Chapter 2 – Software Process & Methodology
Chapter 3 – Systems Engineering

Dr John H Robb, PMP, IEEE SEMC

UT Arlington, Computer Science and Engineering

Chapter 2 - Key Takeaway Points

- A software process defines the phases of activities or what need to be performed to construct a software system. “What/when”
- A software methodology details the steps or how to perform the activities of a software process. A methodology is an implementation of a process. “How/Who”
- Software development needs a software process and a methodology.

Challenges of System Development

Project Reality 1. Many systems require many years to develop.

Project Challenge 1. How do we schedule, and manage the work without knowing exactly what the customer wants, and what may happen in the future?

Project Reality 2. Many software projects require collaboration of multiple departments and teams.

Project Challenge 2. How do we divide the work among the departments and teams, and integrate the components produced by them?

Project Reality 3. Different departments or teams may use different processes, methods, and tools. They may reside at different locations.

Project Challenge 3. How do we ensure proper communication and coordination among the departments and teams?

Challenges of Systems Development

System Reality 1. Many systems need to satisfy numerous requirements and constraints.

System Challenge 1. How do we develop systems to ensure that the requirements and constraints are met?

System Reality 2. Requirements and constraints may change from time to time.

System Challenge 2. How do we design the processes and the products to cope with change?

System Reality 3. A system may consist of hardware, software and third party components using different programming languages and run on multiple platforms and machines located at different places.

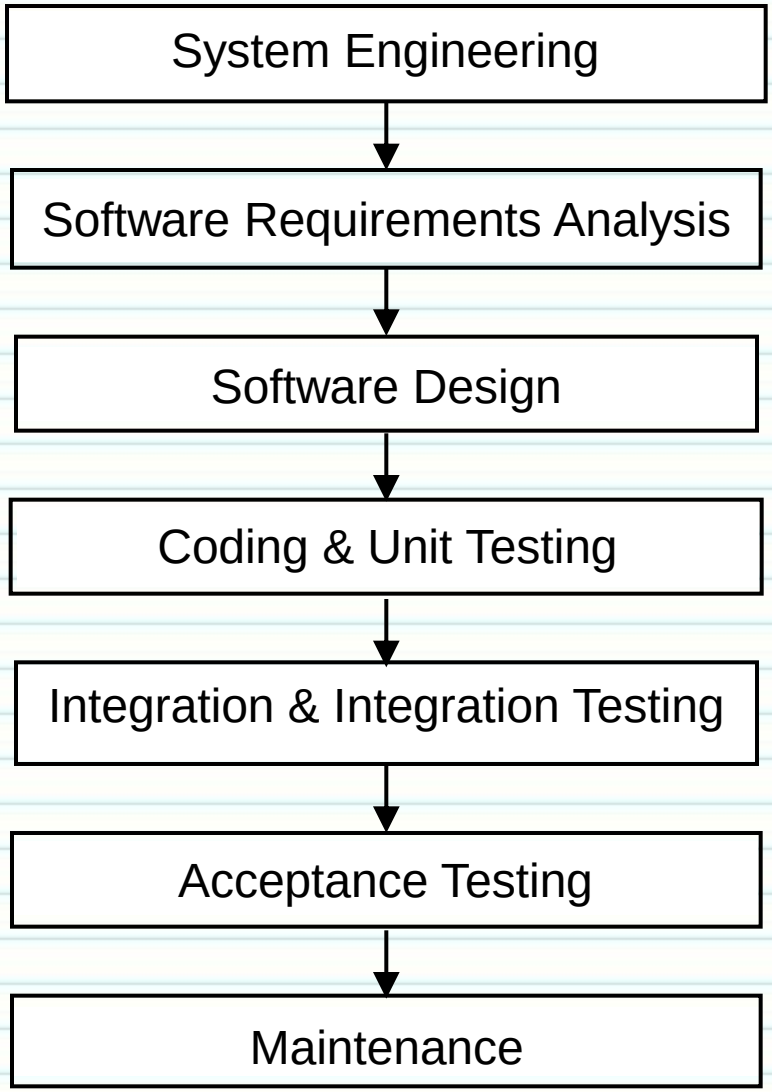
System Challenge 3. How do we design the system to hide these differences?

Software Process

System development challenges call for an engineering approach for software development. A software process is required.

Definition 2.1 A *software process* defines a series of activities performed to construct a software system. Each activity produces some artifacts, which are the input to other phases. Each phase has a set of entrance criteria and a set of exit criteria.

The Waterfall Process

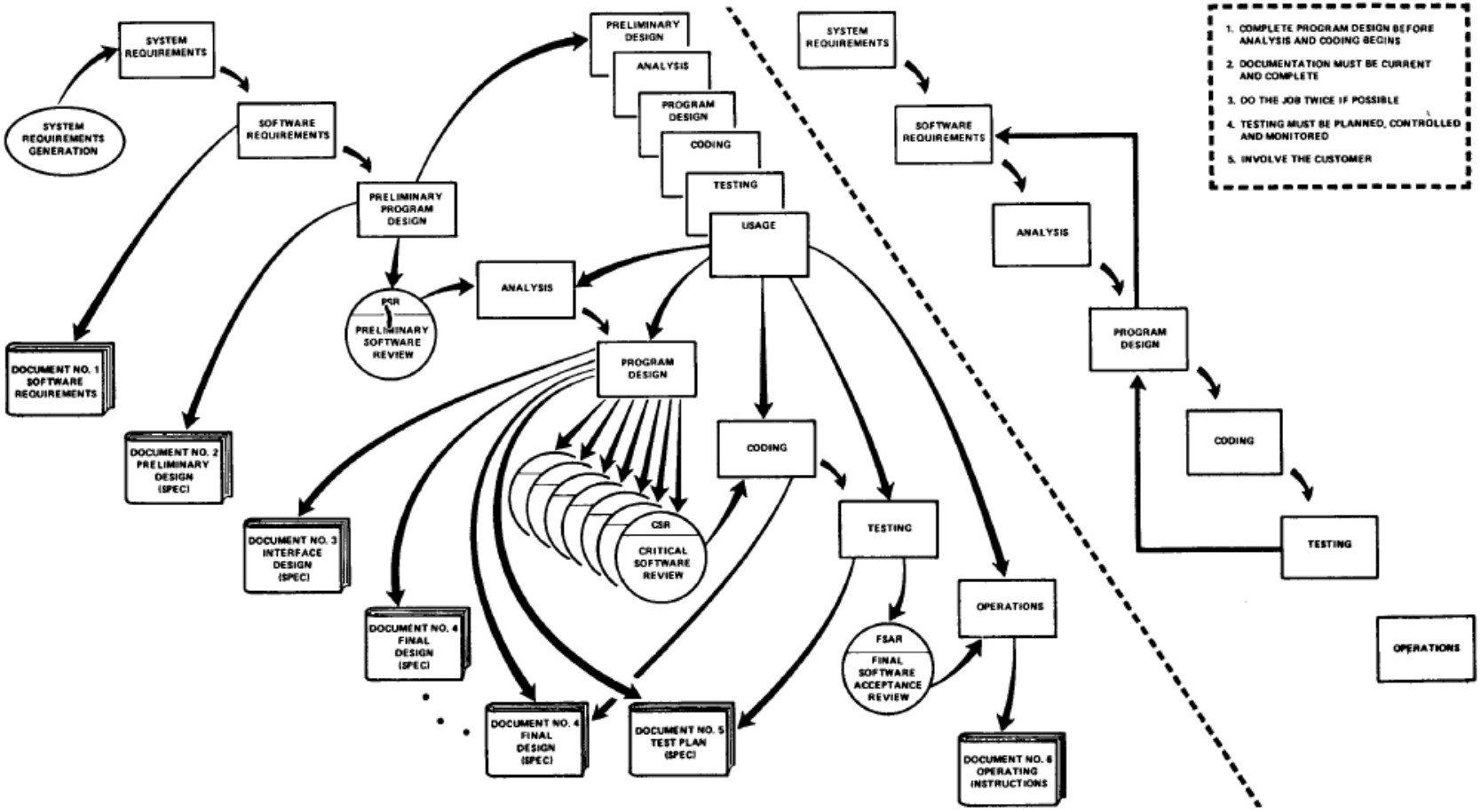


The Waterfall is All Wet!

- The Waterfall is a Myth - Winston Royce the originator of the Waterfall said "the implementation described ... is risky and invites failure"
- Read the Blackboard attachment for M02 - "The remainder of this paper [SIC] presents five additional features that must be added to this basic approach to eliminate most of the development risks."
  1. **PROGRAM DESIGN COMES FIRST -** designer assures that the software will not fail because of storage, timing, and data flux reasons. We now call this architectural design.
  2. **DOCUMENT THE DESIGN** - plan for adequate documentation
  3. **DO IT TWICE** - "the version finally delivered to the customer ... is actually the second version insofar as critical design/operations areas are concerned"
  4. **PLAN, CONTROL AND MONITOR TESTING** - "Without question the biggest user of project resources, whether it be manpower, computer time, or management judgment, is the test phase."
  5. **INVOLVE THE CUSTOMER** - "It is important to involve the customer in a formal way so that he has committed himself at earlier points before final delivery

The Waterfall is All Wet! (cont.)

- This is Winston Royce's Waterfall



The Waterfall is All Wet! (cont.)

- Almost everyone who self-confesses to use the Waterfall actually uses the V-model (discussed later)
- No one in their right mind would use a Waterfall!
- Almost everyone also uses either an incremental or iterative approach - no one makes a single delivery!
- The Waterfall in its correct usage is only a process concept - this is exactly what Winston Royce prescribed it as.
- The next few slides are from the book - you need to know them for the quiz but also know the real usage of the waterfall.

Merits of the Waterfall Model

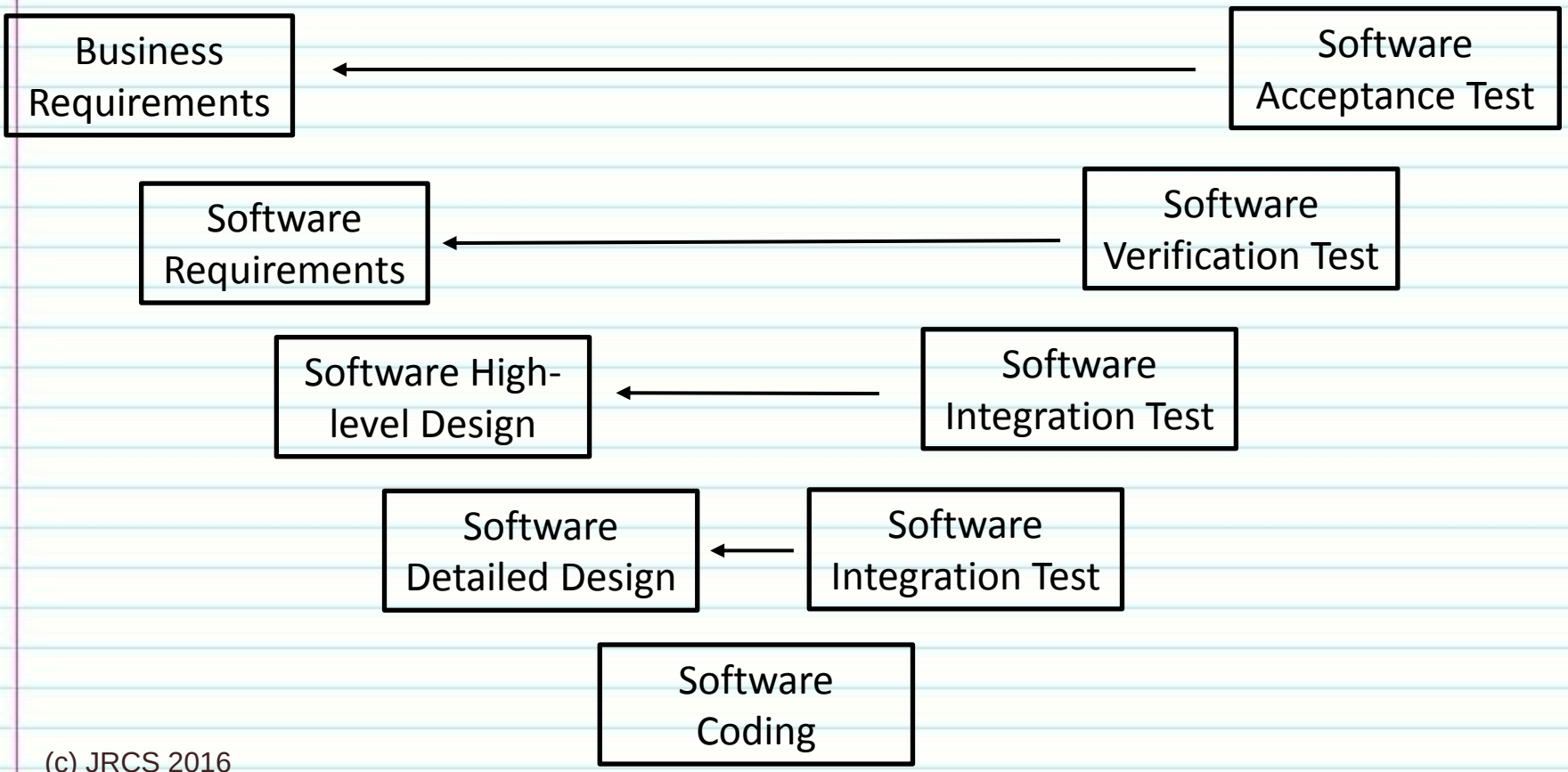
- The simple, straight sequence of phases of the waterfall simplifies project management.
- It supports function-oriented project organization:
  - Each project is carried out by a pipeline of functional teams.
  - Each functional team is specialized in one function such as requirements analysis, design, implementation, integration and testing, and so forth.

Problems with the Waterfall Model

- It is inflexible to requirements change.
- The long development duration means the system is outdated when it is delivered.
- Users cannot experiment with the system to provide early feedback.
- The customer has to wait until the entire system is implemented and deployed to reap the benefits.
- The customer may lose the entire investment if the project fails.

The V-Model

- The V-model places test activities in parallel to development activities - test and development activities occur at the same time - this is needed!
- Most people use an incremental V-model to avoid the at the end of the life-cycle customer delivery issue



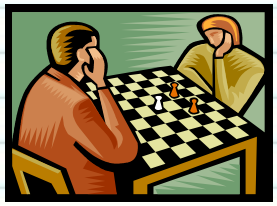
Properties of Problems

- A wicked problem is one that is very difficult or impossible to solve because of incomplete, contradictory, and changing requirements that are often difficult to recognize.
  - Do we see this in Software?
- The use of term "wicked" here has come to denote resistance to solution, rather than a reference to evil
- The effort to solve one aspect of a wicked problem may reveal or create other problems due to its complex interdependencies
- DeGrace and Stahl (1990) described the concept of wicked problems as it relates to software development.
- Later, others pointed out that software development shares many properties with other design approaches (specifically that people, process, and technology-has to be considered in the solution), and have incorporated
- Rittel proposed a number of approaches to deal with the wicked problem but his preferred solution was to engage all stakeholders in the solution

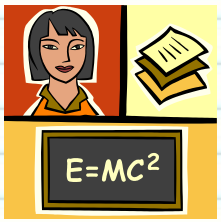
Properties of a Tame Problem

1) A tame problem can be completely specified.
2) For a tame problem, the specification and the solution can be separated.
3) For tame problems, there are stopping rules.
4) A solution to a tame problem can be evaluated in terms of correct or wrong.
5) Each step of the problem-solving process has a finite number of possible moves.
6) These is a definite chain of cause-effect reasoning.
7) The solution can be tested immediately; once tested, it remains correct forever.
8) The solution can be adapted for solving similar problems.
9) The solution process is a scientific process.
10) If the problem is not solved, simply try again.

Examples of Tame Problem



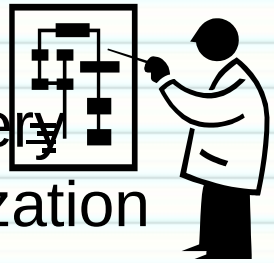
Chess playing



Math problems

Operations research

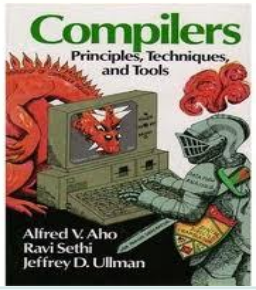
Many computer science problems



Query optimization

Operating systems

AI problems



Compiler construction

Why are these tame problems?

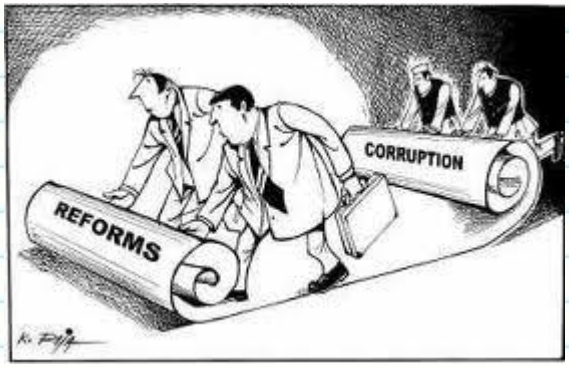
Software Development Is a Wicked Problem

1) A wicked problem does not have a definite formulation.
2) The specification and solution cannot be separated.
3) There is no stopping rule – you can always do it better.
4) The solutions can only be evaluated in terms of good or bad, and the judgment is usually subjective.
5) Each step of the problem-solving process has an infinite number of choices – everything goes as a matter of principle.
6) Cause-effect reasoning is premise-based, leading to varying actions, but hard to tell which one is the best.
7) The solution is subject to life-long testing.
8) Every wicked problem is unique.
9) The solution process is a political process.
10) The problem-solver has no right to be wrong because the consequence is disastrous.

Examples of Wicked Problem

Urban planning

National policy making



Economic reforms

Why are these wicked problems?

Application software development

Software Process Models

- V-Model
- Prototyping Process Model
- Evolutionary Process Model
- Spiral Process Model
- Unified Process Model
- Personal Software Process Model
- Team Software Process Model
- Agile Process Models

Prototyping Process Model

- Prototypes of the software system are constructed to:
  - acquire and validate requirements
  - assess the feasibility of the project and/or the feasibility of the requirements and constraints
- Simple prototypes as well as sophisticated prototypes are used, depending on the needs of the project.
- Prototypes are classified into throwaway prototypes and evolutionary prototypes.

Evolutionary Prototyping Model

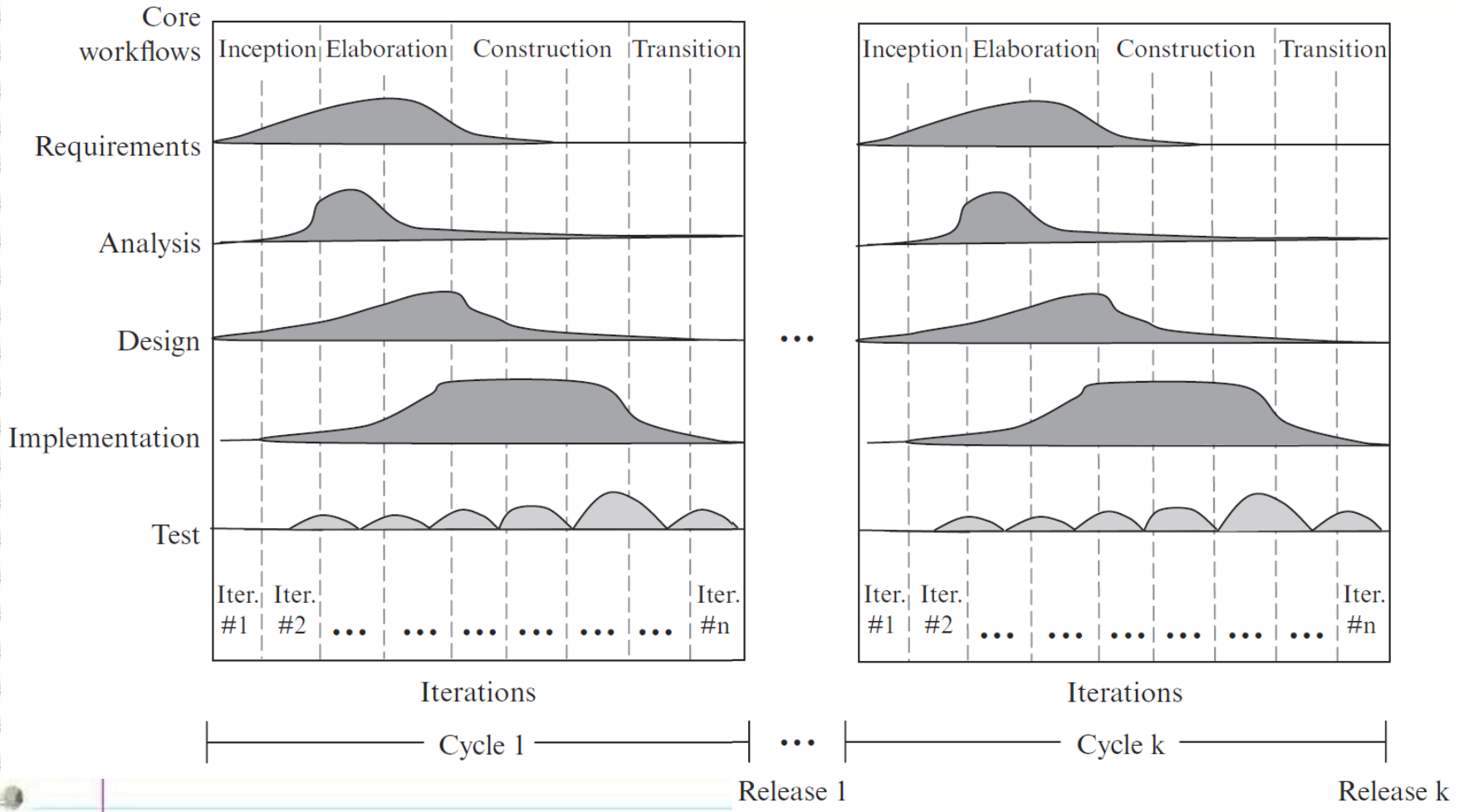
- Throwaway prototypes waste time and effort.
- Evolutionary prototyping model lets the prototype evolve into the production system.
- It is most suited for the development of exploratory types of systems such as intelligent systems, research software, and systems that actively interact with and control the environment.
- It is not suitable for projects that require a predictable schedule of progress.

Spiral Process Model

It begins here.

If risks remains
{plan next phase
conduct prototyping }
else if risks resolved
{proceed as waterfall}
else if prototype works & robust
{proceed as evo. model}

Rational Unified Process (RUP)



Rational Unified Process (RUP)

- **Inception** consists of the first 1-2 iterations. It produces a simplified use case model, a tentative architecture, and a project plan.
- **Elaboration** consists of the next N iterations. It produces the architectural design and implements the most critical use cases.
- **Construction**, during which remaining use cases are iteratively implemented and integrated into the system.
- **Transition,** during which the system is deployed, users are trained, and defects are corrected.

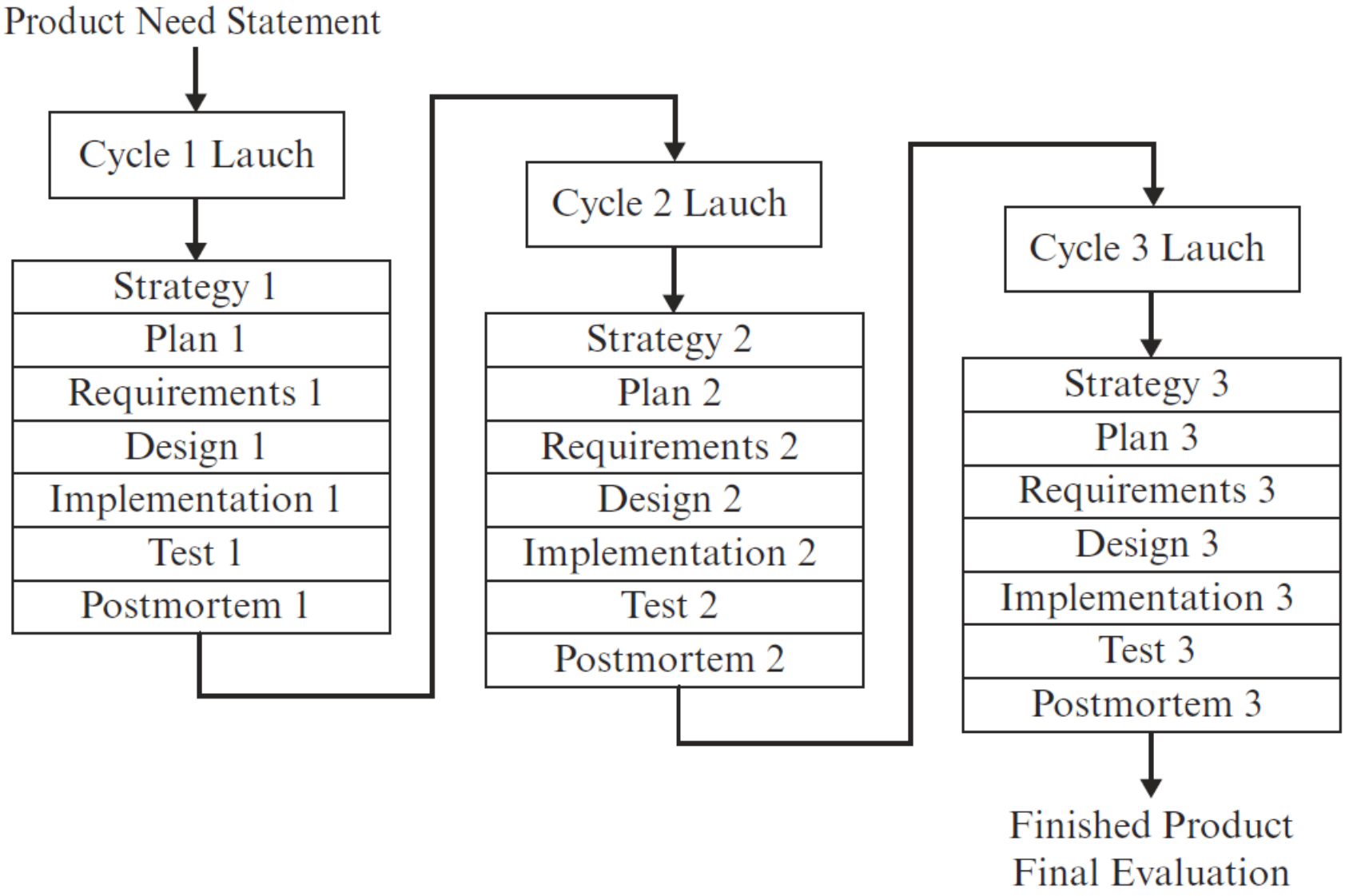
Personal Software Process Model

- PSP is a comprehensive framework for training software engineers.
- It consists of scripts, forms, standards and guidelines used in the training.
- It helps the software engineer identify areas for improvement.
- It prepares the software engineer to work in a team project.

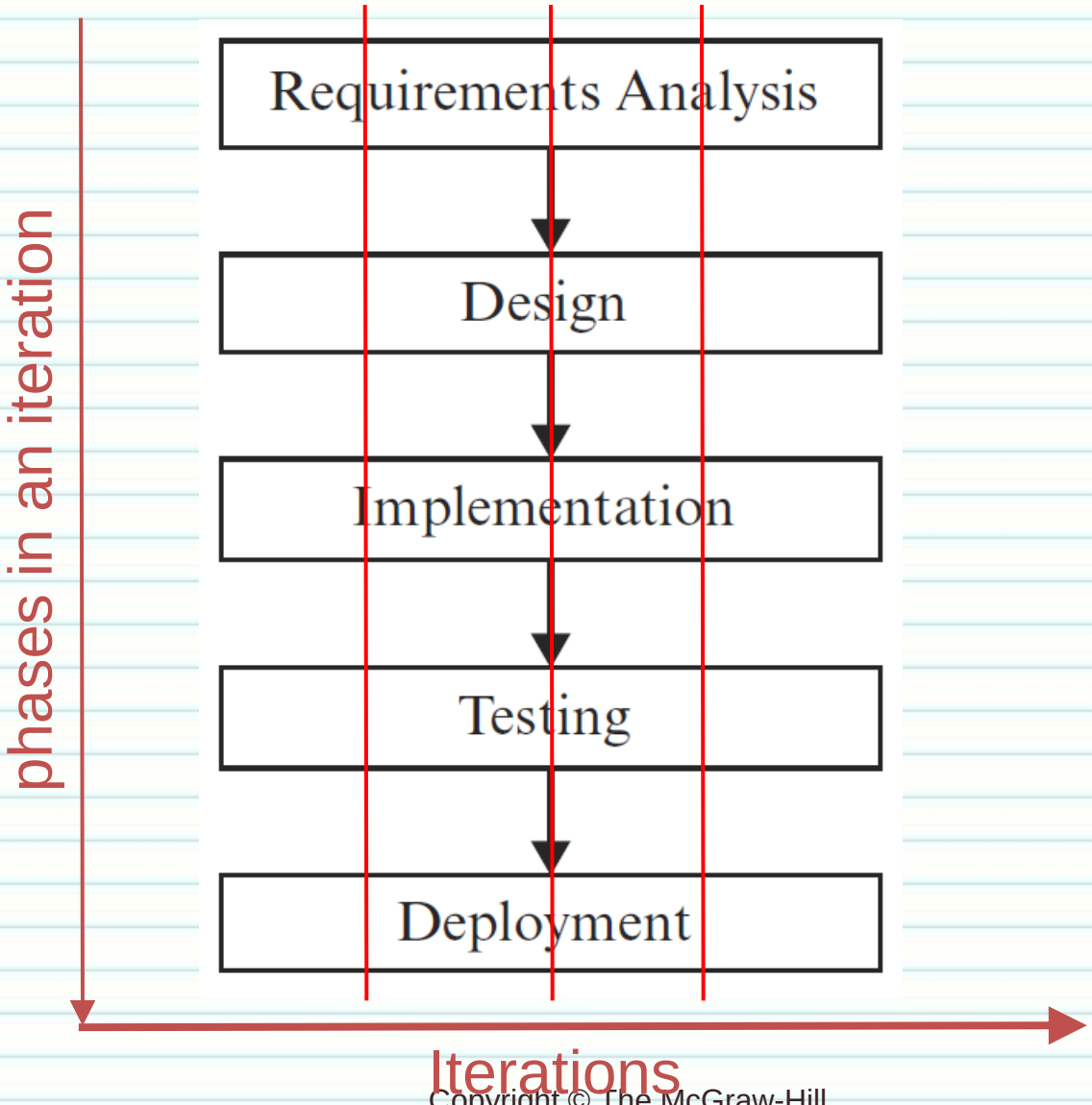
Personal Software Process Model

- PSP framework consists of a series of predefined processes:
  - PSP0 & PSP0.1. These introduce process discipline and measurement including baseline, time recording, defect recording, defect type, coding standards, and process improvement.
  - PSP1 & PSP1.1. These introduce size estimation, planning and scheduling.
  - PSP2 & PSP2.1. These introduce quality management and design including code review and design review.
  - PSP3.0. It guides component level development.

Team Software Process



Agile Process Models



Disciplined Agile Delivery

- According to a Sep/Oct 2016 IEEE software article, Agile Unified Process is quite heavily in the industry, but support of the AUP has ended
- Much of the former AUP work has evolved into the Disciplined Agile Delivery process - this is another 2nd generation Agile process
- DAD incorporates practices used by Agile (scrum/lean) and together with processes from RUP
- It consists of the following phases: Inception, Construction, and Transition
- Roles look familiar to Agile (scrum)
- It was developed to fills in the process gaps that are specifically ignored by Scrum, and attempts to work with the organizations enterprise-level system

Process and Methodology

- Methodology is often confused with process.
- A process is a set of inter-related activities to be carried out to construct something (usually a system).
- The activities are carried out in phases.
- Each phase produces some products which are the input of the next phase.
- The completion of each phase establishes a milestone.
- A methodology implements a process or a phase of a process.

Methodology

- A methodology is a cook-book for performing a task. It describes
  - steps to accomplish a series of subtasks
  - input and output of each step
  - representations of input and output
  - entrance and exit conditions for each step
  - procedures for carrying out each step
  - methods and techniques used by each step
  - relationships, or control flow and data flow between the steps

Process and Methodology

Process

- Defines a framework of phased activities
- Specifies phases of WHAT
- Does not dictate representations of artifacts
- It is paradigm-independent
- A phase can be realized by different methodologies.

Examples

Waterfall, spiral, prototyping, unified, and agile processes

Methodology

- Defines steps to carry out phases of a process
- Describes steps of HOW
- Defines representations of artifacts (e.g., UML)
- It is paradigm-dependent
- Steps describe procedures, techniques & guidelines

Examples

Structured analysis/structured design (SA/SD), Object Modeling Technique (OMT), Scrum, DSDM, FDD, XP, and Crystal Orange

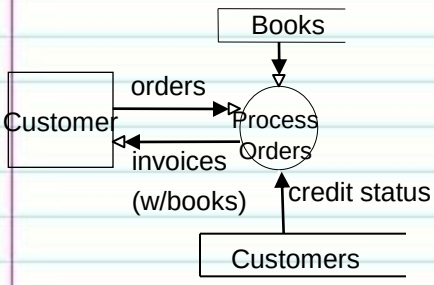
Software Paradigm

- A software paradigm is a style of software development that constitutes a way of viewing the reality.
- Examples:
  - procedural paradigm
  - OO paradigm, and
  - data-oriented paradigm

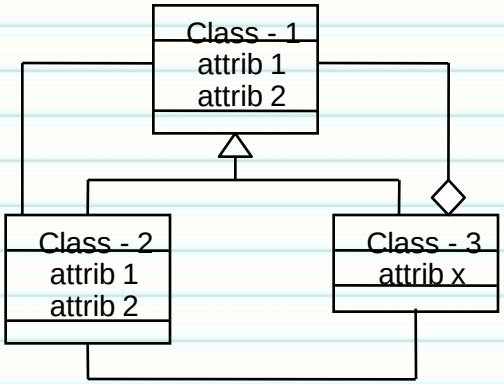
Three Paradigms in History

- Procedural paradigm views the world and system as:
  - a network of processes
  - a process is refined by lower level processes
  - basic building blocks and starting point are processes
- OO paradigm views the world and system as:
  - interrelated and interacting objects
  - basic building blocks are objects
- Data-oriented paradigm views the world and system as:
  - interrelated data entities, processed by transactions
  - basic building blocks are data entities and relationships

Paradigm and Methodology

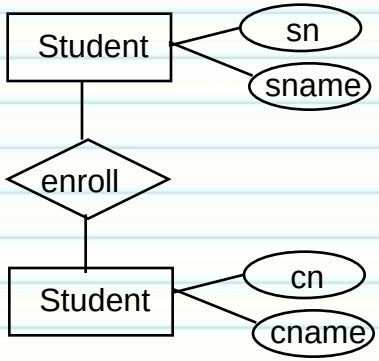


Structured Analysis

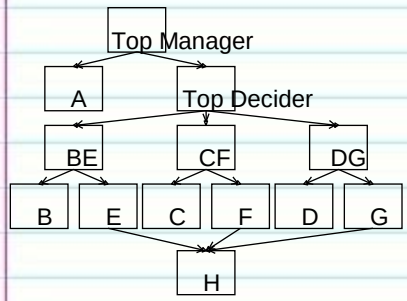


Domain model

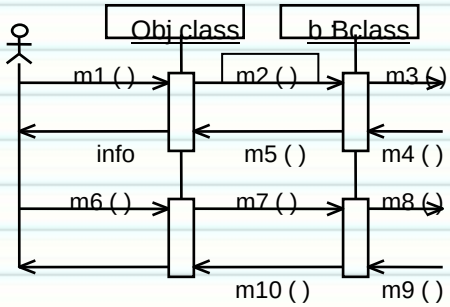
Object-Oriented Analysis



Data-Oriented Analysis

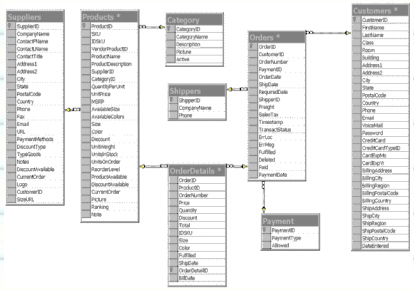


Structured Design



Sequence diagram

Object-Oriented Design



Data-Oriented Design

Structured Programming

Object-Oriented Programming

Programming in 4GL (e.g. SQL)

Procedural Paradigm

OO Paradigm

Data-Oriented Paradigm

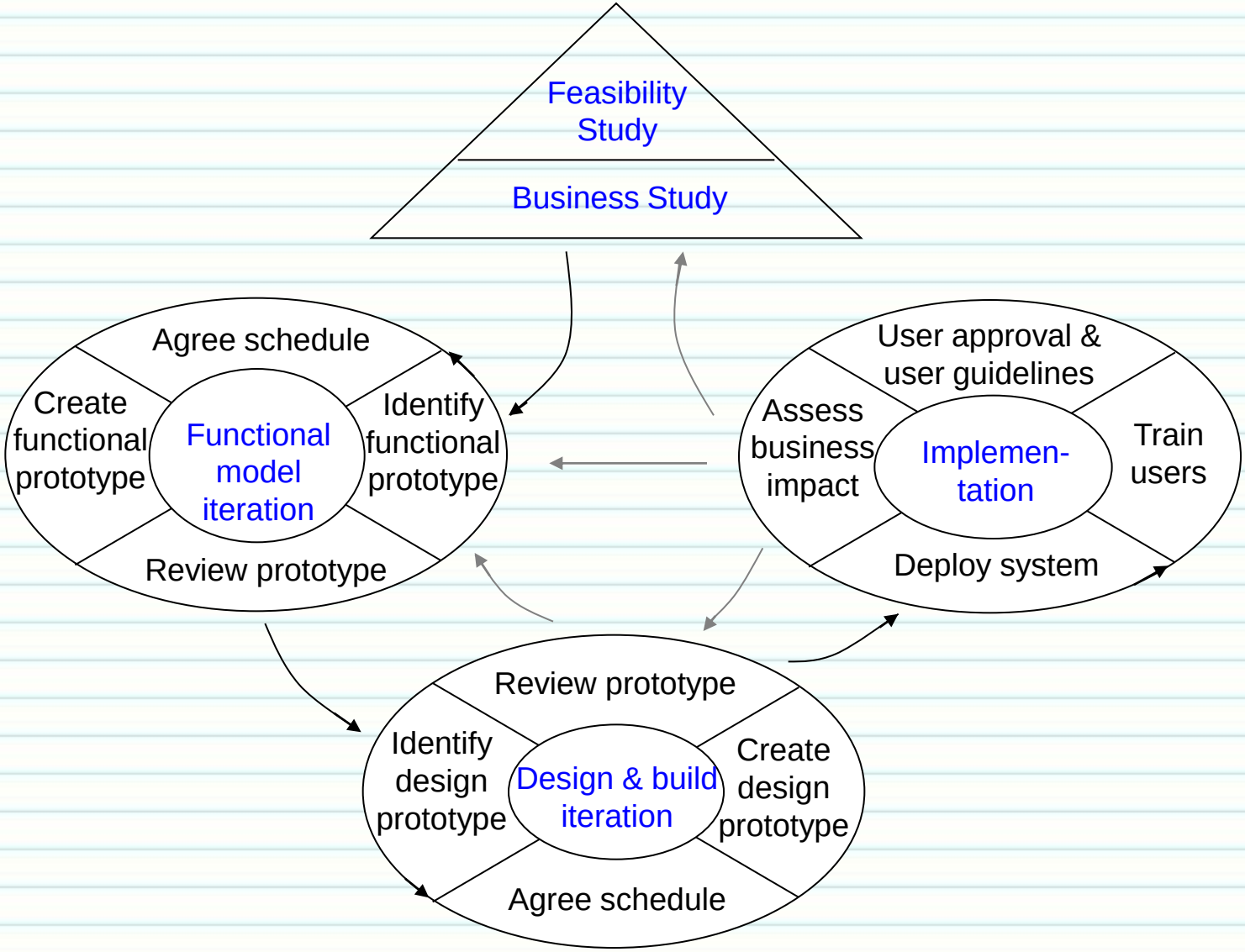
Some Well-Known Agile Methods

- Dynamic Systems Development Method (DSDM)
- Feature Driven Development (FDD)
- Scrum
- Extreme Programming (XP)
- Crystal Clear
- Lean Development

DSDM Unique Key Features

- It is a framework that works with Rational Unified Process and XP.
- It is based on the 80-20 principle.
- It is suitable for agile as well as plan-driven projects.

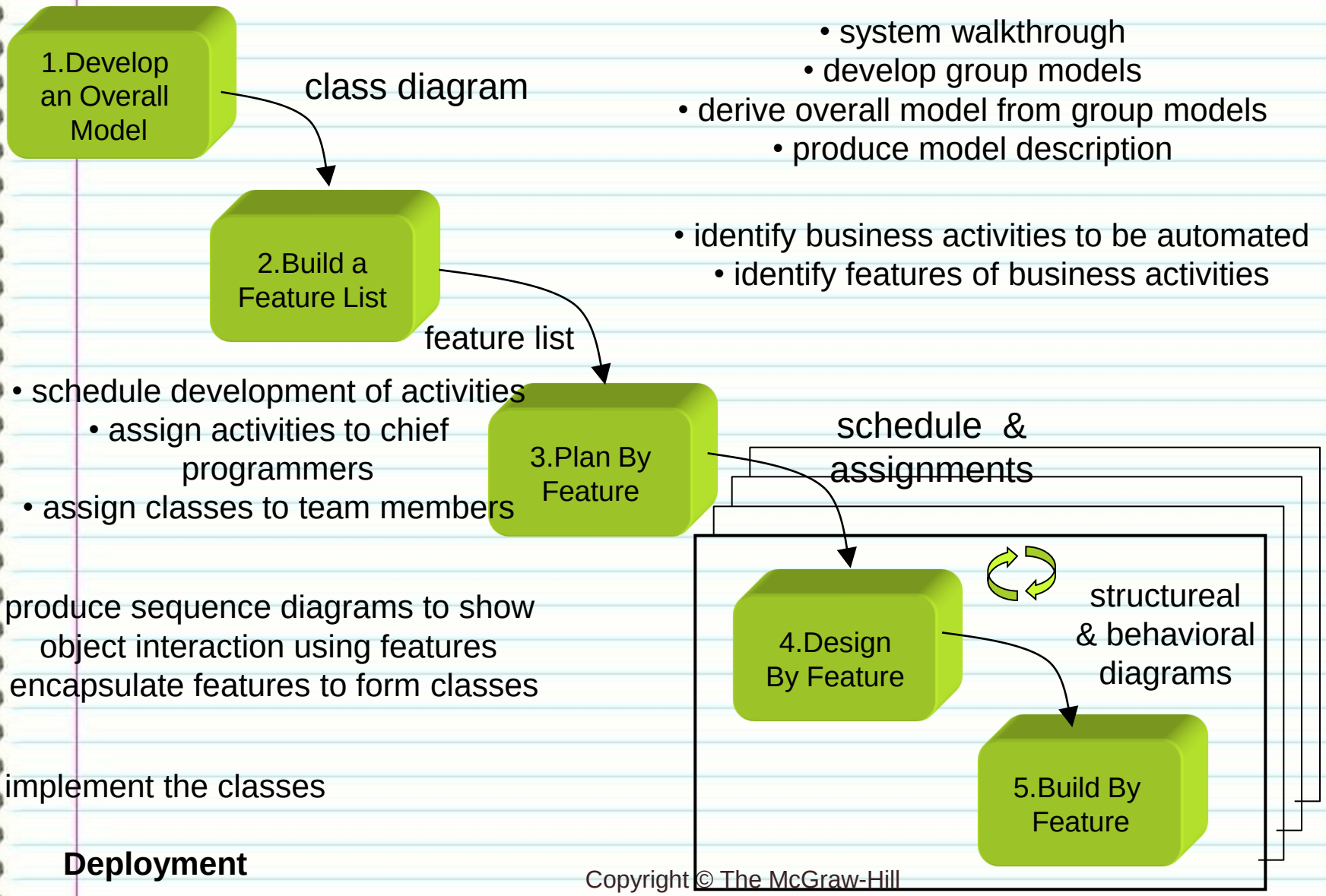
DSDM Lifecycle Activities



Feature Driven Development (FDD)

- Unique key features:
  - feature driven and model driven
  - configuration management, review and inspection, and regular builds
  - suitable for agile or plan-driven projects

FDD Lifecycle Activities



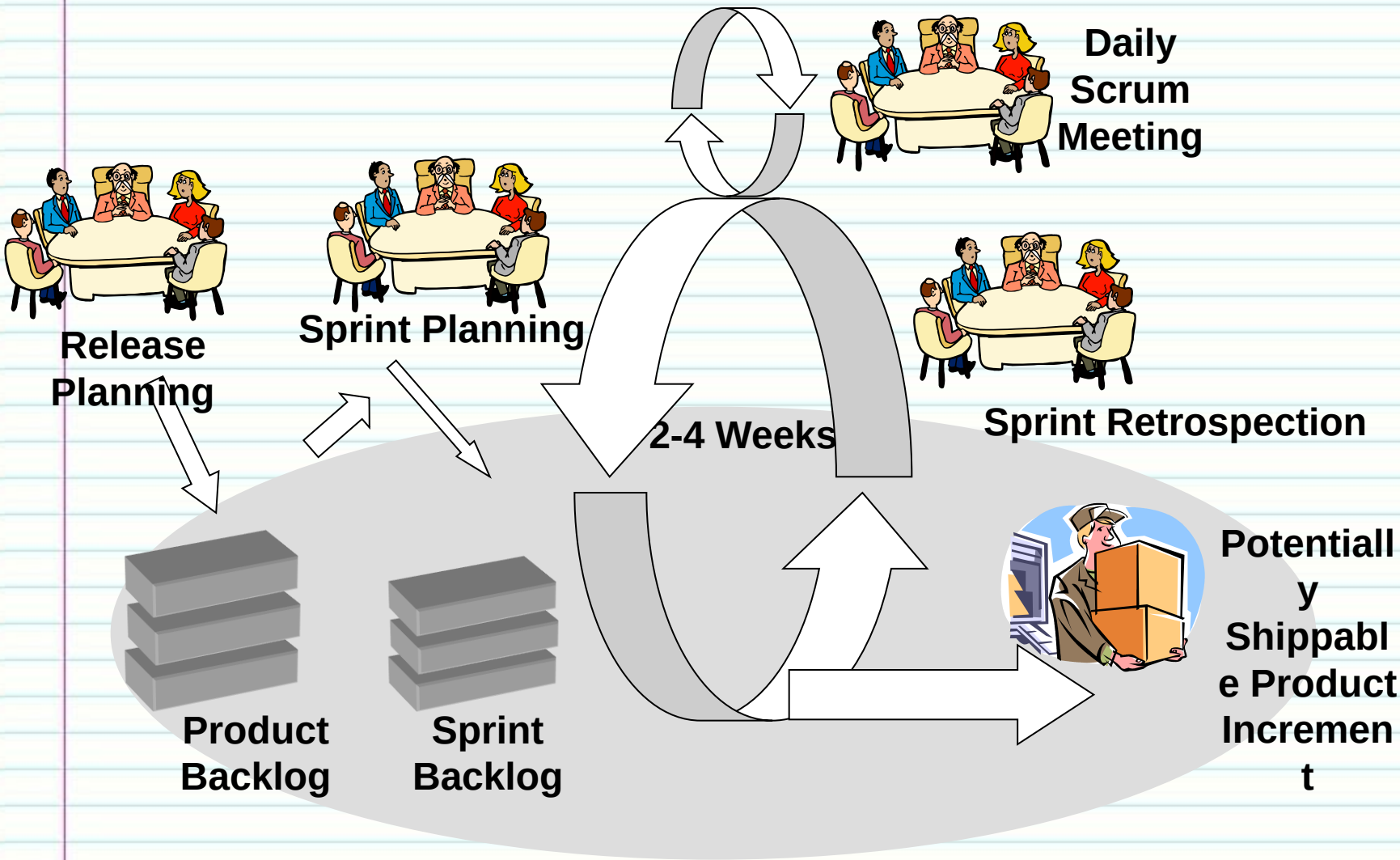
Scrum

- Unique key features:
  - include Scrum Master, Product Owner, and Team roles
  - 15 minute daily status meeting to improve communication
  - team retrospect to improve process

Scrum Lifecycle Activities

- Release planning meeting
  - identify and prioritize requirements, called product backlog
  - identify top priority requirements to be delivered within an increment, called a sprint
  - identify sprint development activities
- Sprint iteration
  - sprint planning meeting to determine what and how to build next
  - daily Scrum meeting to exchange status
- Sprint review meeting
  - increment demo
  - team retrospection
- Deployment

Scrum Process



Extreme Programming (XP)

- Unique Key Features:
  - Anyone can change any code anywhere at any time
  - Integration and build many times a day whenever a task is completed
  - Work ≤ 40 hours a week

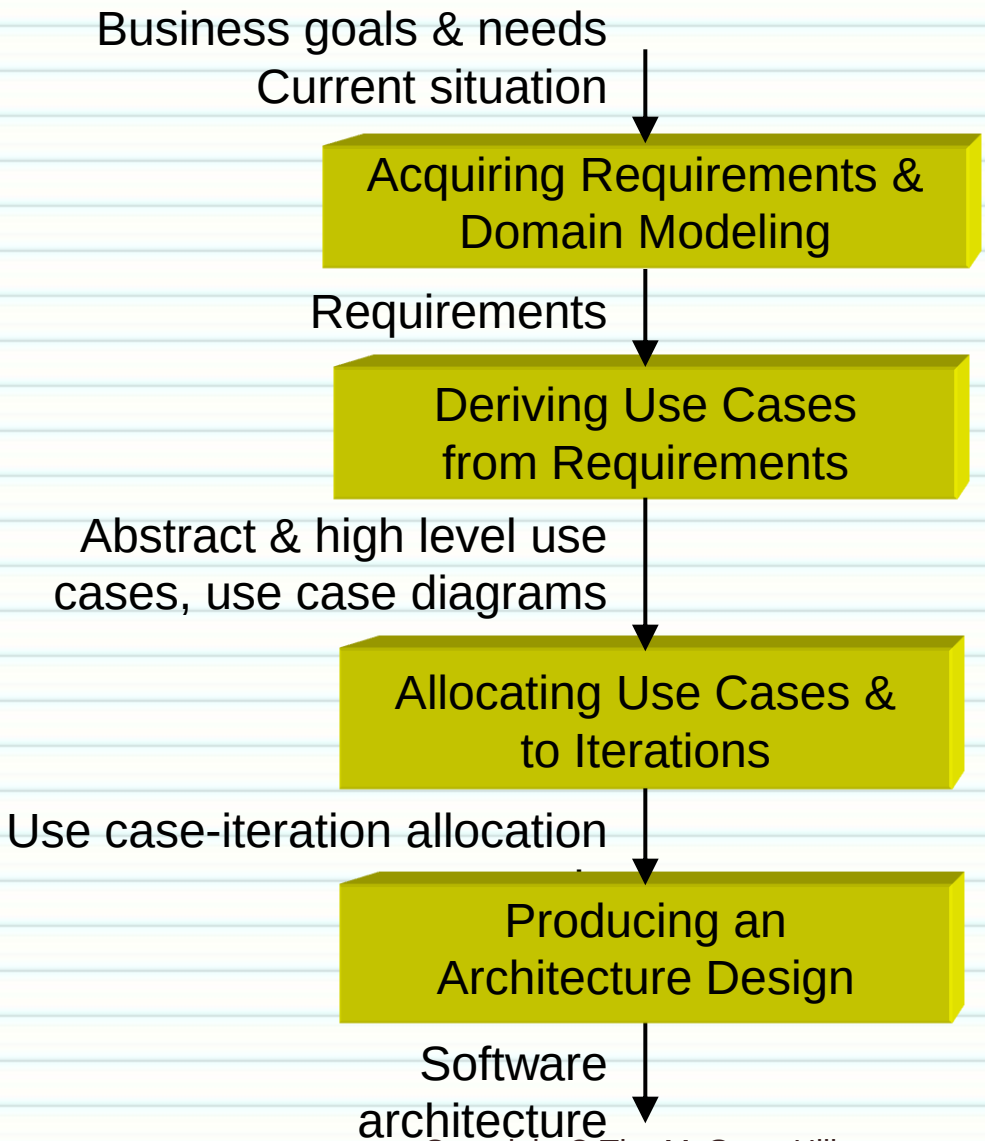
XP Lifecycle Activities

Exploration	1. Collect information about the application 2. Conduct feasibility study
Planning	1. Determine the stories for the next release 2. Plan for the next release
Iteration to 1st Release	1. Define/modify architecture 2. Select and implement stories for each iteration 3. Perform functional tests by customer
Productionizing	1. Evaluate and improve system performance 2. Certify and test system for production use
Maintenance	1. Improve the current release 2. Repeat process with each new release
Death	1. Produce system documentation if project is done, or 2. Replace system if maintenance is too costly

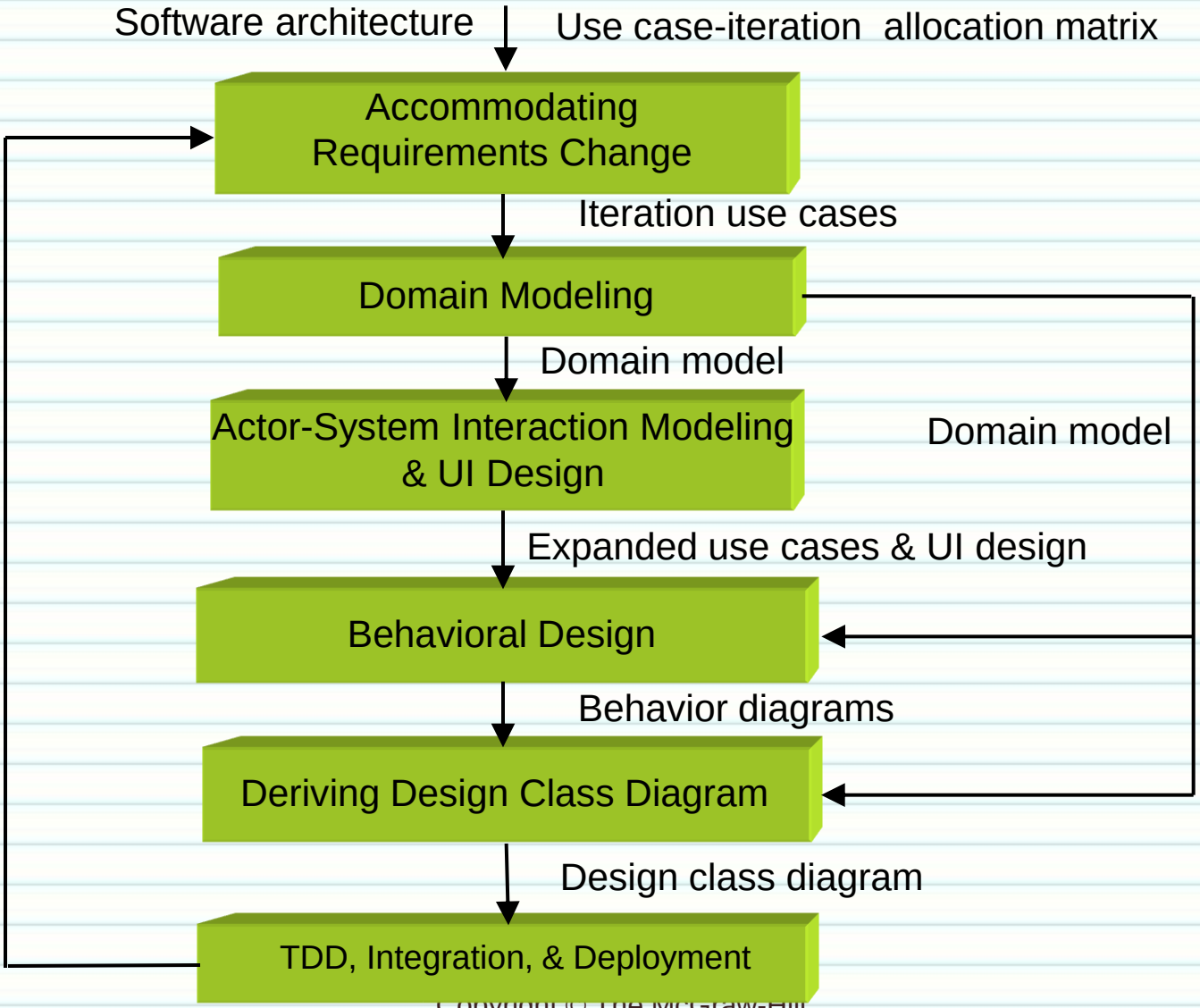
The Methodology Presented in This Book

- It is designed for beginners as well as seasoned developers.
- It is aimed at educating software architects and systems analysts.
- It can be applied to agile as well as plan-driven projects. It has been applied to sponsored as well as industrial projects.
- Team members should work together from project start to completion.
- Many students continue practicing the methodology after graduation.

Methodology Overview – Planning Phase



Methodology Overview – Iterative Phase

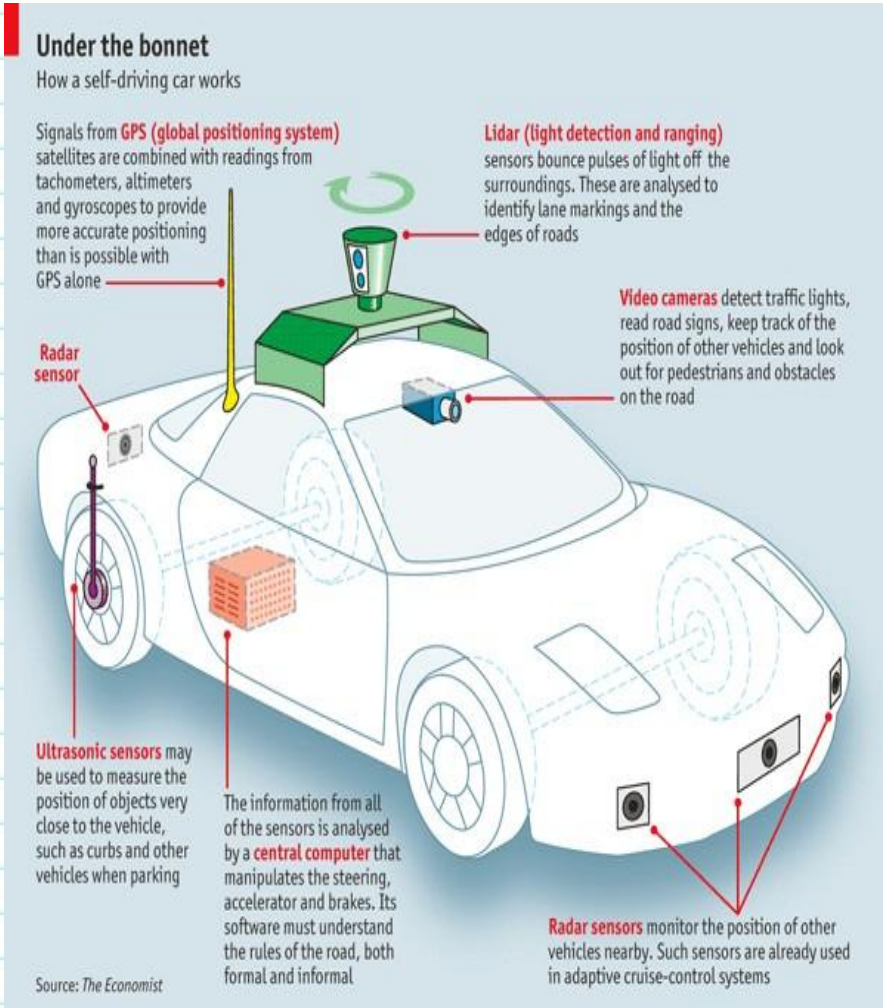


Chapter 3: System Engineering

Chapter 3 - Key Takeaway Points

- System engineering is a multidisciplinary approach to systems development.
- System engineering defines the system requirements and constraints, allocates the requirements to the hardware, software, and human subsystems, and integrates these subsystems to form the system.
- Software engineering is a part of system engineering.

Basic Physical Ecosystem of an Autonomous Vehicle

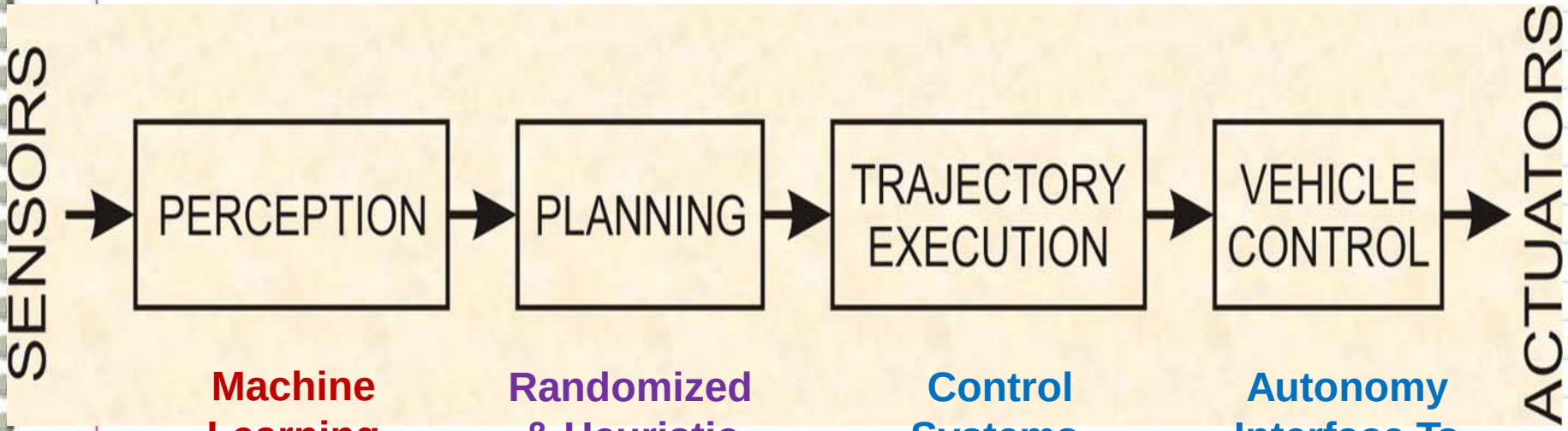


Source: The Economist, "How does a self-driving car work?"

- Global Positioning System (GPS)
- Light Detection and Ranging (LIDAR)
- Cameras (Video)
- Ultrasonic Sensors
- Central Computer
- Radar Sensors
- Dedicated Short-Range Communications-Based Receiver (not pictured)

Validating an Autonomous Vehicle Pipeline

Carnegie Mellon University



Machine Learning Based Approaches

➔ ???

Randomized & Heuristic Algorithms

➔ Run-Time Safety Envelopes

➔ Doer/Checker Architecture

Control Systems

➔ Control Software Validation

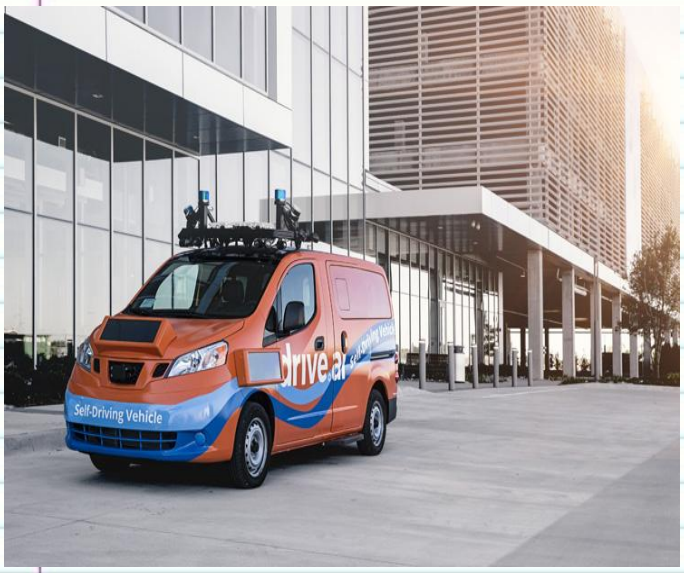
➔ Doer/Checker Architecture

Autonomy Interface To Vehicle

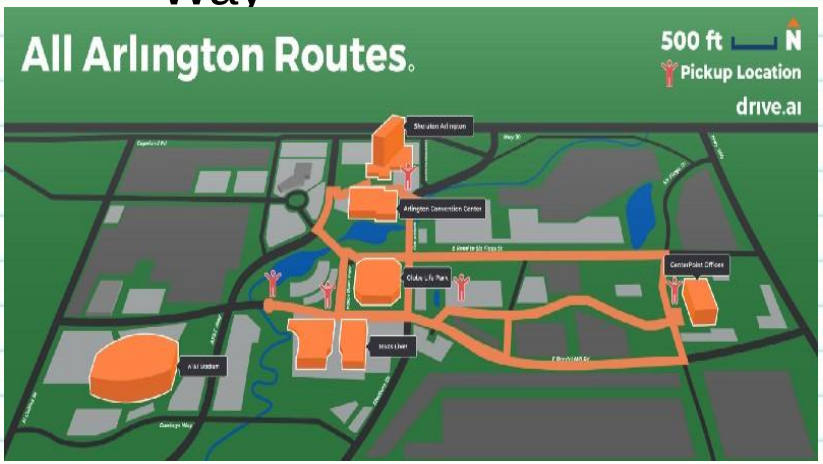
➔ Traditional Software Validation

Perception presents a uniquely difficult assurance challenge

Arlington is Taking a Leading Step



Dallas Cowboys' AT&T Stadium, the Texas Rangers' Globe Life Park and Six Flags Over Texas. Those roads could include Randol Mill Road, Road to Six Flags, Nolan Ryan Expressway, Ballpark Way, Stadium Drive and AT&T Way

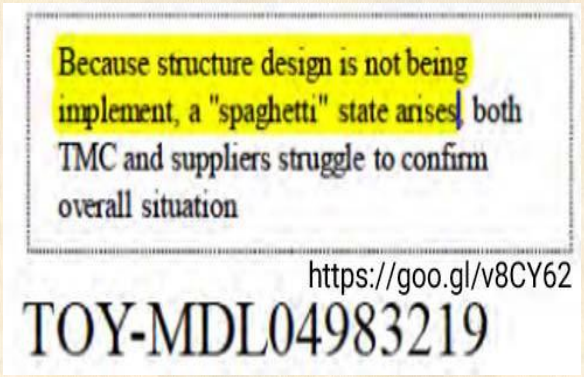


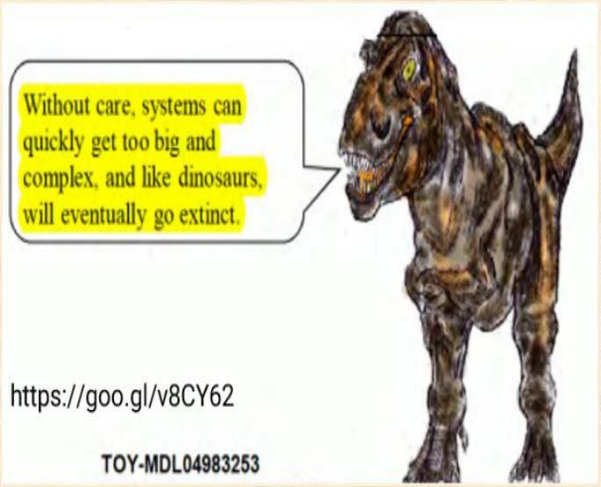
Initially, the vans will have two human operators aboard — one who will serve as a "safety driver" who will sit in the driver's seat in case an emergency requires him or her to take control of the vehicle. The other employee will interact with passengers and answer questions about the experience. The vans will travel in a geo-fenced area at up to 35 miles per hour.

Automotive Engine Control Code Quality

- **$$$B Toyota Unintended Acceleration Fiasco: 2010-...?**
  - "Reckless disregard" Jury verdict in one death
  - Hundreds of death and injury settlements (still ongoing)
  - **Code quality overview (256K SLOC in C):**
    - Throttle angle routine: MCC of 146 (MCC above 50 is "untestable")
      - 80,000 MISRA C violations
        - Uninitialized variables, condition side effects, unsafe type casting
          - 2272 global variable declaration type mismatches
            - 10,000 read/write global variables
    - Missing concurrency locks; confirmed race condition bug
    - Recursion with essentially full stack; no stack protection
      - Ineffective watchdog (kicked from HW timer)
    - No configuration management, no bug tracking, real time scheduling overload, minimal peer reviews

For more, web search: "Koopman Toyota"

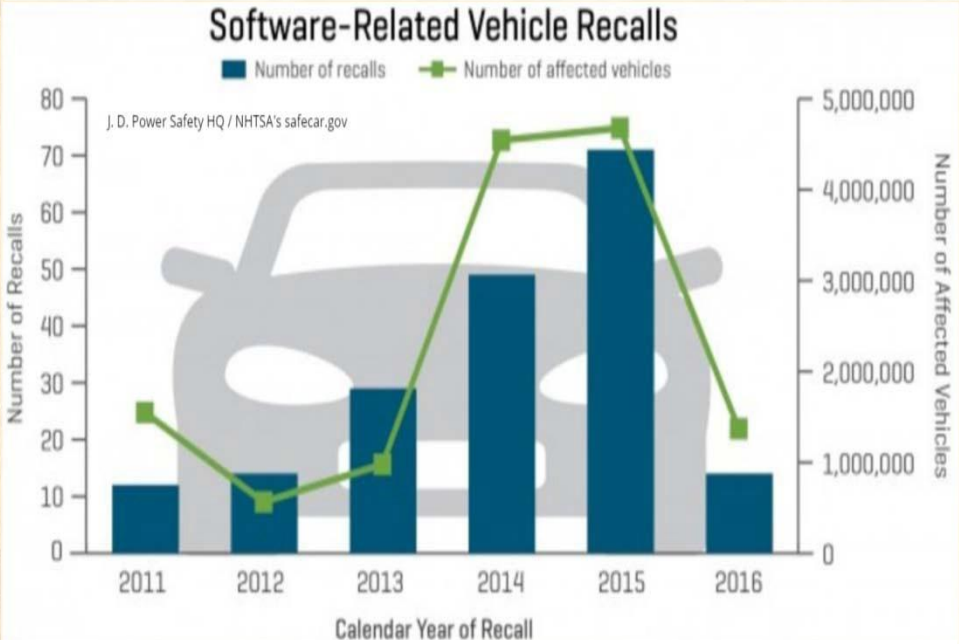




Is Bad Software Eating The World?

SOFTWARE NOW TO BLAME FOR 15 PERCENT OF CAR RECALLS

YOU CAN'T JUST HOLD THE HOME AND LOCK BUTTONS TO SOLVE THIS ONE June 2, 2016



The research firm J.D. Power, through its Safety IQ application, found that there have been 189 distinct software recalls issued over five years—covering more than 13 million vehicles. These weren't merely interface-related issues either; 141 of these presented a higher risk of crashing.

http://www.popsci.com/software-rising-cause-car-recalls

- Software can make or break a company
  - So why is it still so often bad?
- Don't we have tools for that?
  - Automated testing, build tools
    - Static analysis tools
      - Formal methods
  - Better development processes
    - Model-based design
      - CMMI
        - …

Example Software Failures

MyFord Touch problems: Ford to issue upgrade

Glitches in MyFord Touch software that replaces knobs and buttons with a touchscreen have led to plummeting user approval ratings for Ford cars

MyFord Touch has been plagued with software problems PR

Charles Arthur and agencies Monday 7 November 2011 03.51 EST

The motor company Ford has discovered belatedly that touchscreens don't make a great replacement for the knobs and buttons of a dashboard - especially if the touchscreens are plagued with software glitches.

The company says it will send memory sticks to 250,000 customers in the US offering a software upgrades for its glitch-prone MyFord Touch system, which replaces the standard dashboard knobs and buttons with a touchscreen.

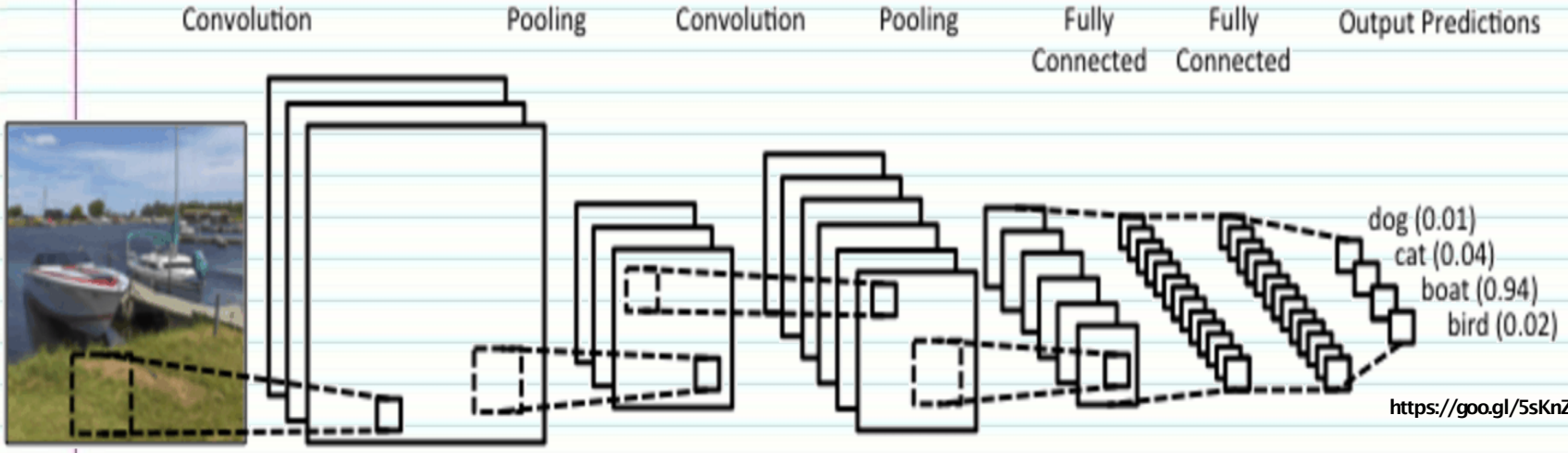
Jaguar recalls 18,000 cars over cruise control software fault

Car system upgrade needed, but no hardware affected

By Leo King | Computerworld UK | Published 10:23, 24 October 11

Some 17,678 vehicles have been recalled, as a result of the potentially dangerous problem. If the fault occurs, cruise control can only be disabled by turning of the ignition while driving - which would mean a loss of some control and in many cars also disables power steering. Braking or pressing the cancel button will not work.

Edge Case Testing: Data Degradation



QuocNet:

Car | Not a Car | *Magnified Difference*

AlexNet:

Bus | *Magnified Difference* | Not a Bus

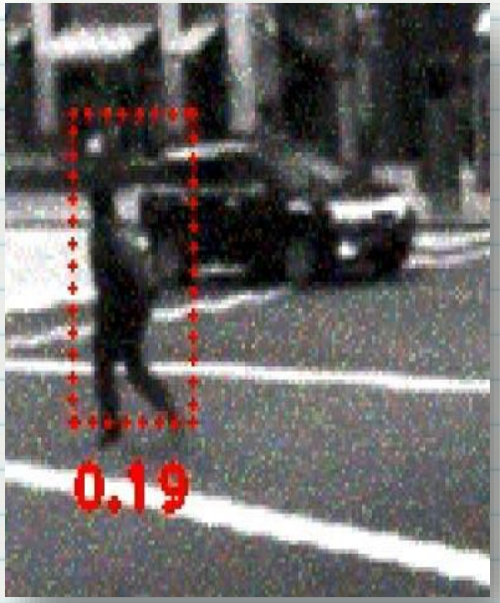
Szegedy, Christian, et al. "Intriguing properties of neural networks." *arXiv preprint arXiv:1312.6199* (2013).

Lessons Learned With Edge Case Testing

- Perception failures are often context-dependent
  - False positives and false negatives are both a problem

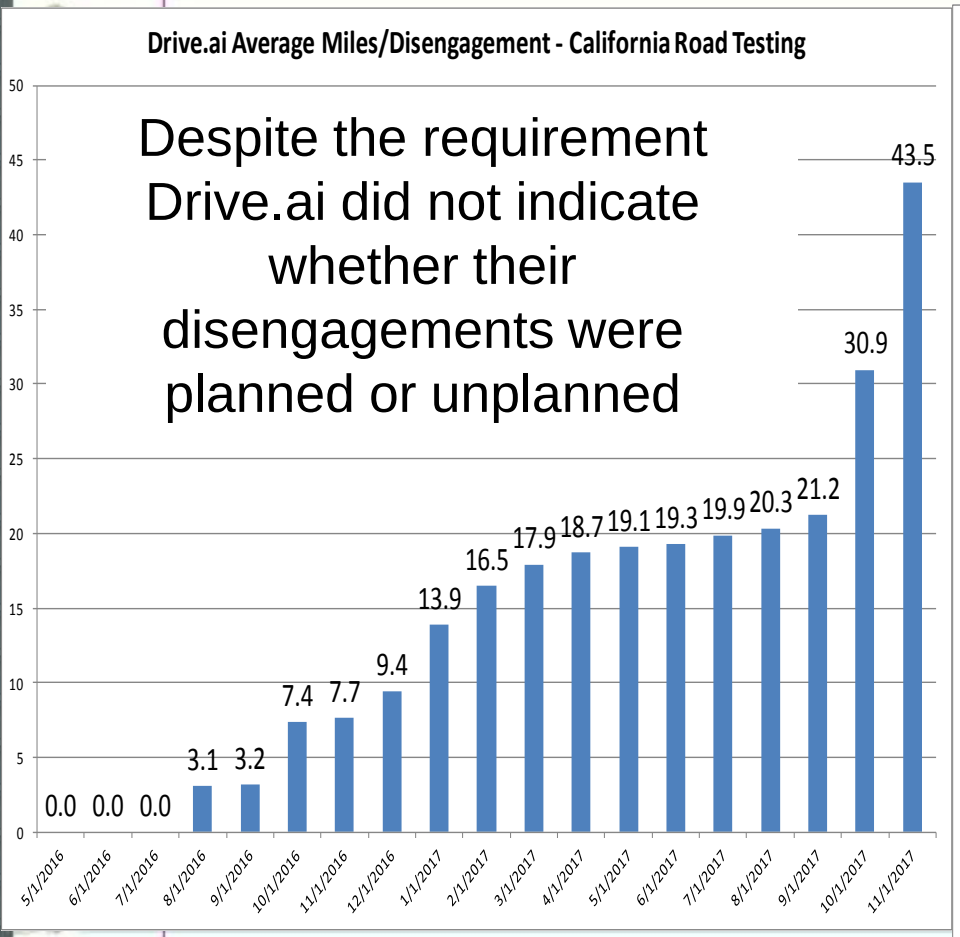
False positive on lane marking
False negative real bicyclist

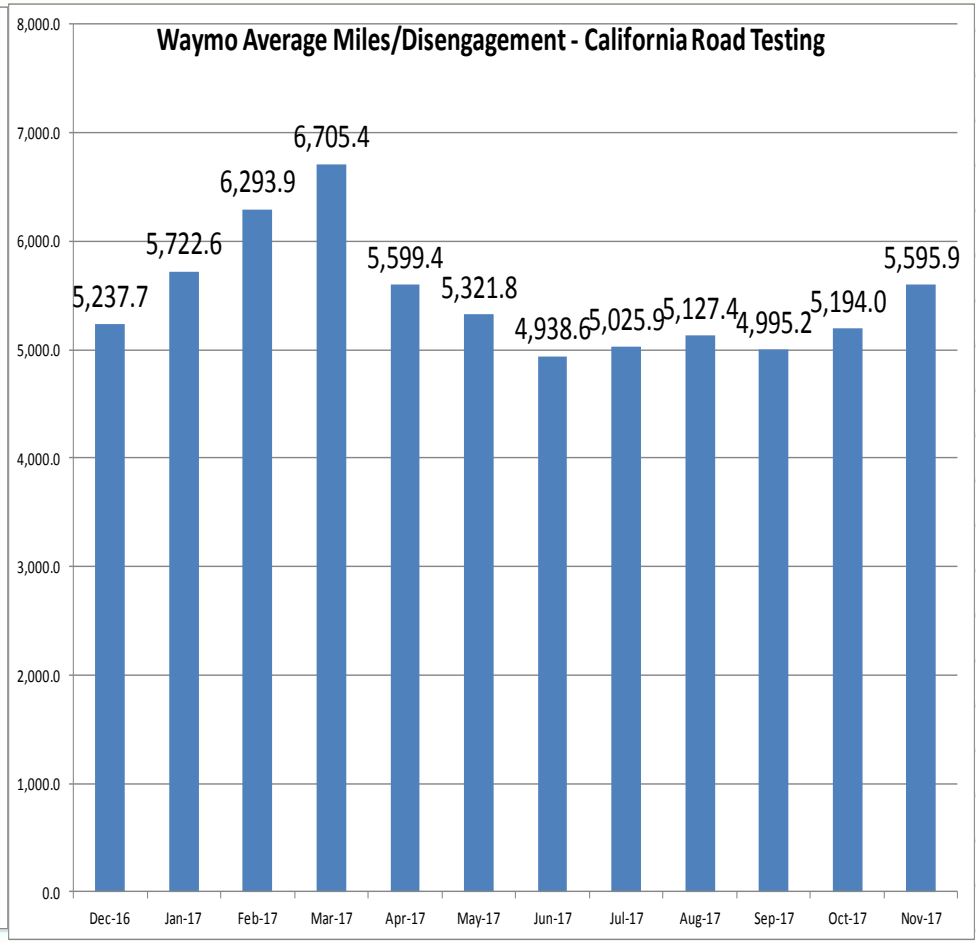
False negative when
person next to light pole



False negative when
in front of dark vehicle

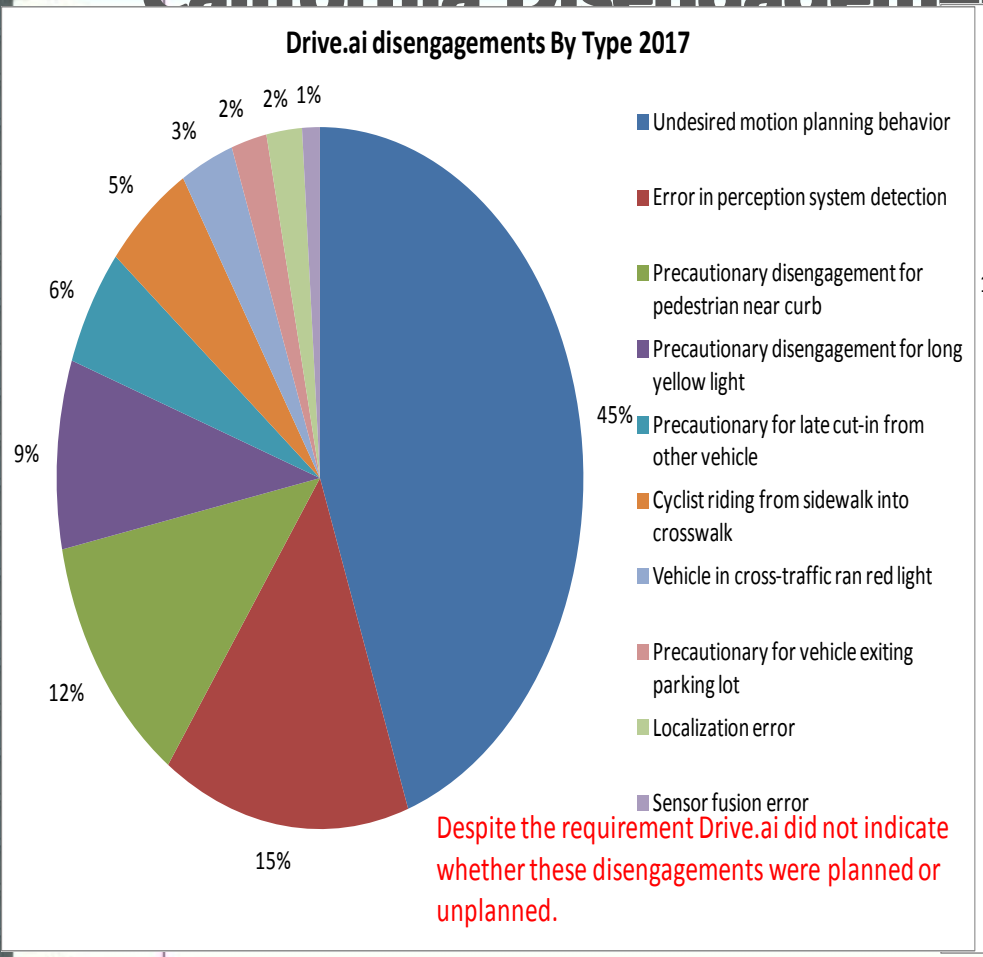
California Disengagement Report Data (cont.)



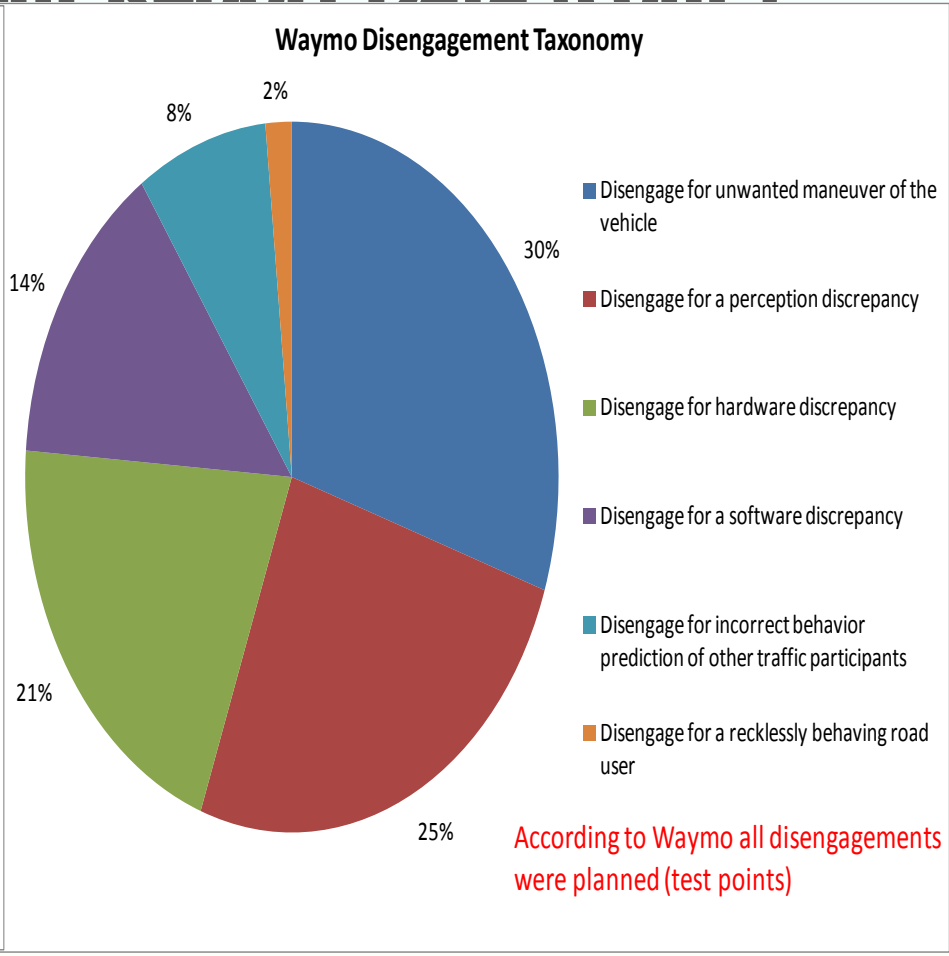


.ai AV systems are in use currently in Arlington with a required Safety driver (for no

California Disengagement Report Data (cont.)



Despite the requirement Drive.ai did not indicate whether these disengagements were planned or unplanned.

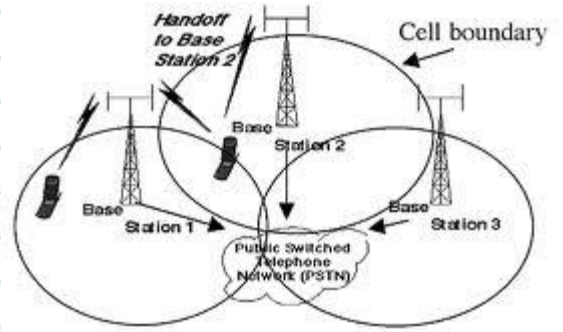


According to Waymo all disengagements were planned (test points)

Drive.ai AV systems are in use currently in Arlington with a required Safety driver (for now!)

Main Characteristics of a System

A system consists of interacting



A system exists in a hierarchy of systems – a system may be a subsystem of another system.

Each system exists in an environment and interacts with the environment.

Systems are ever evolving.

The Requirements Process

The Requirements Process consists of the following steps

A. Requirements Planning (estimating requirements work)
B. Requirements Elicitation (draw-out the requirements)
C. Requirements Analysis (do they work and work together?)
D. Software Requirements Specification (capture requirements)
E. Requirements Validation
F. Requirements Management (requirements will change - they must be managed)
G. Requirements status reporting

Most of the industry is particularly weak in all but D and E above and many are weak here as well.

Questions to discuss:

1. What are typical software estimation measures and how do they apply to software requirements?

What is Requirements Engineering?

- Challenge facing system and software engineers - “How can we ensure that we have specified a system that:
  - Properly meets the customer’s needs
  - Satisfies the customer’s expectations

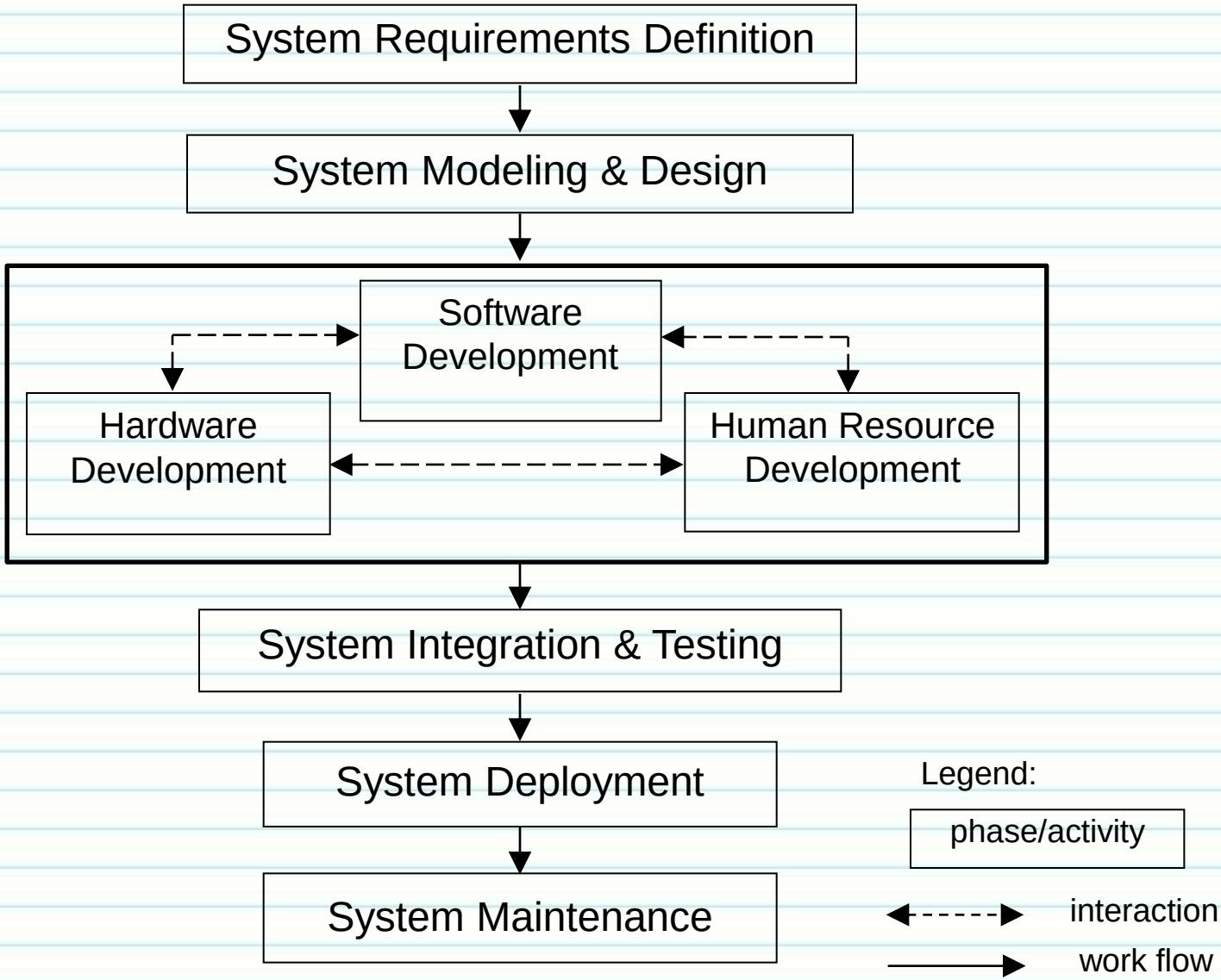
- Requirements engineering provides mechanisms for:
  - Understanding what the customer wants
  - analysing need
  - assessing feasibility
  - negotiating a reasonable solution
  - specifying the solution
  - validating the specification
  - managing the transformation of the requirements into an operational system

What Is System Engineering?

System engineering is characterized by:

- A system engineering process that covers the entire system life cycle.
- A top-down divide-and-conquer approach.
- An interdisciplinary approach to system development.

System Engineering Process



Example of System Requirements

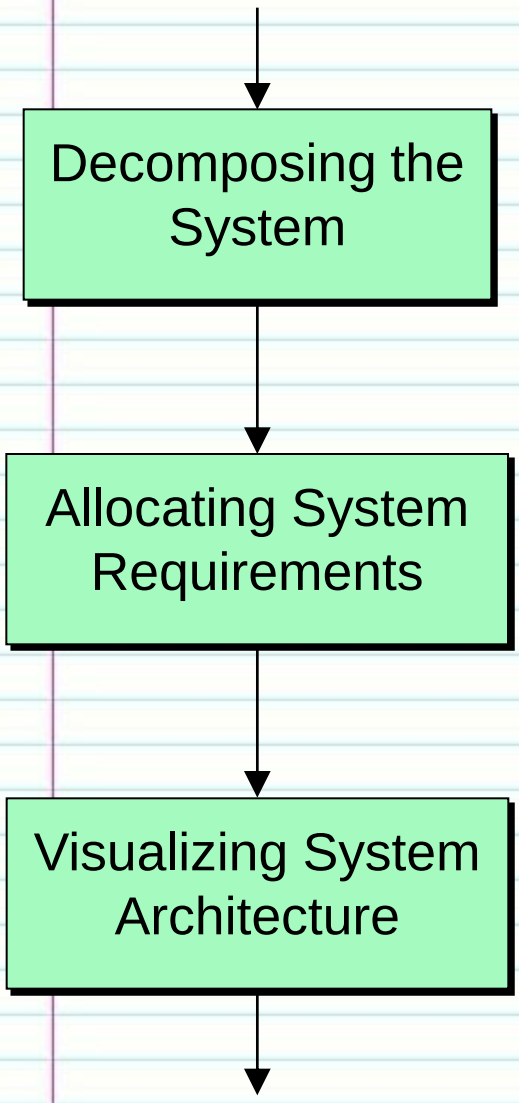
R1. ABHS shall check in and transport luggage to departure gates and baggage claim areas according to the destinations of the passengers.

R2. ABHS shall allowairline agents to inquire about luggage status and to locate luggage.

R3. ABHS shall check all baggage and detect items that are prohibited.

R4. ABHS shall be able to serve 20,000 passengers per day.

System Architectural Design

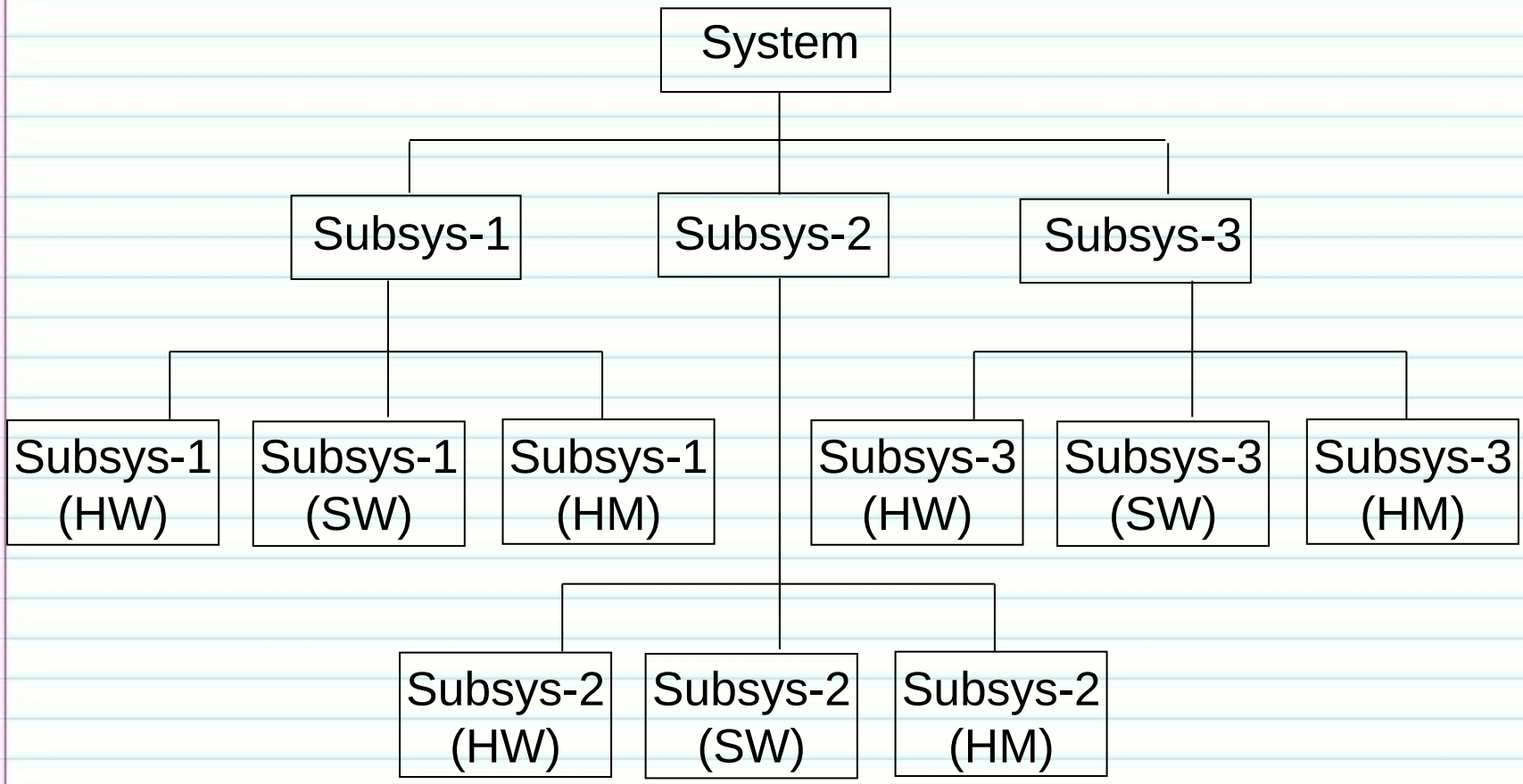


- Decomposing the system into a hierarchy of functional cohesive, loosely coupled subsystems, which partition the system requirements and facilitate reuse of COTS components.
- System requirements are assigned to the subsystems.
- The system architecture is depicted using a certain diagramming technique.

System Decomposition Strategies

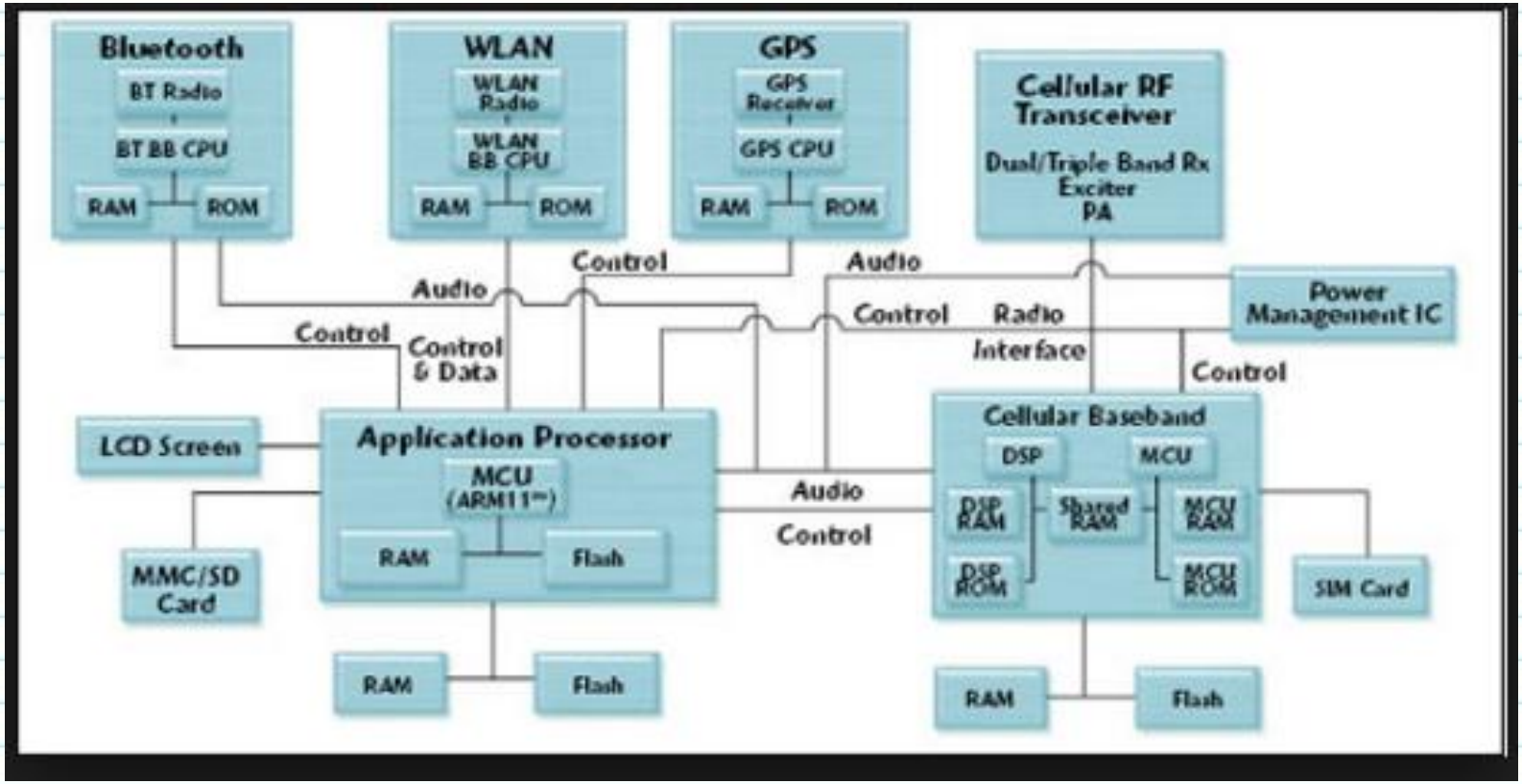
1. Decompose the system according to system functions.
2. Decompose the system according to engineering disciplines.
3. Decompose the system according to existing architecture.
4. Decompose the system according to the functional units of the organization.
5. Decompose the system according to models of the application.

Partition According to Major Functionality

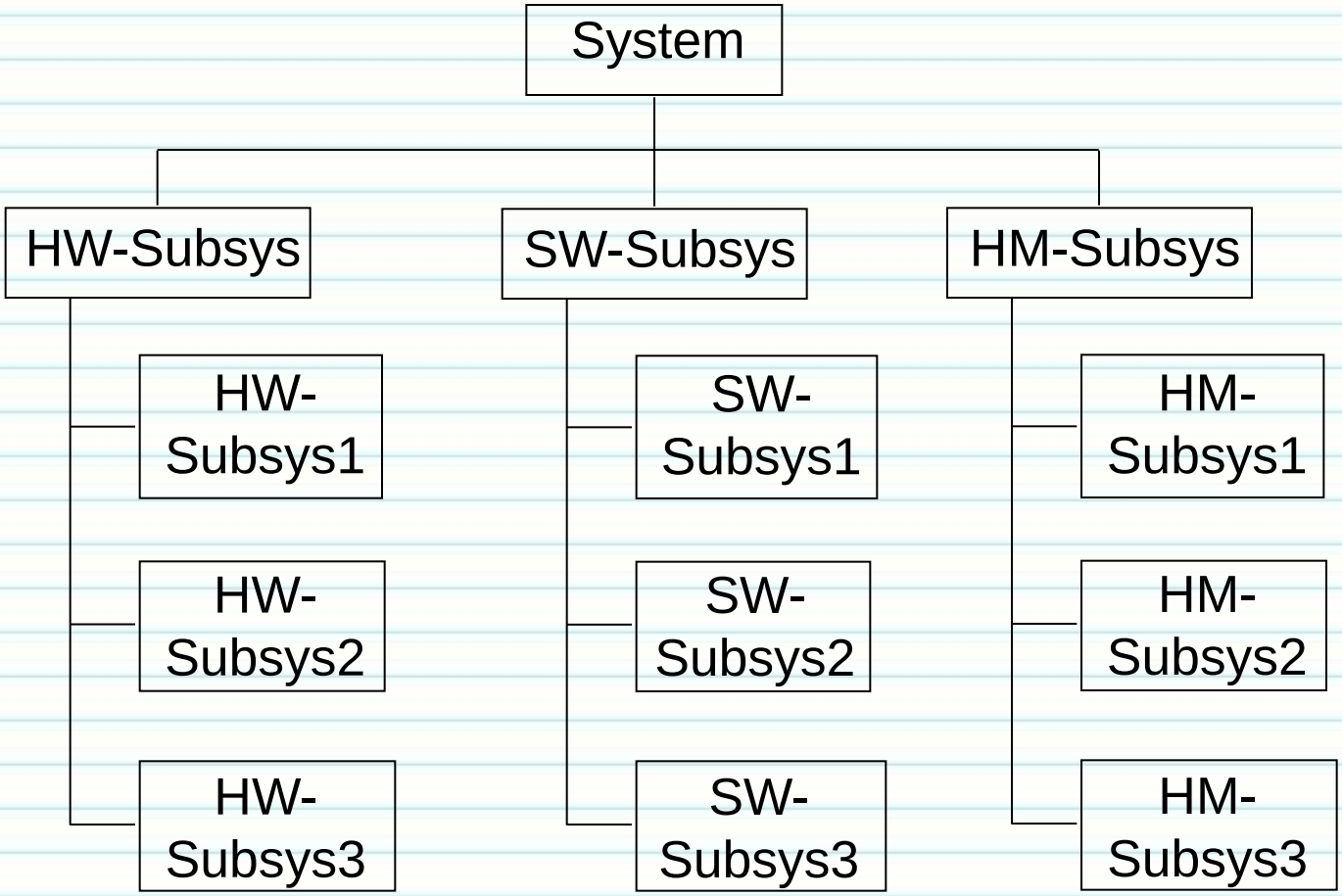


Partition According to Major Functionality

An example Smartphone subsystem architecture



Partition According to HW, SW & Human Subsystems



Requirements Allocation Example

Requirements of an Airport Baggage Handling System:

R1.1. ABHS shall allow airline agents to check in luggage.

R1.2. ABHS shall transport luggage to their destinations within the airport.

R1.2.1. ABHS shall transport luggage from check-in areas to departure gates.

R1.2.2. ABHS shall transport luggage from arrival gates to baggage claim areas.

R1.2.3. ABHS shall transport luggage from arrival gates to departure gates for transfer passengers.

R1.2.4. ABHS shall transport luggage within a terminal using conveyors.

R1.2.5. ABHS shall transport luggage between terminals using DCVs running on high-speed tracks.

R1.3. ABSH shall control the transportation of the luggage within and between the terminals.

Requirements Allocation Example (continued)

R4.1. Each check-in area shall handle 1,150 pieces of check-in luggage per day.

R4.2. Each check-in agent shall check in an average three passengers per minute.

R4.3. Each conveyor hardware shall scan and transport 500 check-in pieces of luggage per hour.

R4.4. ABHS control software shall process 2,300 check-in bags per day and 1,000 bar code scan requests per hour.

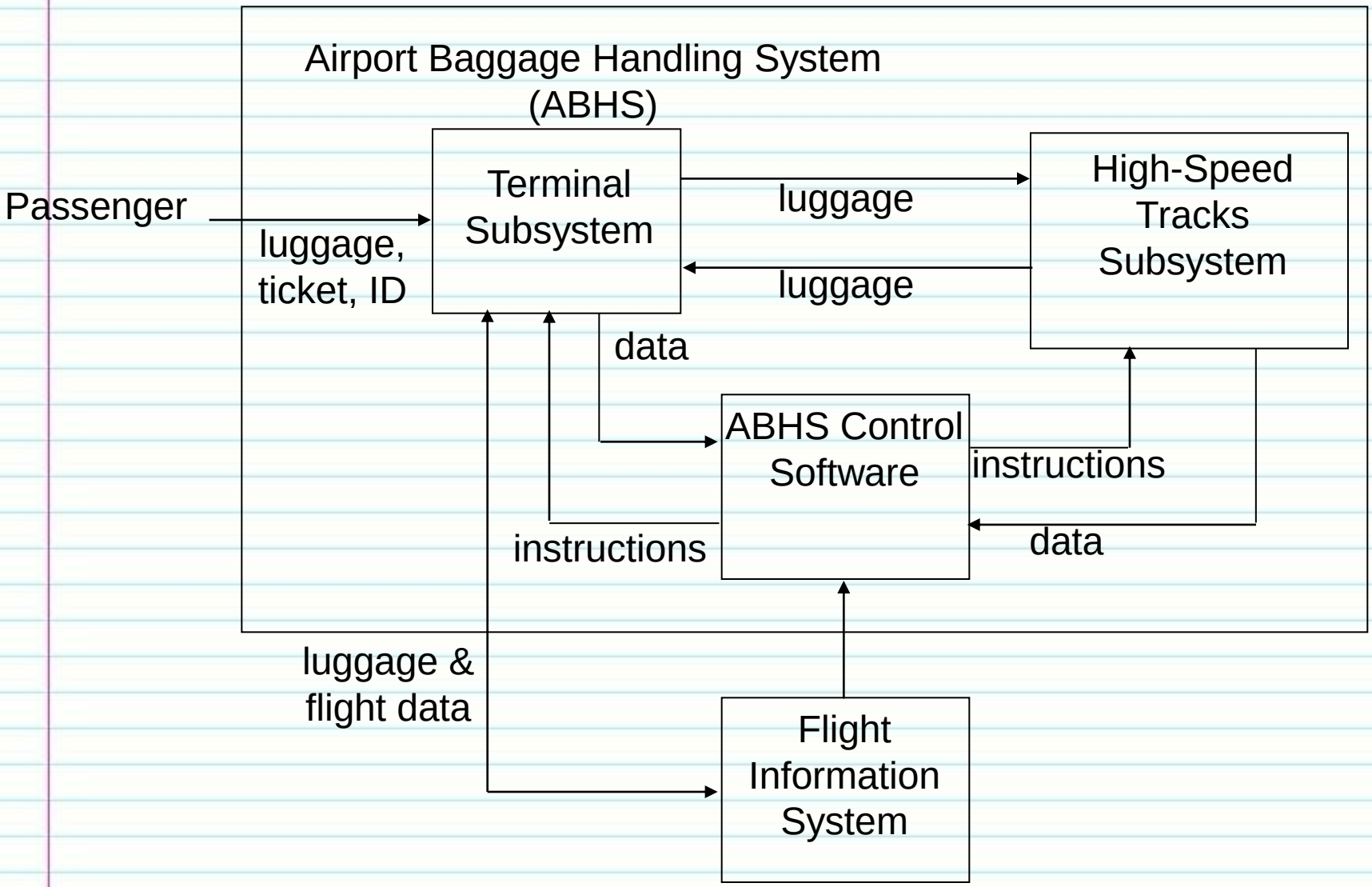
Requirements Allocation

Functional Cluster	Functional Description	System Requirements	Functional Subsystem Identified
Luggage check-in	This functional cluster processes luggage check-in.	R1.1, R4.1, R4.2	Luggage check-in subsystem
Conveyor	This functional cluster is responsible for moving luggage within a terminal.	R1.2.1, R1.2.2, R1.2.3, and R1.2.4, R4.3	Conveyor subsystem
High-speed track	This functional cluster transports luggage between terminals.	R1.2.3 and R1.2.5	High-speed track subsystem
Software control	This functional cluster controls the hardware to transport luggage within and between the terminals.	R1.3, R4.4	Software control subsystem

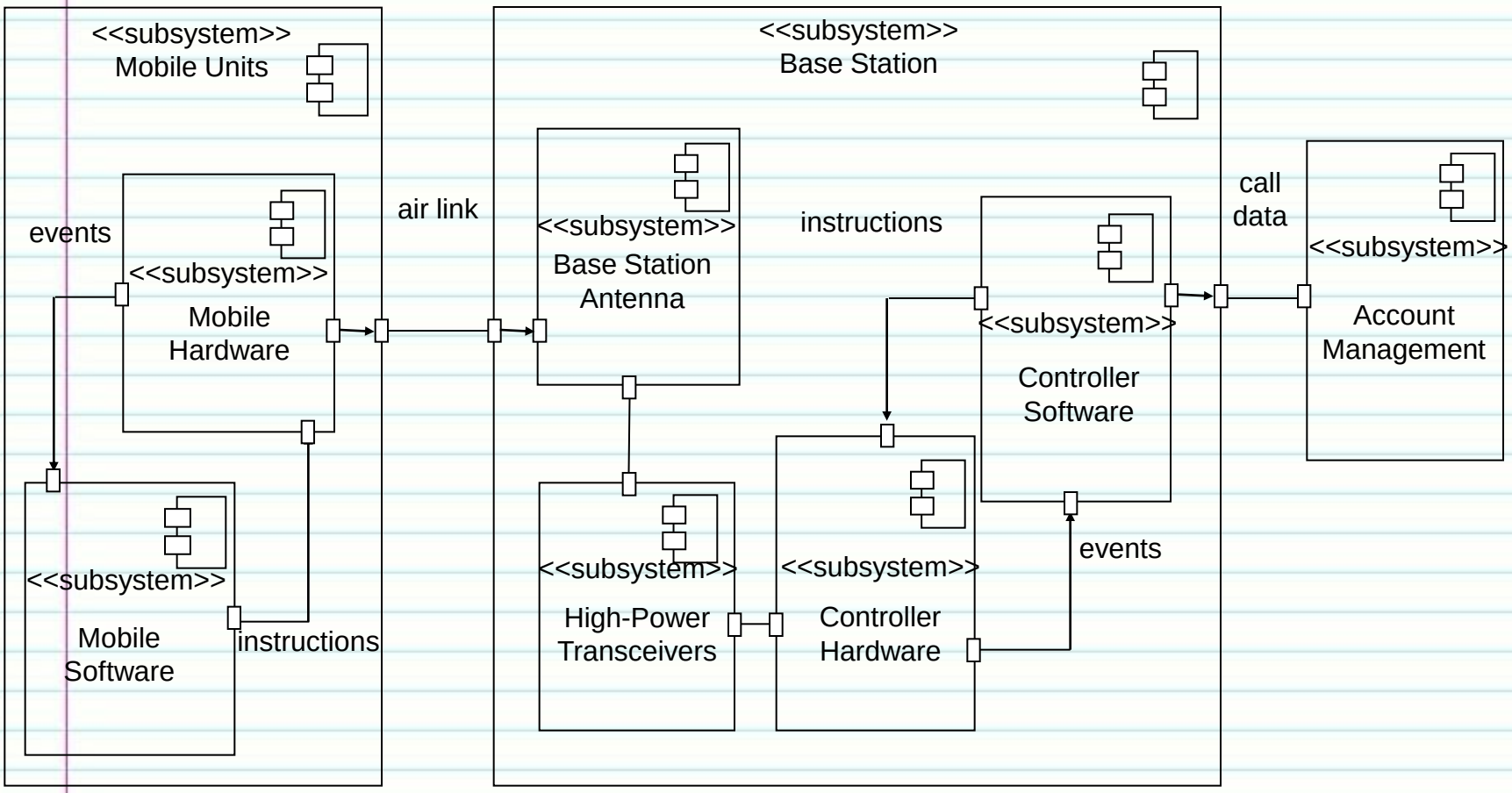
Architectural Design Diagrams

- Block diagram
- UML component diagram
- SysML diagrams
- Data flow diagram
- and more ...

Block Diagram

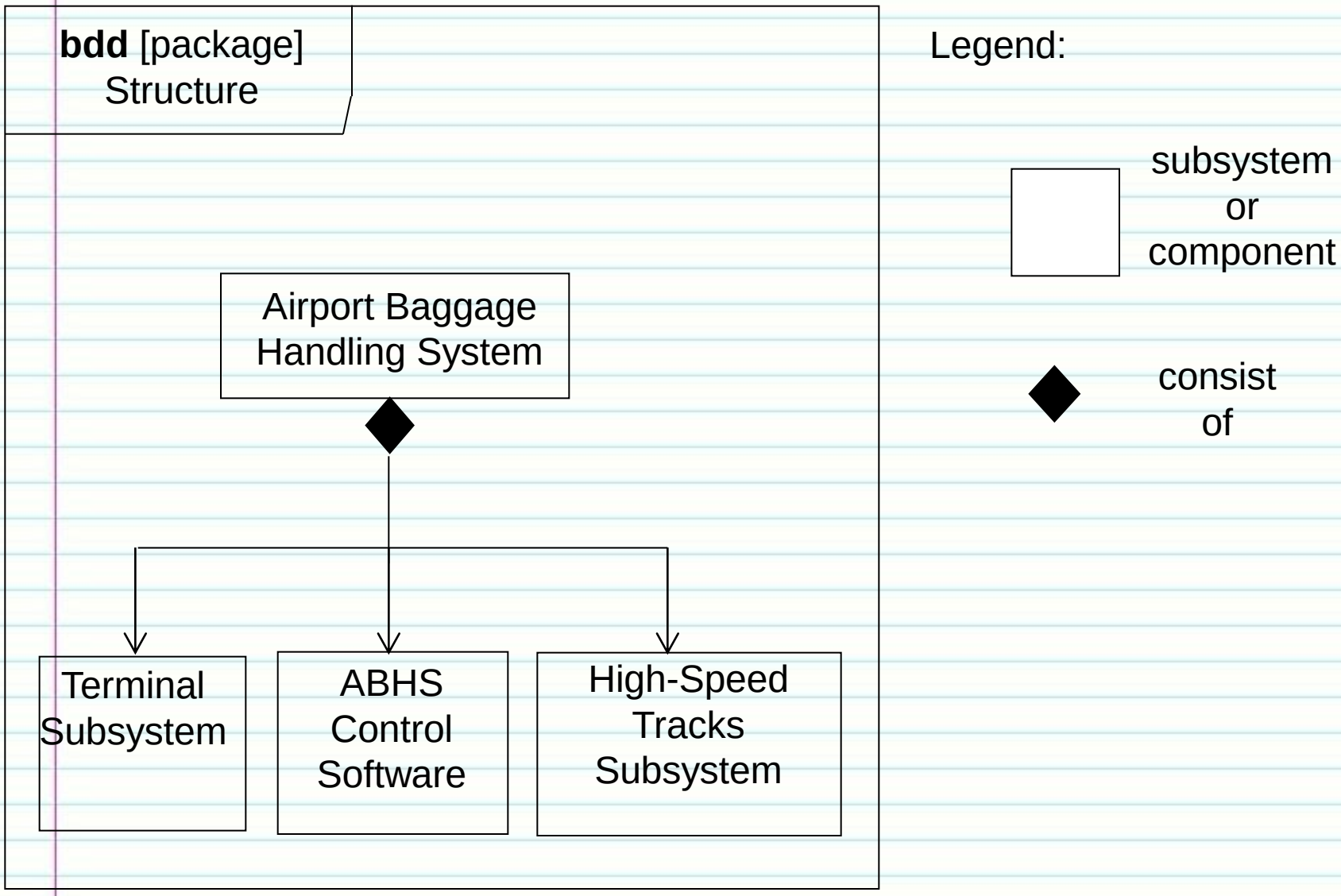


UML Component Diagram

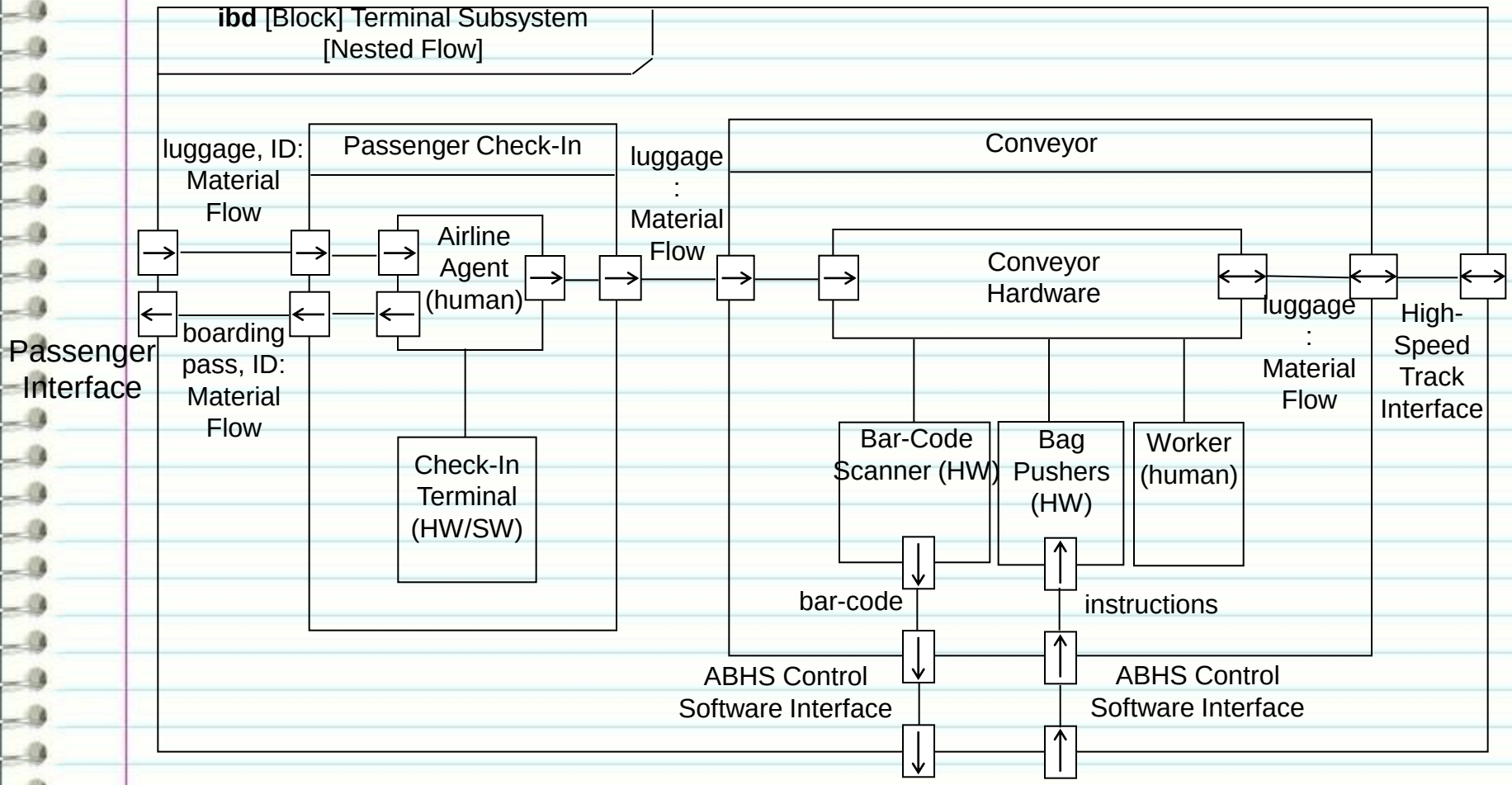


Legend: component port <<subsystem>>stereotype for introducing new modeling constructs

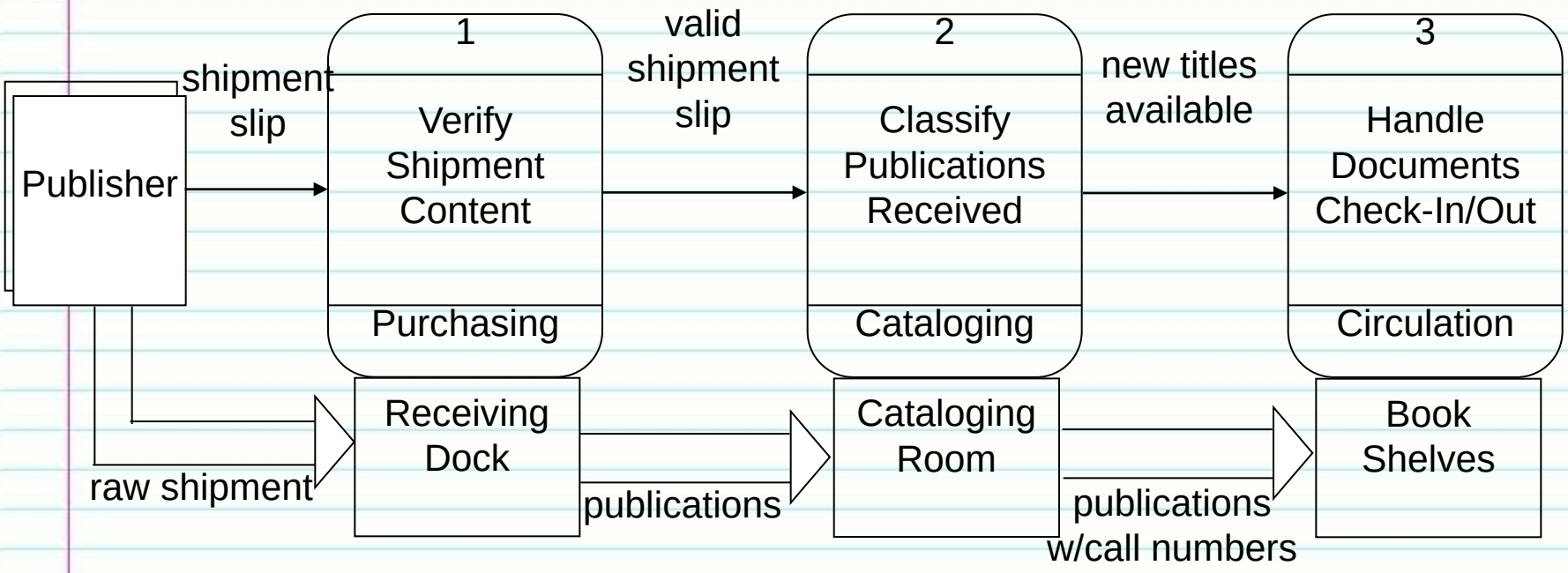
SysML Block Definition Diagram



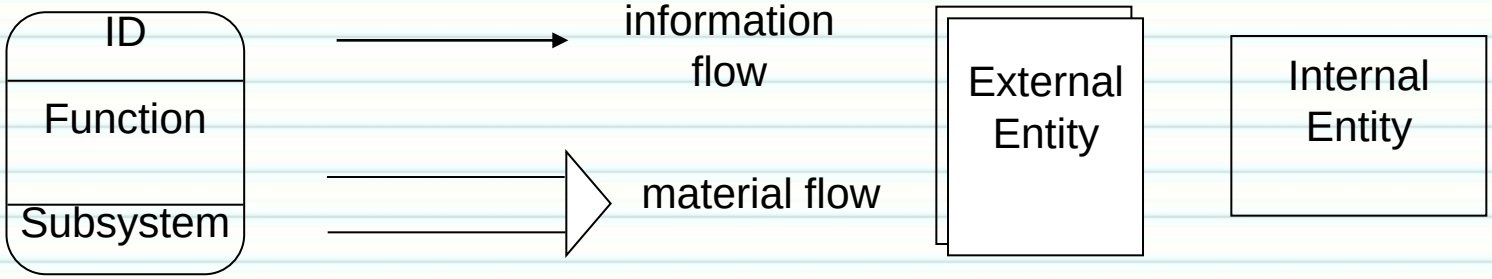
SysML Internal Block Diagram



Data Flow Diagram with Material Flows



Legend:



Other System Engineering Activities

- Development of subsystems
  - The subsystems are developed by different engineering teams.
  - The engineering teams collaborate to jointly solve interdisciplinary problems.
- System integration, testing, and deployment
  - The subsystems and components are integrated and tested for interoperability.
  - The system is tested to ensure that it satisfies the system requirements and constraints.
  - The system is then installed and tested in the target environment.

System Configuration Management

- System configuration management ensures that the system components are updated consistently.
- System configuration management is needed because
  - a system may have different versions and releases to satisfy the needs of different customers,
  - the engineering teams may update the system configuration concurrently.
- It is performed during the development phase as well as the maintenance phase.
- Its functions include configuration identification, configuration change control, configuration auditing, and configuration status reporting.

Class Discussion

- Why is system engineering a multidisciplinary effort?
- What is the relationship between system engineering and software engineering?
- Provide examples of systems that require a system engineering approach.