

Security Governance Master of Science in Cyber Security

AA 2024/2025

INTRODUCTION TO THREAT MODELLING

Introduction

Threat modeling is a structured approach to identify, quantify, and address threats.

It allows system security staff to communicate the potential damage of security flaws and prioritize remediation efforts.

Goals:

- strengthen protection
- improve preparedness
- increase awareness
- Support risk management

Introduction

What is a threat?



Introduction

How is a threat connected with risk?

$$\text{RISK} = \text{THREAT} \times \text{PROBABILITY} \times \text{IMPACT}$$

Learning to Threat Model

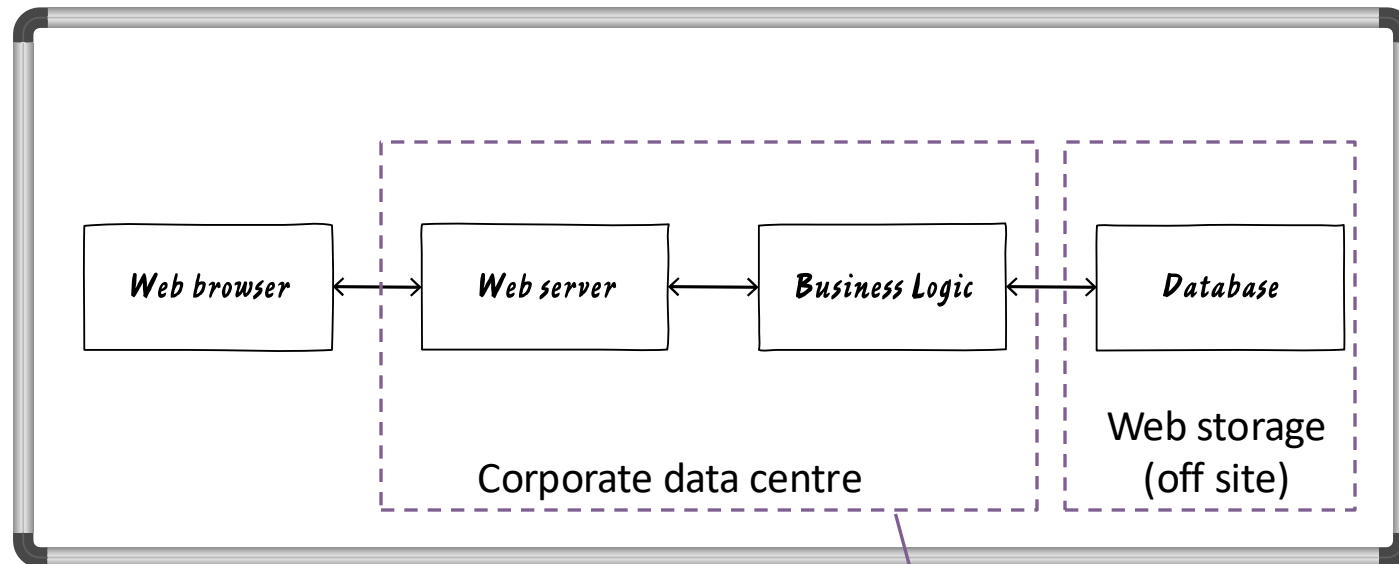
You begin threat modelling by focusing on four key questions:

1. What are you building?
2. What can go wrong?
3. What should you do about those things that can go wrong?
4. Did you do a decent job of analysis?

What Are You Building?

Diagrams are a good way to communicate what you are building

Example:



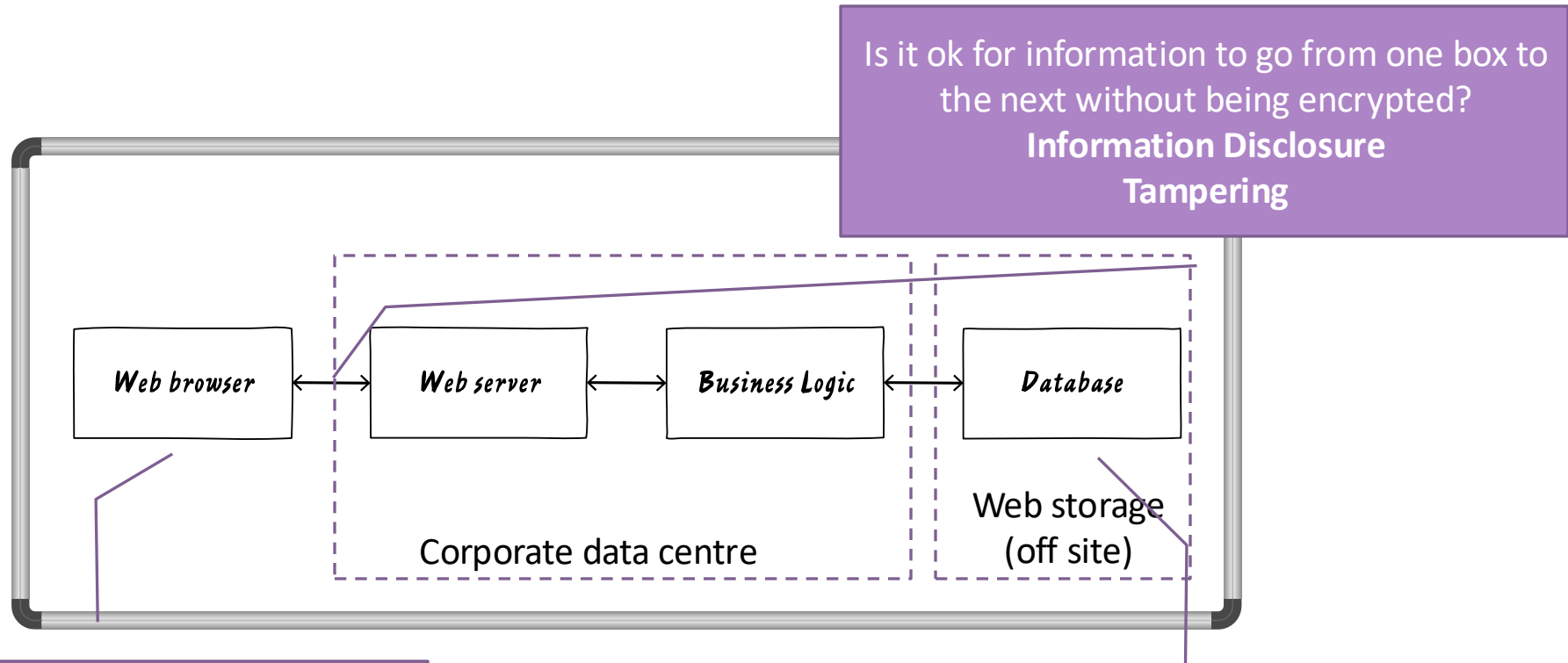
Identify your Trust Boundaries

What Can Go Wrong?

A simple way to answer is using the **STRIDE** Mnemonic to Find Threats

- **Spoofing**: pretending to be something or someone you're not
- **Tampering**: modifying something you're not supposed to modify
 - It can include packets on the wire (or wireless), bits on disk, or the bits in memory
- **Repudiation**: claiming you didn't do something (regardless of whether you did or not)
- **Information Disclosure**: exposing information to people who are not authorized to see it
- **Denial of Service**: attacks designed to harm system availability
 - E.g., by crashing it, making it unusably slow, or filling all its storage
- **Elevation of Privilege**: when a program or user is technically able to do things that they're not supposed to do

Example of STRIDE application



How do you know that the web browser is being used by the person you expect?

Spoofing
Information Disclosure

What happens if someone modifies data in the database?

Tampering

Tips for Identifying Threats

Start with external entities

Never ignore a threat because it's not what you're looking for right now

Focus on feasible threats

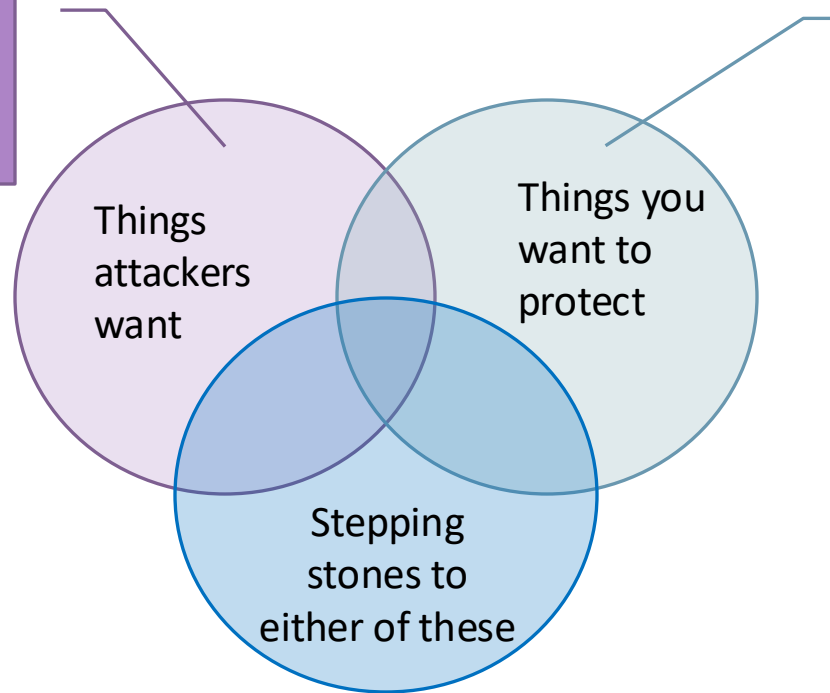
Strategies for Threat Modeling

Threat modelling variants:

- **Asset-centric**
- **Attacker-centric**
- **Software-centric**

Asset-centric TM

Usually assets that attackers want are relatively tangible things



Many of these assets are intangibles e.g., your company's reputation

Implementing Asset-Centric Modelling

1. Make a list of your assets and then consider how an attacker could threaten each
2. connect each item on the list to particular computer systems or sets of systems
3. draw the systems (showing the assets and other components as well as interconnections) until you can tell a story about them

You can use this model to apply either an attack set like STRIDE or an attacker-centered brainstorm to understand how those assets could be attacked

Attacker-centric TM

Security experts may be able to use various types of attacker lists to find threats against a system

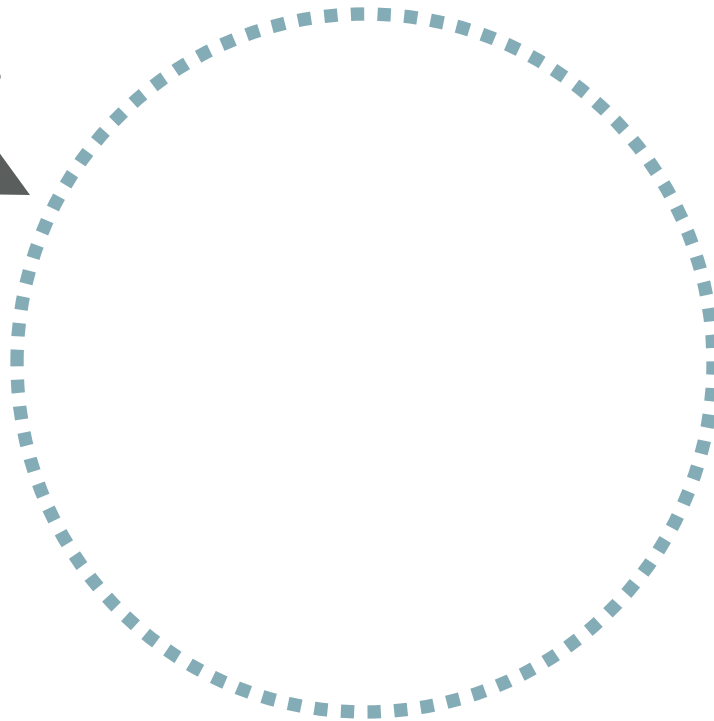
Given a list of attackers, it's possible to use it to provide some structure to a brainstorming approach

Attacker-driven approaches are also likely to bring up possibilities that are human-centred

Software-centric TM

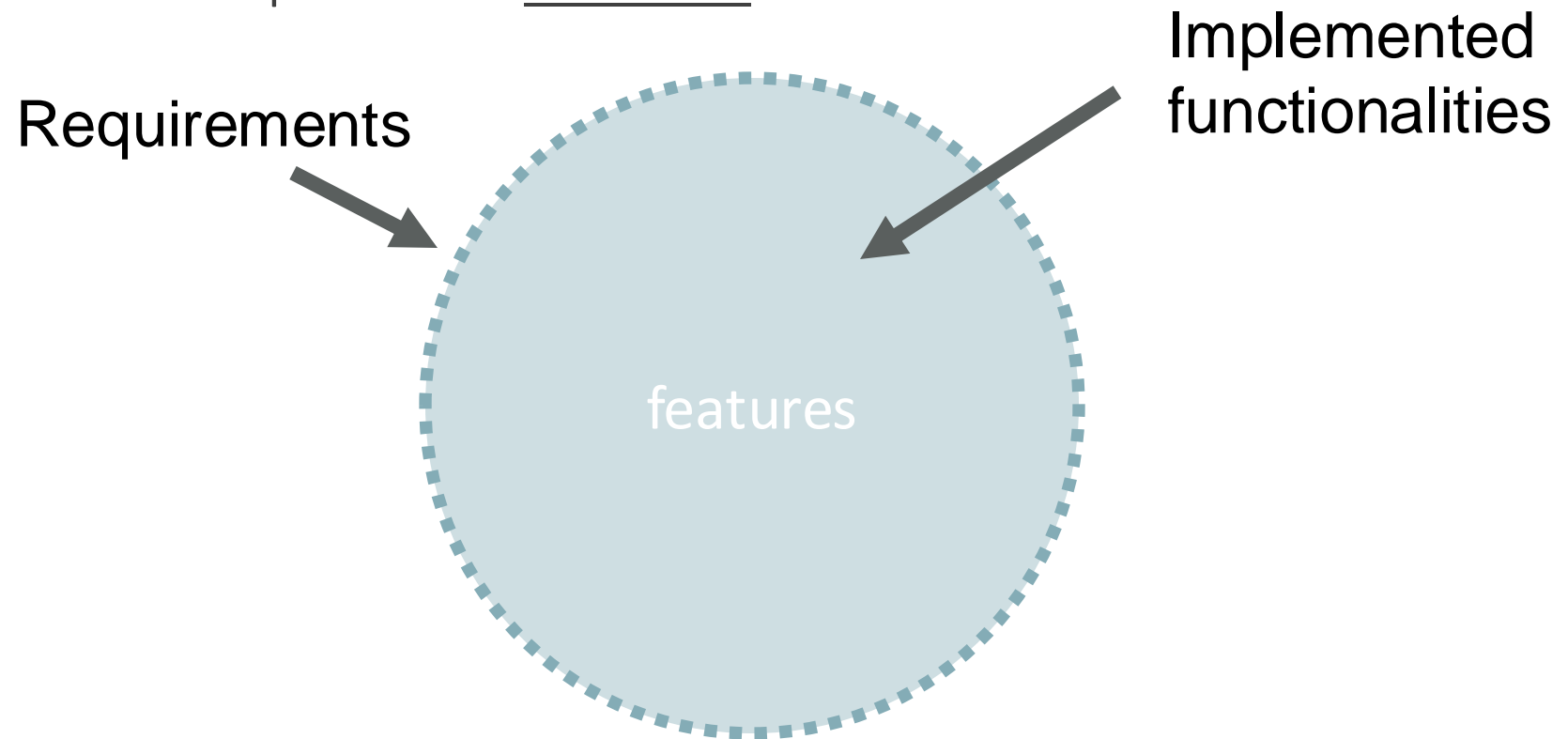
Software development in an ideal world

Requirements



Software-centric TM

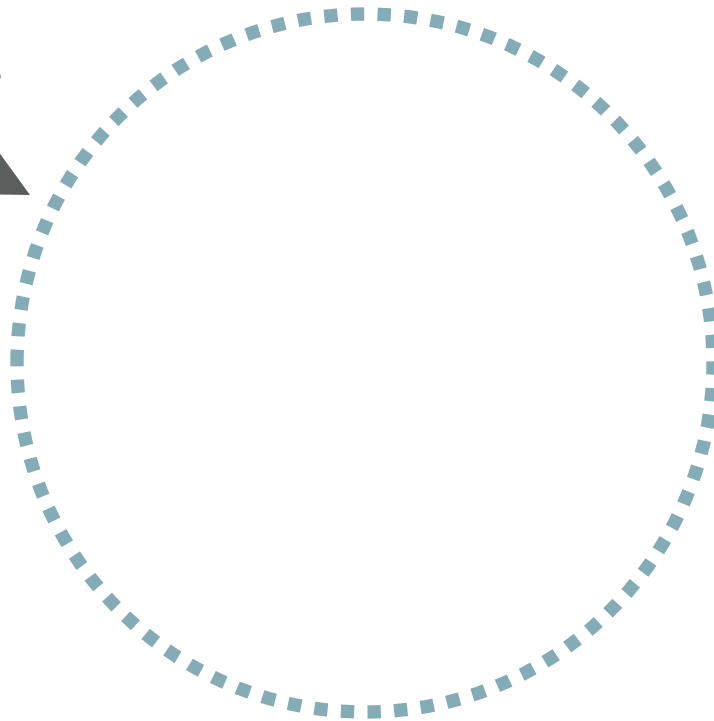
Software development in an ideal world



Software-centric TM

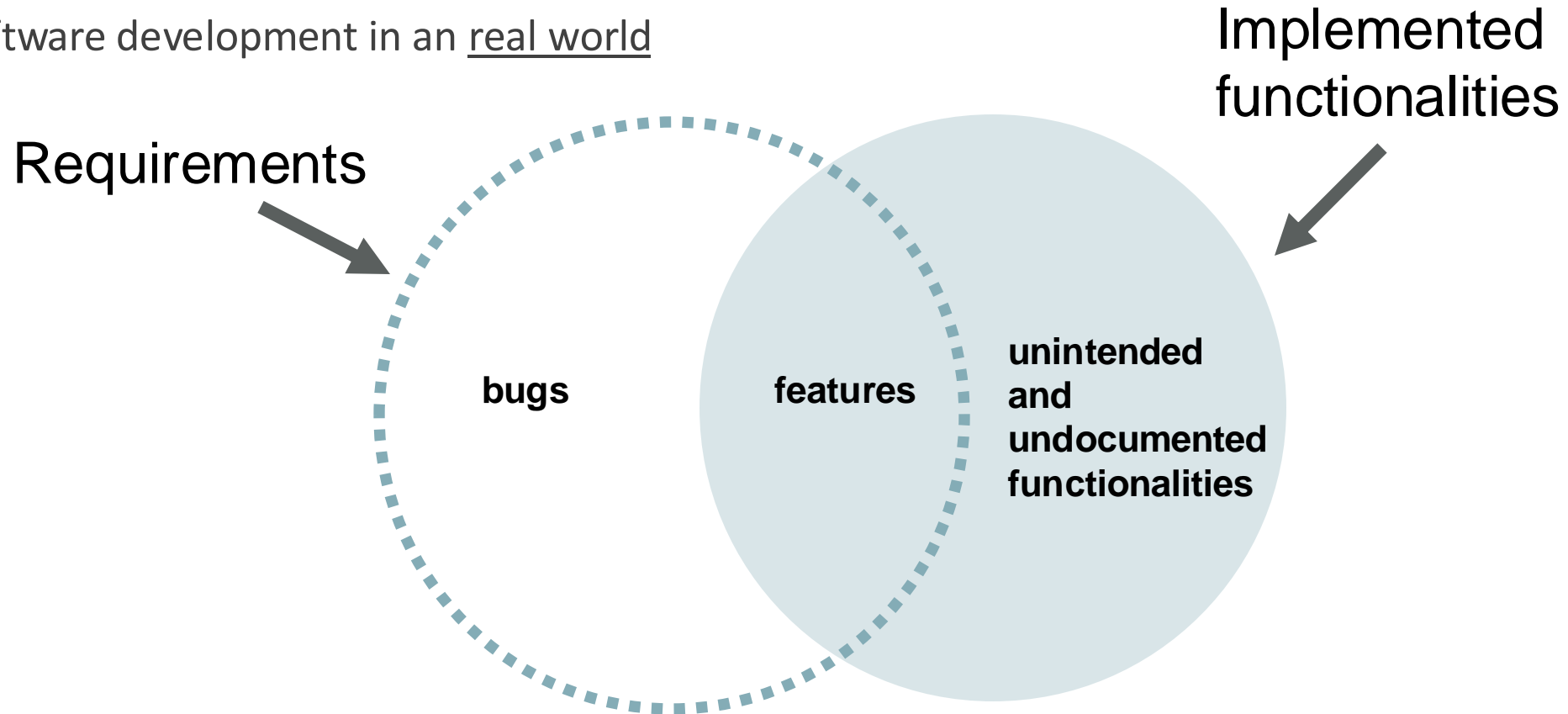
Software development in an real world

Requirements

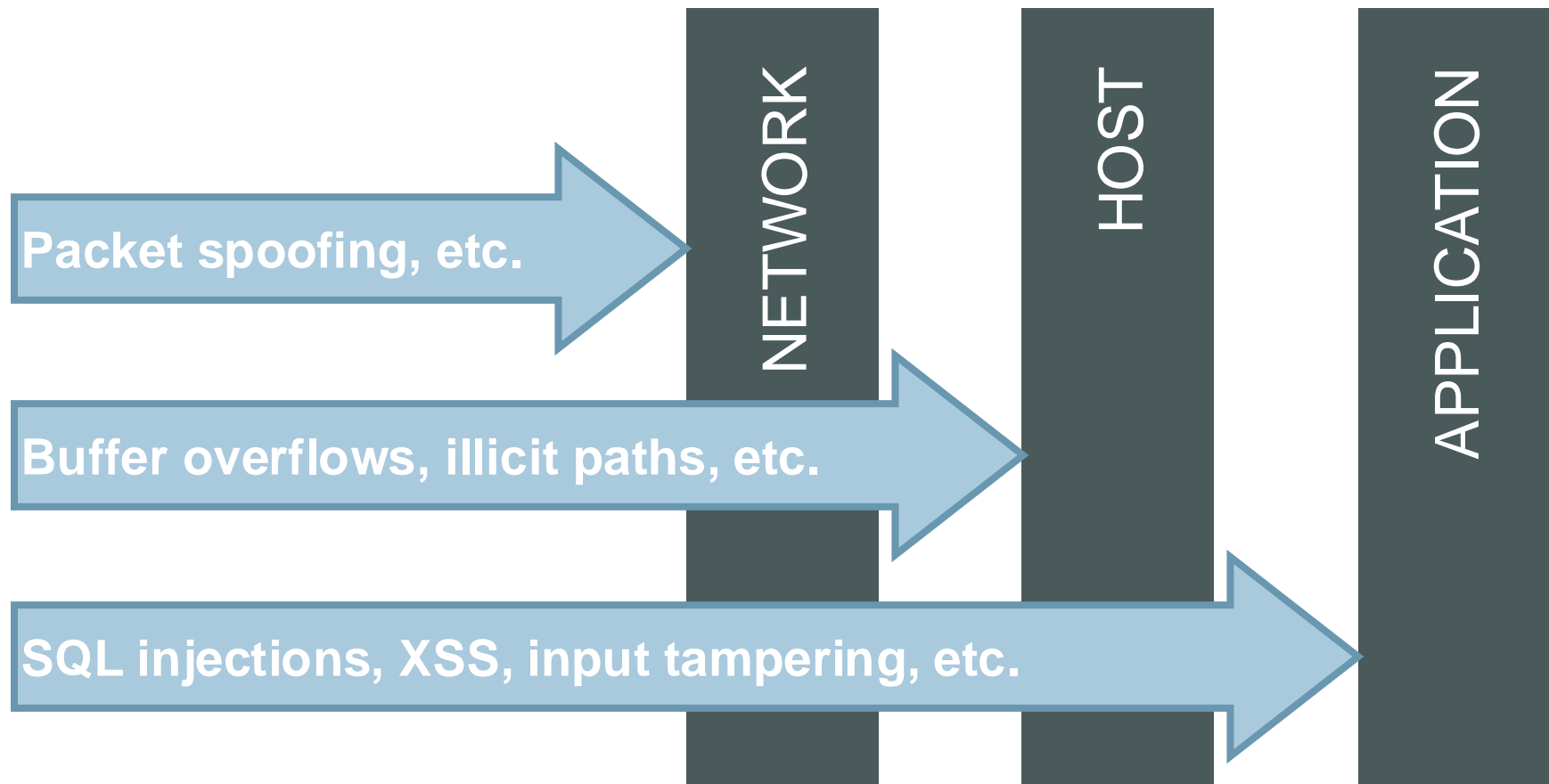


Software-centric TM

Software development in an real world



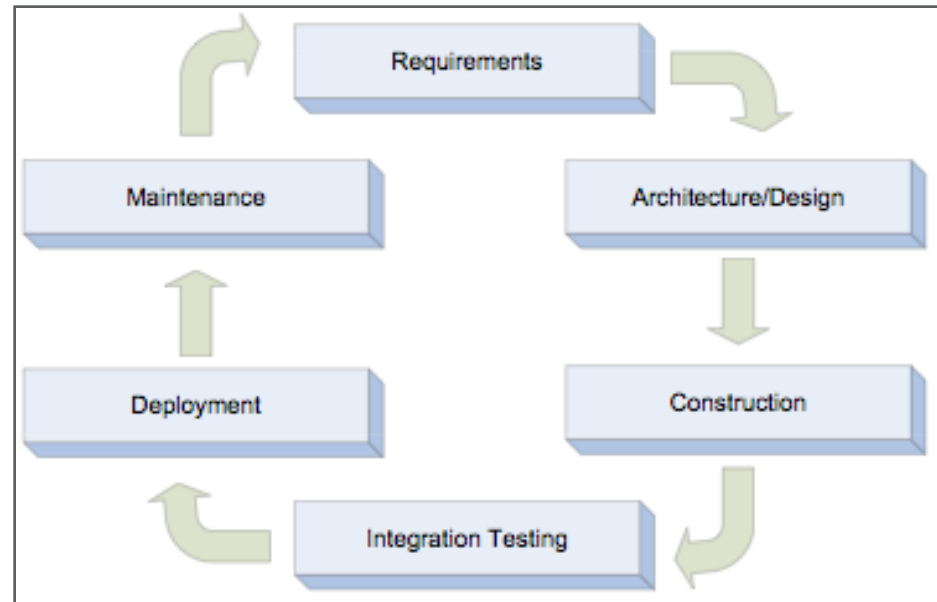
Software-centric TM



Software-centric TM

Ideally, threat models are created during system design prior to any deployment.

In practice, threat models are often created for existing systems, making it part of maintenance.



System designers with security experience are best equipped to identify the threats.

Software-centric TM

Activities	Core	Security
Planning		
Requirements and Analysis	Functional Requirements Non Functional Requirements Technical Requirements	Security Objectives
Architecture and Design	Design Guidelines Architecture and Design Review	Threat Modeling Architectural Risk Analysis
Development	Unit Tests Code Review Daily Builds	Security Code Review
Testing	Integrated Testing System Testing	Security Testing
Deployment	Deployment Review	Penetration Testing
Maintenance		Risk Assessment

How to model your Software

Flow Chart

- Less commonly used
- Difficult to overlay threats

Data Flow Diagrams (DFD)

- Hierarchical in nature
- Focused on data input
- Representation used by Microsoft

Universal Modeling Language (UML)

- Familiar to developers
- OO and component diagramming

DATA FLOW DIAGRAMS

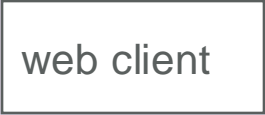



A Data Flow Diagram is a graphical representation of flow of data through an information system

DFD can be used to visualize data processing

- what data is input for the system
- what data is produced as output
- processing steps
- data persistence activities

DFDs are useful for threat modeling as they provide an high-level view of how the information system manages assets that need to be protected.

DATA FLOW DIAGRAMS

Element	Appearance	Meaning	Example
External entity		People, or code outside your control	Your customer, Microsoft.com
Data flow		Communication between processes, or between processes and data stores	Network connections, HTTP, RPC, LPC
Process		Any running code	Code written in C, C#, Python, or PHP
Data store		Things that store data	Files, databases, the Windows Registry, shared memory segments

DATA FLOW DIAGRAMS

DFD elements

- External entity / Terminator



- Can be duplicated on the diagram one or more times
- Are not part of the system being studied.
- May be part of the same organization, but are beyond the influence of the system being described.
- We have no direct control on external entities

DATA FLOW DIAGRAMS

DFD elements

- External entity / Terminator



- Can represent another system or subsystem within the same organization.
- Must receive data from or send data to the system being described.
- Are placed on the edges of the DFD

DATA FLOW DIAGRAMS

DFD elements

- External entity / Terminator



- **Rule 1:** Never include in a DFD direct data flows from one external entity to another.
 - They are irrelevant to the system being described because they are external.
 - Like how a conversation between two people you don't know is irrelevant to you.

DATA FLOW DIAGRAMS

DFD elements

- Data flows

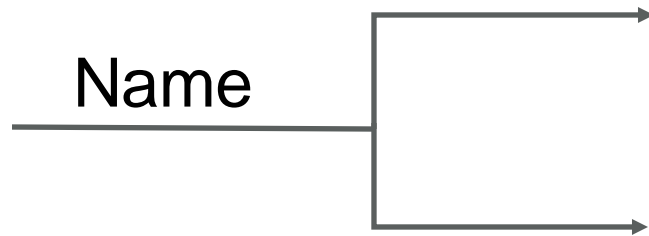


- Represent data moving between elements of the system
- Data movements are always directed
- **Rule 2:** Only represent data, not material goods.
- Each arrow has a name representing the specific data moved through the flow
- **Rule 3:** Only include one type of data per arrow. E.g. “Orders”, “Customer Data”

DATA FLOW DIAGRAMS

DFD elements

- Data flows

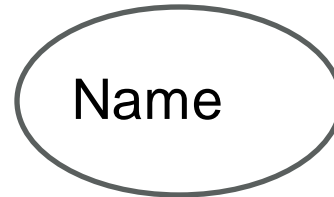


- A fork in a data flow means that the same data goes to two destinations.
- The same data coming from several locations can also be joined.

DATA FLOW DIAGRAMS

DFD elements

- Processes

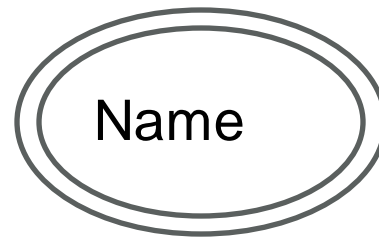


- Where the business logic is implemented
- **Rule 4:** Processes must have at least one data flow in and one data flow out.
- Are named with a verb and an object of the verb (the thing being processed)
 - E.g. “Calculate (*verb*) average income (*object*)”.
- Each process should represent only one function or action

DATA FLOW DIAGRAMS

DFD elements

- Processes

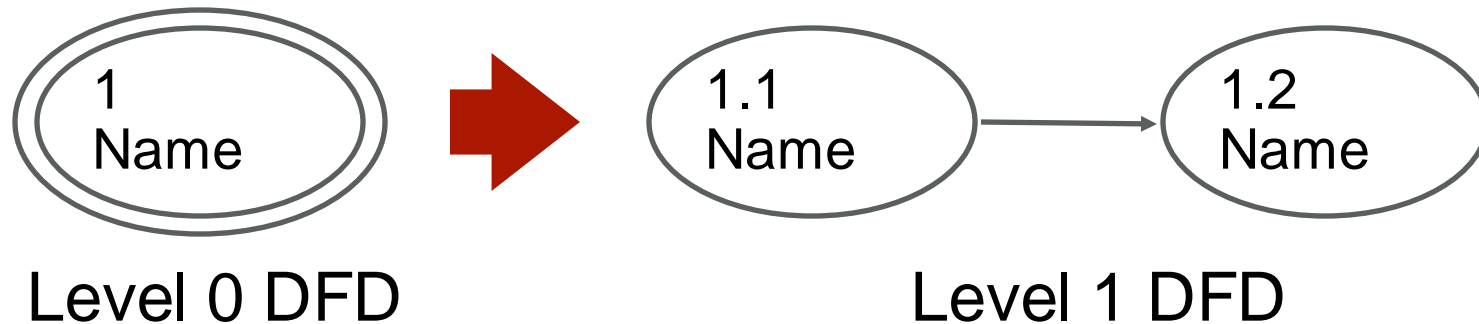


- Processes represented with a double circling line are *composite*
 - They can be further detailed by breaking them in subprocesses connected by flows
 - Their internal structure can be detailed at lower DFD levels

DATA FLOW DIAGRAMS

DFD elements

- Processes

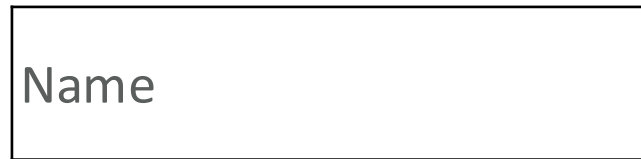


- Processes must be univocally numbered
- The break down of composite process x generates processes numbered as x.y

DATA FLOW DIAGRAMS

DFD elements

- Data store

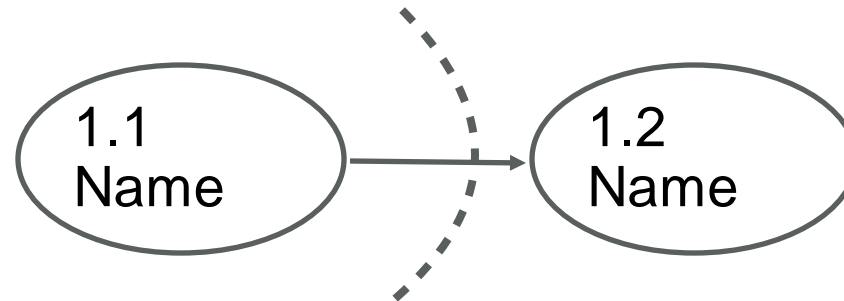


- Where data is stored
- Name is usually the plural form of the data being flowed into it
- Can be duplicated one or more times to avoid line crossing.
- Can be shared by two or more systems.
- Contents of datastore are detailed elsewhere in a data dictionary

DATA FLOW DIAGRAMS

DFD elements

- Trust boundaries



- Represent limits in the trustability of subparts of the system

DATA FLOW DIAGRAMS

Wrong DFD design choices:

- Do not use direct data flows from one data store to another. There must be a process between the stores
- Do not use direct data flows from an external entity to a data store. A process is needed between them.
- Do not show direct data flows between external entities.

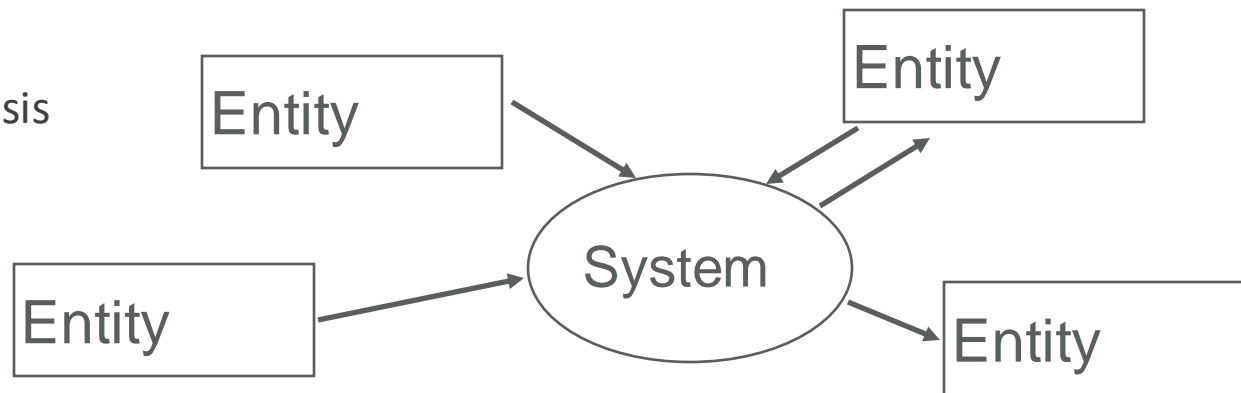
DATA FLOW DIAGRAMS

1. Identify actions and actor with use cases
2. Build context level DFD
 - Separate external entities from the system
3. Build level 0 DFD
 - Identify processes in the system
4. Build level 1 DFD
 - Break down processes
 - Identify inter-process data flows
 - Identify data stores
5. Add trust boundaries

DATA FLOW DIAGRAMS

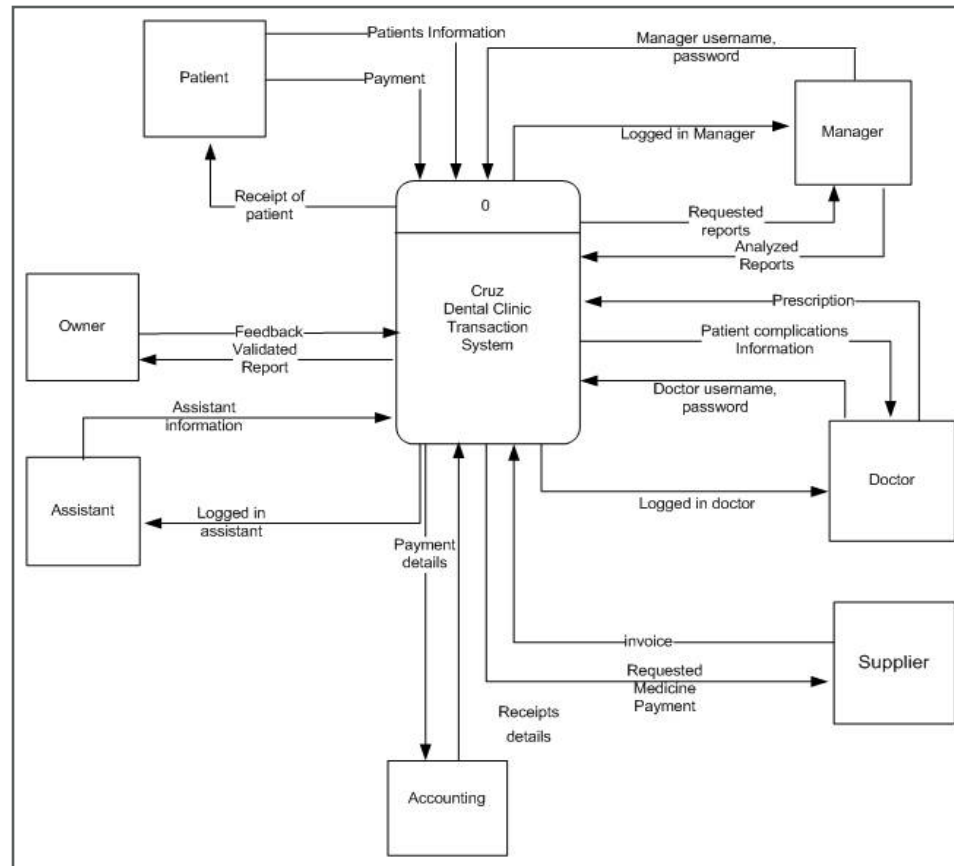
Context level DFD

- Contains only one composite process
 - You can call it “System” or use a more meaningful name
- Its purpose is to focus on
 - External entities
 - Which are the actors that can interact with the system?
 - Data flows between each external entity and the process
- Its a star-like DFD!
- Mainly built on the basis of use-cases



DATA FLOW DIAGRAMS

Example:



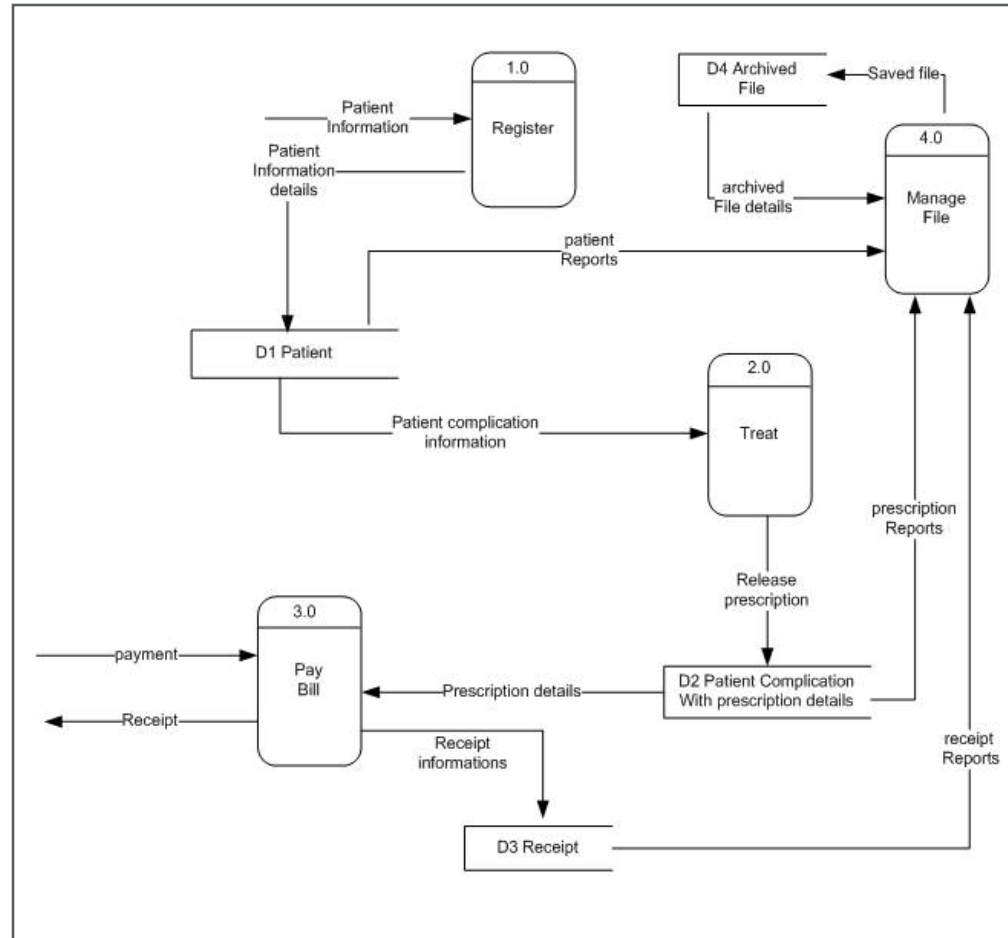
DATA FLOW DIAGRAMS

Level-0 DFD

- Breaks down the system
 - identify main business processes
 - connect them with appropriate data flows
- It may include data stores at a high level of abstraction

DATA FLOW DIAGRAMS

Example:



DATA FLOW DIAGRAMS

Lower level DFDs

- Further detail specific composite processes
- Functional decomposition
 - An iterative process of breaking a system description down into finer and finer detail
 - Uses a series of increasingly detailed DFDs to describe an IS
- Balancing
 - The conservation of inputs and outputs to a data flow process when that process is decomposed to a lower level
 - Ensures that the input and output data flows of the parent DFD are maintained on the child DFD

DATA FLOW DIAGRAMS

Full example: lemonade stand

Step 1: think about activities that take place at a lemonade stand

- Create a list

<u>Activities</u> Customer Order Serve Product Collect Payment Produce Product Store Product

DATA FLOW DIAGRAMS

Full example: lemonade stand

Step 1: include also backend activities needed to support the included ones

Activities

Customer Order

Serve Product

Collect Payment

Produce Product

Store Product

Order Raw Materials

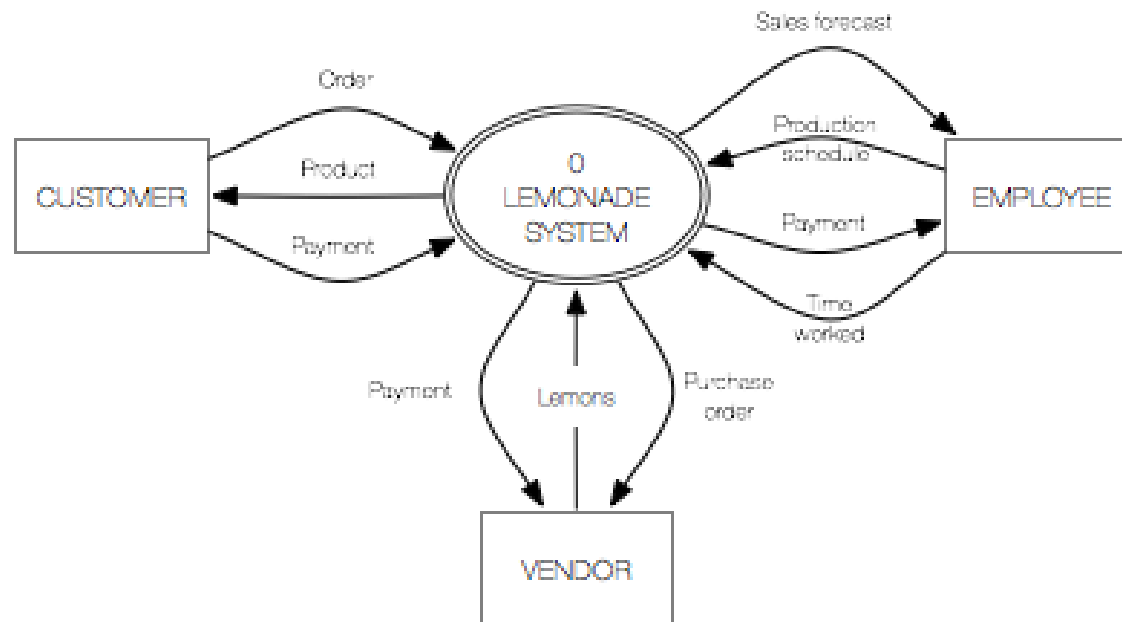
Pay for Raw Materials

Pay for Labor

DATA FLOW DIAGRAMS

Full example: lemonade stand

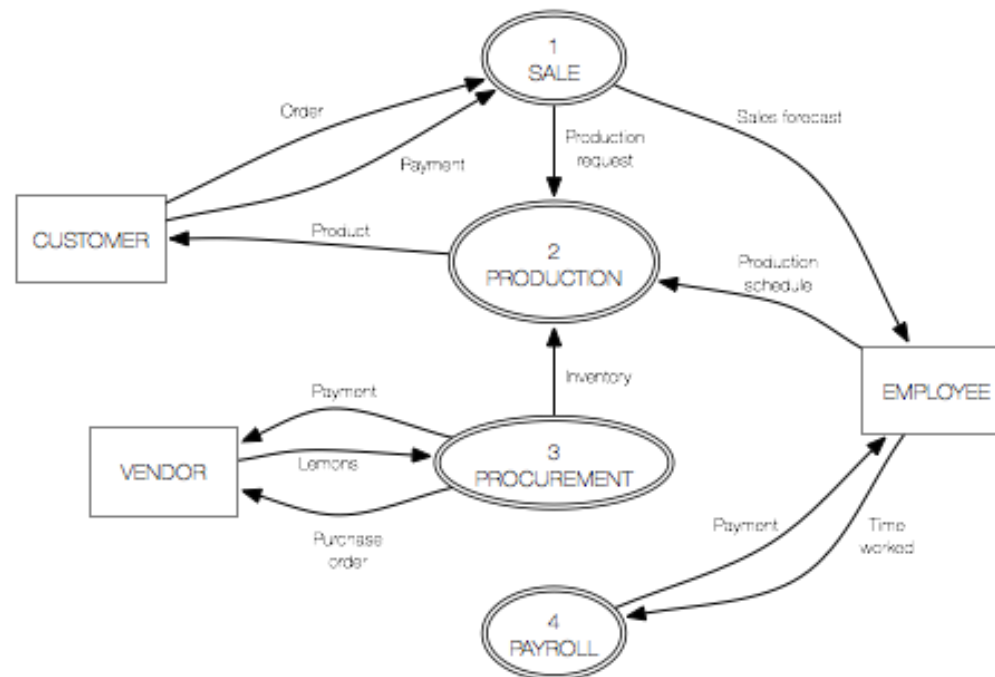
Step 2: build a context-level DFD



DATA FLOW DIAGRAMS

Full example: lemonade stand

Step 3: Drill down the system to build a Level-0 DFD



DATA FLOW DIAGRAMS

Full example: lemonade stand

Step 4: Further break processes apart to detail their internals

