SFWR ENG 3RA3 Summary

Author: Kemal Ahmed
Instructor: Dr. Ryszard Janicki
Date: Fall 2014

Math objects made using MathType; graphs made using Winplot.

Please join GitHub and contribute to this document. There is a guide on how to do this on my GitHub.

Table of Contents

Lecture 2 – Types of Statements

Software Requirements Specification (SRS): description of a software system that will be developed

Descriptive Statement: facts about the system, such as natural laws and physical constraints

- Domain Property (DOM): affecting environmental phenomena, such as physics

Prescriptive Statement: desired behavioural properties of a system; can be negotiated
Types of prescriptive statements:

- **System Requirement** (SYSREQ): when the software interacts with the other system components, i.e. environment
    - vocabulary understandable by all parties
    - Types of SYSREQ:
        - Assumptions (ASM): how the environment should be, usually through sensors and stuff
    - SOFREQ, ASM, DOM |= SYSREQ
        - When the SOFREQ, ASM, and DOM are satisfied, SYSREQ is satisfied
- **Software Requirement** (SOFREQ): relationship between a set of input variables, *I*, and *O*, the set of output variables
    - vocabulary understandable by software developers

Lecture 3

Non-functional requirements

- Look and Feel Requirements:
    - Appearance Requirements
    - Style Requirements
- Usability and Humanity Requirements:
    - Ease of Use Requirements
    - Personalization and Internationalization Requirements
    - Learning Requirements
    - Understandability and Politeness Requirements
    - Accessibility Requirements
- Performance Requirements:
    - Speed and Latency Requirements
    - Safety-Critical Requirements:
    - Precision or Accuracy Requirements
    - Reliability and Availability Requirements
    - Robustness or Fault-Tolerance Requirements
    - Capacity Requirements
    - Scalability or Extensibility Requirements
    - Longevity Requirements
- Operational and Environmental Requirements:
    - Expected Physical Environment
    - Requirements for Interfacing with Adjacent Systems
    - Productization Requirements

    - Release Requirements
- Maintainability and Support Requirements:
    - Maintenance Requirements
    - Supportability Requirements
    - Adaptability Requirements
- Security Requirements:
    - Access Requirements
    - Integrity Requirements
    - Privacy Requirements
    - Audit Requirements
    - Immunity Requirements
- Cultural and Political Requirements
    - Cultural Requirements
    - Political Requirements
- Legal Requirements:
    - Compliance Requirements
    - Standards Requirements
- Open Issues: Issues that have been raised and do not yet have a conclusion
- Off-the-Shelf Solutions: is there anything that is ready made (components or full product) or even something you can copy

Safety Critical Systems: systems that ensure the safety of the users of the system. Generally, they are components of a larger system, e.g.

- Fire alarm
- Circuit breaker
- Airbags

Lecture 5

Defining Requirements

Types of projects:

- Rabbit:
    - Agile
    - Short life
- Horse:
    - Fast, strong, dependable
    - Most common in corporate
    - Medium longevity
- Elephant:
    - Solid, strong, long life

Artifact-driven: basing the requirements on data collection, questionnaires, etc.

- You can often collect too much data
- Only keep what you need to know

- *prune* the document space, so you only keep the useful data.

Scenario: similar to *storyboards*…

Positive Scenario: behaviour system should cover
- **Normal Scenario**: everything proceeds as expected
- **Abnormal Scenario**: an unusual behaviour, but the system still covers it

Negative Scenario: behaviour system should exclude, i.e. a mistake of the system

Knowledge Acquisition

Stakeholders: important to identify when determining who to customize the project towards
- Who is responsible for funding/using/managing the project?
- Caution: interactions with them must be done carefully

Domain expertise: what does the domain know / qualifications? Domain is who the project is directed towards

Lecture 6

Stakeholders-driven Elicitation Techniques: methods of knowledge acquisition
- Interviews
  - Single interview for multiple stakeholders: faster, but less involving
  - Steps:
    - Select stakeholders
    - Background study
    - Predesign sequence of questions, focused on concerns of present stakeholder(s)
    - Begin by asking easy questions
    - Keep focus during interview
    - Ask ‘why’-questions
    - Record answers and reactions
    - Write report from transcripts
    - Confirm report with stakeholders interviewed
  - Types:
    - **Structured**: predetermined set of questions
    - **Unstructured**: free discussion of current system
    - Optimal: start with *structure*, then do *unstructured*
- Observation:
  - people behave differently when observed
  - slow & expensive
  - focus on the system-as-is
- Group sessions: more than 4 people
  - invention from interactions within group of diverse people

Lecture 7

Inconsistencies: conflicting views or incorrect

Boundary Condition: the sample of instances where conditions conflict
Divergence: when two viewpoints have boundary conditions; they must be clarified

Strong conflict: non-satisfiable to the point of being logically inconsistent
Weak conflict: satisfiable without boundary condition

Lecture 8

Defect Detection Prevention (DDP): quantitative approach to risk analysis
DDP Risk management includes:

- Risk Consequence Tables
- Risk Countermeasure Tables

Risk Trees

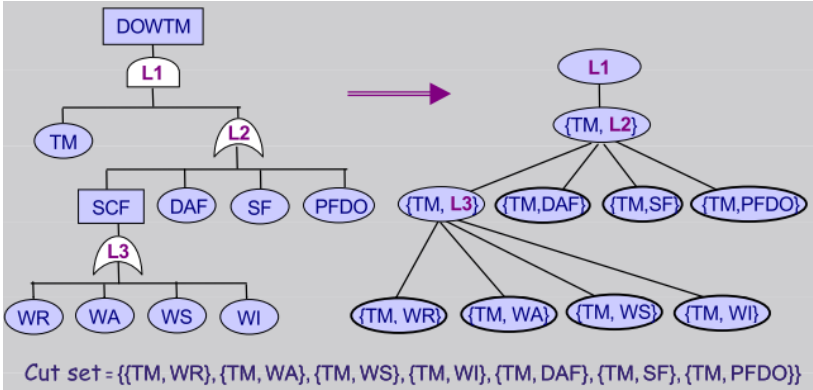
Risk Trees: (a.k.a. **fault tree**) a visual way of breaking down the causes of potential risks to identify where special attention needs to be placed in the design process

Components:

- Rectangles: can have children
- Ellipses: leaf nodes; may not have children
- AND / OR gates: you know how they work…

Cut Set

Cut set: the set of causes that result in the risk occurring
(diagram on right)



Think: sum of products

Qualitative Risk Assessment

Consequences	**Risk Likelihood** (*probability*)		
	Likely	Possible	Unlikely
risk 1	*Outcome*	*Outcome*	*Outcome*

Outcome can be Low, Moderate, High, Severe, or Catastrophic

Quantitative Risk Assessment

Consequences	**Risk Likelihood**					
	Likelihood levels 0.3		0.6		0.1	
	Likely		Possible		Unlikely	
risk 1	*Outcome*		*Outcome*		*Outcome*	

Likelihood levels: the total must equal one for the 

AHP Comparison Matrix

Analytic Hierarchy Process (AHP):

Attribute	Name	C_1	C_2	C_3	C_4	C_5	C_6
Functionality	C_1	≈	≈	>	⊃	>	>
Reliability	C_2	≈	≈	⊃	⊃	>	>
Usability	C_3	<	⊂	≈	⊏	⊐	⊐
Efficiency	C_4	⊂	⊂	⊐	≈	⊐	⊃
Maintainability	C_5	<	<	⊏	⊏	⊏	≈
Portability	C_6	<	<	⊏	⊂	≈	≈

Pairwise Comparisons

This is a way of seeing if your values for your AHP matrix are consistent.
Weights [w]: measure of importance from 0 to 1
Relationship Quantity [a]:

$$w_i = \sqrt[n]{\prod_{j=1}^{n} a_{ij}}$$

Although the sum of your weights, should equal 1, don't worry if it doesn't. Instead, normalize them by dividing them all by the sum of your weights.

$$x = \sum_{i=1}^{n} w_i$$

a_{xy}, where x is columns and y is rows
Also:

- for a(x<y), a(x,y) = 1.0/a(y,x)
- for a(x=y), a(x,y) = 1.0

i, j, and k are index variables with a range of the number of elements

a: a_{ij}; so $i = x$, $j = y$
b: a_{ik}
c: a_{kj}

Inconsistency coefficient [cm_A]: $cm_A = \max_{i,j,k}\left(\min\left(\left|1-\frac{a_{ij}}{a_{ik}a_{kj}}\right|,\left|1-\frac{a_{ik}a_{kj}}{a_{ij}}\right|\right)\right)$

Value and range of a_{ij}		relation	**Definition of intensity**
range	starting value	symbol	**or importance (C_i vs C_j)**
1.00-1.27	1	$C_i \approx C_j$	*indifferent*
1.28-1.94	1.6	$C_i \sqsupset C_j$	*slightly in favour*
1.95-3.17	2.6	$C_i \supset C_j$	*in favour*
3.18-6.14	4.7	$C_i > C_j$	*strongly better*
6.15-	7.0	$C_i \succ C_j$	*extremely better*

If the inconsistency coefficient is > 0.3, then you need to tweak your values.

Risk Consequence Table

For the table, fill in the *italic* parts.
Variables:
w: weight
l: likelihood
i: impact level

Objectives	**Risks**		
	risk 1 (likelihood: *likelihood*)	*risk 2* (likelihood: *likelihood*)	**Loss obj.**
Objective 1 (weight: *weight 1*)	*Impact level*	*Impact level*	$\text{loss}(y) = w(y)\times\sum_x i(x,y)\times l(x)$
Objective 2 (weight: *weight 1*)	*Impact level*	*Impact level*	$\text{loss}(y) = w(y)\times\sum_x i(x,y)\times l(x)$
Risk Criticality	$\text{risk}(x) = l(x)\times\sum_y i(x,y)\times w(y)$	$\text{risk}(x) = l(x)\times\sum_y i(x,y)\times w(y)$	

Risk Countermeasure Table

e: effectiveness

Consequences	**Risks**		
	risk 1 (likelihood: *likelihood*)	*risk 2* (likelihood: *likelihood*)	**Effectiveness of Countermeasure**
Countermeasure 1	*Effectiveness*	*Effectiveness*	$\text{eff}(y) = \sum_x e(x,y)\times\text{risk}(x)$
Countermeasure 2	*Effectiveness*	*Effectiveness*	$\text{eff}(y) = \sum_x e(x,y)\times\text{risk}(x)$

Risk Reduction	$\text{red}(x) = \sum_y e(x, y)$	$\text{red}(x) = \sum_y e(x, y)$	

Entity Relationship (ER) Diagram

UML

Entity: class of concept instances
Attribute 1 … Attribute *n*: intrinsic feature of an entity (regardless of other entities); public variables stored in the class, like hasHair or eyeColour for an Animal class

|
| relationshipName
|

Entity 2

arity: range of entities that contribute to the relationship

e.g.)

participant
Name Address e-mail

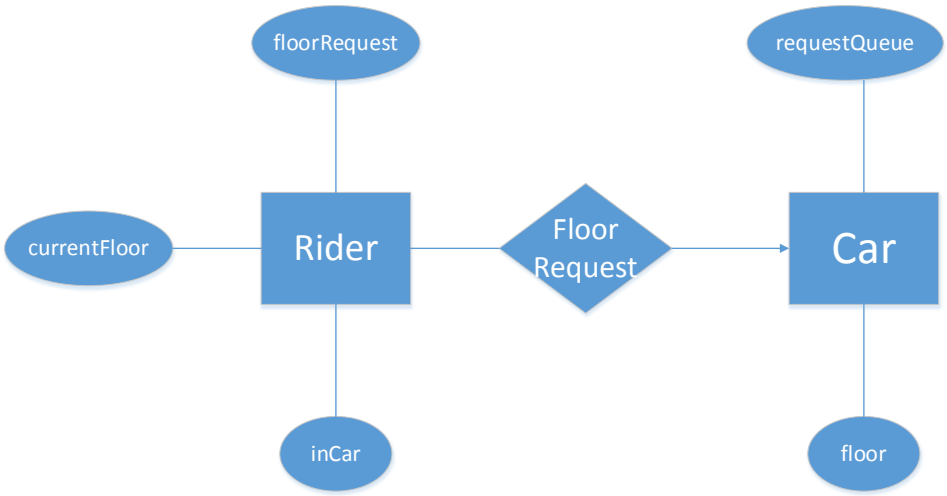
arity↓ |
1..* | invitedTo
| Invitation
0..* | invites
|

Standard Edition

entity: something external that interacts with your system

A way of representing a system, using entities

- **Rectangles** represent entity sets
- **Diamonds** represent relationship sets
- **Lines** link attributes to entity sets and entity sets to relationship sets
- **Ellipses** represent attributes
    - **Double ellipses** represent multivalued attributes
    - **Dashed ellipses** denote derived attributes
- **Dashed ellipses** denote derived attributes
- **Underline** indicates primary key attributes

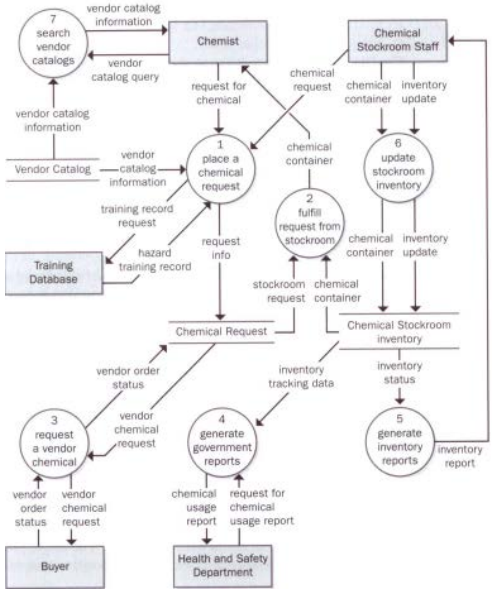


Data Flow Diagrams

Dataflow diagrams: shows the flow of information within and into the system, a more detailed version of a [context diagram]

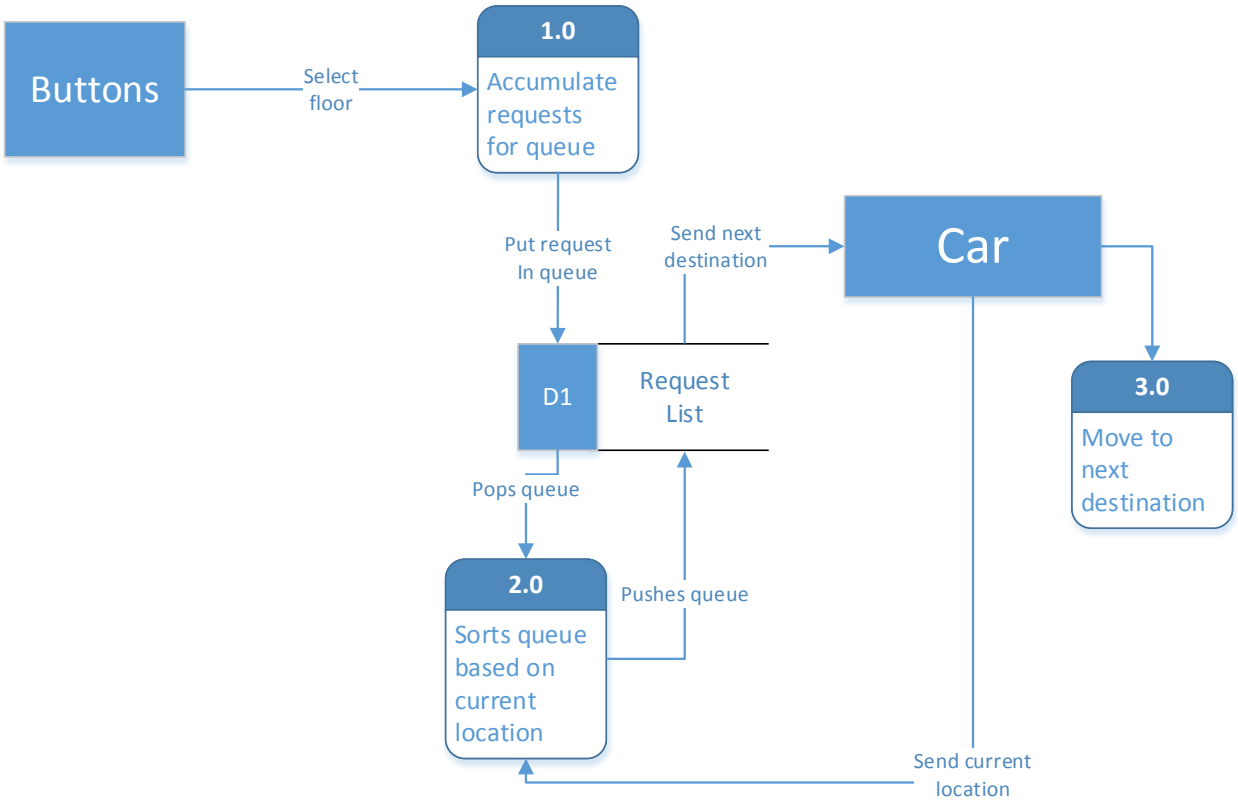
Version 1

Rectangles: actors outside of system (i.e. entities) who either input to or receive output from the system
Arrows: direction of flow of information, the description of the information is usually described along the length of the arrow
Circles / State Boxes: actions by system



Version 2

Solid Rectangles: Source / sink
Database containers: accesses from / inputs to specified data store
Semisweet rectangles: process

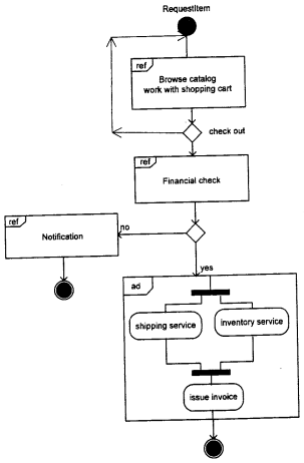


Activity Diagram

(actually not in the slides; copied from SFWR ENG 3A04)

Activity Diagram: data and control flow of system

- Rounded rectangles: actions in system
- Solid hub: fork and joint points
- Surrounded disk: terminate
- Diamond: decision
- Disk: start point



State Machine Diagram

Arrow:

- [constraint]: necessary input to get to next state
- flow: what the machine is doing

Circles: description of state

All states must go to a termination state!

Lecture 13

Fit Criteria

Fit criteria is the criteria that determines how well a solution fulfills the desired requirements

Non-functional: rationale, scale
Functional: how well did it satisfy the functions?

They *fit* if they are measureable

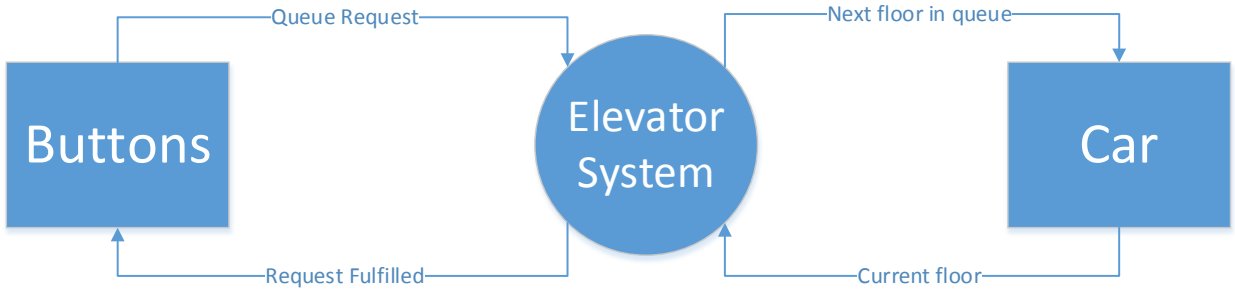
Lecture 14

Context Diagram

Context Diagram: defines the flow of information between entities and the system, a general data flow diagram

Rectangles: entities
Circle: system
Arrows: data flow

e.g.



Lecture 15

Use Case Diagrams: what can the actor(s) do?
Extend:

Include:

Lecture 17

Before-After Predicates

Before:
attribute : entity → {set of potential values of attribute}
After:
Processing based on values of attributes
e.g.
hasAuthorization(p) ∧ carriesPassport(p) ∧ ¬inBuilding(p) ⇒
peopleInBuilding' = peopleInBuilding ∪ {p} ∧
passportsAtDesk' = passportsAtDesk ∪ {passportOf (p)} ∧
inBuilding(p) ∧ ¬carriesPassport(p)

If you (p) *have authorization and a passport and you're not in the building, then peopleInBuilding becomes peopleInBuilding + you. Also, your passport is added to the list of passports on the desk. Also, you enter the building and you're no longer carrying your passport because you handed it into the front desk.*

Lecture 18

Temporal Logic: specifying and verifying properties of time-based systems

Linear Temporal Logic: an infinite sequence of states where each point in time has a unique successor

Linear temporal property: a temporal logic formula that describes a set of infinite sequences for which it is true

Future: the event occurs…	Past: the event occurred…
◊ (**F**) : some times in the ***F**uture*	♦ : some times in the *past*
□ (**G**): always in the *future*; **G**lobally	$\square^{-1}$: always in the *past*
○ (**X**): to be held at the ne**X**t state	
W : always in the *future* unless	**S**: always in the *past* **S**ince
U : always in the *future* until	**B**: always in the *past* **B**ack to

Note: future symbols can be mirrored as past by using the inverse sign or filling them in

Whitebox testing: Inspection

Review Process

Blackbox testing of system

- Free mode: no directive on where to find what
- Checklist-based: specific issues, defect types, RD parts

- Process-based: specific role for each reviewer, specific procedure, defect type, focus, analysis technique

Revision: updates of certain components and removing obsolete parts
Variant: different versions for different purposes

affects:
depends On:

SCR Tables
Each table outlines how one part of a system is set across all modes.

Inspection checklist
Make sure everything works.

Lecture 20
Stability: the probability of a feature to not change

Traceability Matrices
Types:
- Between Requirements
  - One axis is functional requirements
  - Other axis is non-functional requirements
  - *Which non-functional requirements satisfy the functional requirements?*
- Requirements and *test cases*
  - **Test case**: an example of how someone should use the system
  - One axis is functional requirements
  - Other axis is test cases
  - *Are the functional requirements useful?*

Lecture 22
The UNIX/LINEX system defines the rights {read, write, execute, append, list, modify, own}.
To modify the rights of another object, the subject needs to own the other object

Copy Flag:

e.g.
command *copy_all_rights*(*p*,*q*,*s*)
    enter *own* **into** *a*[*p*,*q*]
    if *c* **in** *a*[*p*,*s*]
        enter *read* **in** *a*[*q*,*s*]
        enter *write* **in** *a*[*q*,*s*]
        enter *execute* **in** *a*[*q*,*s*]
        enter *append* **in** *a*[*q*,*s*]

```
            enter list in a[q,s]
            enter modify in a[q,s]
            enter own in a[q,s]
        end
        delete c from a[q,s]
end
```

Lecture 24

Confidentiality: prevents the unauthorized disclosure of information
Integrity: unauthorized alternation of information
Availability:

disclosure: unauthorized access to information
disruption: interruption or prevention of correct operation
deception: acceptance of false data
usurpation: unauthorized control of some parts of a system

Security Levels

Think of **category sets** as a ring of keys that a person has. If someone has the keys {A, B}, they can open a door (access the document) with the locks {A}, {B}, and {A,B}. However, they can't open the doors {C}, {A,C}, {B,C}, nor {A,B,C}, even though they have some of the keys for the last 3 examples.

Mathematically
C(person) $\supsetneq$ C(document), you get read,
C(document) $\supsetneq$ C(person), you get write,
but if neither, then neither.

Note: $\varnothing \subseteq \{\text{every set}\}$

Bell-LaPadula Model

Security Clearance [I] OR **[i(s)]**: the privileges of a subject
Agent Security Clearance [I_S]: what is the person authenticated to? also could refer to a program / function that can execute other programs / functions
Object Security Clearance [I_O]: what security does the information have?

Hierarchy [L] (from highest to lowest):

1. TOP SECRET
2. SECRET
3. CONFIDENTIAL
4. UNCLASSIFIED

This model works in conjunction with the security levels:
If the security levels work, then assume *S* is true:

- Write: $I_S \leq I_O \wedge S_{write}$
- Read: $I_O \leq I_S \wedge S_{read}$

Lecture 25

Biba

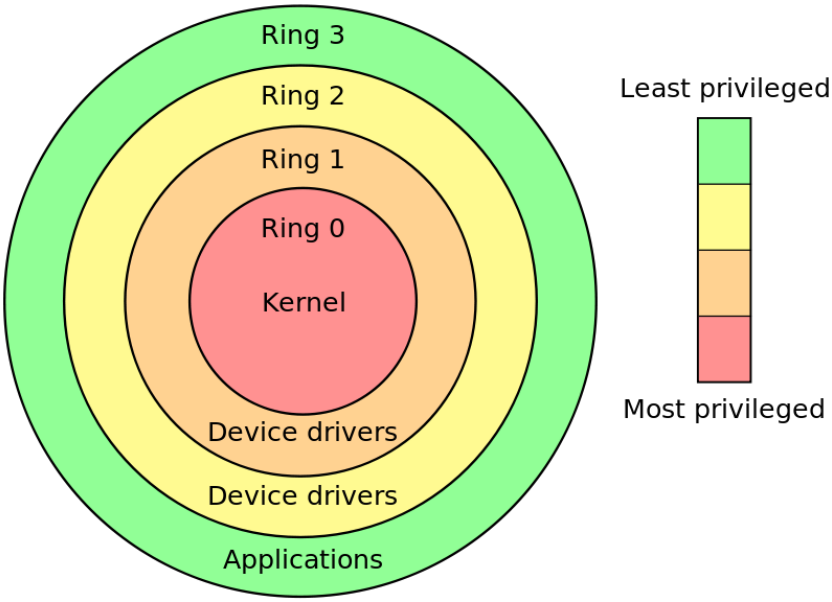
- Low-Water-Mark:
- Ring:
- Strict Integrity:

Low Water-Mark Policy

1. if s ∈ S reads o ∈ O, then i'(s) = min(i(s), i(o)), where i'(s) is the subject's integrity level after the read,
2. Subjects can write to objects iff i(o) ≤ i(s),
3. Subjects can execute anything that other subjects can iff they have higher integrity than them

Ring

- Subjects may read any objects, but there are restrictions on modification and execution
- Subjects can write to objects iff i(o) ≤ i(s)
- Subjects can execute anything that other subjects can iff they have higher integrity than them



Strict Integrity

- Pretty much Bell-LaPadula
- Read: i(s) ≤ i(o)
- Write: i(o) ≤ i(s)
- Execute: Subjects can execute anything that other subjects can iff they have higher integrity than them

Lecture 26

Private key [k]: only one person can access it
Public key [K]: everyone can access it

Diffie-Hellman scheme: the first public-private key encryption method

Find k such that $n = g^k \bmod p$, for a given set of natural numbers n, g and a prime number, p.

$K_a = g^{k_a} \bmod p$

$S_{A,B} = K_B^{k_A} \bmod p$

$S_{B,A} = K_A^{k_B} \bmod p$

$S_{A,B} = S_{B,A}$

Non-repudiation of origin: when you know for sure and where the message came from

Lecture 28

Vigenère Cipher

A version of Caeser cipher, which uses a word as a key (cipher), instead of a single letter to shift. It takes the message, splits it into blocks the length of the key and shifts each block by the key.