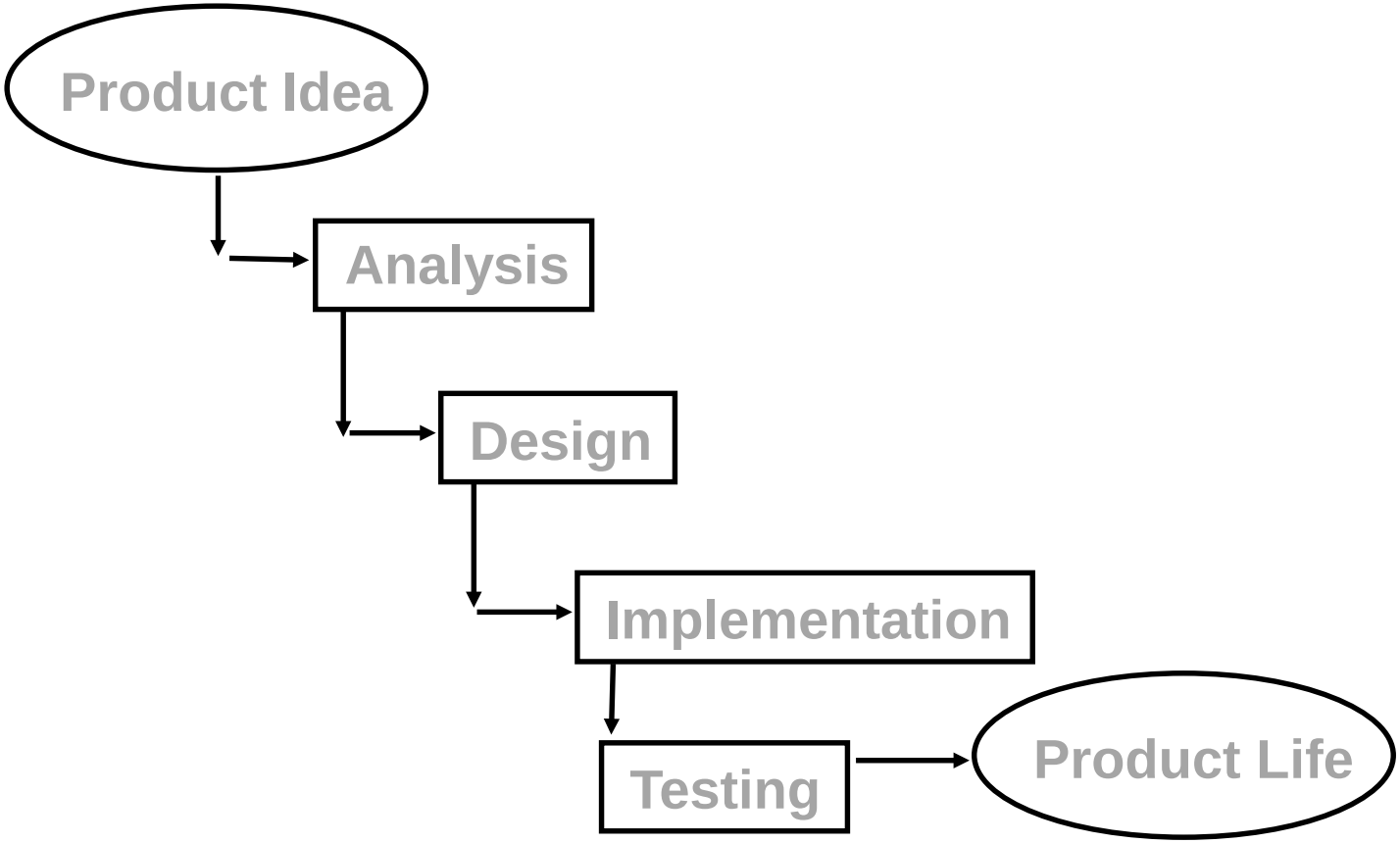
- slow code, compile, and debug cycles

Reflected the way that other engineering disciplines build things.

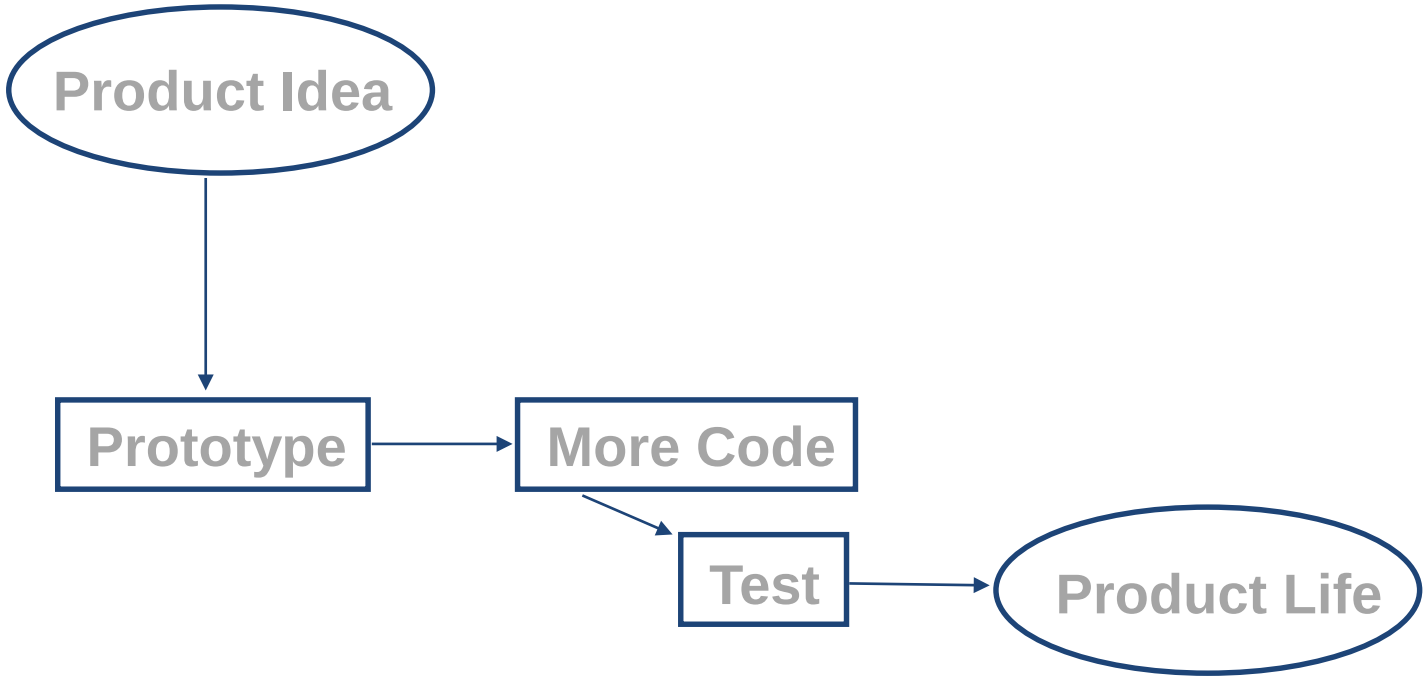
Formed the basis of the earliest software process frameworks.

Waterfall is still used today (but no one will admit it). It has a bad reputation. Why?

# Waterfall (Linear) (Classic) Model Intent



# A Common Misuse of the Rapid Prototype Model



# What Are the Problems with the Prototype Lifecycle?

When would you use it?:

Weaknesses:

# Incremental Model
## (One of the Most Misused Definitions)

The incremental model prescribes *developing* and *delivering* the product in *planned* increments.

- The product is *designed* to be *delivered* in *increments*.
- Each increments provides (in theory) more functionality than the previous increment.

Reality: Projects called “incremental” really do increments in Waterfall phases….

# However, the incremental model is used…

On almost all developments… (or the term “incremental” is used)

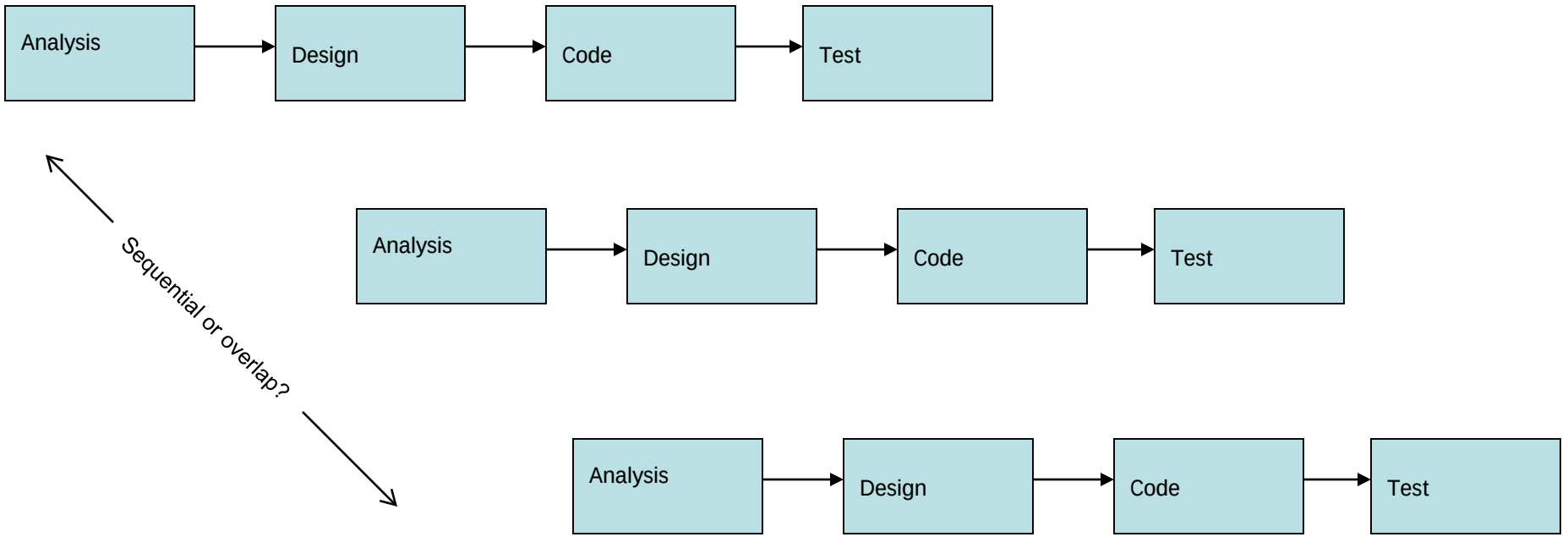
On anything done in pieces

- Agile – are these planned in advance?
- No knowing the next step until you do an increment

Be very careful to define what you mean when you say “incremental.”

It is “iterative” but so are most….

# Incremental Model
# (What Blocks Are Missing?)




**These are sequences of what?**

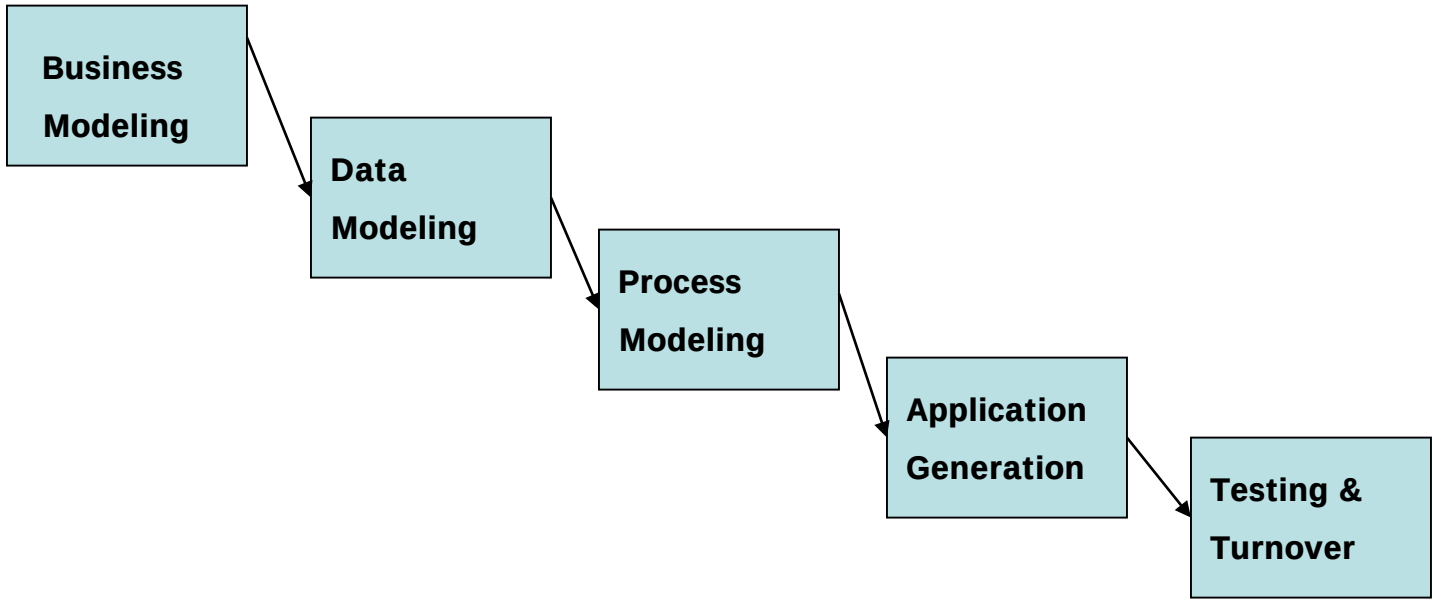
# Rapid Application Development (RAD)

Incremental

60-90 days per release

Information systems

4th generation techniques

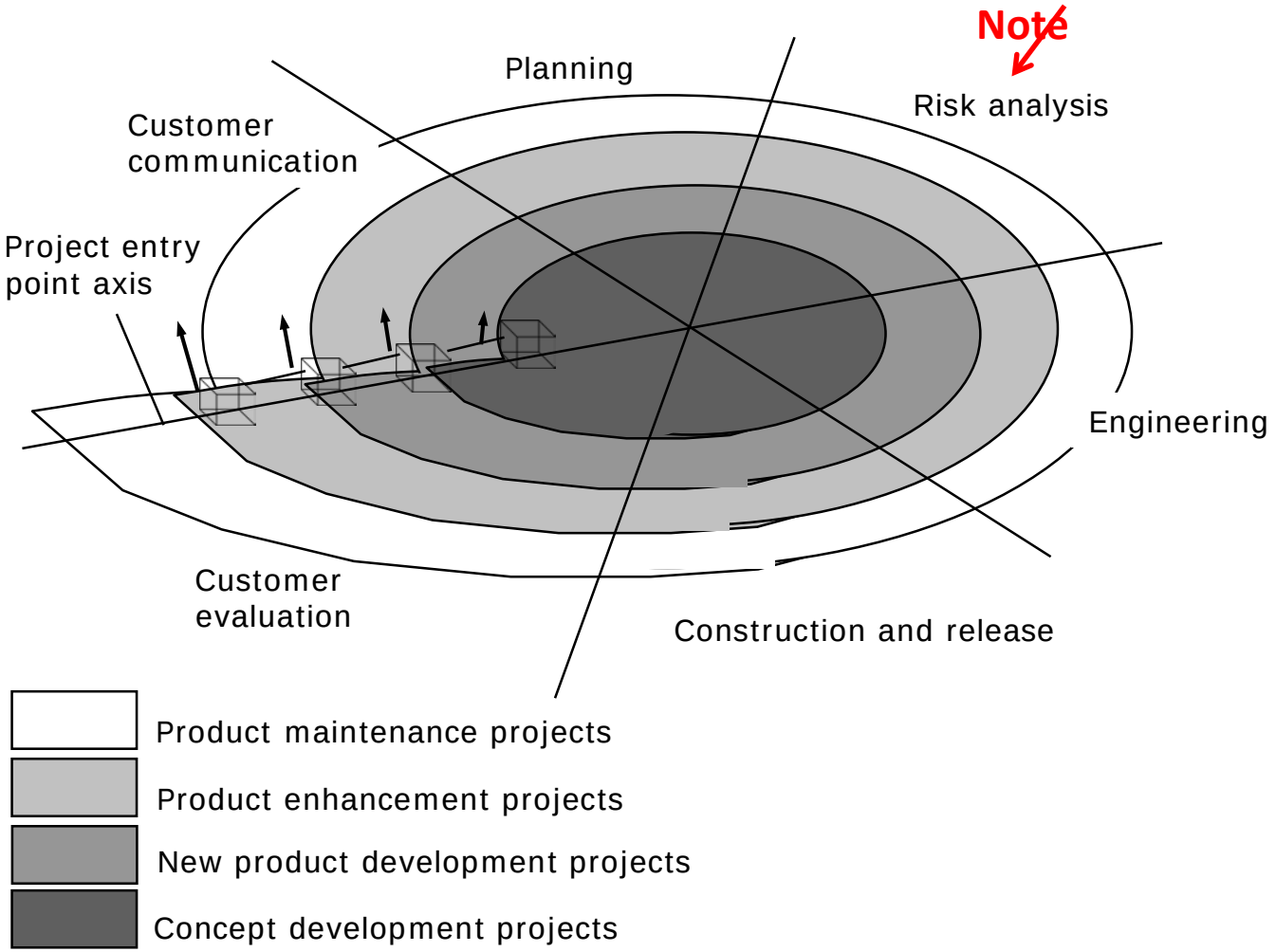


# Spiral Model -1

The spiral model

- First defined by Barry Boehm
- Combines elements of
  - Evolutionary, incremental, and prototyping models
- First model to explain
  - Why iteration matters
  - How iteration could be used effectively
- The term *spiral* refers to successive iterations outward from a central starting point.

# Spiral Model -2




Roger S. Pressman, “Software Engineering, A Practitioners Approach”

# Spiral Model -3

The goal is to

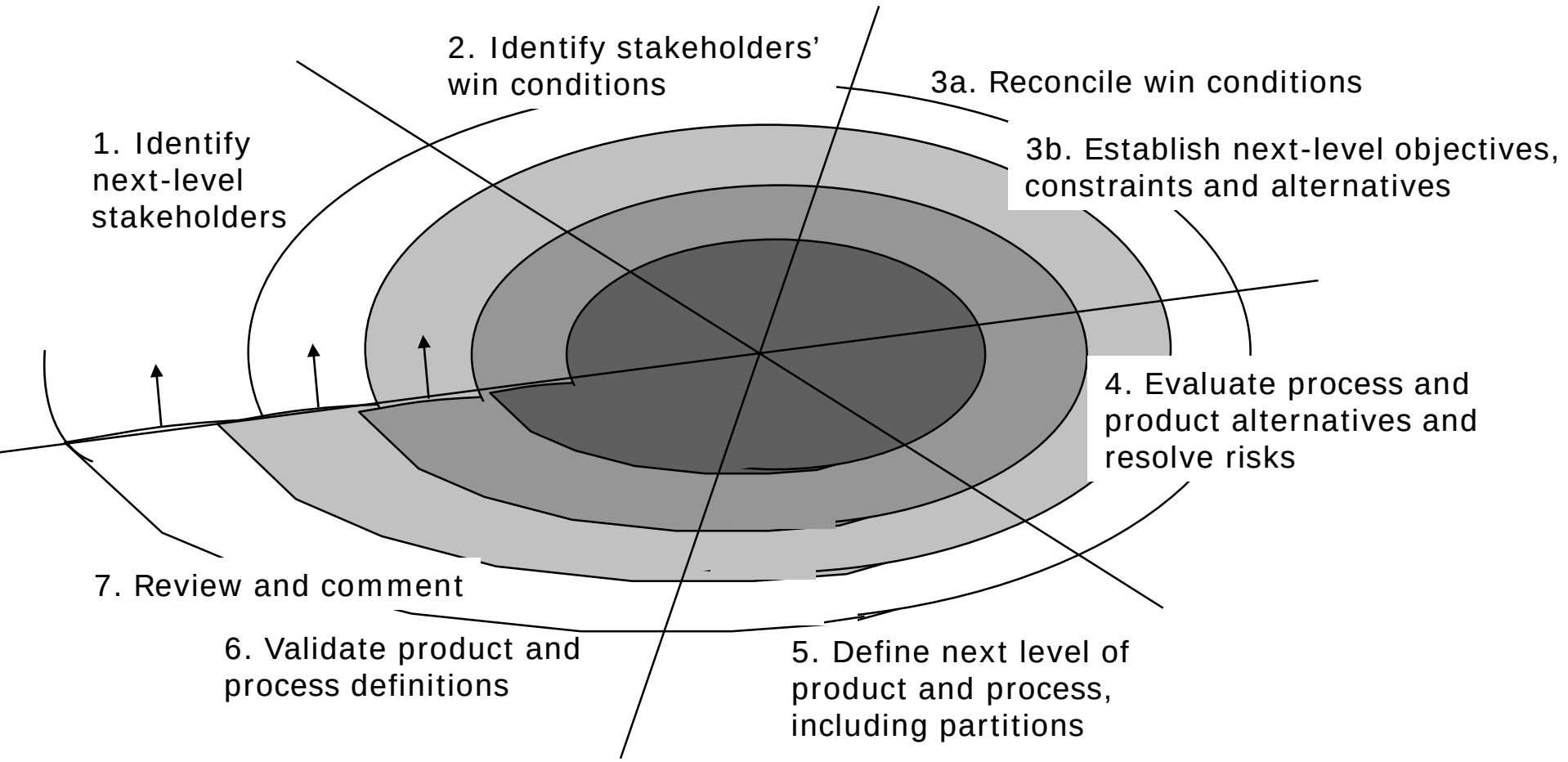
- identify risk
- focus on it early

In theory, risk is reduced in outer spirals as the product becomes more refined.

Each spiral

- starts with design goals
- ends with the client reviewing the progress thus far and future direction
- was originally prescribed to last up to two years

# WINWIN Spiral




Roger S. Pressman, Software Engineering, A Practitioners Approach

# V Model -1

Often used in system engineering environments to represent the system development lifecycle.

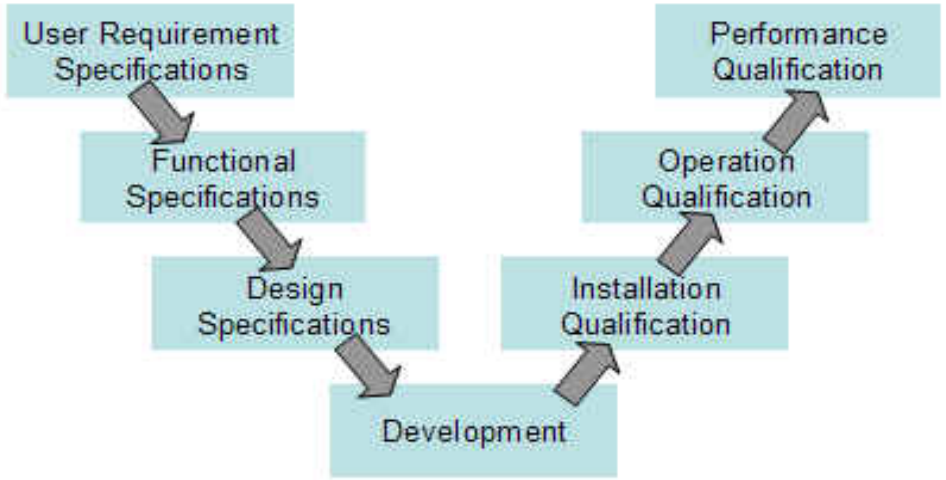
- summarizes the main steps taken to build *systems, not specifically software*
- describes appropriate deliverables corresponding with each step in the model

# V Model -2

The left side of the V represents the specification stream where the system specifications are defined.

The right side of the V represents the testing stream where the system is being tested against the specifications defined on the left side.

The bottom of the V—where the streams meet—represents the development stream.



# Which of these do you use?

# Summary

Need to define and understand SDLCs

Variables/criteria that impact selection

- Resources, time, scope, and quality

Advantages/disadvantages of each

Be careful of “easy” paths

# Other. . .

# Chaos Model -1

Extends the spiral and waterfall model defined by L.B.S. Raccoon.

Espouses the notion that the lifecycle must address all levels of a project, from the larger system to the individual lines of code.

The whole project, system, modules, functions and each line of code must by defined, implemented, and integrated holistically.

# Chaos Model -2

Chaos Theory underlies the fundamental concepts of the Chaos Model:

- Software projects are non-linear systems exhibiting random motion (linear systems are rare in nature).
- Non-linear systems can be more than the sum of their parts.
  - To characterize the behavior of a non-linear system, one needs principles to study the system as a whole and not just its parts in isolation (i.e., it is senseless to study architecture design in isolation).

# Chaos Model -3

*Chaos strategy* resembles the way that programmers work toward the end of a project:

- when they have a list of bugs to fix and features to create
- usually someone prioritizes the remaining tasks
- programmers fix bugs one at a time

Chaos strategy states that this is the only valid way to do the work.

# Components

COTS

Cycle

- Identify possible ones
- Check library
- Use (if they exist)
- Build new ones (if they don’t)
- Put new ones in library

Problems with COTS?

# SEI Process Models for COTS

PECA

- **P**lan the evaluation – stakeholders, goals, constraints, timeframe
- **E**stablish criteria – measurable, not abstract
- **C**ollect data based on criteria
- **A**nalyze – careful of first fit compared to best fit

CURE

- **C**OTS **U**sage **R**isk **E**valuation

# Concurrent

Complementary Applications

- High Interdependence with Modules

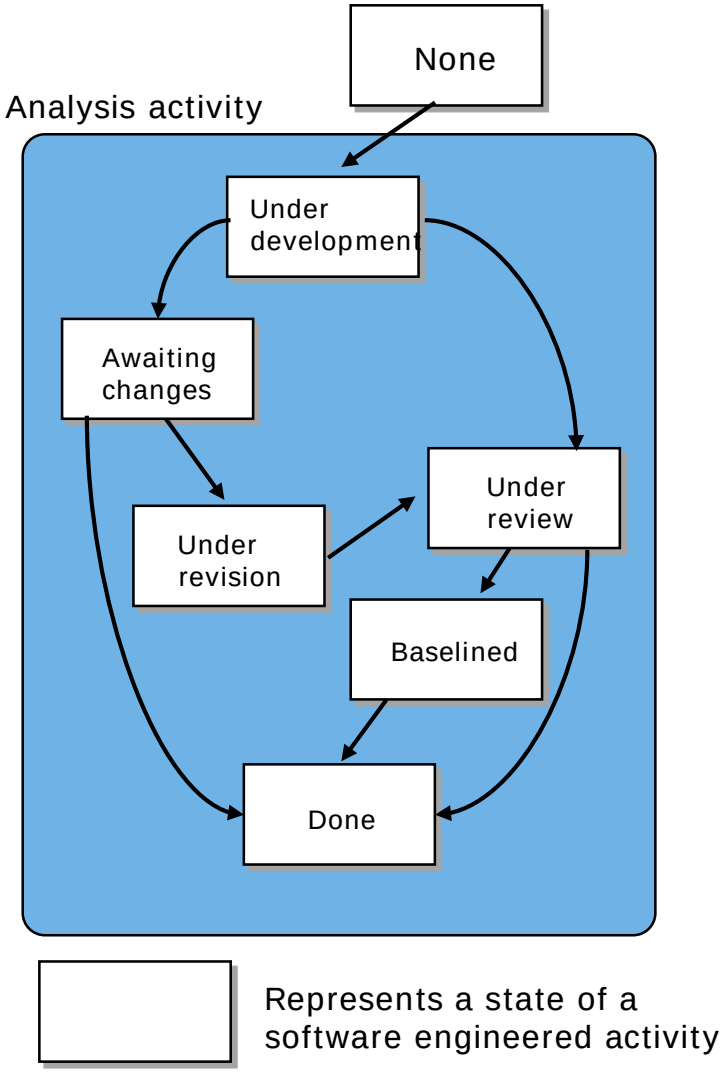
State Charts

Triggers for Transition

Examples

- Client – Server
- OBUS

# Concurrent Development Model




Roger S. Pressman, Software Engineering, A Practitioner's Approach

# Are These Different?

Different names for traditional?

Does it matter?

What do you as project managers need to take away from this?

# Current State of the Art

Iterative, cyclic development (or so stated)

Agile Processes?

Software is grown rather than birthed whole

Short cycles

Small teams

Component development

More integration vs new development?

# When Looking at a New Project

DO NOT make your project fit a SDLC!!!

INSTEAD, find the right SDLC and tailor it to your project (if it can be).

Your organization may drive this, but any lifecycle process should be seen as a tool to assist development, not an end in and of itself.

# Questions?


# Module 3: Project Processes

(Developed by Dan Shoemaker)

Introduction to Assured Software Engineering

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

# Notices



This material is based upon work funded and supported by the Independent Agency under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center sponsored by the United States Department of Defense.

# Project Planning Process – Purpose

To produce and communicate effective and workable project plans

This process

- Determines the scope of the project management and technical activities
- Identifies process outputs, project tasks and deliverables
- Establishes schedules for project task conduct, including achievement criteria
- Establishes required resources to accomplish project tasks.

# Project Planning Process – Results

From successful implementation

- The scope of the work for the project is defined.
- The feasibility of achieving the goals of the project with available resources and constraints are evaluated.
- The tasks and resources necessary to complete the work are sized and estimated.
- Interfaces between elements in the project, and with other project and organizational units, are identified.
- Plans for the execution of the project are developed.
- Plans for the execution of the project are activated.

# Project Planning Process – Project Initiation

The manager shall establish the requirements of the project to be undertaken, including identifying the project's objectives, motivations and boundaries.

The manager shall establish feasibility of the project by checking that resources are available, adequate, and appropriate and that the timescales to completion are achievable.

As necessary, and by agreement of all parties concerned, the requirements of the project may be modified at this point to achieve the completion criteria.

# Project Planning Process – Project planning

The manager shall prepare the plans for execution of the project containing descriptions of the associated activities and tasks and identification of the software products.

These plans shall include, but are not limited to

- Schedules for the timely completion of tasks
- Estimation of effort
- Adequate resources needed to execute the tasks
- Allocation of tasks
- Assignment of responsibilities
- Quantification of risks associated with the tasks or the process itself
- Quality assurance measures to be employed throughout the project
- Costs associated with the process execution
- Provision of environment and infrastructure
- Definition and maintenance of a lifecycle model

# Project Planning Process – Project Activation

The manager shall obtain authorization for the project.

The manager shall submit requests for necessary resources to perform the project.

The manager shall initiate the implementation of the project plan/s to satisfy the objectives and criteria set, exercising control over the project.

# Project Assessment and Control Process – Purpose

To determine the status of the project and ensure that the project performs according to plans and schedules, and within projected budgets, and that it satisfies technical objectives

This process includes

- Redirecting the project activities, as appropriate, to correct identified deviations and variations from other project management or technical processes
  - Redirection may include re-planning as appropriate.

# Project Assessment and Control Process – Results

From successful implementation

- Progress of the project is monitored and reported.
- Interfaces between elements in the project, and with other project and organizational units, are monitored.
- Actions to correct deviations from the plan and to prevent recurrence of problems identified in the project are taken when project targets are not achieved.
- Project objectives are achieved and recorded.

# Project Assessment and Control Process – Project Monitoring and Project Control

## Project Monitoring

- The manager shall monitor the overall execution of the project, providing both internal reporting of the project progress and external reporting to the acquirer as defined in the contract.

## Project control

- The manager shall investigate, analyze, and resolve the problems discovered during the execution of the project. The resolution of problems may result in changes to plans.
  - It is the manager's responsibility to ensure the impact of any changes is determined, controlled, and monitored. Problems and their resolution shall be documented.
- The manager shall report, at agreed points, the progress of the project, declaring adherence to the plans and resolving instances of the lack of progress

# Project Assessment and Control Process – Project assessment and Project closure

Project assessment

- The manager shall ensure that the software products and plans are evaluated for satisfaction of requirements.
- The manager shall assess the evaluation results of the software products, activities, and tasks completed during the execution of the project for achievement of the objectives and completion of the plans.

Project closure

- When all software products, activities, and tasks are completed, the manager shall determine whether the project is complete, taking into account the criteria as specified in the contract or as part of organization's procedure.
- These results and records shall be archived in a suitable environment as specified in the contract.

# Decision Management Process

Selects the most beneficial course of project action where alternatives exist

- This process responds to a request for a decision encountered during the system lifecycle, whatever its nature or source, in order to reach specified, desirable or optimized outcomes.
  - Alternative actions are analyzed and a course of action selected and directed.
- Decisions and their rationale are recorded to support future decision-making.

# Decision Management Process – Results

From successful implementation

- A decision-making strategy is defined.
- Alternative courses of action are defined.
- A preferred course of action is selected.
- The resolution, decision rationale and assumptions are captured and reported.

# Decision Management Process – Decision Planning

The project shall

- Define a decision-making strategy
- Involve relevant parties in the decision-making in order to draw on experience and knowledge
- Identify the circumstances and need for a decision
- Promote learning from experience

# Decision Management Process – Decision Analysis

The project shall

- Select and declare the decision-making strategy for each decision situation
- Identify desired outcomes and measurable success criteria
- Evaluate the balance of consequences of alternative actions, using the defined decision-making strategy, to arrive at an optimization of, or an improvement in, an identified decision situation

# Decision Management Process – Decision Tracking

The project shall

- Record, track, evaluate and report decision outcomes to confirm that problems have been effectively resolved, adverse trends have been reversed and advantage has been taken of opportunities
- Maintain records of problems and opportunities and their disposition, as stipulated in agreements or organizational procedures and in a manner that permits auditing and learning from experience

# Risk Management Process

A continuous process for systematically addressing risk throughout the lifecycle of a system or software product or service

Can be applied to risks related to the acquisition, development, maintenance or operation of a system

# Risk Management Process – Results

From successful implementation

- The scope of risk management to be performed is determined.
- Appropriate risk management strategies are defined and. Implemented.
- Risks are identified as they develop and during the conduct of the project.
- Risks are analyzed, and the priority in which to apply resources to treatment of these risks is determined.
- Risk measures are defined, applied, and assessed to determine changes in the status of risk and the progress of the treatment activities.
- Appropriate treatment is taken to correct or avoid the impact of risk based on its priority, probability, and consequence or other defined risk threshold.

# Risk Management Process – Risk Management Planning

Risk management policies describing the guidelines under which risk management is to be performed shall be defined.

A description of the Risk Management Process to be implemented shall be documented.

The parties responsible for performing risk management and their roles and responsibilities shall be identified.

The responsible parties shall be provided with adequate resources to perform the Risk Management Process.

A description of the process for evaluating and improving the Risk Management Process shall be provided.

# Risk Management Process – Risk Analysis

Risks shall be identified in the categories described in the risk management context.

The probability of occurrence and consequences of each risk identified shall be estimated.

Each risk shall be evaluated against its risk thresholds.

For each risk that is above its risk threshold, recommended treatment strategies shall be defined and documented. Measures indicating the effectiveness of the treatment alternatives shall also be defined and documented.

# Risk Management Process – Risk Treatment

Stakeholders shall be provided recommended alternatives for risk treatment in risk action requests.

If the stakeholders determine that actions should be taken to make a risk acceptable, then a risk treatment alternative shall be implemented.

If the stakeholders accept a risk that exceeds its threshold, it shall be considered a high priority and monitored continuously to determine if any future risk treatment actions are necessary.

Once a risk treatment is selected, it shall receive the same management actions as problems do, in accordance with the assessment and control activities in subclause 6.3.2 of this standard or ISO/IEC 15288:2008.

# Risk Management Process – Risk Monitoring

All risks and the risk management context shall be continuously monitored for changes.

- Risks whose states have changed shall undergo risk evaluation.

Measures shall be implemented and monitored to evaluate the effectiveness of risk treatments.

The project shall continuously monitor for new risks and sources throughout its lifecycle.

# Risk Management Process – Evaluation

Information shall be collected throughout the project's lifecycle for purposes of improving the Risk Management Process and generating lessons learned.

The Risk Management Process shall be periodically reviewed for its effectiveness and efficiency.

Information on the risks identified, their treatment, and the success of the treatments shall be reviewed periodically for purposes of identifying systemic project and organizational risks.

# Configuration Management Process – Purpose and Results

Purpose

- To establish and maintain the integrity of all identified outputs of a project or process and make them available to concerned parties

Results from successful implementation

- A configuration management strategy is defined.
- Items requiring configuration management are defined.
- Configuration baselines are established.
- Changes to items under configuration management are controlled.
- The configuration of released items is controlled.
- The status of items under configuration management is made available throughout the lifecycle.

# Configuration Management Process – Planning and Execution

Planning

- The project shall define a configuration management strategy.
- The project shall identify items that are subject to configuration control.

Execution

- The project shall maintain information on configurations with an appropriate level of integrity and security.
- The project shall ensure that changes to configuration baselines are properly identified, recorded, evaluated, approved, incorporated and verified.

# Information Management Process

Provides relevant, timely, complete, valid, or confidential information to designated parties

This process

- Generates, collects, transforms, retains, retrieves, disseminates and disposes of information
- Manages designated information, including technical, project, organizational, agreement and user information

# Information Management Process – Results

From successful implementation

- Information to be managed is identified.
- The forms of the information representations are defined.
- Information is transformed and disposed of as required.
- The status of information is recorded.
- Information is current, complete and valid.
- Information is made available to designated parties.

# Information Management Process – Planning

The project shall

- Define the items of information that will be managed during the system life cycle and, according to organizational policy or legislation, maintained for a defined period beyond
- Designate authorities and responsibilities regarding the origination, generation, capture, archiving and disposal of items of information
- Define the rights, obligations and commitments regarding the retention of, transmission of and access to information items
- Define the content, semantics, formats and medium for the representation, retention, transmission and retrieval of information
- Define information maintenance actions

# Information Management Process – Execution

The project shall

- Obtain the identified items of information
- Maintain information items and their storage records according to integrity, security and privacy requirements
- Retrieve and distribute information to designated parties as required by agreed schedules or defined circumstances
- Provide official documentation as required
- Archive designated information, in accordance with the audit, knowledge retention and project closure purposes
- Dispose of unwanted, invalid or unverifiable information according to organization policy, and security and privacy requirements

# Measurement Process

Collects, analyzes, and reports data relating to the products developed and processes implemented within the unit

Supports effective management of the processes, and objectively demonstrates the quality of the products

# Measurement Process – Results

From successful implementation

- The information needs of technical and management processes are identified.
- An appropriate set of measures, driven by the information needs are identified and/or developed.
- Measurement activities are identified and planned.
- The required data are collected, stored, analyzed, and the results interpreted.
- Information products are used to support decisions and provide an objective basis for communication.
- The Measurement Process and measures are evaluated.
- Improvements are communicated to the Measurement Process owner.

# Measurement Process – Planning

The project shall

- Describe the characteristics of the organization that are relevant to measurement
- Identify and prioritize the information needs
- Select and document measures that satisfy the information needs
- Define data collection, analysis, and reporting procedures
- Define criteria for evaluating the information products and the measurement process
- Review, approve, and provide resources for measurement tasks
- Acquire and deploy supporting technologies

# Measurement Process – Performance

The project shall

- integrate procedures for data generation, collection, analysis and reporting into the relevant processes
- Collect, store, and verify data
- Analyze data and develop information products
- Document and communicate results to the measurement users

# Measurement Process – Evaluation

The project shall

- Evaluate information products and the measurement process
- Identify and communicate potential improvements

# Questions?


# Module 4: Process Frameworks

(Developed by David Root)

Introduction to Assured Software Engineering

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

# Notices



This material is based upon work funded and supported by the Independent Agency under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center sponsored by the United States Department of Defense.

# Topics

Define Process/Method/Framework

PSP/TSP

Architecture-Centric Development Method (ACDM)

Discuss Agile concepts

- XP and Scrum
- Rational Unified Process (RUP)
- Agile Unified Process (AUP) and Open Unified Process (OUP)

How do you really use these processes?

# Discussion

In this lecture process and method are interchangeable.  Should they be?

Also, the assumption is that you have a starting framework.

# Defining Processes – A Review

When defining processes

- Be sure that you know why you are using/developing a process.
- **Ensure that processes are in line with business goals.**
- Involve stakeholders: They should *develop the process*; you should *facilitate.*
- Be sure that the granularity is appropriate for the organization/program/project.

# Don’t Make This Too Hard

Define what you are/will be doing.

- What you need to do vs everything you might want to do

It does not have to be a book.

- Checklists can suffice; must be understandable and usable

Think of metrics.

- How will you know you did it?
- Data collection

Do you need to measure “how well?”

- Or just that you did it. You decide.

# Be Very Careful

If adopting a process framework then

**DO SO.**

Don’t immediately “tailor” the process.

Don’t just pick and choose specific parts of different frameworks.

More overhead costs aren’t necessarily bad.

# Painful Experience

If you use a process framework to establish or improve processes

- Understand and follow the spirit of the framework, not the blind letter of the law.
- Use the framework as-is before you tailor it.
- Tailor, measure, tailor, measure...
- THINK about what you are doing.
- It’s better to start with more.
  - Too easy to justify too little

# Software Methodology Wars
# Ken Orr/Cutter Consortium

**Question:**

*What is the difference between a bank robber and a methodologist?*

**Answer:**

*You can negotiate with a bank robber.*

# Review: Remember Process ≠ Lifecycle

Software process is not the same as lifecycle models.

- Process refers to the specific steps used in a specific organization to build systems.
- Process indicates the specific activities that must be undertaken and artifacts that must be produced.
- Process definitions include more detail than provided lifecycle models.

Software processes are sometimes defined in the context of a lifecycle model.

# Process Definition

Benefits of Process Definition

- Being able to make changes (recipe)

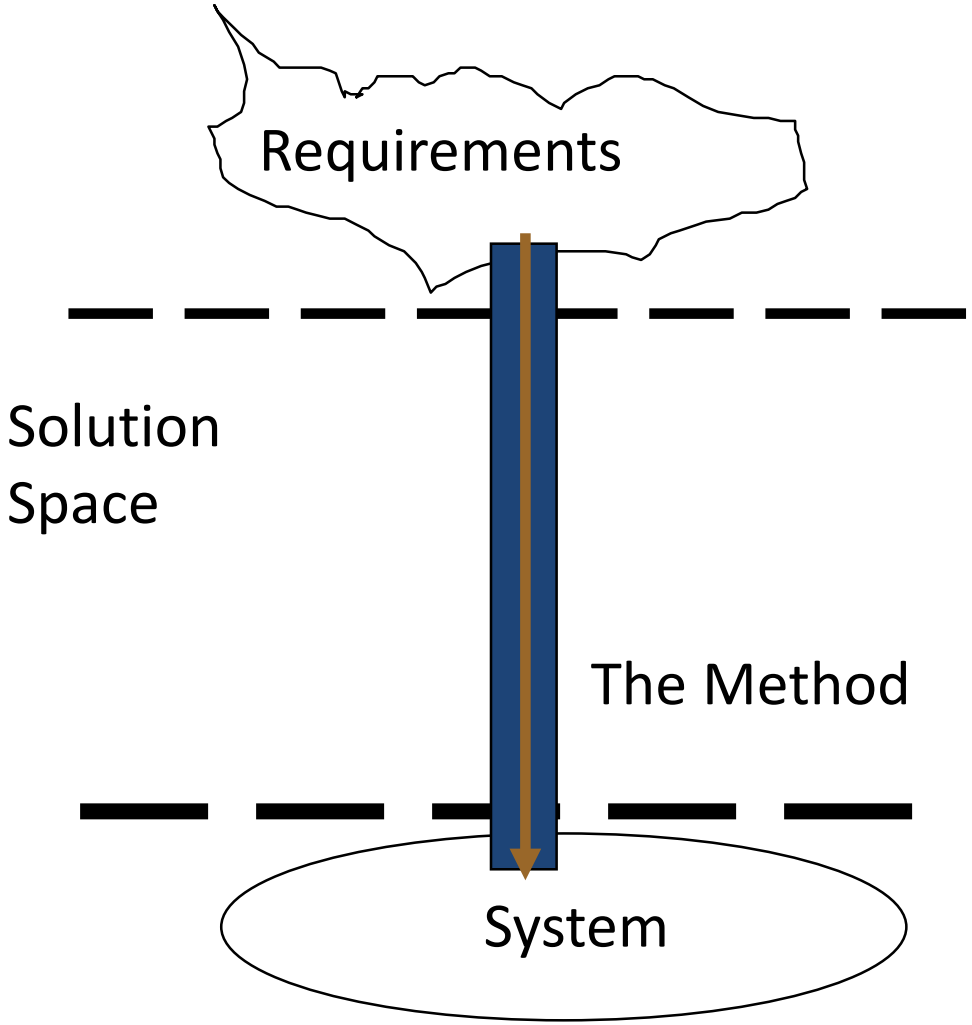
Process Components

- Scripts
- Forms
- Standards
- Process improvement capability

Defining Phases in the Process (ETVX)

- Entry, Task, Validation and eXit

# What Is a Method?



# Our Philosophy

There is no one method suitable for all problem domains.

All good methods are based on timeless principles like abstraction and information hiding.

# Remember…

The only source of defects in software development is the human element.

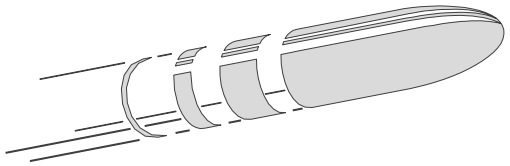
Processes are needed to

- Control the human variable
- Identify problem sources
- Make outcomes repeatable

But, can you have too much, or too little process? How would you know?

# Why Would I Want To Use an Established Process Framework?

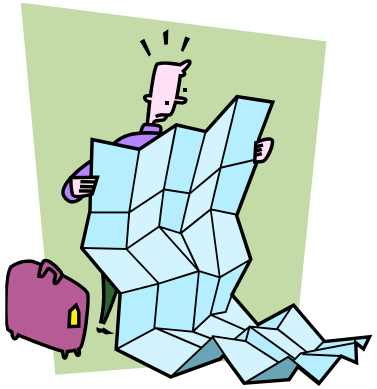
# Process Myths and Abuses -1



Belief that any one model is ***the*** Silver Bullet

Mandating processes from above without involving process owners

Beginning a process improvement effort without a baseline of current practices



# Process Myths and Abuses -2

Unwillingness or inability to interpret, tailor, or apply judgment regarding a maturity model in light of business needs

- Undertaking process improvement without consideration of business goals
- Following the “letter of the law” instead of the “intent of the law”

# Process Myths and Abuses -3

The assumption that high quality processes automatically mean high quality designs, code, and implementations

- Chances are good that the quality of these artifacts will be better, but there is no guarantee.

# Process Myths and Abuses -4

The assumption that low maturity organizations will automatically produce low quality designs, code, and implementations

- Successful organizations that have low maturity processes typically have lots of virtuosos.
- Often these organizations produce reasonable, even innovative systems, but the results are unpredictable.

# Process Myths and Abuses -5

High-maturity-level organizations are guaranteed to enjoy high profitability.

- Royal Enfield example…improved 1950's design

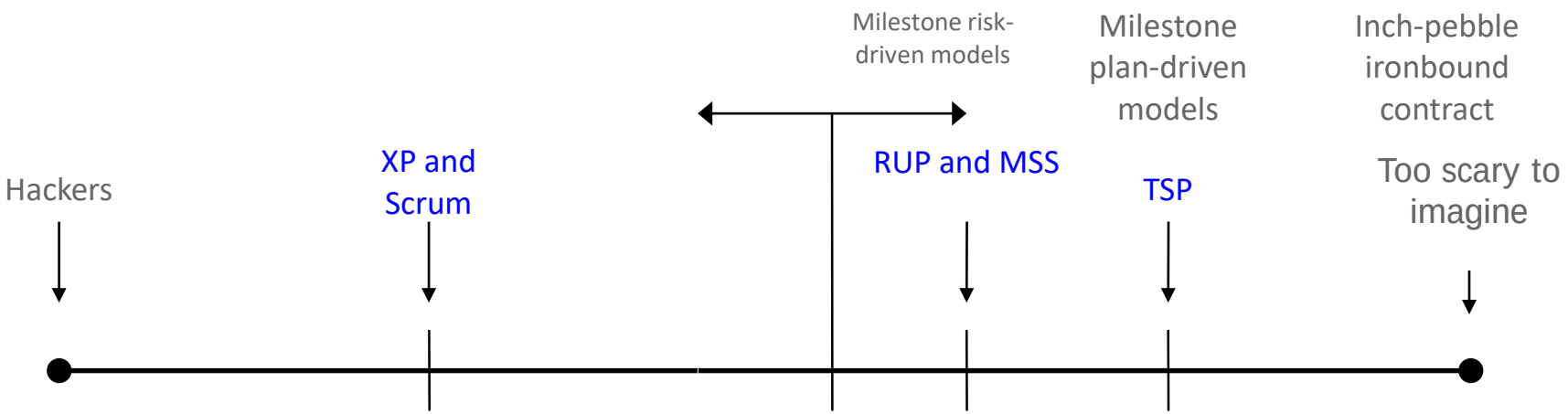

High maturity can only be achieved through high "ritualization."

- Red Bead experiment

# Sample Processes

# Process Spectrum

Weight = amount of project overhead/code




"Light"

Artifact Challenged

"Heavy"

Artifact Heavy

Adapted from Justin Rockwood, "Choose your Weapon Wisely," 2003

# As We Look at These, Ask Yourself…

Can the “best” techniques be combined?

Can weaknesses be mitigated?

Do they tell you “everything” you need to know (e.g., requirements elicitation)?

You can determine if the process benefit outweighs its costs (what costs?).

More important:

***What are the important aspects of your project?***

# Questions?


# Module 5: PSP and TSP

## (Developed by Mel Rosso Llopart)

Introduction to Assured Software Engineering

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

# Notices



This material is based upon work funded and supported by the Independent Agency under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center sponsored by the United States Department of Defense.

# Who Studies PSP

Students

Software engineers

Managers (people) who care about quality

People who want to control their time

People who want to understand the value of data collection

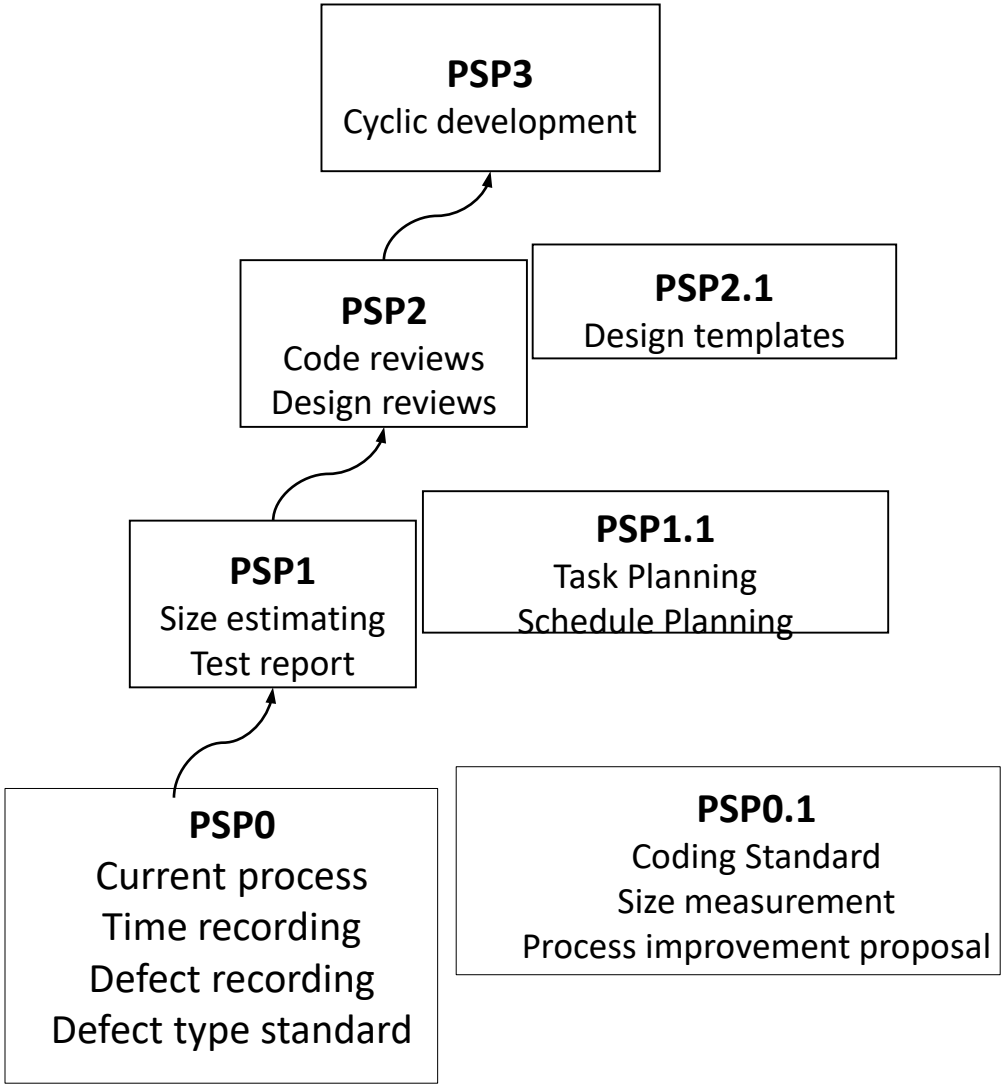
# PSP Framework

Cyclic Personal Process

Personal Quality Management

Personal Planning Process

Baseline Personal Process



# Why PSP?

All improvements stem from personal practice

- “no one should do intellectual work in a particular way,” but…

The Capability Maturity Model (CMM) defines corporate processes.

PSP is unique to individuals.

PSP is a path to excellence.

- “Level five for individuals”

# What Is the Big PSP Strategy?

Identify effective software development practices that can be used by individuals.

Define them in a form usable in small programs (or in cycles).

Introduce the concepts by performing graduated exercises.

# PSP Concentrates on Metrics with Highest ROI

Reducing overall defect rates

Spending more time up front in the development cycle

- To gain more time at the end

Eliminating or nearly eliminating compile and test defects

Accurately estimating the time it takes to build software

# What You Learn About PLANNING

Planning is essential.

Taking more time up front means you will save time later, but...

- Better use of resources
- How to make a plan

# What You Learn About ESTIMATION

You must estimate the size of an effort.

There are many ways to estimate.

- Volume metrics (LOC)
- Function metrics (FP)

PSP teaches you the Probe estimation method.

- Find a proxy and use that to get a value.

You get better at estimating the more you try.

# To Summarize: PSP

A good process helps produce a good product.

Design and code reviews have a greater positive effect on quality than any other activity.

Taking time up front means less time in the end.

You cannot improve without measurement.

Improving the software process begins with the least common denominator…

… the software developer!

# PSP Is a First Step to Improvement, but ...

Team Software Process (TSP) can be used after PSP to improve teams.

PSP is needed for the Team Software Process (TSP).

TSP is one method by which projects can improve.

- A definite way to get a team moving

# Team Software Process (TSP)

Defined framework for team software engineering

- provides balanced emphasis on process, product, and teamwork
- stresses the use of software engineering and process principles in a team-working environment
- defines roles and responsibility for each team member

# Principal Concepts of TSP

Provide process framework for small teams

Develop products in several cycles

Establish standards for quality and performance

Provide measurements for the team

Use role and team evaluations

Require process discipline

Provide guidance on solving team problems

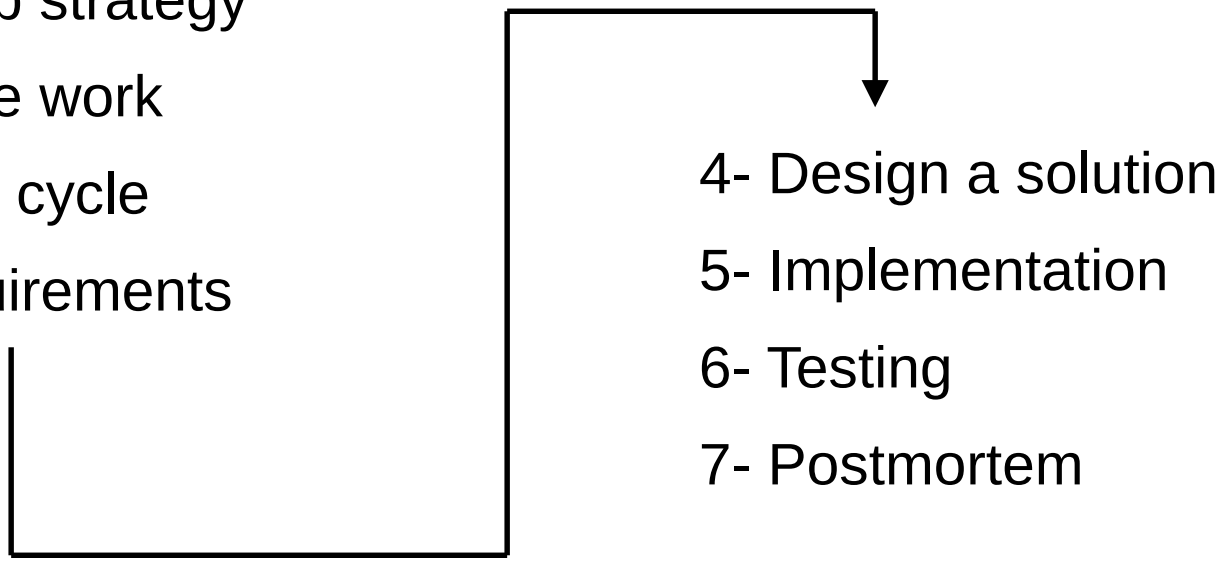
# TSP Cycle

0- Launch

1- Develop strategy

2- Plan the work

3- Review cycle requirements



4- Design a solution

5- Implementation

6- Testing

7- Postmortem

Next Launch

# Step 0: Project Launch and Step 1: Develop a Strategy

0- Project Launch

- Projects begin with a project launch.
  - Introduce the overall product objectives and criteria for success.
  - After launch, the seven steps begin.

1- Develop Strategy

- Review project goals and planning schedule.
- Agree on cycle objectives and criteria for success.
- Assess risks.

# Step 2: Plan the Work

Overall project plan

- use standard SPMP, review each cycle

Plan cycle activities

- size estimate
- resource estimate
- schedule estimate
- establish quality goals
- implementation goals

# Step 3: Review Cycle Requirements

Project requirements document

- use standard SRS, review each cycle

Cycle requirements

- decide which requirements will be satisfied
- identify test methods for each requirement

# Step 4: Design a Solution

System-level architecture and design

Cycle design

- map requirements to design abstractions
- ensure that design is consistent with system-level design and architectures
- ensure that cycle products can be integrated with overall product

# Step 5: Implementation

System-level implementation

- component construction and integration

Cycle-level implementation

- detailed component/module design
- component/module code
- component/module inspection

# Step 6: Testing

System-level test

- develop system-level test plan, quality review
- develop quality standards and goals

Cycle-level test

- develop component/module test plans
- develop partial system test plans

# Step 7: Postmortem

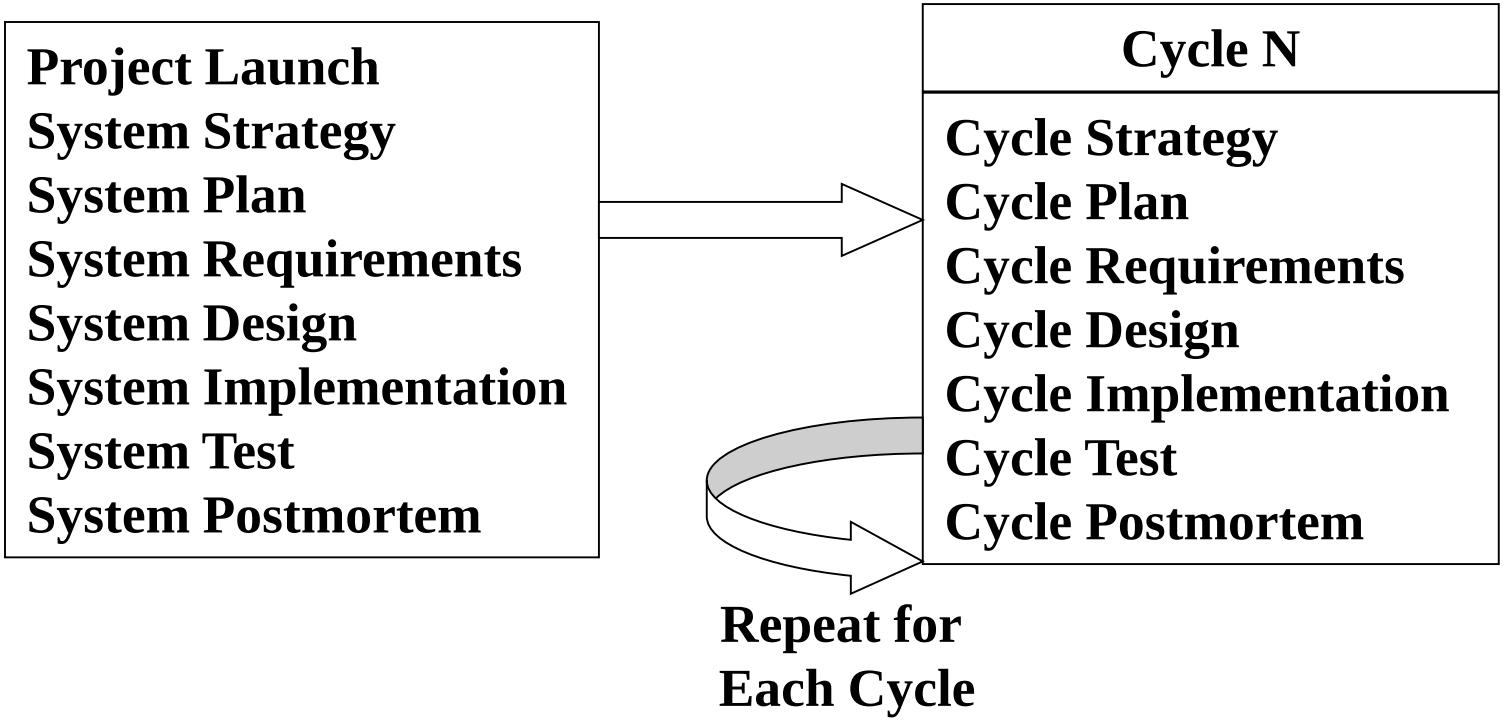
System- and cycle-level postmortem

- review performance data
- review quality data
- conduct role evaluations
- identify opportunities for improvement
- ensure all items for project/cycle are under CM control

# Implementing TSP

Actual implementation will vary.

Here is a practical overview...



# Each step of the process has

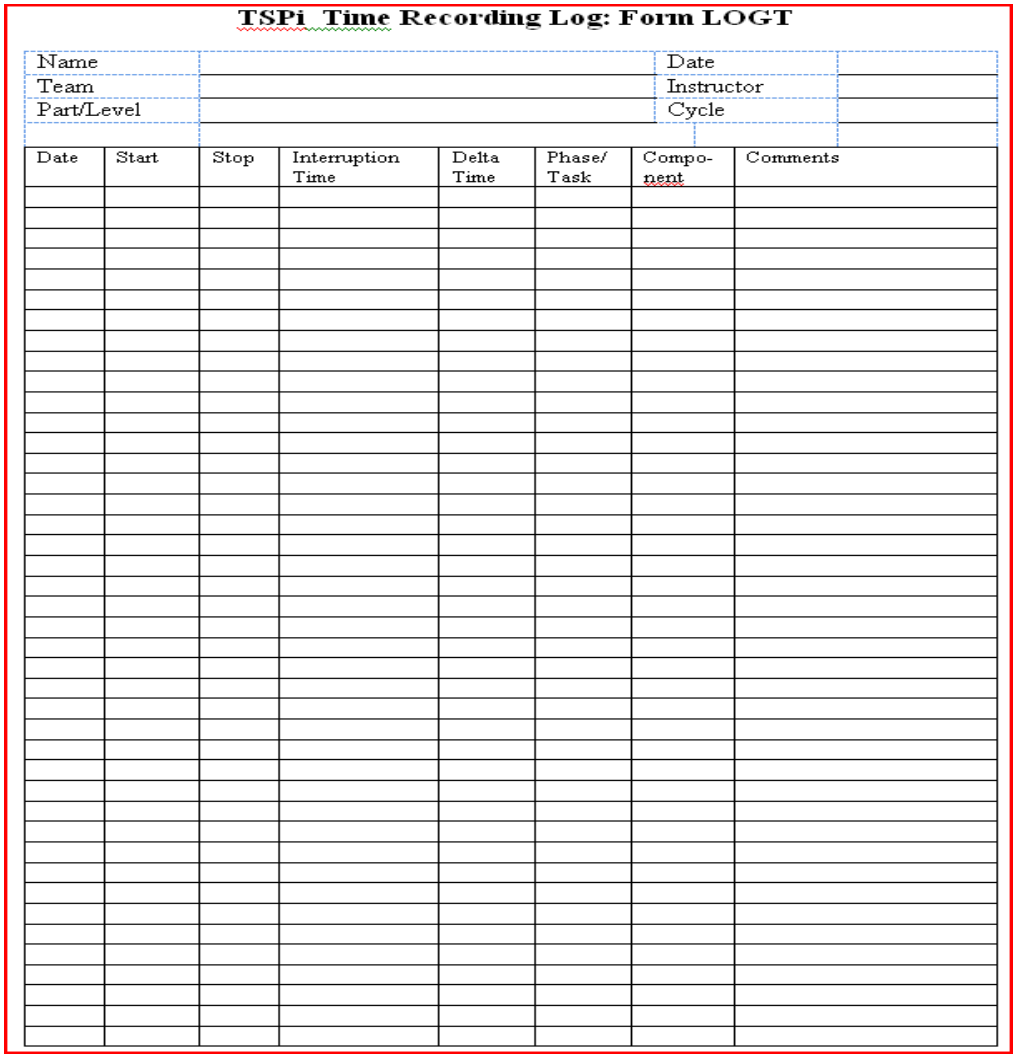
A script with

- entry criteria
- the tasking that must be done
- evaluation of how you know it is done
- exit artifacts that should exist

Forms that help you collect data about the process

# Sample Form

Available as an Excel workbook



**TSPi Time Recording Log: Form LOGT**

| | | | |
|---|---|---|---|
| Name | | Date | |
| Team | | Instructor | |
| Part/Level | | Cycle | |

| Date | Start | Stop | Interruption Time | Delta Time | Phase/ Task | Compo-nent | Comments |
|---|---|---|---|---|---|---|---|
| | | | | | | | |

# SAMPLE TIME RECORDING LOG INSTRUCTIONS

| | |
|---|---|
| Purpose | This form is for recording the time spent in each project phase.<br>These data are used to complete the Project Plan Summary. |
| General | - Record all the time you spend on the project.<br>- Record the time in minutes.<br>- Be as accurate as possible.<br>If you need additional space, use another copy of the form. |
| Header | Enter the following:<br>- Your name<br>- Today's date<br>- The instructor's name<br>- The number of the program<br>If you are working on a non-programming task, also enter a job description in the Program# field. |
| Date | Enter the date when the entry is made. |
| Example | 10/18/2013 |
| Start | Enter the time when you start working on a task. |
| Example | 8:20 |
| Stop | Enter the time when you stop working on that task. |

# TSP Roles

TSP prescribes roles for each person on the team, their activities, and their goals.

- Team Leader
- Development Manager
- Planning Manager
- Quality/Process Manager
- Support Manager

# Example of TSP Roles: Leader

Team Leader goals

- build and maintain an effective team
- motivate all team members to work aggressively on the project
- resolve all the issues brought to you by team members
- keep managers informed on progress
- act as an effective meeting facilitator for the team

# TSP Users

U.S. Navy

Microsoft

Xerox

Bechtel-Bettis

Advanced Information Services

# PSP/TSP Reviewed

Tool Support:

http://processdash.sourceforge.net/

# Questions?


# Module 6: Architecture-Centric Development Method (ACDM)

(Developed by Dan Shoemaker)

Introduction to Assured Software Engineering

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

# Notices



This material is based upon work funded and supported by the Independent Agency under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center sponsored by the United States Department of Defense.

# ACDM Architecture-Centric Development Method

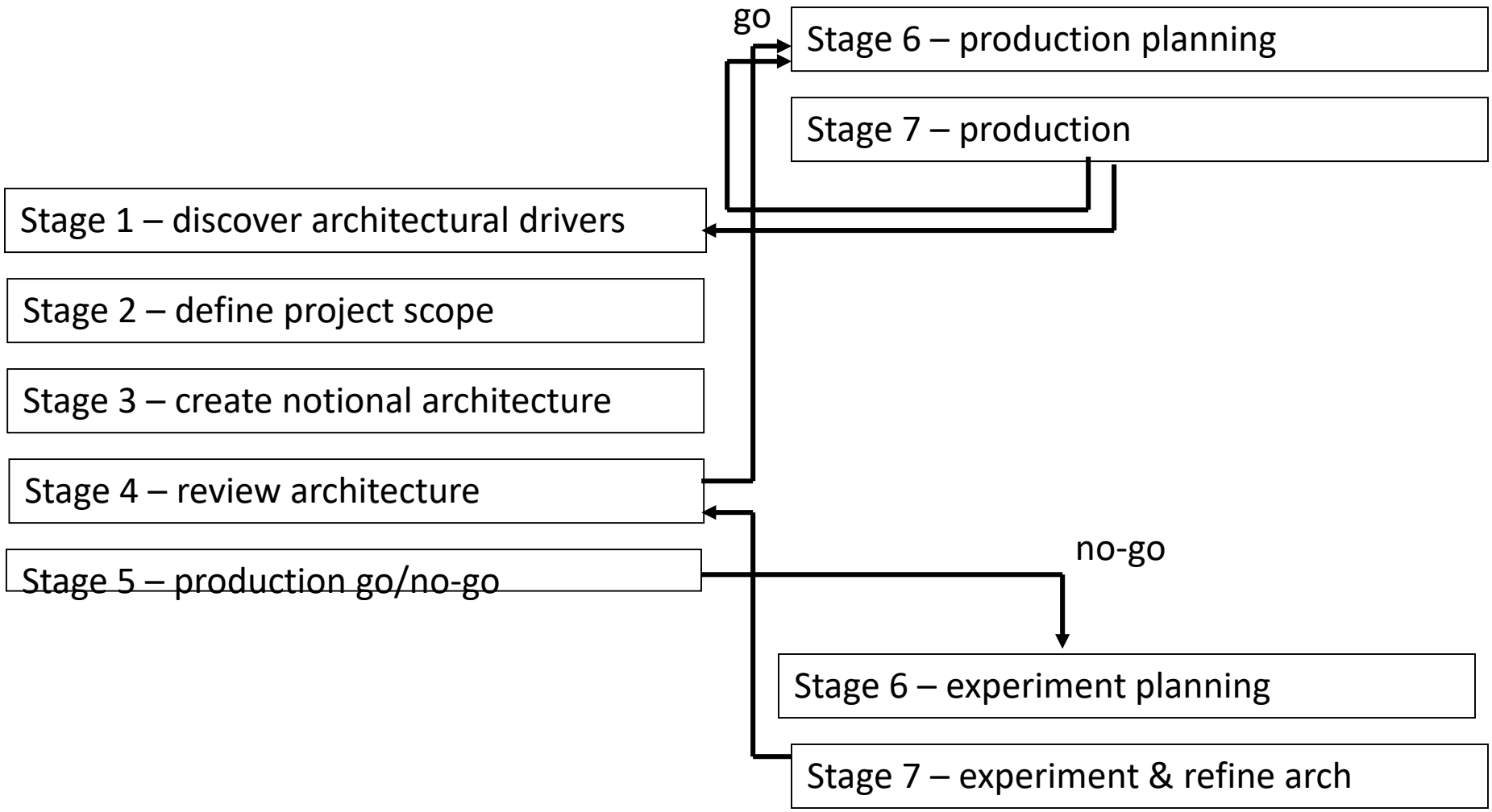
http://reports-archive.adm.cs.cmu.edu/isri.html

ACDM is an *iterative* development method.

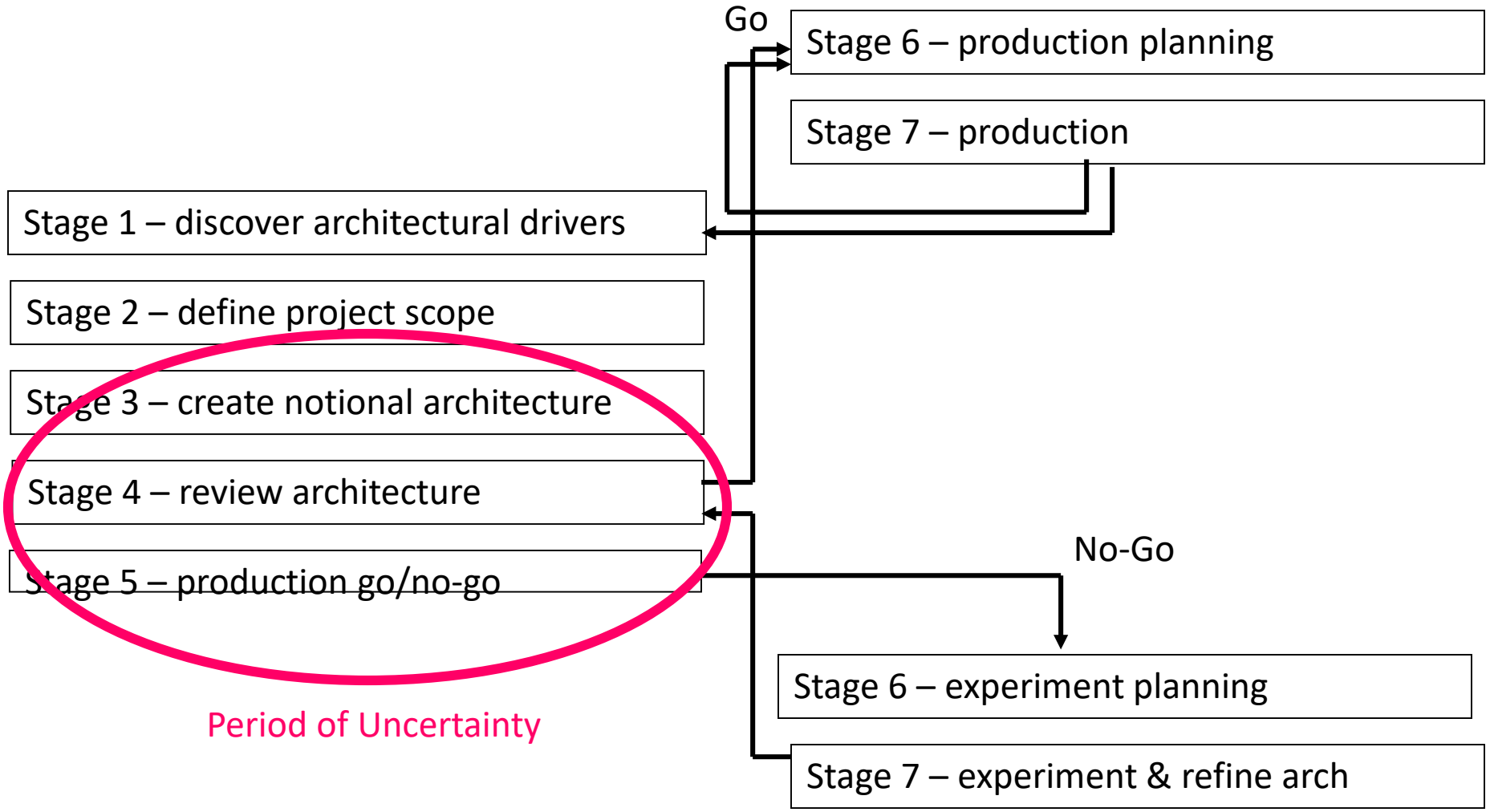
- iteratively refines and reviews the architecture until it is deemed fit for the purpose
- permits iteration in the production of the elements/system/products

ACDM has seven stages in the development method.

# ACDM Stages -1



# ACDM Stages -2



# ACDM Strengths

Product and architectural focus

*Agile and relatively lightweight*

Structured but flexible and tailorable

Iterative

Derived from CMU graduate student practitioners

Provides guidance for roles, activities, and artifacts

Derives requirements (architectural drivers) from business drivers

# ACDM Weaknesses

Design process (missing other phases?)

- Research to combine with other process frameworks

Industrial experience or data

- New but gaining data all the time

Unclear how well ACDM scales up to large projects (looks good though)

Still maturing

Limited tool support and templates

Requires a relatively good understanding of architectural concepts

# ACDM Centerpiece

Architecture

- Complexity/scope driving need for more abstraction
- Key to describing and predicting quality attributes
- Lots of development and research
- Easily misunderstood

# Summary

Process/methods defined

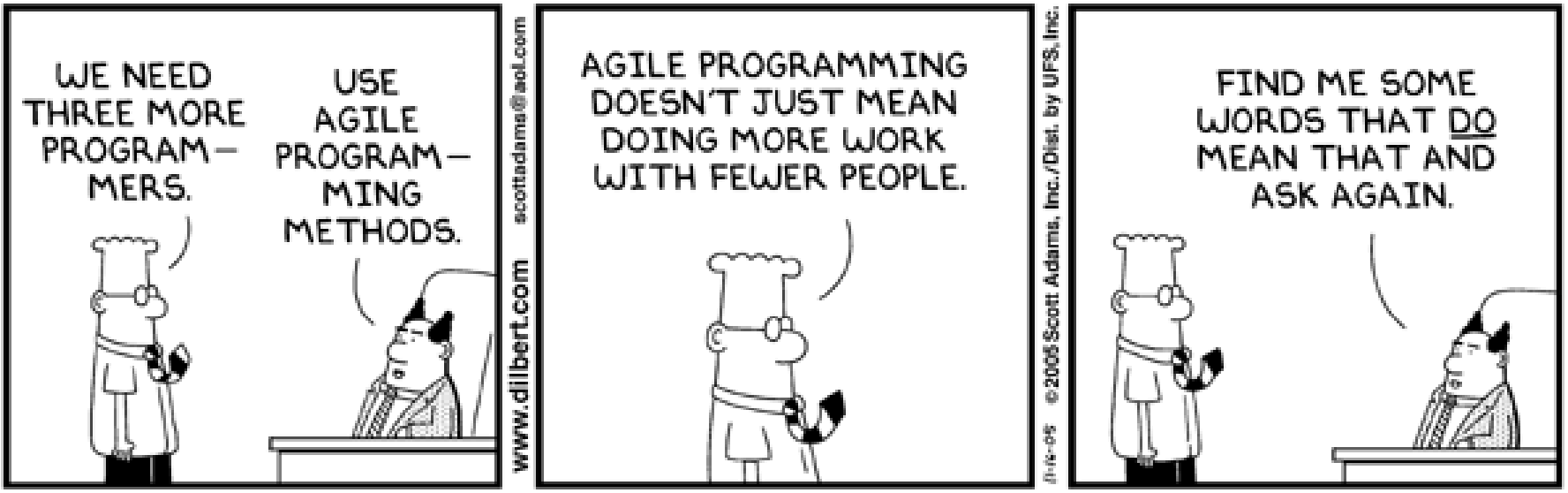
Process frameworks

Process problems and myths

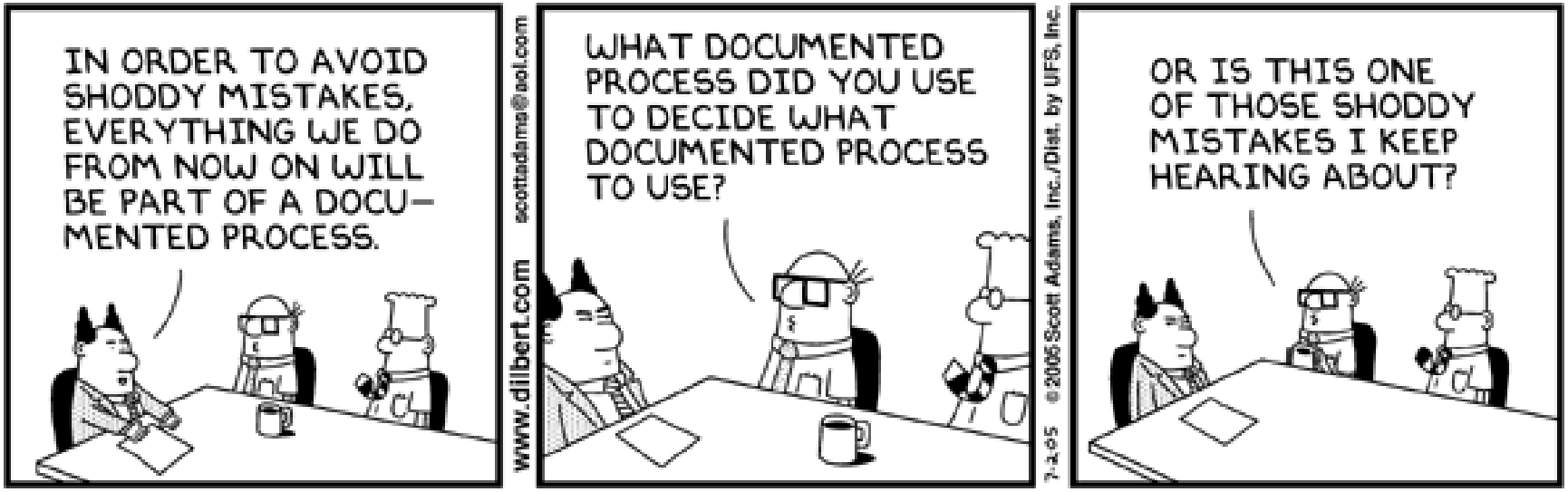
PSP, TSP, and ACDM

# Agile Processes XP and Scrum

(Developed by David Root)




© Scott Adams, Inc./Dist. by UFS, Inc.




© Scott Adams, Inc./Dist. by UFS, Inc.

# What Is Agile?

Webster's Dictionary:

"Marked by ready ability to move with quick easy grace"

Alistair Cockburn (as applied to software development):

"Ability to change development in response to changing requirements"

# Why Agile Processes?

What Agile proponents say:

- Flexibility
  - Market changes
  - Technology changes (Moore's Law)
  - Unclear requirements
- More coding, less paper
- Higher quality, quicker

# But, opponents say…

No plan, no structure

- Architecture?
- Easily derailed
- Focus on short term makes teams lose sight of final goal

Inefficient use of developers

- pair programming

No documentation

Unrealistic customer involvement

# One Data Point

“ More than two thirds of all corporate IT organizations will use some form of agile software development process in the next 18 months.” Giga Information Group Inc., 2002

Cutter Report “Agile vs. Heavy”

Use is increasing.

# Agile Processes
## From Agile Alliance

**XP**

**Scrum**

Crystal

Feature Driven

Open Source Software Develop

**RUP**

Dynamic Systems Develop Method

Adaptive Software Develop

Synch and stabilize

Agile Modeling

Pragmatic Programming

# Common Characteristics -1
From Agile Alliance

Individuals and Interactions **over** Processes and Tools

- Team dynamics
  - experience mix, team size
- Physical workspace, communality, meetings

Working Software **over** Comprehensive Documentation

- Code primary artifact
- Iterative (subscription)
- Value to the customer
- QA inherent

# Common Characteristics -2
## From Agile Alliance

Customer Collaboration **over** Contract Negotiation

- Customer onsite (involved/knowledgeable)
- Requirements-centric
- Rapid return of perceived value
- Customer expectation management

Responding to Change **over** Following a Plan

- Developer/customer team
- Emergent requirements
- Short iterations
  - Smaller changes
- Adaptation

# Weaknesses

Communication is critical

Projects with non-decomposability/coupled functionality

Scalability

Architecture?

Reliance on corporate knowledge

- Document at end

“Green field” development vs. legacy extension or modification

**Used as an excuse to not do process**

# More Weaknesses…

Maintenance

Long lifecycle

Centralized management control

“Big” specifications

Required documentation

- Safety critical

Non-flexible work environment

- Distributed development?

Fixed price and scope

# Agile Users

Microsoft

Thoughtworks*

Valtech Technologies

RADsoft

Boeing

- < 5 on a team

Google

# Minimal Research on Agile Methods … Why?

There is research on specific techniques.

Small Scale comparisons

- No “normal” size projects

Usually academically based

Short term…

Some studies are aging.

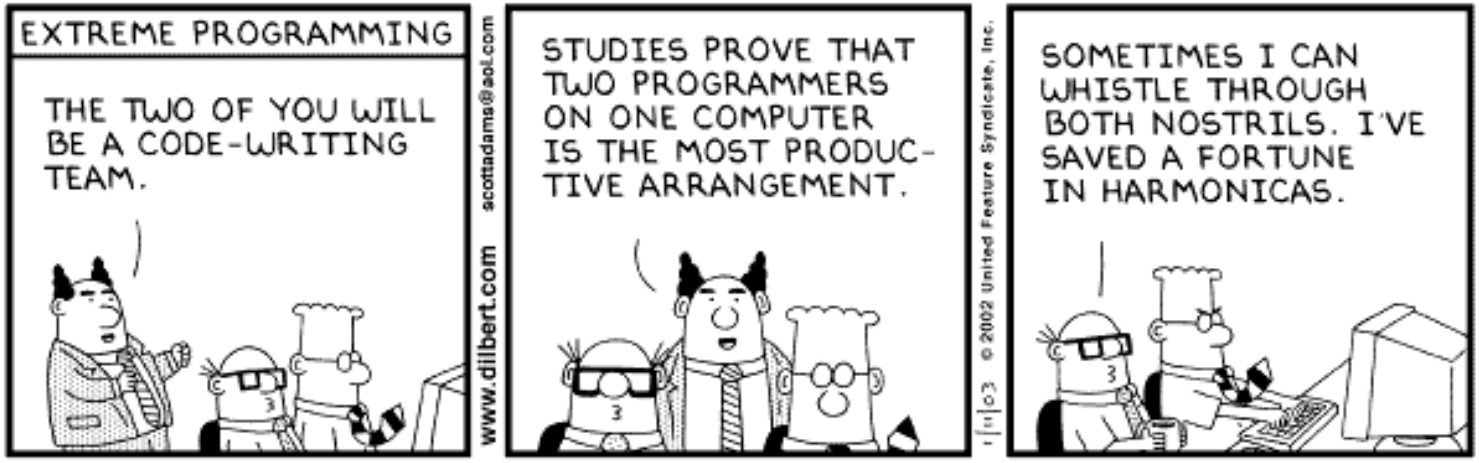
- Changes in technology?
- Changes in theory (architecture)

# eXtreme Programming

Background

- Kent Beck in the 90s
- Primary focus was from risk
  - Schedule slips
  - Rapid changes
  - Business drivers misunderstood
  - Defects
- Taking programmer strengths to extreme



# Four Values of XP

Communication

- Source of most problems

Simplicity

- Less complexity, fewer problems

Feedback

- Customer (or representative) onsite

Courage (to experiment or change code)

# XP Practices

*Pair programming*

Collective ownership

Continuous integration

40 hour week

Customer onsite

*Planning game*

Small releases

Metaphor

Simple design

*Testing - TDD*

Refactoring

# Planning Game

Use “stories”

- Different than scenarios?
- Customer understanding

Onsite customer

- Immediate feedback – both ways
  - Problems
  - Correct functionality, etc.

Prioritization – value and difficulty

How does this work with “green field”?

# Test-Driven Development (TDD)

Write tests first

- Until current code fails test
- Better focus

Auto test suite – regression testing

- Any additions, changes, etc. must pass tests
- Annoying for small changes?
- Time/resources as much as coding
- Nice deliverable with code? Maintenance?

# Pair Programming

Controversial – looking over the shoulder

Output

Quality – inspection on the fly

Corporate knowledge

Right pairing?

Most studies from academia

- On average about same quantity with higher quality

# XP Roles

Programmer/Tester

- Write tests, then code

Customer

Tracker – data collection and analysis

- Velocity

Coach – process guru

Consultant – technical expert

“Big Boss” – final decisions

# XP Workplaces

Co-located

- Some distributed

Open – no cubicles

- White boards

Hiding and relaxing places – “decompress”

Food (rewards, like M&M’s)

# Where It Seems to Work Best

Small teams 5-10

Some XP experience

- Excuse to hack?

Extension of existing application

Low need for documentation, tracking

Onsite customer

# Scrum

More management focus

Iterative – sprints

Works well with XP practices

Pig and chicken roles (ham and eggs)

- Committed or just participating
- Product owner, team, and ScrumMaster
- End users, managers

# Meetings – Time Limited

Sprint planning every cycle

- Select work
- Estimation for work – sprint backlog

Daily scrum – 15 minutes max

- Standup
- What was (yesterday) and is to be done (today)
- Problems

Scrum of scrums – coordinating teams

- Does this really work?

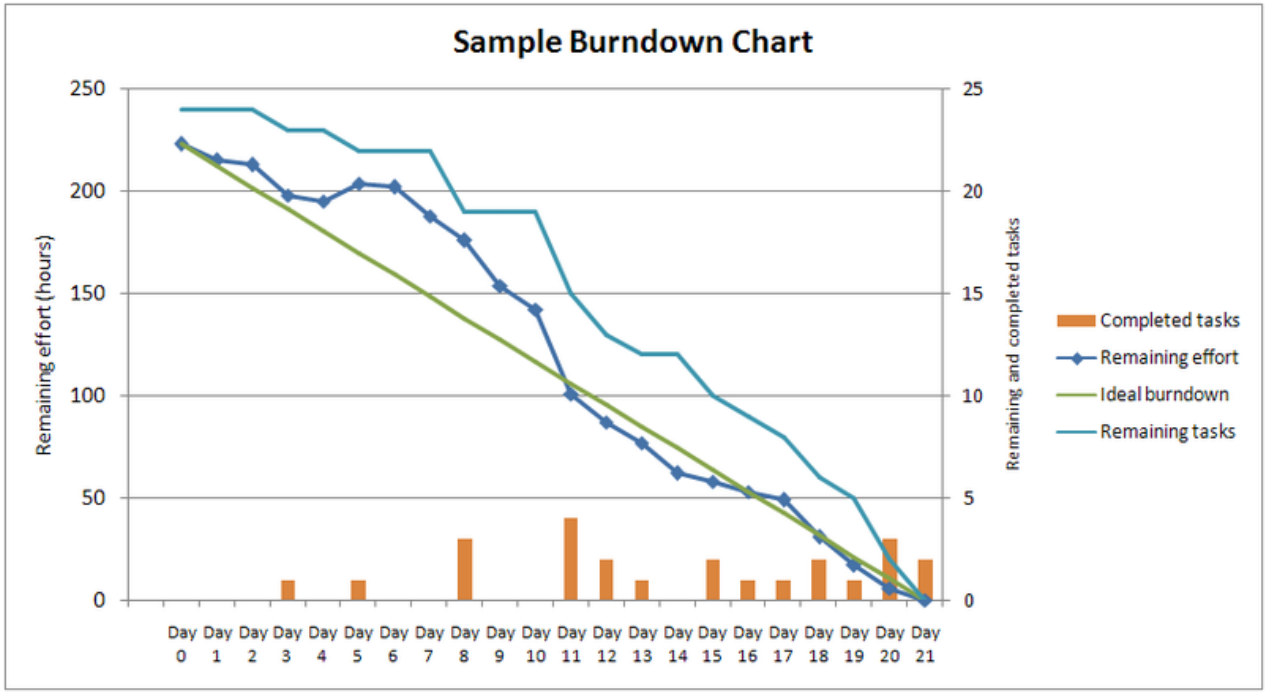
Reviews

# Artifacts

Product backlog – entire project tasks

Sprint backlog – that sprint's tasks

Burndown chart




http://en.wikipedia.org/wiki/File:SampleBurndownChart.png

# Some criticisms of Agile

Too easily used as an excuse to not have formal process

- Need discipline, or experienced coach

Tendency toward short term view

- Too easy to push off tasks to next cycle

Project environment critical

- Need for documentation?

# SPAWAR Reference for Agile

RITE Agile Incremental Development Process v1.1 010613 UPDATED

# Questions?


# Module 7: Rational, Agile, and Open Unified Processes (RUP, AUP, OUP)

(Developed by David Root)

Introduction to Assured Software Engineering

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

# Notices



This material is based upon work funded and supported by the Independent Agency under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center sponsored by the United States Department of Defense.

# Topics

RUP, AUP, OUP

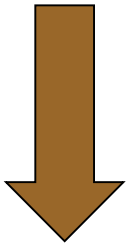
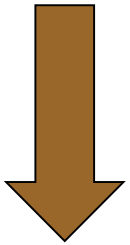
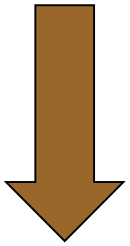
- Rational Unified Process, and related Agile and Open Unified Processes

Agile processes

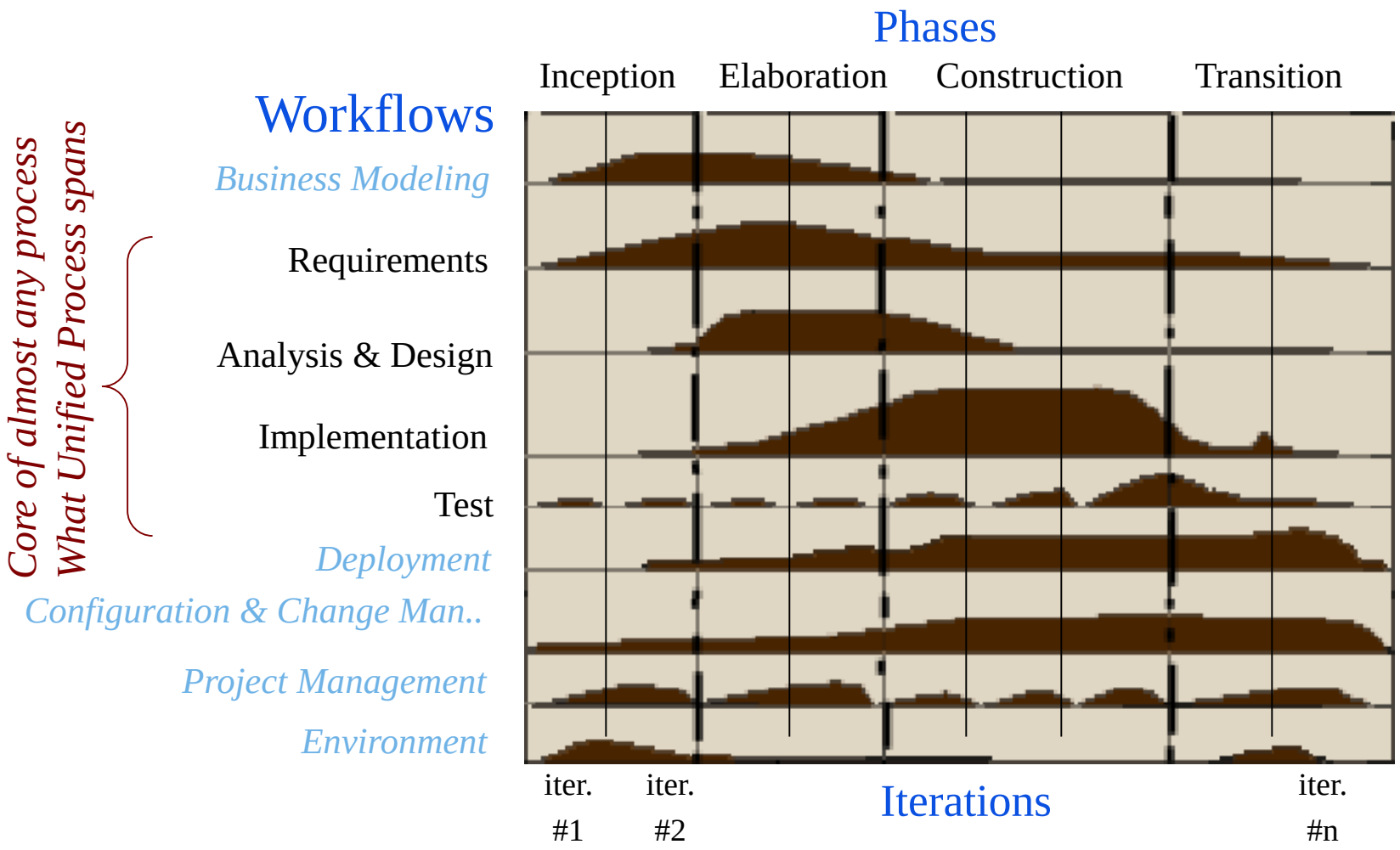
- XP and Scrum

Choosing a process

# Rational Unified Process

| *Rational* | *Unified* | *Process* |
| --- | --- | --- |
|  |  |  |
| Idea of rationalizing the common practices.<br>Result of the industrialization effort | *Communication*<br>Unified Modeling Language | Use-case-driven<br>Architecture-centric<br>Iterative<br>Incremental |

# The Process Outline - Phases Versus Iterations



# Characteristics

Iterative and incremental

Semi-formal

- UML

Generic framework rather than specific

OO process approach

Modeling -based

Use-case-driven

Component-based

Architecture-centric

# RUP Modeling

Abstractions to understand domains

- Use case
- Analysis
- Design
- Deployment
- Implementation

# Planning

Development – phase plans

- Iteration plans

Monitoring plans

- Measurement
- Risk
- Problems
- Acceptance

# RUP Roles (about 26 of them)

Developer Worker Set

- Architect
- Architect reviewer
- Capsule designer
- Code reviewer
- DB designer
- Design reviewer
- Designer
- Implementer
- Integrator

Analysts

- Business process, designer, reviewer
- Requirements
- System
- Use case specifier
- User interface designer

# More Roles...

Manager

- Change control
- Configuration
- Deployment
- Process
- Project
- Project reviewer

Tester

- Test designer
- Testers

Additional

- Fit all categories

# RUP Review – Strengths

Structured

Iterative

Use cases – strong concept, used widely

Tied to UML

Robust tool support – see all at IBM

Tailorable/Scalable

“Agile”?

# RUP Review – Weaknesses

Use cases – Do they work for all projects?

Learning curve

- Tied to tools
- UML

Architecture?

Roles – Too many? How to combine?

Non-functional requirements

- Quality attributes?

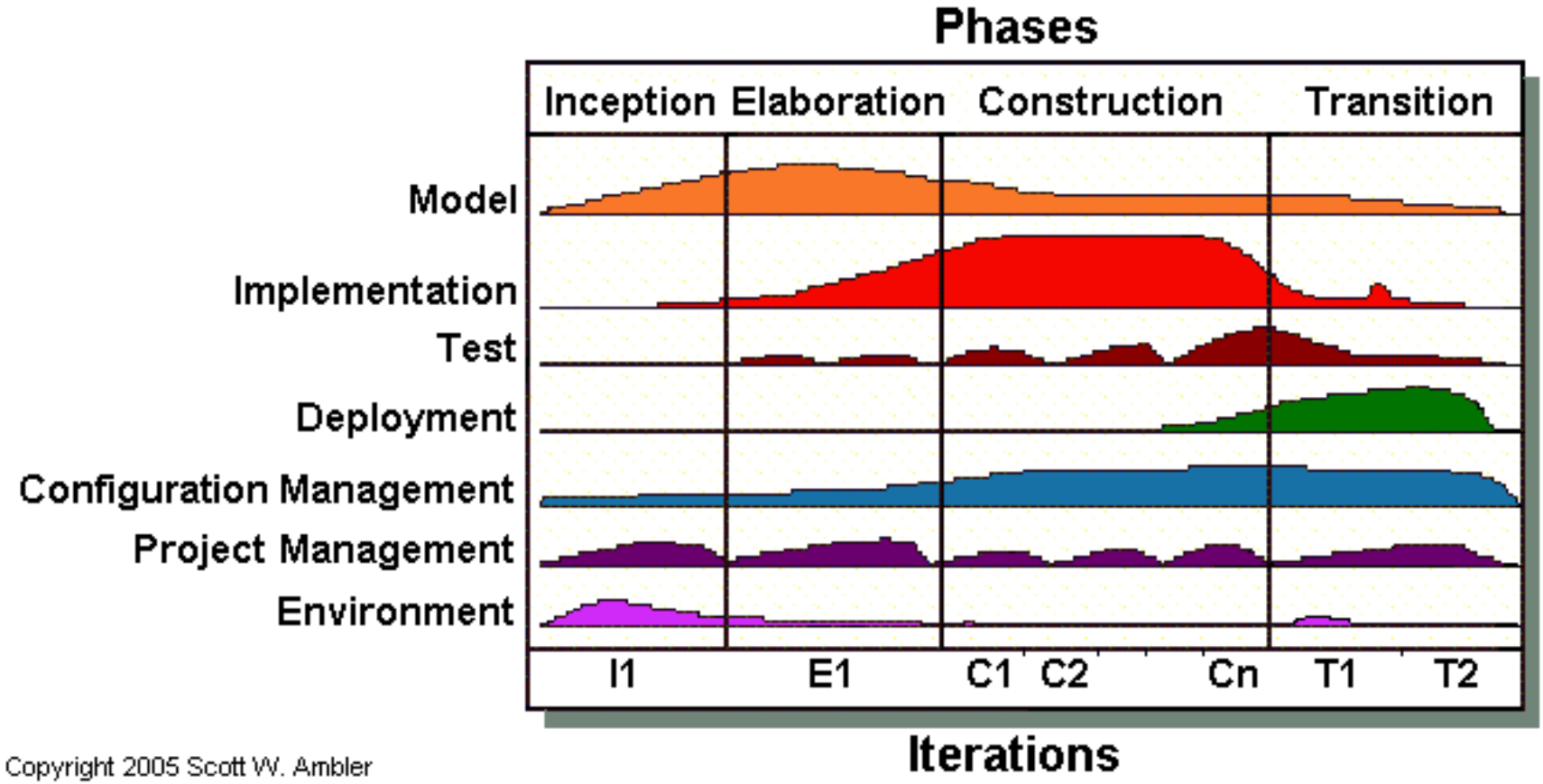
# RUP Centerpiece

Model analysis

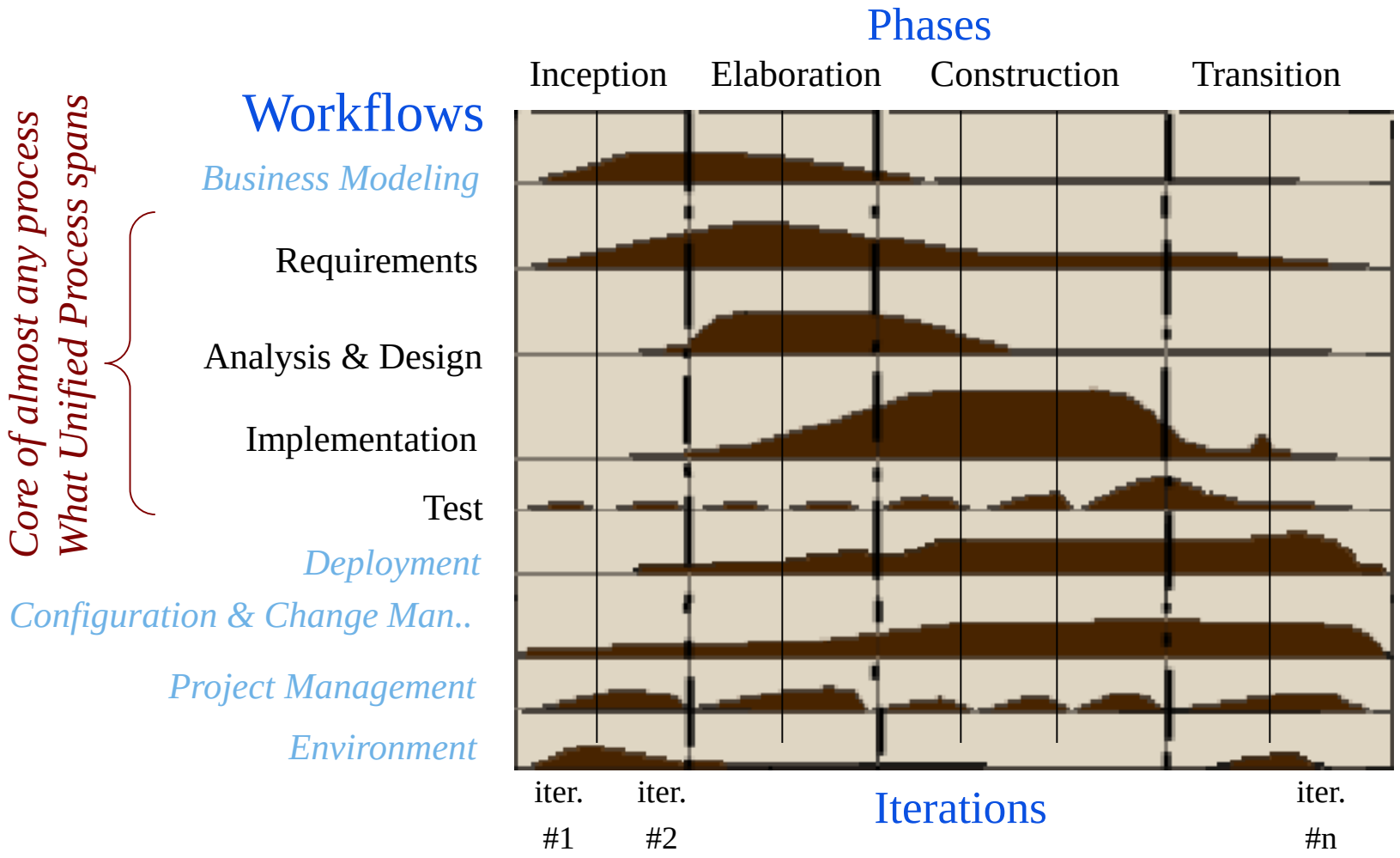
- Modeling allows stakeholders to understand the problem.
- Great for functionality
- Not so good for quality attributes

# Agile Unified Process

Simplified from RUP



# The Process Outline – Phases Versus Iterations



# Disciplines

Model – compare to RUP

Implementation

Test

Deployment

Configuration management

Project management

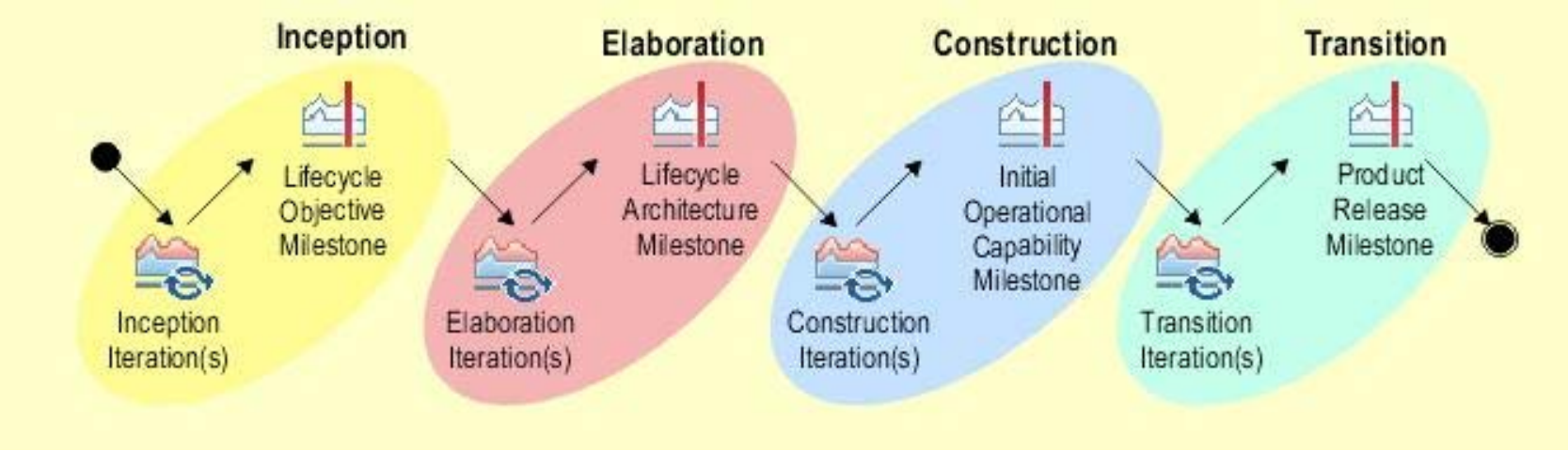
Environment

# Open Unified Process

Eclipse

Very lean UP

All phases include risk analysis



# Open UP roles

Analyst

Any role (I like this one, take out trash…)

Architect

Developer

Project manager

Stakeholder

Tester

# AUP and OUP Summary

More Agile than RUP

Keeps basic phases, iterations, work flows

Smaller teams

Fewer roles

Less overhead?

Big and small picture?

# Summary

Agile processes

- XP and Scrum

RUP, AUP, OUP

Comparing Processes

- There are no unique perfect solutions to any software project.
- Need to learn how to adapt and adopt as warranted.

# Choosing. . .

# Common Errors in Choosing

*Going from framework to project*

Looking for a recipe…

- There is no silver bullet.
- Do not tailor your project to a process, instead tailor the “right process.”

Supermarket shopping…

- Do not pick all the “best” techniques within processes and mix them together.
- But you can use some in tailoring…

# Let’s Compare!

# Comparative Matrix

## Where would you put ACDM?

| | AGILE | TSP | RUP |
|---|---|---|---|
| **TEAM** | | | |
| Size | 5-10 | 3-7 | > 10 |
| Experience Required | High | Low | High |
| Location (co-located or distributed) | co-located | Either | Either |
| **REQUIREMENTS** | | | |
| Avail. (Undocumented or available) | Undoc | Available | Undoc |
| Changability | Volatile | Static | Mid-volatile |
| **CUSTOMER** | | | |
| Heavily involved | Yes | No | Yes |
| Experienced | Yes | No | No |
| **DEVELOPMENT ENVIRONMENT** | | | |
| Organizational hierarchy | Democratic | Autocratic | Autocratic |
| Culture | Decentralized | Centralized | Centralized |
| Tools | Low | Med | Heavy |
| **PRODUCT** | | | |
| Documentation requirements | Poor | Good | Good |
| Traceability | Poor | Good | Very Good |

# Comparative Matrix with ACDM

| | AGILE | TSP | RUP | ACDM |
|---|---|---|---|---|
| **TEAM** | | | | |
| Size | 5-10 | 3-7 | > 10 | >3 |
| Experience Required | High | Low | High | Med |
| Location (co-located or distributed) | co-located | Either | Either | Either |
| **REQUIREMENTS** | | | | |
| ability (Undocumented or available) | Undoc | Available | Undoc | Avail |
| Changability | Volatile | Static | Mid-volatile | Mid |
| **CUSTOMER** | | | | |
| Heavily involved | Yes | No | Yes | Yes |
| Experienced | Yes | No | No | No |
| **DEVELOPMENT ENVIRONMENT** | | | | |
| Organizational hierarchy | Democratic | Autocratic | Autocratic | Mid |
| Culture | Decentralized | Centralized | Centralized | Either |
| Tools | Low | Med | Heavy | Low |
| **PRODUCT** | | | | |
| Documentation requirements | Poor | Good | Good | Average |
| Traceability | Poor | Good | Very Good | Good |

# Criteria…

## You should be asking questions.

Is the preceding all-inclusive?

What other criteria might apply?

- Look at teams – What about culture, personalities?

Would these apply to all projects?

# Choosing Suitable SDLC

*Choose Your Weapon Wisely* - Justin Rockwood 2003

Weighted Matrix model

- 1- weakness
- 2- push
- 3- strength

Compares suitability of 5 methods

- RUP
- MS Synch and Stabilize
- TSP
- XP
- Scrum

(Haven't added ACDM yet…)

# *Choose Your Weapon Wisely*
## Justin Rockwood 2003

Weighted score for following project characteristics:

- Organization-wide processes
- New process adoption
- Type of product
- Requirements stability
- Requirements traceability
- Average team size and total developers
- Product size and complexity
- Developer competence and experience
- “Hacker sentiment”
- Management style

# Example for Total Developers

How many developers in total project?

a) < 40
b) 41 – 100
c) Hundreds…

| | RUP | MSS | TSP | XP | Scrum |
|---|---|---|---|---|---|
| a) | 2 | 2 | 2 | 2 | 2 |
| b) | 3 | 3 | 1 | 1 | 2 |
| c) | 3 | 3 | 1 | 1 | 2 |

# Example for Type of Product

What type of project?

a) Life-critical (e.g., patient monitor, ATC)
b) Non-life-critical, but mission-critical (e.g., banking)
c) Embedded, neither life- or mission-critical
d) Application, neither life- or mission-critical

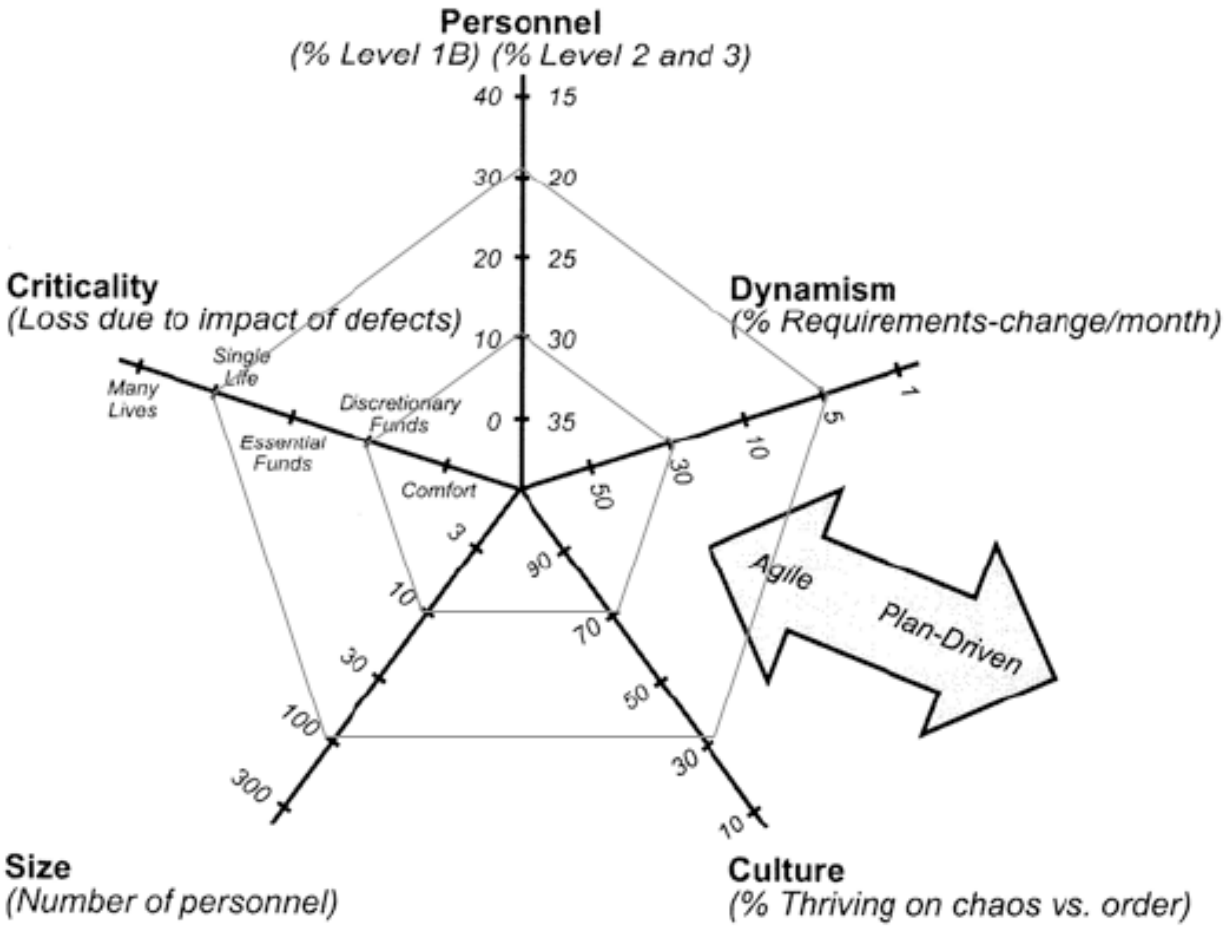
| | RUP | MSS | TSP | XP | Scrum |
|---|---|---|---|---|---|
| a) | 3 | 2 | 3 | 1 | 1 |
| b) | 2 | 2 | 2 | 2 | 2 |
| c) | 3 | 3 | 2 | 2 | 2 |
| d) | 2 | 2 | 2 | 2 | 2 |

# Add Up Scores

11 project areas, but…

- Not definitive – Starting point for research
  - General direction
  - Tied scores, or little variation
- Lower scoring method may still be suitable
- Not complete, needs more work
  - Weighting some criteria more than others?
  - Defining other processes

# Boehm and Turner: "Balancing Agility with Discipline," 2004



# Personnel Discriminator…

## Boehm and Turner: "Balancing Agility with Discipline," 2004

| Level | Characteristics |
|---|---|
| 3 | Is able to revise a method (break its rules) to fit an unprecedented new situation. |
| 2 | Is able to tailor a method to fit a precedented new situation. |
| 1A | With training, is able to perform discretionary method steps (e.g. sizing stories to fit increments, composing patterns, compound refactoring, and complex COTS integration). Can become level 2 with experience. |
| 1B | With training, is able to perform procedural method steps (*e.g.*, coding a simple method, simple refactoring, following coding standards and capability model procedures, and running tests). Can master some Level 1A skills with experience. |
| -1 | May have technical skills, but is unable or unwilling to collaborate or follow shared methods. |

# And Then

Apply discriminators

- Criticality – Measured by loss, annoyance to $ to life
- Size – Gradient?
- Cultural – Chaos or planned
- Dynamism – Volatility of requirements

Closer to center promotes Agile

# Recommendations

The methods shown just point to the right direction and are not absolute answers.

- Analysis of current method

Plan any adoption of a new method.

ROI is important.

- As is cost to benefit

All methods work for the right.

- Project
- Team
- Organization

(But some may be better.)

# Summary

Review the criteria that you would use to choose a process framework.

Don’t lie to yourself or cheat if you are going to adopt one.

- Adopt whole, then tailor.

# Summary

Agile processes

- XP and Scrum

RUP, AUP, OUP

Choosing a process

- There are no unique perfect solutions to any software project.
- Need to learn how to adapt and adopt as warranted.

# Questions?


# Module 8: Software Assurance Lifecycle and Maturity Models

## Introduction to Assured Software Engineering

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

# Notices



This material is based upon work funded and supported by the Independent Agency under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center sponsored by the United States Department of Defense.

# Outline

Software assurance practices

Software assurance lifecycle models

Software assurance maturity models

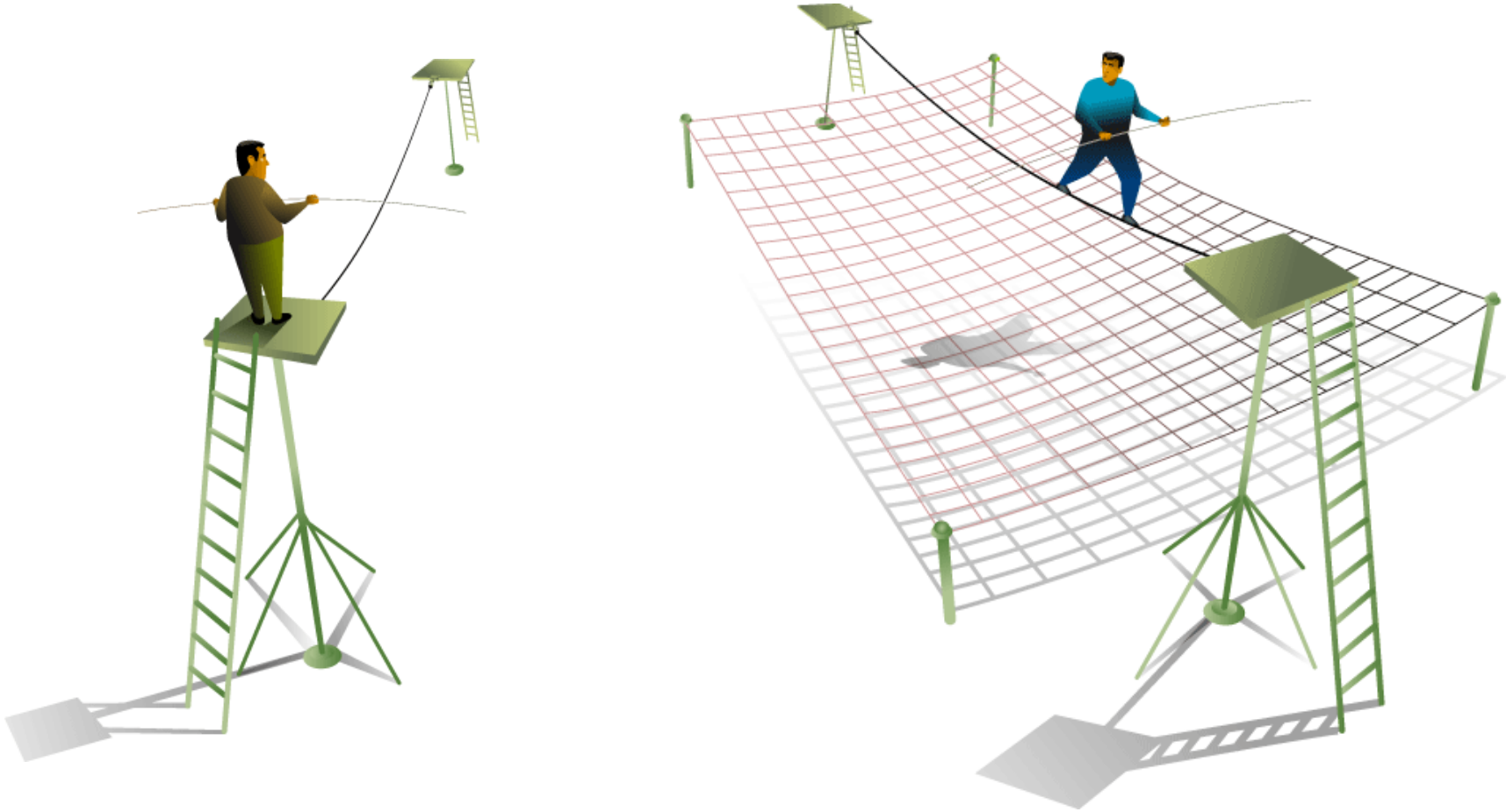
CLASP overview

# Software Assurance Practices

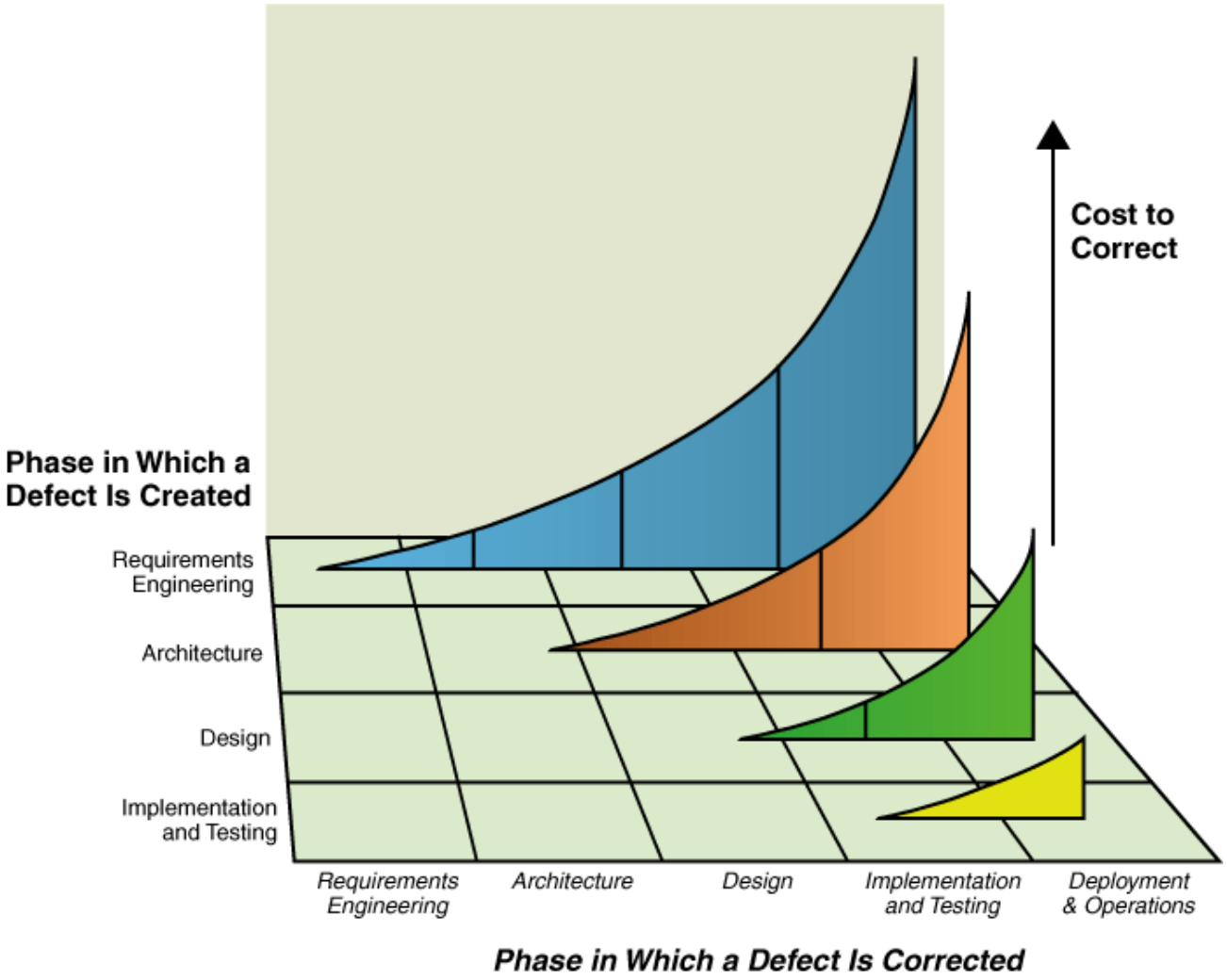
# Security Perspectives

http://security.gloriad.org/blog/2007/10/21/traditional-thinking/

# So What Should We Do?



# Understand the Cost of Correcting Software Defects




McConnell, Steve. “Software Quality at Top Speed.” August 1996.

http://www.stevemcconnell.com/articles/art04.htm

# Example Security Practices -1

Project management

- Enterprise software security framework
- Security development lifecycle
- Risk management and ongoing assessment

Full lifecycle

- Attack patterns: a structured representation for how attackers think
- Assurance cases: demonstration that a system satisfies its security properties

Requirements engineering

- Misuse/abuse cases: anticipate abnormal behavior

# Example Security Practices -2

Architecture and design

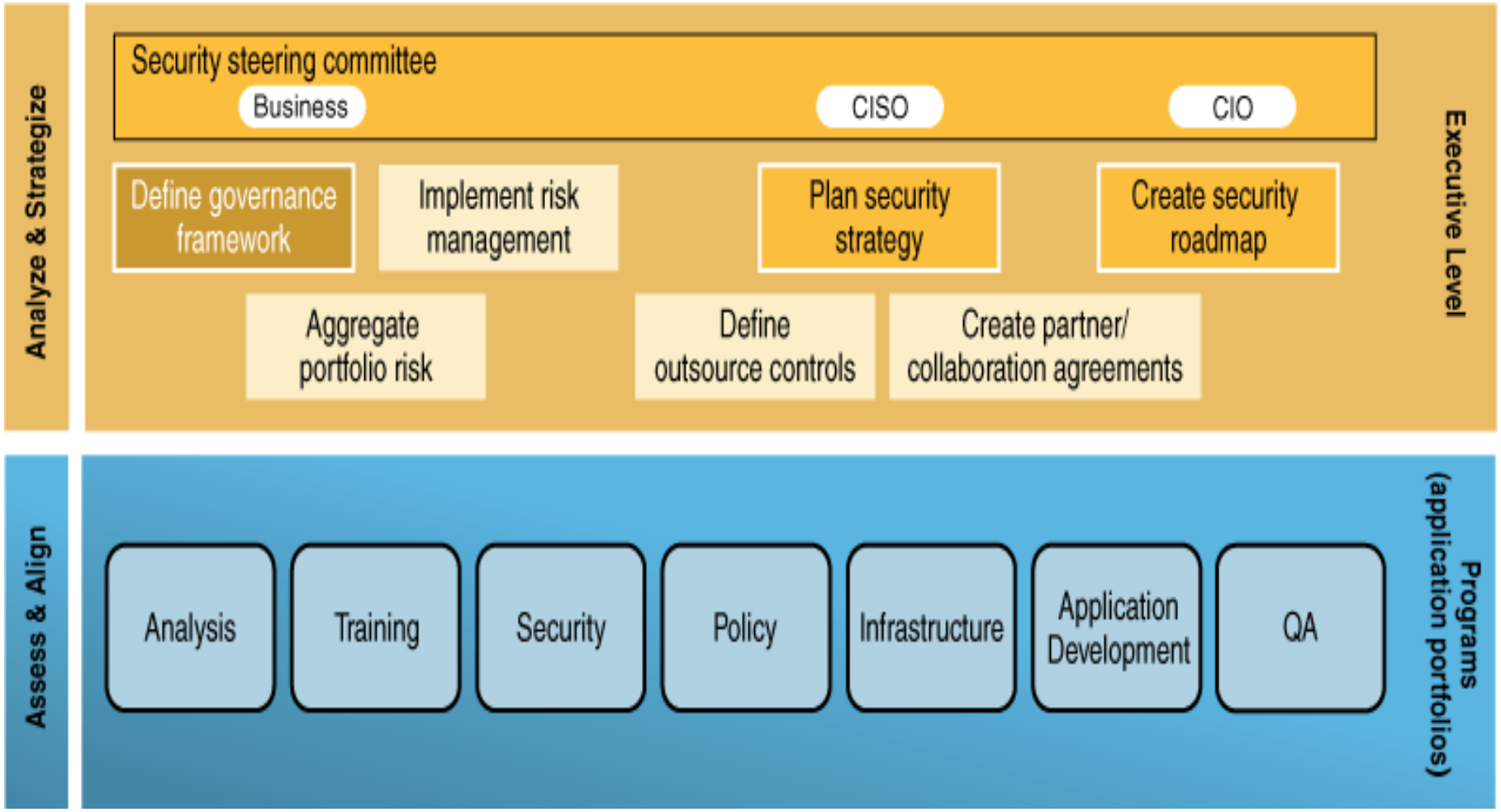
- Architectural risk analysis

Code and test

- Secure code reviews
- White box, black box, and penetration testing

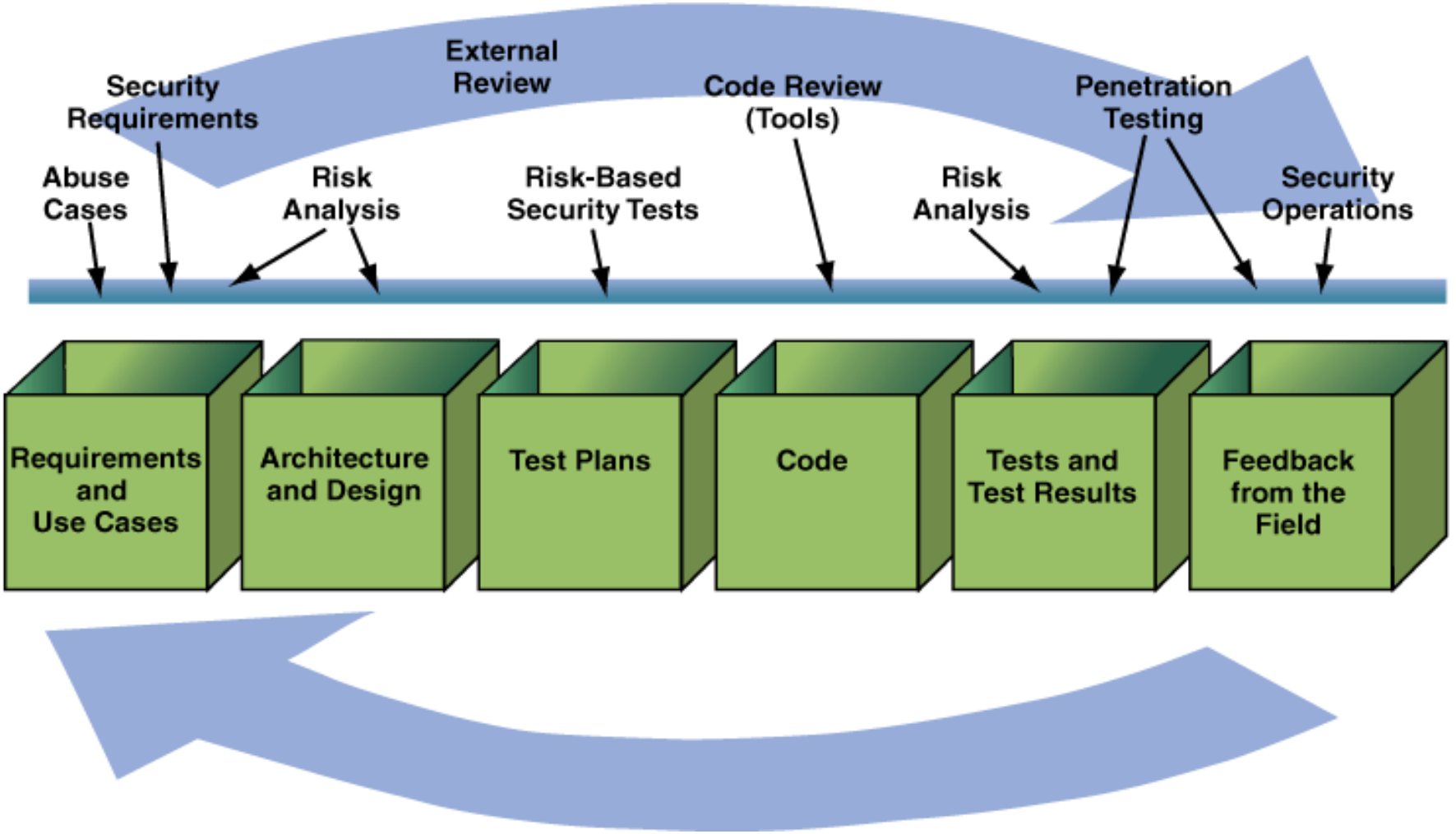
# Software Assurance Lifecycle Models

# Enterprise Software Security Framework




Steven, John. "Adopting an Enterprise Software Security Framework." *IEEE Security & Privacy 4*, 2 (March/April 2006): 84–87. https://buildsecurityin.us-cert.gov/daisy/bsi/resources/published/series/bsi-ieee/568.html

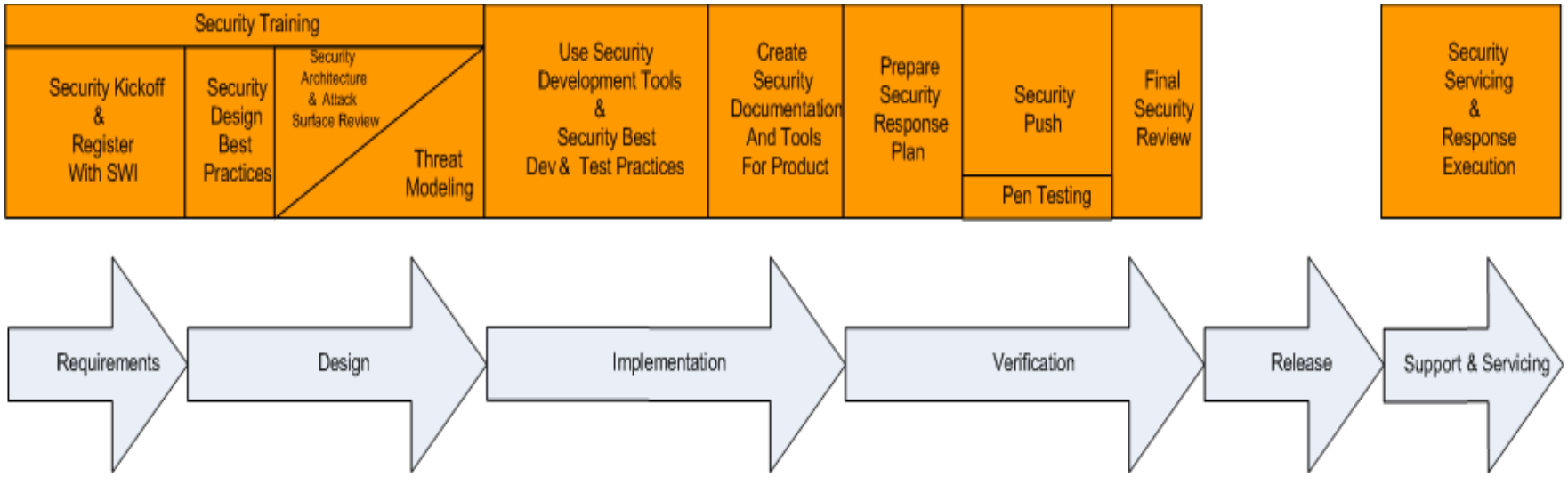
# SDLC with Defined Security Touchpoints




SDLC: Software Development Life Cycle

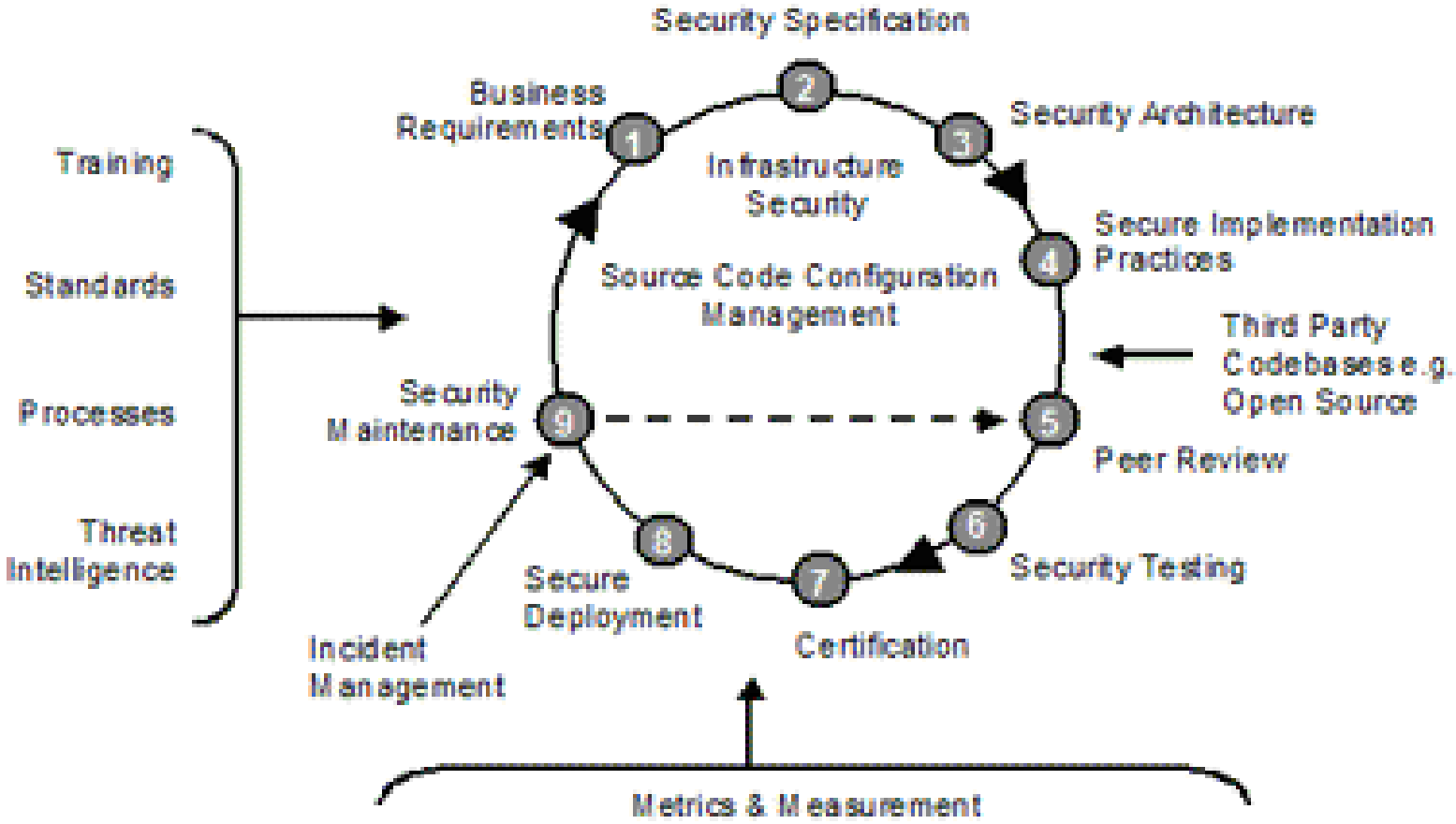
McGraw, Gary. *Software Security: Building Security In*. Boston, MA: Addison-Wesley Professional, 2006.

# Microsoft's Security Development Lifecycle



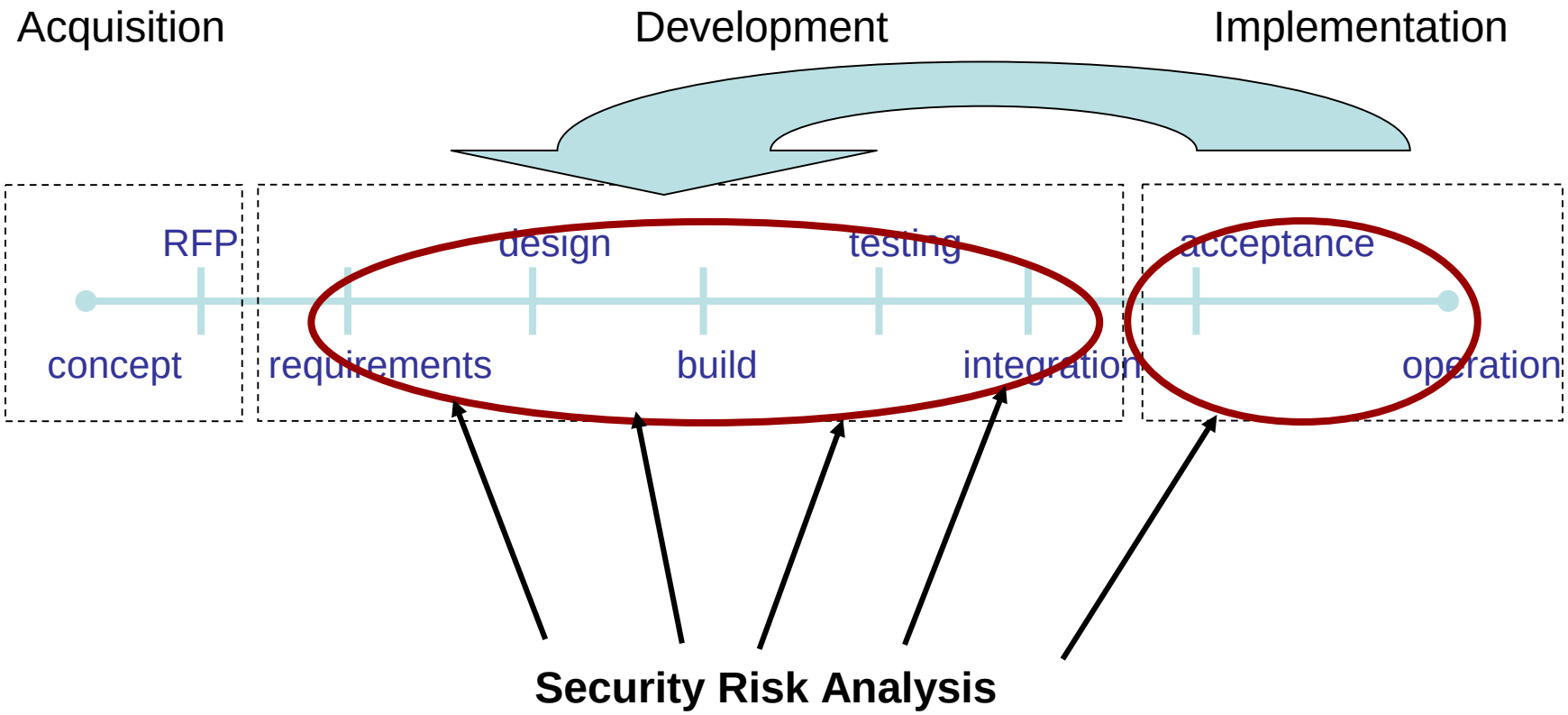

http://msdn2.microsoft.com/en-us/library/ms995349.aspx

# Assurent Software Security Lifecycle




http://www.assurent.com/index.php?id=59

# Assess Security Risk Across the SDLC



# Discussion

Are any of these security lifecycle models familiar?

Do you know of organizations or people using them?

Which do you think would be easiest to use?

# Attack Patterns

Blueprint for creating an attack (like a sewing pattern)

Consists of

- Attack prerequisites
- Attack description
- Related vulnerabilities
- Method of attack
- Skills and resources required to execute attack
- Applicable contexts
- Prevention and mitigation strategies

Consult CAPEC: Common Attack Pattern Enumeration and Classification http://capec.mitre.org/

# Assurance Cases

Applicable during all phases of software development

Similar to a legal case

Presents arguments showing how a top-level claim is supported by evidence

- The system is acceptably secure.
- The system has none of the common coding defects that lead to security vulnerabilities.

Considers people, process, and technology

# Misuse/Abuse Cases

Document a priori how software should react to illegitimate use (can’ts and won’ts).

- Brainstorm with designers and software security experts.
  - How does the software distinguish between good and bad input?
  - Between legitimate application vs. rogue application requests?
  - How can an attacker disrupt software communication interfaces?
  - Does the database server assume that the client manages all data access permissions?

Ask:

- What assumptions are implicit in our system?
- What things make our assumptions false?
- What are some candidate attacks (consult attack patterns)?

Strike a balance between cost and value.

- Prioritize which cases to develop.
- Risk analysis helps guide case selection.

# Architecture and Design

Not the same as security architecture

- architecture of security components (firewalls, IDS, other sensors, network monitoring points, etc.)

Architectural Risk Analysis

- software characterization
- threat analysis
- architectural vulnerability assessment
- risk likelihood determination
- risk impact determination
- risk mitigation planning

Perform inspections and peer reviews

# Secure Code Review/Scanning

Adopt a secure coding standard.

- Validate input
- Perform bounds checking (buffer overflows)
- Check for conditions that could lead to exceptions
- Base access decisions on permission, not exclusion (default deny)
- Enforce the principle of least privilege for processes
  - Time out elevated privileges
- Sanitize data sent to other systems
- Guard against race conditions (infinite loops, deadlocks, resource collisions)
- Review code against attack patterns and misuse/abuse cases

Conduct structured code inspections and peer review of source code.

Use static source code analysis tools.

# Security Testing -1

Test approach and selection determined based on risk analysis

- Use attack patterns and abuse cases

Emphasizes what an application should not do

- "Unauthorized users should not be able to access data."
  - Validate least privilege
  - Time-limited escalation of privilege
  - Disable account after x unsuccessful login attempts

# Security Testing -2

White box testing

- validate design decisions and assumptions
- analyze data, control, information flows; coding practices; exception and error handling

Black box testing

- focus on externally visible behavior
- examine requirements, protocols, interfaces, attempted attacks
- vulnerability scanning is one example

Penetration testing (revised)

- final production environment; final configuration
- structured to demonstrate impact of likely risks

# Software Assurance Maturity Models and Frameworks

(Developed by Dan Reddy, EMC-2)

# Product Security Office: Delivers Product Security from Concept to Customer

Concept

Security Development Lifecycle

Security Certifications

Vulnerability Response

Software Supply Chain Risk Management

Customer

## Cross Industry Involvement

SAFECode — Software Assurance Forum for Excellence in Code — Driving Security and Integrity

*Founding member '07*

BSIMM

*"… The data show that EMC's Product Security Office practices have improved greatly over time and currently rank among the most advanced."*

THE Open GROUP

***Trusted Technology Forum**: Building Industry Standard for Supply Chain*

# BSIMM8: The Building Security In Maturity Model

(Authored by Gary McGraw, Sammy Migues, and Jacob West; Revised by Ole Villadsen)

# Prescriptive vs. Descriptive Models

Prescriptive models describe what you should do.

- SAFECode
- SAMM
- SDL
- Touchpoints

Every firm has a methodology they follow (often a hybrid).

You need an SSDL.

Descriptive models describe what is actually happening.

The BSIMM is a descriptive model that can be used to measure any number of prescriptive SSDLs.

# BSIMM: Software Security Measurement

Real data from (109) real initiatives

256 measurements

36 over time

McGraw, Migues, and West

SYNOPSYS®

cigital

FORTIFY® An HP Company

# A Software Security Framework

| The Software Security Framework (SSF) | | | |
|---|---|---|---|
| **Governance** | **Intelligence** | **SSDL Touchpoints** | **Deployment** |
| Strategy and Metrics | Attack Models | Architecture Analysis | Penetration Testing |
| Compliance and Policy | Security Features and Design | Code Review | Software Environment |
| Training | Standards and Requirements | Security Testing | Configuration Management and Vulnerability Management |

Four domains

Twelve practices

Three levels within each Practice

# 109 Firms in BSIMM8 Community

| | | |
|---|---|---|
| Adobe | Epsilon | NXP Semiconductors N.V. |
| Aetna | Experian | Oracle NSGBU |
| Amgen | F-Secure | PayPal |
| ANDA | Fannie Mae | Principal Financial Group |
| Autodesk | Fidelity | Qualcomm |
| Axway | Freddie Mac | Royal Bank of Canada |
| Bank of America | General Electric | Scientific Games |
| Betfair | Genetec | Siemens |
| BMO Financial Group | Highmark Health Solutions | Sony Mobile |
| Black Knight Financial Services | Horizon Healthcare Services, Inc. | Splunk |
| Box | HPE Fortify | Symantec |
| Canadian Imperial Bank of Commerce | HSBC | Synopsys SIG |
| Capital One | Independent Health | Target |
| City National Bank | iPipeline | TD Ameritrade |
| Cisco | JPMorgan Chase & Co. | The Advisory Board |
| Citigroup | Lenovo | The Home Depot |
| Citizen's Bank | LGE | The Vanguard Group |
| Comerica Bank | LinkedIn | Trainline |
| Cryptography Research, a division of Rambus | McKesson | Trane |
| Dell EMC | Medtronic | U.S. Bank |
| Depository Trust & Clearing Corporation | Morningstar | Veritas |
| Elavon | Navient | Verizon |
| Ellucian | NetApp | Wells Fargo |
| | NVIDIA | Zendesk |
| | | Zephyr Health |

BSIMM8, Page 3

Plus additional anonymous firms

# Building BSIMM

Build a maturity model from actual data gathered from 9 well known large-scale software security initiatives.

- Create a software security framework.
- Interview nine firms in-person.
- Discover 110 activities through observation.
- Organize the activities in 3 levels of increasing maturity.
- Build scorecard.

# BSIMM8 Scorecard

| GOVERNANCE | | INTELLIGENCE | | SSDL TOUCHPOINTS | | DEPLOYMENT | |
|---|---|---|---|---|---|---|---|
| ACTIVITY | BSIMM8 FIRMS (109) | ACTIVITY | BSIMM8 FIRMS (109) | ACTIVITY | BSIMM8 FIRMS (109) | ACTIVITY | BSIMM8 FIRMS (109) |
| **Strategy & Metrics** | | **Attack Models** | | **Architecture Analysis** | | **Penetration Testing** | |
| [SM1.1] | 55 | [AM1.2] | 68 | [AA1.1] | 90 | [PT1.1] | 95 |
| [SM1.2] | 56 | [AM1.3] | 36 | [AA1.2] | 30 | [PT1.2] | 71 |
| [SM1.3] | 52 | [AM1.5] | 50 | [AA1.3] | 24 | [PT1.3] | 68 |
| [SM1.4] | 92 | [AM2.1] | 9 | [AA1.4] | 49 | [PT2.2] | 23 |
| [SM2.1] | 46 | [AM2.2] | 8 | [AA2.1] | 14 | [PT2.3] | 20 |
| [SM2.2] | 36 | [AM2.5] | 14 | [AA2.2] | 12 | [PT3.1] | 8 |
| [SM2.3] | 40 | [AM2.6] | 14 | [AA3.1] | 2 | [PT3.2] | 7 |
| [SM2.5] | 21 | [AM2.7] | 10 | [AA3.2] | 0 | | |
| [SM2.6] | 33 | [AM3.1] | 4 | [AA3.3] | 2 | | |
| [SM3.1] | 15 | [AM3.2] | 1 | | | | |
| [SM3.2] | 9 | | | | | | |
| **Compliance & Policy** | | **Security Features & Design** | | **Code Review** | | **Software Environment** | |
| [CP1.1] | 66 | [SFD1.1] | 85 | [CR1.2] | 69 | [SE1.1] | 49 |
| [CP1.2] | 89 | [SFD1.2] | 70 | [CR1.4] | 65 | [SE1.2] | 91 |
| [CP1.3] | 56 | [SFD2.1] | 29 | [CR1.5] | 34 | [SE2.2] | 33 |
| [CP2.1] | 27 | [SFD2.2] | 41 | [CR1.6] | 37 | [SE2.4] | 29 |
| [CP2.2] | 37 | [SFD3.1] | 5 | [CR2.5] | 26 | [SE3.2] | 15 |
| [CP2.3] | 35 | [SFD3.2] | 11 | [CR2.6] | 16 | [SE3.3] | 4 |
| [CP2.4] | 40 | [SFD3.3] | 2 | [CR2.7] | 23 | [SE3.4] | 4 |
| [CP2.5] | 41 | | | [CR3.2] | 3 | | |
| [CP3.1] | 22 | | | [CR3.3] | 2 | | |
| [CP3.2] | 14 | | | [CR3.4] | 3 | | |
| [CP3.3] | 5 | | | [CR3.5] | 5 | | |
| **Training** | | **Standards & Requirements** | | **Security Testing** | | **Config. Mgmt. & Vuln. Mgmt.** | |
| [T1.1] | 73 | [SR1.1] | 66 | [ST1.1] | 87 | [CMVM1.1] | 92 |
| [T1.5] | 31 | [SR1.2] | 69 | [ST1.3] | 79 | [CMVM1.2] | 96 |
| [T1.6] | 22 | [SR1.3] | 71 | [ST2.1] | 25 | [CMVM2.1] | 78 |
| [T1.7] | 44 | [SR2.2] | 33 | [ST2.4] | 11 | [CMVM2.2] | 83 |
| [T2.5] | 16 | [SR2.3] | 25 | [ST2.5] | 9 | [CMVM2.3] | 44 |
| [T2.6] | 18 | [SR2.4] | 25 | [ST2.6] | 10 | [CMVM3.1] | 4 |
| [T3.1] | 3 | [SR2.5] | 26 | [ST3.3] | 4 | [CMVM3.2] | 6 |
| [T3.2] | 6 | [SR2.6] | 15 | [ST3.4] | 3 | [CMVM3.3] | 7 |
| [T3.3] | 5 | [SR3.1] | 10 | [ST3.5] | 4 | [CMVM3.4] | 12 |
| [T3.4] | 7 | [SR3.2] | 9 | | | | |
| [T3.5] | 4 | | | | | | |
| [T3.6] | 5 | | | | | | |

113 Activities

3 levels

Top 12 activities in Yellow
- 68 (62%) of 109 firms

Comparing scorecards between releases is interesting.

# BSIMM8 Scorecard (cont'd)

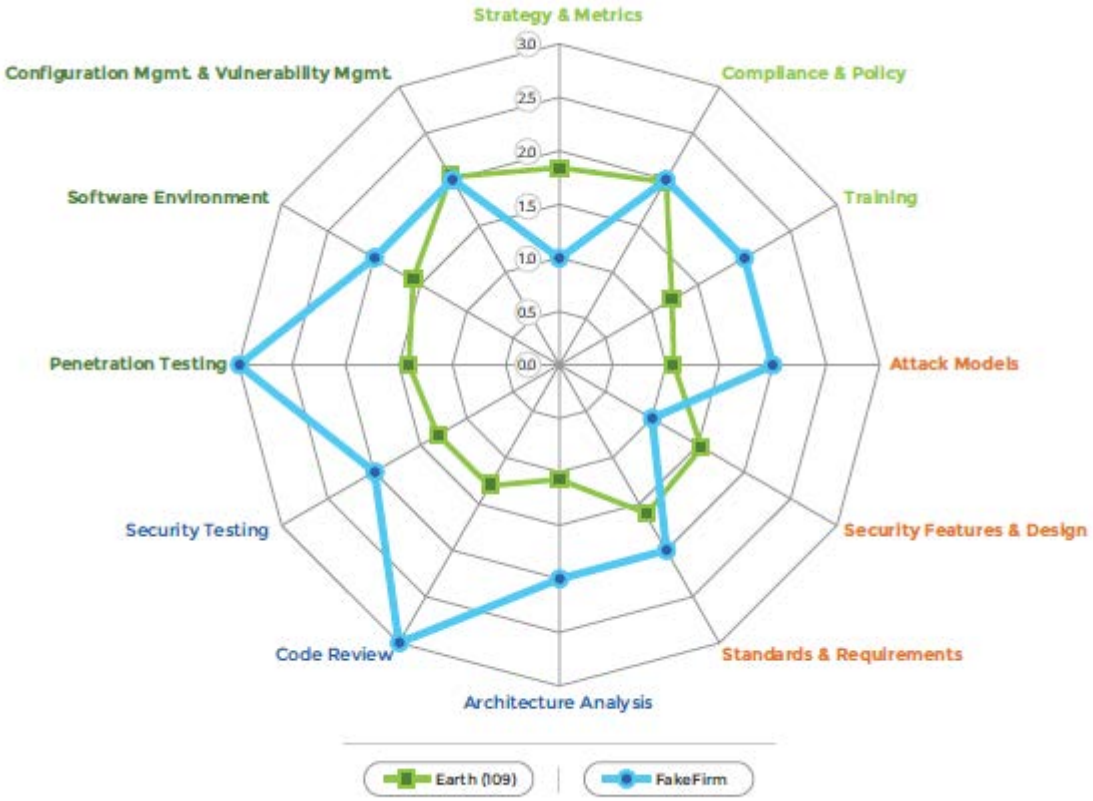
| TWELVE CORE ACTIVITIES "EVERYBODY" DOES | |
|---|---|
| ACTIVITY | DESCRIPTION |
| [SM1.4] | Identify gate locations and gather necessary artifacts. |
| [CP1.2] | Identify PII obligations. |
| [T1.1] | Provide awareness training. |
| [AM1.2] | Create a data classification scheme and inventory. |
| [SFD1.1] | Build and publish security features. |
| [SR1.3] | Translate compliance constraints to requirements. |
| [AA1.1] | Perform security feature review. |
| [CR1.2] | Have SSG perform ad hoc review. |
| [ST1.1] | Ensure QA supports edge/boundary value condition testing. |
| [PT1.1] | Use external penetration testers to find problems. |
| [SE1.2] | Ensure host and network security basics are in place. |
| [CMVM1.2] | Identify software bugs found in operations monitoring and feed them back to development. |

# BSIMM8 as a Measuring Stick

Compare a firm with peers using the high water mark view.

Compare business units.

Chart an SSI over time.



# BSIMM8 as a Longitudinal Study

36 firms measured at least twice

Raw score increased in 29 of 36 firms

Observation count increased by 33.4%

“SSI’s mature over time”

# BSIMM8

BSIMM8 released September 2017 under creative commons

http://bsimm.com

BSIMM is a yardstick.

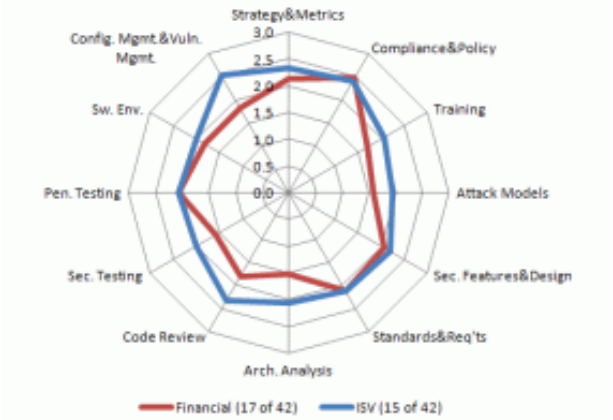
- Use it to see where you stand.
- Use it to figure out what your peers do.

# An Assurance Ecosystem

(Developed by Dan Reddy, EMC-2)

# One View as to How the Pieces Fit

## BSIMM




Shows data congruence of security activities found in companies that were analyzed

## THE Open GROUP

*Making standards work®*

- Standard that outlines best practices of ICT Providers to mitigate vs. *tainted* and *counterfeit* products.
- Method to accredit Trusted Technology Providers.

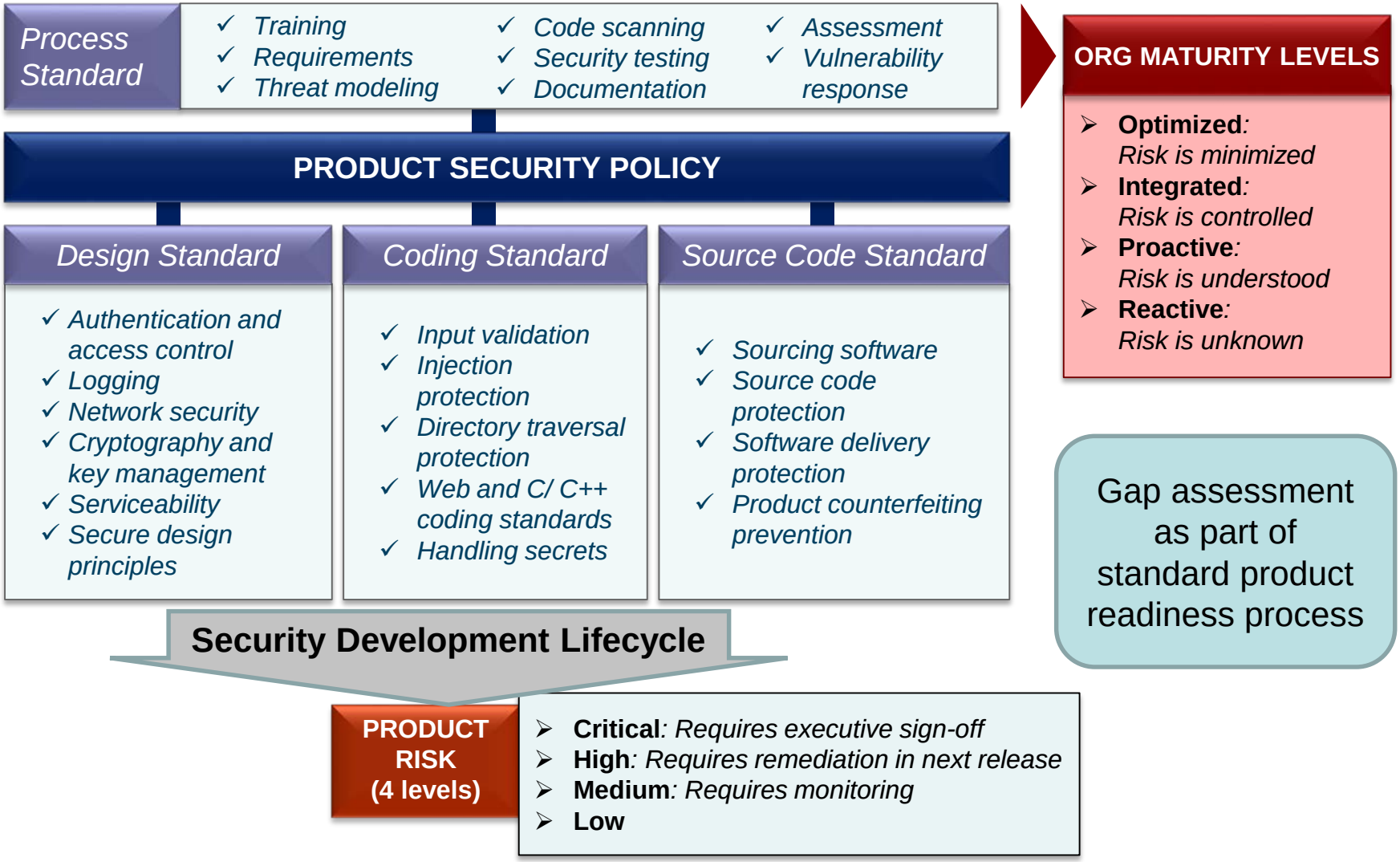
## SAFECode

Software Assurance Forum for Excellence in Code

Driving Security and Integrity

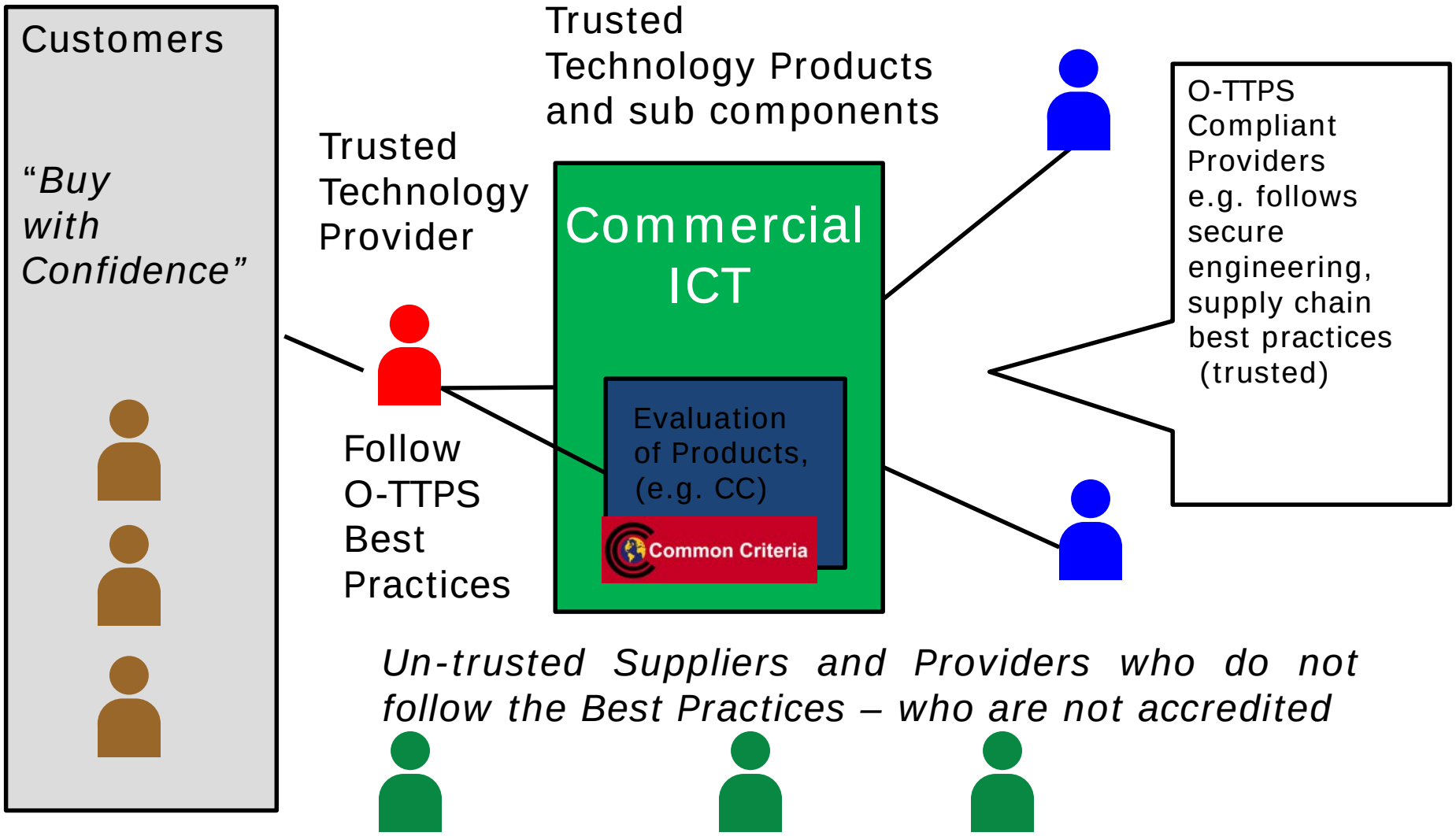
- Building secure products
- Prescriptive
- How should I do it?
- Where should I start?

# EMC-Wide Standard with Focus on Risk and Organization Maturity



# Customers Buy with More Confidence:

*Providers and Suppliers Can Extend Supply Chain Integrity*



# Classifying Vulnerabilities: Some Useful Resources

CVE: Common Vulnerabilities & Exposures Database

http://cve.mitre.org

CWE: Common Weakness Enumeration

- A community-developed dictionary of software weakness types

  http://cwe.mitre.org/

NVD: National Vulnerability Database

http://nvd.nist.gov

- 56,965 CVE Vulnerabilities

Bugtraq mailing list: how to exploit and fix vulnerabilities

http://www.securityfocus.com/archive/1

# Questions?


# Module 9:
# OWASP CLASP Overview
## (Developed by Nick Coblentz)

Introduction to Assured Software Engineering

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

# Notices



This material is based upon work funded and supported by the Independent Agency under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center sponsored by the United States Department of Defense.

# OWASP CLASP Presentation Outline

Role-Based View
- Introduction to each role

Activity-Assessment View
- Examples

Activity-Implementation View
- Examples

CLASP Roadmap

What Is CLASP?

CLASP Best Practices

CLASP Organization

Bird's-Eye View of CLASP Process

Concepts View
- Security Services
- Vulnerability View

# What Is CLASP?

Comprehensive, Lightweight, Application Security Process

OWASP project

“Activity driven, role-based set of process components whose core contains formalized best practices for building security into your existing or new-start software development lifecycles in a structured, repeatable, and measurable way”

# What Is CLASP?

Method for applying security to an organization's application development process

Adaptable to any organization or development process

OWASP CLASP is intended to be a complete solution that organizations can read and then implement iteratively

Focuses on leveraging a database of knowledge (CLASP vulnerability lexicon, security services, security principles, etc.) and automated tools/processes

# CLASP Best Practices

## Institute security awareness programs

- Provide security training to stakeholders
- Present organization's security policies, standards, and secure coding guidelines

## Perform application assessments

- Is a central component in overall strategy
- Find issues missed by implemented “Security Activities”
- Leverage to build a business case for implementing CLASP

## Capture security requirements

- Specify security requirements alongside business/application requirements

## Implement secure development process

- Include “Security Activities,” guidelines, resources, and continuous reinforcement

# CLASP Best Practices

Build vulnerability remediation procedures

- Define steps to identify, assess, prioritize, and remediate vulnerabilities

Define and monitor metrics

- Determine overall security posture
- Assess CLASP implementation progress

Publish operational security guidelines

- Monitor and manage security of running systems
- Provide advice and guidance regarding security requirements to end-users and operational staff

# CLASP Organization

Concepts View

Role-Based View

Activity-Assessment View

- Implementation costs
- Activity applicability
- Risk of inaction

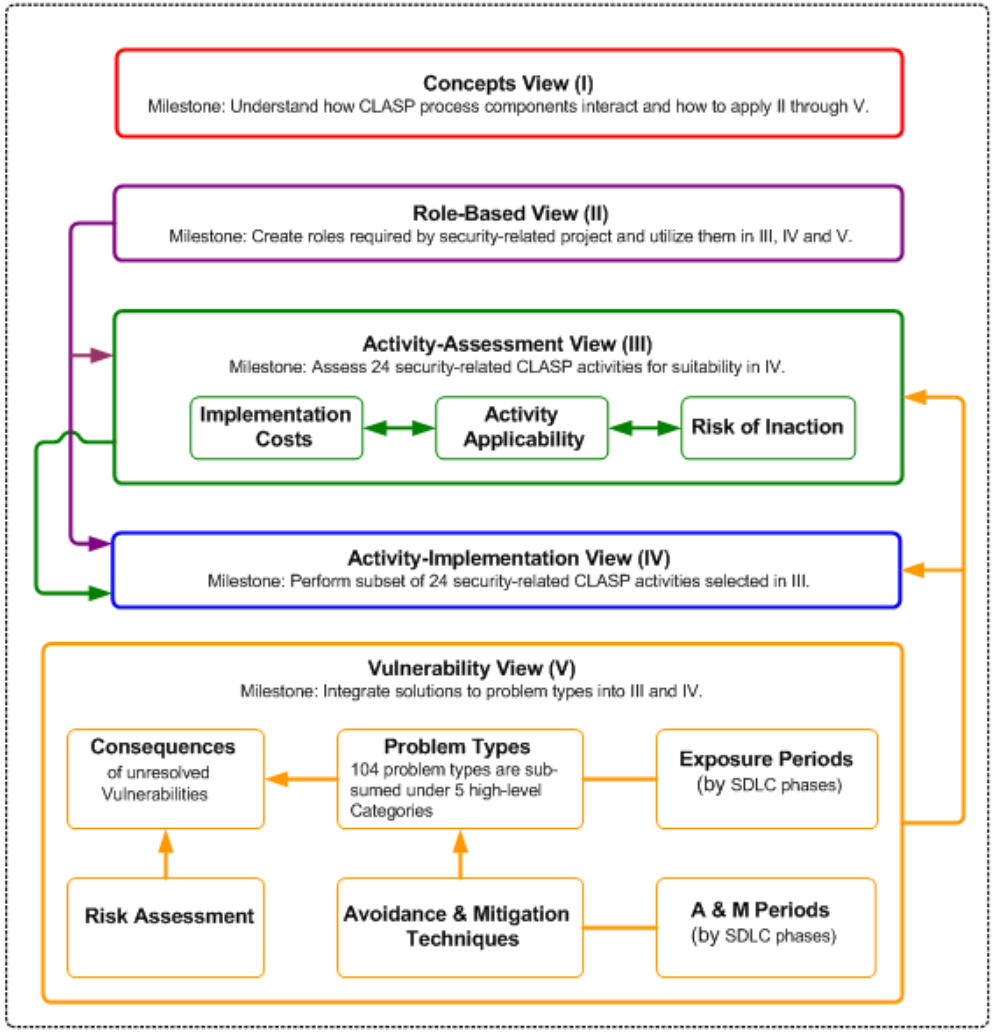
Activity-implementation View

- 24 “Security Activities”

Vulnerability Lexicon

- Consequences, problem types, exposure periods, avoidance and mitigation techniques

Additional Resources



# Bird's-Eye View of CLASP Process

Stakeholders

- Read and understand "Concepts View"
- Read and understand "Role-Based View"

Project manager

- Reads and understands "Activity-Assessment View"
- Determines applicable and feasible "Security Activities" to implement
- Ties stakeholder roles to "Security Activities"
- Facilitates "Roles" to learn and execute "Security Activities"
- Measures progress and holds "Roles" accountable (Metrics)

Roles (PM, Architect, Designer, Implementer, etc.)

- Execute "Security Activities" leveraging automated tools and CLASP and Organization knowledge base (Vulnerability Lexicon and other Resources)

# Concepts View – CLASP Security Services

Fundamental security goals that must be satisfied for each resource:

- Authorization (access control)
- Authentication
- Confidentiality
- Data Integrity
- Availability
- Accountability
- Non-Repudiation

# Concepts View – Overview of Vulnerability View

Vulnerability

- Platforms
  - Language, OS, DB, etc.
- Resources
- Risk assessment
  - Severity
  - Likelihood
- Avoidance and mitigation periods
- Additional Info
  - Overview, description, examples, related problems

Knowledge Base Provided!

Vulnerability (Continued)

- Platforms
  - 104 types
  - Example: Buffer Overflow
- Categories:
  - Range and Type Errors
  - Environmental Problems
  - Synchronization and Timing Errors
  - Protocol Errors
  - General Logic Errors
- Exposure periods
  - Development artifact
- Consequences
  - Violated Security Service

# Role-Based View - Introduction

CLASP ties “Security Activities” to roles rather than development process steps

Roles:

- Project Manager
  - Drives the CLASP initiative
- Requirements Specifier
- Architect
- Designer
- Implementer
- Test Analyst
- Security Auditor

# Role-Based View – Project Manager

Drives CLASP initiative

Management buy-in mandatory

Security rarely shows up as a feature

Responsibilities:

- Promote security awareness within team
- Promote security awareness outside team
- Manage metrics
  - Hold team accountable
  - Assess overall security posture (application and organization)

Possibly map this to a Security Manager and Project Manager because

- PM may not have expertise
- SM may want to apply over the entire organization
- PM would still be responsible for day-to-day tasks

# Role-Based View – Requirements Specifier

Generally maps customer features to business requirements

Customers often don't specify security as a requirement

Responsibilities:

- Detail security relevant business requirements
- Determine protection requirements for resources (following an architecture design)
- Attempt to reuse security requirements across organization
- Specify misuse cases demonstrating major security concerns

# Role-Based View – Architect

Creates a network and application architecture

Specify network security requirements such as firewall, VPNs, etc.

Responsibilities:

- Understand security implications of implemented technologies
- Enumerate all resources in use by the system
- Identify roles in the system that will use each resource
- Identify basic operations on each resource
- Help others understand how resources will interact with each other
- Explicitly document trust assumptions and boundaries
- Provide these items in a written format and include diagrams (for example: network component model, application)

# Role-Based View – Designer

Keep security risks out of the application

Have the most security-relevant work

Responsibilities:

- Choose and research the technologies that will satisfy security requirements
- Assess the consequences and determine how to address identified vulnerabilities
- Support measuring the quality of application security efforts
- Document the “attack surface” of an application

Designers should

- Push back on requirements with unrecognized security risks
- Give implementers a roadmap to minimize the risk of errors requiring an expensive fix
- Understand security risks of integrating third-party software
- Respond to security risks

# Role-Based View – Implementer

Application developers

Traditionally carries the bulk of security expertise

- Instead this requirement is pushed upward to other roles

Responsibilities:

- Follow established secure coding requirements, policies, standards
- Identify and notify designer if new risks are identified
- Attend security awareness training
- Document security concerns related to deployment, implementation, and end-user responsibilities

Bulk of security expertise is shifted to designer, architect, and project manager

- Pros and Cons?

# Role-Based View – Test Analyst

Quality assurance

Tests can be created for security requirements in addition to business requirements/features

- Security testing may be limited due to limited knowledge

May be able to run automated assessment tools

- May only have a general understanding of security issues

# Role-Based View – Security Auditor

Examines and assures current state of a project

Responsibilities:

- Determine whether security requirements are adequate and complete
- Analyze design for any assumptions or symptoms of risk that could lead to vulnerabilities
- Find vulnerabilities within an implementation based on deviations from a specification or requirement

# Activity-Assessment View Overview

There are 24 CLASP “Security Activities”

- Added iteratively

Activity-Assessment View allows a project manager to determine appropriateness of CLASP activities

Guide provides

- Activity applicability
- Risks due to omission of activity
- Estimation of implementation cost
- Roles that will execute activity

# Activity-Assessment and Roles

## Table: Roles and Related Activities

The following table relates the security-related project roles to the 24 CLASP activities to be assessed.

| CLASP Activity | Related Project Role |
|---|---|
| Institute security awareness program | • Project Manager |
| Monitor security metrics | • Project Manager |
| Specify operational environment | • Owner: Requirements Specifier<br>• Key Contributor: Architect |
| Identify global security policy | • Requirements Specifier |
| Identify resources and trust boundaries | • Owner: Architect<br>• Key Contributor: Requirements Specifier |
| Identify user roles and resource capabilities | • Owner: Architect<br>• Key Contributor: Requirements Specifier |
| Document security-relevant requirements | • Owner: Requirements Specifier<br>• Key Contributor: Architect |
| Detail misuse cases | • Owner: Requirements Specifier<br>• Key Contributor: Stakeholder |

# Activity-Assessment Example Item

## Identify resources and trust boundaries

| | |
|---|---|
| Purpose: | Provide a structured foundation for understanding the security requirements of a system. |
| Owner: | Architect |
| Key contributors: | Requirements Specifier |
| Applicability: | All projects |
| Relative impact: | High |
| Risks in omission: | • Design process will consider these items intuitively, and overlook important resources. That is, the design process becomes much more *ad hoc*.<br>• Intuitive consideration is still an application of this activity, without the benefit of structure or documentation. Not performing the activity at all leads to inability to perform other CLASP design activities, thereby pushing the cost of initial security assurance to more expensive parts of the lifecycle. |
| Activity frequency: | Generally, once per iteration. |
| Approximate man hours: | • Usually 8 hours in the first iteration.<br>• < 3 hours in subsequent iterations. |

# Activity-Implementation View Introduction

Defines the purpose or goals for the “Security Activity”

Provides details regarding

- Sub goals such as
  - “Provide security training to all team members”
  - “Appoint a project security officer”
- Describes in detail how to carry out tasks or accomplish goals
  - Details which CLASP resources support these tasks
    - ex: vulnerability lexicon to examine secure coding practices
    - ex: Security Services to examine threats to a resource (threat modeling)

For example: “Perform security analysis of system requirements and design (threat modeling)”

# CLASP Roadmaps

Legacy application roadmap:

Minimal impact on ongoing development projects

Introduce only highest relative impact on security

Key steps (12 total):

- 1 – Security awareness program
- 6 – Security assessment
- 8 – Source-level security review

Green-field roadmap:

Holistic approach

Ideal for new software development

- Especially Spiral and Iterative models

Key steps (20 total):

- 1 – Security awareness program
- 2 – Metrics
- 3 – 8 Security related planning and design
- 9 – Security principles
- 12 – Threat modeling
- 16 – Source-level review
- 17 – Security assessment

# Resources

More information:

http://www.owasp.org/index.php/Category:OWASP_CLASP_Project

Downloadable “Book”

http://www.list.org/~chandra/clasp/OWASP-CLASP.zip

# Questions?


# Module 10: What are Requirements?

(Authored by Kevin Gary, Arizona State University)

Introduction to Assured Software Engineering

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

# Notices



This material is based upon work funded and supported by the Independent Agency under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center sponsored by the United States Department of Defense.

***"The hardest single part of building a system is deciding what to build... No other part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later."***

**-- Fred Brooks**

# Definitions

Software Requirements

- Descriptions of the services and constraints of a software system
- Tells what to build, not how to build it

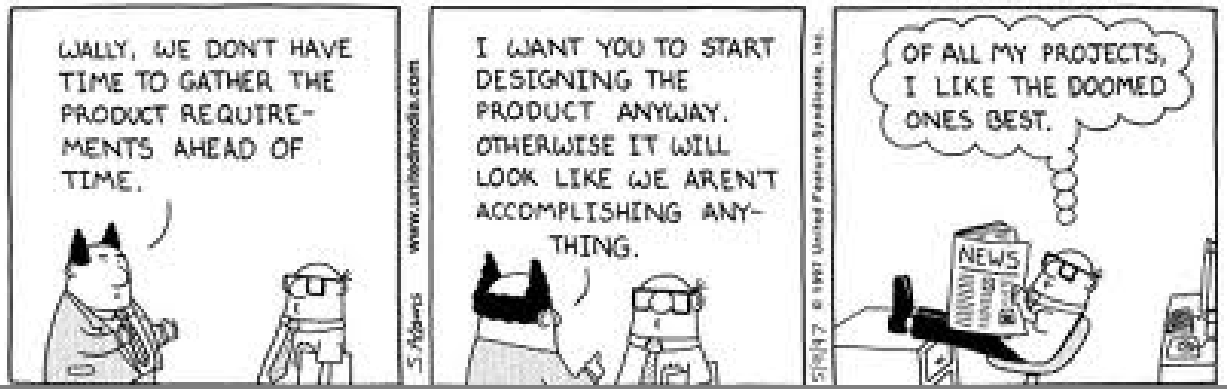
Why Spend a Lot of Time?

Requirements are the source for all future steps in the software life cycle.

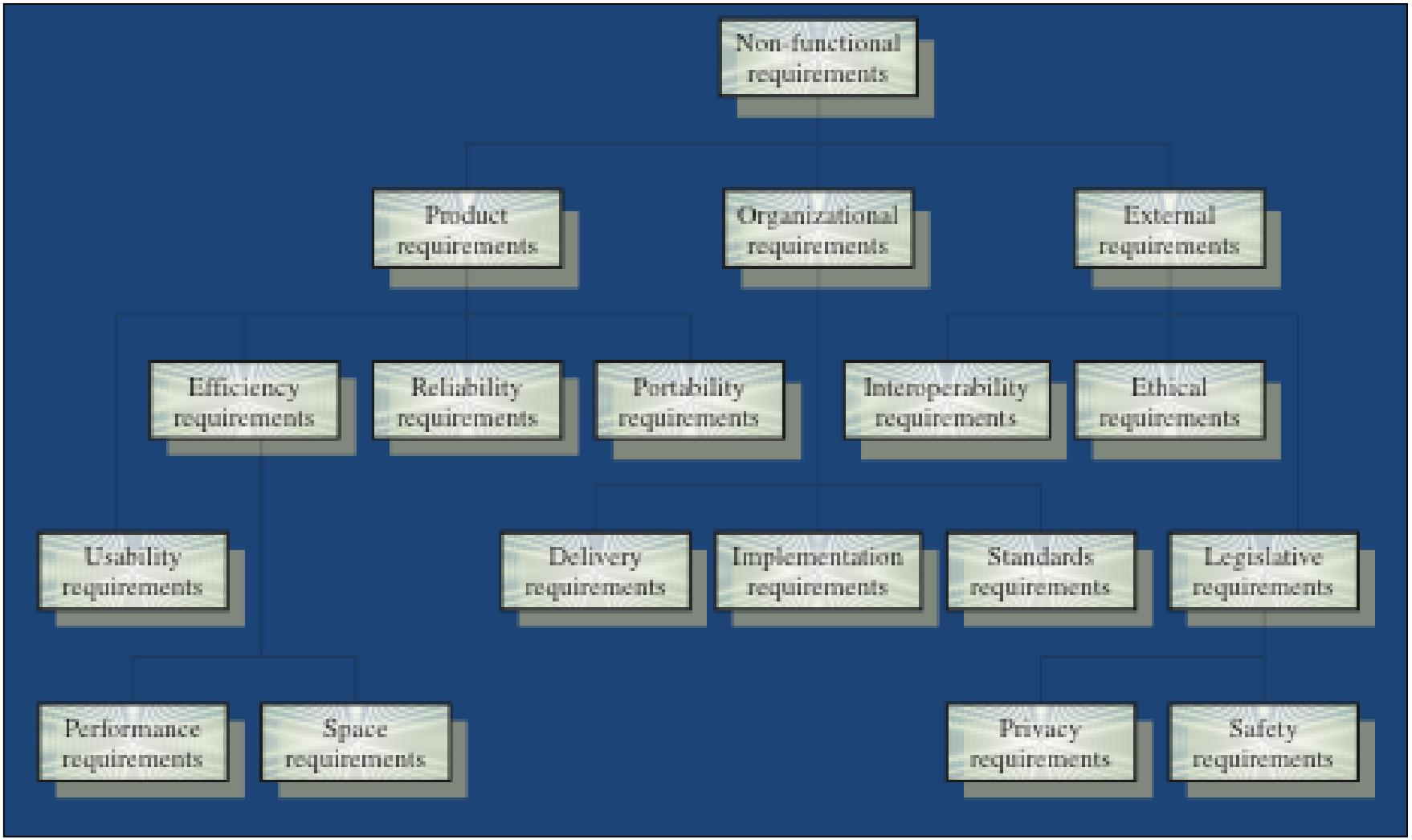
- Lays the basis for a mutual understanding
  - Consumer (what they get)
  - Software producer (what they build)
- Identifies fundamental assumptions
- Potential basis for future contracts

Better get it right - upon delivery, some software is rejected by customers.

Changes are not cheap - better make them now rather than later.



# Non-functional Requirement Types



# User vs. System Requirements

User requirements

- Statements in natural language plus diagrams of the services the system provides and its operational constraints
  - written for customers
- Should describe functional and non-functional requirements so that they are understandable by system users who don't have detailed technical knowledge
- Defined using natural language, tables and diagrams

System requirements

- A structured document setting out detailed descriptions of the system services
  - A contract between client and contractor.
- More detailed specifications of user requirements
- Serve as an initial basis for designing the system

# Requirements Examples

Specify whether the following are

- Functional, Nonfunctional, and/or Domain
  - If nonfunctional, are they Product, Organizational, or External?
- System or User

1. "The user shall be able to toggle between displaying and hiding all HTML markup tags In the document being edited with the activation of a specific triggering mechanism."
2. "The online credit-card payment facility shall support a minimum of 1000 credit-card transactions per hour".
3. "The doctor shall be able to search the patient tracking system for similar symptoms By typing keywords into a dialog box on the application's main web page."
4. "The XML-based content management system shall support UTF-8 encoding"
5. "The system shall be up and running 99.9999% of the time".
6. "The system shall support the EDI standard for medical patient data exchange"
7. "The user shall save files by selecting the'File→Save' menu choice"

# Other Requirements Classifications

Change is a Risk!

- The priority of requirements from different viewpoints changes during the development process.
- System customers may specify requirements from a business perspective that conflict with end-user requirements.
- The business and technical environment of the system changes during its development.

Enduring requirements

- Stable requirements derived from the core activity of the customer organization.
  - e.g. a hospital will always have doctors, nurses, etc.
- May be derived from domain models

Volatile requirements

- Requirements that change during development or when the system is in use.
  - e.g. In a hospital, requirements are derived from health-care policy.

# Summary

*Requirements* are the representation of what the customer *wants* not *how* you will implement it.

Requirements can be classified several ways:

- Functional vs. Non-functional
- User vs. System
- Domain-specific vs. domain-independent
- Enduring vs. Volatile

Requirements can be *annotated* to help manage change.

**Dr. Gary's tip:** *Annotate your features and requirements!!!*

- For each feature/requirement, note the classification above.
- For each feature/requirement, annotate in as many ways that are useful to managing the scope of impact when they change.

# Requirements Checklist Example

| Attribute | Values | Description |
|---|---|---|
| 1. *Verifiable* | Yes/No | Can you (did you) write a test to check for it? |
| 2. *Traceable* | GUID | Assign a unique identifier to the feature/req |
| 3. *Volatility* | *%* | 0% = Enduring, 100% = (very) Volatile |
| 4. *Behavioral* | Funct/NF | if NF, classify (slide 7-8, WhatAreReqs slides |
| 5. *Perspective* | User/System | |
| 6. *Domain-specific* | Yes/No | if Yes, describe source |
| 7. *Priority* | High/Med/Low | Later you can use “scale of 1 to 10” or biz value |

| REQ | V | T | Vol. | B | P | D | Pri | Notes |
|---|---|---|---|---|---|---|---|---|
| R1 | No | BN0 | 10% | F | U | Y | L | *Stable; but need a test* |
| R2 | Yes | XYZ1 | 50% | F | U | N | M | *Worried user may change mind* |
| R3 | No | | 80% | NF-Org | S | N | H | *We don’t understand at all!* |

# Requirements Elicitation

# Overview

What is “Elicitation”?

Who are the Players?

Where is the Information?

Techniques for eliciting requirements

How do you organize and prioritize the information?

# Requirements Elicitation

What is “Elicitation”?

*From Webster’s online (my emphasis):*

*“Main Entry:* ***elic·it***
*Pronunciation: i-'li-s&t*
*Function: transitive verb*
*Etymology: Latin elicitus, past participle of elicere, from e- + lacere to allure*
***1:to draw forth or bring out*** *(something latent or potential) <hypnotism elicited his hidden fears>*
***2:*** *to call forth or draw out (as information or a response) <her remarks elicited cheers> “*

Requirements Elicitation is the task of drawing out latent information. Make explicit that which is known.

# Requirements Elicitation

Who are the players?

- Before you charge in front of the customer, you need to know all the people involved in the process.
  - Some are information sources, others stakeholders
- Sources:
  - Users – The 'end-user" that will use your software
    - Note: This might be another system, so the representation may be the Chief Architect of that other system.
  - Buyers – The person responsible for acquiring your software and applying it to the target problem
    - Note buyer != user in many cases!
  - Experts – "Outside" people who bring experience and/or domain expertise to bear on your problem domain
- Consider getting an initial "wishlist" from each player so you understand where s/he is coming from.

# Information Sources

Where is the information?

- People sources – “players” listed on previous slide
- Documentation
  - Textbooks
  - Training materials (online or printed)
  - Reference works
- Ad hoc conversation and experience – “osmosis”
- The WWW
  - The “Google” effect
  - Discussion forums

How do you qualify and apply information from these sources?

# Requirements Elicitation Techniques

Techniques

- Individual Interviews
- Group Meetings
- Storyboarding / Prototyping
- Questionnaires
- Observation / Ethnography / User-centered design
- Perform research
- Joint Application Development (JAD)

# Interviews -1

Modality

- 2-way communication process
- Participants may be with User, Buyer, or Expert
- Structured versus Unstructured

How-to

- Set meeting expectations with interviewee a priority.
- Identify information targets to acquire.
- Meeting notes should be precise and undistilled.
  - Do not pre-analyze up front.
  - If possible, get interviewee signoff “for the record”.
- Time-sensitive: marathon sessions can lead to burnout and hasty decisions that become chains later.
- Get permission to record.
- Verify data with second sources or repeat interview.

# Interviews -2

Pros

- Richest form of information expression and capture
- Get customer buy-in

Cons

- Time-intensive
- Social tensions – burnout, personality conflict, control
- Single-source of information
- Does the interviewee have “sign-off” authority?

# Group Meetings -1


Modality

- 2-way communication
- Participants may be groups of customers, cross-functional teams, buyers, experts, focus groups, etc.
- Typically “semi-structured”
  - Want structured activities to facilitate unstructured conversations!
  - The “workshop” concept vs. the “brainstorming” concept
  - May be facilitated by groupware

How-to

- Set your expectations and targets ahead as before.
- Scheduling: Can you get undisturbed time?
- Decide on the level of structure.
  - Leave brainstorming sessions open for discovery.
  - Plan the semi-structured tasks for “workshops”.

# Group Meetings -2

How-to (cont)

- Plan the meeting environment.
  - Break-out rooms
  - Collaborative tools
  - Information capture tools
- Distribute meeting notes as before.
  - Consider assigning action items (if possible)
  - Consider establishing smaller followups – online tools?

Pros

- Groups can be self-reinforcing, build consensus
- “Real-time” requirements validation

Cons

- More complex to schedule and administer
- Social dynamics – who controls the meeting

# Storyboarding / Prototypes

Modality

- 2-way
- Participants are end users
- Provides a structure for individual or group interaction
  - Storyboarding more conducive to small group interaction

How-to

- Develop functionality based on vague requirements.
  - Throw-away code!
- Present to end user for direct feedback.
- Robustness of prototype needs only to be “sufficient to facilitate effective user feedback”.
- Technology base is chosen based on RAD, not based on the non-functional requirements.

# Storyboarding / Prototypes

Pros

- Making the solution “visible” provides you a precise means of agreeing on things with the user.
  - May also facilitate your design and test cases

Cons

- Cost to develop (need a RAD framework)
- May pigeon-hole user into early requirements commitments
- May pigeon-hole developers into early design commitments
- Throw-away solution becomes a BBOM

# Questionnaires

Modality

- 1-way communication
- May be anonymous
- Technology-assisted

How-to

- Determine your distribution list.
- Use careful vocabulary.
- Validate questions and answers – try not to show bias.
- Repeat questions to ensure consistency.
- Ensure results are quantifiable.
  - One of the benefits is using technology to reach a large group of stakeholders, so be sure you can aggregate results.

# Questionnaires

Pros

- Ability to reach a lot of stakeholders
- Objective, Quantifiable results
- Broad topic coverage
- Relatively fast
- Anonymity


Cons

- Ambiguity in questions
- Long time to design
- Poor response rates
- Restricted (rigidly structured) feedback

Lesson: Do not use in isolation, confirm data with secondary methods.

# Perform Research

Modality

- “0-way” communication
- Guidance?

How-to

- Identify appropriate information sources.
  - Market surveys
  - Industry studies – association groups, market research, technical standards organizations, research community
  - Be wary of the “Google factor”!
    - Get expert guidance!
    - Research as a team for common understanding.
- Create a taxonomy of the technical and market space.
- Attempt to get feedback on your understanding.
- Learn the user/customer vocabulary and enhance communication, not necessarily derive requirements!

# Perform Research

Pros

- 0-way means it can be planned and executed individually (or in small teams).
- Understanding current practices facilitates other techniques.

Cons

- May build a bias toward one solution space
  - You are not supposed to write your own requirements!
- Ability to assess the proper or best sources
- Time to tackle the learning curve

Again, goal is to learn enough to facilitate communication using one of the other methods, not to write your own requirements!

# Observation

Also called “ethnography”

Modality

- 1-way communication
- Real-time vs. video capture vs. event capture
- Staged environment versus real environment

How-to

- Determine modality.
  - Will you observe live or capture via video or some other technology?
- Review organizational & regulatory policies, NDAs, etc.
- Prepare a debriefing memo.
- Determine a recording format and method.
- Embed into environment with minimal intrusion.

# Observation

Pros

- Observing how customer works allows you to see how the technology benefits.
- Removes the 0th-order interpreter – the user

Cons

- Time-consuming
- Observee will not behave “naturally” (Hawthorne effect)
- Disruption to the workplace

To Note

- Ethnography is a well-known elicitation technique in research circles, and may be suitable for inception.
- Requirements elicitation using ethnography is often too time-consuming, too disruptive – simply too awkward.

# Observation

To Note (cont)

- A variant on embedded real-time observation is a staged observation for HCI evaluation.
  - Observe a user interacting with the system.
    - o Video
    - o Event tracking (mouse clicks, screen visit sequence, etc.)
  - Not really “ethnography”
  - Called “User-centered design” when designing a HCI
- Another variant on Observation is ***Apprenticing.***
  - The users train the BA on how to perform the job.
  - BA then performs in that role for some time to learn first-hand the issues for end users.
  - Time-consuming but very effective

# Joint Application Design

Modality

- Multi-way: customers, users, designers, and experts
- A cross between group meetings and prototypes
- Similar to evolutionary style of concurrent development, except the stakeholders are part of the development team

How-to

- Carefully assemble a team.
- Ensure roles are blurred – everyone is a peer and everyone's opinions are important. Design is not just for the designers.
- JAD sessions require a clear statement of the purpose of the session and its goals.
- JAD sessions are usually run by a facilitator who keeps the participants focused.
  - Can have observers, but observers must remain silent according to the rules of JAD.

# Joint Application Design

Pros

- Workshop-type feel facilitates participation
- All stakeholders feel ownership and teamwork
- All concerns laid out on the table

Cons

- Requires a skilled facilitator
- Social issues – some individuals may dominate
- Consensus building can be difficult in a large group

# Elicitation Issues

"Yes, but…"

- Issue: Software is "Infinitely malleable", so users continue to add/morph features. A *scope* issue!
- Solution: Make the software "real" (prototype).

"Undiscovered Ruins"

- Issue: "the more you find the more that remain"
- Solution: Iterate!

"User and Developer"

- Issue: *Communication gap* between the two
- Solution: Burden is on the solution provider! Use multiple techniques, reviews, and checks.

- These ideas are from Chapter 8. Leffingwell & Widrig. You should also read the Davis 2003 paper on elicitation concepts posted on the class website.

# Requirements Elicitation Summary

Elicitation and the Requirements Workflow

- Your deliverable is “uncovered” knowledge
- Format may or may not be important
  - We are assuming not important yet. Natural language is fine.
- The responsibility of the BA
- Next steps involve documenting, organizing, prioritizing requirements
- Reminders
  - This is a “soft science” – there is no recipe for success.
  - Iterate with the customer until you get convergence.
  - Cross-check results with multiple people and methods.
  - Understand what is *volatile* and what is *enduring* early!
  - Do not burn the customer out! Do not burn yourself out!
  - Do not over-commit or ask the customer to over-commit!

# Questions?


# Module 11: Requirements Analysis

(Authored by Thomas Hilburn, Embry-Riddle Aeronautical University)

Introduction to Assured Software Engineering

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

# Notices



This material is based upon work funded and supported by the Independent Agency under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center sponsored by the United States Department of Defense.

# Topics

Analysis and Modeling Basics

Analysis Report

Analysis and Modeling Issues

Analysis Process

Approaches to Analysis

- Structural Analysis
- Use Case Analysis
- OO Analysis

# Software Modeling Foundations -1

A software model is an abstract simplification of aspects or features of a software component to help software engineers understand and communicate aspects of the software to appropriate stakeholders.

Modeling Principles

- Model the Essentials – good models do not usually represent every aspect or feature of the software under every possible condition. (e.g., how primitive data types will be implemented)
- Provide Perspective – modeling provides “views” of the software under study using a defined set of rules for expression of the model within each view. (e.g., structural and behavioral views of a system)
- Enable Effective Communications – models facilitate effective communication of software information to and between project stakeholders. (e.g., a design model can provide the basis for integration planning.)

# Software Modeling Foundations -2

For our purposes, a software model is a description (textual, graphical, or mathematical) of some aspect or element used in the development of a software product.

Models in more traditional engineering fields have a long history of use and evolution.

- What are some examples of models used in other engineering fields?
- What are some examples of models used in software engineering?

A good model

- consists of multiple views, describing different aspects of the product
- makes use of abstraction and information hiding

# Software Modeling Foundations -3

Abstraction is a development approach that emphasizes essential external features and behavior, and obscures details about internal structure and logic.

- Abstraction is a basic concept in modern design and development.

Information hiding is a concept for “abstracting” or hiding the details of a module not needed by a user of a module.

Data encapsulation is collecting the essential features and behavior of a data object into a single entity.

# Requirements Analysis

The purpose of requirements analysis is to take information provided by the stakeholders, and analyze and model that information so that

- the developers better understand the stakeholder needs and requirements
- the functional and non-functional requirements can be specified clearly, correctly, and completely
- there is solid foundation for design, construction, and testing of the software product

During the requirements analysis phase we will build a **"conceptual model"** for our system that will support effective communication between users, domain specialists, and application developers.

# Conceptual Model

A representation of the key concepts in a problem domain from which a system's functional requirements can be derived

- Elements of the conceptual model are sometimes referred to as the **"conceptual design"**. Although it may contain components of an eventual solution, its purpose is to better understand "what" the software should to.

For our purposes, the conceptual model will contain the following:

- a context diagram
- a UI prototype
- a Use Case model
- a Conceptual Design (class diagrams and sequence diagrams)

# Requirements Analysis Process

What should it include?

- Purpose, Entry Criteria
- Phases
  - Select Analysis Techniques
  - …
  - …
  - Review and Revise Analysis Results
  - Prepare and Submit Analysis Report
- Exit Criteria

# Analysis & Modeling Issues

How do we model the functional requirements for a software system?

- Concentrate on the external behavior of a system.
  - However, “internal” behavior must also be considered so that non-user functional requirements can be determined.

How do we go from a requirements specification to design?

- This depends on the application domain and the analysis methodology chosen?

How do we keep from making design decisions too early?

- Focus on the goal of “understanding” rather than “solving”.

# Analysis Models

There are lots of them:

- Structured Text
- Context Diagram
- User Stories
- Use Cases
- Data Flow Diagrams
- Entity Relationship Diagrams
- Interaction diagrams
- GUI Diagrams
- Data Dictionaries
- Decision Trees
- Dialog Maps
- Module Diagrams (class diagrams, structure charts)
- State Transition Diagrams
- P-Specs
- Petri Nets
- Z Schemas
- Formulas (e.g., first order logic, CSP, Temporal Logic)
- …

# Context Diagram

In software engineering and systems engineering a Context Diagram is a diagram that represents the Actors outside a system that could interact with that system.

This diagram is the highest level view of a system.

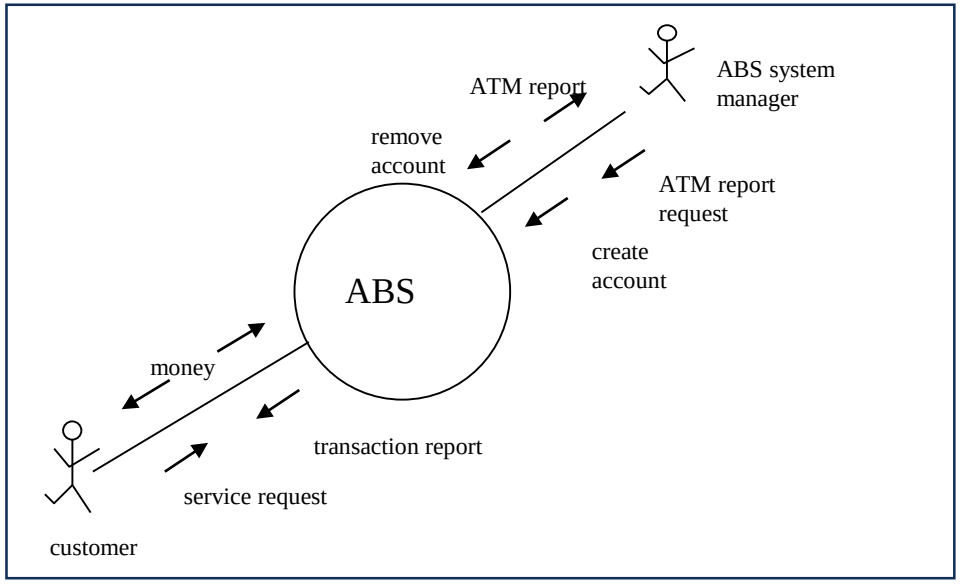
Context diagrams can be developed with the use of two types of building blocks:

- *Entities (Actors):* labeled boxes; one in the center representing the system, and around it multiple boxes for each external actor
- *Relationships:* labeled lines between the entities and system

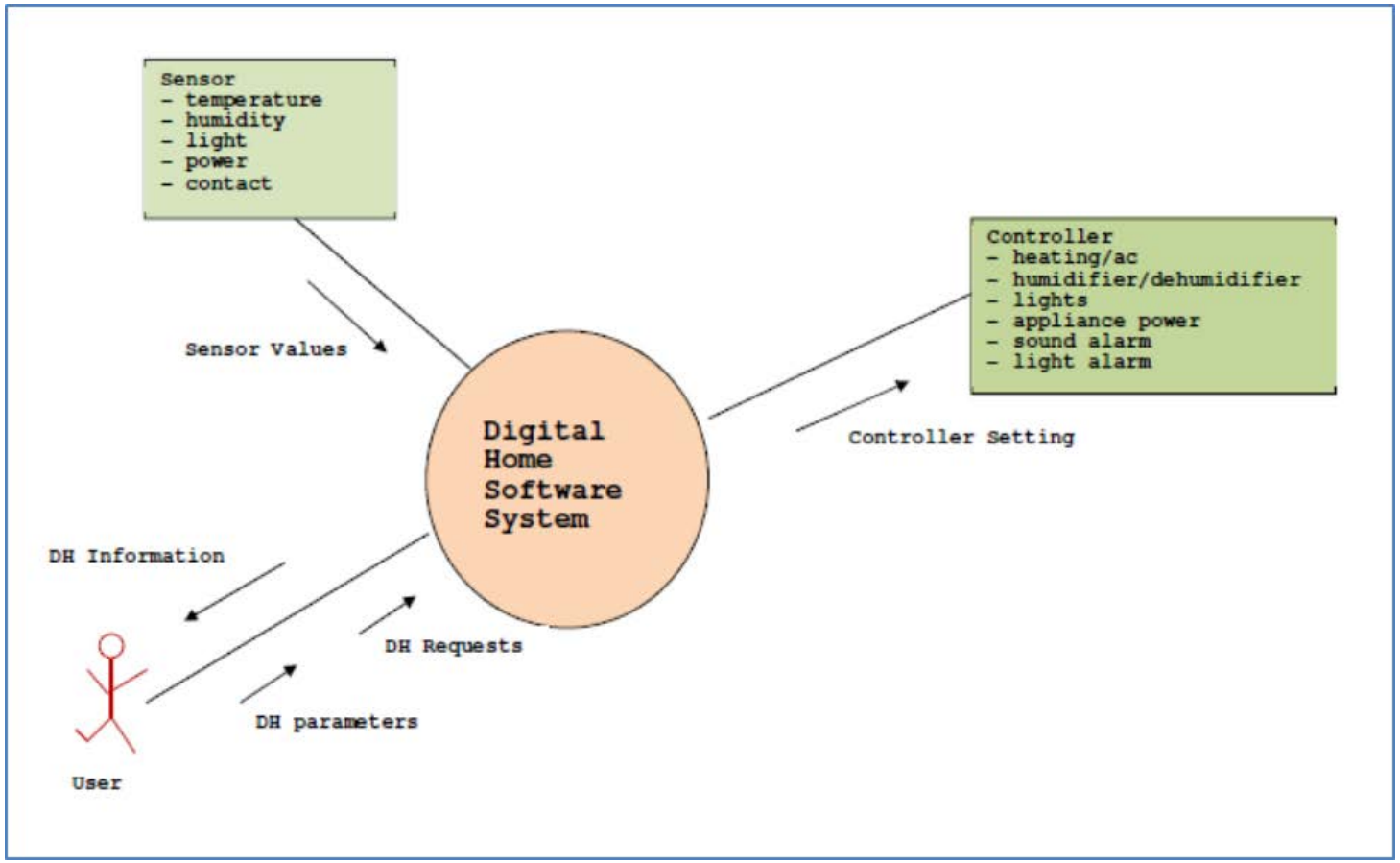
# An Example Problem

Problem Statement: Develop an Automatic Banking System (ABS) that will interact with banking customers, through an ATM, to provide automated banking services (deposit money, withdraw money, provide account information - balance, transaction information, etc.).

- An ABS manager can create a new account or close-out an existing account.



# DigitalHome Context Diagram



# Requirements Modeling Principles

Requirements models help stakeholders better understand the problem:

- Enable a user to understand how human-machine interaction will occur.
- Provides means for requirements  verification and validation.
- Supports clear, correct, precise, complete requirements specification.
- Forms basis for development of software architecture and design.

Need a basic understanding of the problem before requirements modeling can begin.

Should use multiple views of requirements.

Where appropriate, requirements models should use decomposition, abstraction and information hiding.

# Requirements Analysis Modeling

Static (Structural) Modeling

- models the system elements and their relationships without regard to time

Dynamic (Behavioral) Modeling

- models how the system behavior changes over time

Data Modeling

- models the key data elements and their relationships

Popular Approaches

- Structural Analysis
- Object-Oriented Analysis
- Formal Modeling

# Structural Analysis

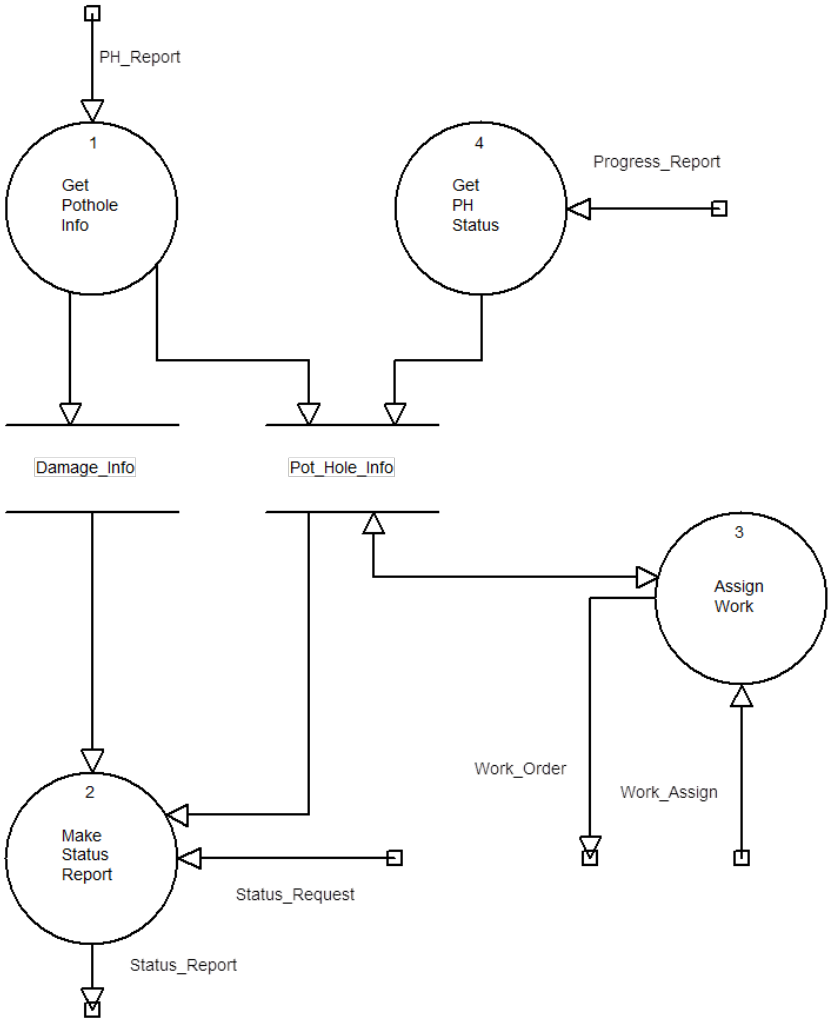
A methodology used to build a system model that depicts information flow and content

- Special notation and graphical symbols are used to describe and partition the functionality of system.

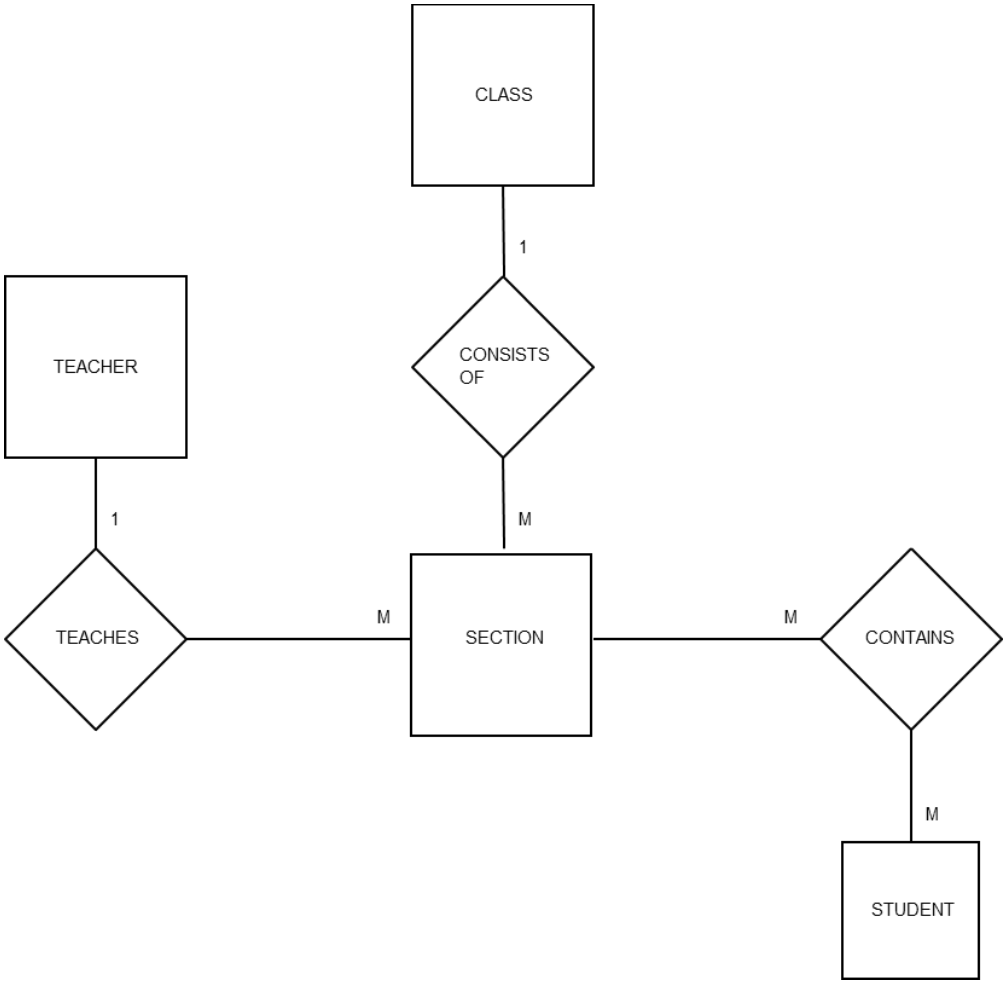
Elements

- Data Flow Diagram (DFD)
- Data Dictionary (DD)
- Process Specification (P-Spec)
- Entity-Relationship Diagram (ERD)
- State Transition Diagram (STD)

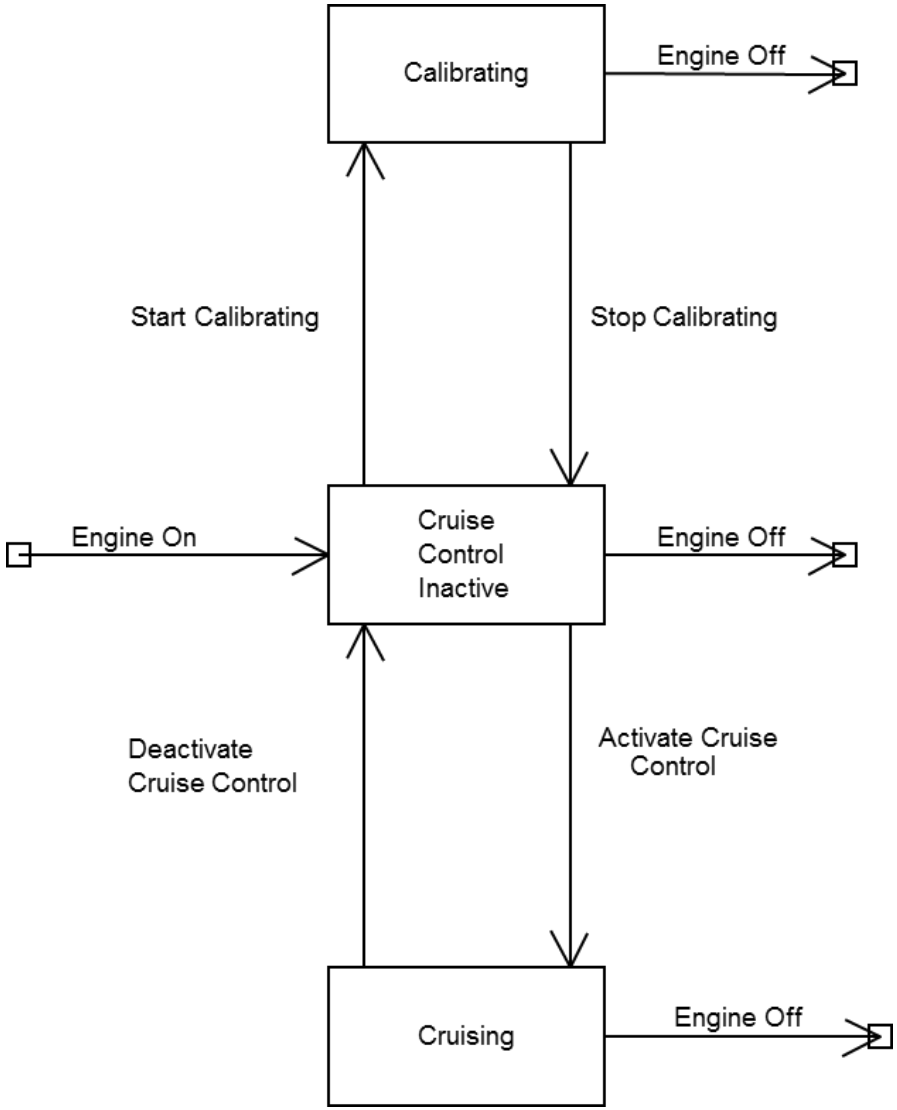
# Data Flow Diagram



# Entity Relationship Diagram



# State Transition Diagram



# Structured Design

Transforms a requirements specification that was modeled using structured analysis into a design

- based on DFD, ERD, DD, STD

Uses a Hierarchical Design Structure

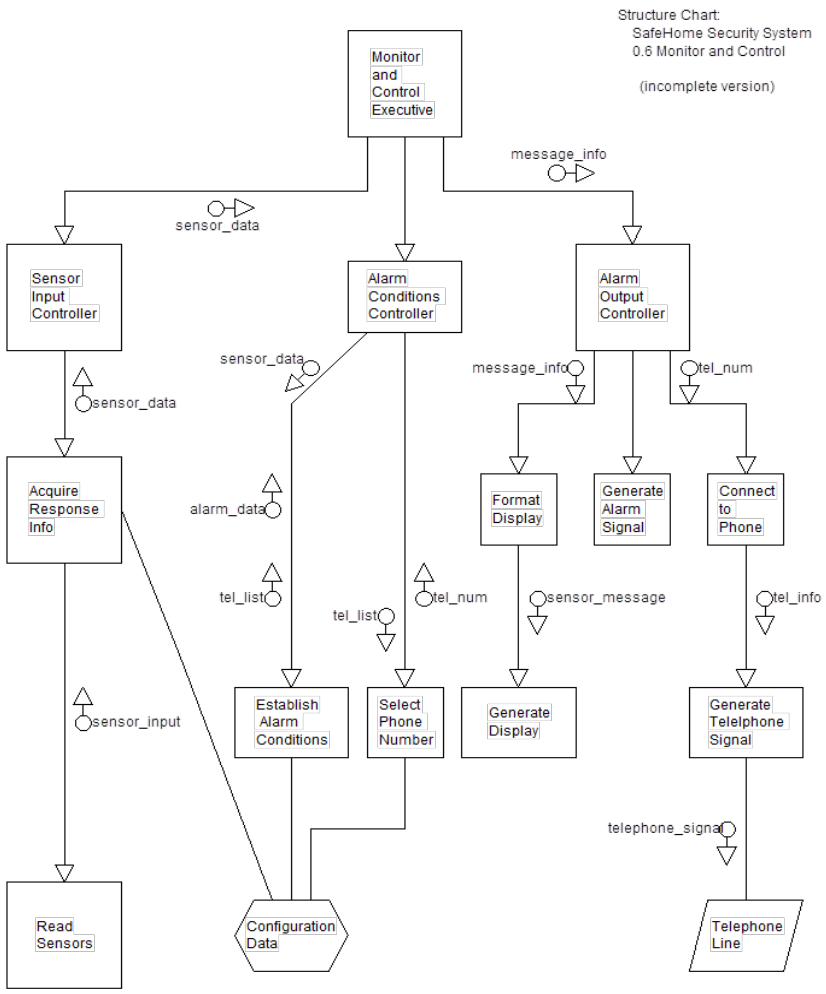
Design Elements

- data dictionary (refined from DD in SRS)
- structure chart
- procedure design elements
  - flow charts, tables, program design language (PDL)

Problem: The transformation from structured analysis to a design architecture is not easy or natural:

- The analysis and design models are orthogonal.
- A technique called “transform mapping” is one popular approach.

# Structure Chart



# Object-Oriented Analysis

Object-oriented development can be viewed as an integration of the functional-driven approaches ("structural" approaches) and the data-driven approaches (for database systems).

Object-oriented analysis (OOA) is concerned with developing and modeling software requirements that is:

- based on Objects from the problem domain
- the first step in an evolutionary refinement process that moves from OO analysis to OO design to OO implementation
- a high-level conceptual description of how the solution will be organized

OOA Elements

- Use Case Model
- Package/Class Diagrams
- Interaction Diagrams: Sequence, Collaboration
- State Diagram

# Unified Modeling Language (UML)

A standardized general-purpose modeling language used in object-oriented software development

Developed by Grady Booch, Ivar Jacobson and Jim Rumbaugh at Rational Software in the 1990s

Used to analyze, specify, design, construct and document the artifacts of software-intensive systems

Tools

- ArgoUML – http://argouml.tigris.org/
- TOPCASE – http://www.topcased.org/
- Visual Paradigm – http://www.visual-paradigm.com/

# Messages to Remember

Requirements analysis involves study, analysis and modeling of the problem to be solved, in order to ensure effective requirements specification.

The requirements model serves as a basis for requirements V&V and development of system architecture and design.

# Questions?


# Module 12: Use Case Models

(Authored by Thomas Hilburn, Embry-Riddle Aeronautical University)

Introduction to Assured Software Engineering

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

# Notices



This material is based upon work funded and supported by the Independent Agency under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center sponsored by the United States Department of Defense.

# Use Case Basics -1

In 1987, Ivar Jacobson [Jacobson 1992] introduced the use case concept for modeling the behavior of a system.

Use case modeling is a technique that can help in the difficult task of eliciting, analyzing and specifying requirements.

- Use cases are now widely used to model and analyze software requirements.
- Some believe they should be the primary requirements modeling tool, almost to the exclusion of other techniques.

# Use Case Basics -2

Use Case (UC): a structured statement of software functionality

- a representation of one or more functional software requirements
- “the ways in which a user uses a system” [Jacobson 1992]
- “a collection of possible sequences of interactions between the system under discussion and its external actors, related to a particular goal.” [Cockburn 1997]

There are lots of different opinions, approaches, techniques, and styles associated with use case modeling.

Use cases are most helpful in requirements elicitation and analysis.

Use cases are also helpful with system test planning.

# Use Case Basics -3

A use case represents an external view of interaction with a system.

Use cases can be used to develop a system users manual.

Use cases are stories about using a system (that represent the system functional needs).

Although use cases are often used in OO analysis and are a formal part of the UML, they are not “object-oriented”.

- Use cases are “functional” in nature and work well with functional focused development methodologies.

Use cases are like “black boxes”; they describe what the system must do, not how it will do it.

# Use Case Basics -4

The number and granularity of use cases influences the time and difficulty to understand, maintain, and manage the requirements.

Some of the advantages cited for use case modeling are:

- Use cases are a good communication tool between developers and users, clients and domain experts.
- Use cases are a convenient way to capture and categorize the behavior of a system.
- Use case modeling supports effective requirements elicitation. Use case construction leads very naturally to questions and research about unsupported assumptions and hidden requirements.
- Use cases can be used to validate the behavior of a system during and after development.

# Developing Use Case Model

Review customer need statement or other information about functional requirements.

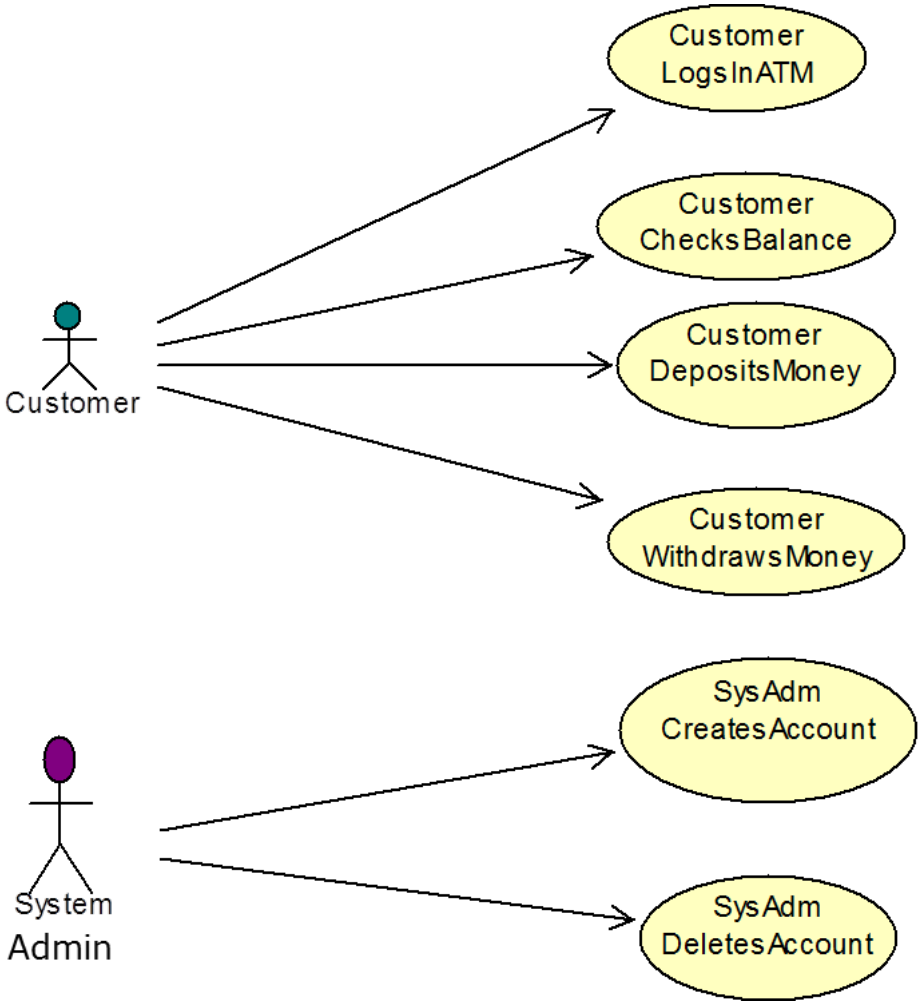
- In particular, for a class project, review the Need Statement and Elicitation Report focusing on the user classes and user stories.

Determine the system boundary and the external entities (develop a Context Diagram).

List the actors in the system. Candidate actors are:

- people that use the system
- other systems that use the system
- people that  install, start up, operate, or maintain the system

# Use Case Diagram



# Use Case Name Format

*Actor-Action-Object*

- Actor:  the role that a user (or another external entity) plays with respect to the system
- Action: functionality requested by an actor
- Object: item acted on by an Actor

Example Name: *CustomerDepositsMoney*

- Or *DepositMoney* if the Actor is apparent or there are multiple actors.

# Goals and Actors

The use case goal represents the defining purpose of a use case. It represents what is intended to be achieved by initiator of the use case.

- e.g. a bank customer may have the goal of withdrawing money from her account.

Actors are not only roles played by people, but may be organizations, software, and machines. Actors can be classified as follows:

- primary actor: the principal actor that calls on the system to fulfill the UC goal (e.g., bank customer)
- supporting actor: an actor that provides a service to the system (e.g., a bank accounts database that is external to the system)
- offstage actor: an actor that has an interest in the use case, but is not primary or supporting (e.g., a bank examiner)

# Use Case Scenarios

Scenario: formatted description of the steps required for the completion of a Use Case – to achieve the UC goal. One particular story of using the system.

Main Scenario: a scenario that describes the successful completion of the UC goal.

- It is possible for there to be more than one success scenario.

Alternate Scenario: a scenario that represents an alternate to the main scenario that also achieves the UC goal.

A scenario that does not lead to the UC goal is called a failure scenario.

- An exception is a condition that prevents successful completion of the UC goal.

A given use case would typically be made up of multiple scenarios.

# Use Case Template 1

**Use Case ID:** <Use Case ID, e.g. UC-1>
**Use Case Name:** <Use Case Name, e.g. Customer Deposits Money>
**Primary Actor:** <has goals/needs satisfied by the UC, e.g. Customer>
**Goal:** <high-level description of use case purpose>
**Pre-Conditions:** <list of conditions that must be true before Use Case can be executed >
**Post-Conditions:** <list of conditions that must be true when the Scenario is complete >

**Main Scenario:**

| Step | Actor Action | Step | System Reaction |
|---|---|---|---|
| | | | |
| | | | |
| | | | |

**Alternate Scenario:**

| Step | Actor Action | Step | System Reaction |
|---|---|---|---|
| | | | |
| | | | |

**UC GUI:** < names of GUIs used in the Scenario>
**Exceptions:** < list of failure conditions that could occur during execution of the scenario and a description of how the system would respond> >
**Use Cases Utilized:** < list of other use cases used>
**Notes and Issues:** < description of supporting actors, concurrency of actions, and any additional information such as nonfunctional requirements related to the UC.>

# Use Case Template 2

For the *Actor Action* and *System Reaction* fields, be careful about the level of detail in the description:

- Include enough detail so that developers can understand the meaning of the action or reaction.
  - e.g. “Customer requests bank loan” would be better than “Customer requests money”.
- There is no need, especially at an early stage, to include too much detail.
  - e.g. “Customer enters address” may be better than “Customer enters street address, city name, state name, and zip code”

The *Exceptions* field would contain the conditions that would required special handling by the system.

- In some approaches to use case modeling “exception handling” is specified with alternate scenarios or scenario extensions.

# Use Case Template 3

The *Use Cases Utilized* field indicates structuring of a scenario so that it is an extension of an existing use case, or by using “sub-interaction”, where a system may react to some action by initiating another use case.

- e.g., a use case of Customer_Login_Account might initiate a use case of  Manager_Checks_PIN.

Use case modeling is part of the evolving nature of requirements elicitation, analysis and modeling; hence, in the early stages of use case development, some items (such as Alternate Use Cases and Use Cases Utilized) may not be clear or complete.

There is no standard UC template; a variety of styles and formats are used.

- One popular alternative is to depict the scenario steps in a single column, intermixing the actor action and the system reaction.

# Use Case Description Example

**Use Case :** CustomerDepositsMoney
**Primary Actor:** Customer
**Goal:** This Use Case enables a bank Customer to make a deposit via an ATM.
**Preconditions:**
1. Customer has logged into the ATM
2. a service menu is displayed

**Post Conditions:**
1. Customer deposit amount added to balance
2. Customer logged off (if answer NO) or Services Menu displayed (if answer YES)

**Main Scenario:**

| Step | Actor Action | Step | System Reaction |
|---|---|---|---|
| 1 | Customer selects deposit menu item | 2 | ATM requests deposit amount |
| 3 | Customer enters amount of deposit | 4 | ATM requests the customer to place money in the deposit slot |
| 5 | Customer places money in deposit slot | 6 | ABS adds deposit amount to account balance and files a transaction report about the deposit. |
| | | 7 | ATM asks if customer wants to request another service |
| 8a | Customer answers NO | 9a | ATM prints transaction report and logs the customer off |
| 8b | Customer answers YES | 9b | ATM displays services menu |

**UC GUIs:** GUI3 (Service Menu), GUI6 (Deposit GUI)
**Exceptions:**
1. failure to select deposit menu item

response: error message displayed after 30 seconds; customer logged off

2. non-positive deposit amount entered

response: error message displayed; customer re-enters deposit

3. failure to enter deposit in deposit slot

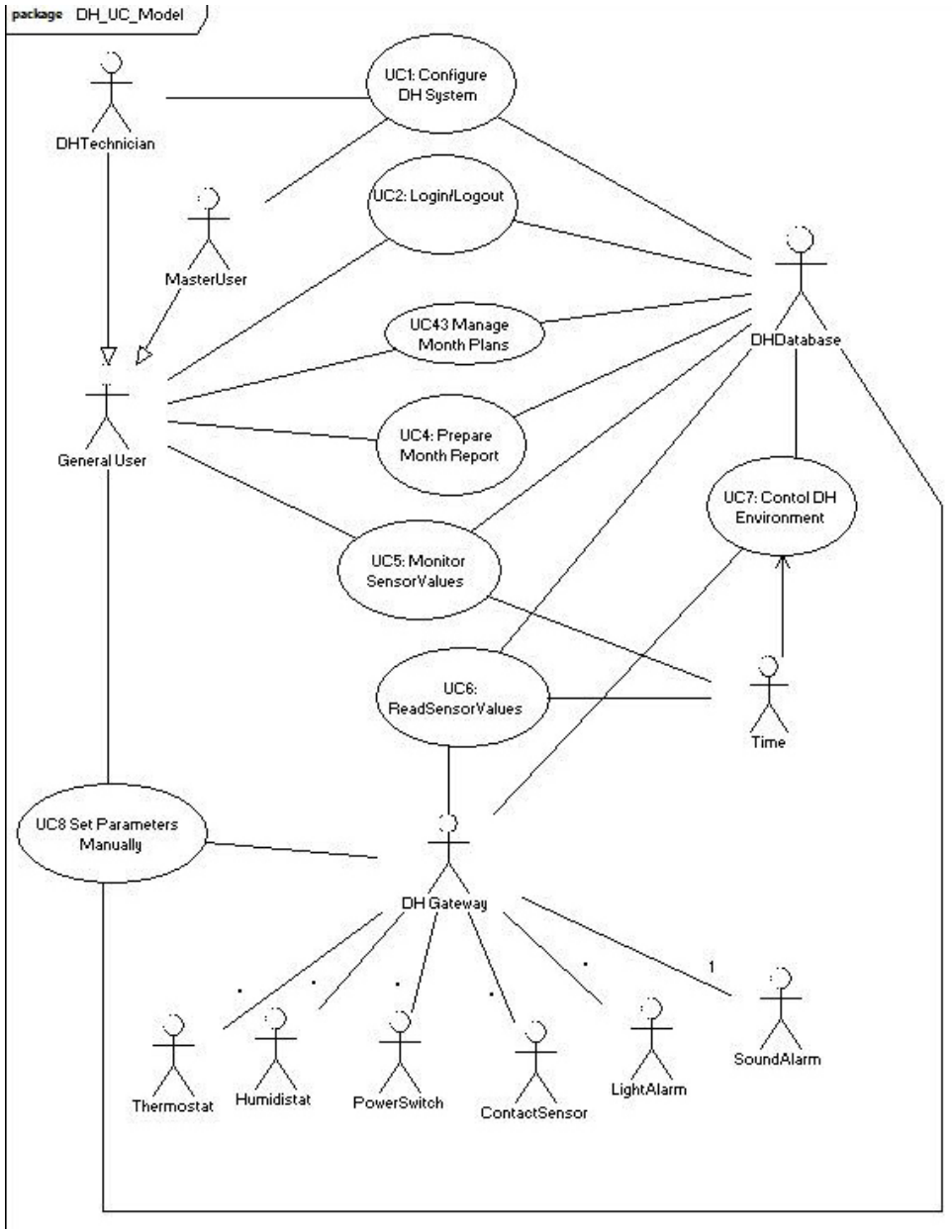
response: error message displayed after 30 seconds; customer logged off

4. failure to enter YES or NO

response: error message displayed after 30 seconds; ATM prints transaction report and logs the customer off

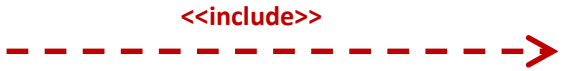
**Use Cases Utilized :** None
**Notes and Issues:** None

# DigitialHome Use Case Diagram 1



# Use Case Associations

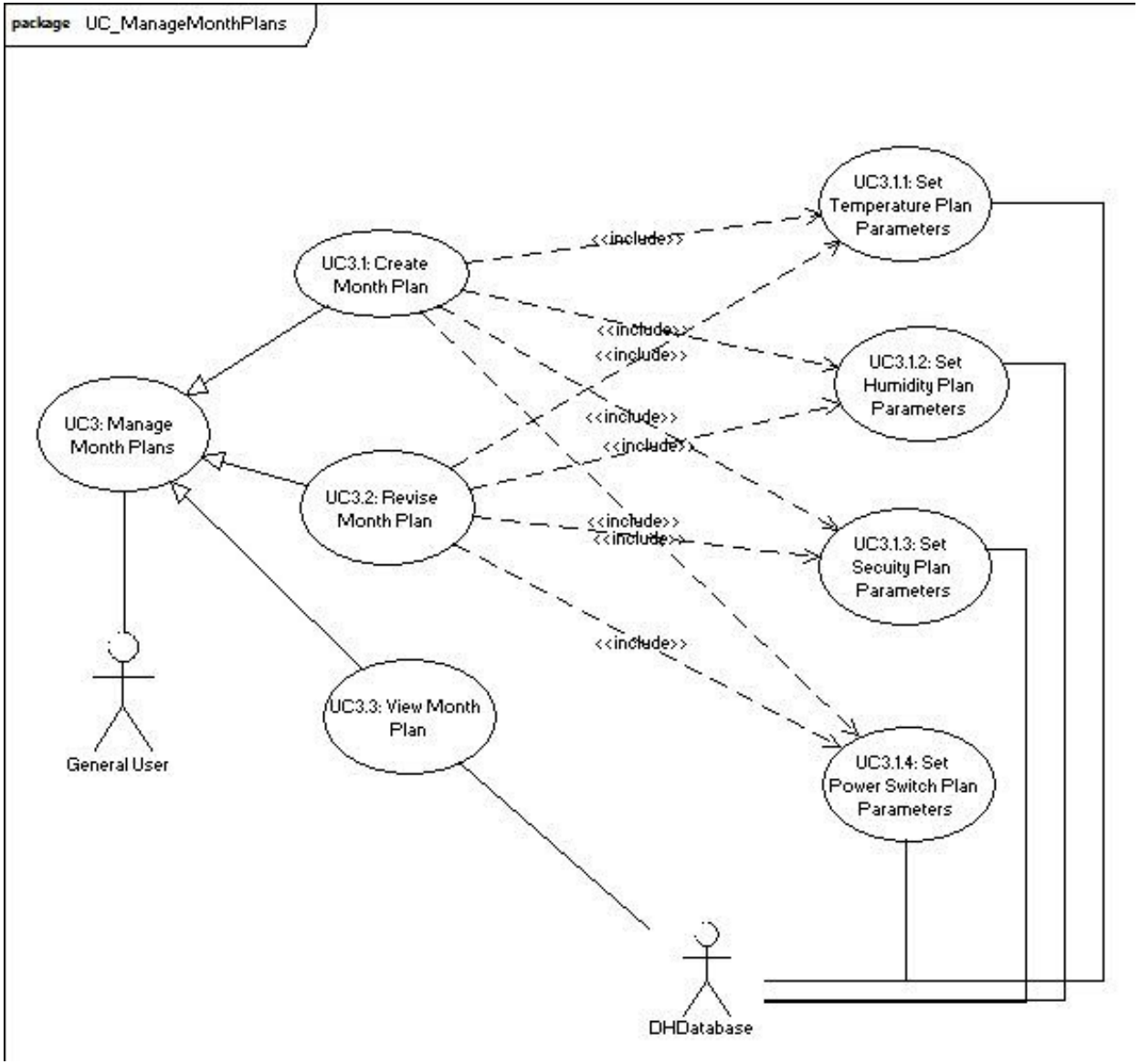

Extension is an association between two use cases where the functionality of some use cases is similar but could have some deviations or additions depending on the scenario.




Inclusion is an association between use cases where there is a chunk of functionality that is inclusive in multiple cases and can be abstracted into a separate model and essentially "included" in other use cases

Generalization is an association between use cases where the basic functionality is the same, but specific implementation details may differ

# DigitialHome Use Case Diagram 2



# DigitalHome Use Case Scenario

**Use Case ID**: UC3
**Use Case**: Manage Month Plans
**Goal**: Manage the monthly plans which control the environment of a Digital Home.
**Primary Actor**: General User
**Secondary Actor:** DH Database
**Pre**:
1. User is logged into her/his DH account.
2. DH Main Page is displayed on the user display device.

**Post:**
3. Month Plan operation has been completed
4. DH Main Page is displayed on the user display device.

**Main Success Scenario:**

| Step | Actor Action | Step | System Reaction |
|---|---|---|---|
| 1 | Select Manage Month Plan option | 2 | Requests user to enter month and year |
| 3 | Enter month and year | 4 | Display month calendar |
| | | 5 | Display "Which operation do you want to perform:<br>a. Create a month plan.<br>b. Revise a month plan.<br>c. View a month plan.<br>d. Exit Month Plan Page |
| 6A | Select a. | 7A | Invoke UC 3.1 |
| 6B | Select b. | 7B | Invoke UC 3.2 |
| 6C | Select c. | 7C | Invoke UC 3.3 |
| 6D | Select d. | 7D | continue |
| | | 8 | Display DH Main Page |

**UC GUIs:** DH Main Page, Thermostat Month Plan Page, Humidity Month Plan Page, Security Month Plan Page, Power Switch Month Plan Page

**Exceptions**:
5. User fails to make an entry.
- Response: Timeout after 5 minutes and logout of system

6. Improper data entered (e.g., wrong data type or out of range data)
- Response: Prevented with GUI design (drop down lists, list boxes, bounded range components, etc.)

**Use Cases Utilized**: UC-3.1, UC-3.2, UC-3.3

**Notes and Issues**: When the DH Technician sets up and configures a DH System, he/she sets the default values for the system (see UC-1 Configure DH System)

# Review of Use Case Model -1

A thorough review of the Use Case model is critical.

- A major error in the UC model (if not detected early) can cause major problems in later stages of development.
- Where possible, include customers, users, domain experts, and other members of the development team as part of the review group.
- Make up a checklist for the review and follow it carefully.
  - The following slides will give you some ideas for the checklist.

# Review of Use Case Model -2

Use Case Coverage

- Each elicited functional requirement has at least one UC designated.
  - That is, all the UC goals taken together cover all of the functional requirements.

Use Case Level and Granularity

- UCs are not at too low/detailed level
- UCs are not at too high/broad level
- The level of detail is consistent across the UCs.
- The number of UCs is appropriate for the size and complexity of the system being developed.

# Review of Use Case Model -3

Use Case Template

- Each item in the UC Scenario template has been addressed.
- The Pre-Conditions and Post-Conditions represent system states that are appropriate for the achievement of the UC goal
- The sequence of steps in the main success scenario correspond to expected interaction between the actors and the system.
- The scenario does not require any non-domain information to be understood.
- All functional elements concentrate on what the system should do and not how it should be implemented.
- Any condition that could prevent a success scenario from being completed is listed in the Exceptions field.

Use Case Diagram

- The UC Diagram includes all actors and UCs, and their correct relationships

# Problems with Use Cases -1

Use in OO development

- Use Cases are part of the Unified Modeling Language and are widely used in objected-oriented analysis.
- Since use cases are “function-centric”, there is a danger that when UCs are used as part of OOA, an analyst might drift into a functionally focused approach to development of the system.

The advantages of use case modeling can be lost in a sea of UC explosion if there is not some control over the number of use cases, their scope and their level of detail.

- A UC modeler should not lose sight of the UC model’s purpose in organizing the description of system behavior and its role as a communication tool between system stakeholders.
- As in other areas of software  development, use abstraction, modularity and information hiding to reduce complexity.

# Problems with Use Cases -2

When are you finished?

- The true answer might be not until the system is delivered. When systems are developed iteratively, one could see changes and enhancements to the UC model near the end of the development life-cycle.
- However, if one is interested in some sort of quasi-closure at a given stage of development, the test would be agreement from the stakeholders that the UC model accurately depicts a description of system behavior.

Scaling up for a larger system.

Too much emphasis on UI design details.

- Do not let UI design drive requirements analysis.
- At this stage simple mock-ups are fine.

# Use Cases and Requirements

Functional requirements can be specified in a number of different ways.

- Use Cases only
- Separate Use Case and SRS Documents
- Incorporate Use Cases in SRS as the organizing focus

For your project, you will document your Use Cases in the Analysis Report and then organize the SRS around Ignite features and use cases.

As part of the SRS development and review you need to insure that the UCs and the functional requirements correlate.

# Messages to Remember

Use cases are an informal, but explicit way to specify the functional requirements for a system.

The Use Case can be used to support system test planning and the development of user manuals.

As in any software artifact a thorough review of the Use Case model is necessary to insure it is correct and complete.

# Questions


# Module 13: Security Requirements and SQUARE Overview

Introduction to Assured Software Engineering

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

# Notices



This material is based upon work funded and supported by the Independent Agency under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center sponsored by the United States Department of Defense.

# Outline

Requirements Engineering

Introduction to SQUARE

SQUARE for Acquisition

# Requirements Engineering

# Requirements Engineering Issues

RE defects cost up to 200 times more once fielded than if caught in requirements engineering.

Reworking defects consumes >50% of project effort.

>50% of defects are introduced in requirements engineering.

Takeaway: Errors during requirements engineering are costly!

# Requirements Problems

Requirements

- identification may not include relevant stakeholders
- analysis may or may not be performed
- specification are typically haphazard

# Effects of Requirements Problems

Bad requirements cause projects to

- exceed schedule
- exceed budget
- have significantly reduced scope
- deliver poor-quality applications
- deliver products that are not significantly used
- be cancelled

# Discussion

Have you seen examples of requirements problems that caused projects to

- exceed schedule
- exceed budget
- have significantly reduced scope
- deliver poor-quality applications
- deliver products that are not significantly used
- be cancelled

# Security Requirements

Address security in a particular application

Are often ignored in the requirements elicitation process

Incur high costs when incorporated later

Must be addressed early

# Security Requirements Engineering Issues – Example

Cost of Fixing Vulnerabilities Later

| Stage | Critical Bugs Identified | Cost of Fixing One Bug | Cost of Fixing All Bugs |
|---|---|---|---|
| Requirements | | $139 | |
| Design | | $455 | |
| Coding | | $977 | |
| Testing | 50 | $7,136 | $356,800 |
| Maintenance | 150 | $14,102 | $2,115,300 |
| **Total** | **200** | | **$2,472,100** |

Cost of Fixing Vulnerabilities Early

| Stage | Critical Bugs Identified | Cost of Fixing One Bug | Cost of Fixing All Bugs |
|---|---|---|---|
| Requirements | | $139 | |
| Design | | $455 | |
| Coding | 150 | $977 | $146,550 |
| Testing | 50 | $7,136 | $356,800 |
| Maintenance | | $14,102 | |
| **Total** | **200** | | **$503,350** |

**As can be seen, identifying defects early in the lifecycle reduced costs by nearly $2 million.**

# Microsoft Security Lifecycle Results

**Microsoft Windows: 45% Fewer Vulnerabilities in Windows Vista**

Windows Vista was the first Microsoft operating system to benefit from the SDL. After the first year, Windows Vista had 45% fewer vulnerabilities than Windows XP. In a comparison of security vulnerabilities, Windows Vista also fares better than competing operating systems.

**Microsoft SQL Server: 91% Fewer Vulnerabilities in SQL Server 2005**

SQL Server serves as an excellent example for security improvements resulting from incorporating the SDL. Within the three years after release, Microsoft has issued three security bulletins for the SQL Server 2005 database engine.

Reference:

http://www.microsoft.com/security/sdl/learn/measurable.aspx

# Security Requirements Methods -1

SQUARE

CLASP

Core Security Requirements Artifacts

SREP

Security Patterns

TROPOS

Others

# Security Requirements Methods -2

SQUARE

- Security Quality Requirements Engineering
- Nine-step process
- SQUARE-Lite
- SQUARE for Privacy
- SQUARE for Acquisition
- Can be used with existing requirements engineering process

# SQUARE Methodology

## What is it? Who is involved?

# SQUARE

Developed by the CERT Division at the SEI, Carnegie Mellon University

Stepwise methodology for eliciting, categorizing, and prioritizing security requirements for information technology systems and applications

Security requirements are quality attributes

# SQUARE

Who is involved?

- stakeholders of the project
- requirement engineers with security expertise

In SQUARE, security requirements are

- treated at the same time as the system's functional requirements, AND
- specified in the early stages of the SDLC
- specified in similar ways as software requirements engineering and practices
- determined through a process of nine discrete steps

# SQUARE Steps

## The Nine Steps

# SQUARE Steps

1. Agree on definitions.
2. Identify assets and security goals.
3. Develop artifacts to support security requirements definition.
4. Assess risks.
5. Select elicitation technique(s).
6. Elicit security requirements.
7. Categorize requirements.
8. Prioritize requirements.
9. Inspect requirements.

# Step 1

| **1** | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|
| **Def.** | *Goals* | *Artifacts* | *Risk* | *Technique* | *Elicit* | *Categorize* | *Prioritize* | *Inspect* |

## Agree on Definitions

- Requirements engineers and stakeholders agree on a set of definitions.
- Process is carried out through interviews.
- Exit criteria: documented set of definitions
- Examples: non-repudiation, denial-of-service (DoS), intrusion, malware

# Step 2

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|
| **Def.** | **Goals** | *Artifacts* | *Risk* | *Technique* | *Elicit* | *Categorize* | *Prioritize* | *Inspect* |

## Identify Assets and Security Goals

- Identify assets to be protected in the system.
- Goals are required to identify the priority and relevance of security requirements.
- Security goals must support the business goal.
- Goals are reviewed, prioritized, and documented.
- Exit criteria: one business goal, several security goals

# Step 3

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|
| **Def.** | **Goals** | **Artifacts** | *Risk* | *Technique* | *Elicit* | *Categorize* | *Prioritize* | *Inspect* |

## Develop Artifacts

- Collect or create artifacts that will facilitate generation of security requirements.
- Jointly verify their accuracy and completeness.
- Examples: system architecture diagrams, use/misuse case scenarios/diagrams, attack trees, templates and forms

# Step 4

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|
| **Def.** | **Goals** | **Artifacts** | **Risk** | *Technique* | *Elicit* | *Categorize* | *Prioritize* | *Inspect* |

Perform Risk Assessment

- Identify threats to the system and its vulnerabilities.
- Calculate likelihood of their occurrence. Classify them. This will also help in prioritizing requirements later.
- Risk expert might be required.
- Exit criteria: documentation of all threats, their likelihood and classifications

# Step 5

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|
| **Def.** | **Goals** | **Artifacts** | **Risk** | **Technique** | *Elicit* | *Categorize* | *Prioritize* | *Inspect* |

## Select Elicitation Technique

- Select appropriate technique for the number and expertise of stakeholders, requirements engineers, and size and scope of the project.
- Techniques: structured/unstructured interviews, **accelerated requirements method** (ARM), soft systems methodology, issue based information systems (IBIS), Quality Function Deployment

# Step 6

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|
| **Def.** | **Goals** | **Artifacts** | **Risk** | **Technique** | **Elicit** | *Categorize* | *Prioritize* | *Inspect* |

Elicit Security Requirements
***(Heart of SQUARE)***

- Execute the elicitation technique.
- Avoid non-verifiable, vague, ambiguous requirements.
- Concentrate on what, not how.
  Avoid implementations and architectural constraints.
- Exit criteria: initial document with requirements

# Step 7

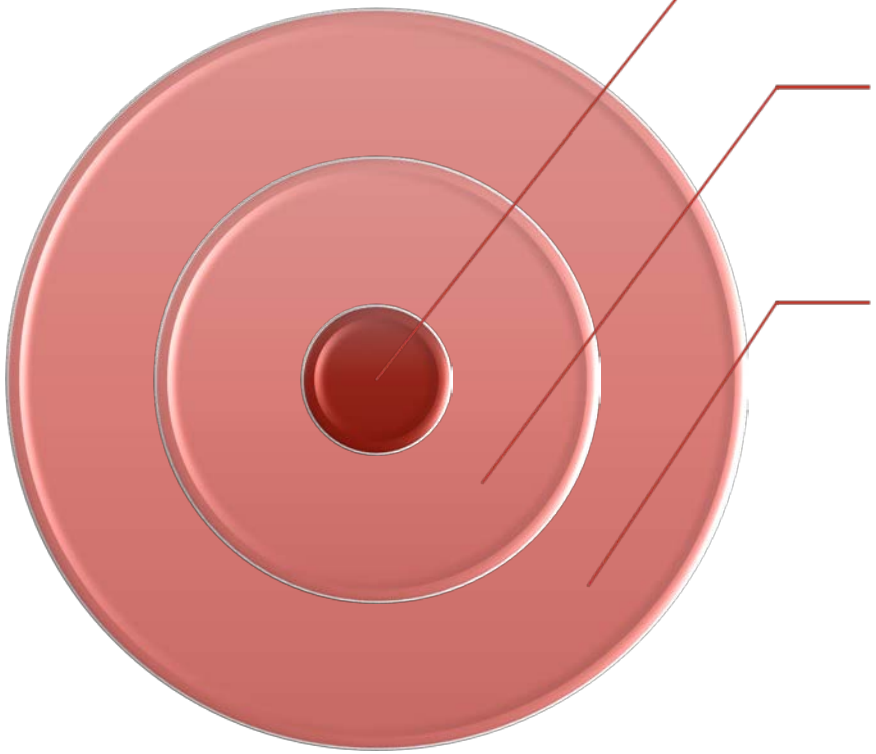
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|
| **Def.** | **Goals** | **Artifacts** | **Risk** | **Technique** | **Elicit** | **Categorize** | *Prioritize* | *Inspect* |

## Categorize Requirements

- Classify requirements into essential, non-essential, system, software, or architectural constraints.
- Sample table:

| | System level | Software level | Architectural constraint |
|---|---|---|---|
| Reqt. 1 | | | |
| Reqt. 2 | | | |

# Step 7 – Categorize Requirements Examples



**Software Level:** Users cannot exceed their access privileges.

**System Level:** The system is required to have strong authentication measures in place at all system gateways/entrance points.

**Architectural Constraints:** The system should be able to support the capabilities of a distributed network.

# Step 8

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|
| **Def.** | **Goals** | **Artifacts** | **Risk** | **Technique** | **Elicit** | **Categorize** | **Prioritize** | *Inspect* |

## Prioritize Requirements

- Use risk assessment and categorization results to prioritize requirements.
- Prioritization techniques: Triage, Win-Win, Analytical Hierarchy Process
- Requirements engineering team should produce a cost-benefit analysis to aid stakeholders.

# Step 9

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|
| Def. | Goals | Artifacts | Risk | Technique | Elicit | Categorize | Prioritize | Inspect |

## Requirements Inspection

- Inspection aids in creating accurate and verifiable security requirements.
- Look for ambiguities, inconsistencies, mistaken assumptions.
- Fagan inspections / peer reviews
- Exit criteria: all requirements verified and documented

# Approach

The SQUARE process

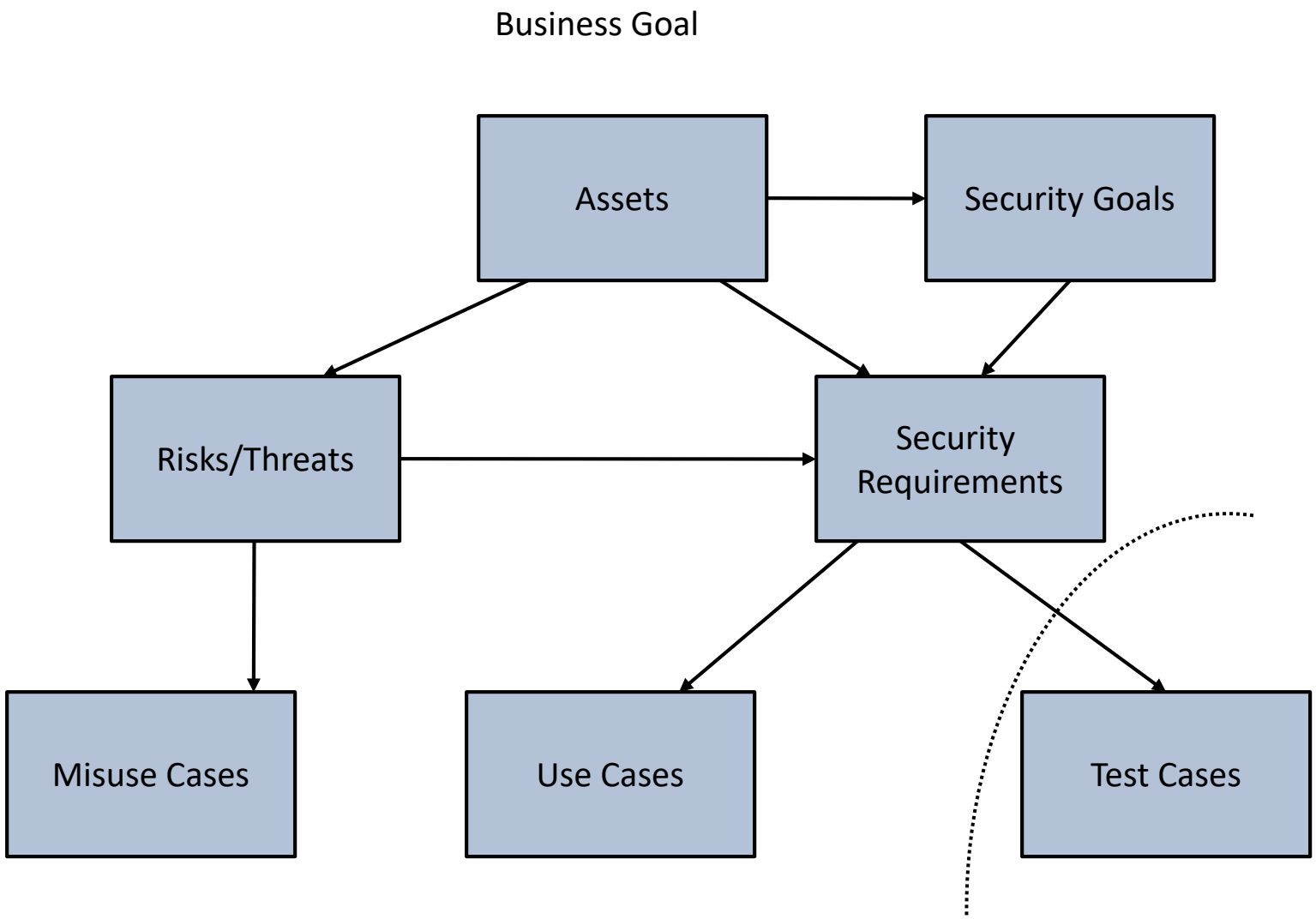
- takes about three months calendar time to complete
- has been implemented in several case studies

SQUARE-Lite

- Agree on definitions.
- Identify assets and security goals.
- Perform risk assessment
- Elicit security requirements.
- Prioritize requirements.

SQUARE-Lite has been implemented in one case study.

# Traceability in the SQUARE Tool



# Summary

# Summary

SQUARE – Security Quality Requirements Engineering

Nine steps:

(1) agree on definitions
(2) identify assets and security goals
(3) develop artifacts
(4) assess risks
(5) select elicitation technique(s)
(6) elicit security requirements
(7) categorize requirements
(8) prioritize requirements
(9) inspect requirements

SQUARE-Lite, P-SQUARE, A-SQUARE

# Additional Resources

R. Anderson – Home Page

http://act-r.psy.cmu.edu/people/ja/

Mary Shaw – Research Activities

http://spoke.compose.cs.cmu.edu/shaweb/r/research.htm

# Additional Resources

BSI content on requirements engineering

https://buildsecurityin.us-cert.gov/articles/best-practices/requirements-engineering

SQUARE Technical Report – SEI web site

www.sei.cmu.edu/pub/documents/05.reports/pdf/05tr009.pdf

SQUARE Case Study Reports – SEI web site

"Integrating Security and Software Engineering"
IDEA Group Publishing

www.idea-group.com

SQUARE-Lite

http://resources.sei.cmu.edu/asset_files/SpecialReport/2008_003_001_14912.pdf

# SQUARE Demo Video

http://resources.sei.cmu.edu/library/asset-view.cfm?assetID=73347

# Questions?


# Module 14: Artifacts to Support Cybersecurity Requirements

Introduction to Assured Software Engineering

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

# Notices

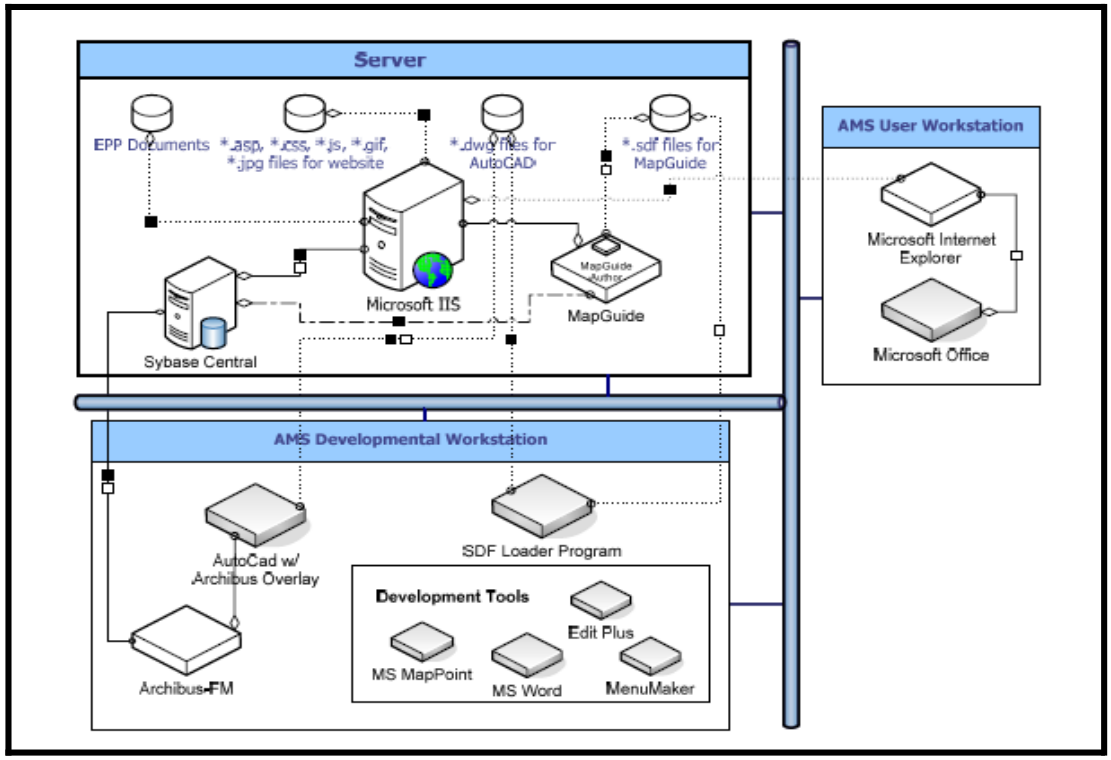


This material is based upon work funded and supported by the Independent Agency under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center sponsored by the United States Department of Defense.

# Develop Artifacts (corresponds to SQUARE Step 3)

Types of artifacts to collect

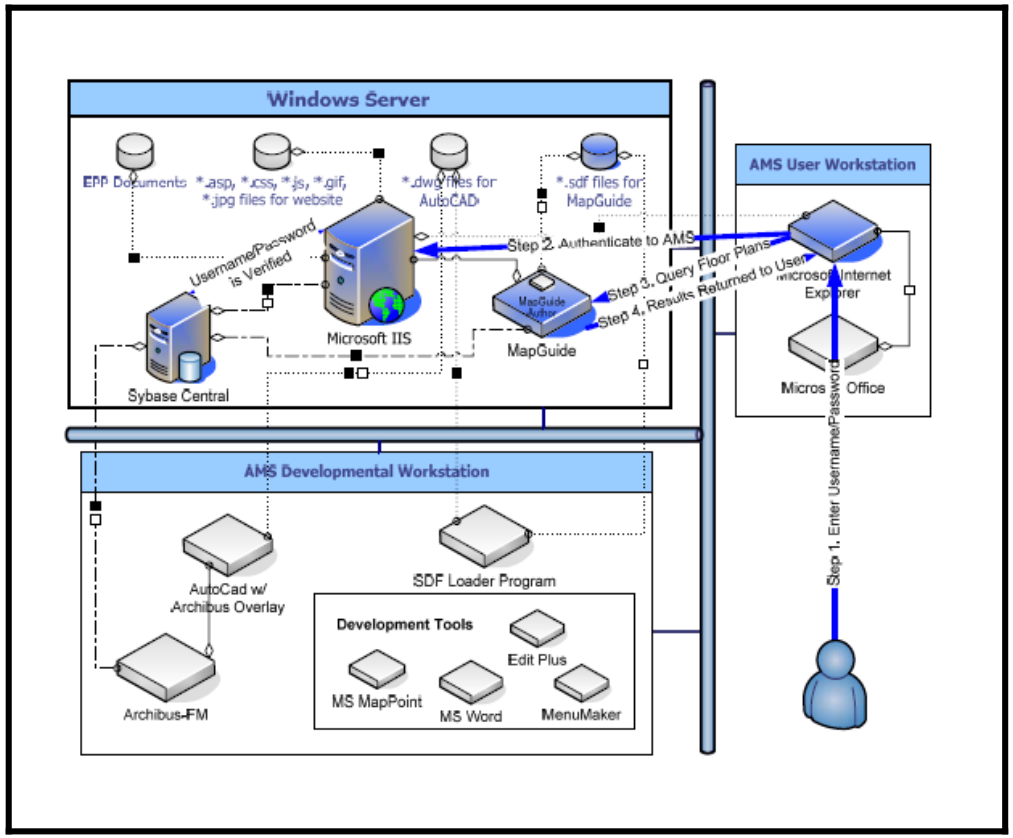
- System architecture diagrams (should exist for the project)



# Develop Artifacts -1

Types of artifacts to collect

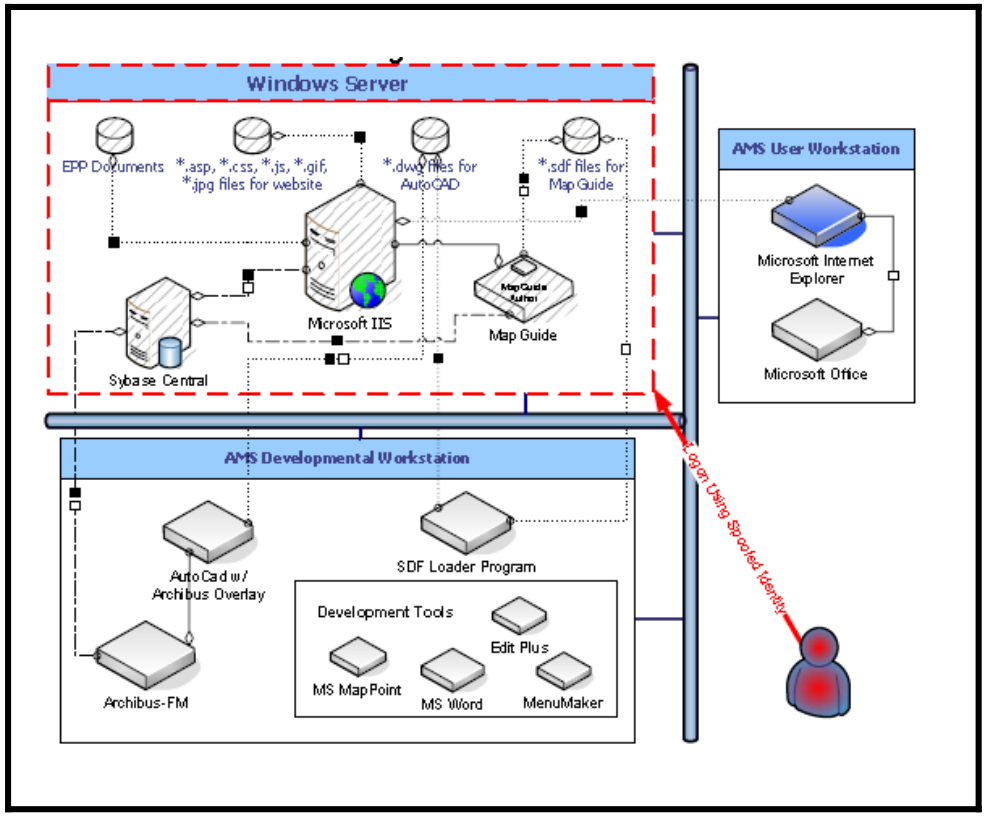
- Use case scenarios/ diagrams (should exist for the project)



# Develop Artifacts -2

Types of artifacts to collect

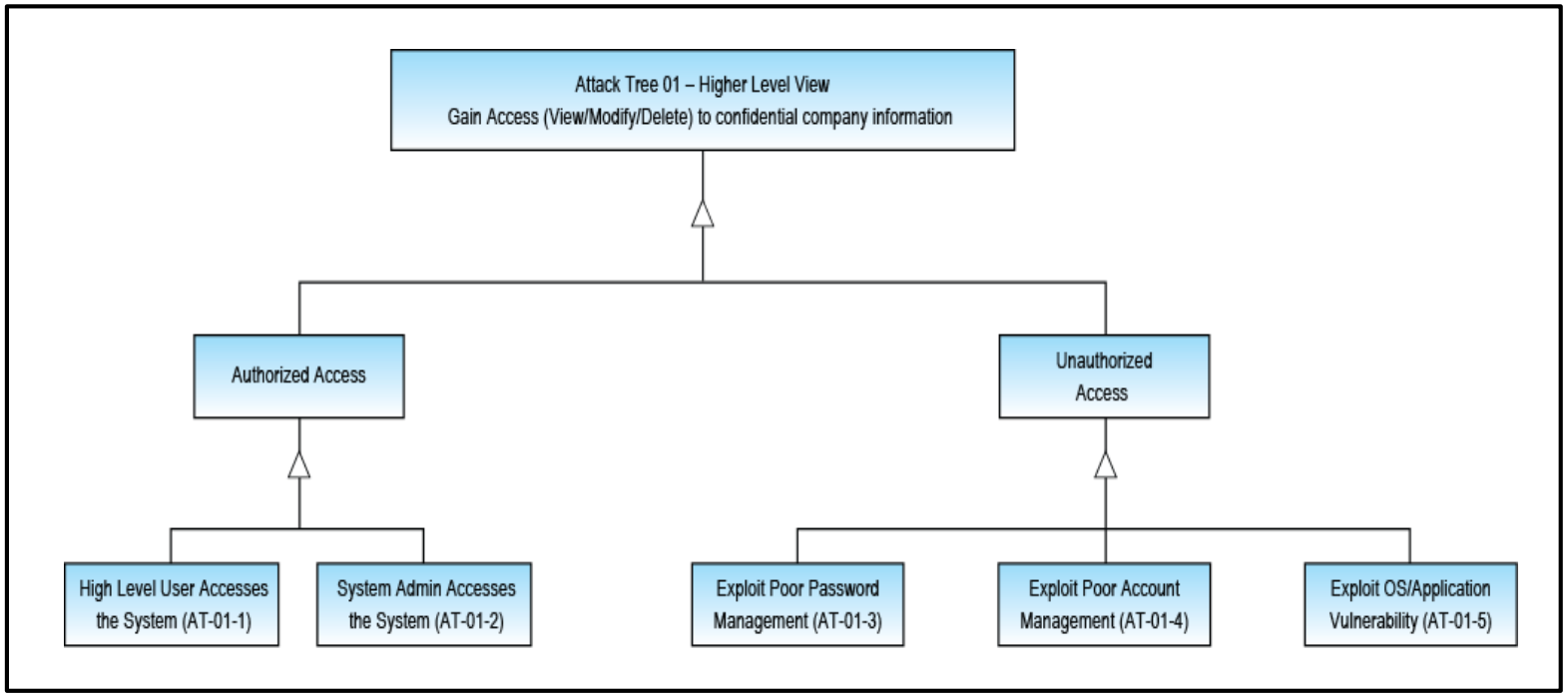
- Misuse case scenarios/ diagrams (exemplar misuse cases)



# Develop Artifacts -3

Types of artifacts to collect

- Attack trees



# Questions?


# Module 15: SQUARE for Acquisition

Introduction to Assured Software Engineering

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

# Notices



This material is based upon work funded and supported by the Independent Agency under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center sponsored by the United States Department of Defense.

# Outline

Background

Introduction to A-SQUARE

Three Cases for Square for Acquisition (A-SQUARE)

Summary and further work

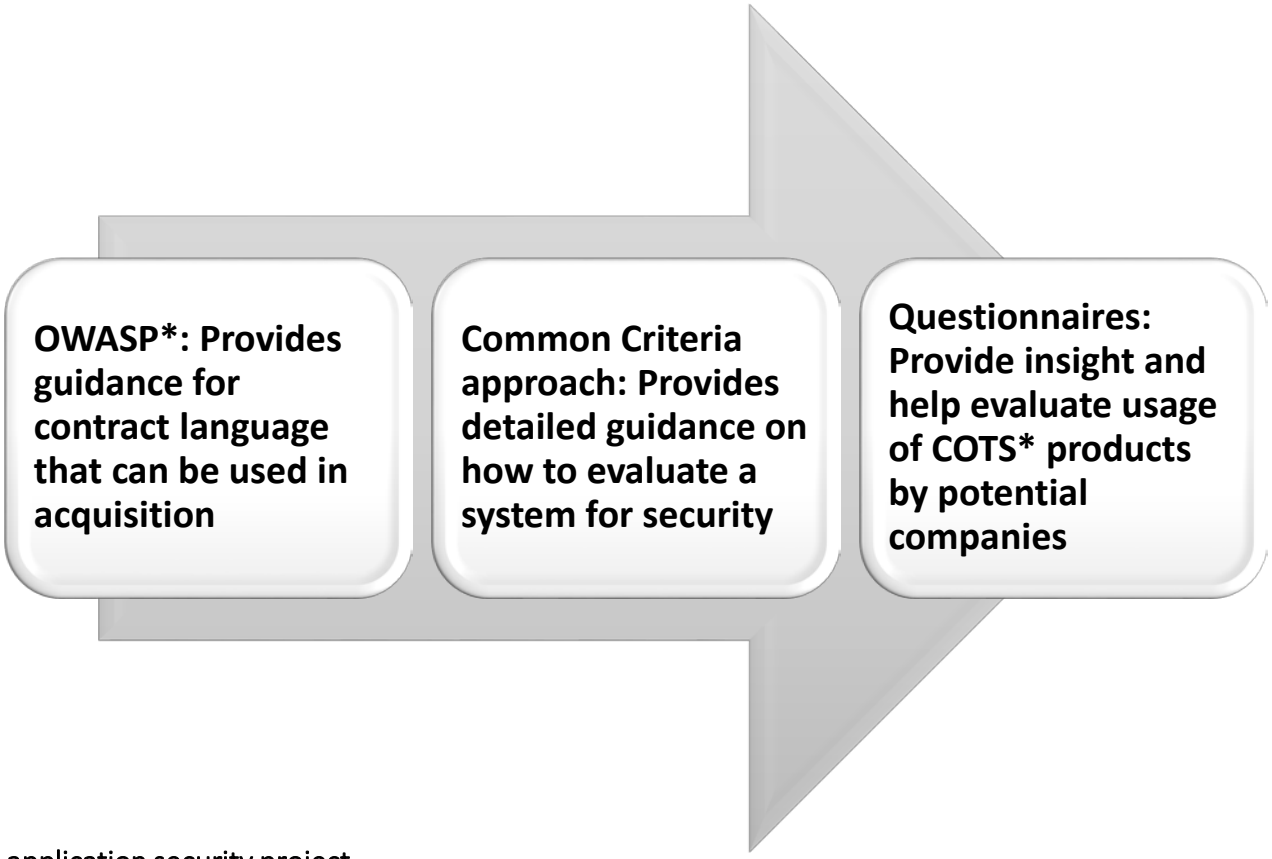
Class Exercise

Questions

# Background

# Background

Current efforts in the field of software acquisition




OWASP – open web application security project
**COTS – commercial off the shelf**

# What Is Acquisition?

**Acquisition:** The process of obtaining a system, software product, or software service. Software products may include commercial-off-the-shelf (COTS) products, modified-off-the-shelf (MOTS) products, open source products, or fully developed products.

The above definition was derived from these references:

Software & Systems Engineering Standards Committee, IEEE Computer Society. ISO/IEC 12207, IEEE Std. 12207-2008, Systems and Software Engineering - Software Life Cycle Processes, Second Edition. IEEE Computer Society, 2008.

Software & Systems Engineering Standards Committee, IEEE Computer Society. IEEE Std. 1062, IEEE Recommended Practice for Software Acquisition. IEEE Computer Society, 1998.

# The Need for SQUARE

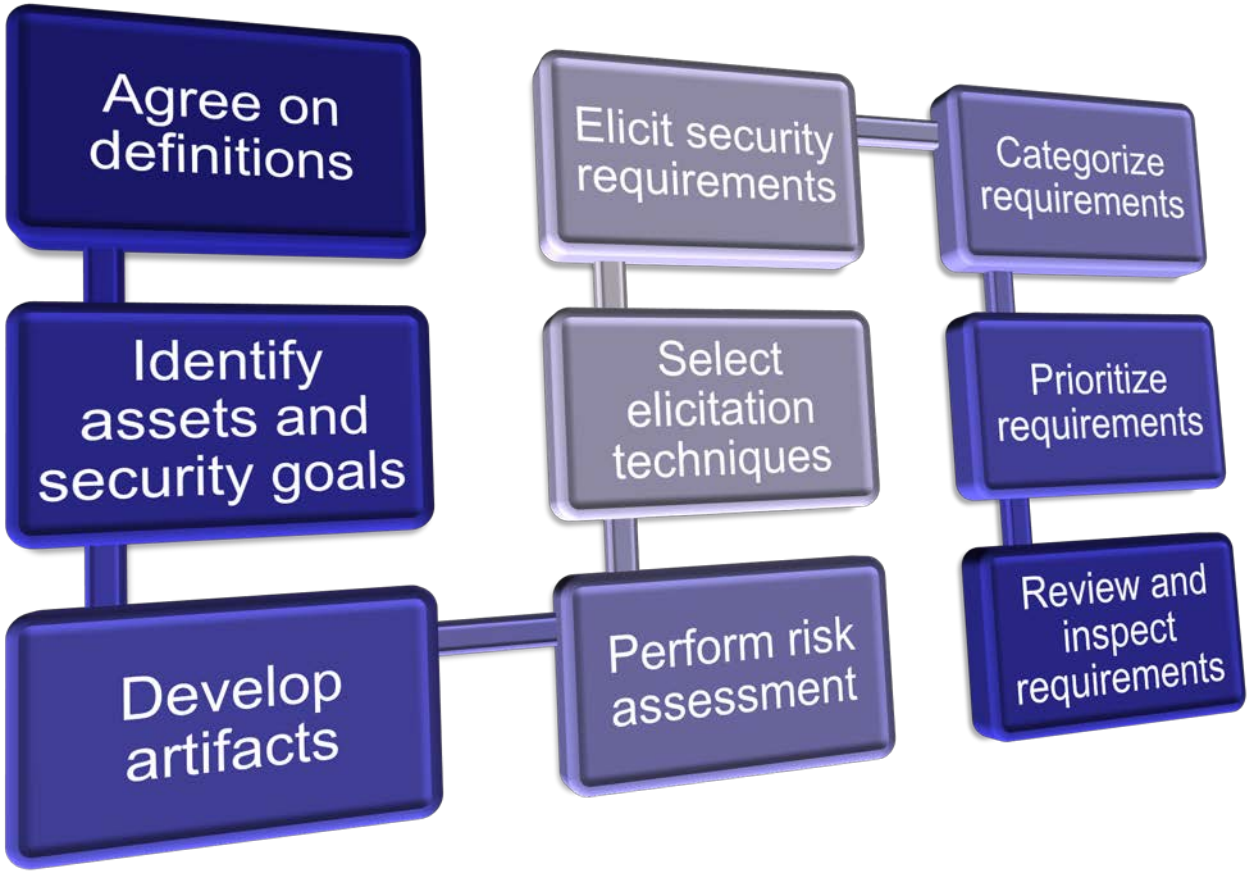
Current problems:

- Lack of control on security requirements of the product by the acquiring company
- Current work lacks level of detail needed, which is specific to security requirements

Benefits of adapting SQUARE for Acquisition:

- Can be easily tailored and modified for various acquisition scenarios
- Well-defined process framework with clear roles and responsibilities defined for each of the stakeholders
- A-SQUARE helps address security requirements early in the project

# Recap of the SQUARE Process



# Introduction to A-SQUARE

# A-SQUARE: Three Cases

Case 1 – acquisition organization has typical client role for new software

Acquisition Org. Contractor Requirements

Case 2 – acquisition organization does requirements specification

Acquisition Org. Requirements Contractor

Case 3 – acquisition organization is purchasing COTS software

Acquisition Org. COTS

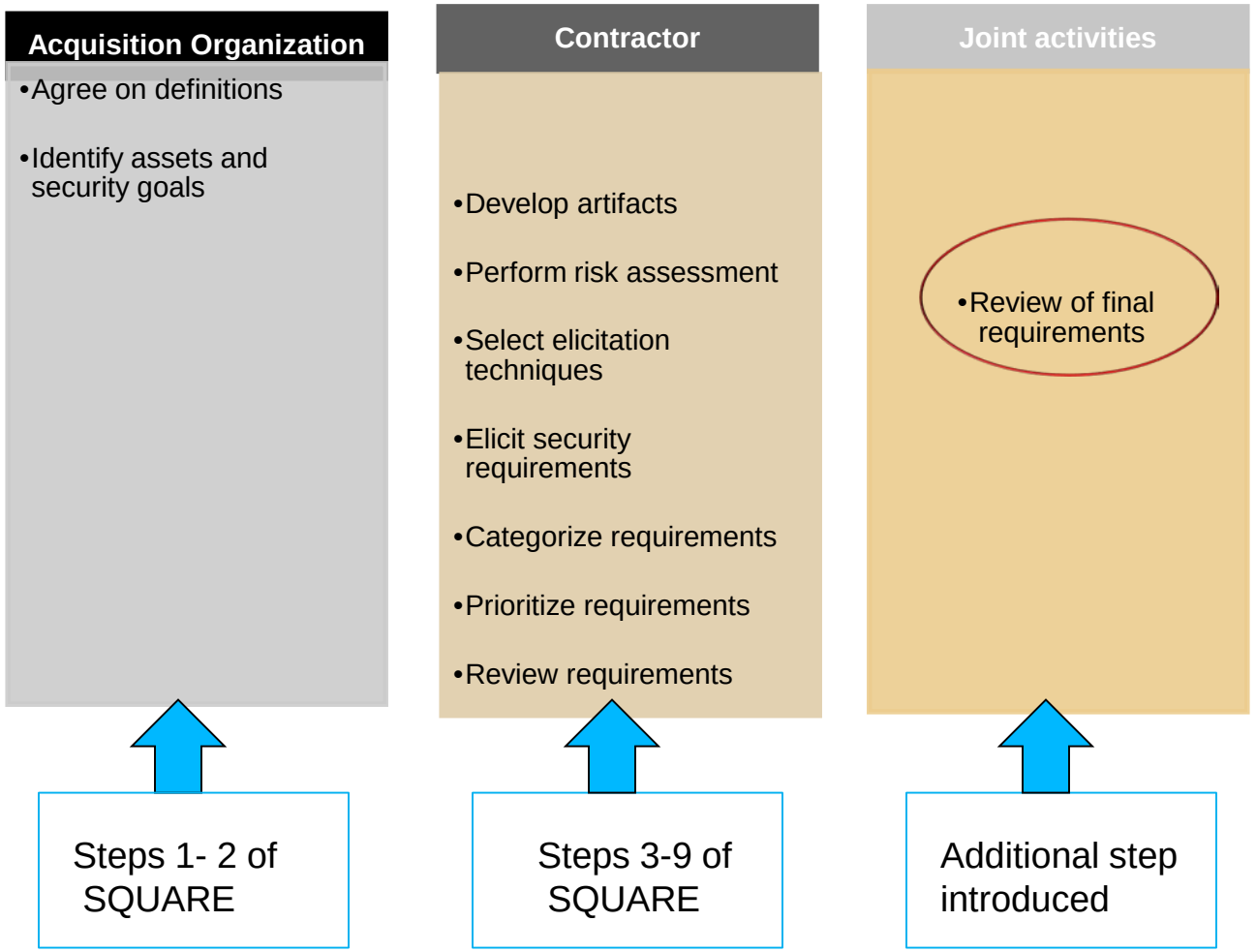
# Case 1

# A-SQUARE: Case 1 Introduction

Nature of software acquisition:

- contractor is responsible for the requirements definition
- contractor should be on board and the contract is awarded
- acquisition organization plays a typical client role

# Case 1: Process Workflow



# Case 1: Important Points

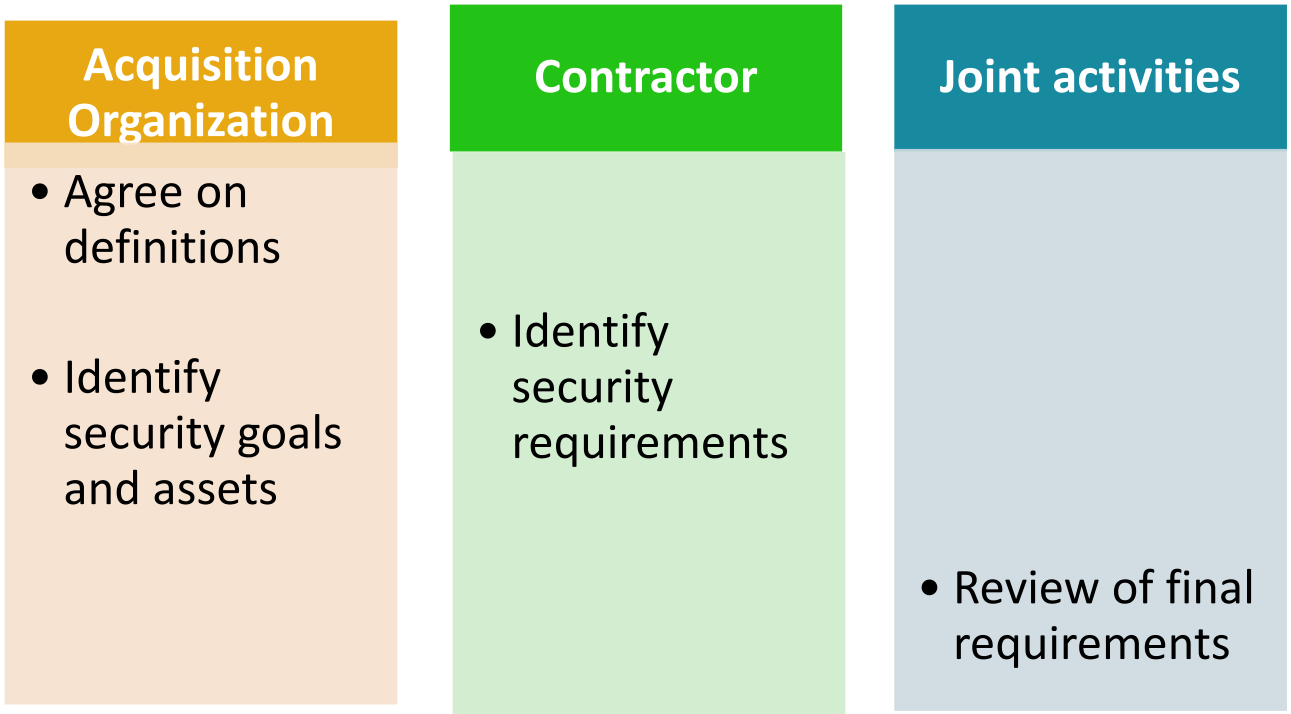
The client has no formal role in requirements elicitation for the project.

The contractor uses SQUARE as the driving process framework for identifying security requirements.

The additional step (as shown in workflow) may not be needed if both the parties work together.

# Case 1: Compressed Workflow

In the event that the client is unaware of the requirements engineering process, the resultant workflow is compressed as shown here.



# Case 2

# A-SQUARE: Case 2 Introduction

Nature of software acquisition:

- acquisition organization specifies requirements as part of request for proposal (RFP)
- original SQUARE should be used by the contractor
- requirements specified will have relatively high-level security requirements

# Case 2: Important Points

The process workflow is similar to the nine-step SQUARE process.

Level of detail in the requirements definition is crucial.

- Too much detail can constrain the contractor.
- The contractor needs some flexibility in defining the requirements.
- The exit criteria for this process is the final review and approval of the requirements by both parties.

# Case 3

# A-SQUARE: Case 3 Introduction

Nature of software acquisition

- acquisition of COTS products

What is COTS?

- computer software products that are ready-made and available for use
- serve as good alternatives for in-house developments

Benefits of using COTS

- applications can be built “out-of-the-box”
- improves overall productivity and reduces company costs

# A-SQUARE: Case 3 Introduction

Examples of well-known COTS applications acquired by organizations

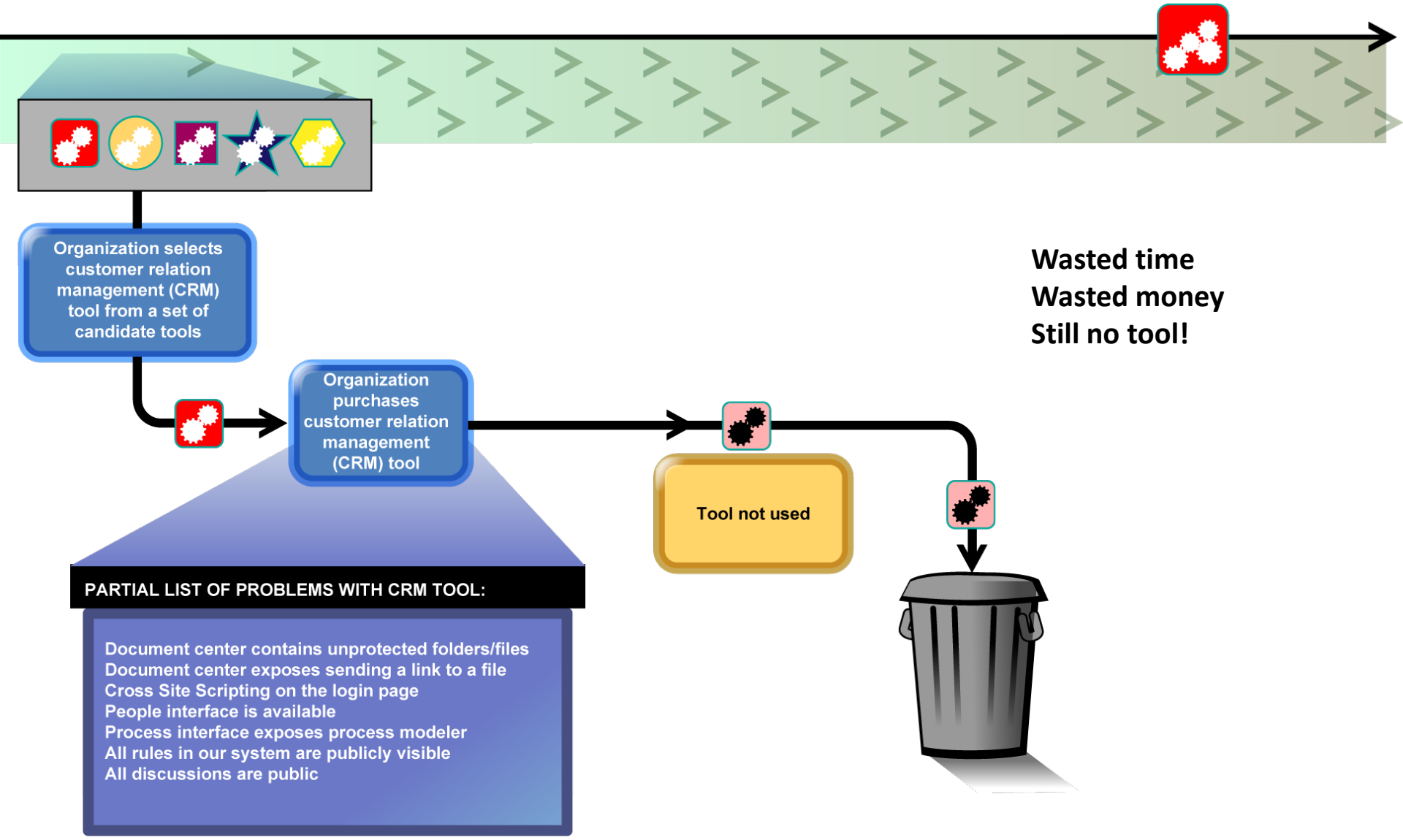
Spreadsheets

Databases

Document management Systems

Emails

# Is There Really a COTS Security Problem?



# A-SQUARE Case 3 – Steps 1-4

## Process for acquiring COTS software

| | Step | Input | Techniques | Participants | Output |
|---|---|---|---|---|---|
| 1 | **Agree on definitions** | Candidate definitions from IEEE and other standards | Structured interviews, focus group | Acquisition organization – stakeholders, security specialists | Agreed-to definitions |
| 2 | **Identify assets and security goals** | Definitions, candidate goals, business drivers, policies and procedures, examples | Facilitated work session, surveys, interviews | Acquisition organization – stakeholders, security specialists | Assets and goals |
| 3 | **Identify preliminary security requirements** | Assets and goals | Work session | Acquisition organization – security specialists | Preliminary security requirements |
| 4 | **Review COTS software package information and specifications** | Assets, goals, preliminary security requirements | Study security features of various packages and documents them, in a spreadsheet, for example | Acquisition organization – security specialists, COTS vendors | Spreadsheet of security features of various packages |

# A-SQUARE Case 3 – Steps 5-7

## Process for acquiring COTS software

| | Step | Input | Techniques | Participants | Output |
|---|---|---|---|---|---|
| 5 | **Finalize security requirements** | Preliminary security requirements, features of various packages | Work session – use the spreadsheet to refine and modify the preliminary security requirements to arrive at a final set | Acquisition organization – security specialists | Final security requirements |
| 6 | **Perform tradeoff analysis** | Final security requirements, spreadsheet of security features | Tradeoff analysis of COTS products relative to final security requirements | Acquisition organization – stakeholders, security specialists | Prioritized list of COTS products relative to security requirements |
| 7 | **Final product selection** | Prioritized list of COTS products relative to security, other important COTS product features | Tradeoff analysis | Acquisition organization – stakeholders | Final COTS product selection |

# Case 3: Important Points

## Prioritization

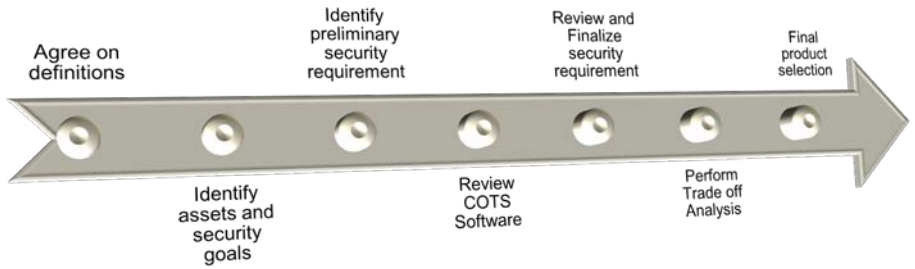
- Security requirements need to be prioritized together with other requirements when acquiring COTS software.

## Tradeoff

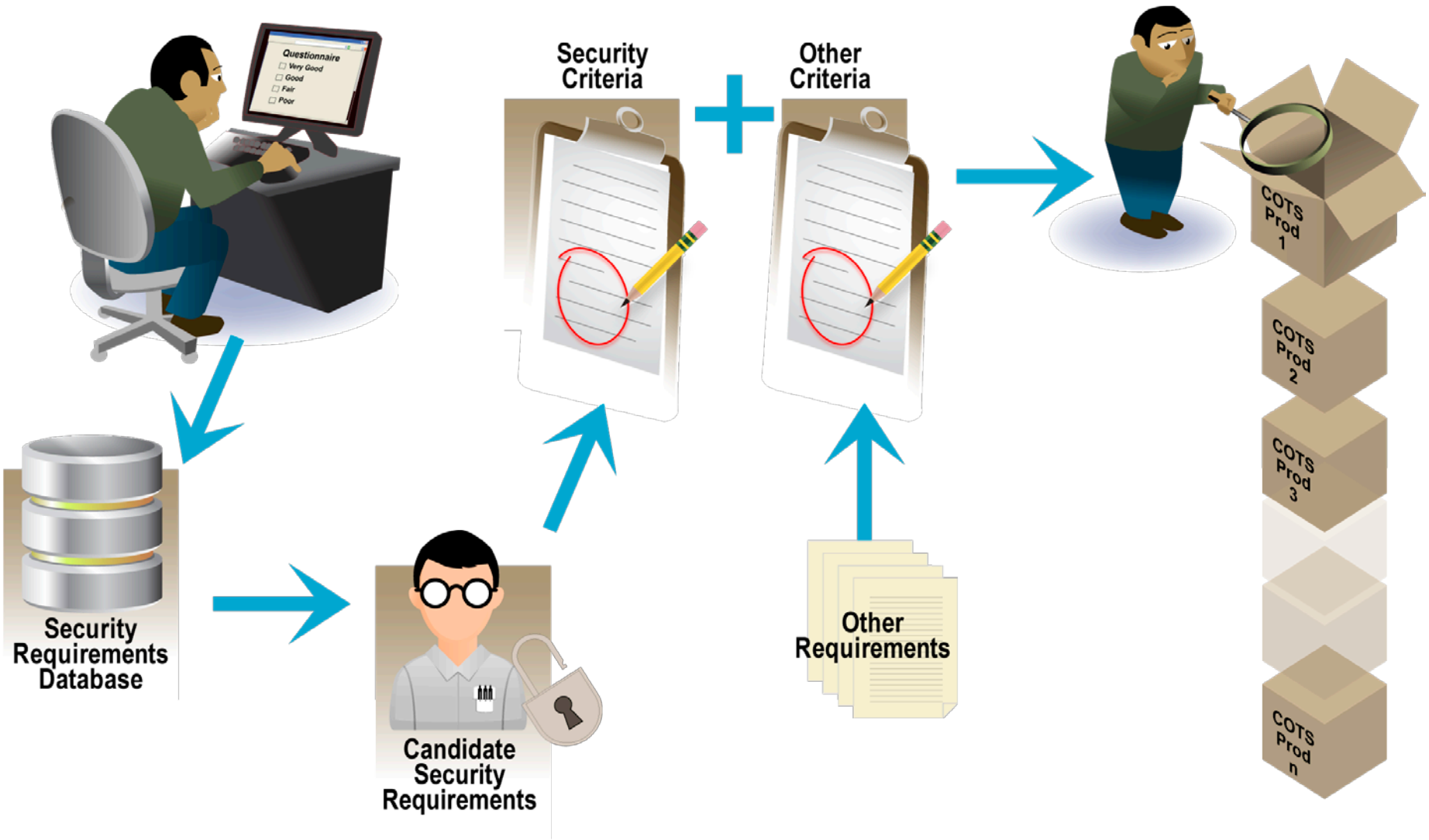
- Tradeoffs and compromises might have to be made since the software might not meet all the security goals of the organization.

## Review

- Reviewing the requirements may help the acquiring organization to identify important security requirements.



# Future Vision – A New Scenario



# Discussion

Do you focus more on features or quality factors when you acquire a COTS product?

Do you consider security when you buy a COTS product? For yourself? Your organization?

Would you change anything in the future?

# Summary

# Summary and Further Work

A-SQUARE helps identify security requirements early into the project.

It can reduce the risk associated with software acquisition.

Prototype A-SQUARE tool developed by CMU MSIT Team – robust tools are needed

Application of A-SQUARE on projects would help:

- support acquisition organizations
- validate the practices of A-SQUARE
- understand the tailoring needed

# Additional Resources

Allen, Julia H., Barnum, Sean, Ellison, Robert J., McGraw, Gary, & Mead, Nancy R. Software Security Engineering: A Guide for Project Managers. Addison Wesley Professional, 2008. (Available from Amazon.com.)

U.S. Department of Homeland Security. Build Security In: Requirements Engineering. https://buildsecurityin.us-cert.gov/daisy/adm-bsi/articles/best-practices/requirements.html

IDEA Group Publishing. http://www.idea-group.com

Mead, Nancy R., Hough, Eric, & Stehney II, Ed. Security Quality Requirements Engineering (CMU/SEI-2005-TR-009). Software Engineering Institute, Carnegie Mellon University, 2005. http://www.sei.cmu.edu/library/abstracts/reports/05tr009.cfm

Mead, Nancy R. “Identifying Security Requirements Using the Security Quality Requirements Engineering (SQUARE) Method” Integrating Security and Software Engineering: Advances and Future Visions. Edited by H. Mouratidis and P. Giorgini. Idea Group, pp. 44-69, 2006 (ISBN: 1-59904-147-2).

# Additional Resources

SQUARE case study reports:

- Gayash, Ashwin, Viswanathan, Venkatesh, & Padmanabhan Deepa. Advisor: Nancy R. Mead. SQUARE-Lite: Case Study on VADSoft Project (CMU/SEI-2008-SR-017). Software Engineering Institute, Carnegie Mellon University, 2008. http://www.sei.cmu.edu/library/abstracts/reports/08sr017.cfm
- Hough, Eric, Ojoko-Adams, Don, Chung, Lydia, & Hung, Frank. Security Quality Requirements Engineering (SQUARE): Case Study Phase III (CMU/SEI-2006-SR-003). Software Engineering Institute, Carnegie Mellon University, 2006. http://www.sei.cmu.edu/library/abstracts/reports/06sr003.cfm
- Panusuwan, Varokas & Batlagundu Prashanth. Faculty Advisor: Nancy Mead. Privacy Risk Assessment Case Studies in Support of SQUARE (CMU/SEI-2009-SR-017). Software Engineering Institute, Carnegie Mellon University, 2009. http://www.sei.cmu.edu/library/abstracts/reports/09sr017.cfm

# Questions?


# Module 16: Risk Analysis for Software Assurance (Part 1)

(Developed by Christopher Alberts, SEI)

Introduction to Assured Software Engineering

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

# Notices



This material is based upon work funded and supported by the Independent Agency under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center sponsored by the United States Department of Defense.

# Topics

Risk Concepts

Two Approaches for Analyzing Risk

Security Engineering Risk Analysis (SERA) Concepts

Summary

# Risk Concepts

# Software Assurance[1]

Application of technologies and processes to achieve a required level of confidence that software systems and services

- Function in the intended manner
- Are free from accidental or intentional vulnerabilities
- Provide security capabilities appropriate to the threat environment
- Recover from intrusions and failures

We will examine risk management in a software assurance context.

[1] SEI Software Assurance Curriculum Project. Software Assurance Curriculum Project Volume I: Master of Software Assurance Reference Curriculum (CMU/SEI-2010-TR-005). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2006. http://www.sei.cmu.edu/reports/10tr005.pdf

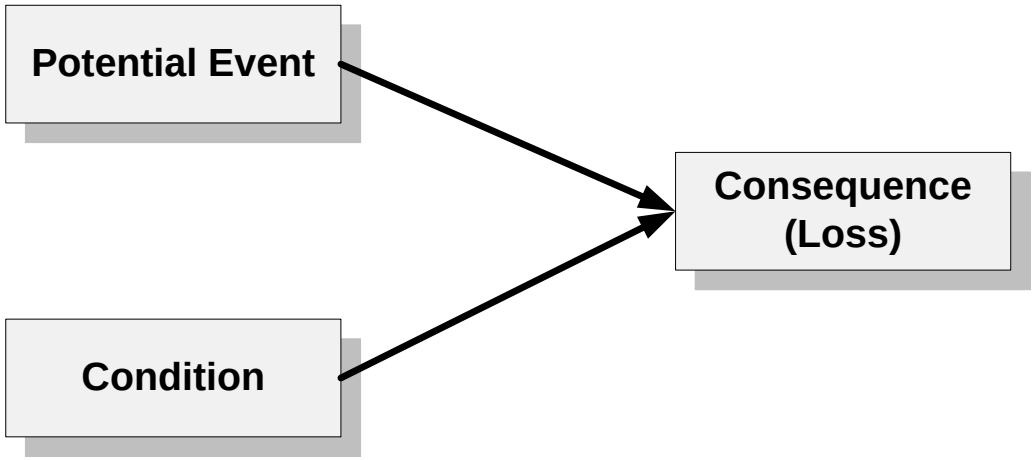
# What Is Risk?

The probability of suffering harm or loss

A measure of the likelihood that an event will lead to a loss coupled with the magnitude of the loss

Risk requires the following conditions:[1]

- A potential loss
- Likelihood
- Choice




1. Charette, Robert N. *Application Strategies for Risk Analysis*. New York, NY: McGraw-Hill Book Company, 1990.

# Risk Management Activities

Assess risk

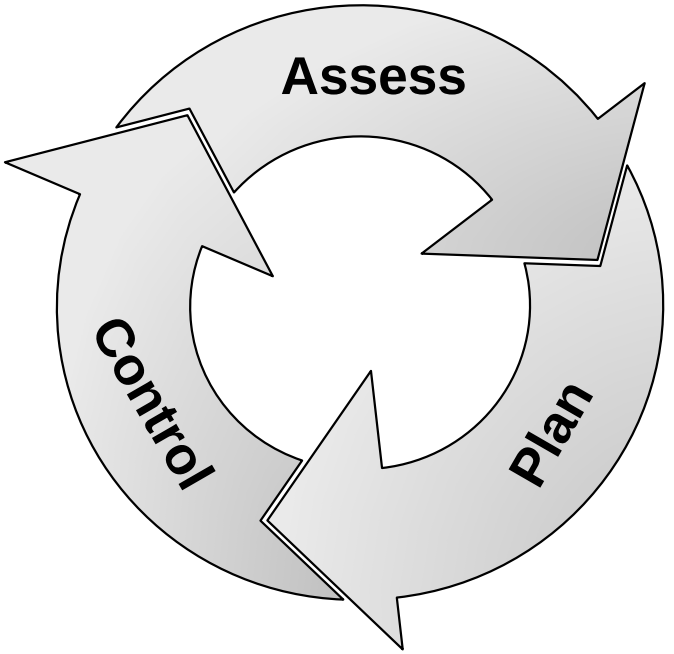
- Transform the concerns people have into distinct, tangible risks that are explicitly documented and analyzed

Plan for risk control

- Determine an approach for addressing each risk; produce a plan for implementing the approach

Control risk

- Deal with each risk by implementing its defined control plan and tracking the plan to completion



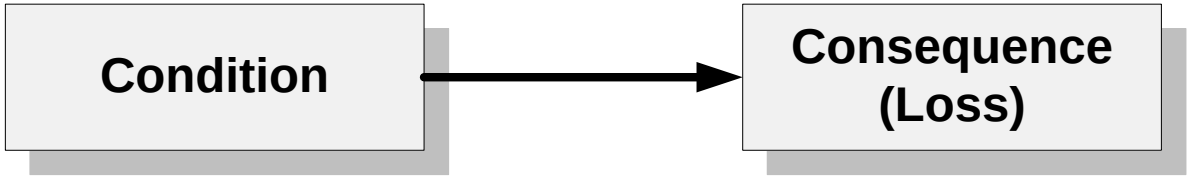
# Issue/Problem

A condition that directly produces a loss or adverse consequence

- No uncertainty exists.
- The condition exists and is having a negative effect on performance.

Issues can also lead to (or contribute to) other risks by

- Creating a circumstance that enables an event to trigger additional loss
- Making an existing event more likely to occur
- Aggravating the consequences of existing risks



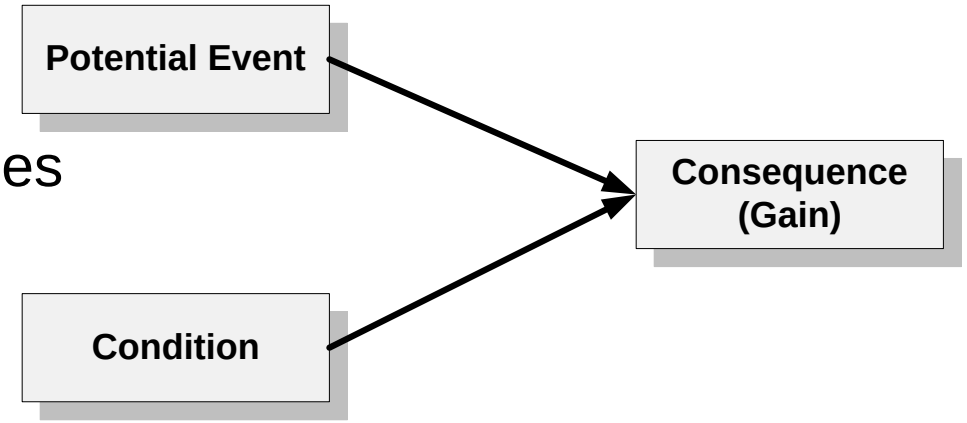
# Opportunity

The probability of realizing a gain

- Defines a set of circumstances that provides the potential for a desired gain
- Enables an entity to improve its current situation relative to the status quo
- Can require an investment or action to realize that gain (i.e., to take advantage of the opportunity)

Pursuit of an opportunity can

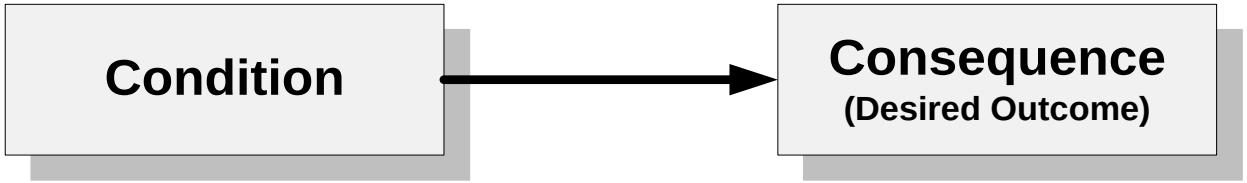
- Produce new risks or issues
- Change existing risks or issues



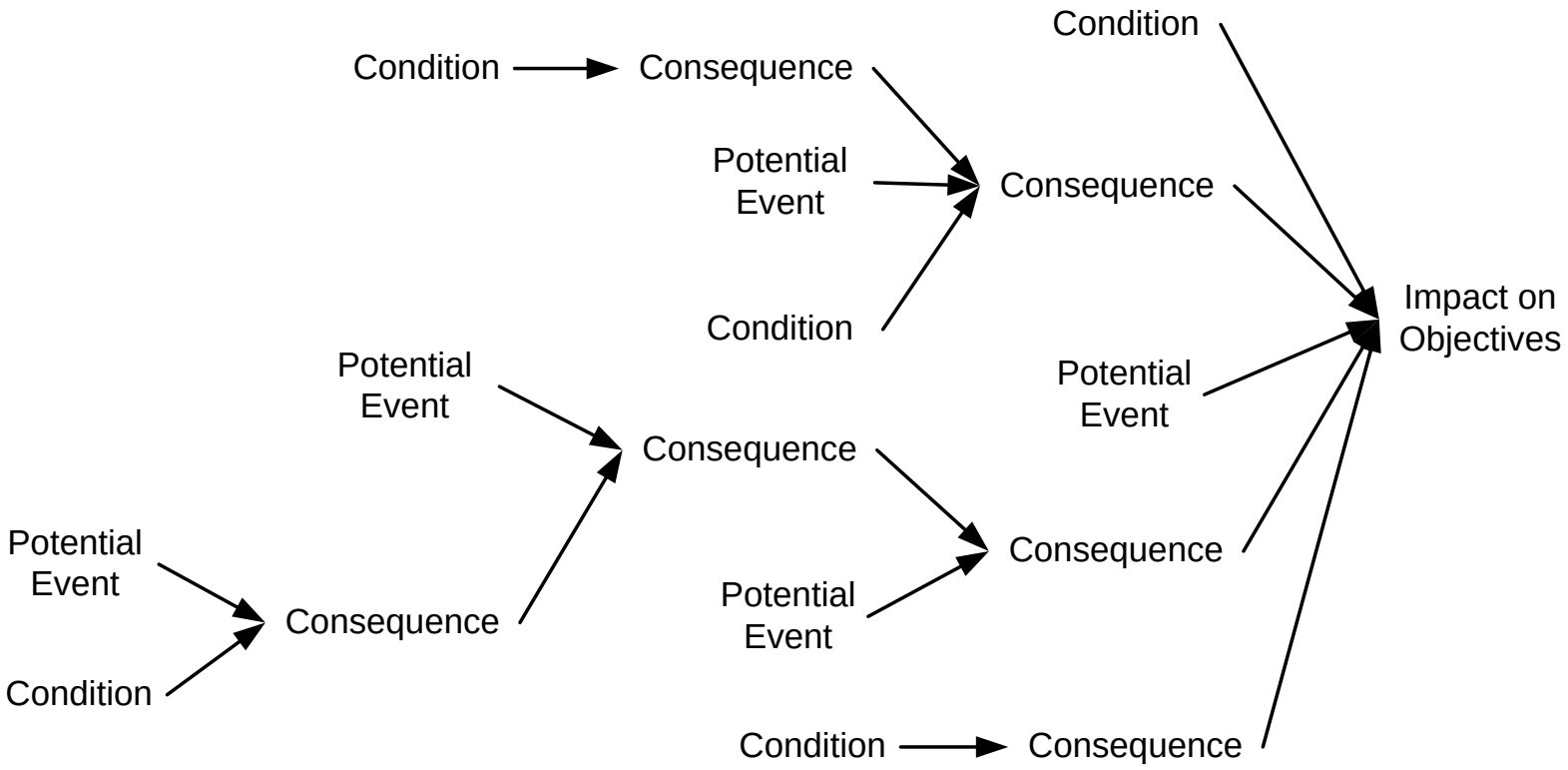
# Strength

A condition that is driving an entity (e.g., project, system) toward a desired outcome

- No uncertainty exists
- The condition exists and is having a positive effect on performance (i.e., driving an entity toward a desired outcome)



# Causal Chain of Conditions and Events




Risks, issues/problems, opportunities, and strengths are part of an interrelated causal chain of conditions and events that must be managed.

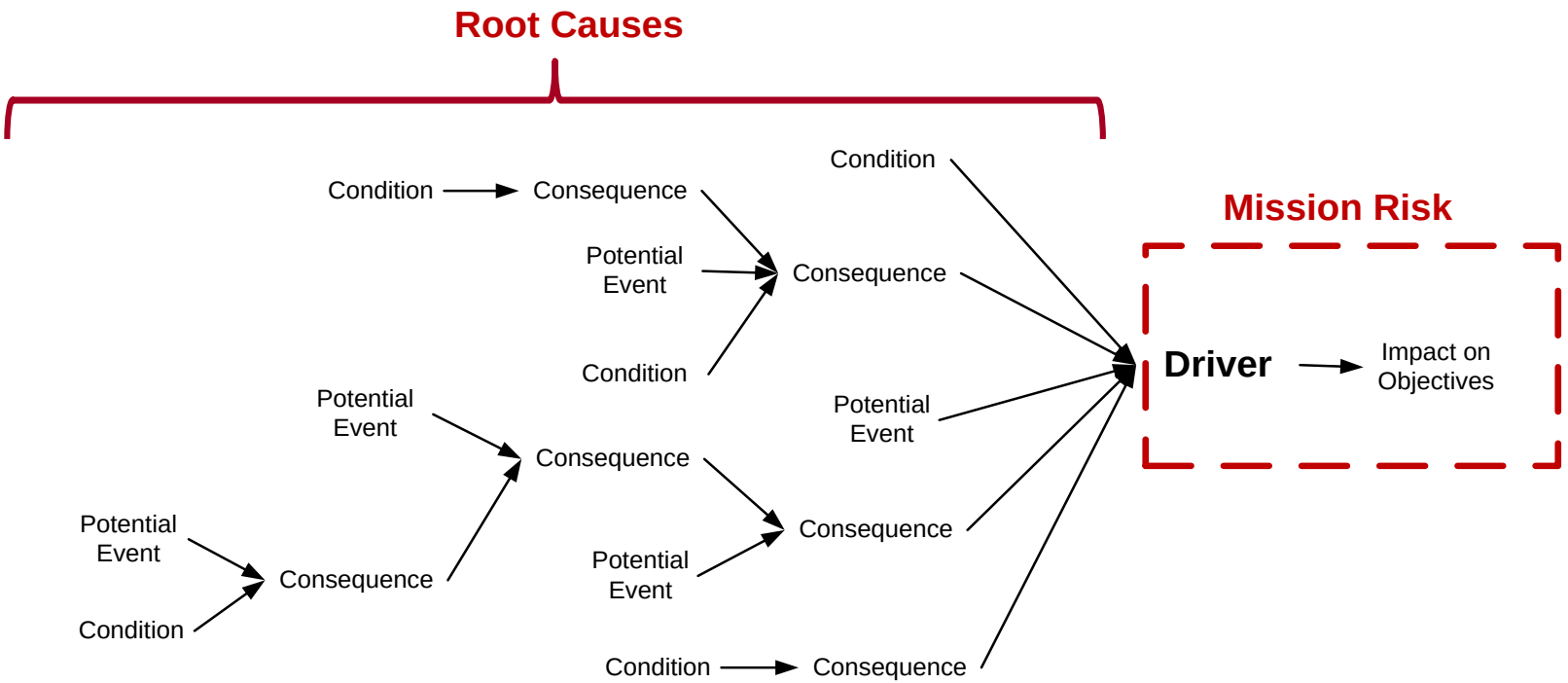
# Two Approaches for Analyzing Risk

# Two Type of Risk Analysis

Two distinct risk analysis approaches can be used when evaluating systems:

1. Mission risk analysis
2. Event risk analysis

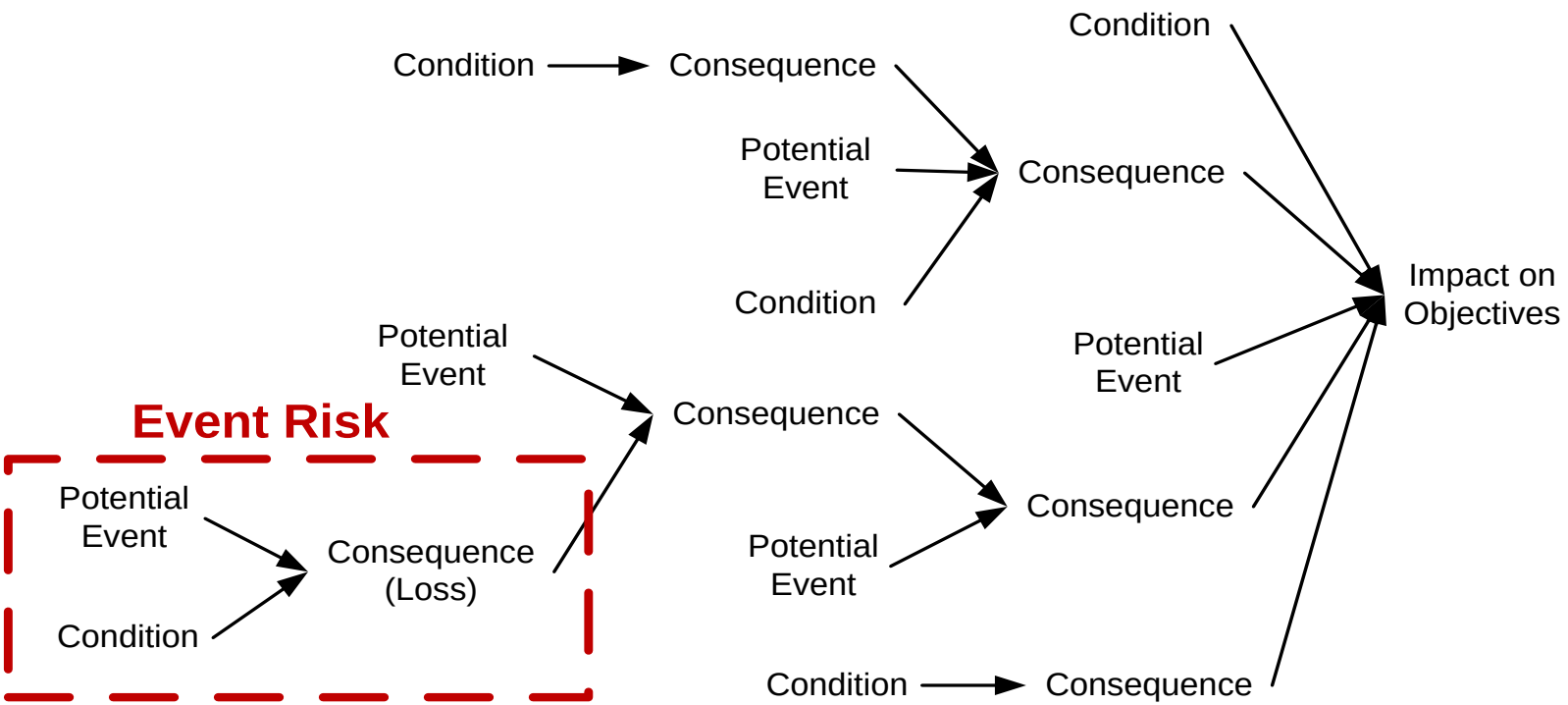
# Elements of Mission Risk




Mission risk is the probability of mission failure (i.e., not achieving key objectives).

Mission risk aggregates the effects of multiple conditions and events on a system's ability to achieve its mission.

# Elements of Event Risk




Event risk is the probability that an event will lead to a negative consequence or loss.

# Security Engineering Risk Analysis (SERA) Concepts

# Current State: High Residual Security Risk

Security in the acquisition and development of software-reliant systems:

- Focus on meeting functional requirements
- Defer security to later lifecycle activities

Security features

- Addressed during system operation and sustainment
- Typically not engineered into a system

Software-reliant systems are typically deployed with significant residual security risk.

- High residual security risk puts operational missions at risk.

# Goal: Reduce Residual Security Risk

Three main causes of operational security vulnerabilities:

- Design weaknesses
- Implementation/coding vulnerabilities
- System configuration errors

Design vulnerabilities are not easily addressed during operations.

Early detection and remediation of design vulnerabilities will reduce residual security risk during operations.

# Complex Nature of Security Risk

Performing risk analysis early in lifecycle does not guarantee less risk during operations.

Traditional security risk analyses cannot address complexity of security attacks.

- Traditional Analysis
  - Single threat actor exploits single vulnerability in single system to cause an adverse consequence
- Current Reality
  - Multiple actors exploit multiple vulnerabilities in multiple systems as part of a complex chain of events.

Traditional methods can be ineffective at analyzing complex security attacks.

# Security Engineering Risk Analysis (SERA)

Assesses operational security risks early in the software lifecycle

- Requirements
- Architecture
- Design

Employs structured, systematic risk analysis to handle the complex nature of security risk

Goal:

- To identify and address design weaknesses early in the lifecycle (i.e., build security in)

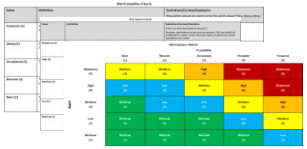
# SERA Method

1. **Establish operational context.**
2. **Identify risk.**
3. **Analyze risk.**
4. **Determine control approach.**
5. **Develop control plan.**

**Mission Thread Worksheet**

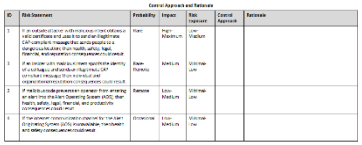
**Risk Identification Worksheet**

**Risk Evaluation Criteria**



**Risk Analysis Worksheet**

**Control Approach Worksheet**



**Control Plan Worksheet**

# Establish Operational Context (Task 1)

Target of the analysis (e.g., the software application or system that is being assessed) is determined initially.

The operational environment for the target is characterized to establish baseline operational performance.

Security risks are analyzed in relation to this baseline.

Sub-tasks:

- Set scope of risk analysis.
- Define workflow/mission thread.

# Task 1 Questions: Set Scope of Risk Analysis

What technology/system is the focus of the analysis?

Which workflows or mission threads does the target support?

Which workflow or mission thread will be included in the security risk analysis?

# Task 1 Questions: Define Workflow/Mission Thread

What are the mission and objective(s) of the workflow/mission thread?

What steps are required to complete the workflow/mission thread?

- Who or what (e.g., person, technology) performs each step in the workflow/mission thread?
- What technologies (e.g., systems, applications, software, hardware) support each step in the workflow/mission thread?

How does the target of the analysis support the workflow/mission thread?

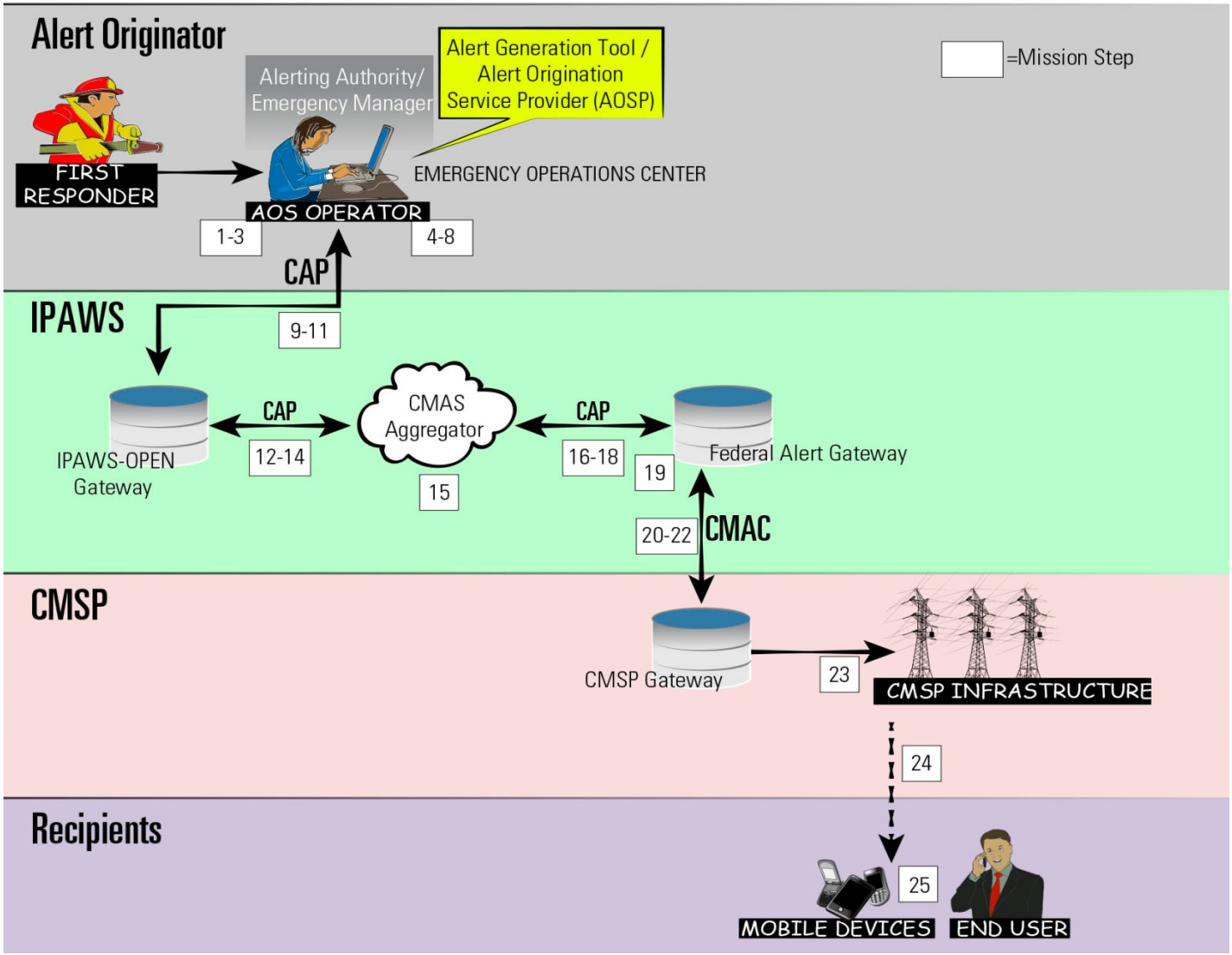
- How does the target of the analysis interface with other technologies?
- What is the flow of data in relation to the target of the analysis?

# Example: *Wireless Emergency Alerts (WEA) Service*

WEA is a major component of the Federal Emergency Management Agency (FEMA) Integrated Public Alert and Warning System (IPAWS).

- Enables federal, state, territorial, tribal, and local government officials to send targeted text alerts to the public via commercial mobile service providers (CMSPs)
- Customers of participating wireless carriers with WEA-capable mobile devices will automatically receive alerts in the event of an emergency if they are located in or travel to the affected geographic area.

# Example: *Swimlane Diagram for the WEA Service*



# Example: Mission Thread -1

| Step | Supporting Technologies |
|---|---|
| Alert Originating System (AOS) operator attempts to log on to the AOS. | • Server (valid accounts/authentication information)<br>• Logon application<br>• Communications between logon software/ server/AOS |
| AOS logon activates auditing of the operator's session. | • Auditing application<br>• Communications from accounts to auditing application<br>• Local/remote storage devices |
| AOS operator enters alert/cancel/update message with status of "actual." | • Alert scripts<br>• Graphical user interface (GUI) application<br>• Communications between GUI application and alert-generation software (including server and application) |
| AOS converts message to Common Alerting Protocol (CAP) compliant format. | • Conversion application |

# Example: Mission Thread -2

| Step | Supporting Technologies |
| --- | --- |
| CAP-compliant message is signed by two people. | • Signature entry application<br>• Signature validation application<br>• Public/private key pair for every user |
| AOS transmits message to the IPAWS OPEN Gateway. | • Application that securely connects to IPAWS<br>• AOS and IPAWS |

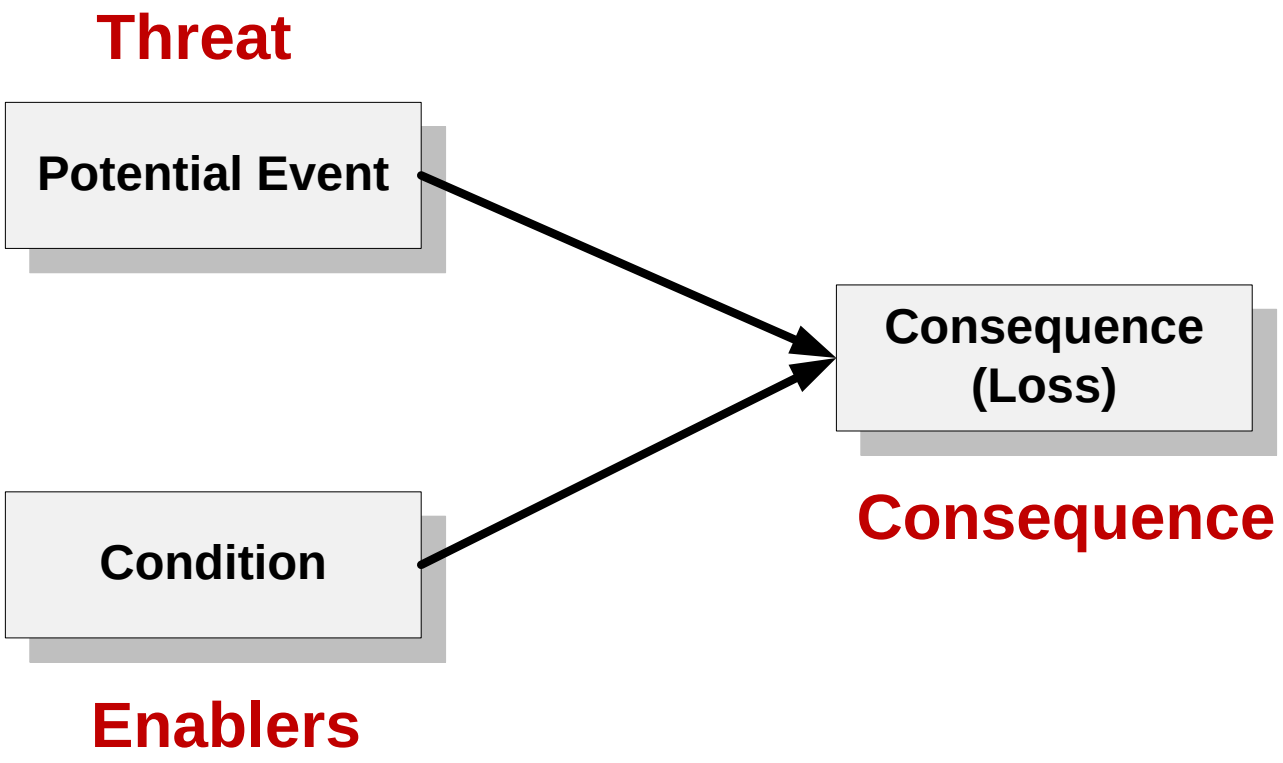
# Identify Risk (Task 2)

Security concerns are transformed into distinct, tangible risk scenarios that can be described and measured.

Sub-tasks:

- Identify threat.
- Establish consequence.
- Identify enablers.
- Document risk statement.

# Security Risk Components



# Task 2 Questions: Identify Threat

What scenarios are putting the target at risk?

- The actor poses as another actor or entity.
- Information or code is modified.
- Sensitive or proprietary information is viewed by the actor or other individuals.
- Access to important information or services is interrupted, temporarily unavailable, or unusable.
- Information is destroyed or lost.
- The actor (human) denies having performed an action that other parties can neither confirm nor contradict.
- The actor or other gains system access and privileges that he or she is not supposed to have.

Who or what is the source of the risk?

What is the motive of the source (if applicable)?

How is the target affected?

# Example: *Threat*

An outside attacker with malicious intent obtains a valid certificate and uses it to send an illegitimate CAP-compliant message that sends people to a dangerous location.

Threat components :

- *Actor*—a person with an outsider's knowledge of the organization
- *Motive*—malicious intent
- *Action*—the actor obtains a valid certificate and uses it to send an illegitimate CAP-compliant message that sends people to a dangerous location

# Task 2 Question: *Establish Consequence*

If the threat occurs, what impacts might ensue?

- Health and safety issues
- Financial losses
- Productivity loses
- Loss of reputation
- Other

# Example: *Consequence*

People could be put in harm's way, resulting in injuries and death.

Alert originators and state approvers could be held liable for damages.

The reputation of WEA could be damaged.

The reputations of alert originators could be damaged.

Future attacks could become more likely (i.e., copy-cat attacks).

# Task 2 Question: *Identify Enablers*

What conditions or circumstances are enabling the risk to occur?

- Organization, policy, or procedure weaknesses
- Technical weaknesses or vulnerabilities
- Actions of organizations staff (e.g., IT staff, users)
- Actions of collaborators or partners
- Interfaces of systems
- Data flows
- Software or system design
- Other

# Example: Enablers -1

A valid certificate could be captured by an attacker.

- Certificates are sent to recipients in encrypted email. This email is replicated in many locations, including
  - Computers of recipients
  - Email servers
  - Email server/recipient computer back-ups
  - Off-site storage of backup tapes
- The attacker could compromise the Emergency Operations Center or vendor to gain access to the certificate (e.g., through social engineering).
- Limited control over the distribution and use of certificates could enable an attacker to obtain access to a certificate.

Unencrypted certificates could be stored on recipient's systems.

Management of certificates is performed manually.

# Example: Enablers -2

An Emergency Operations Center's certificate would provide an attacker with access to all IPAWS capabilities.

The knowledge of what constitutes a CAP-compliant message is publicly documented.

The number of vendors that provide Alert Originating System (AOS) software is small. Each vendor controls a large number of certificates. A compromised vendor could provide an attacker with many potential targets.

# Task 2: *Risk Statement*

A risk statement is a succinct and unique description of a risk.

Risk statements typically describe

- A circumstance with the potential to produce loss (i.e., threat)
- The loss that will occur if that circumstance is realized (i.e., consequence)

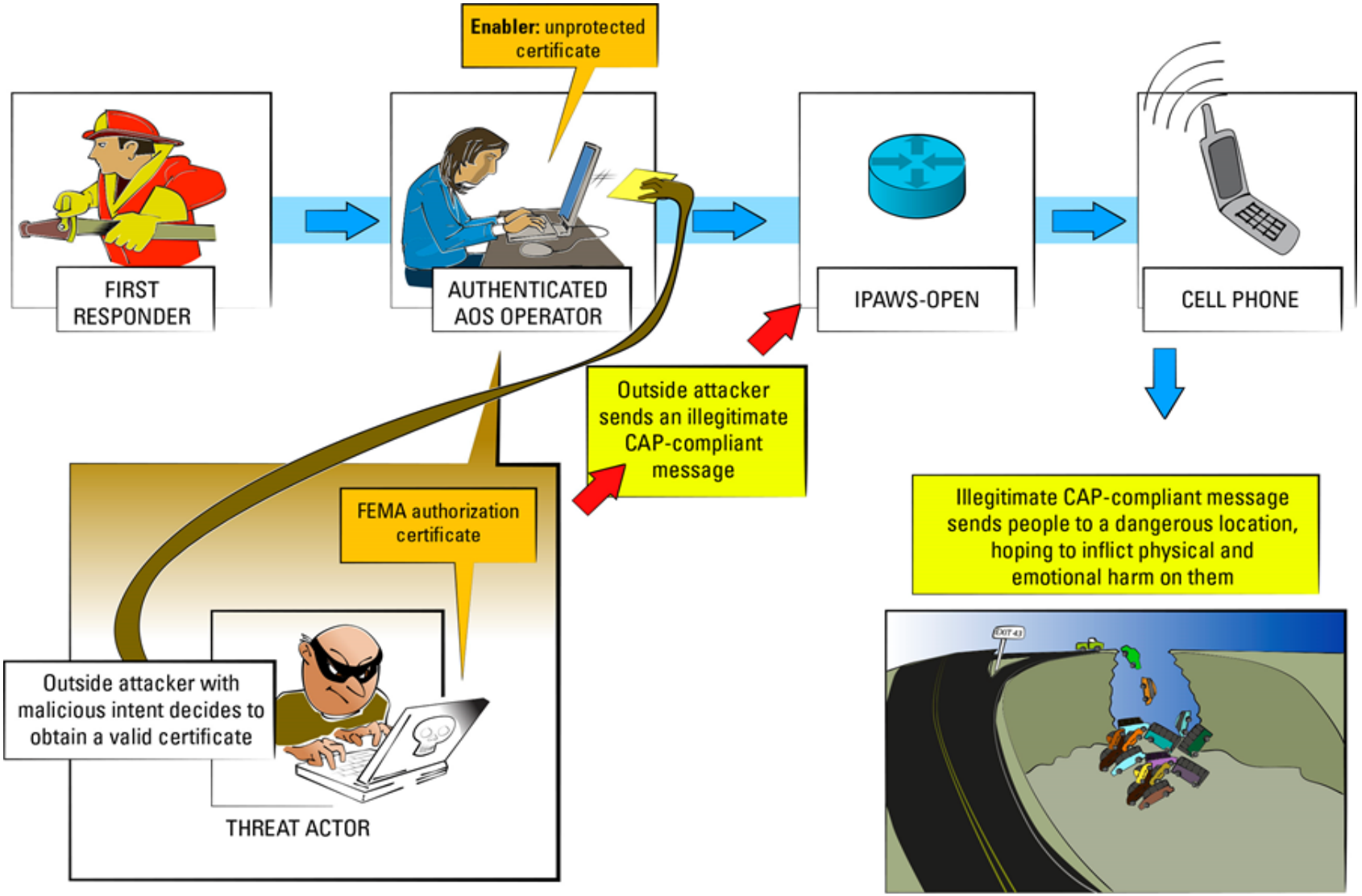
The if-then format is often used to capture a risk.

- The *if* part of the statement describes the threat.
- The *then* part conveys a summary of the consequences.

# Example: *Risk Statement*

If an outside attacker with malicious intent obtains a valid certificate and uses it to send an illegitimate CAP-compliant message that sends people to a dangerous location, then health, safety, legal, financial, and reputation consequences could result.

# Example: *Risk Scenario*



# Questions?


# Module 17: Risk Analysis for Software Asurance (Part 2)

(Developed by Christopher Alberts)

Introduction to Assured Software Engineering

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

# Notices



This material is based upon work funded and supported by the Independent Agency under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center sponsored by the United States Department of Defense.

# Analyze Risk (Task 3)

Each risk is analyzed in relation to predefined criteria.

Sub-tasks:

- Establish probability.
- Establish impact.
- Determine risk exposure.

# Task 3 Questions: *Establish Probability*

What is the probability that the risk will occur?

What is the rationale for your estimate of the risk's probability?

# Probability Criteria

| Value | Definition | Context/Guidelines/Examples |
|---|---|---|
| **Frequent (5)** | The scenario occurs on numerous occasions or in quick succession. It tends to occur quite often or at close intervals. | ≥ one time per month (≥ 12 / year) |
| **Likely (4)** | The scenario occurs on multiple occasions. It tends to occur reasonably often, but not in quick succession or at close intervals. | |
| **Occasional (3)** | The scenario occurs from time to time. It tends to occur “once in a while.” | ~ one time per 6 months (~ 2 / year) |
| **Remote (2)** | The scenario can occur, but it is not likely to occur. It has "an outside chance" of occurring. | |
| **Rare (1)** | The scenario infrequently occurs and is considered to be uncommon or unusual. It is not frequently experienced. | ≤ one time every 3 years (≤ .33 / year) |

# Example: *Probability*

Probability: Rare

Rationale:

- This risk requires that a complex sequence of events occurs.
- The attacker has to be highly motivated.
- An event that requires an alert to be issued must already be imminent. People will likely verify WEA messages through other channels. To make maximize the impact, the attacker will likely take advantage of an impending event.
- WEA will need to have an established track record of success for this risk to be realized. Otherwise, people might not be inclined to follow the instructions provided in the illegitimate CAP-compliant message.

# Task 3 Questions: *Establish Impact*

If the risk were to occur, what would its impact be?

What is the rationale for your estimate of the risk's impact?

# Impact Criteria

| Value | Definition |
|---|---|
| **Maximum (5)** | The impact on the organization is severe. Damages are extreme in nature. Mission failure has occurred. Stakeholders will lose confidence in the organization and its leadership. The organization either will not be able to recover from the situation, or recovery will require an extremely large investment of capital and resources. Either way, the future viability of the organization is in doubt. |
| **High (4)** | The impact on the organization is large. Significant problems and disruptions are experienced by the organization. As a result, the organization will not be able to achieve its current mission without a major re-planning effort. Stakeholders will lose some degree of confidence in the organization and its leadership. The organization will need to reach out to stakeholders aggressively to rebuild confidence. The organization should be able to recover from the situation in the long run. Recovery will require a significant investment of organizational capital and resources. |
| **Medium (3)** | The impact on the organization is moderate. Several problems and disruptions are experienced by the organization. As a result, the organization will not be able to achieve its current mission without some adjustments to its plans. The organization will need to work with stakeholders to ensure their continued support. Over time, the organization will be able to recover from the situation. Recovery will require a moderate investment of organizational capital and resources. |
| **Low (2)** | The impact on the organization is relatively small, but noticeable. Minor problems and disruptions are experienced by the organization. The organization will be able to recover from the situation and meet its mission. Recovery will require a small investment of organizational capital and resources. |
| **Minimal (1)** | The impact on the organization is negligible. Any damages can be accepted by the organization without affecting operations or the mission being pursued. No stakeholders will be affected. Any costs incurred by the organization will be incidental. |

# Example: *Impact*

Impact: High-Maximum

Rationale:

- The impact will ultimately depend on the severity of the event that is about to occur.
- Health and safety damages could be severe, leading to potentially large legal liabilities.
- The reputation of WEA could be severely damaged beyond repair.

# Task 3 Question: *Determine Risk Exposure*

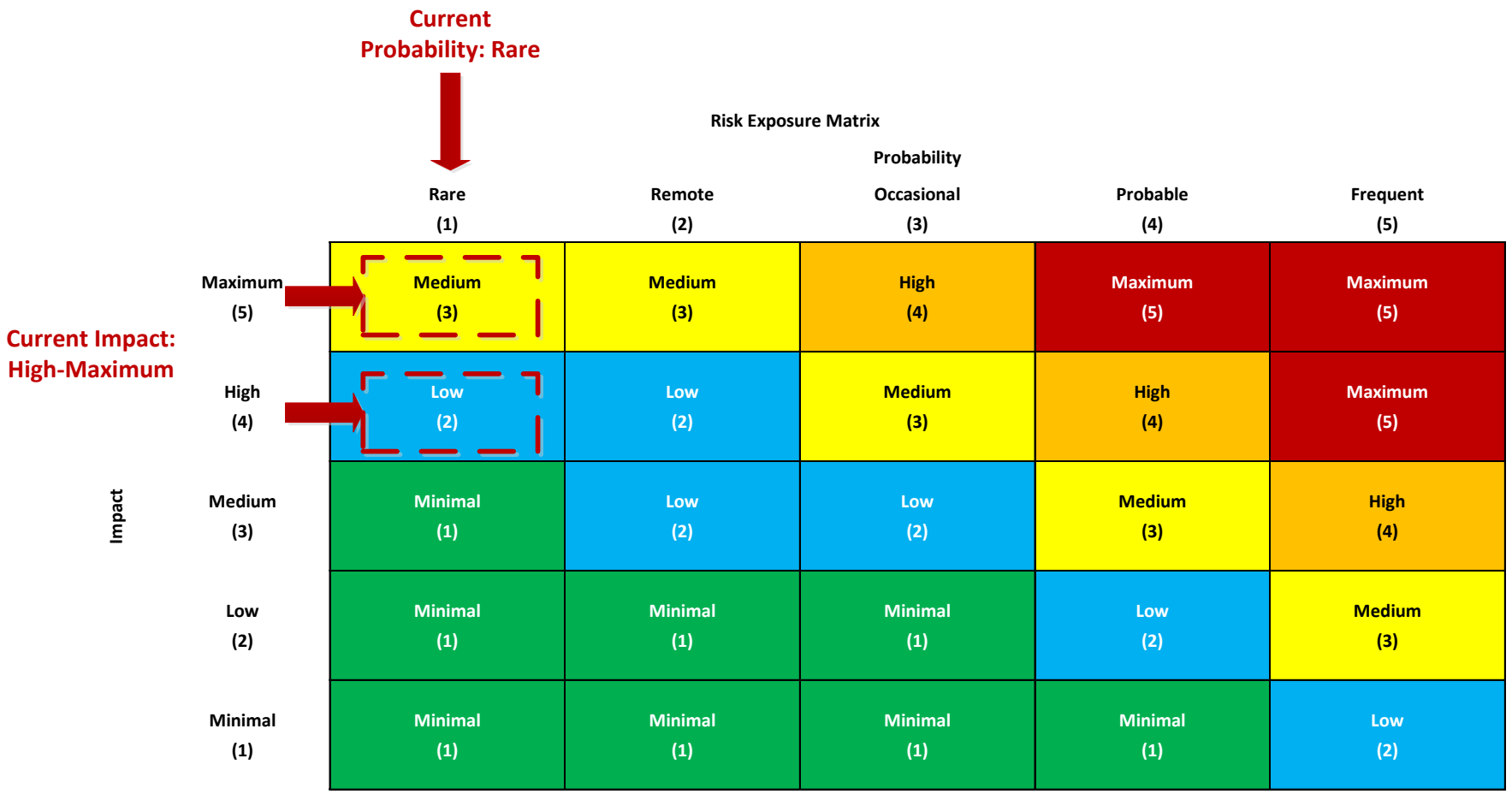
Based on the estimated values of probability and impact, what is the resulting risk exposure?

# Risk Exposure Matrix

**Risk Exposure Matrix**

| Impact \ Probability | Rare (1) | Remote (2) | Occasional (3) | Probable (4) | Frequent (5) |
|---|---|---|---|---|---|
| Maximum (5) | Medium (3) | Medium (3) | High (4) | Maximum (5) | Maximum (5) |
| High (4) | Low (2) | Low (2) | Medium (3) | High (4) | Maximum (5) |
| Medium (3) | Minimal (1) | Low (2) | Low (2) | Medium (3) | High (4) |
| Low (2) | Minimal (1) | Minimal (1) | Minimal (1) | Low (2) | Medium (3) |
| Minimal (1) | Minimal (1) | Minimal (1) | Minimal (1) | Minimal (1) | Low (2) |

# Example: *Risk Exposure*



# Determine Control Approach (Task 4)

A strategy for controlling each risk is determined based on

- Predefined criteria
- Current constraints (e.g., resources and funding available for control activities)

Control approaches for security risks include the following:

- Accept—If a risk occurs, its consequences will be tolerated.
- Transfer—A risk is shifted to another party (e.g., through insurance or outsourcing).
- Avoid—Activities are restructured to eliminate the possibility of a risk occurring.
- Mitigate—Actions are implemented in an attempt to reduce or contain a risk.

Sub-tasks:

- Prioritize risks.
- Select control approach.

# Task 4 Question: Prioritize *Risks*

Which risks are of highest priority?

- Use impact as the primary factor for prioritizing security risks.
  - Risks with the largest impacts are deemed to be of highest priority.
- Use probability as the secondary factor for prioritizing security risks.
  - Probability is used to prioritize risks that have equal impacts.
  - Risks of equal impact with the largest probabilities are considered to be the highest priority risks.

# Example: *Prioritized Risk Spreadsheet*

| ID | Risk Statement | Impact | Prob | Risk Exp |
|---|---|---|---|---|
| 1 | If an outside attacker with malicious intent obtains a valid certificate and uses it to send an illegitimate CAP-compliant message that directs people to a dangerous location, then health, safety, legal, financial, and reputation consequences could result. | High-Max | Rare | Low-Med |
| 3 | If an insider with malicious intent spoofs the identity of a colleague and sends an illegitimate CAP-compliant message, then individual and organizational reputation consequences could result. | Med | Rare-Remote | Min-Low |
| 2 | If malicious code prevents an operator from entering an alert into the Alert Originating System (AOS), then health, safety, legal, financial, and productivity consequences could result. | Low-Med | Remote | Min-Low |
| 4 | If the internet communication channel for the AOS is unavailable due to a cybersecurity attack on the internet service provider, then health and safety consequences could result. | Low-Med | Remote | Min-Low |

# Task 4 Questions: *Select Control Approach*

What approach will be used to control the risk?

- Accept
- Transfer
- Avoid
- Mitigate

What is the rationale for choosing that approach?

# Example: *Control Approach*

Control approach: Mitigate

Rationale:

- This risk could cause severe damages if it occurs, which makes it a good candidate for mitigation.
- Mitigations for this risk will be relatively cost effective.

# Example: *Risk Spreadsheet with Control Approach*

| ID | Risk Statement | Impact | Prob | Risk Exp | Control Approach |
|---|---|---|---|---|---|
| 1 | If an outside attacker with malicious intent obtains a valid certificate and uses it to send an illegitimate CAP-compliant message that directs people to a dangerous location, then health, safety, legal, financial, and reputation consequences could result. | High-Max | Rare | Low-Med | Mitigate |
| 3 | If an insider with malicious intent spoofs the identity of a colleague and sends an illegitimate CAP-compliant message, then individual and organizational reputation consequences could result. | Med | Rare-Remote | Min-Low | Mitigate |
| 2. | If malicious code prevents an operator from entering an alert into the Alert Originating System (AOS), then health, safety, legal, financial, and productivity consequences could result. | Low-Med | Remote | Min-Low | Mitigate |
| 4 | If the internet communication channel for the AOS is unavailable due to a cybersecurity attack on the internet service provider, then health and safety consequences could result. | Low-Med | Remote | Min-Low | Mitigate |

# Develop Control Plan (Task 5)

A control plan is defined and documented for all security risks that are not accepted (i.e., risks that will be mitigated, transferred, or avoided).

Sub-tasks:

- Review data.
- Establish control requirements.

# Task 5: Review Data

Operational context from Task 1:

- Mission and objective(s) of the workflow/mission thread
- Steps required to complete the workflow/mission thread
- Technologies (e.g., systems, applications, software, hardware) that support the workflow/mission thread
- How the target of the analysis supports the workflow/mission thread
- How the target of the analysis interfaces with other technologies
- The flow of data in relation to the target of the analysis

Risk data:

- Threat, enablers, and consequences from Task 2
- Impact and rationale, probability and rationale, and risk exposure from Task 3
- Control approach and rationale from Task 4

Risk spreadsheet

# Task 5 Questions: *Establish Control Requirements* -1

*Transfer:*

- What can be done to transfer the risk?
- How can the risk be shifted to another party?
- How will you know that the transfer works? Will you be adversely affected if the other party ignores the transfer?

*Avoid:*

- What can be done to avoid the risk?
- How can activities be restructured [or requirements altered] to eliminate the possibility of the risk occurring?

# Task 5 Questions: *Establish Control Requirements -2*

*Mitigate:*

- What can be done to mitigate the risk?
- Which actions can be implemented to reduce or contain the risk?
  - *Monitor and Respond:*
    - What can be done to monitor and respond to the threat?
  - *Protect/Resist:*
    - What can be done to protect against or resist the threat? What can be done to protect against or resist the consequence?
  - *Recover:*
    - What can be done to recover from the risk when it occurs?

# Example: *Mitigation Plan* -1

Monitor and Respond

- IPAWS should send an alert receipt acknowledgement to an email address designated in the Memorandum of Agreement (MoA). (This approach uses an alternate communication mechanism from the sending channel.) The alert originator should monitor the IPAWS acknowledgements sent to the designated email address. The alert originator should send a cancellation for any false alerts that are issued.
- The alert originator should designate a representative for each distribution region to monitor for false alerts. The representative should have a handset capable of receiving alerts that are issued. If a false alert is issued, the designated representative would receive the alert and should then initiate the process for sending a cancellation for the false alert.

# Example: *Mitigation Plan -2*

Protect

- The alert originating system should use strong security controls to protect certificates.
  - Access to certificates should be monitored.
  - Encryption controls should be used for certificates during transit and storage.
  - Access to certificates should be limited based on role.
- All alert transactions should have controls (e.g., time stamp) to ensure that they cannot be rebroadcast at a later time. (Note: This requirement requires that the sender time stamps the alert appropriately. The receiver of the alert would need to check the time stamp to determine whether the alert is legitimate or a relay of a previous alert.)
- Certificates should expire and be replaced on a periodic basis.
- The alert originator should provide user training about security procedures and controls.

# Example: *Mitigation Plan -3*

Protect (cont.)

- Certificates should expire and be replaced on a periodic basis.
- The alert originator should provide user training about security procedures and controls.

# Example: *Mitigation Plan -4*

Recover

- The alert originator should quickly issue a cancellation before people have a chance to respond to the false alert (i.e., before they have a chance to go to the dangerous location). This might require alert originators to provide additional training and to conduct additional operational exercises.
- The alert originator should notify FEMA to determine how to cancel the compromised certificate.

# Summary

# Key Points -1

The basic goal of risk analysis is to provide decision makers

- With the information they need
- When they need it
- In the right form

If decisions are not influenced by risk analysis activities, then risk analysis provides no added value.

Risks, issues/problems, opportunities, and strengths are part of an interrelated causal chain of conditions and events that must be managed.

- Mission risk aggregates the effects of multiple conditions and events on a system's ability to achieve its mission.
- Event risk is the probability that an event will lead to a negative consequence or loss.

# Key Points -2

The Security Engineering Risk Analyses (SERA) assesses operational security risks early in the software lifecycle.

- Requirements
- Architecture
- Design

The SERA method employs structured, systematic risk analysis to

- Handle the complex nature of security risk
- Identify and address design vulnerabilities early in the lifecycle (i.e., build security in)

# Publications and Resources -1

Cyber Security Engineering (CSE) Team Web Page

http://www.cert.org/sse/

Alberts, Christopher & Dorofee, Audrey. Mission Risk Diagnostic (MRD) Method Description (CMU/SEI-2012-TN-005). Software Engineering Institute, Carnegie Mellon University, 2012.

http://www.sei.cmu.edu/reports/12tn005.pdf

Alberts, Christopher; Allen, Julia; & Stoddard, Robert. Risk-Based Measurement and Analysis: Application to Software Security (CMU/SEI-2012-TN-004), Software Engineering Institute, Carnegie Mellon University, 2012.

http://www.sei.cmu.edu/reports/12tn004.pdf

# Publications and Resources -2

Alberts, Christopher & Dorofee, Audrey. A Framework for Categorizing Key Drivers of Risk (CMU/SEI-2009-TR-007). Software Engineering Institute, Carnegie Mellon University, 2009.

http://www.sei.cmu.edu/library/abstracts/reports/09tr007.cfm

SEI Mission Success in Complex Environments (CSE) Special Project

http://www.sei.cmu.edu/risk/

# Questions?


# Module 18: Design Patterns

(Authored by Kevin Gary, Arizona State University)

Introduction to Assured Software Engineering

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

# Notices



This material is based upon work funded and supported by the Independent Agency under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center sponsored by the United States Department of Defense.

# Topics

What are patterns?

Patterns history

Pattern types and examples

# What Are Patterns?

Patterns are *reusable solutions* in a *context.*

- Context: when, where, tradeoffs, lesson-learned
- There are all kinds of patterns:
  - Design patterns
  - Analysis patterns – tend to be domain-specific interpretations
  - Process patterns (Coplien) – reusable business process segments

Patterns capture domain experiences from master practitioners that solve real problems.

Patterns form a common vocabulary within a team or organization.

# Patterns, Styles, Idioms and DSSAs -1


Design Patterns

- Tactical decisions, choice has local scope
- Typically described with class-level interactions

Architectural Patterns and Styles

- Strategic decisions impacting broad portions of a system
- Fundamental structural organization for software system

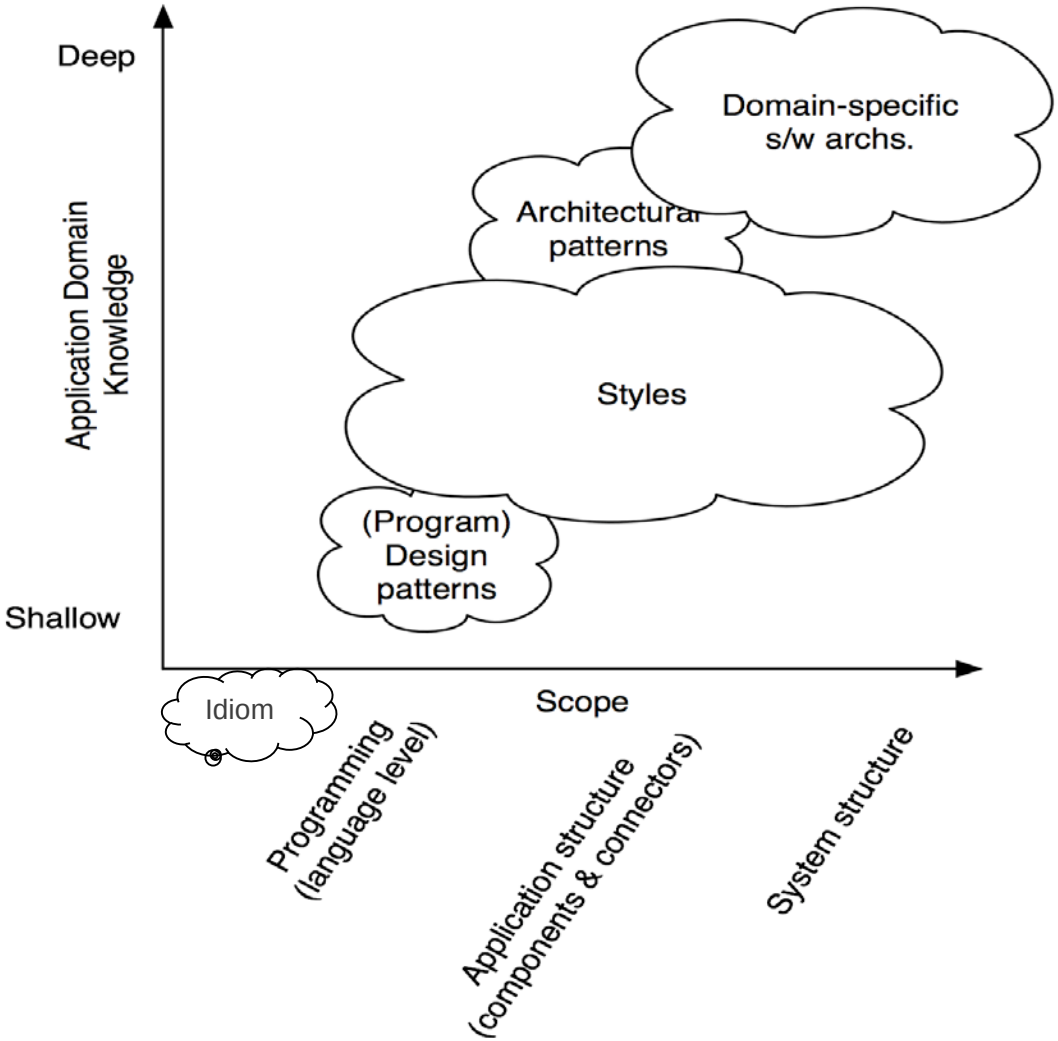
Idioms

- Tribal knowledge particular to a language or technology

Domain-specific Patterns

- Vertical domains sharing solutions to common problems

# Patterns, Styles, Idioms and DSSAs -2




*Software Architecture: Foundations, Theory, and Practice*; Richard N. Taylor, NenadMedvidovic, and Eric M. Dashofy; © 2008 John Wiley & Sons, Inc. Reprinted with permission.

# Patterns History

Pattern coined by architect Christopher Alexander

> *"Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing the same thing twice. For convenience and clarity, each pattern has the same format."*


Software patterns began in late 80's

- Cunningham & Beck's Smalltalk UI patterns
- Jim Coplien's C++ idioms
- Erich Gamma's work on recording design structures

Milestone book Design Patterns by Gang of Four (GoF)

- Defines and categorizes 24 patterns used commonly in object-oriented designs – have become part of the community's vocabulary

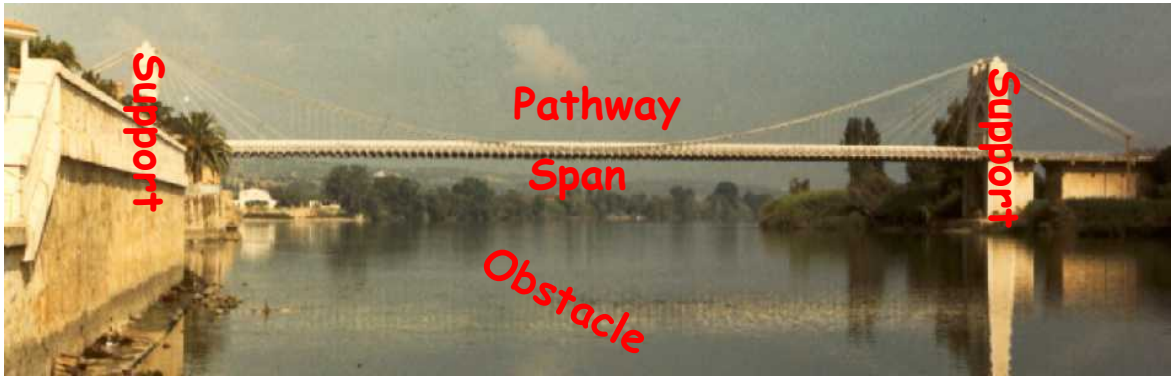
# A Simple Pattern Example

Name: Bridge Pattern (not the GoF version)

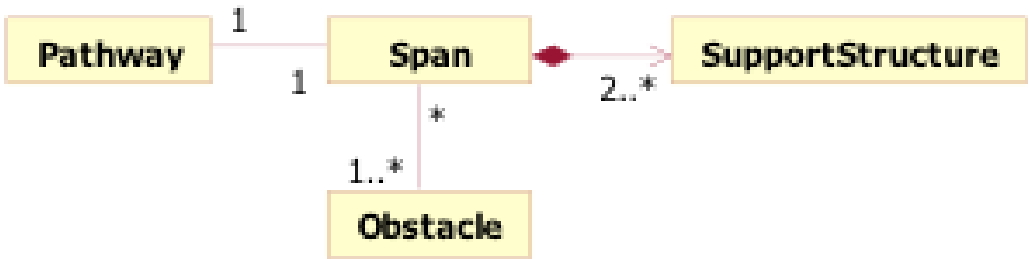
Problem: An obstacle blocks a pathway

Solution: Attach a span across support structures that accommodates required travel on the pathway

Example:




Pattern Structure:



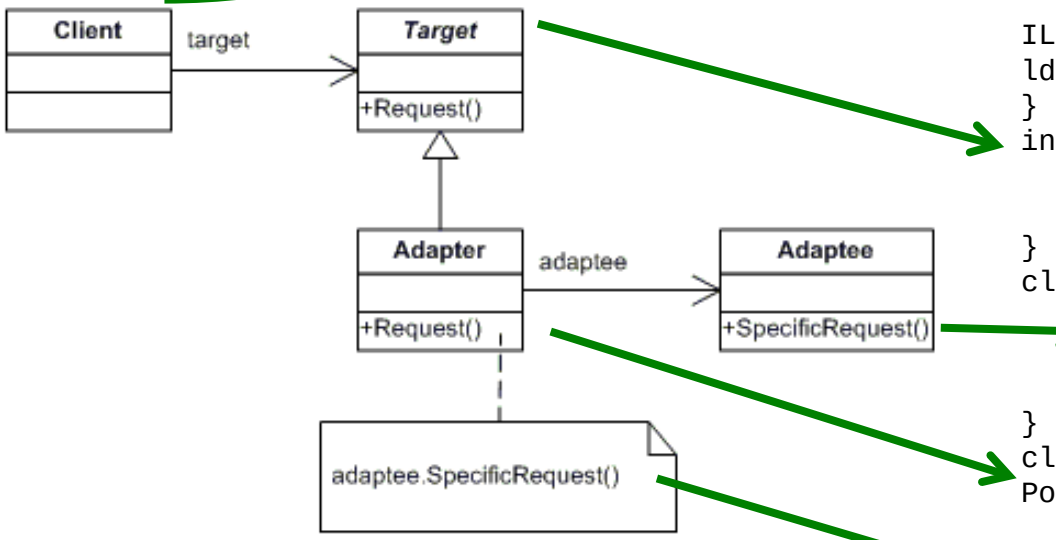
# Pattern Format (GoF format)

| | |
|---|---|
| **Name**: | *Bridge* |
| **Intent**: | To provide access to a pathway over an obstacle. The obstacle is commonly at or below the level of the pathway. |
| **Problem**: | An obstacle blocks a pathway requiring travelers to journey long distances around the obstacle to complete their passage. |
| **Solution**: | A span is built on top of multiple, anchored support structures so it clears the obstacle and supports travelers on the pathway. |
| **Structure**: | In general, two or more fixed support structures bear the load of the span providing access over an obstacle. |
| **Behavior**: | The pathway continues over the span. As weight on the span increases, it is transferred to the fixed support structures. |
| **Implementation**: | *<description in appropriate notation, commonly C/C++/Java for software>* |
| **Known uses**: | Used successfully over broad waterways and deep valleys |
| **Consequences**: | Useful where construction costs can achieve ROI of enabled route. Must use creative designs (see Draw Bridge Pattern) when obstacle is water and requires ship passage. |

# Pattern Example: Adapter

Translates an interface to a compatible interface.

- The adapter pattern is useful in situations where an already existing class provides some or all of the services you need but does not use the interface you need.




```
class App {
  public static void main(Stringargs[]) {
         // some initialization here
ILineDrawer ld = LineFactory.getDrawer()
ld.drawLine();
}
interface ILineDrawer {
   // draw a line from origin to point
   public void drawLine (intx, inty);
}
class PolarLine {
  public void line(double angle) {
         // draws line based on radial
   }
}
class PolarAdapter {
PolarLinepline = new PolarLine();
   public void drawLine(intx, inty)
         double rad = Math.atan((double)x / y);
pline.line(rad);
   }
}
```

# Pattern Types (from GOF)

Creational patterns

- Decouple clients from knowing how an object is created
- Examples: Singleton and Abstract Factory

Behavioral patterns

- Provides guidance on where to implement responsibilities
- Examples: Observer and Strategy

Structural patterns

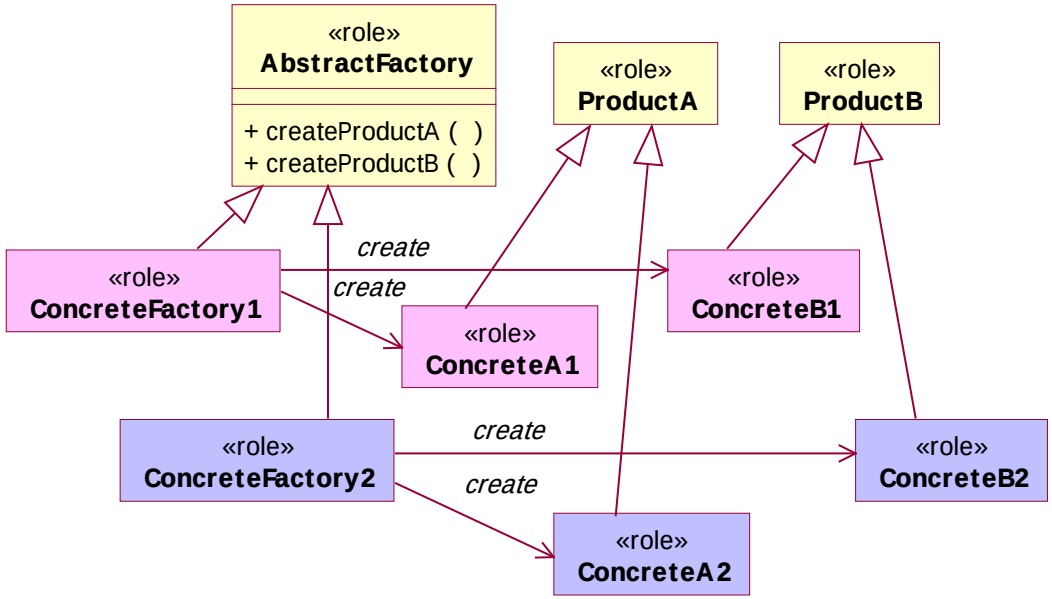
- Provides guidance on composing objects and classes
- Example: Decorator and Adapter

***A pattern is a (recurring) solution to a problem in a given context.***

# Creational Patterns: Abstract Factory

Intent: Provide an interface for *creating* families of related or dependent objects without specifying their concrete classes

- Decouple clients from knowing how to create *product*



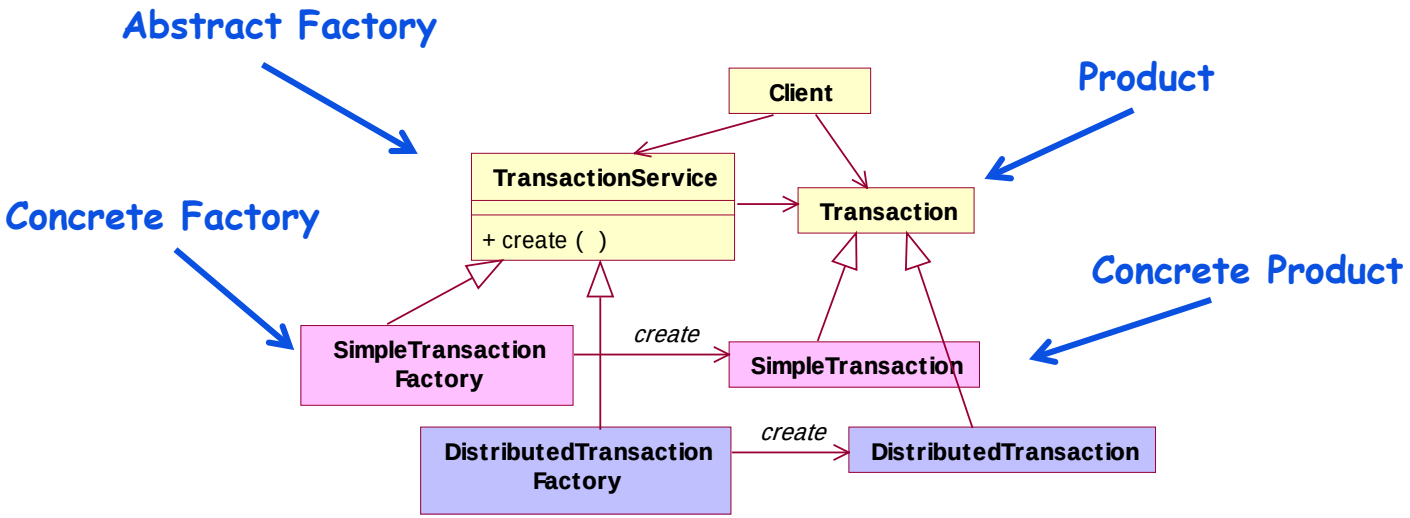

**Abstractions (what the client sees)**

**One instantiation**

**Another instantiation**

# Abstract Factory Example

Decouple clients from specific service implementations



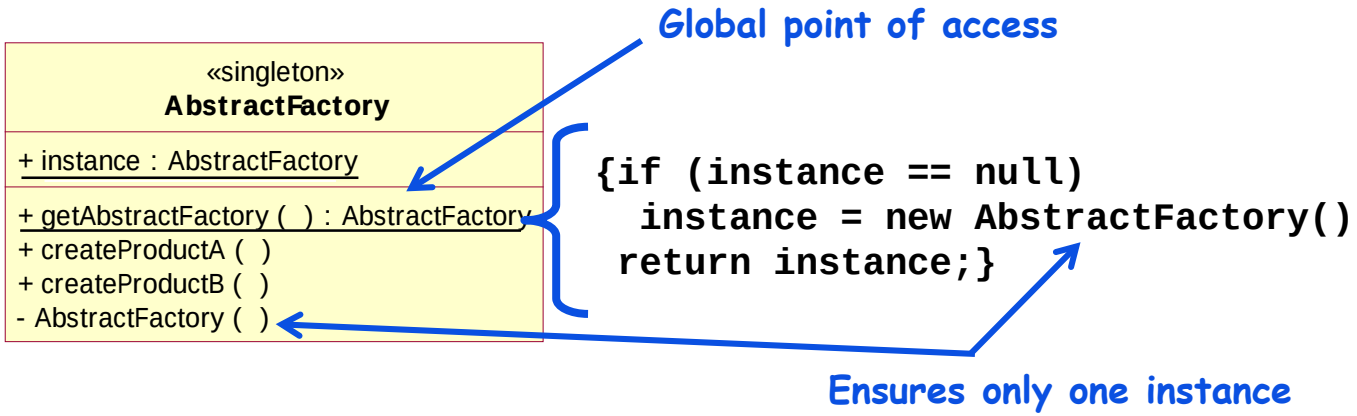
# Creational Patterns: Singleton

Ensures a class has only one instance, and provides a global point of access to it

- A very popular pattern and commonly needed by other patterns
- Implementations typically use class-scope to provide global access
- Different than using a class with static variables and methods, as it is still a stateful object

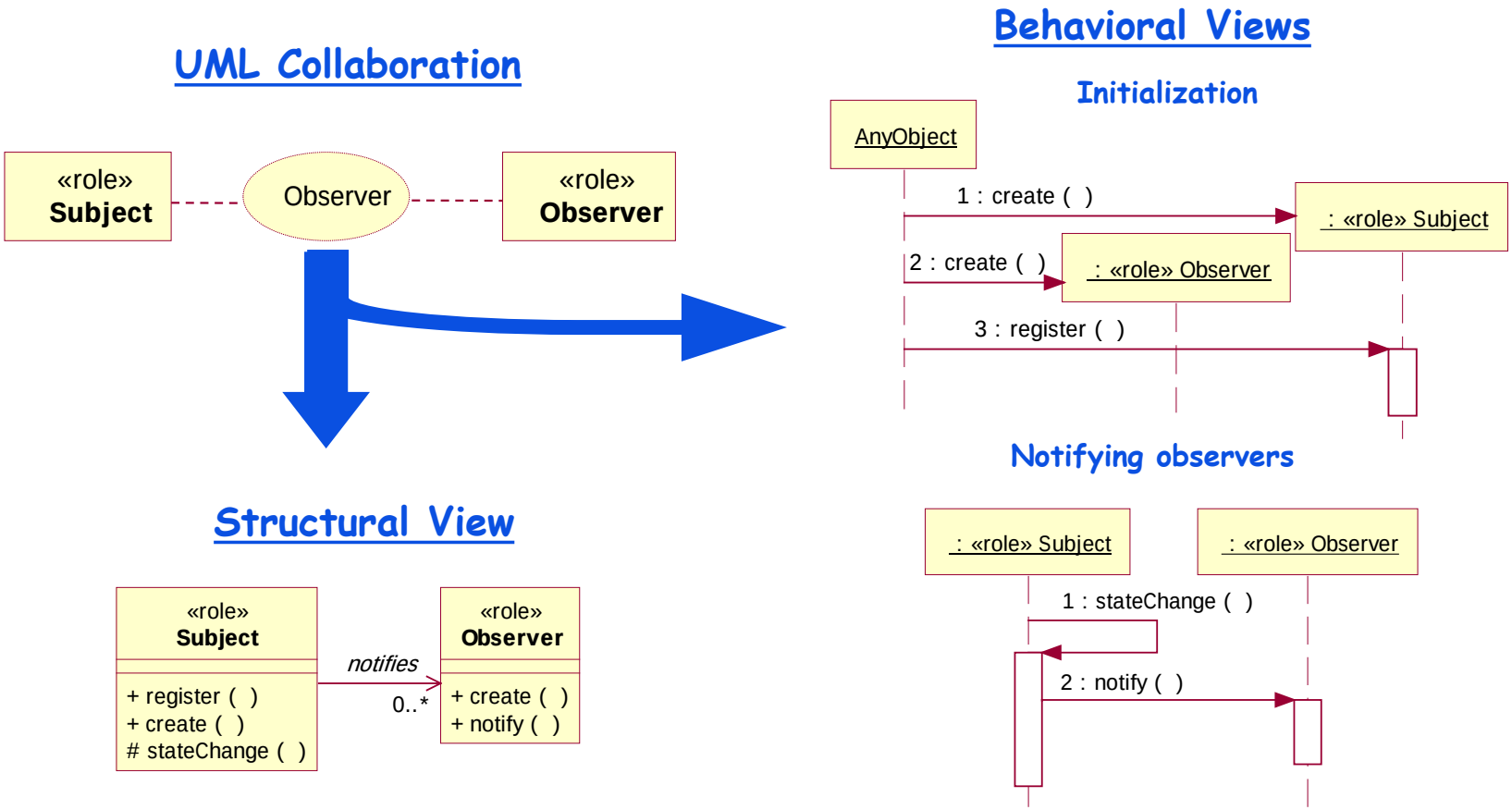
Example – how do clients obtain the AbstractFactory?

- One strategy is a singleton



# Behavioral: Observer Pattern

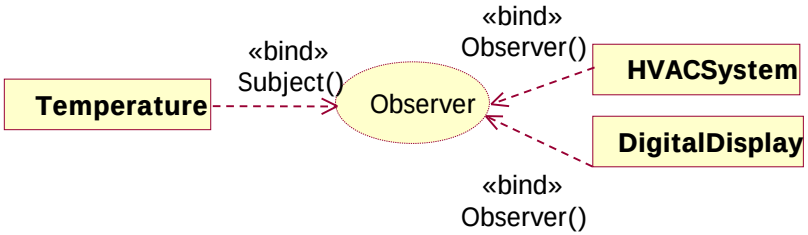
*Defines a one-to-many dependency between a subject and observers. When the subject object changes state, all its observer objects are notified.*



# Applying the Observer Pattern

## Collaboration Template


## Instantiation




```
interface Observer {
public abstract void notify();
}
```

```
class DigitalDisplay realizes Observer {
public void notify() {
        //update digital display
    }
}
```

Participating in a pattern adds responsibilities to an object's implementation

```
class Temperature {
List<Observer>myObservers;

    public register (Observer newOne) {
myObservers.add(newOne);
    }

    // Internally detect state change
    protected detectTemperatureChange() {
        for each observer in myObserver
observer.notify();
    }
}
```

# Java Support for Observers

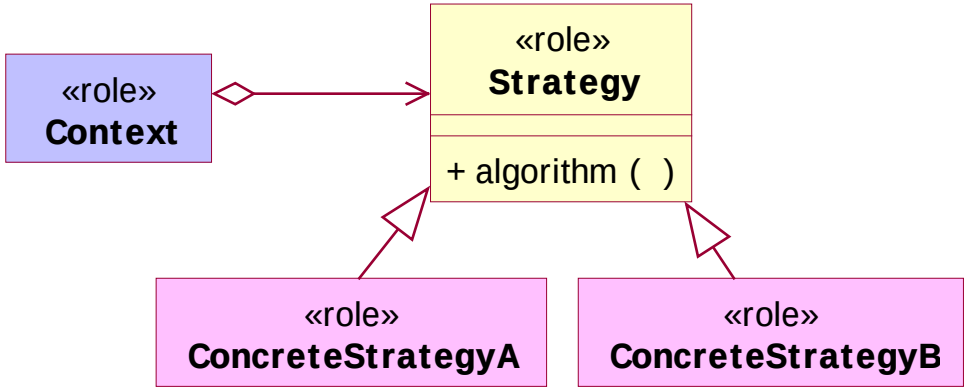
Java provides two types of Observer mechanisms:

- Listeners
  - Commonly associated with GUI applications
  - "Lightweight" in that the observed object is responsible for maintaining the list of listeners and notifying them.
- Observable/Observer
  - A class that allows itself to be watched must extend Observable
  - A class that watches an Observable implements Observer
  - Thoughts:
    - Basically provides an implementation of the Listener approach for you by providing methods on Observable like *addObserver*, *notifyObservers*, etc.
    - Since your class must extend Observable, it becomes tightly coupled to that inheritance hierarchy.

# Behavioral Patterns: Strategy

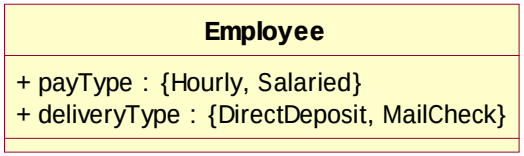
Intent: Define a family of algorithms, encapsulate each one, and make them interchangeable

- Allows algorithm to be managed independently instead of inside the method with a large *switch* statement
- Allows new algorithms to be easily added
- Allows context to change strategy dynamically



# Strategy Pattern Example

Example: An employee's pay type (hourly, salaried) and method (direct deposit, mail check) vary based on type.



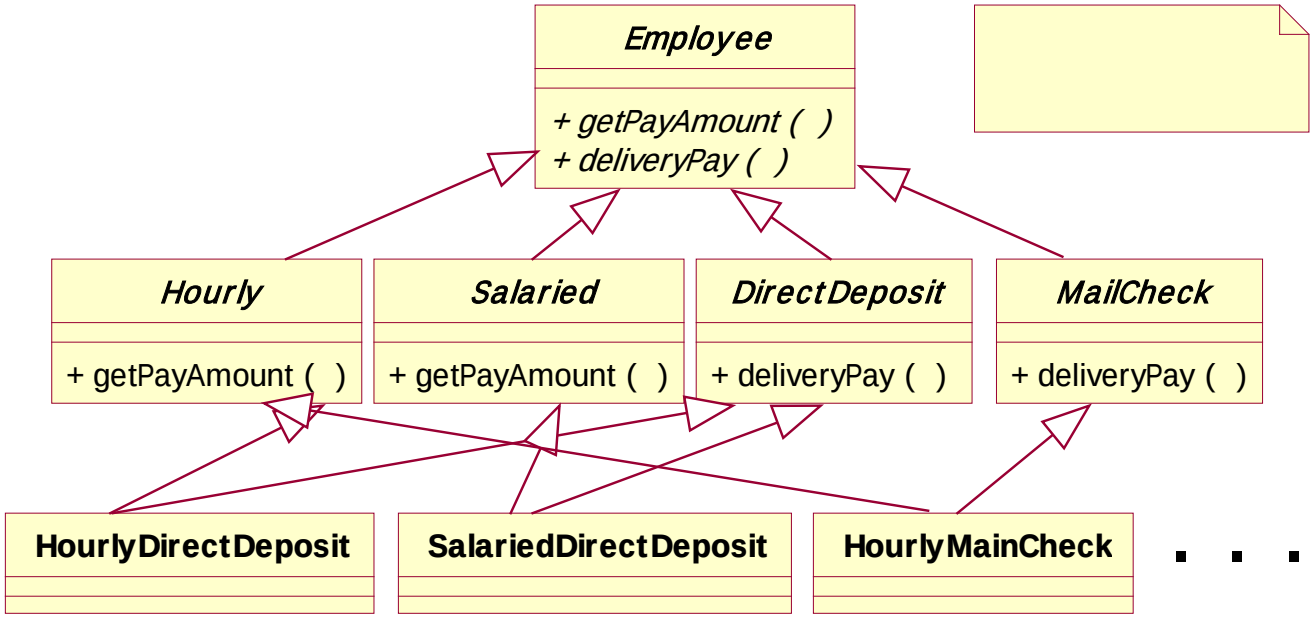

```
switch(employee.payType) {
    case Hourly:
        switch(employee.deliveryType) {
            case DirectDeposit: ...
            case CheckMailedToEmployee: ...
        }
    case Salaried:
        switch(employee.deliveryType) {
            case DirectDeposit: ...
            case MailCheck: ... break;
        }
}
```

**Clients are responsible for knowing all types and permutations!**
***Poor Separation of Concerns***

# Strategy Pattern Example – Inheritance

Behold the power of inheritance to really confuse a design!

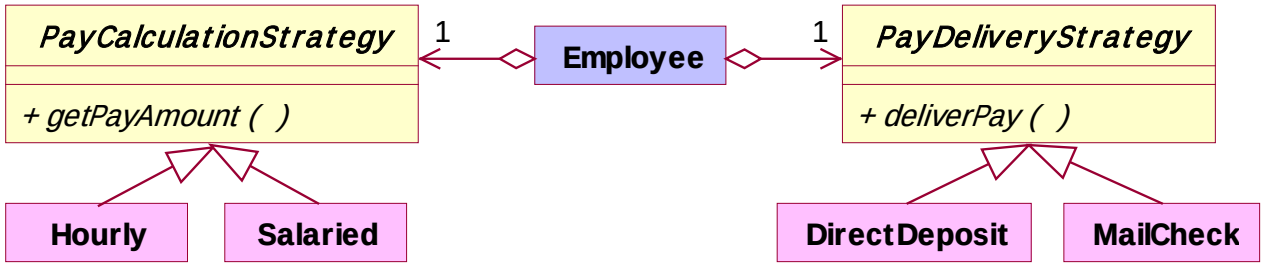
- Still exposes all permutations



# Strategy Pattern Example

Encapsulate each algorithm under a strategy.

- Manage each behavior in its own location.
- Dynamically change binding of behavior to its context.
- Limitations – many single-method classes



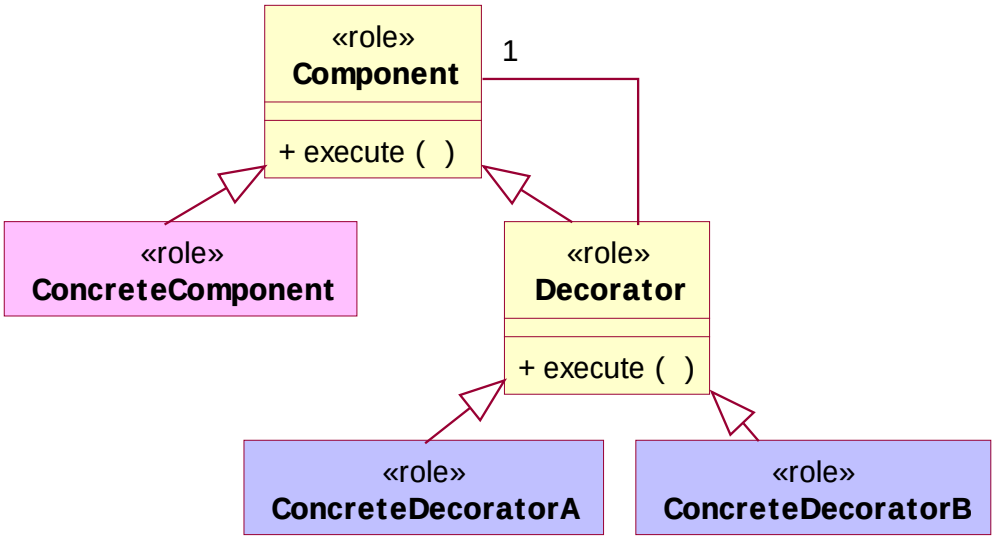

```
class Employee implements PayCalculationStrateg,
PayDeliveryStrateg {
   // assume constructor initializes these
PayCalculationStrategcalcStrategy;
PayDeliveryStrategdeliveryStrategy;

   public float getPayAmount() {
           return calcStrategy.getPayAmount();
   }
   public void deliverPay() {
deliveryStrategy.deliverPay();
   }
}
```

```
class Hourly implements PayCalculationStrateg {
   public float getPayAmount() {
            return (rate * hoursWorked);
}  }
class Salaried implements PayCalculationStrateg {
   public float getPayAmount() {
            return (salary / payPeriods);
}  }
class DirectDeposit implements PayDeliveryStrateg {
   public void deliverPay() {
            // deliver electronically;
}  }
class MailCheckimplements PayDeliveryStrateg {
   public void deliverPay() {
            // deliver by snail mail;
}  }
```

# Structural Patterns: Decorator

Intent: Attach additional responsibilities to an object dynamically.

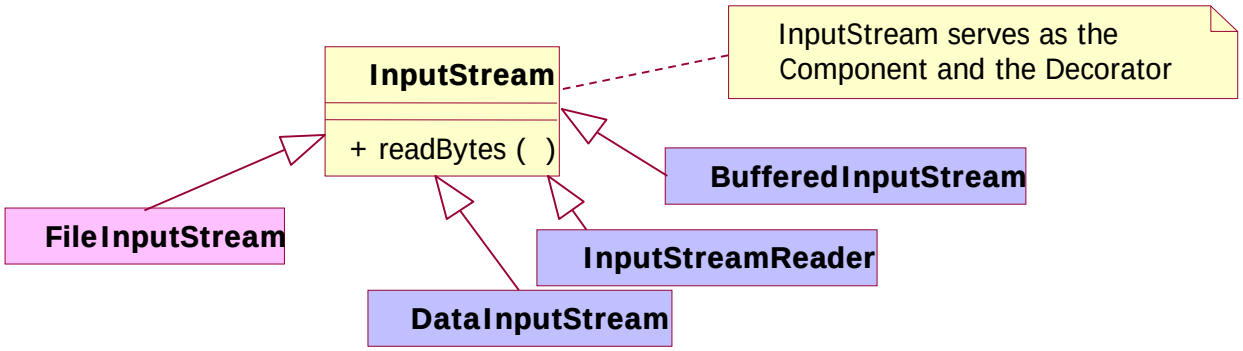
- Can easily add new decorations to existing system
- Decorators provide an alternative to subclassing for extending functionality.



# Decorator Example (Java Streams)

Java provides a rich set of stream classes for reading bytes and interpreting them.

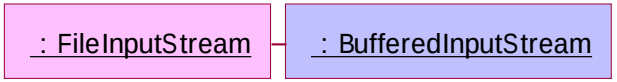
- *Input stream* interface for reads bytes
- *Buffered stream* read bytes from another stream & buffers
- *Data stream* reads bytes& converts to primitive types
- *InputStreamReader* reads bytes and converts to char



# Decorator Example (Java Streams)

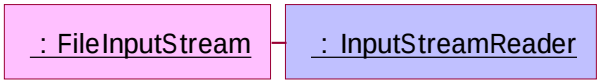
Client reads bytes


Client reads bytes that are buffered




Client reads primitive data with bytes being buffered


Client reads characters given a byte encoding (ASCII, UTF-8), no buffering




Other Java technologies define extensions of InputStream so their streams can play in the decoration process.

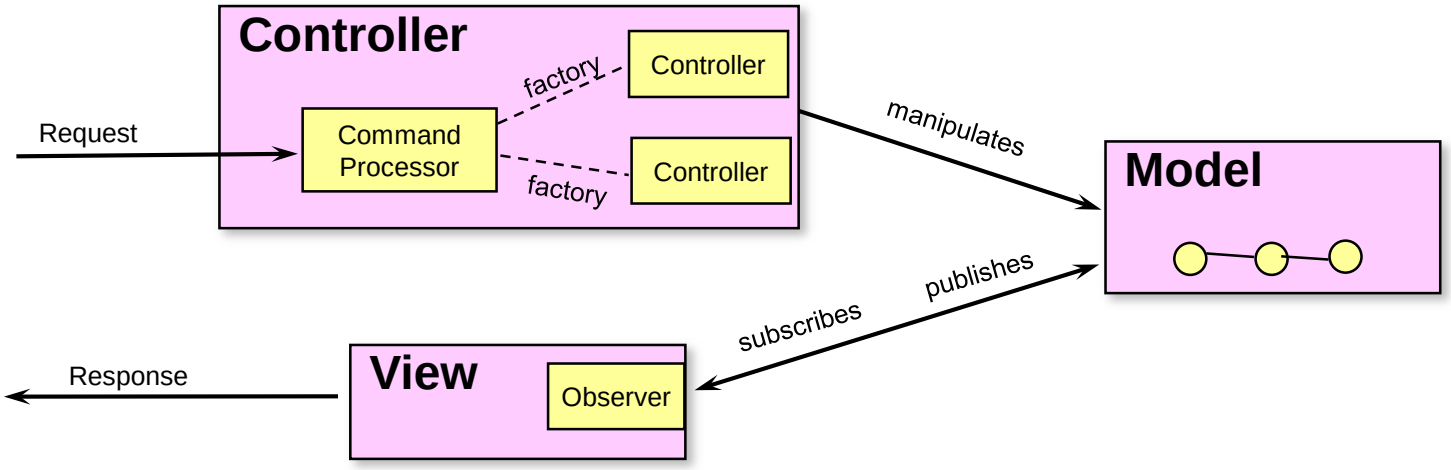
# Pattern Systems

Pattern systems: relating patterns to solve problems

> *"There is a structure on patterns which describes how each pattern is itself a pattern of other small patterns [and] the way they can be arranged with respect to other patterns"* – Alexander

Example – implementing MVC might use:

- *Publish-Subscribe* to define an *Observer* for model changes
- *Command Processor* to decouple behavior from controllers
- *Factory Method* to create the controller



# Anti-Patterns

A *bad* solution to a problem in a context

- What not to do is as important as what to do.

Examples - “Vendor Lock-in”, “Analysis Paralysis”

- See “Big Ball of Mud”

# Design Patterns Summary

Patterns capture domain experiences from master practitioners that solve real problems.

- Provide good separation of concerns by placing system responsibilities in the “best” location
- Loose coupling – indirection provides flexibility and reuse
- Favor delegation over inheritance

Patterns raise the vocabulary of models & teams.

- Less effort explaining parts of systems
- Less effort understanding code

Exploiting patterns requires thinking abstractly vs. thinking code-centrically.

Patterns convey more than a design solution - also a context for the solution.

# Design Patterns Wrap-up

Design pattern themes

- Provide good separation of concerns by placing system responsibilities in the “best” location
  - “Best” is subjective as there are many conflicting concerns
  - Design is trade-offs!
- Loose coupling – indirection provides flexibility and reuse
- Favor delegation over inheritance

Patterns raise the vocabulary level of models and teams.

- Less effort explaining parts of systems
- Less effort understanding code when we understand system’s common patterns

# References and Interesting Reading

"A Pattern Language", Christopher Alexander, 1977.

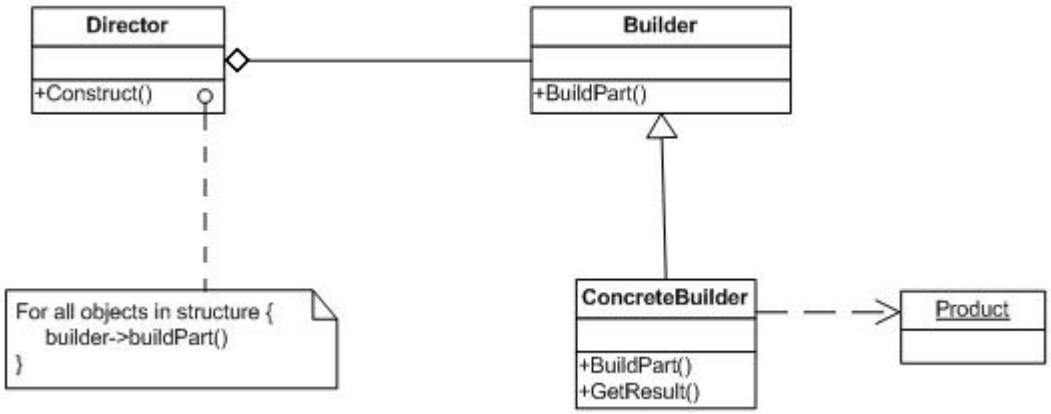
"Design Patterns: Elements of Reusable Object-Oriented Software" Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides.

"Pattern-Oriented Software Architectures: A System of Patterns", Frank Buschmann, ReginеMeunier, Hans Rohnert, Peter Sommerland, Michael Stal.

"Analysis Patterns : Reusable Object Models", Martin Fowler

http://www.martinfowler.com - Martin Fowler's web site

# Design Pattern: Builder




Goal of pattern is to produce a complex object.

- Often this “object” is a Façade or Composite.

Object model is not built “in one shot”.

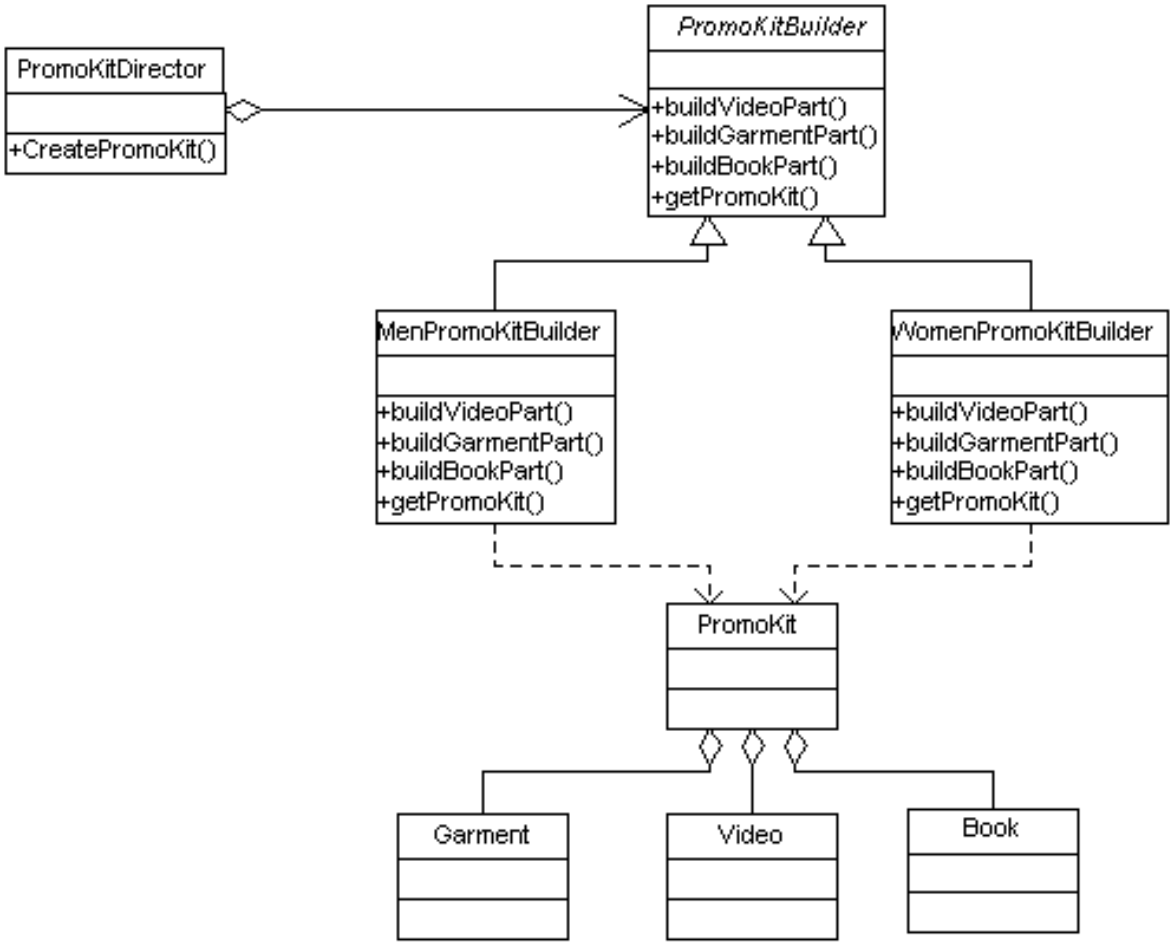
- The creation process is itself complex, has many conditions, and may partially succeed or fail.

You often refactor into a Builder from a Factory Method or Abstract Factory.

- You don’t set out to use this pattern in the first place.

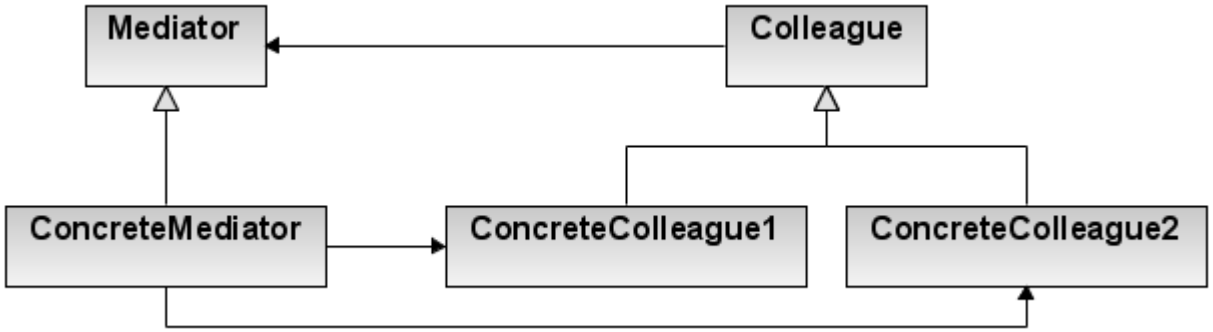
See also: Prototype

# Builder Example




From http://www.apwebco.com/gofpatterns/creational/Builder.html

# Design Pattern: Mediator



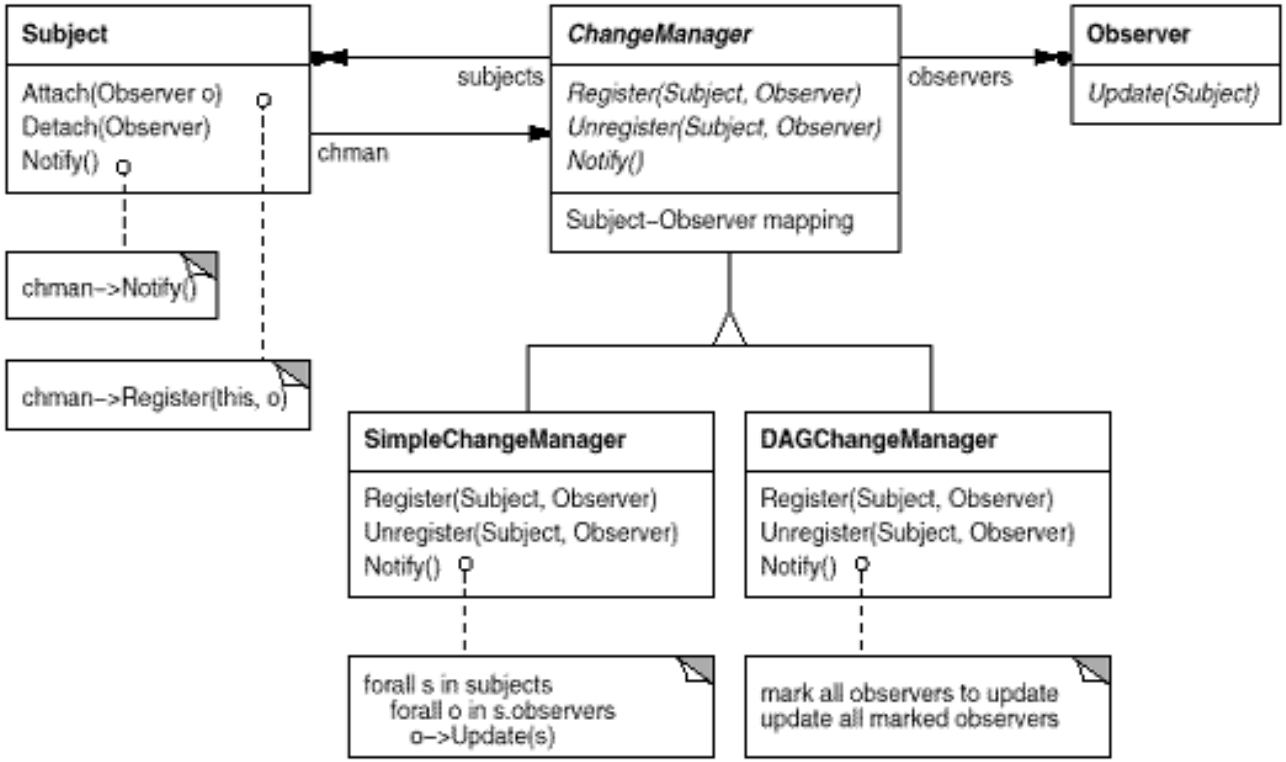

Not enough to have structural patterns that define the static relationships in a complex object structure

Also need behavioral patterns to describe how complex object structure communicate

*Mediator* does this by encapsulating the communication between collections of objects.

- Goal is similar to Observer – decouple the objects by externalizing the communication.
- Implementation is opposite however, Mediator encapsulates, Observer separates.

# Mediator Example




ChangeManager is the Mediator.

Note that Observers can be used internally within the Mediator!

Observers can be used side-by-side with Mediators too!

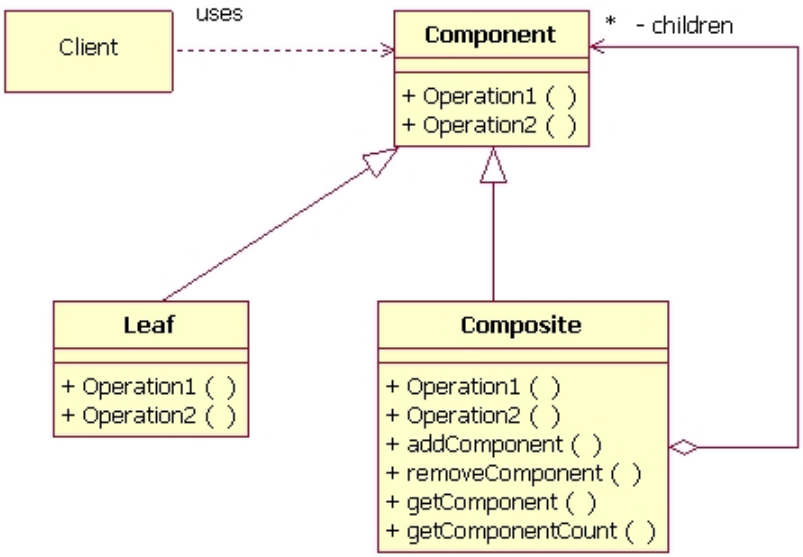
How does the interaction diagram change between Observer& Mediator?

# Design Pattern: Composite

Structural Pattern used when:

- An object may have substructure of the same type as itself!
- Example: definition of a binary tree
  - "A binary tree is a data structure where the tree may possess two trees, a left tree and a right tree."
  - Example: definition of a well-formed XML document
  - "A well-formed XML document is comprised of elements, which themselves may have elements nested within them."
- Example: definition of a company
  - "A company is comprised of employees, each of whom has a supervisor who is her/himself an employee, except the CEO. Some employees do not supervise anyone."
- The first 2 definitions suggest containment, but the last 1 is not a containment, but a "supervises" relationship.

# Composite Design Pattern




Note the initial awkwardness

- Composite specializes Component
- Composite holds 1-n references to children of type Component
- Therefore, a child of a Composite can be a Composite itself or a Leaf

**To Note:**

Client only sees Component; Composite methods for structure used by someone else (*Builder*?).

Often a special case is made of leaf nodes of the inherent tree; not strictly necessary.

- Usually Leaf would *Decorate* Component (meaning an operation3)

The Composite's implementation of business operations is usually to delegate to children (down the tree) and aggregate results back up (roll up the tree).

Composite often used in conjunction with Decorator, Visitor, Factory, Builder, and lots more!

# Composite Source Code

```
abstract class FileSysComp {
  String name;


  public FileSysComp(String cName) {
    name = cName;
  }
  public void addComponent(FileSysComp comp)
      throws Exception {
    throw new Exception("Invalid Operation");
  }
  public FileSysComp getComponent(int compNum)
      throws Exception {
    throw new Exception("Invalid Operation");
  }
  public abstract long getComponentSize();
} // End FileSysComp



class FileComp extends FileSysComp  {

  private long size;

  public FileComp(String cName, long sz) {
    super(cName);
    size = sz;

  }
  public long getComponentSize() {
    return size;
  }
} // End of class
```

```
class DirComp extends FileSysComp {
  Vector dirCnts = new Vector();

  public DirComp(String cName) {
    super(cName);
  }
  public void addComponent(FileSysComp fc)

          throws Exception {
    dirContents.add(fc);
  }
  public FileSysComp getComponent(int l)
      throws Exception {
  return (FileSysComp)dirCnts.elementAt(l);
 }
  public long getComponentSize() {
    long sizeOfAllFiles = 0;
    Enumeration e = dirContents.elements();
    while (e.hasMoreElements()) {
      FileSysComp Comp =

          (FileSysComp) e.nextElement();
      sizeOfAllFiles +=

          (Comp.getComponentSize());
    }
    return sizeOfAllFiles;
  }
```

# Chain of Responsibility (CoR) Pattern

"Avoid coupling the sender of a request to its receiver by giving more than 1 object a chance to handle the request."

"…decouple senders and receivers by giving multiple objects a chance to handle a request. The request gets passed along a chain of objects until 1 of them handles it."

- Each object supports a common interface for receiving a request.
- Each object must forward requests to the next object in the chain.

Use this pattern when more than one object can handle the same request, and you don't know who beforehand.

Issue: what if nobody handles it? What if an object in the chain makes a global assumption?

# Command Pattern

"Encapsulate a request as an object, thereby letting you parameterize clients with different requests…"

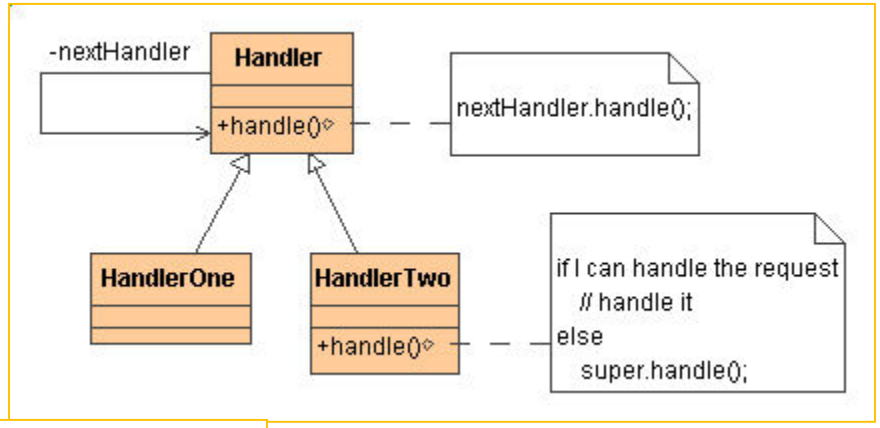
"Commands are an OO replacement for callbacks"

The Command pattern lets you decouple the representation of an action from when and where it actually occurs.

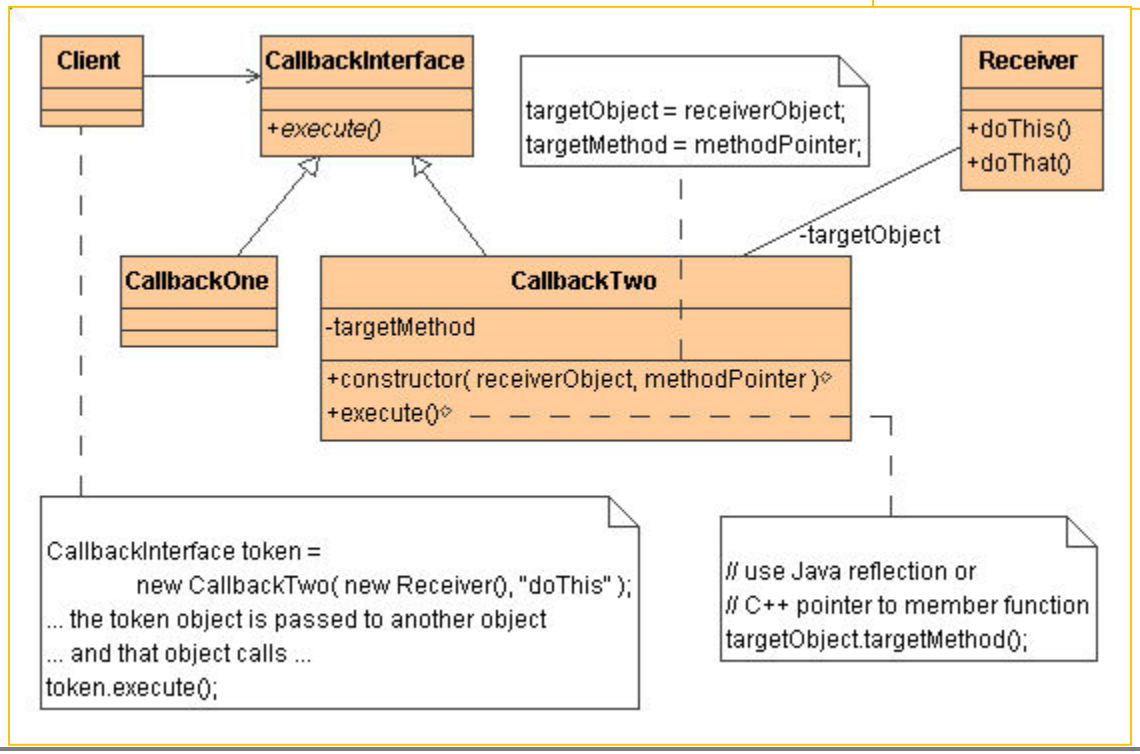
Helps support "undo", logging, and design centered around a few key operations universally applied within a system

# CoR and Command Patterns

Chain of Responsibility:




Command:



# Questions?


# Module 19: UML Overview

(Authored by Kevin Gary, Arizona State University)

Introduction to Assured Software Engineering

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

# Notices



This material is based upon work funded and supported by the Independent Agency under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center sponsored by the United States Department of Defense.

# Topics

What is a model?

UML overview / refresher

UML Activity Diagrams and examples

# Overview of Modeling

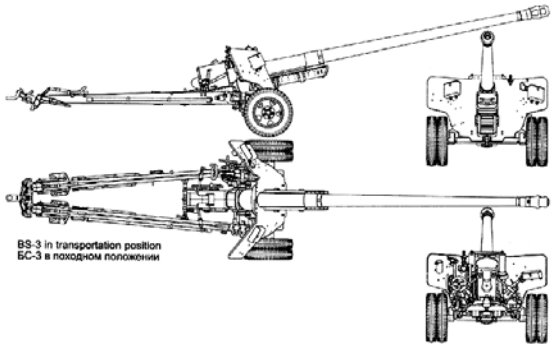
A model is an abstraction of the real world.

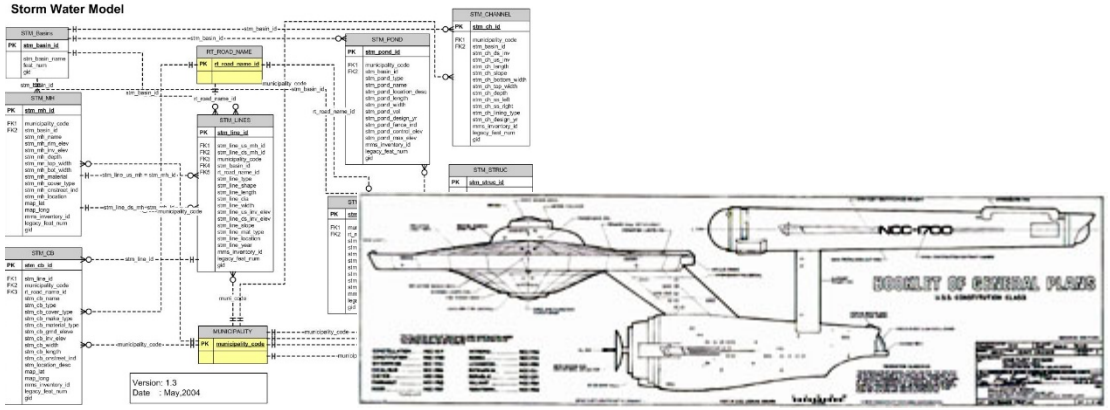
Modeling is a heavily practiced, proven engineering technique.

- architectural modeling of buildings
- mathematical modeling of systems

Models include important details and omit minor aspects.

Why do people model?



# Why Do We Model?

To help us understand *what* a system should do, and *how* it should do it

To communicate our decisions of what and how

To detect and prevent misunderstandings and miscommunications

To generate (portions) of the system's artifacts

To help understand existing systems (reverse engineering)

# UML (Unified Modeling Language)

> The Unified Modeling Language (UML) is a language for specifying, visualizing, constructing, and documenting the artifacts ofsoftware systems, as well as business modeling and other non-software systems"

Standard language for modeling software-intensive systems

Can be used to generate code

- UML 2.0 additions focused on formal semantics for transformations

Notation independent of development process

- UML is not a methodology or process

Numerous tools exist which implement UML

- http://www.objectsbydesign.com/tools/umltools_byCompany.html
- Several tools reverse engineer code into UML

# Why UML?

UML has became the **de facto language** for describing the artifacts of software-intensive systems.

- Several extensions exist for systems engineering (SysML) and others (software engineering, business process modeling).
- Can serve as a standard for the definitions of the concepts of object-orientation (OO)
  - But in practice, there is no such standard.
- Includes recommendations about how to describe systems

**UML is a syntactically and semantically complete language!**

- The visual elements are expressed and related in specific ways (syntax).
  - You can have the equivalent of a compiler error - i.e. you can't just connect things with a line anywhere you like!
- Even if you can connect them, the type of line matters (semantics).
  - If it is dashed or not, uses a hollow or thin arrow, etc.

# UML Diagrams Support Multiple System Views

Use case view (Use Cases, Activity)

- behavior of the system as seen by the end users

Design view (Class, Object)

- both static and dynamic view of classes and objects; their relationships and their interaction

Process view (Timing, Interaction, Statechart)

- illustrates concurrency and synchronization issues

Implementation view (Component, Package)

- configuration management issues about assembly and release of code files

Deployment view (Deployment)

- distribution of parts among hardware elements

# Categories of UML Diagrams

UML 2.0 has (at least) 18 different diagrams.

A few different ways to think about them (in addition to previous slide):

- *Descriptor vs. Instance*
  - Descriptor diagrams describe the system's general case, instance show system snapshot
    - Descriptor diagrams - class, use case, deployment, component
    - Instance diagrams– object, sequence, static structure, deployment, component
- *Behavior vs. Structure*
  - Need both behavior and structure to communicate
    - Behavior – statechart, interaction, timing, use case
    - Structure – class, component, deployment, block (SysML)
    - Flow might be another form here – activity, sequence, timing

# UML Activity Diagrams

# What Is an Activity Diagram?

An activity diagram in the use-case model can be used to capture the activities and actions performed in a use case.

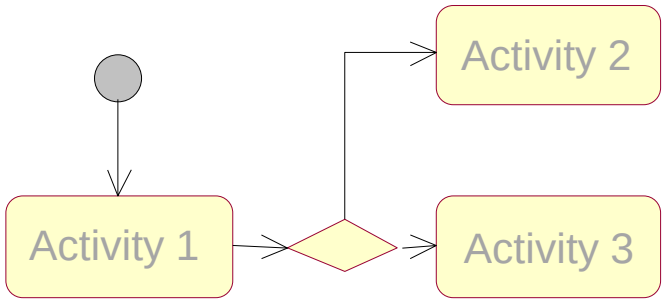
It is essentially a flow chart, showing flow of control from one activity or action to another.

*Flow of Events*

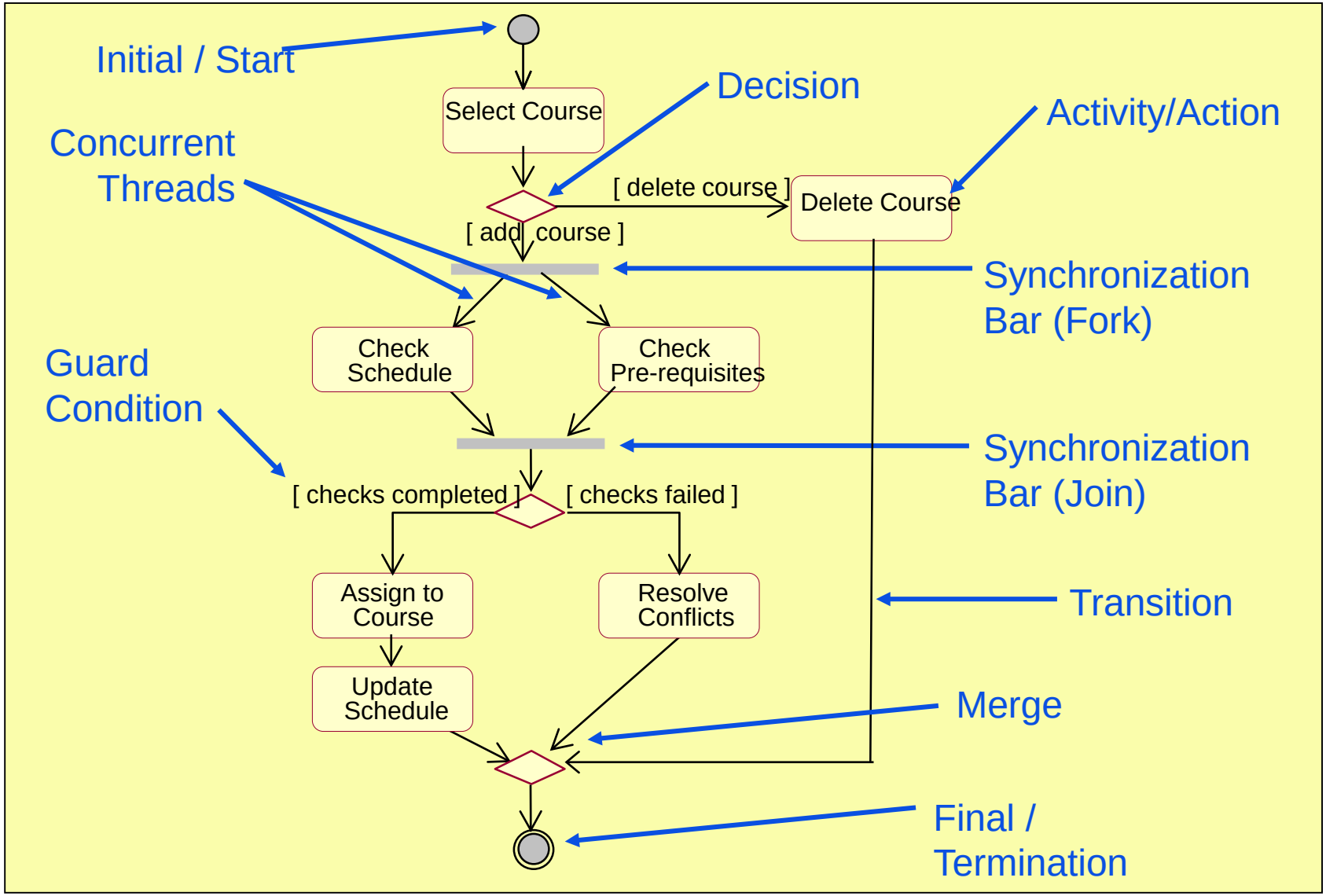
This use case starts when the Registrar requests that the system close registration.

1. The system checks to see if registration is in progress. If it is, then a message is displayed to the Registrar and the use case terminates. The Close Registration processing cannot be performed if registration is in progress.

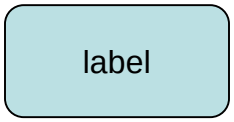
2. For each course offering, the system checks if a professor has signed up to teach the course offering and at least three students have registered. If so, the system commits the course offering for each schedule that contains it.



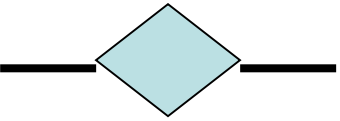
# Example: Activity Diagram



# Activity Diagrams








Summary of Notation:

- **Action/Activity state** – Action states cannot be decomposed, Activity states may be (UML 1.5 - as of UML 2.0 replaced with Activity *frames)*
- **Transition** – control flow; a transition is triggered upon completion of some activity
- **Decision/Merge point** – standard if-else style logic; also supports iteration. Guard conditions indicated in brackets in each transition.
- **Object node** – may be (typically not) included to show where an object's state may change.
- **Synchronization bar** – supports fork/join semantics for concurrent processing
- **Swimlanes** – partition by responsibility, not thread

# Activity Diagram

**Action**
A step in the flow of events

**Decision**
Flows split based on a guard condition

**Fork**
Beginning of concurrent flows

**Join**
End of concurrent flow

**Flow**
Show the sequence of activities

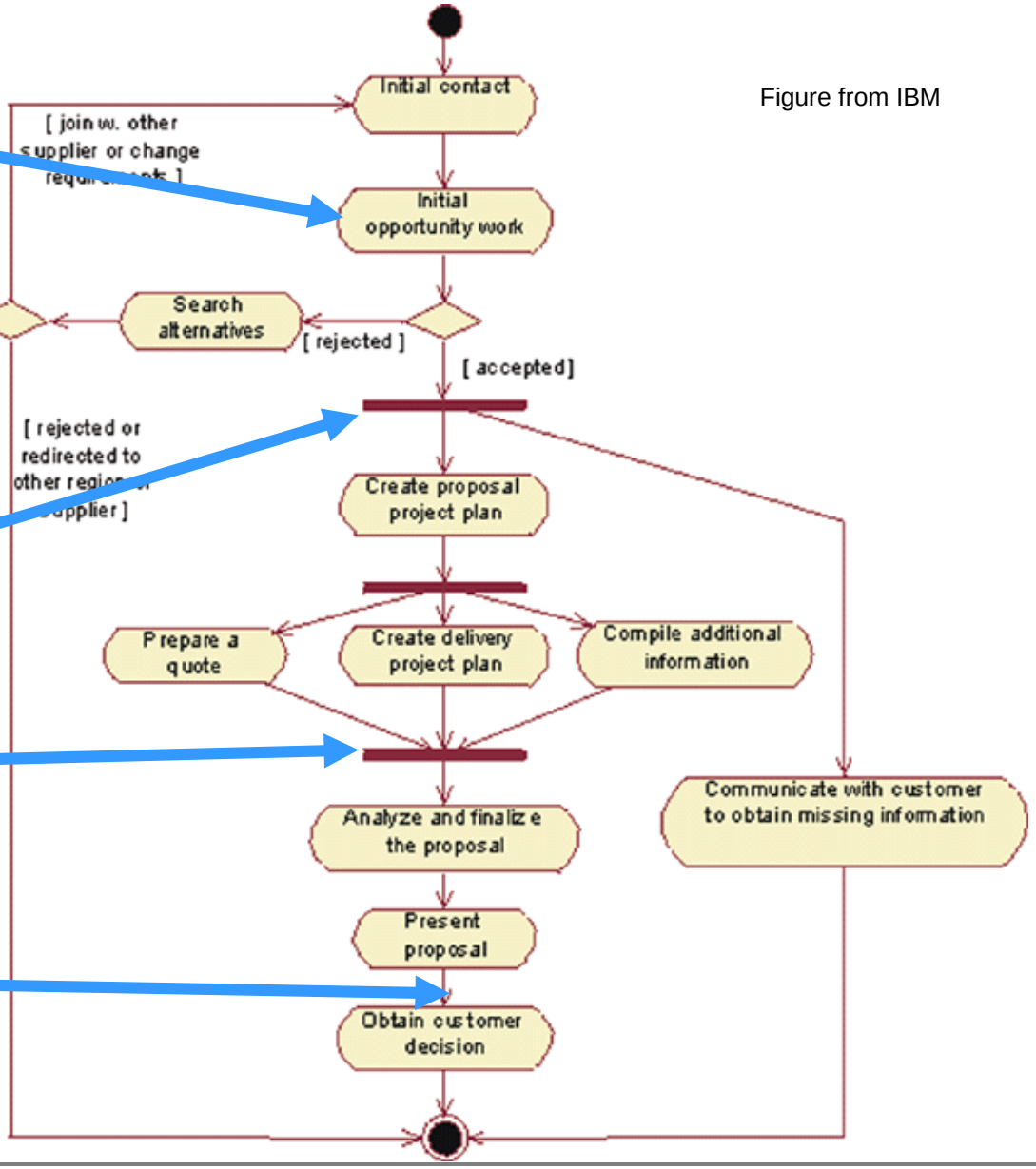



Figure from IBM

# Mapping Who Does What to Whom

Examples so far show us what actions happen.

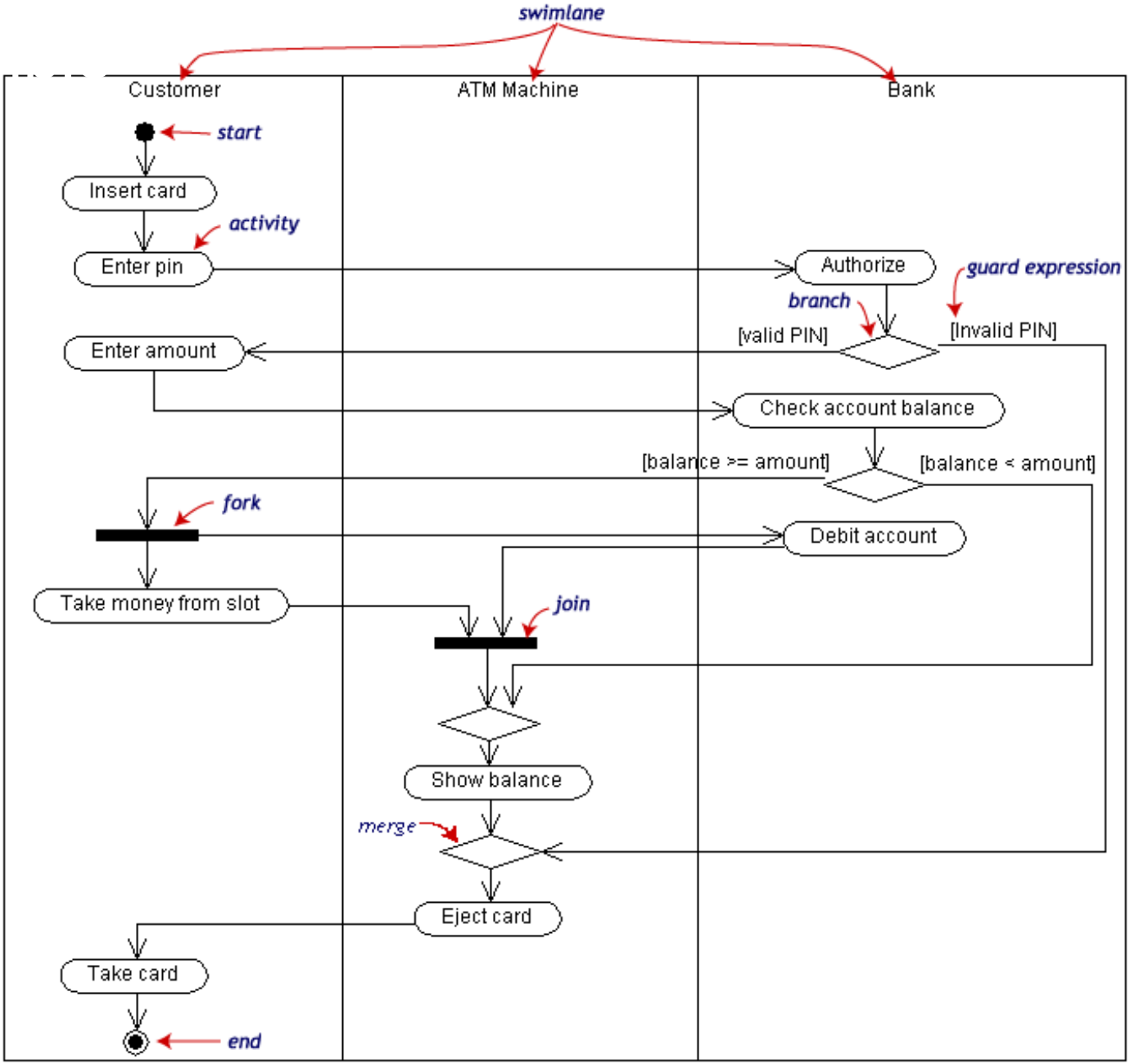
- But “WHO” does each action and “WHEN”?

*Swimlanes*

- Partition activities according to who does them.
- Who can be actors, system components, whatever.
- When is indicated top-to-bottom (like a sequence diagram) or left-to-right.

To Whom?

- Activity diagrams can show relationships to objects that are affected by actions.




Example by
Bau Yoon Teck

# How do you derive Swimlanes?

Swimlanes add an analysis step.

- You are assigning a **responsibility** to an actor.
- Note, we did not say to an object - to an actor.

How to do?

- Technique: Return to your scenarios
  - Outline format the steps based on who does them

1. PA selects “Check Out”
2. System renders summary
3. PA enters credit card
4. PA selects “Submit”
5. System asks 3rd party verify
6. 3rd party verifies
7. System confirms for PA

1. PA selects “Check Out”
    2. System renders summary
3. PA enters credit card
4. PA selects “Submit”
    5. System asks 3rd party verify
        6. 3rd party verifies
    7. System confirms for PA

# What Role Do Objects Play?

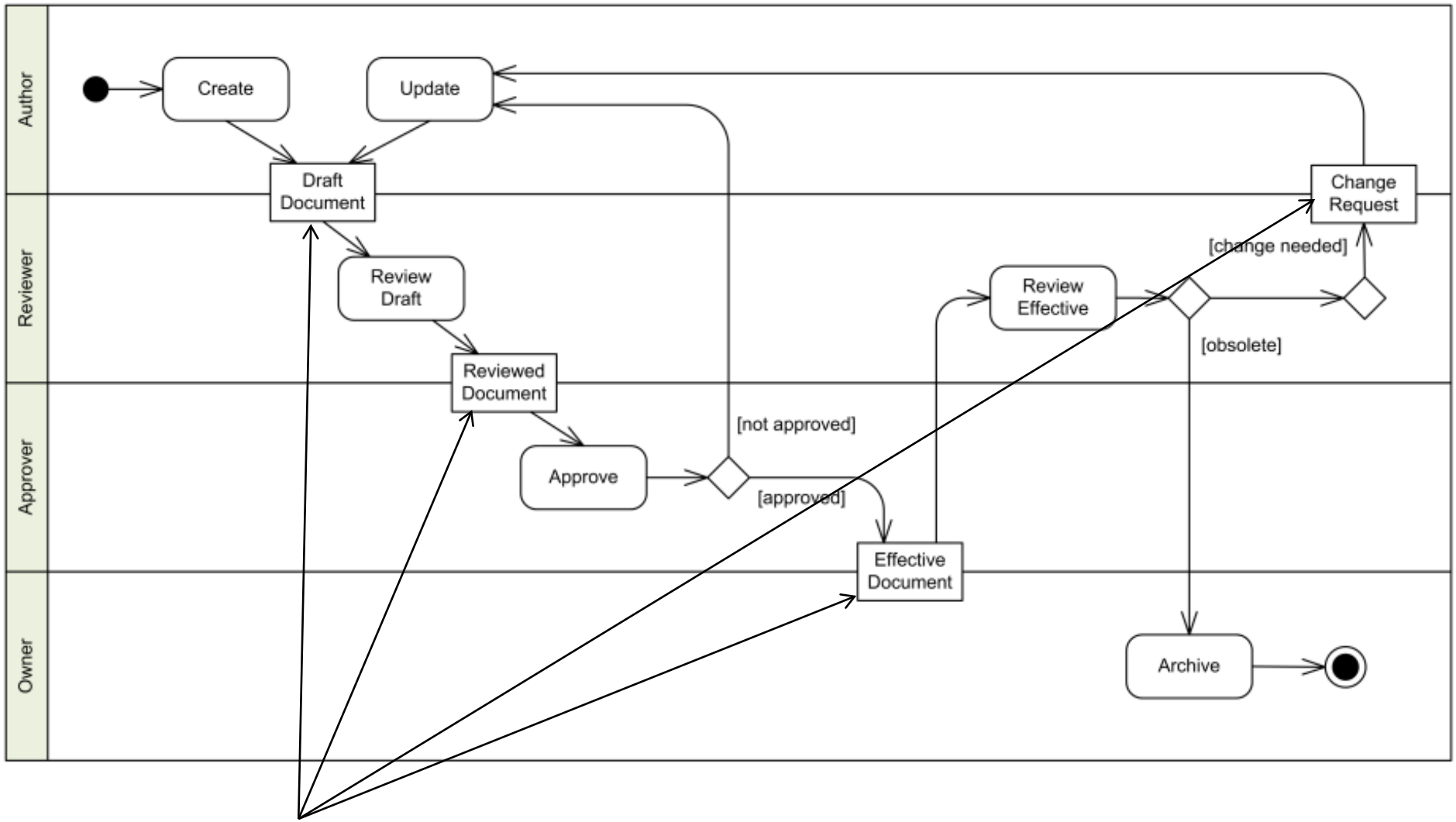
We think of UML as OO, but rarely do we see an Object on an Activity diagram.

- Activity diagrams are for *flow* modeling.
- Object behaviors are typically modeled using an UML *Statechart.*

But yes objects can be shown:

- Do if it is important to show critical object state changes or dataflow in the activity diagram context!

# Example: Activity Diagram w/ Objects




Objects! (should really use [] for state)

Example from http://www.uml-diagrams.org/activity-diagrams-examples.html

# Activity Diagram Summary

Pros

- Map use case scenarios directly on to actions
- Most intuitive for most procedural programmers
- Includes constructs concurrency and task assignment
- Includes constructs for top down decomposition (activity frames)

Cons

- Some confusion of the relationship between activity diagrams and statecharts
- Some changes in terminology from 1.5 to 2.0
- Relatively poor tool support

Recommendation: Useful early in analysis, after use cases but before interaction diagrams

# Questions?


# Module 20: Behavioral Modeling Using UML State Machines

(Authored by Kevin Gary, Arizona State University)

Introduction to Assured Software Engineering

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

# Notices



This material is based upon work funded and supported by the Independent Agency under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center sponsored by the United States Department of Defense.

# Topics

Behavioral Modeling

State Machines

UML Statecharts

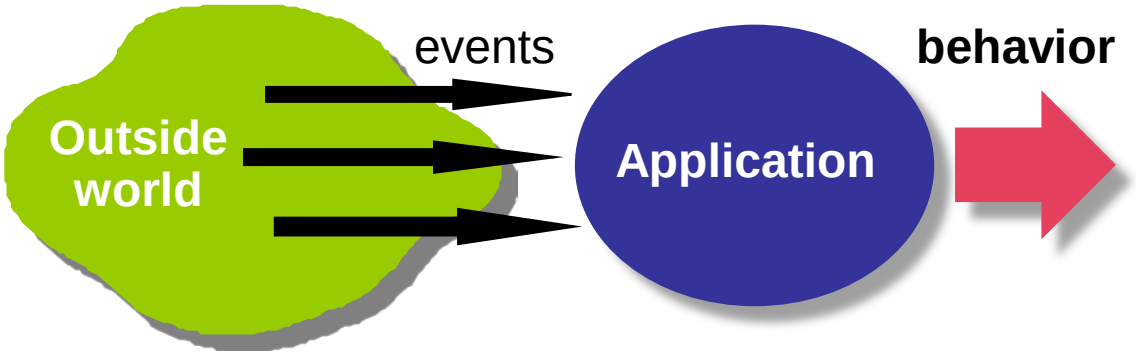
Summary

# Behavorial Modeling

Domain Models capture the object vocabulary of the problem space.

These models describe objects, relationships, and (some) interactions independent of a specific problem of solution.

- Examples: data dictionaries, glossaries, statecharts




Behavioral Models capture the observable behaviors of a system.

- Capture how the system reacts to external stimuli
- Examples: Stimulus-Response models, Event-oriented decomposition, Statecharts

Domain vs. Behavioral Models:

- Domain models are usually problem independent.
- Behavior models can describe a domain yet also introduce problem-specific constructs.

# In-depth: The need for SMs

Definition of an object:

- *An object has state, behavior, and a unique identity.*
- Object models are expressed using attributes, behaviors, & messages.
  - An object has one or more attributes.
  - An object exhibits one or more behaviors.
  - Objects communicate via messages.

**We need a way to capture dynamic behaviors!**

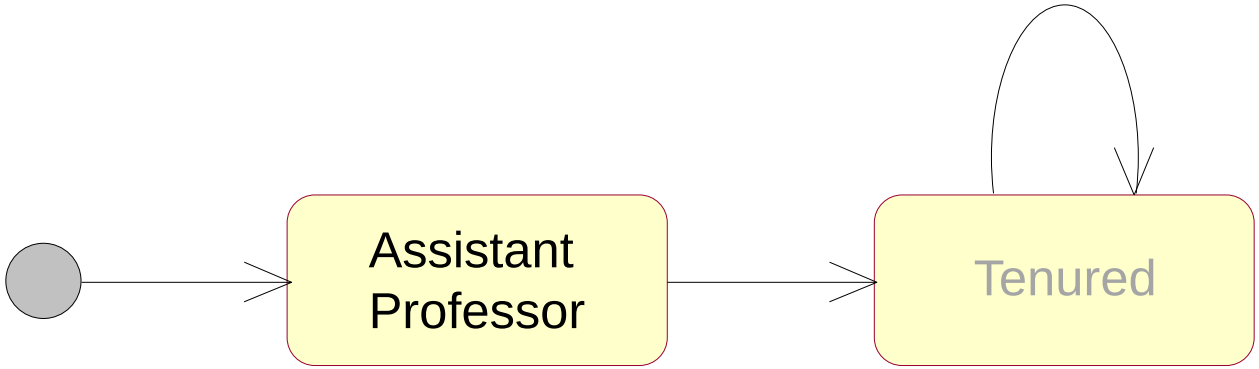
- The way in which an object's state changes over time
  - The object may change (transition) state for many different reasons, which we will model using events.
  - One reason might be that a *message* arrives from another object!

# What Are State Machine Diagrams?

A state machine diagram models dynamic behavior.

It specifies the sequence of states in which an object can exist:

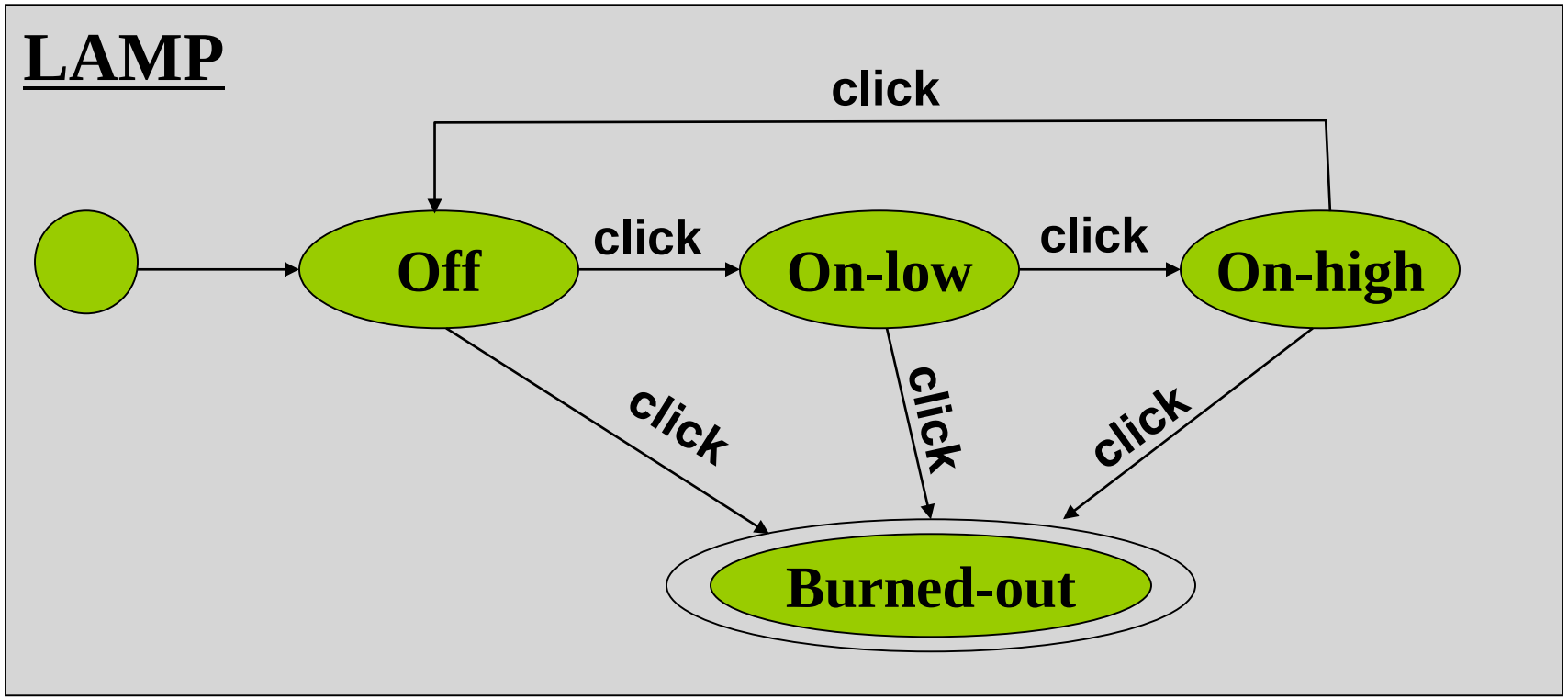
- The events and conditions that cause the object to reach those states
- The actions that take place when those states are reached



# State Machine Example

Example

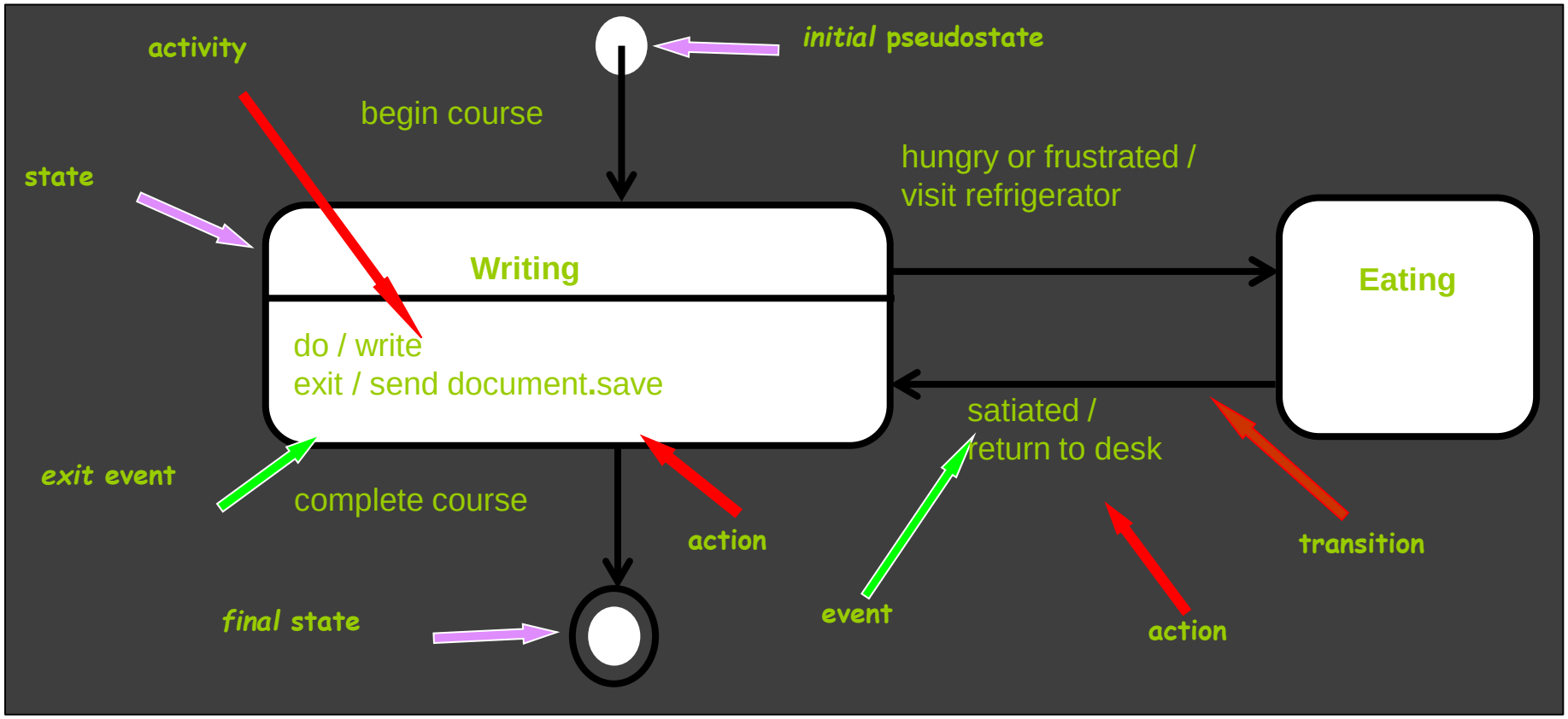
- Consider a lamp with two separate brightness settings:



# UML Statechart 1-slide cheatsheet

Previous diagram was *not* UML!

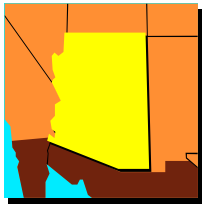
*Statechart syntax*:



# State Machine Overview

Three elements of a statechart diagram

- *States*:
  - Condition or situation during the life of an object during which it satisfies some condition, performs some activity, or waits for some event.
  - Some states are “special”.



- *Events*:
  - Internal and external occurrences that impact or are generated by an object.
- *Transitions*:
  - Response of an object based on events and present state
  - May have *Guards* or *Activities* associated with them
  - On such a change of state, the transition is said to *fire*

# Anatomy a State

A State can have several parts:

- Name
- Entry/exit actions
  - Entry action: **entry** / *action*
  - Exit action: **exit** / *action*
- Internal transitions
  - These bypass the entry/exit actions and guard conditions.
- Substate, deferred events, & other things we won't use

When in a given state, an object may be *active.*

- *Active* means it is doing something, some *action*.
- That action is interruptable, or may run to completion.
  - It may generate a *completion event* which results in a transition.
  - If the state exits via some other event, then the action is terminated.

# Special States

The initial state is entered when an object is created.

- Exactly one initial state is permitted (mandatory).
- The initial state is represented as a solid circle.

A final state indicates the end of life for an object.

- A final state is optional.
- A final state is indicated by a bull's eye.
- More than one final state may exist.

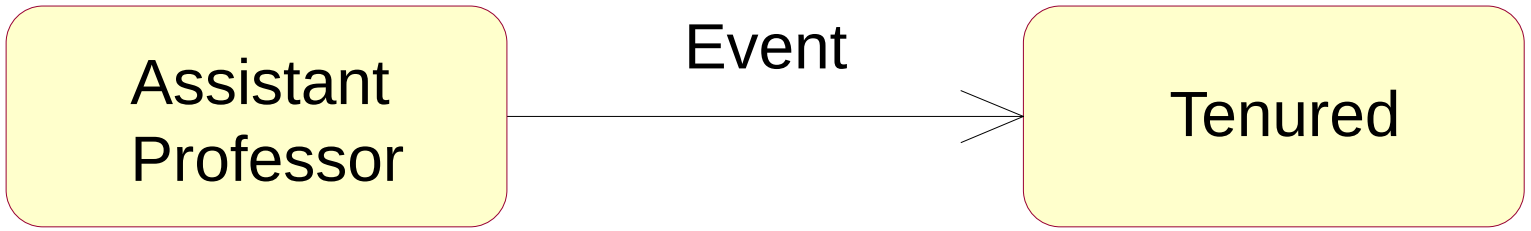


Other special (pseudo) states exist in UML, but are not relevant to what we will do with Statecharts.

# What Are Events?

The specification of a significant occurrence that has a location in time and space

- An event is an occurrence of a stimulus that can trigger a state transition.
- Example:
  - Successful publication of numerous papers



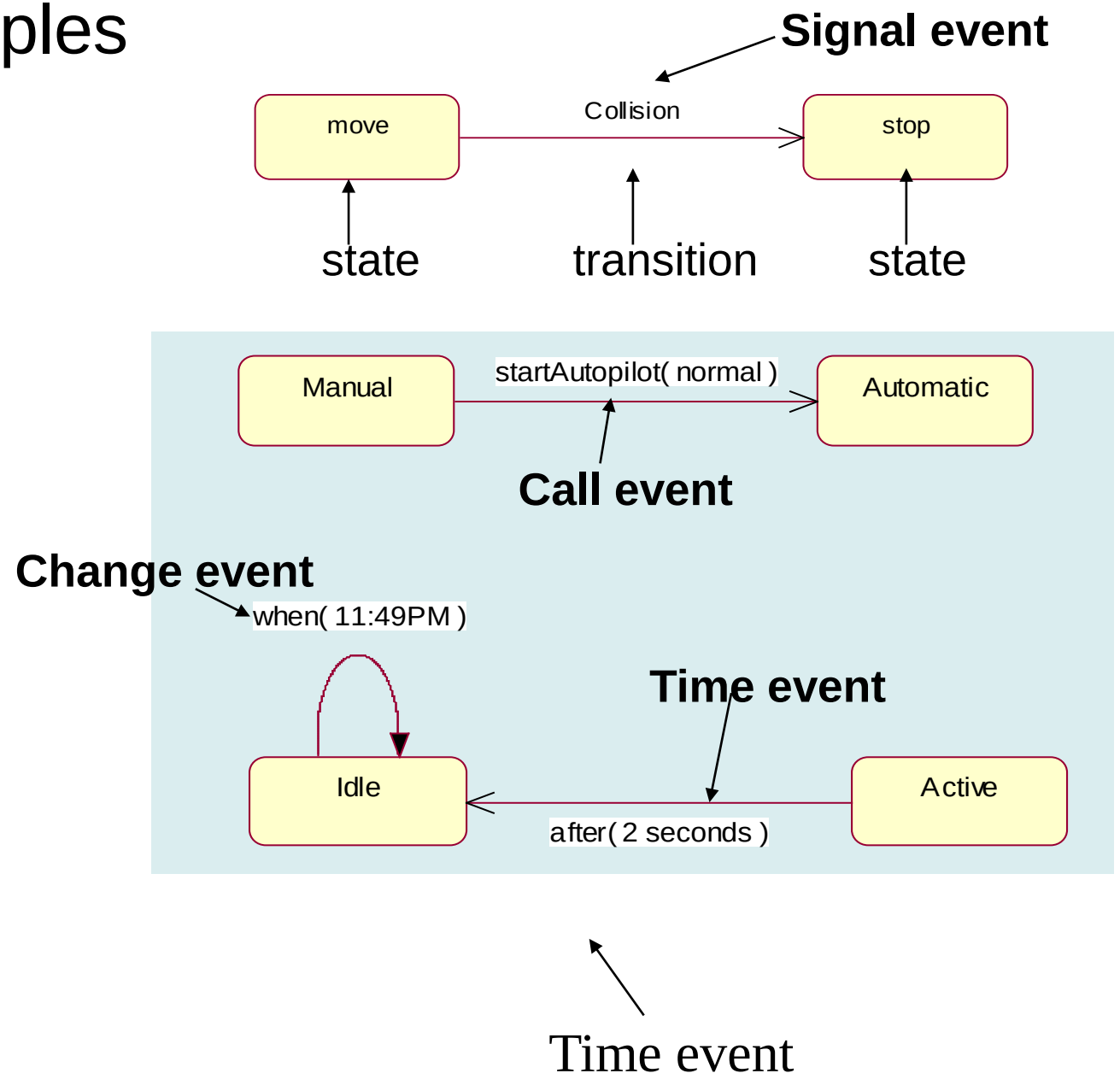
# UML Event Types

UML defines 4 event types:

- signal event: sname(p:T )
  - Receipt of an explicit, named, asynch communication among objects
- call event: op(p:T )
  - Receipt of an explicit synchronous request among objects that wait for a response
- change event: when (exp)
  - A change in value of a Boolean expression (continuous check)
- time event: after (time)
  - Arrival of an absolute time or passage of a relative amount of time

Most common type of event is the signal event.

# Event Examples



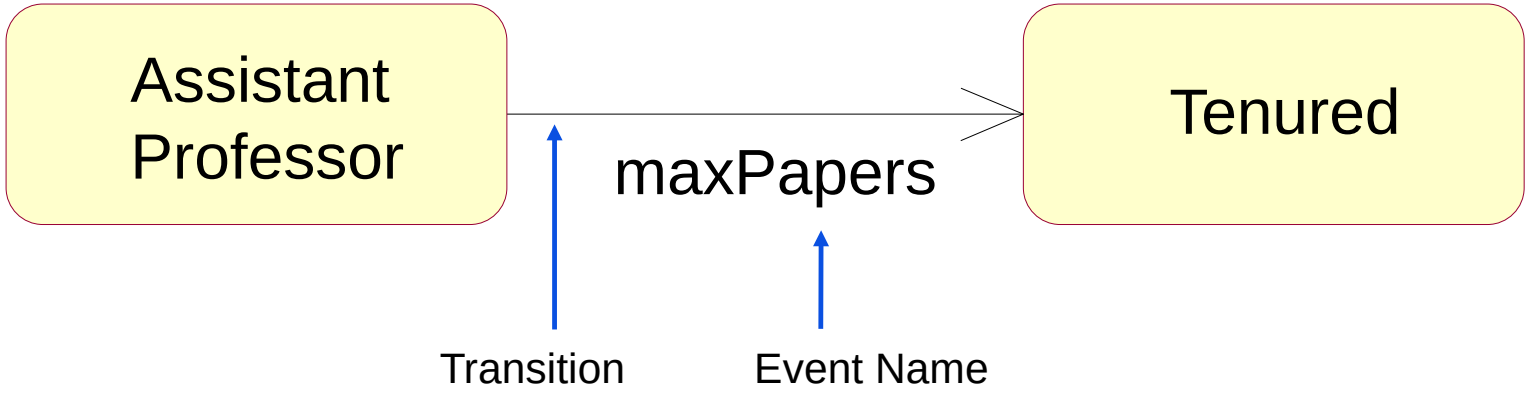
# What Are Transitions?

A *transition* is a change from an originating state to a successor state as a result of some stimulus.

- The successor state could be the originating state.
- Transitions typically take place in response to an event.
- Transitions may take place when an object completes an *activity*.

Transitions can be labeled with event names.

- But keep in mind that events and transitions are not the same thing!
  - Event: something happened, or changed, or was communicated in the world
  - Transition: specific to an object, indicates a change in its state



# What Are Guards/Actions/Activities?

Transition: *event* [ *guard* ] / *activity*

A *Guard* is a boolean condition that may optionally be present on a transition.

- If the condition if false, no transition.

An *Activity* is an optional behavior that is executed during a transition.

*Note: UML 1.x used the term "action" for a behavior on a transition, and used "activity" for ongoing tasks.*

- Activity (within a state) could be interrupted
- Actions (on a transition) could not (run-to-completion semantics)

# Internal versus External Transitions

UML also defines some special Activities:

- An *entry activity* executes whenever you enter a state.
- An *exit activity* executes whenever you leave a state.
- An *internal activity* is one where a SM can react to an event without leaving the state.
  - Note, this is slightly different than a self-transition in that no entry/exit activities are executed.

**InReverse**

**onEntry:** *"beep"*
**onExit:** *"beep off"*
**Internal transition:** put in reverse

# How All This Works

1. An instance begins life in the state pointed to by the initial pseudostate.
2. Entry actions are executed every time the state is entered.
3. Exit actions are executed every time the state is exited.
4. The guard condition is evaluated after the event instance occurs.
5. If an event instance matches a transition label, that transition fires if its guard condition allows.
6. If an activity is running when a trigging event instance occurs, the activity is terminated.
7. The completion of all activity in a state is considered an event—a completion event.
8. A transition without an explicit event label is triggered by the completion event instance if its guard condition allows.
9. Event instances that don't match some transition label are ignored and lost.

*Note the synergy with Activity diagrams; Statecharts use many of the same concepts and symbols, but turn an Activity diagram "inside out" to view interactions from the object's perspective!*

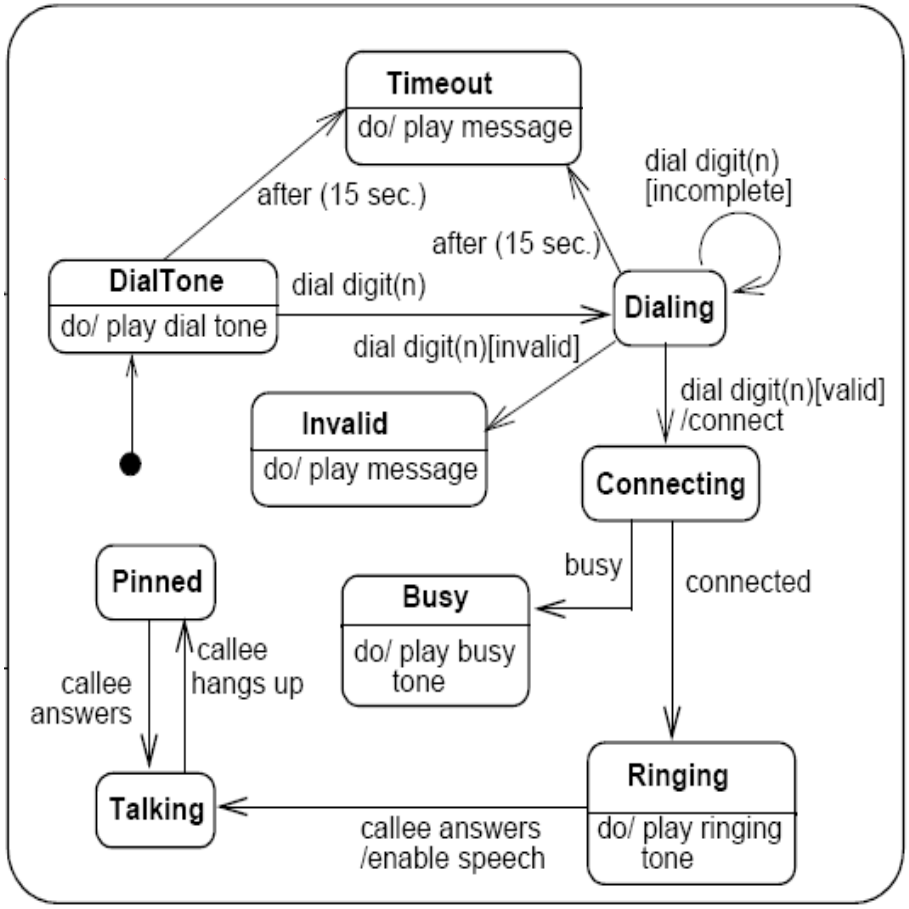
# Statechart Example 1

How many states:

- Are final states?
- Have an activity?
- Have an entry or exit activity?
- Represent exceptional circumstances?

How many transitions:

- Are self-transitions?
- Are internal transitions?
- Have guard conditions?
- Have an action?
- Are signal events? Call? change? Time events?



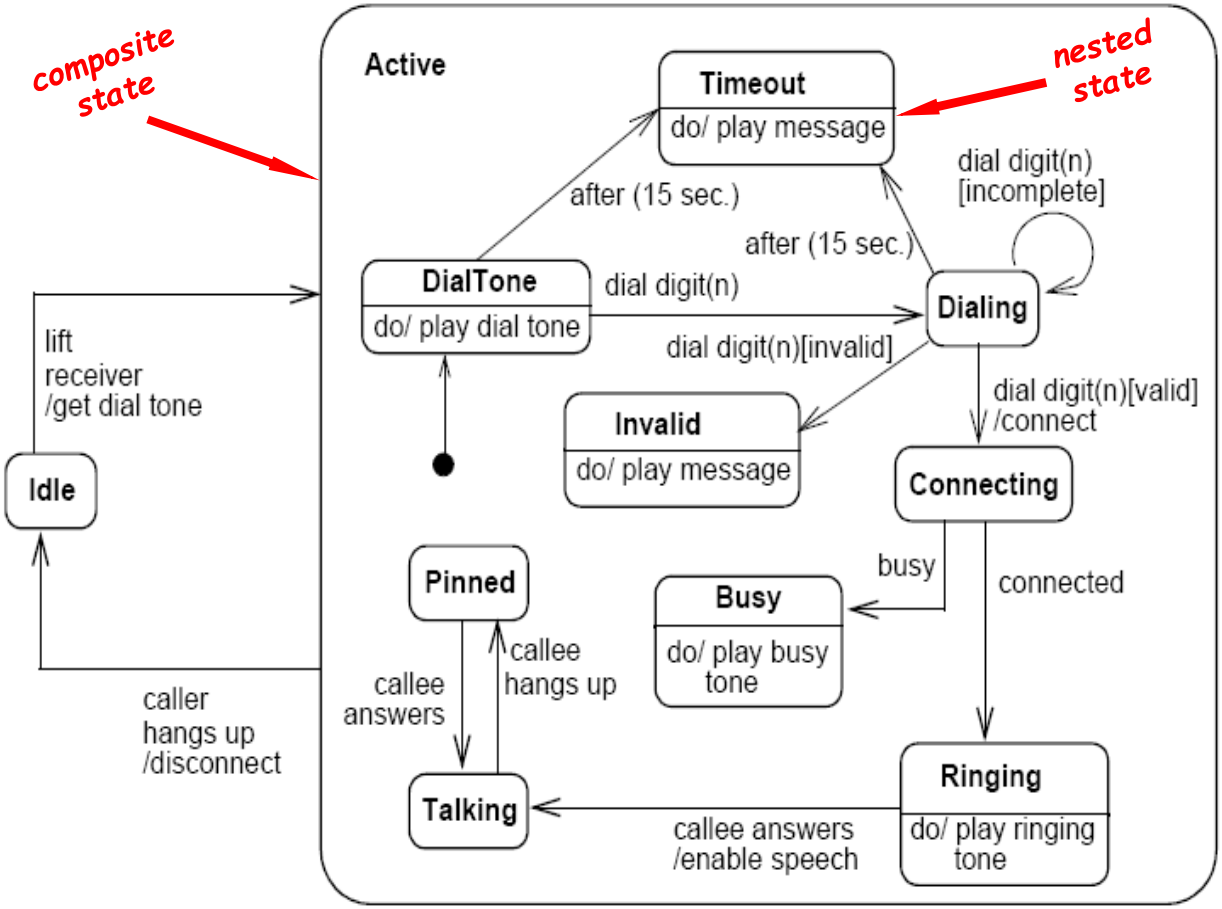

*Anything a little strange in this model to you?*

# Statechart Example 1 “strangeness”

There were nodes that weren’t final states but had no outgoing transitions!

- *Invalid, Busy, (Pinned,Talking)*
- *This is not illegal according to UML!*

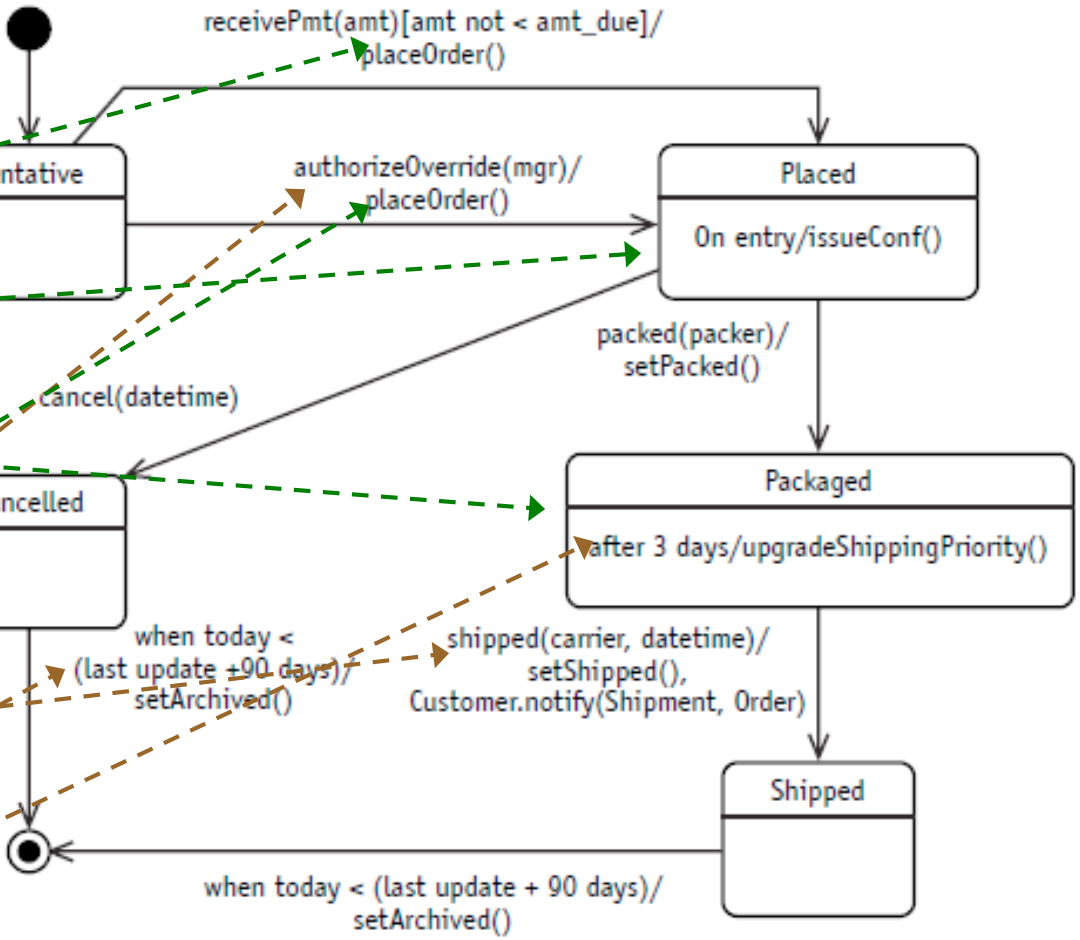
But it is strange; what we want is a blanket condition that says whenever callee hangs up we should disconnect on our end.



# Statechart Example 2

Note:

- Multiple transitions
- Guard conditions
- On entry
- Internal transition
- Actions on transitions
- Events
  - Signal
  - Call
  - Change
  - Timer




Statechart taken from http://blog.jstoutenburg.com/2010/12/uml-statechart-diagram.html

# More SM Concepts

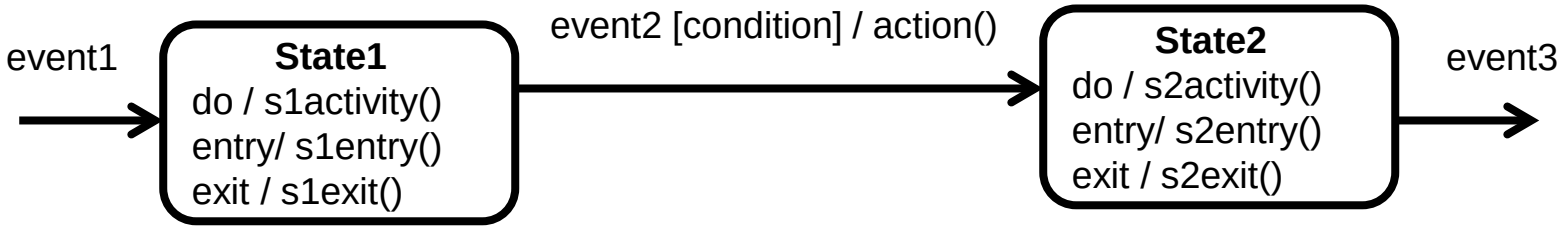
Exceptional conditions

- Do you have to specify transitions for every state that correspond to every single possible (or impossible) thing that could ever happen in the Universe?

Example:

- "Your Bank Account standing goes from Good to Bad if a bunch of 3-legged Martians riding flying camels descend from the sky and deep-fry your bank."
- How would you specify a transition for this? You wouldn't!
- However, you can characterize all "unexpected" states and transitions as "exceptional".
  - All transitions you do not specify as valid are assumed invalid.
  - You have three options:
    - test for, and disallow, the transition
    - do not test for, and risk, an invalid transition
    - throw an exception - "I've reached an unexpected state"

# Statechart Summary




1. The event1 causes s2entry() to execute and s1activity() to start.
2. When event2 occurs
    1. If [condition] is false the object does nothing, otherwise we continue
    2. s1activity() is aborted if running
    3. s1exit() action, transition action(), and s2entry() execute sequentially
    4. s2activity() begins
3. If event3 occurs while processing the sequence above, it is queued until after all the actions (s1exit(), action(), s2entry())
    - Run to completion semantics requires the object completely arrive in a state before starting the transition to another state
    - Ensures objects are “thread-safe” and cannot be corrupted by asynchronous events

# Statechart Tips

Think about the set of different states that object can be in at any given time – name them!

Think about how many final states the object may have.

- Maybe it is zero – an object that “lasts forever”.

Think about how you get from one state to another.

- This is the “how” part of an object - the dynamic behavior.
- It is important to capture all of the transitions!
- There can be more than 1 way to get from one state to another!
- A transition may go from one state back to itself!

Think about what actions occur as “side-effects”, and when!

Are there expected/unexpected “weird” situations?

Importance for your project:

1. You start modeling a domain by identifying and describing objects.
2. Statecharts describe object state and behaviors.
3. Nice jumping off point from Activity diagrams, syntactically & semantically.

# Questions?


# Module 21: Inspections

(Developed by Mel Rosso-Llopart and Anthony Lattanze)

Introduction to Assured Software Engineering

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

# Notices



This material is based upon work funded and supported by the Independent Agency under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center sponsored by the United States Department of Defense.

# Outline

Software work product inspection

Benefits of work product inspections

The cost of inspections

The formal work product inspection process

# Software Work Product Inspections -1

A process where a software work product is reviewed by a group of peers

- documents, designs, code

Two general types

- more formal  inspections (formal)
- less formal inspections (informal)

In either case, the work product is always reviewed by a group of peers.

# Software Work Product Inspections -2

There are formal and informal inspection methods.

Formal inspections feature

- highly structured process for preparation, meeting protocol, data collection, and post meeting activities
- table-top meeting oriented
- excellent for ensuring final revision product quality and standards

# Software Work Product Inspections -3

Informal inspections feature

- less structure, formality, and rigidity
- more open discussion (peer review)
- table-top or presentation oriented
- excellent for initial artifact presentation, artifact discussion, or selection of alternatives

Both formal and informal inspections have their place in a project.

# Benefits of Inspections -1

The benefit of the inspection process is threefold:

- ensure that work products are of the highest possible quality
- ensure that software work products meets organizational and legal standards
- standard products are more maintainable than those that do not follow conventions for coding, commenting, and documentation

# Benefits of Inspections -2

Provides visibility into the products and services that each engineer provides individually

Facilitates detailed technical communication which helps:

- foster reuse
- acts as a training vehicle
- provides fresh insight into technical problems

# Benefits of Inspections -3

The earlier a defect is found in the software development process, the less it costs to repair.

The longer a defect goes undetected the more it will cost to repair. WHY?

Watts Humphrey, *A Discipline for Software Engineering*,

Addison Wesley, 1995.

# The Cost of Inspections -1

The dollar cost of finding a defect during system test vs. finding the defect in design is 100:1.

- The time cost is 200:1.

Jet Propulsion Lab (JPL) reported the cost of finding a defect in an inspection is $100, in a test $10,000.

On average, design and code reviews **reduce the cost of testing** by 50-80% including the cost of the reviews

# The Cost of Inspections -2

Exact cost will vary with organization.

Rule of thumb for code inspections:

- ≈3 person hours per 100 lines of source code written

Cost for design or requirements inspections varies widely.

# Formal Inspection Process Roles

Producer: creator of the artifact to be inspected

Reviewers: will review the document; there should be at least 3 reviewers for formal inspections

Moderator: keeps the inspection meeting focused, can also be a reviewer

Time Keeper: can also be a reviewer, producer, or moderator

# Formal Inspection Process -1

Pre-Inspection

- set expectations
- schedule inspection and provide agenda 48 hours prior to the meeting
  - recommendation: limit meeting to 2 hours
  - plan on at least 3 reviewers (including producer)
  - provide supporting documents
  - reviewers ***must*** review artifact prior the inspection meeting

# Formal Inspection Process -2

Inspection Meeting for Documents

- moderator will lead inspection
- paragraphs, or major sections should be called out

Inspection Meeting for Code

- moderator will lead inspection
  - line numbers can be called out
  - code can be read directly or paraphrased

# Formal Inspection Process -3

Agree on reading style before inspection meeting.

Reviewers make comments in turn.

Producer will record each issue.

- may formally address each issue after the inspection meeting
  - not all issues will be defects

# Formal Inspection Process – Hints for Success

Keep meeting within time constraints

- Various studies have shown that yield declines after 2 hours or too many Line of code reviewed per hour.

Moderator will keep discussion on track

- Don’t solve problems during the inspection meeting.
  - raise issues
  - record issues
  - address them later

# Guidelines for Informal Inspections

Stay focused and maintain time constraints.

- avoid degenerating into a freeform discussion
- set expectations
- record relevant discussion such as
  - decisions
  - issues
  - points of contention
  - action items

# Summary

Understand

- what software work product inspections are
- the cost of software work product inspections
- the benefits of software work product inspections

Know basically how to conduct software work product inspections.

# Discussion

Did you have occasion to do formal inspections on projects?

What about peer reviews?

What about pair programming in Agile?  Any experience with it?

# References

Hoover, C., Rosso-Llopart, M., Taran, G. *Evaluating Project Decisions: case studies in software engineering*. Boston, MA: Addison Wesley, 2009.

Pressman, R. *Software Engineering: A Practitioner's Approach*. McGraw-Hill Science/Engineering/Math; 7 edition (January 20, 2009).

# Questions?


# Module 22: System Testing

(Developed by Eduardo Miranda)

Introduction to Assured Software Engineering

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

# Notices



This material is based upon work funded and supported by the Independent Agency under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center sponsored by the United States Department of Defense.

# Topics

Purpose of software verification

Verification methods

Inspections

Testing

# Learning Objectives

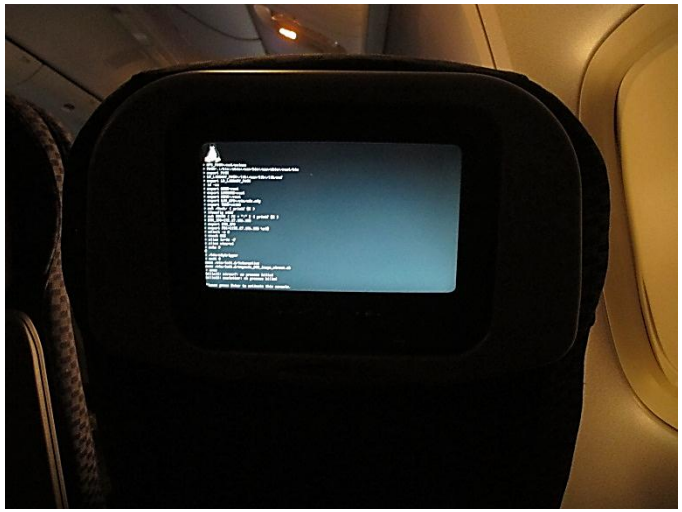
Understand the importance of software verification

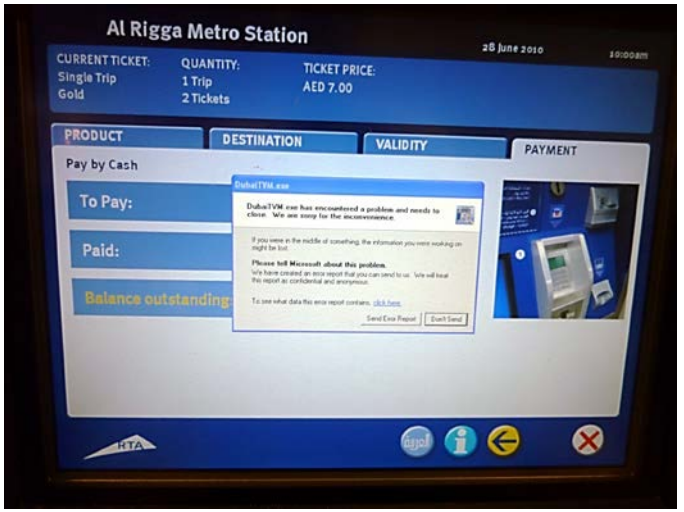
Recognize the strengths and weaknesses of different verification approaches

Become acquainted with the concept of coverage

Acquire a basic knowledge of test case design techniques

# But just remember there's a lot of bad and beware “Oh, baby, baby, it's a wild world”







- In-flight entertainment system rebooting, Delta, 2011
- Ticket machine, Dubai Metro 2010
- Electronic billboard crash, Panama City, 2011

# Purpose of Software Verification

Finding faults before the software is released to its users

Justify confidence in the program by demonstrating that

- It does what it is supposed to do under its stated conditions
- It doesn't do what it is not supposed to do under adverse conditions

# Quality Characteristics & Verification Techniques

## Techniques

- Functionality
- Reliability
- Usability
- Efficiency
- Maintainability
- Portability

## Quality characteristics

- Inspection
- Testing
- Analysis
- Demonstration

# Verifying Quality Characteristics: Inspection

International Council on System Engineering (INCOSE)

- The verification method of determining performance by examining (a) engineering documentation produced during development or modification or (b) the item itself using visual means or simple measurements not requiring precision measurement equipment

Software practitioner

- The scrutiny by people other than the producer, of human oriented development artifacts with the aim of meeting contractual obligations, finding non-compliances with standards or uncovering defects based on the premise that individuals might be blind to some of the trouble spots in their own work and in consequence it is beneficial to have someone else look at it

Uses

- Complement  testing. Come earlier in the process. Germane to the verification of faults of omission, design problems, style issues

Examples

- Is the software maintainable?
- The review of code to find if it is properly commented and styled

2011 (c) Eduardo Miranda

# Verifying Quality Characteristics: Testing

INCOSE

- The verification method of determining performance by exercising or operating the system or item using instrumentation or special test equipment that is not an integral part of the item being verified. Any analysis of the data recorded in the test and that is needed to verify compliance (such as the application of instrument calibration data) does not require interpretation or interpolation/extrapolation of the test data.

Software practitioner

- The, more or less, thorough execution of the software with the purpose of finding bugs before the software is released for use and to establish that the software performs as expected

Uses

- Verification of functional and performance requirements

Examples

- When provided with the correct user name and password the user is able to login into the system
- The software is capable of handling a load of a thousand transactions per minute

# Verifying Quality Characteristics: Analysis

INCOSE

- The performance and assessment of calculations (including modeling and simulation) to evaluate requirements or design approaches or compare alternatives.
- The verification method of determining performance (a) by examination of the baseline, (b) by performing calculations based on the baseline and assessing the results, (c) by extrapolating or interpolating empirical data of collected using physical items prepared according to the baseline, or (d) by a combination of all of the above.

Software practitioner

- The verification of software properties through the use of behavioral or structural information from the software, e.g. the state space of a program, its patterns of execution, an abstract model, etc.; in contrast to the computed values used in testing
- There are two types of software analysis: Dynamic and static

Uses

- Verifies non-local consistency. Path checking. Non deterministic choices such as race conditions

Examples

- Security vulnerabilities, memory leaks, non-compliances
- Resource usage

2011 (c) Eduardo Miranda

# Verifying Quality Characteristics: Demonstration

INCOSE

- The verification method of determining performance by exercising or operating the item in which instrumentation or special test equipment is not required beyond that inherent to the item and all data required for verification is obtained by observing operation of the item.

Practitioner

- Demonstration is the actual operation of an item to provide evidence that it accomplishes the required functions under specific scenarios

Uses

- Mostly user acceptance and obtaining feedback through the development process

Examples

- You walk the user through the different usage scenarios and verify that the different screens are shown in a display
- You power on the equipment and observe whether a light comes on or not

# Relevance

Inspections

Testing

# Inspections

The scrutiny by people other than the producer, of human oriented development artifacts with the aim of meeting contractual obligations, finding non-compliances with standards or uncovering defects based on the premise that individuals might be blind to some of the trouble spots in their own work and in consequence it is beneficial to have someone else look at it

# Benefits of Inspections

Inspections reduce the number of defects in the software throughout the development process.

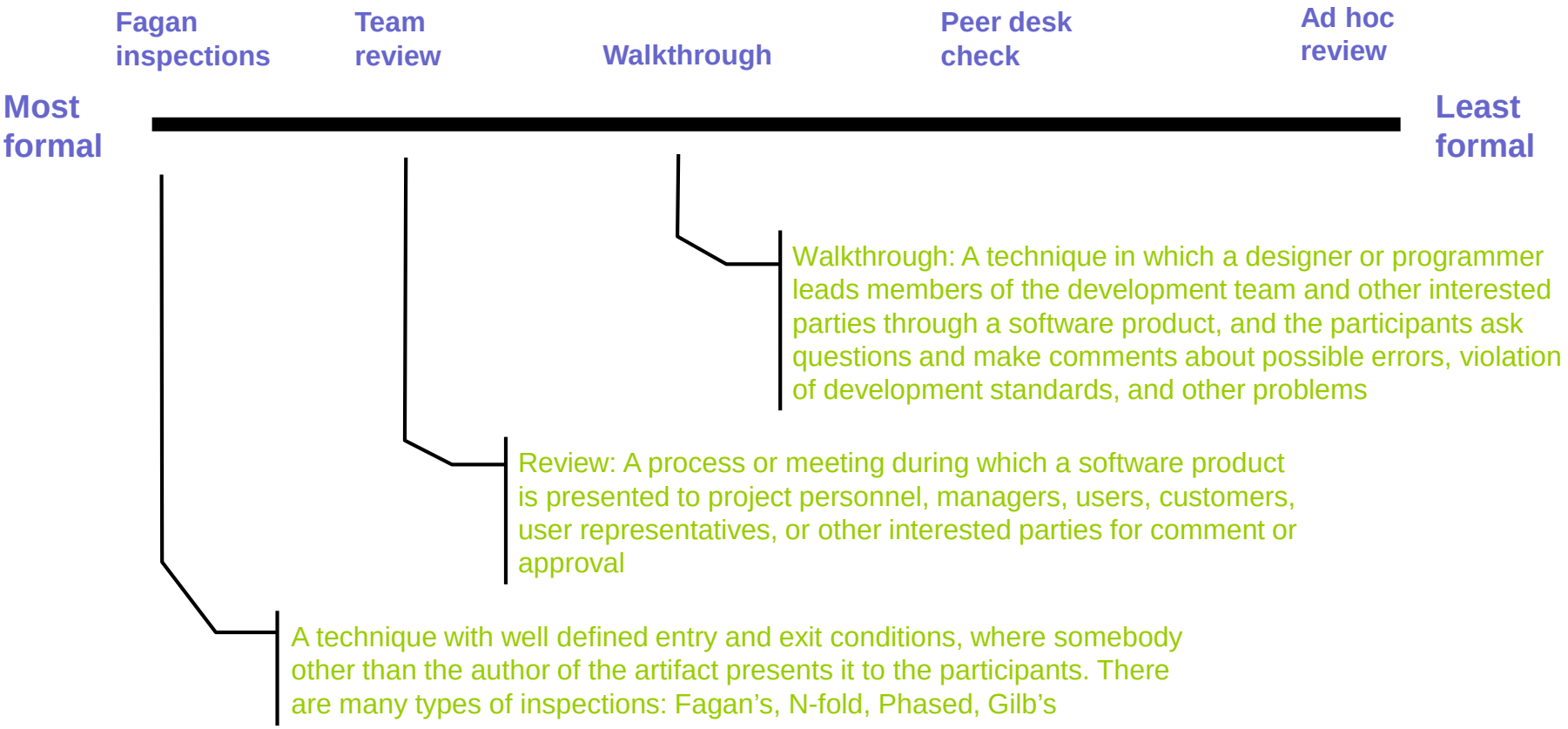
- Hewlett-Packard, ROI 10 to 1. Savings estimated at $21.4 million per year. [1]
- AT&T Bell, ten-fold improvement in quality and a 14 percent increase in productivity at Laboratories [2]
- Bell Northern Research, average savings of 33 hours of maintenance effort per defect discovered [3]

They uncover defects that would be difficult or impossible to discover by means of testing.

Inspections improve learning and communication within the software team.

[1] Grady, Robert B., and Tom Van Slack. “Key Lessons in Achieving Widespread Inspection Use,” IEEE Software, Vol. 11, No. 4 (July 1994), pp. 46-57.
[2] Humphrey, Watts S. Managing the Software Process. Reading, Massachusetts: Addison-Wesley, 1989
[3] Russell, Glen W. “Experience with Inspection in Ultra large-Scale Developments,” IEEE Software, Vol. 8, No. 1 (January 1991), pp. 25-31.

# The Family of Inspection Techniques




Adapted from K. Wiegers, Peer Reviews in Software: A Practical Guide, 2002

# Best Practices -1

Presentation made by somebody other than the author

- Forces another person to seriously read the work
- It exposes conflicts of understanding between what the author intended to express and what others interpreted

Participants & duration

- The author, the presenter, a facilitator, a reviewer
- Never more than 2 hours
- Exclude:
  - Anyone with known personality clashes with other reviewers
  - Anyone who is not qualified to contribute
  - Direct management

s. Rakitin, Software Verification and Validation A Practitioner's Guide, 1997

# Best Practices -2

Preparation: Independent review of materials

- Material is independently reviewed by the presenter and the reviewer.
- Use checklists to highlight know trouble spots.

During the inspection

- Review the product, not its author.
- Identify problems but don't try to solve them.
- Take notes.
- Before ending the inspection meeting summarize the issues to be resolved and review how the meeting itself went.

Make it fun

- Avoid presenting the material word by word.
- Avoid showing off how much smarter you are.
- Always remember: An examination of your work is coming next.

s. Rakitin, Software Verification and Validation A Practitioner's Guide, 1997

# Testing

The, more or less, thorough execution of the software with the purpose of finding bugs before the software is released for use and to establish confidence that the software performs as expected

# What things do we want to test?

Functions. See that each function does what it's supposed to do and does not, what it isn't.

Scenarios. Imagine use situations. Do one thing after another. Do not reset the system from test to test

Efficiency. Does the system provide appropriate performance, relative to the amount of resources used, under stated conditions

Robustness testing. Imagine calamities. The possibilities are endless. How does the system react to them?

# Testing Is Performed for Different Units of Analysis



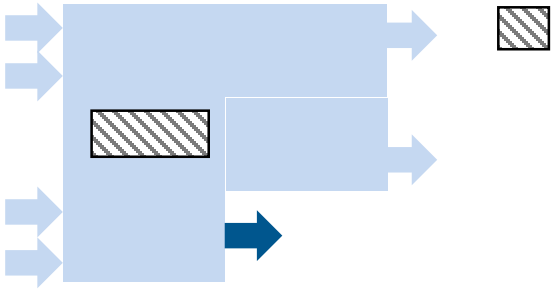
Unit testing

Integration testing

System testing

Acceptance testing

Regression testing



Modification to existing software

# Functional Testing Paradigms

Partition testing

- Equivalence classes
- Boundary value analysis
- Basis paths testing

Random testing

- Random testing
- Fuzz testing

Exploratory testing

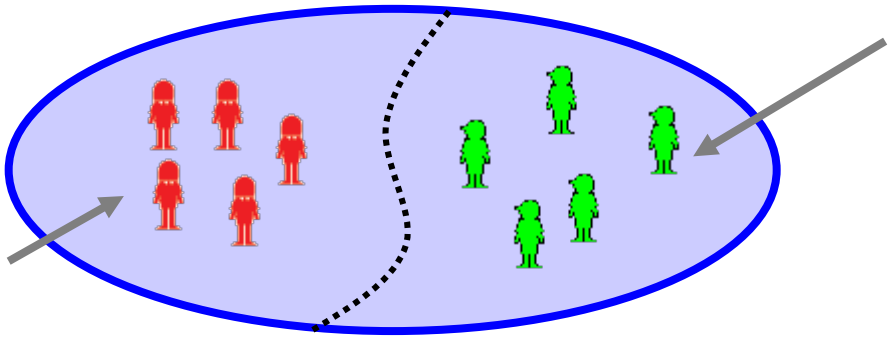
- Ad-hoc
- Session base testing

# Partition Testing

A partition is the division of the input domain of the software under test into a number of subsets for which the behavior is assumed to be the same for all values belonging to each of them.

The partition criteria utilized is what differentiates one test design technique from another.

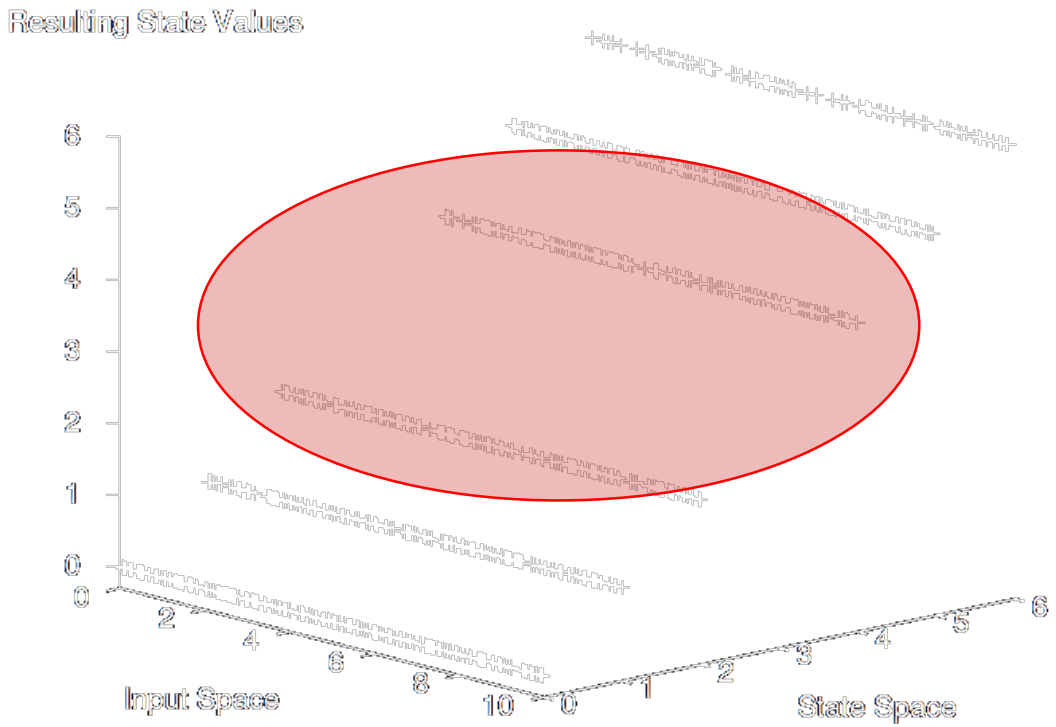
Any green results in "green behavior"



Any red results in "red behavior"

All values in a partition either result in a failure or produce a correct result.

# Random Testing




The systematic variation of values through the input space with the purpose of identifying abnormal output patterns

- When such patterns are identified a root cause analysis is conducted to identify the source of the problem.
- In this case the “state 3” outputs seem to be missing.

When Only Random Testing Will Do Dick Hamlet, 2006

# Exploratory (ad-hoc) Testing

Testing is performed on the fly, based on the skill and experience of the testers.

Useful for:

- Testing if few system specifications are available, but knowledge of the application and the anticipated goal is
- As a supplement to the “scripted” test design techniques

Limitations

- Poor exploration may result in a false sense of coverage and effectiveness.
- Lack of repeatability. There is no guarantee that a particular function will be tested in the same way by a different tester.

# Faults & Failures

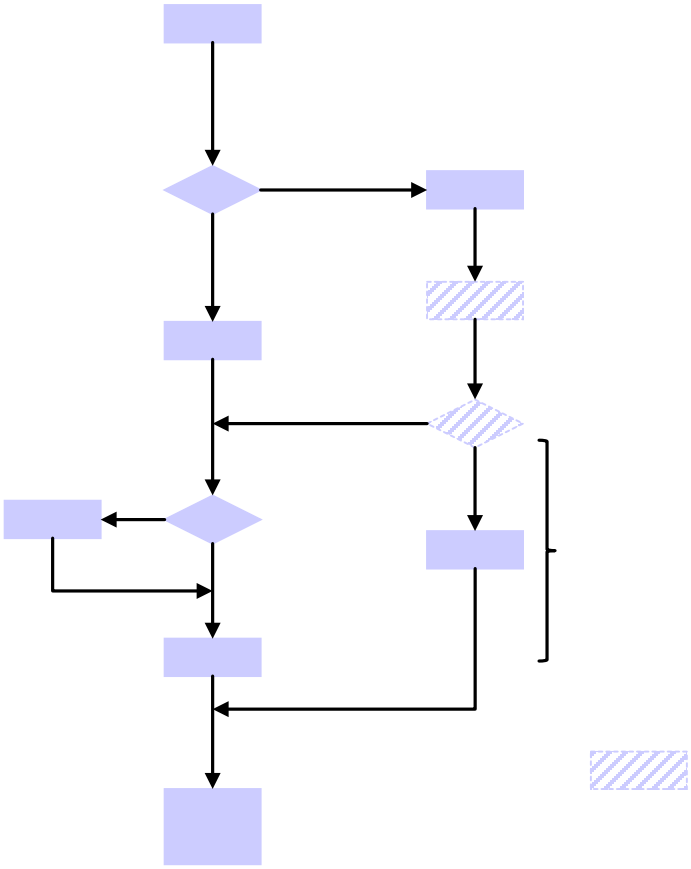
Fault

- An incorrect step, process, data definition or the lack of it in a computer program
- A *latent fault* is a fault that so far has not been discovered because the program was not executed with the data that triggers it.
- There are two types of faults:
  - Faults of omission
  - Faults of commission

Failure

- The inability of a software or software component to perform its required functions within specified performance requirements
- The manifestation of a fault

# Faults of Omission



Specified behavior that for some reason is not present in the software, e.g. the programmer forgot to program it in

- Initializations
- Validations
- Handling of special cases

They make for 22 to 54% of the total number of faults [1]

Should have been programmed but they weren't

[1] B. Marick, Faults of Omission, 2000

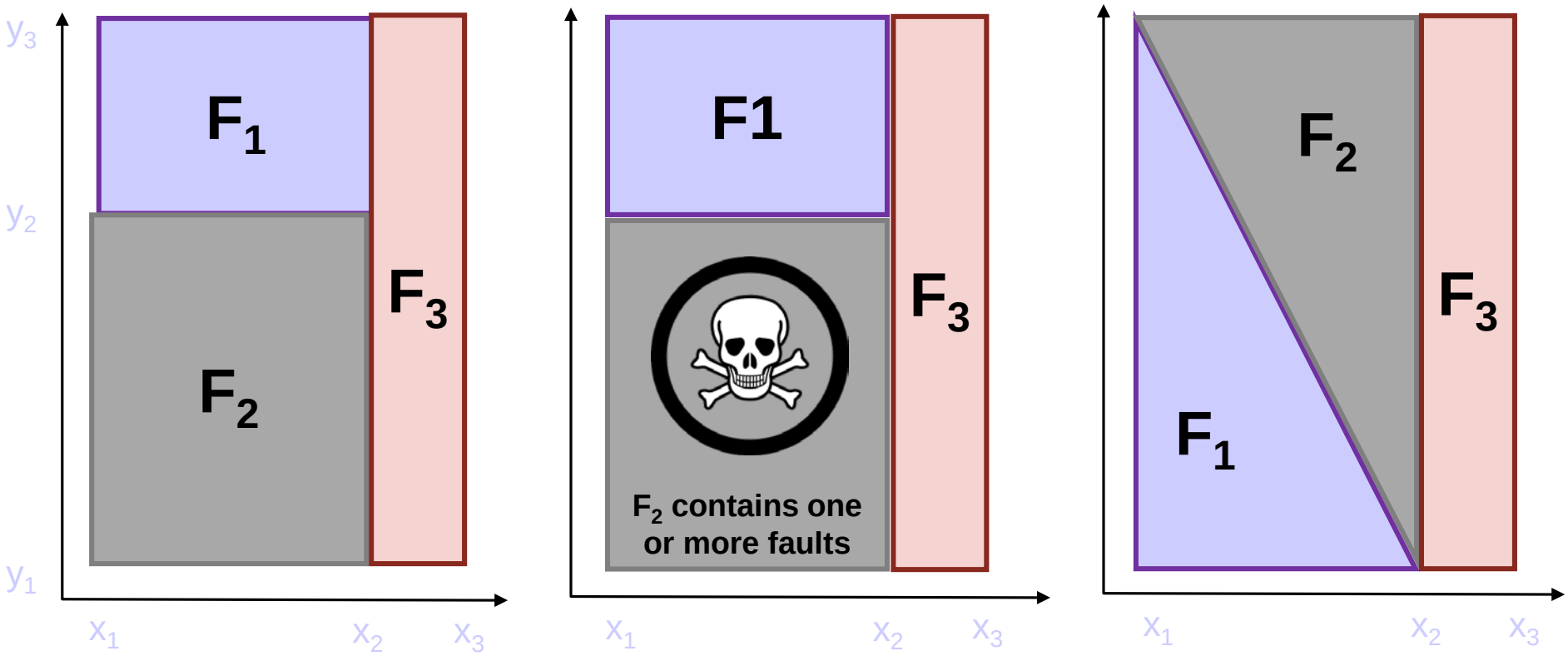
# Computation and Domain Faults

Specified behavior

Programmed behavior

Computation fault

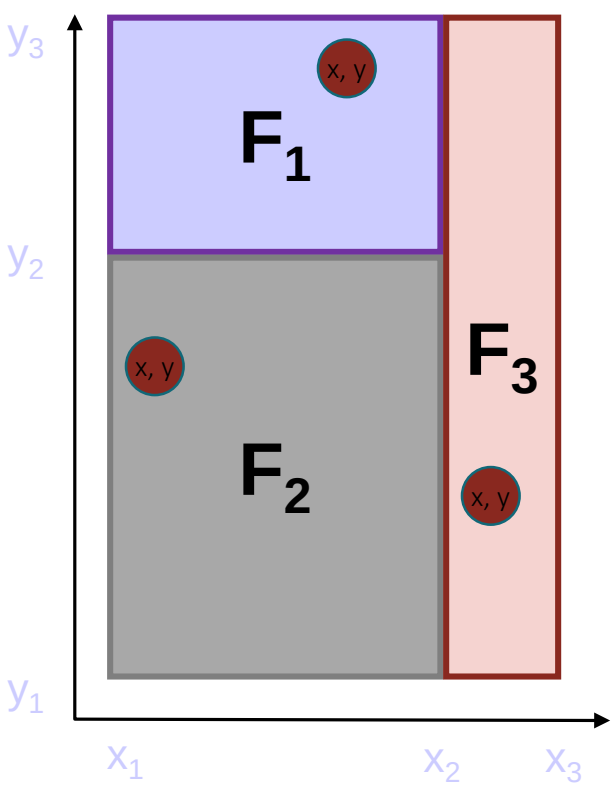
Domain fault



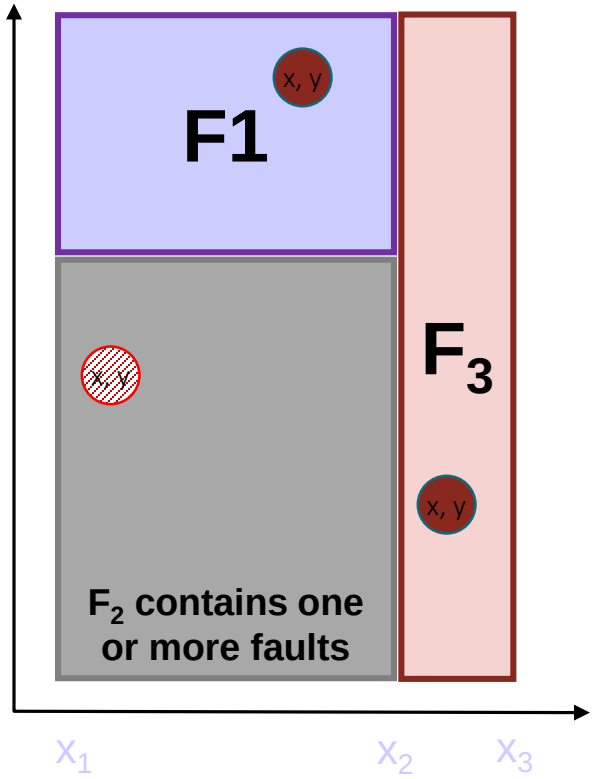
# Some Testing Techniques Are Better at Discovering Some Problems than Others

Specified behavior

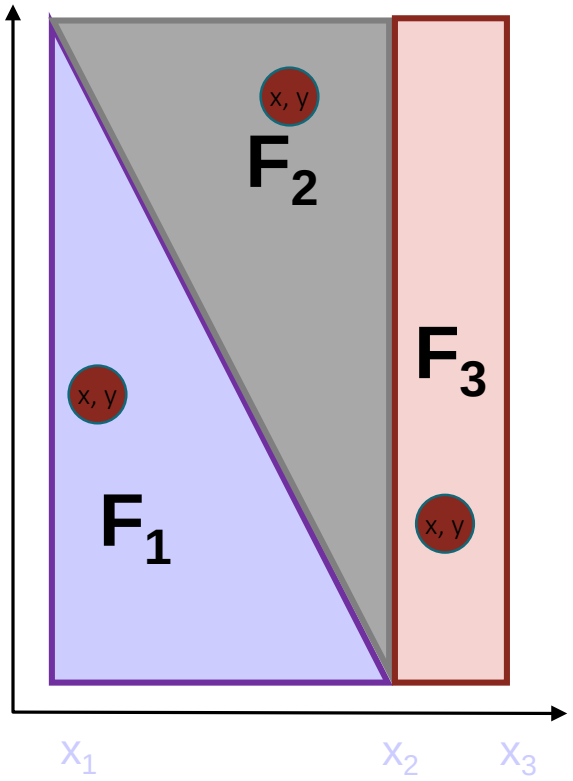
Programmed behavior



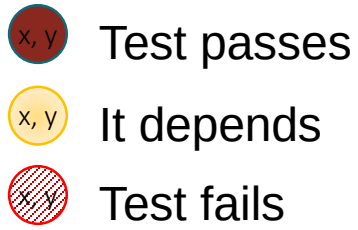

Computation fault




Domain fault

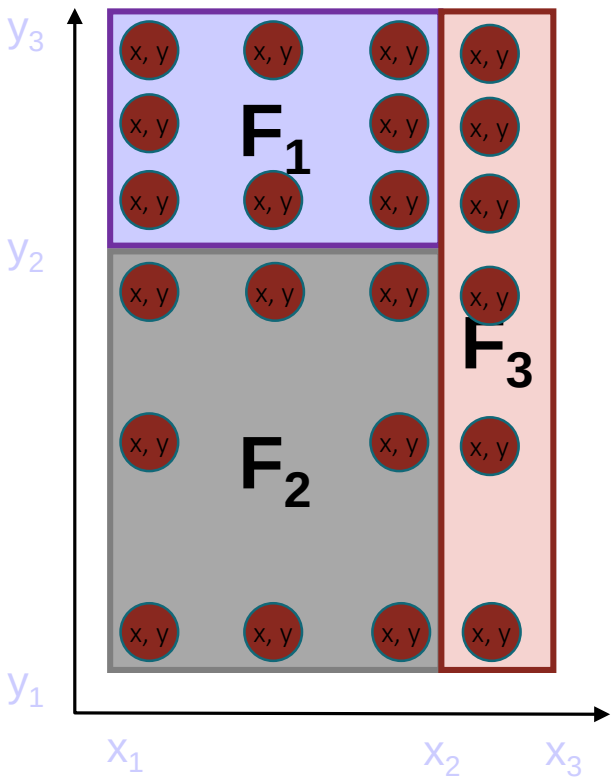


# Boundary Value Analysis Technique

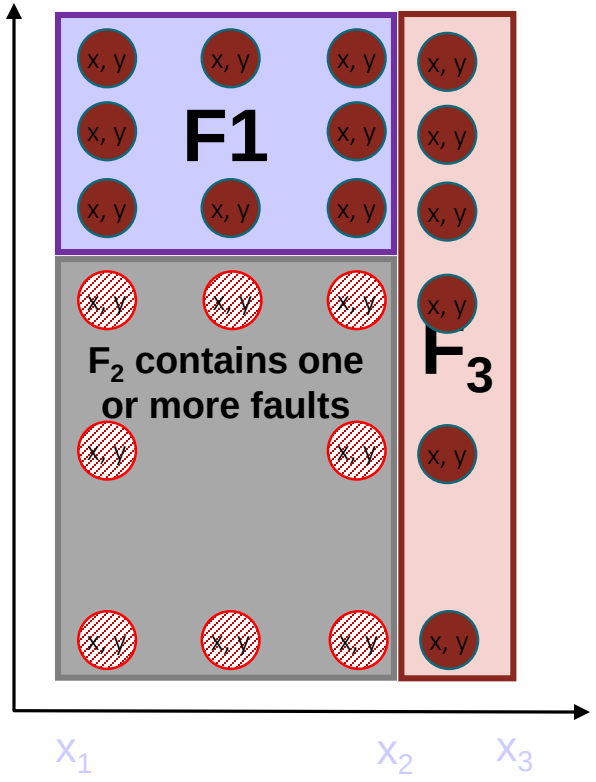



Specified behavior

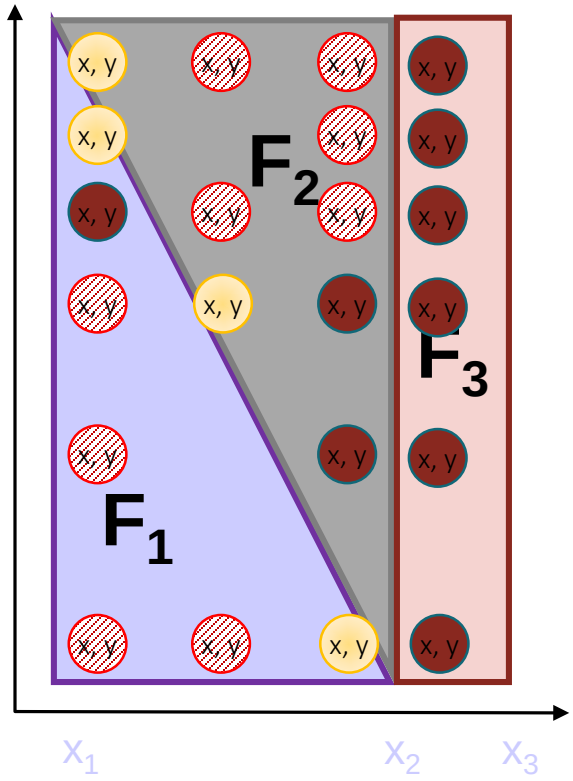
Programmed behavior




Computation fault




Domain fault

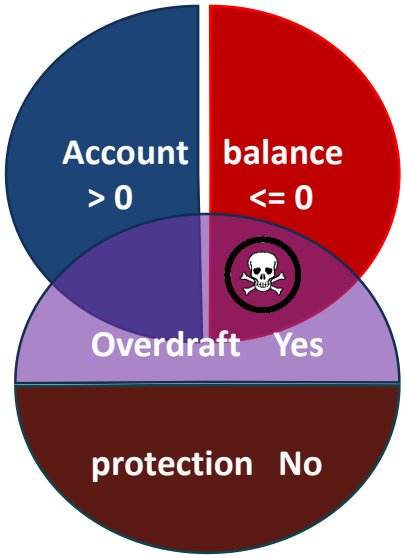


# What Triggers a Failure

The values of a single variable

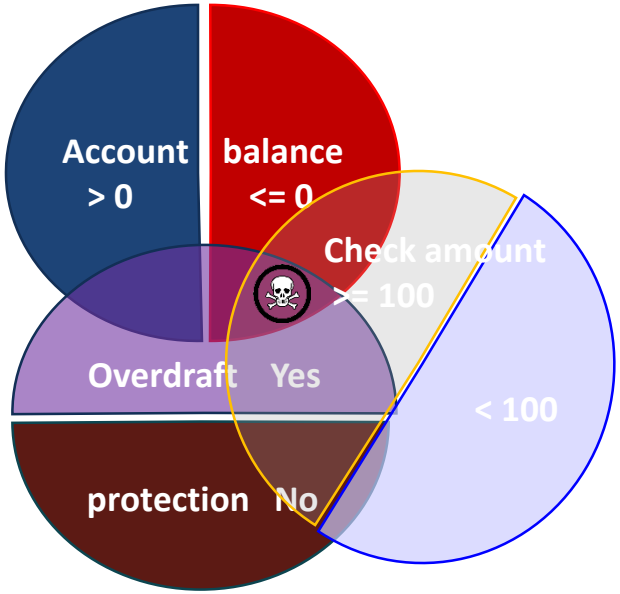

If AccountBalance < 0 then ☹!##$!!!

A combination of values of two variables



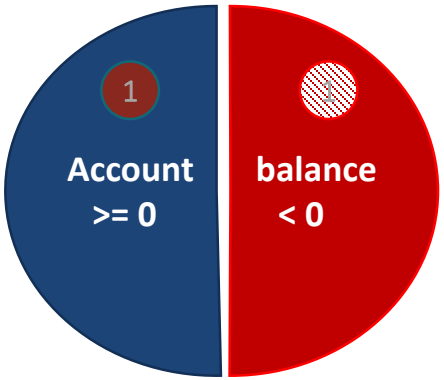

If AccountBalance < 0 and OverdraftProtection then ☹!##$!!!

A combination of values of many variables

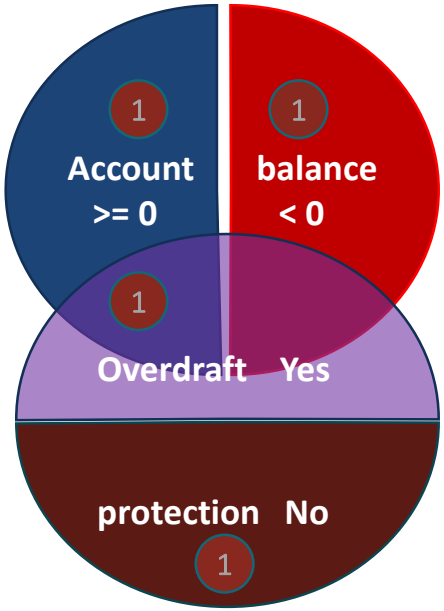



If AccountBalance < 0 and OverdraftProtection and Check Amount >= 100 then ☹!##$!!!

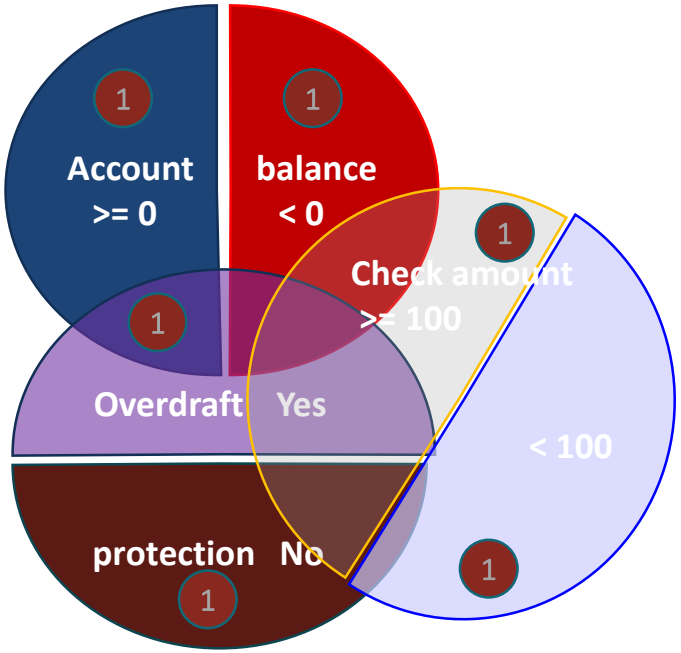
# Testing All Single Values




If AccountBalance < 0 then ☹!##$!!!




If AccountBalance < 0 and OverdraftProtection then ☹!##$!!!

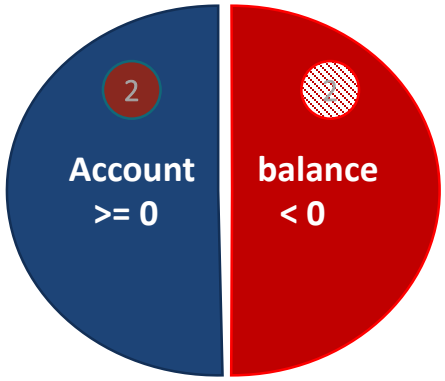



If AccountBalance < 0 and OverdraftProtection and Check Amount >= 100 then ☹!##$!!!

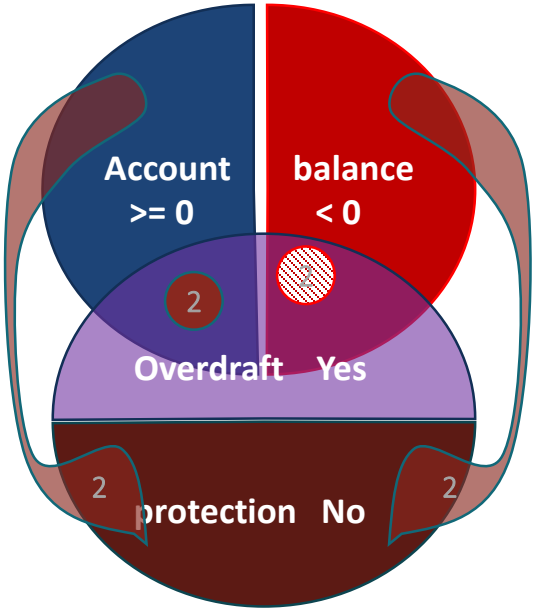
1 Test passes

1 Test fails

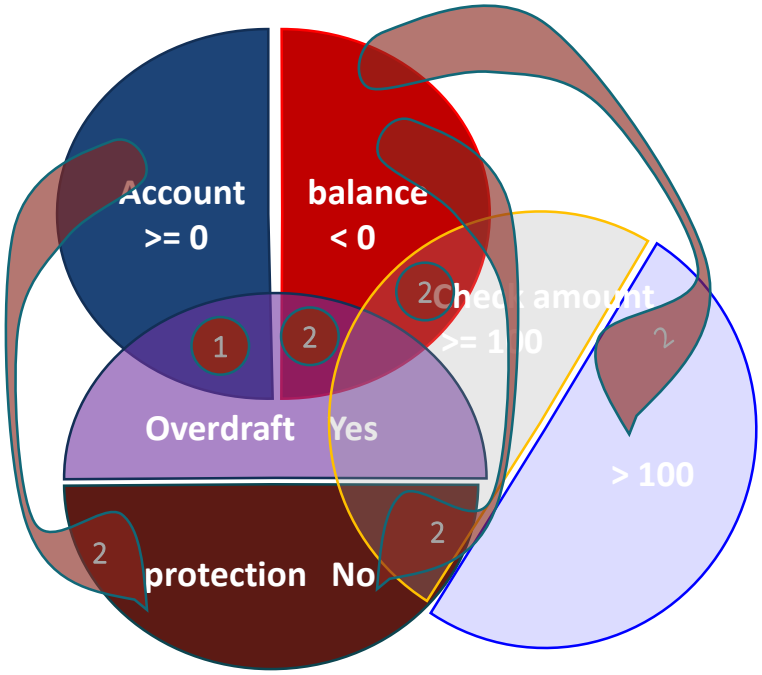
# To uncover interaction problems we need to systematically test for them. All pairs, …




If AccountBalance < 0 then ☹!##$!!!




If AccountBalance < 0 and OverdraftProtection then ☹!##$!!!




If AccountBalance < 0 and OverdraftProtection and Check Amount >= 100 then ☹!##$!!!

2 Test passes

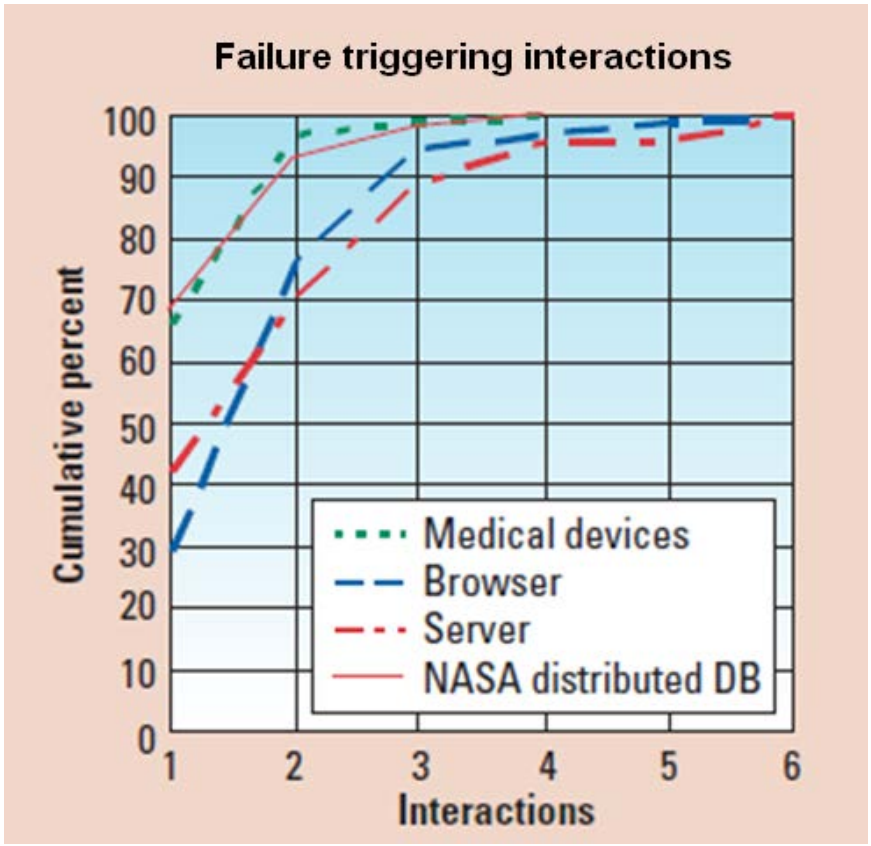
2 Test fails

# …all triples …

| Test Case | Account balance | Overdraft protection | Check amount | Result |
|---|---|---|---|---|
| 1 | < 0 | Yes | >= 100 | Fails |
| 2 | < 0 | Yes | < 100 | Passes |
| 3 | < 0 | No | >= 100 | Passes |
| 4 | < 0 | No | < 100 | Passes |
| 5 | >= 0 | Yes | >= 100 | Passes |
| 6 | >= 0 | Yes | < 100 | Passes |
| 7 | >= 0 | No | >= 100 | Passes |
| 8 | >= 0 | No | < 100 | Passes |

If Account_Balance < 0 and OverdraftProtection and Check Amount >= 100 then ☹**!##$!!!**

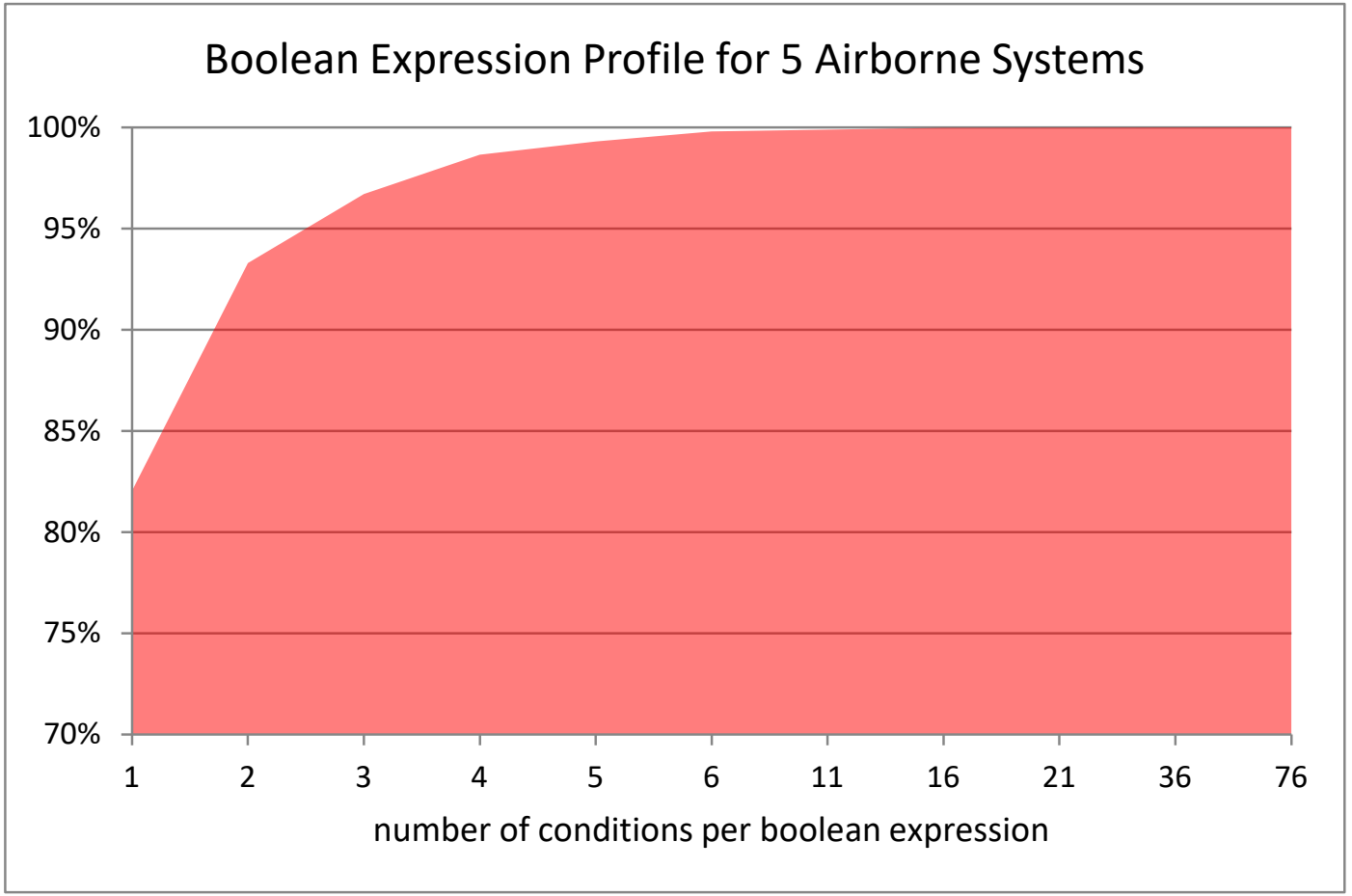
# Do we need to test for all combinations of values?




Combinatorial testing is based on empirical findings, there is no underlying "software physics". So while testing for 4 or 5 interactions is economically effective, and probably more thorough than what many organizations do today, there is no guarantee that it will find all faults.

Practical Combinatorial Testing: Beyond Pairwise, R. Kuhn, Yu Lei, R. Kacker, 2008

# A Hypothesis About What Causes Fault Interaction Distribution




NASA Practical Tutorial on Modified Condition/Decision Coverage, 2001

# Testing Techniques According to the Source of Information for Generating Test Cases

Black box (aka specification based testing and functional testing)

- Based on the input domain/expected behavior/specifications and knowledge of how software might fail
- Examples: Boundary value analysis, combinatorial testing, decision tables

White box (aka structural testing)

- Based on the structure of the software
- Examples: All basis paths, All definitions and use of a variable

# Black Box Testing: Looking at the Specification and at Our Knowledge of How Software Might Fail

**Specification**

Identify the software under test.

- Things that the product can do (functions and sub functions).

Identify relevant test aspects.

- Values and other attributes of the data
- Execution conditions

Design test cases.

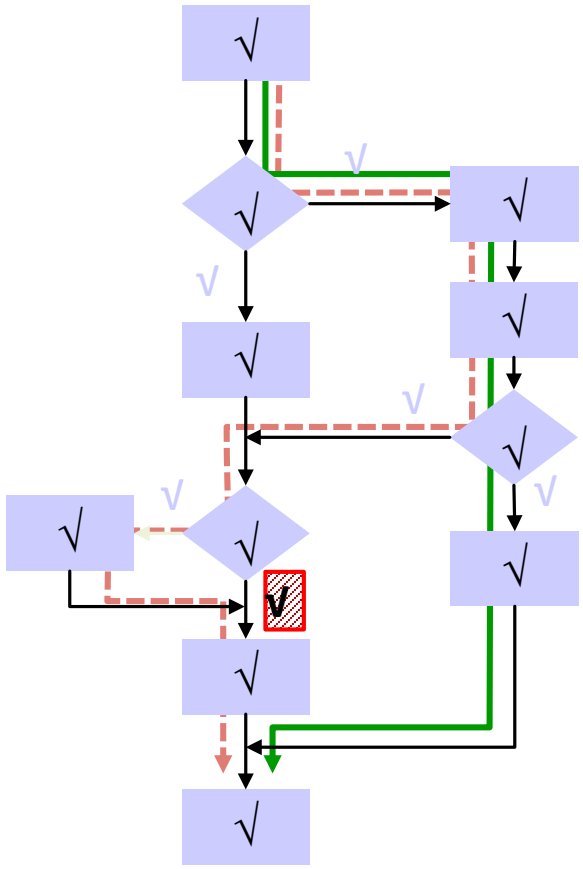
- Decide which particular data to test with. Consider things like boundary values, typical values, convenient values, invalid values, or best representatives.
- Consider combinations of data worth testing together.

Determine how you would know if a function is working (expected test result).

Test each function, one at a time. See that each function:

- Does what it's supposed to do; and
- Does not do what it isn't supposed.

# White Box Testing: Looking at the Code



Not verified

Verify that all tests are passed and:

- All the code statements have been executed (statement coverage)
- Both branches of each condition have been executed (branch coverage)
- All paths (impossible in most but the simplest situations) have been traversed (path coverage)

The fact that all statements and all branches have been executed does not guarantee that the software is fault free since the coverage criteria can be met and still there might be some faults that were not triggered by the data.

# White or black box testing? Both

## White box testing

Good for identifying incorrect or missing implementation of stated requirements

Test cases can be written by users and technologist alike

On its own, gives no indication of how thoroughly the program code has been tested

Can be used to assess whether any features in the requirements remain untested

## Black box testing

Test what is written, not what was intended

Knowledge of the implementation helps to include test cases that may not be identified from specifications alone

Good for discovering additional, perhaps unwanted, functionality, e.g. intrusive or unreachable code

Can be used to assess precisely what code features remain untested

On its own, gives no indication of how thoroughly the stated requirements have been tested

# Questions?


# Module 23: Specific Techniques

(Developed by Eduardo Miranda)

Introduction to Assured Software Engineering

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

# Notices



This material is based upon work funded and supported by the Independent Agency under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center sponsored by the United States Department of Defense.

# Topics

Equivalence Classes

Testing Methods

Code Coverage

Advanced Testing Techniques

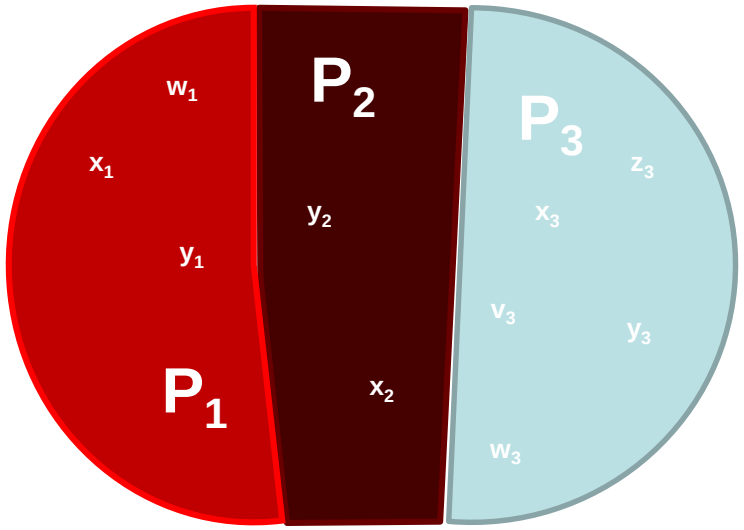
# Equivalence Classes

Based on the specification or some other information it is possible to hypothesize that the input domain is made up of a number of partitions (e.g.: P1, P2 and P3) →

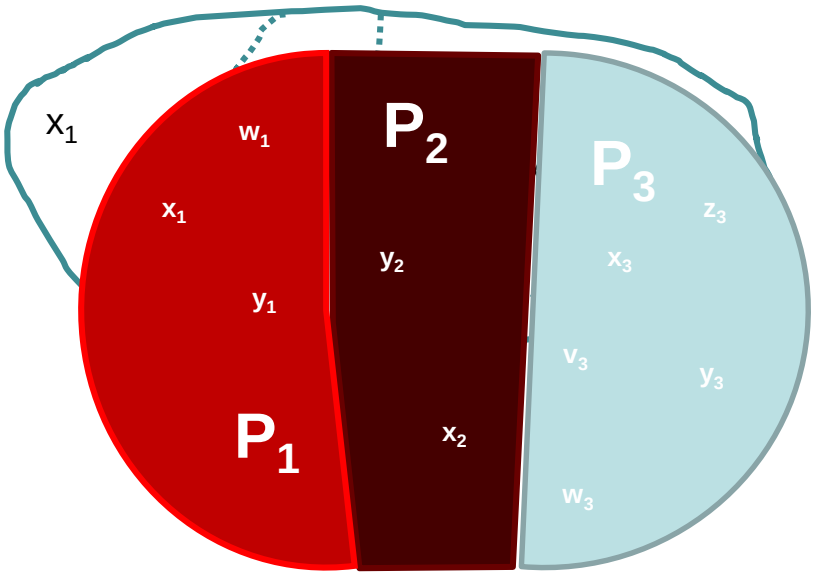
1. Values from the same partition exhibit the same behavior not the same result (e.g., they are calculated using the same procedure).
2. Values from different partitions exhibit different behavior.

Two situations

- The members of the partition are defined by enumeration.
- The partition can be defined by comprehension (intention, formula).



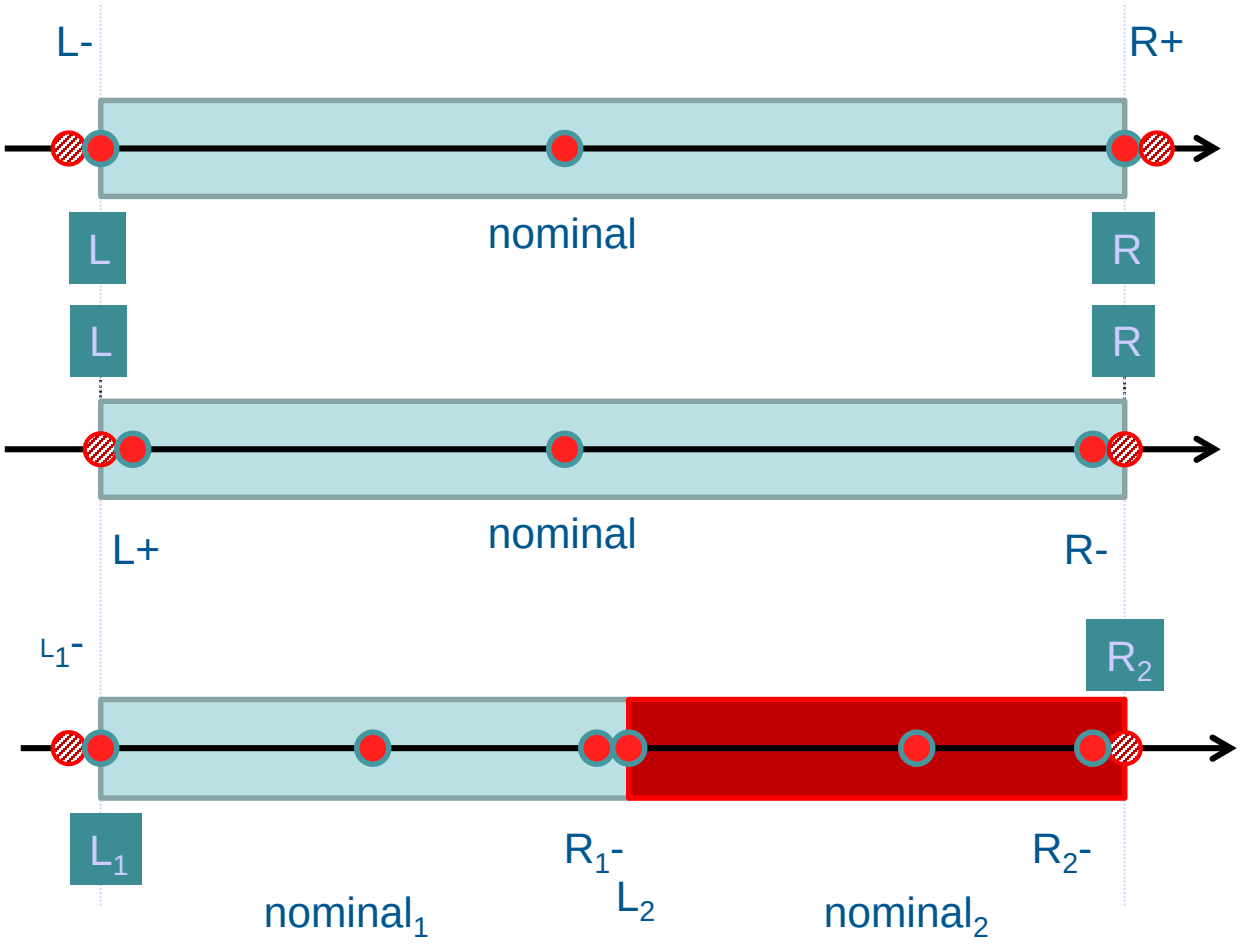
# Case 1: Membership to an Equivalence Class Is Defined by Enumeration




Once the equivalence classes have been identified, create two test cases (if possible) for each equivalence class.

- If any or both test cases do not result in the expected values you can discard the equivalence hypothesis.
- Additional test cases for each equivalence class may give us more confidence, but the best we can hope for, without testing all members, is to disprove the equivalence hypothesis.

# Case 2: Membership to the Equivalence Class is Defined by Intension: Boundary Value Analysis




- Test cases identified for single-variable, single-range closed interval. Shaded area indicates valid values of the variable ($L \leq X \leq R$)
- Test cases for single variable, single-range, open interval ($L < X < R$)
- Single-Variable, with two equivalence classes. Notice that adjacent classes do not overlap. If the upper bound of one is closed the lower bound of the other is open or vice versa ($L1 \leq X1 < R1$, $L2 \leq X2 < R2$)

# Applying Equivalence Class Testing to Factors Other Than the Value of a Variable

The use of equivalence classes is not restricted to the values of a variable, it can also be applied to other attributes such as the length of a string, the number of occurrences of an element, etc.

It is a well known fact that many software faults are caused by the treatment of “special” cases such as (notice that these are not invalid data):

- Null entries
- Entries with maximum lengths
- Whether a character string contains spaces or not
- Whether the treatment a particular piece of data is the same irrespective of
  - Its order in a sequence
  - Whether it occurs once or more

# How many test cases do we need in order to be reasonably reassured that we have done a comprehensive testing job?


- If we wanted to test all possible combinations of the 34 switches in the panel we would need $2^{34}$ = 1.7 x $10^{10}$ test cases.
- What if we suspected that all faults involved only 3-way interactions among the 34 switches? In this case we could do it with only 33 tests.
- What if we were 99% certain that all faults involved at most 4 interactions? In this case we could do the job with only 85 tests.

Adapted from Combinatorial Methods for Cybersecurity Testing,Rick Kuhn and Raghu Kacker, National Institute of Standards and Technology, 2009

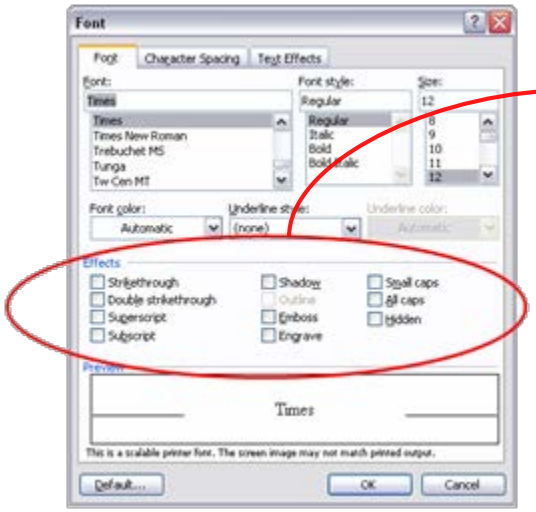
# Combinatorial Testing Defined

A technique that seeks to test all m combinations of a set of n >= m variables while minimizing the number of test cases by employing clever algorithms that package multiple unique combinations into each test case

Very useful in detecting faults involving the interaction of many independent variables

All but the simplest cases require the use of a computerized tool to efficiently generate the test cases.

# How does it work? All Triples, 10 Parameters Example




Adapted from Combinatorial Methods for Cybersecurity Testing,Rick Kuhn and Raghu Kacker, National Institute of Standards and Technology, 2009

1) Each variable to test becomes a column in a covering array

2) The generation algorithm searches, for each row of the array, the configuration of values that cover most combinations until all of them have been covered

| K1 | K2 | K3 | K4 | K5 | K6 | K7 | K8 | K9 | K10 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 | * | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | * | 1 | 1 | 1 |
| * | * | * | * | 1 | 0 | * | * | 0 | 1 |
| * | * | * | * | 0 | 1 | * | * | 1 | 0 |

3) In general the number of test cases needed to test all $t$-interactions of $k$ variables with $v$ values each is proportional to $\lceil v^t \ln k \rceil$ or to $\lceil v_{most\ values} \times v_{2nd\ most\ values} \times \cdots \times v_{t-most\ values} \times \ln k \rceil$

# Combinatorial Generation Examples

## 4 way interaction test suite

| | ACCOUNT... | OVERDRAFT... | CHECK... | HOUSING | CREDITCARD |
|---|---|---|---|---|---|
| 1 | >=0 | Yes | >=100 | Rent | Yes |
| 2 | <0 | No | <100 | Rent | No |
| 3 | >=0 | No | >=100 | Rent | With other |
| 4 | <0 | Yes | <100 | Mortgage with us | Yes |
| 5 | >=0 | Yes | >=100 | Mortgage with us | No |
| 6 | <0 | No | <100 | Mortgage with us | With other |
| 7 | >=0 | No | <100 | Morgage with other | Yes |
| 8 | <0 | Yes | >=100 | Morgage with other | No |
| 9 | * | Yes | * | Morgage with other | With other |
| 10 | >=0 | No | >=100 | Own | Yes |
| 11 | <0 | Yes | <100 | Own | No |
| 12 | * | * | * | Own | With other |

Test suites generated using the ACTS tool developed by NIST

## 2 way interaction test suite

| | ACCOUNT... | OVERDRAFT... | CHECK... | HOUSING | CREDITCARD |
|---|---|---|---|---|---|
| 1 | >=0 | Yes | >=100 | Rent | Yes |
| 2 | <0 | No | <100 | Rent | Yes |
| 3 | >=0 | No | >=100 | Rent | No |
| 4 | <0 | Yes | <100 | Rent | No |
| 5 | >=0 | Yes | <100 | Rent | With other |
| 6 | <0 | No | >=100 | Rent | With other |
| 7 | >=0 | No | <100 | Mortgage with us | Yes |
| 8 | <0 | Yes | >=100 | Mortgage with us | Yes |
| 9 | >=0 | Yes | >=100 | Mortgage with us | No |
| 10 | <0 | No | <100 | Mortgage with us | No |
| 11 | >=0 | No | >=100 | Mortgage with us | With other |
| 12 | <0 | Yes | <100 | Mortgage with us | With other |
| 13 | >=0 | Yes | <100 | Morgage with other | Yes |
| 14 | <0 | No | >=100 | Morgage with other | Yes |
| 15 | >=0 | No | <100 | Morgage with other | No |
| 16 | <0 | Yes | >=100 | Morgage with other | No |
| 17 | >=0 | Yes | >=100 | Morgage with other | With other |
| 18 | <0 | No | <100 | Morgage with other | With other |
| 19 | >=0 | Yes | >=100 | Own | Yes |
| 20 | <0 | No | <100 | Own | Yes |
| 21 | >=0 | No | >=100 | Own | No |
| 22 | <0 | Yes | <100 | Own | No |
| 23 | >=0 | Yes | <100 | Own | With other |
| 24 | <0 | No | >=100 | Own | With other |

# Combinatorial Testing Process

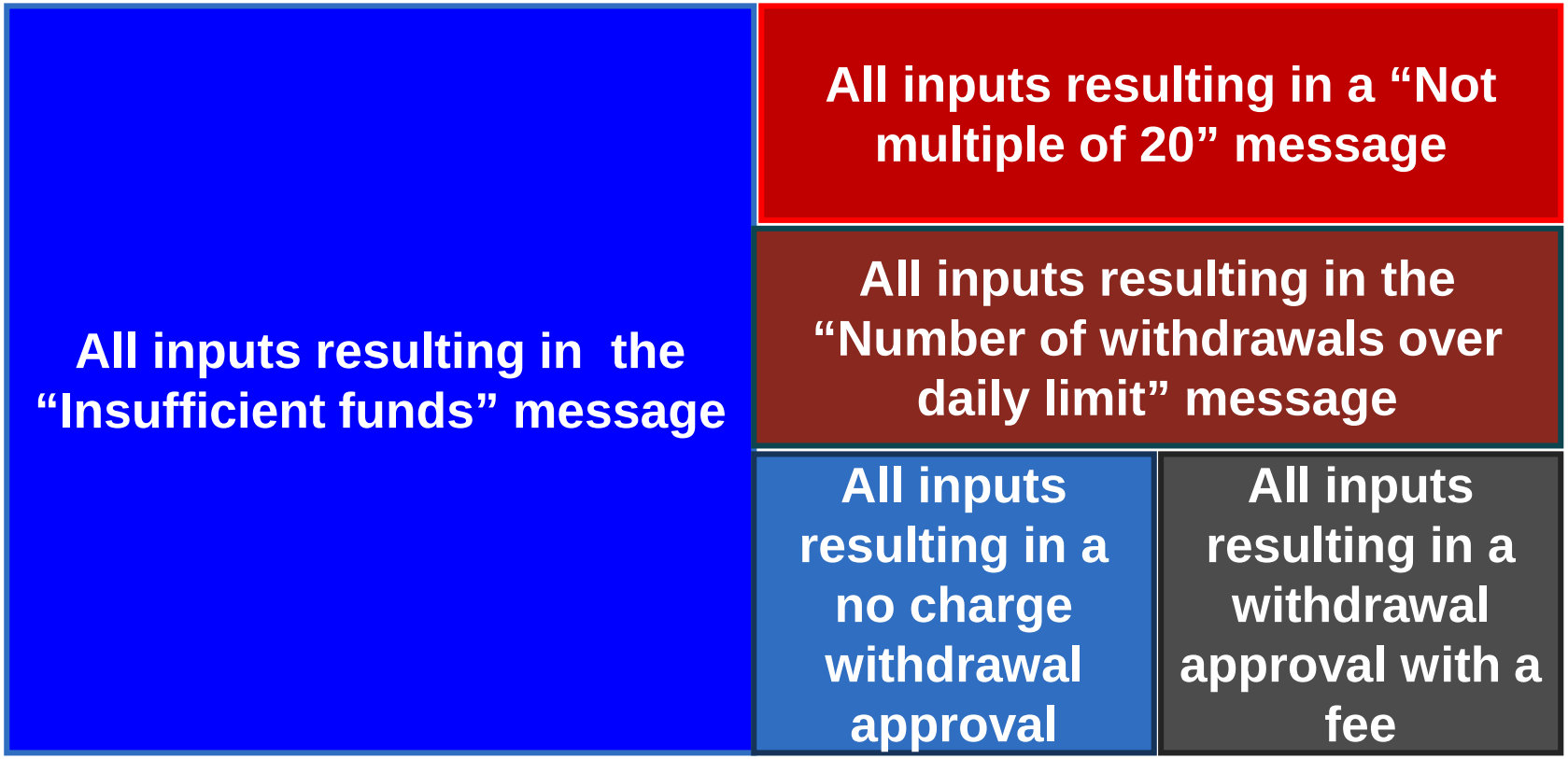
1. Individually verify the equivalence of each value to be used in in this process.
2. Choose the strength of the interaction to be tested (all pairs, all triples, etc).
   - In general avoid mixing negative and positive testing.
   - Do not test interactions among invalid values.
3. Generate test cases.
4. Complete test suite.
   - Add missing cases.
   - Remove impossible combinations.
5. Run and verify results.

# Decision Tables

## ATM withdraw decision logic

| Conditions (Inputs) | Rule 1 | Rule 2 | Rule 3 | Rule 4 | Rule 5 |
|---|---|---|---|---|---|
| Requested amount <= Balance | No | Yes | Yes | Yes | Yes |
| Requested amount divisible by 20 | * | No | Yes | Yes | Yes |
| Number of withdrawals | * | * | <= Free limit | <= Paid limit | > Paid limit |
| **Actions (Expected results)** | | | | | |
| Approve withdraw | | | X | X | |
| Insufficient funds message | X | | | | |
| Not multiple of 20 message | | X | | | |
| Charge withdrawal fee | | | | X | |
| Number of withdrawals over daily limit message | | | | | X |

# Decision Tables Partition the Input Domain from an Output Perspective



# When to Use Decision Tables

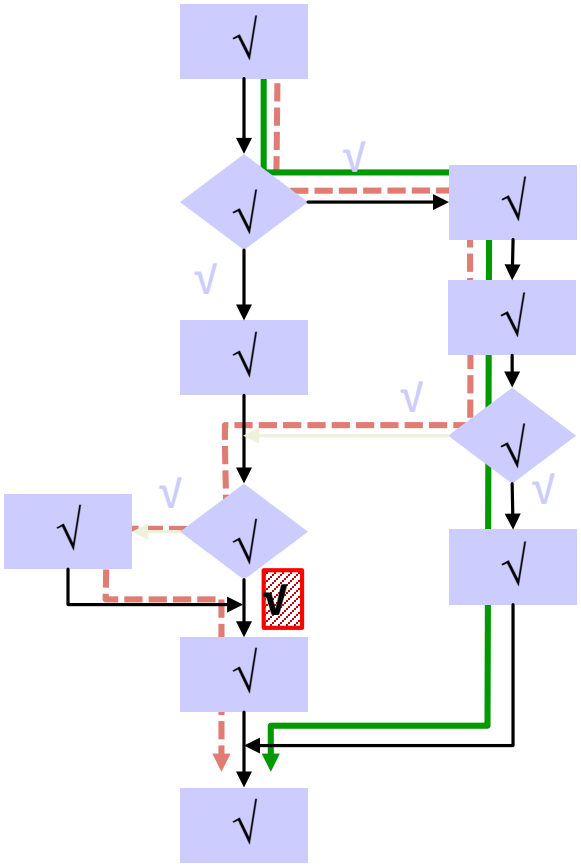
There are important logical relationships among input variables, i.e., the choosing of one value in one variable constrains the relevant values other variables might assume.

There are calculations involving subsets of input variables.

Actions are unequivocal.

- The order in which the conditions are evaluated do not affect the interpretation of the rules.
- The order of rule evaluation has no effect on which actions are selected.
- Once a rule is satisfied and the action selected, no other rule needs to be examined.

# White Box Testing



Control flow

- McCabe's Basis Path Testing
- Linear Code Sequences And Jumps (LCSAJ)

Data flow

- Program slicing

Logic flow

- Unique cause approach

# The Unique Cause Approach to Generate Test Cases that Satisfy the Modified Condition Criteria (MC)

The modified condition criteria originated in Boeing in 1994 and has been adopted by the aerospace industry (DO-178B) as the standard for coverage for high integrity systems.

MC criteria were developed with the purpose to achieve a degree of confidence in the software comparable to that provided by exhaustive testing, while requiring fewer test cases.

This is done by requiring the verification that each condition independently affects the outcome of a decision, i.e., one must demonstrate that the outcome of a decision changes as a result of changing the true values of a single condition.

This test is of special relevance to the testing of clinical remainders.

Applicability of modified condition decision coverage to software testing, J. Chilenski and S. Miller, 1994

# The Purpose of MC Is to Achieve a Degree of Confidence in the Software Comparable to that Provided by Exhaustive Testing, While Requiring Fewer Test Cases

Exhaustive test cases for an "AND" decision

| A | B | C | A & B & C |
|---|---|---|---|
| T | T | T | T |
| F | T | T | F |
| T | F | T | F |
| T | T | F | F |
| F | F | T | F |
| F | T | F | F |
| T | F | F | F |
| F | F | F | F |

Notice that the shaded cases do not contribute any new information since as soon as one condition is false the value of the others do not matter

Modified condition test cases for the same decision

| A | B | C | A & B & C |
|---|---|---|---|
| T | T | T | T |
| F | T | T | F |
| T | F | T | F |
| T | T | F | F |

The test cases demonstrates that the individual changes of each condition affect the outcome of the decision

Exhaustive test cases for an "OR" decision*

| A | B | C | A \| B \| C |
|---|---|---|---|
| F | F | F | F |
| T | F | F | T |
| F | T | F | T |
| F | F | T | T |
| F | T | T | T |
| T | F | T | T |
| T | T | F | T |
| T | T | T | T |

Modified condition test cases for the same decision

| A | B | C | A \| B \| C |
|---|---|---|---|
| F | F | F | F |
| T | F | F | T |
| F | T | F | T |
| F | F | T | T |

* The symbol "|" is used to represent the "Or" operator

## (A or B) and (C or D)

| | A | B | C | D | Decision |
|---|---|---|---|---|---|
| 1 | F | F | F | F | F |
| 2 | F | F | F | T | F |
| 3 | F | F | T | F | F |
| 4 | F | F | T | T | F |
| 5 | F | T | F | F | F |
| 6 | T | F | F | F | F |
| 7 | T | T | F | F | F |
| 8 | F | T | F | T | T |
| 9 | F | T | T | F | T |
| 10 | F | T | T | T | T |
| 11 | T | F | F | T | T |
| 12 | T | F | T | F | T |
| 13 | T | F | T | T | T |
| 14 | T | T | F | T | T |
| 15 | T | T | T | F | T |
| 16 | T | T | T | T | T |

## A more complicated example

| Condition | Test cases required | |
|---|---|---|
| A | 2 | 11 |
| B | 2 | 8 |
| C* | 6 | 12 |
| D | 6 | 11 |
| Test suite 2, 6, 8, 11, 12 | | |

* Notice that the solution is not unique. The same condition could have been tested by selecting rows 7 and 15

# Session Based Exploratory Testing

Time-boxed sessions

- Periods of two hours to one day at the end of which testing is considered done unless explicitly extended

Charters

- A clear mission for the session which suggests what should be tested, how it should be tested, and what problems to look for
- Things that should not be tested at this time, for example, because a separate charter was defined for these items

Debriefing

- How did you spend your time?
- Did you need special knowledge?
- Do you think there's more to do?

# Code Coverage

Coverage is a measure of the extent to which a given verification activity has achieved its objectives. It is calculated by dividing the measured items: statements, branches, conditions, etc., executed or evaluated at least once by their total number.

Appropriate coverage measures give the people doing, managing, and auditing verification activities a sense of the adequacy of the verification accomplished

# What situations do not give us a reasonable reassurance that we have done a comprehensive testing job?

After executing the software with our test suite we find that:

- Only 50% of the code statements were covered
- 90% of all statements were executed, but only 60% of the conditional ones were thoroughly (true and false values) evaluated
- 80% of the conditional statements were thoroughly evaluated, but only 40% of the conditions that made then up were shown to influence a result

# The Salutation Program*

```
1.  get (name, title, gender, maritalStatus)
2.  if title <> "" then
3.      salutation = title
4.  else
5.      if gender == "M" then
6.          salutation = "Mr."
7.      endif
8.      if gender == "F" && maritalStatus = "S" then
9.          salutation = "Ms."
10.     else
11.         salutation = "Mrs."
12.     endif
13. endif
14. print (salutation, name)
```

Code statistics

14 statements [1-14]

3 conditional statements [2, 5, 8]

Number of simple conditions affecting the outcome of a conditional statement [2, 5, 8, 8]

* There is a deliberate fault in the logic implemented

# Testing the Salutation Program

1. *get* (name, title, gender, maritalStatus)
2. *if* title <> “” *then*
3. salutation = title
4. *else*
5. *if* gender == “M” *then*
6. salutation = “Mr.”
7. *endif*
8. *if* gender == “F” && maritalStatus = “S” *then*
9. salutation = “Ms.”
10. *else*
11. salutation = “Mrs.”
12. *endif*
13. *endif*
14. *print* (salutation, name)

Test case 1

- (John, Dr., M, M)
- Expected result = “Dr. John”

Test Case 2

- (Mary, ,F, S)
- Expected result= “Ms. Mary”

Test Case 3

- (Laura, , F, M)
- Expected result= “Mrs. Laura”

# Test Case Adequacy Measures

| Test case | Counts from execution | Test result | Statistics |
|---|---|---|---|
| (John, Dr., M, M) Expected result = "Dr. John" | Stmts. = 5 (1, 2, 3, 13, 14) | "Dr. John" | Statement coverage = 5 / 14 = 35.7% |
| | Cond. = 1 (2) | | Branch coverage = 1 / 6 = 16.6% |
| | Branches =1 (3) | | Modified condition coverage = 0 / 4 = 0% |
| | Simple cond. = 0 | | |
| (Mary, ,F, S) Expected result= "Ms. Mary" | Stmts. = 10 (1, 2, 4, 5, 7, 8, 9, 12, 13, 14) | "Ms. Mary" | Statement coverage = 10 / 14 = 71.4% |
| | Cond. = 3 (2, 5, 8) | | Branch coverage = 3 / 6 = 50% |
| | Branches = 3 (4, 7, 9) | | Modified condition coverage = 0 / 4 = 0% |
| | Simple cond. = 0 | | |
| (Laura, , F, M) Expected result= "Mrs. Laura" | Stmts. = 11 (1, 2, 4, 5, 7, 8, 10, 11, 12, 13, 14) | "Mrs. Laura" | Statement coverage = 11 / 14 = 78.5% |
| | Cond. = 3 (2, 5, 8) | | Branch coverage = 3 / 6 = 50% |
| | Branches = 3 (4, 7, 10) | | Modified condition coverage 0 / 4 = 0% |
| | Simple cond. = 0 | | |
| Totals for the 3 tests | Stmts. = 13 (1, 2, 3, 4, 5, 7, 8, 9, 10, 11, 12, 13, 14) | | Statement coverage = 13 / 14 = 92.8% |
| | Cond. = 3 (2, 5, 8) | | Branch coverage = 5 / 6 = 83.3% |
| | Branches =5 (3, 4, 7, 9, 10) | | Modified condition coverage = 2 / 4 = 50% |
| | Simple cond. = 2 (2 - Title, 8 – maritalStatus) | | |

# 100% Coverage Does Not Imply All Faults Have Been Exposed

## Intended behavior

*If A **or** B then*

*Z = Z + 1*

*else*

*Z = Z + 2*

*endif*

## Programmed behavior

*If A **and** B then*

*Z = Z + 1*

*else*

*Z = Z + 2*

*endif*

Test suite

- A = True, B = True
- A = False, B = False

100% branch coverage without exposing the fault

# The Meaning of Coverage

Coverage directly measures the thoroughness of the test and only indirectly the quality of the software.

It must be emphasized that having 100% Statement Coverage or any other coverage metric does not guarantee that the code is 100% fault free.

# Summary

| | Applicability | What does it test? | Able to detect faults of omission | Type of data | Targets faults caused by (may uncover other problems) | Relations between variables |
|---|---|---|---|---|---|---|
| Inspections | Source code and documents | Functional and non-functional requirements | Yes | Any | Any | |
| Equivalence classes | Executable software | Functional requirements | Yes | Not ordered / Logical | A single variable | |
| Boundary value analysis | Executable software | Functional requirements | Yes | Ordered | A single variable | |
| Combinatorial testing | Executable software | Functional requirements | Yes | Any. Mostly valid data. Nominal values | Interactions among variables | Better suited for situations were each variable takes its values independent of others |
| Decision tables | Executable software | Functional requirements | Yes | Any | Interactions among variables | The value of one variable defines the possible values other variables may take |
| Modified condition | Executable software | Functional requirements | No | Logical | Compound logical predicates | |

# Advanced Testing Techuiques

# Developing Test Cases From Decision Tables

1. List all inputs and expected results.
2. Calculate the number of rules.
3. Fill columns with all possible combinations.
4. Specify expected outputs for each combination.
5. Reduce test combinations by identifying common actions.
6. Check covered combinations.
7. Develop test cases.

# Step 1: List All Inputs and Expected Results

| Conditions (Inputs) | Values | | |
|---|---|---|---|
| Requested amount <= Balance | Yes, No (2) | | |
| Requested amount divisible by 20 | Yes, No (2) | | |
| Number of withdrawals | <= Free limit, <= Paid limit, > Paid limit (3) | | |
| **Actions (Expected results)** | | | |
| Approve withdraw | | | |
| Insufficient funds message | | | |
| Not multiple of 20 message | | | |
| Charge withdrawal fee | | | |
| Number of withdrawals over daily limit message | | | |

Hints

- List each condition starting with the most dominant and putting the one with most values last.
- Each condition corresponds to a variable, relation or predicate.
- Write down representative values for each equivalence class the input variable or condition can assume.

# Step 2: Calculate the Number of Rules Required

| Conditions (Inputs) | Values | $R_1$ | … | $R_{12}$ |
|---|---|---|---|---|
| Requested amount <= Balance | Yes, No (2) | | | |
| Requested amount divisible by 20 | Yes, No (2) | | | |
| Number of withdrawals | <= Free limit, <= Paid limit, > Paid limit (3) | | | |
| **Actions (Expected results)** | | | | |
| Approve withdraw | | | | |
| Insufficient funds message | | | | |
| Not multiple of 20 message | | | | |
| Charge withdrawal fee | | | | |
| Number of withdrawals over daily limit message | | | | |

Hints

- A raw decision table will have as many rules as the product of the values of each condition.
- In this example 2 x 2 x 3 = 12 rules.

# Step 3: Fill Columns With All Possible Combinations

Hints

- Write down each value of each condition in repeating sequences of length k = combinations left / by the number of values of the row's input.
- In this example:
  1st row, k=12/2 = 6; 2nd row, k=6/2= 3, 3rd row, k=3/3=1

| Conditions (Inputs) | Values | $R_1$ | $R_2$ | $R_3$ | $R_4$ | $R_5$ | $R_6$ | $R_7$ | $R_8$ | $R_9$ | $R_{10}$ | $R_{11}$ | $R_{12}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Requested amount <= Balance | Yes, No (2) | No | No | No | No | No | No | Yes | Yes | Yes | Yes | Yes | Yes |
| Requested amount divisible by 20 | Yes, No (2) | No | No | No | Yes | Yes | Yes | No | No | No | Yes | Yes | Yes |
| Number of withdrawals | <= Free limit, <= Paid limit, > Paid limit (3) | <= FL | <= PL | > PL | <= FL | <= PL | > PL | <= FL | <= PL | > PL | <= FL | <= PL | > PL |
| Actions (Expected results) | | | | | | | | | | | | | |

# Step 4: Specify Expected Outputs for Each Combination

Hints

- Verify that there are no unmarked action rows (no rule triggers the associated action).
- Verify that the are no unmarked action columns (the rule does not trigger any action).

| **Conditions (Inputs)** | $R_1$ | $R_2$ | $R_3$ | $R_4$ | $R_5$ | $R_6$ | $R_7$ | $R_8$ | $R_9$ | $R_{10}$ | $R_{11}$ | $R_{12}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Requested amount <= Balance | No | No | No | No | No | No | Yes | Yes | Yes | Yes | Yes | Yes |
| Requested amount divisible by 20 | No | No | No | Yes | Yes | Yes | No | No | No | Yes | Yes | Yes |
| Number of withdrawals | <= FL | <= PL | > PL | <= FL | <= PL | > PL | <= FL | <= PL | > PL | <= FL | <= PL | > PL |
| **Actions (Expected results)** | | | | | | | | | | | | |
| Approve withdraw | | | | | | | | | | X | X | |
| Insufficient funds message | X | X | X | X | X | X | | | | | | |
| Not multiple of 20 message | | | | | | | X | X | X | | | |
| Charge withdrawal fee | | | | | | | | | | | X | |
| Number of withdrawals over daily limit message | | | | | | | | | | | | X |

# Step 5: Reduce Test Combinations by Identifying Common Actions

Hint

- The “Insufficient funds” message is not influenced by the divisibility of the amount by 20 nor by the number of withdrawals. $R_1$ to $R_6$ can be consolidated into a single rule with “don’t matter values”.

| **Conditions (Inputs)** | $R_1$ | $R_2$ | $R_3$ | $R_4$ | $R_5$ | $R_6$ | $R_7$ | $R_8$ | $R_9$ | $R_{10}$ | $R_{11}$ | $R_{12}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Requested amount <= Balance | No | No | No | No | No | No | Yes | Yes | Yes | Yes | Yes | Yes |
| Requested amount divisible by 20 | No | No | No | Yes | Yes | Yes | No | No | No | Yes | Yes | Yes |
| Number of withdrawals | <= FL | <= PL | > PL | <= FL | <= PL | > PL | <= FL | <= PL | > PL | <= FL | <= PL | > PL |
| **Actions (Expected results)** | | | | | | | | | | | | |
| Approve withdraw | | | | | | | | | | X | X | |
| Insufficient funds message | X | X | X | X | X | X | | | | | | |
| Not multiple of 20 message | | | | | | | X | X | X | | | |
| Charge withdrawal fee | | | | | | | | | | | X | |
| Number of withdrawals over daily limit message | | | | | | | | | | | | X |

# Step 6: Check Covered Combinations

| Conditions (Inputs) | Values | $R_1$ | $R_2$ | $R_3$ | $R_4$ | $R_5$ |
|---|---|---|---|---|---|---|
| Requested amount <= Balance | Yes, No (2) | No | Yes | Yes | Yes | Yes |
| Requested amount divisible by 20 | Yes, No (2) | * | No | Yes | Yes | Yes |
| Number of withdrawals | <= FL, <= PL, > PL (3) | * | * | <= FL | <= PL | > PL |
| **Actions (Expected results)** | | | | | | |
| Approve withdraw | | | | X | X | |
| Insufficient funds message | | X | | | | |
| Not multiple of 20 message | | | X | | | |
| Charge withdrawal fee | | | | | X | |
| Number of withdrawals over daily limit message | | | | | | X |
| **Check sum** | **12 =** | **6** | **+ 3** | **+ 1** | **+ 1** | **+ 1** |

Hints

- For each column calculate the number of combinations it represents.
- An " *" stands for as many combinations as values the input has.
- The total number of combinations for each column is either 1 or the product of the "*" rows in it.
- Add up the total for each row and and compare with the result of step 2. It must be equal.

# Step 7: Develop Test Cases

Create a test case for each rule.

Develop test cases covering each value in the “don’t matter conditions” to verify that they really don’t matter.

Example to test Rule 1

| Test case | Requested Amount <= Balance | Requested amount divisible by 20 | Daily withdrawals |
|---|---|---|---|
| 1 | No | Yes | <= Free Limit |
| 2 | No | No | <= Paid Limit |
| 3 | No | * | > Paid Limit |

# Process to Generate Test Suites Satisfying the Modified Condition Criteria -1

1. Breakdown the compound decision into its elementary conditions and label them A, B, C, etc.
2. Create a truth table for the decision.
3. Select the test cases that uniquely affect the outcome. There will be at least the number of elementary conditions + 1 test cases.

# Process to Generate Test Suites Satisfying the Modified Condition Criteria -2

4. Transform the true and false value of each condition in actual values to be used in the tests. Example:
   - Condition A is BloodPressure <= 140 mmHg
   - A = True will translate into a test case with a value of BloodPressure <= 140, e.g. 130 mmHg
   - A = False will translate into a test case with BloodPressure > 140, e.g. 150 mmHg
5. Assemble the selected test cases using the actual values.

# Example Approach

## (A or B) and (C or D)

| | A | B | C | D | Decision |
|---|---|---|---|---|---|
| 1 | F | F | F | F | F |
| 2 | F | F | F | T | F |
| 3 | F | F | T | F | F |
| 4 | F | F | T | T | F |
| 5 | F | T | F | F | F |
| 6 | T | F | F | F | F |
| 7 | T | T | F | F | F |
| 8 | F | T | F | T | T |
| 9 | F | T | T | F | T |
| 10 | F | T | T | T | T |
| 11 | T | F | F | T | T |
| 12 | T | F | T | F | T |
| 13 | T | F | T | T | T |
| 14 | T | T | F | T | T |
| 15 | T | T | T | F | T |
| 16 | T | T | T | T | T |

## Steps 2 & 3: Creating the truth table and selecting test cases

| Condition | Test cases required | |
|---|---|---|
| A | 2 | 11 |
| B | 2 | 8 |
| C* | 6 | 12 |
| D | 6 | 11 |
| Test suite 2, 6, 8, 11, 12 | | |

* Notice that the solution is not unique. The same condition could have been tested by selecting rows 7 and 15

# Questions?


# Module 24: Continuous Integration and Testing

(Authored by Kevin Gary, Arizona State University)

Introduction to Assured Software Engineering

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

# Notices



This material is based upon work funded and supported by the Independent Agency under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center sponsored by the United States Department of Defense.

# Topics

Traditional Testing Practices

Agile Testing Practices

# Traditional Testing Practices

Testing occurs once, near end of project

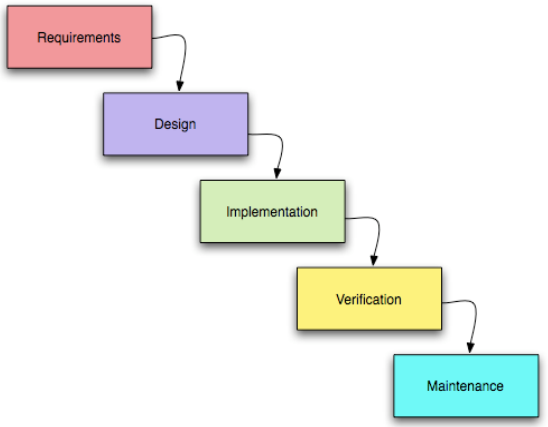
- Lots of lead time for test planning, test case generation, test lab and infrastructure setup

Test cases don't change (or don't change often)

- Cost of creating is paid once, not continuously
- Few changes to system once it is specified and designed

Tests executed periodically

- Initially to ensure system meets requirements
- Regression testing after significant change to ensure nothing broke



# Development Process Is Continuous

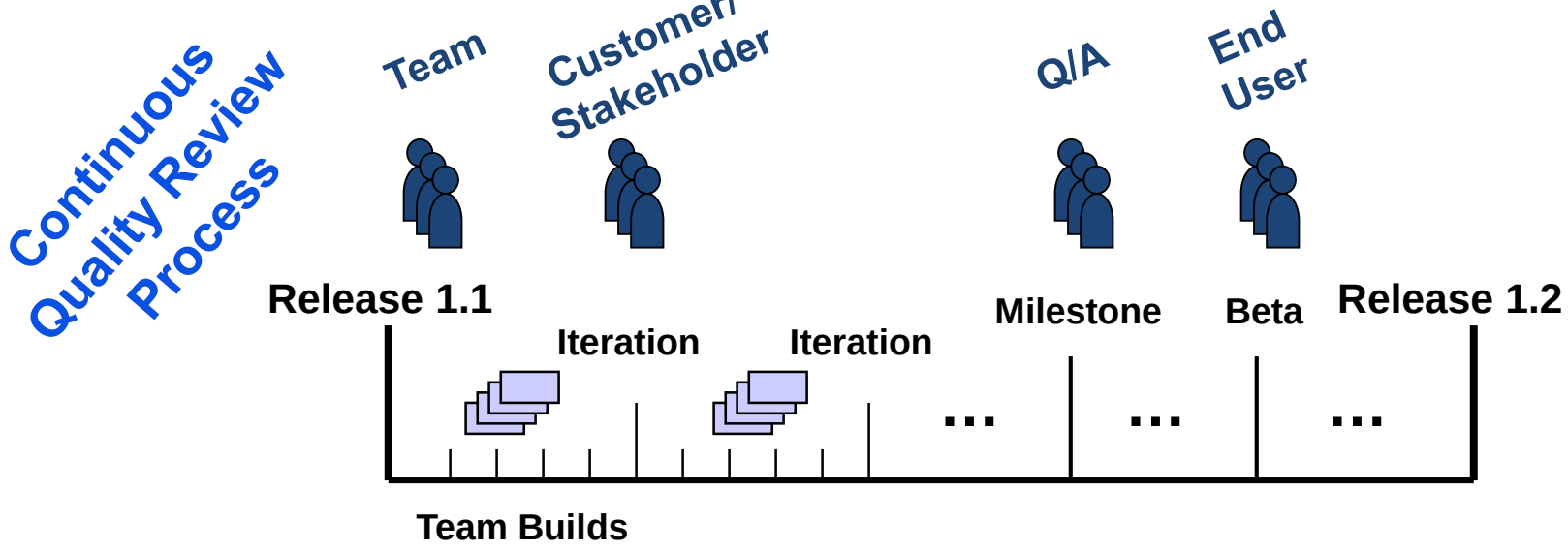
No separate “test” phase – integrate and test continuously

Features change during release – testing must adapt

Testing starts on project’s Day 1

- Initial plans, strategies, infrastructure required very early

Nightly builds
+ Adaptive planning
+ Continuous integration
= Testing nightmare



# Continuous Development => Test Automation

Continuous delivery and builds require automated testing

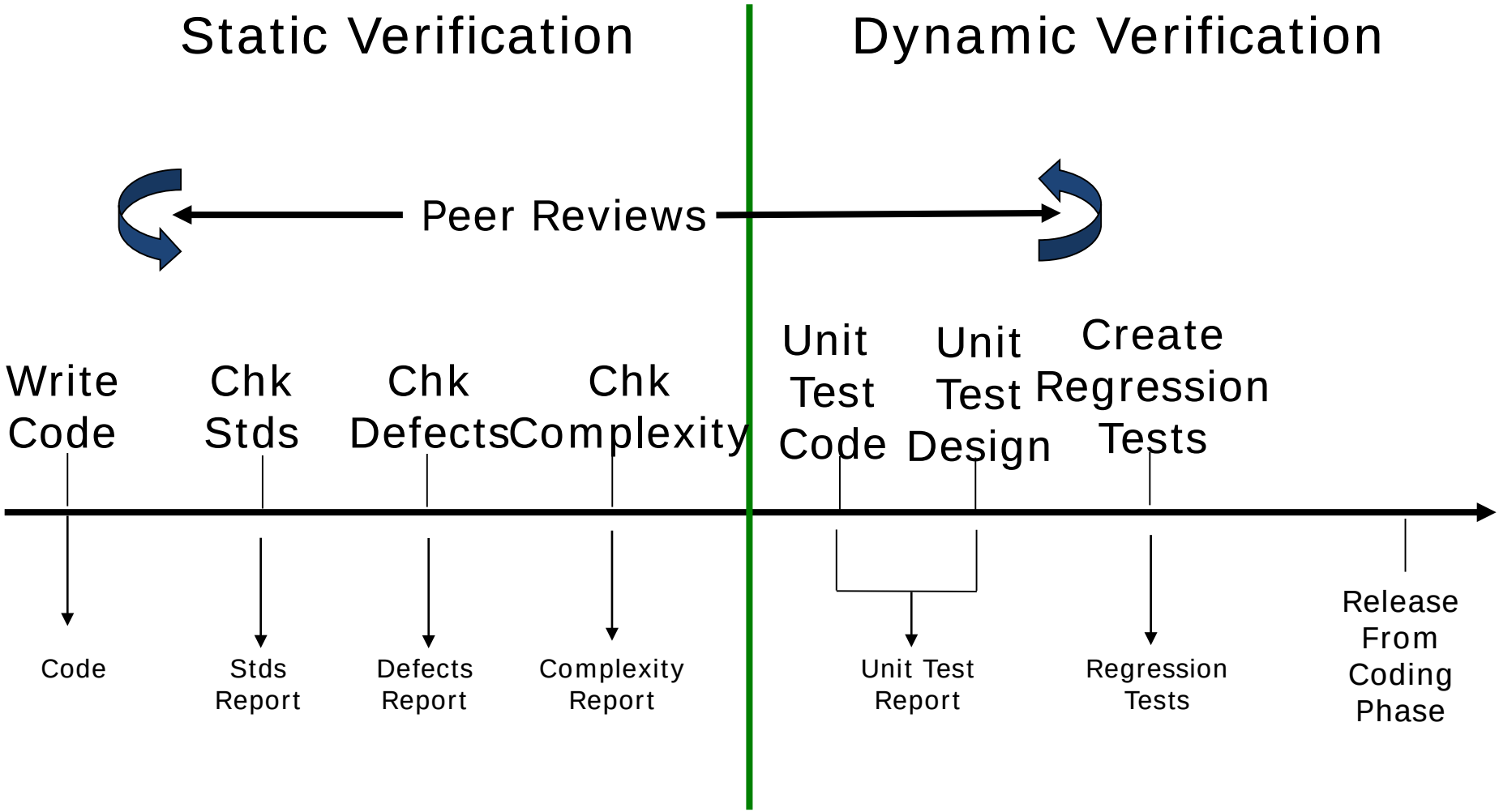
- Each build must be validated so future integrations build on a known quantity.

Test frameworks provide infrastructure to quickly standup unit testing.

- Governance and visibility – which test, on which build, metrics, trends

| Type of build | What tests? | Level of automation |
|---|---|---|
| Developer delivery to CM | • "Unit" tests (per component) | All automated |
| Team "nightly" builds | • Add "Smoke test" for integration | Most automated, limited manual |
| Iteration | • Add quality tests for coverage, static analysis, metrics, etc. | Quality numbers obtained automatically |
| Milestone iteration | • Add additional scripts per test plan – performance, scalability, stability, etc. | Mixed automation/manual, but as automated as possible |

# An Example “Code Acceptance Process”



# XP Best Practices: Continuous Integration

What is Continuous Integration?

- Integrate & build the system several times a day
- Integrate every time a task is completed
- Let's you know every day the status of the system

Continuous integration and relentless testing go hand-in-hand.

By keeping the system integrated at all times, you increase the chance of catching defects early and improving the quality and timeliness of your product.

Continuous integration helps everyone see what is going on in the system at all times.

*If **testing** is good, why not do it all the time? (continuous testing)*

*If **integration** is good, why not do it several times a day? (continuous integration)*

*If **customer involvement** is good, why not show the business value and quality we are creating as we create it (continuous reporting)*

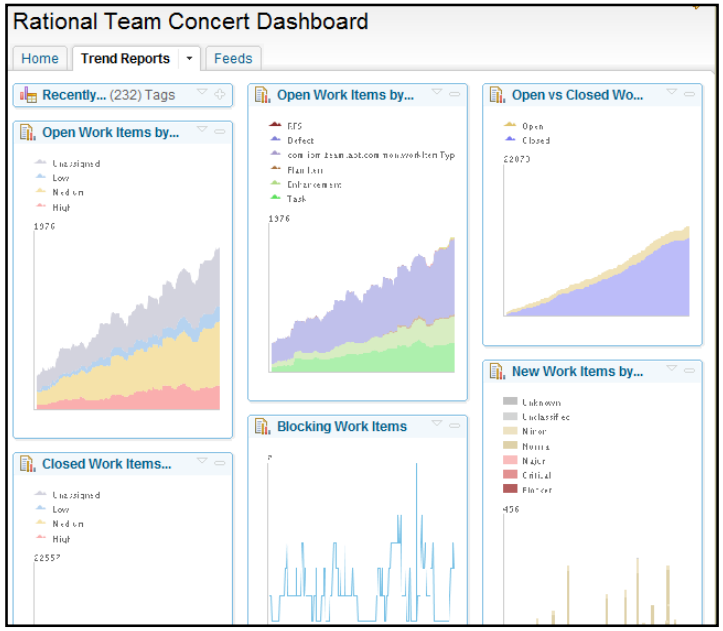
# Agile Best Practice: Continuous Testing

Unit, System, and Integration tests can be run continuously!

- Requires test automation and reporting framework
- Post results to a dashboard for all to see
  - Daily standup in the morning starts by checking if the dashboard is “green”.

Report on your static analysis / metrics while you’re at it!

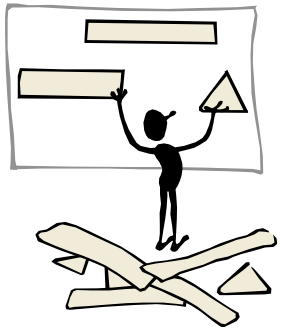
*Together with burndown charts, these show business value being built, with an attention to quality, at a sustainable pace.*

# Fowler's 10 Best Practices for CI

From

http://martinfowler.com/articles/continuousIntegration.html:



1. Maintain a Single Source Repository
2. Automate the Build
3. Make your Build Self-testing
4. Everyone Commits Everyday
5. Every Commit should Build the Mainline on an Integration Machine
6. Keep the Build Fast
7. Test in a Clone of the Production Environment
8. Make it easy for Anyone to get the Latest Executable
9. Everyone can see what's Happening
10. Automate Deployment

# Questions?