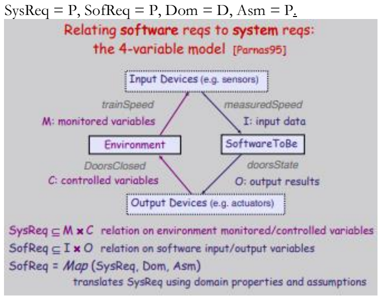
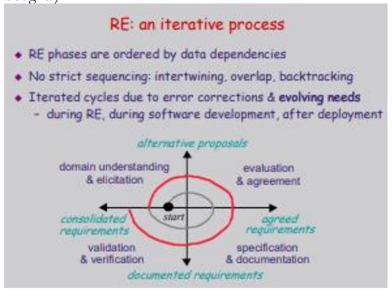
- <u>**Pre-Midterm**</u>

Descriptive statements (D): state system properties holding regardless of how the system should behave.
Prescriptive statements (P): state desirable properties holding or not depending on how the system behaves (uses "shall").
SysReq = P, SofReq = P, Dom = D, Asm = P.



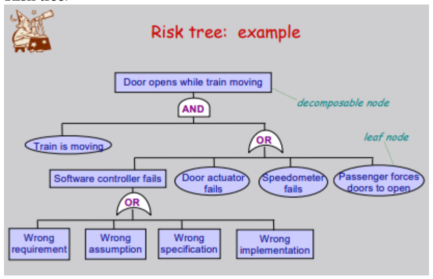
$SysReq \subseteq M \times C$ relation on environment monitored/controlled variables
$SofReq \subseteq I \times O$ relation on software input/output variables
SofReq = *Map* (SysReq, Dom, Asm)
translates SysReq using domain properties and assumptions

Functional reqs. are those which are related to the technical functionality of the system. **Non-functional req** is a req that specifies criteria that can be used to judge the operation of a system conditions, rather than specific behaviors.
Non-functional req categories: Look and Feel reqs, Usability reqs (ease of use, ease of learning), Performance reqs (speed reqs, safety critical reqs i.e. is the product physically dangerous?, precision reqs, reliability and availability reqs, capacity reqs), Operational reqs (expected physical env., expected technological env., partner aps), Maintainability and Portability reqs (how easy must it be to maintain this product? are there special conditions that apply to the maintenance of this product? portability reqs), Security reqs (is this product confidential?, File integrity reqs, audit reqs), Cultural and Political reqs (are there any special factors about the product that would make it unacceptable for some political reason?), Legal reqs (does the product fall under the jurisdiction of any law? are there any standards with which we must comply?), Open Issues (issues that have been raised and do not yet have a conclusion), Off-the-Shelf Solutions (is there a ready-made product that could be bought?).

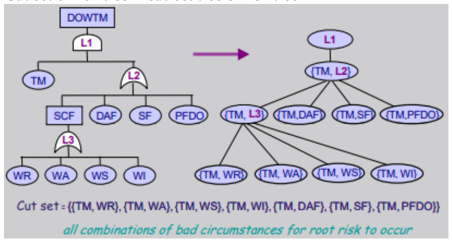

RE: an iterative process
- RE phases are ordered by data dependencies
- No strict sequencing: intertwining, overlap, backtracking
- Iterated cycles due to error corrections & **evolving needs**
  - during RE, during software development, after deployment

alternative proposals
domain understanding & elicitation
evaluation & agreement
start
consolidated requirements
agreed requirements
validation & verification
specification & documentation
documented requirements

Types of Projects: Rabbit (most agile), Horse (fast, strong, dependable), Elephant (solid, strong, long life, and long memory).
Knowledge Acquisition: organization, domain, system-as-is.
Stakeholders: people, groups, and orgs. that might be affected by the outcome of a project (WCO board of directors, WCO's marketing team, technical team, engineering team)
Artifact-driven elicitation techniques: background study (collect, read, synthesize doc), data collection (gather undocumented facts and figures), questionnaires, **scenarios**, prototypes.
Scenarios:
positive = one behavior the system should cover.
negative = one behavior the system should exclude.
normal = everything proceeds as expected.
abnormal = a desired interaction sequence in exception situation (still positive).
Requirements evaluation process (2nd step): inconsistency management -> risk analysis -> evaluating alternative options for decision making -> requirement prioritization.
Types of inconsistencies: terminology clash (same concept named differently in different statements), designation clash (same name for different concepts in different statements), structure clash (same concept structured differently in different statements), **strong conflict** (statement not satisfiable together, i.e. logically inconsistent statement: S, not S), **weak conflict** (statement not satisfiable together under some boundary condition, i.e. strongly conflicting if B holds, i.e. strongly conflicting if the boundary condition holds therefore it is a potential conflict).
Types of risk: product-related and process-related.
Risk tree:



Cut set of risk tree + cut-set tree of risk tree:



Cut set = {{TM, WR}, {TM, WA}, {TM, WS}, {TM, WI}, {TM, DAF}, {TM, SF}, {TM, PFDO}}
all combinations of bad circumstances for root risk to occur

Qualitative risk assessment table:

	Risk likelihood		
Consequences	**Likely**	**Possible**	**Unlikely**
Loss of life	Catastrophic	Catastrophic	Severe
Train car damaged	High	Moderate	Low
No. of passengers decreased	High	High	Low
Bad airport reputation	Moderate	Low	Low

Quantitative risk assessment table (likelihood levels must total 100%):

	Risk likelihood					
	0.3		0.6		0.1	
Consequences	**Likely**		**Possible**		**Unlikely**	
Loss of life	Catastr.	7.0	Catastr.	7.0	Severe	4.7
Train car damaged	High	2.6	Moderate	1.6	Low	1.0
# passengers decreased	High	2.2	High	2.6	Low	1.0
Bad airport reputation	Moderate	1.6	Low	1.0	Low	1.0

Risk exposure *Exp* for risk *r* with independent consequences *c*:

$$Exp(r) = \sum_c L(c)S(c),$$

where
- $L(c)$ - likelihood of c (probability, total equals 1.0)
- $S(c)$ - severity of c (grade)

For our example we have $Exp(r) = 0.3(7.0+2.6+2.0+1.6) + 0.6(7.0+1.6+2.6+1.0) + 0.1(4.7+1.0+1.0+1.0) = 12.71$

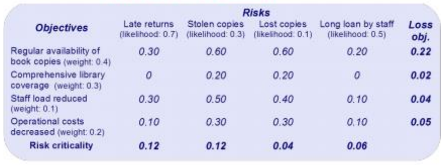
Countermeasures:
- Cost-effectiveness is measured by **risk-reduction leverage**:

$$RRL(r, cm) = \frac{Exp(r) - Exp(rcm)}{Cost(m)},$$

where: $Exp(r)$: exposure of risk r, and
$Exp(rcm)$: new exposure of r if countermeasure cm is selected

⇒ Select countermeasures with highest RRLs (be careful, numbers are not reliable!)

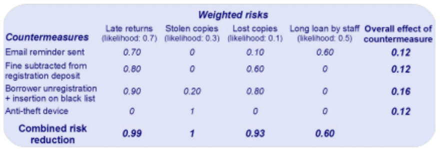
DDP (Defect Detection Prevention): elaborate risk impact matrix -> elaborate countermeasure effectiveness matrix -> determine optimal balance risk reduction / countermeasure cost.
Impact matrix:
- For each objective *obj* and risk *r*, we provide:
  $Impact(r, obj)$ = estimated loss of satisfaction of *obj* by *r*
  0 (no loss) → 1 (total loss)



	Risks				
Objectives	Late returns (likelihood: 0.7)	Stolen copies (likelihood: 0.3)	Lost copies (likelihood: 0.1)	Long loan by staff (likelihood: 0.5)	Loss obj.
Regular availability of book copies (weight: 0.4)	0.30	0.60	0.60	0.20	**0.22**
Comprehensive library coverage (weight: 0.3)	0	0.20	0.20	0	**0.02**
Staff load reduced (weight: 0.1)	0.30	0.50	0.40	0.10	**0.04**
Operational costs decreased (weight: 0.2)	0.10	0.30	0.30	0.10	**0.05**
Risk criticality	**0.12**	**0.12**	**0.04**	**0.06**	

- Risk Criticality:
  $RC(r) = Likelihood(r) \sum_{obj} Impact(r, obj) Weight(obj)$
- Objective Loss:
  $Loss(obj) = Weight(obj) \sum_r Impact(r, obj) Likelihood(r)$

Effectiveness matrix:
- For each countermeasure *cm*, weighted risk *r*, we provide:
  $Reduction(cm, r)$ = estimated reduction of *r* if *cm* applied
  0 (no reduction) → 1 (risk eliminated)

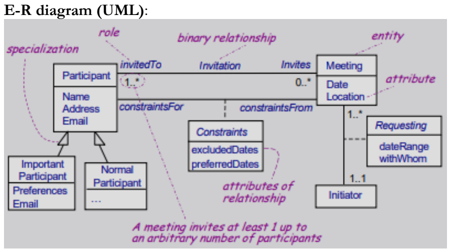


	Weighted risks				
Countermeasures	Late returns (likelihood: 0.7)	Stolen copies (likelihood: 0.3)	Lost copies (likelihood: 0.1)	Long loan by staff (likelihood: 0.5)	Overall effect of countermeasure
Email reminder sent	0.70	0	0.10	0.60	**0.12**
Fine subtracted from registration deposit	0.80	0	0.60	0	**0.12**
Borrower unregistration + insertion on black list	0.90	0.20	0.80	0	**0.16**
Anti-theft device	0	1	0	0	**0.12**
Combined risk reduction	**0.99**	**1**	**0.93**	**0.60**	

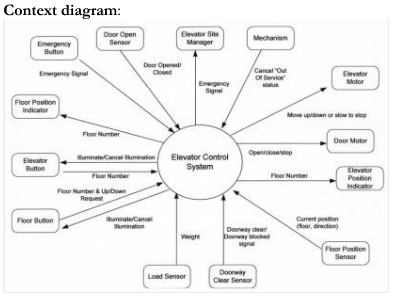
- Combines Reduction:
  $CR(r) = 1 - \prod_{cm}(1 - Reduction(cm, r))$
- Overall Effect: $OE(cm) = \sum_r Reduction(cm, r) RC(r)$,
  where $RC(r)$ is 'Risk Criticality' defined in *Impact matrix*.

- <u>**Post-Midterm**</u>

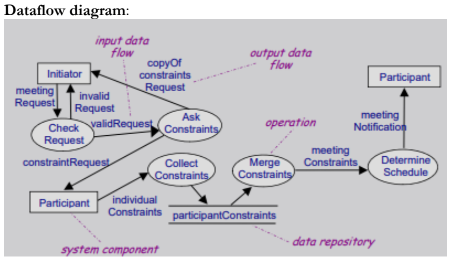
Fit criteria: makes statements measurable (does not indicate how this conformance is to be tested).
Scale of measurement: the unit used to test conformance of the product.
Rationale: the reason or justification for a req.
If the req is give in correctly quantified terms, then the fit criterion and the req are the same.
Diagrammatic notations: conceptual structures (entity-relation diagrams), information flows (context diagrams and dataflow diagrams), system behaviors (state machine diagrams), system operations (use case diagrams), modeling scenarios (activity diagram), everything (UML).
E-R diagram (UML):



Context diagram:



Dataflow diagram:



- <u>**Formal methods and Discrete Mathematics**</u>

Types of formal methods: standard discrete mathematics and predicate logic, temporal logic, tabular expressions.
Union: the union of two sets A and B is the set of elements which are in A, in B, or in both A and B.
Intersection: the intersection of two sets A and B is the set of all objects that are members of both the sets A and B.
A x B = {(a, b) | a ∈ A ∧ b ∈ B}: the **set** of all ordered pairs (a, b) such that a ∈ A and b ∈ B.
Formal Specification:
- Domain and Notation.

  FLOOR : *set of all floors served by elevators*
  ELEVATOR : *set of all elevators in the building*
  $elevatorDirection : ELEVATOR \rightarrow \{up, down, hold\}$
  $floorsPressed$: set(MAX_FLOORS)
  $isPressed : FLOOR \rightarrow \{true, false\}$
  $curFloor : \rightarrow FLOOR$

  $f \in FLOOR;$
  $e \in ELEVATOR$

- Statement.

  $(isPressed(f) \wedge (f > curFloor)) \Rightarrow$
  $(floorsPressed' = floorsPressed \cup \{f\} \wedge doorState(e) =$
  $closed \wedge elevatorMoving = true \wedge elevatorDirection = up)$

  $(isPressed(f) \wedge (f < curFloor)) \Rightarrow$
  $(floorsPressed' = floorsPressed \cup \{f\} \wedge doorState(e) =$
  $closed \wedge elevatorMoving = true \wedge elevatorDirection = down)$

LTL (Linear Temporal Logic):
$\bigcirc\varphi$ - φ is true in the **next** moment in time (*Next*)
$\Box\varphi$ - φ is true in **all** future moments (*Always*)
$\Diamond\varphi$ - φ is true in **some** future moments (*Sometimes*)
$\varphi\mathbf{U}\psi$ - φ is true **until** ψ is true (*Until*)
$\neg\Box\varphi = \Diamond\neg\varphi$
$\Diamond\varphi = true\mathbf{U}\varphi$
$\Diamond(\varphi \vee \psi) \equiv \Diamond\varphi \vee \Diamond\psi$
$\Box(\varphi \wedge \psi) \equiv \Box\varphi \wedge \Box\psi$
$\neg\bigcirc\varphi \equiv \bigcirc\neg\varphi$
$\neg(\varphi\mathbf{U}\psi) \equiv (\neg\psi\mathbf{U}(\neg\varphi \wedge \neg\psi)) \vee \Box\neg\psi$

1. A passenger entering the elevator at 5th floor and pushing 2nd floor button, will never reach 6th floor, unless 6th floor button is already lighten or somebody will push it, no matter if she/he entered an upwards or upward travelling elevator.

   In this case you might use the following atomic predicates: $floor = 2, direction = up, direction = down, ButtonPres2, floor = 5$, etc

   $\Box((floor = 5 \wedge ButtonPressed2 \wedge (direction = up \vee direction = down)) \Rightarrow \neg floor = 6 \vee \Diamond(floor = 6 \wedge ButtonPressed6))$

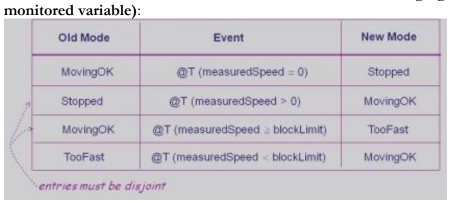
2. It is possible to get a state where started holds but neither ready nor payed holds. *started*, *ready*, and *payed* are atomic predicates.

   $\Diamond(start \wedge \neg ready \wedge \neg payed)$

3. Between the events *started* and *finished*, *not_ready* is never true. *started*, *finished*, and *not_ready* are atomic predicates.

   $started \Rightarrow \neg not\_ready\ U\ finished$

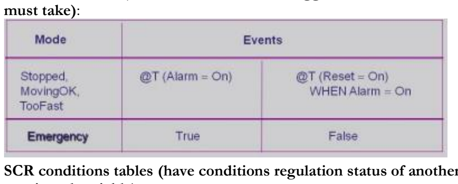
Tabular expression:
SCR mode transition Tables (state machines based on a certain set of monitored variables and a transition event occurs at the changing of the monitored variable):



Old Mode	Event	New Mode
MovingOK	@T (measuredSpeed = 0)	Stopped
Stopped	@T (measuredSpeed > 0)	MovingOK
MovingOK	@T (measuredSpeed ≥ blockLimit)	TooFast
TooFast	@T (measuredSpeed < blockLimit)	MovingOK

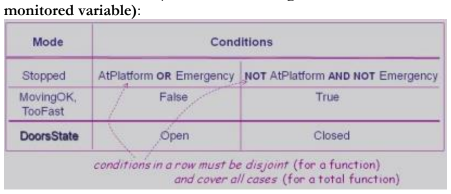
entries must be disjoint

SCR event tables (we define an event that triggers actions that the system must take):



Mode	Events	
Stopped, MovingOK, TooFast	@T (Alarm = On)	@T (Reset = On) WHEN Alarm = On
Emergency	True	False

SCR conditions tables (have conditions regulation status of another monitored variable):



Mode	Conditions	
Stopped	AtPlatform OR Emergency	NOT AtPlatform AND NOT Emergency
MovingOK, TooFast	False	True
DoorsState	Open	Closed

conditions in a row must be disjoint (for a function)
and cover all cases (for a total function)

- <u>**Assignment 3 Related**</u>

Process-based reviewing plan:
Inspector 1: Risk Management Role: This inspector is responsible for evaluating the risks associated with each failure. The inspector will need to evaluate the functional requirements in order to determine whether any requirements can generate costly risks as well as non-functional requirements in order to produce a weighted matrix of consequences. Included in this role is the responsibility of reviewing any legal requirements and the inspector will be expected to report the highest ranked risk along with the counter measures to minimize those risks.

Inspector 2: WCO Management Role: This inspector is responsible for checking the functional and non-functional Requirements for adequacy and completeness from the perspective of WCO management. The purpose is to verify that the system supports all the objectives that WCO has for the system. This inspector shall ensure that all functionalities needed for upper management to use the system are present.

Inspector 3: User Role: This inspector is responsible for checking the functional requirements for adequacy and completeness from the perspective of a user or patron at WCO. Their purpose is to ensure that all users can interact with the system in the desired way. The inspector should include evaluations for various interactions, such as interacting with the WCO software, task scheduling, GPS navigation and song selection.

Inspector 4: Implementation: This inspector is responsible for assuring the feasibility in implementing the system with respect to budget and time. This inspector will review the non-functional requirements of the system so that they can accurately access the feasibility of implementing a system that satisfies all non-functional requirements. The focus should be on gauging the expected cost of the system-to-be, and reporting whether initial estimates remain realistic

Inspection checklist:

Defect-based Checklist: OMISSION: (Are the objs. of the system clearly defined? Are all acronyms defined within the reqs. doc?), CONTRADICTIONS: (Are all reqs. consistent with objs. and constraints? Are all requirements created by different stakeholders consistent with each other?), INADEQUACIES: (Do all reqs. capture what WCO Management expects?), AMBIGUITIES: (Are there reqs. that can be interpreted differently than the og intention? Are all terms used w/in the reqs. doc. consistent?) IMMEASURABILITY: (Do all reqs. contain a measurable fit criterion? Are all measurements feasible to perform?), OVER SPECIFICATION: (Do any reqs. specify a design decision?).

Quality-specific Checklist: Are the criteria for an easy to navigate application defined? Is the domain of easily assessable information defined? Are there any unspecified responses to receiving any input values? Are there any inputs unused by the system? Is there a data consistency check made before saving the data in the system (or sharing it with external systems)?

Domain-specific Checklist: further specialization to domain concepts and operations (are there mechanisms in place to ensure the security of user data?).

Language-based Checklist: Is there a value for every field in this template instantiation?

Estimating stability level:

High stability reqs: Functional req more stable than non-func., stuff unlikely to change in the future, is a part of authentication process, is a part of process to allow users to use app., an extension of the app's functionality.

Medium stability reqs: Things that MIGHT need adjustments, things that MIGHT have more options in the future.

Low stability reqs: Things that system has no control over (external systems), Definition of words (simple, age), things that can change easily at customer's request and w/ technological advancements (things running faster).

Traceability matrix:

Illustrates logical links b/w individual functional reqs. and other stuff like other func. reqs., test cases, etc...

Name reqs. R1, R2, etc... Name tests. T1, T2, etc...

Tests: first -> scenario (user starts app), second -> expected results (display info), last -> malfunction (Nothing).

Confidentiality: The concealment of information or resources.

Integrity: Trustworthiness of data or resources. Data integrity refers to the content of the information and the origin integrity refers to the source of the data, often called authentication. Integrity mechanism: two classes -> prevention mechanism: blocking, detection mechanism: analyzing.

Example of a situation in which a compromise of confidentiality leads to compromise in integrity:

If a company terminates an employee, and does not immediately disable the employee's credentials, the employee would then still be able to access a system which should be confidential to them. This breach in confidentiality leads to a compromise in integrity since the company can no longer be sure of the accuracy of the data contained in those files accessed through the breach.

- <u>**Access Control Matrix**</u>

Protection state: the state of a system is the collection of the current values of all memory locations, all secondary storage, and all registers and other components of the system. The subset of this collection that deals w/ protection is the protection state of the system.

Access control matrix: a tool that can describe the current protection state.

Example

Objects

Subjects	file 1	file 2	process 1	process 2
process 1	rwo	r	rwxo	w
process 2	a	ro	r	rwxo

$R = \{$**read**, **write**, **execute**, **own**, **append**$\}$

- The **own** right is a distinguished right, it often involves ability to add and delete rights for other users.
- For example, *process 1* could alter $A[x, file1]$ for any subject, and *process 2* could alter $A[x, file2]$ for any subject.

The relationship b/w these entities is captured by a matrix A w/ rights drawn from a set of rights R. The subject s has the set of rights A[si, oj] over the object o.

Verbs have a default rule: "closed": access denied unless explicitly granted (0), "open": access granted unless explicitly denied (1). The set of protection states of the system is represented by the triple (S, O, A).

Protection state is (S, O, A) before the execution of each command and (S', O', A') after each command (prime notation).

Copy right: allows the possessor to grant rights to another. By principle of attenuation, only those rights the grantor possesses may be copied (copy flag (right considered a flag attached to other rights): r is read right that cannot be copied and rc is read right that can be copied).

Own right: enables possessors to add or delete privileges for themselves. Also allows the possessor to grant rights to others.

Principle of attenuation: a subject may not give rights it does not possess to another. (i.e. if not applied at all any subject can grant any other subject ownership rights, or any rights, over any object. That subject can then gran the original subject the same privileges. In this way, subjects can gain full access to any part of the system. If only applied to access rights such as read and write, this prevents users from granting each other read or write access unless they possess the right themselves, but since ownership and rights granting do not adhere to the principle, subjects can still grant each other ownership rights. With ownership rights, a subject could then grant themselves read or write access w/o violating the principle, so restricting the principle to read and write does not change anything).

Primitive commands: create subject s, create object o, add right: enter r into A[s, o], delete right: delete r into A[s, o], destroy subject: destroy subject s, destroy object: destroy object o.

First command causes p to delete all rights the subject q has over an object s. Second command modifies first command so that deletion can occur only if p has modify rights over s. The third command modifies the first command so that the deletion can occur only if p has modify rights over s and q does not have own rights over s.

```
command delete_all_rights(p,q,s)
        enter own into a[p,q]
        delete read from a[q,s]
        delete write from a[q,s]
        delete execute from a[q,s]
        delete append from a[q,s]
        delete list from a[q,s]
        delete modify from a[q,s]
        delete own from a[q,s]
end
```

```
command delete_all_rights(p,q,s)
        if modify in a[p,q]
                enter own into a[p,q]
                delete read from a[q,s]
                delete write from a[q,s]
                delete execute from a[q,s]
                delete append from a[q,s]
                delete list from a[q,s]
                delete modify from a[q,s]
                delete own from a[q,s]
        end
end
```

```
command delete_all_rights(p,q,s)
        if modify in a[p,q] AND own NOT in a[p,s]
                enter own into a[p,q]
                delete read from a[q,s]
                delete write from a[q,s]
                delete execute from a[q,s]
                delete append from a[q,s]
                delete list from a[q,s]
                delete modify from a[q,s]
                delete own from a[q,s]
        end
end
```

- <u>**Security**</u>

Definition (Secure, Precise and Broad Security Mechanisms)

Let P be the set of all possible states. Let Q be the set of secure states (as specified by the security policy). Let the security mechanisms restrict the system to some set of states R.

- A security mechanism is secure if $R \subseteq Q$
- A security mechanism is precise if $R = Q$
- A security mechanism is broad if $\exists (r \mid r \in R : r \notin Q)$

Three main components of security: confidentiality, integrity, availability.

Availability: the ability to use information or resources desired.

Threats: a threat is a potential violation of security. Four classes: disclosure (unauthorized access to info), deception (acceptance of false data), disruption (interruption or prevention of correct operation), usurpation (unauthorized control of some parts of a system). Common threats: snooping (unauthorized interception of info), alteration or modification (an unauthorized change of info), spoofing (an impersonation of one entity by another), repudiation of origin (a false denial that an entity has sent something), denial of receipt (a false denial that an entity received some information or message), delay (temporary inhibition of a service), denial of service (a long term inhibition of a service)

Security policy: statement of what is, and what is not allowed. A statement that partitions the state of the system into a set of authorized and a set of unauthorized states (Let X be a set of entities and I be some info. Then I has the property of: confidentiality w.r.t X if no member of X can obtain info about I, integrity w.r.t X if all members of X trust I, availability w.r.t X if all members of X can access I).

Types of security policies: Military (security policy developed primarily to provide confidentiality. Also called governmental), commercial (commercial security policy is a security policy developed primarily to provide integrity).

Types of access control: discretionary (an individual user can set access control mechanism to allow or deny access to an object), mandatory (a system mechanism controls access to an object and an individual user cannot alter the access), originator controlled (the create of an object maintains the access control).

Security mechanism: method, tool, or procedure for enforcing a security policy.

Strategies for security: prevent an attack (refers to failure of an attempted attack on system), detect an attack (accept that an attack will occur and attempts to determine if an attack is underway or has occurred), recover from an attack (refers to ability to stop an attack and to assess and repair any damage caused by that attack).

Quality assurance: basis for determining "how much" to trust a system: specification, design, implementation.

Operational issues: any useful policy and mechanism must balance the benefits of protection against the cost of designing, implementing and using the system. Cost benefit analysis, risk analysis and laws and customs.

Human issues: if a security is configured or used incorrectly then the best security control can be useless and worse. Organizational problems, people problems.

Confidentiality policies: prevents the unauthorized disclosure of information.

Subject has a security clearance and object has a security classification.

Bell-LaPadula model: TS -> S -> C -> UC. Subject S can read an obj O if its clearance >= clearance of O and S has discretionary read access to O. A subject S can write to an obj O if its clearance <= clearance of O and has discretionary write access to O (subjects can write w/o reading). No subject should be able to read objs unless reading them is necessary for that subject to perform its functions. Subject can access if level is appropriate AND if subject has access to the same categories or a subset of the categories. Security level = (LEVEL, CATEGORY). A subject S can read an obj O <-> S dom O and S has discretionary read access to O. A subject S can write to an object O <-> O dom S ^ S has discretionary write access to O.

Definition (Biba Model)

A system consists of a set S of subjects, a set O of objects, and a set I of integrity levels. For each $i_1, i_2 \in I$, we say $i_1 < i_2$ if i_2 *dominates* i_1. We assume $<$ is a total order (for each i_1, i_2: $i_1 < i_2 \wedge i_2 < i_1 \wedge i_1 = i_2$).

- The function $\min(i_1, i_2)$ giver the smaller (w.r.t. $<$) element.
- The function $i : S \cup O \to I$ returns the integrity level of an object or a subject.
- The relation $\mathbf{r} \subseteq S \times O$ defines the ability of a subject to read an object.
- The relation $\mathbf{w} \subseteq S \times O$ defines the ability of a subject to write to an object.
- The relation $\mathbf{x} \subseteq S \times O$ defines the ability of a subject to invoke (execute) another object.

Low water mark policy:

Definition

Whenever a subject accesses and object, the policy changes the integrity level of the subject to the lower of the subject and the object. Specifically:

1. $s \in S$ can write $o \in O \iff i(o) \leq i(s)$,
2. if $s \in S$ reads $o \in O$, then $i'(s) = \min(i(s), i(o))$, where $i'(s)$ is the subject's integrity level after the read,
3. $s_1 \in S$ can execute $s_2 \in S \iff i(s_2) \leq i(s_1)$.

Ring policy:

Definition (Ring Policy)

1. Any subject may read any object, regardless of integrity level.
2. $s \in S$ can write $o \in O \iff i(o) \leq i(s)$.
3. $s_1 \in S$ can execute $s_2 \in S \iff i(s_2) \leq i(s_1)$.

Difference b/w ring and low water mark is in ring. any subject can read any objects.

Strict integrity policy:

Definition (Strict Integrity Policy)

1. $s \in S$ can read $o \in O \iff i(s) \leq i(o)$.
2. $s \in S$ can write $o \in O \iff i(o) \leq i(s)$.
3. $s_1 \in S$ can execute $s_2 \in S \iff i(s_2) \leq i(s_1)$.

- The Policy Constraint Theorem still holds.
- Rules (1) and (2) imply that if both *read* and *write* are allowed: $i(o) = i(s)$.
- Like the low water mark policy, this policy prevents indirect as well as direct modification of entities without authorization.

Cryptography:

Definition (Cryptosystem)

A cryptosystem is a 5-tuple (E, D, M, K, C), where:

- M is the set of plaintexts,
- K is the set of keys,
- $E = \{E_k \mid k \in K\}$, where each $E_k : M \to C$, is the set of enciphering functions,
- $D = \{D_k \mid k \in K\}$, where each $D_k : C \to M$, is the set of deciphering functions.

Example (Caesar Cipher)

- Idea: letters are shifted and *key*=shift. If $k = 3$ the $A \to D, B \to E, \dots, Z \to C$, and *"HELLO"* $\to$ *"KHOOR"*.
- M = *all sequences in Roman letters* $= \{A, B, C, \dots, Z\}^*$,
- $K = \{i \mid 0 \leq i \leq 25\}$, or $K = \{i \mid 1 \leq i \leq 26\}$,
- $E = \{E_k \mid k \in K\}$, where for each $m \in M, E_k(m) = m_k$, and m_k is derived from m by shifting each letter by k,
- $D = \{D_k \mid k \in K\}$, where for each $c \in C, D_k(c) = c_k$, and c_k is derived from c by shifting back each letter by k
- $C = M$.

'WE ARE DISCOVERED. FLEE AT ONCE' is used, the cipherer writes out:

```
W . . . E . . . C . . . R . . . L . . . T . . . E
. E . R . D . S . O . E . E . F . E . A . O . C .
. . A . . . I . . . V . . . D . . . E . . . N . .
```

Then reads off to get the ciphertext:

```
WECRLTEERDSOEEFEAOCAIVDEN
```

^rails = 3.

Definition

The Vigenère cipher chooses a sequence of keys, represented by a string. The key letters are applied to successive plaintext characters, and when the end of the key is reached, the key starts over. The length of the key is called the *period* of the cipher.

Letters enumeration:

$A \to 0, \dots, G \to 6, \dots, I \to 8, \dots, V \to 21, \dots, Z \to 25.$

Message: THE BOY HAS THE BALL

Key: VIG or 21-8-6.

We encipher using Caesar cipher for each letter:

Plaintext	T H E B O Y H A S T H E B A L L
Keys	V I G V I G V I G V I G V I G V
Ciphertext	O P K W W E C I Y O P K W I M G,

since $(T + V) \mod 26 = O, (H + I) \mod 26 = P$, $(E + G) \mod 26 = K$, etc.

Public key cryptography:

Two keys:

- PRIVITE KEY known only to individual
- PUBLIC KEY available to anyone

A has private key k_A and public key K_A, while B has private key k_B and public key K_B. A message send by A and encrypted using k_A and K_B practically can only be decrypted when B will use k_B and K_A. How is it possible?

The private key k and the public key K *are not random.*

The public key K is a *function* of the private key k, i.e. $K = f(k)$ for **some** function f (hence $K_A = f(k_A)$ and $K_B = f(k_B)$).

The function f must have the property that for any K, finding $f^{-1}(K)$ is practically impossible.

Algorithm (Diffie-Hellman Protocol)

- *Shared Knowledge: p and g, where* $g \neq 0, 1, p-1$.
- *Each user chooses a private key* k_a *and computes a public key* $K_a = g^{k_a} \mod p$.
- *If A and B want to communicate, they encipher the other's public key using they own public key using the formulas:*
  $S_{A,B} = K_B^{k_A} \mod p$ *(used by A), and*
  $S_{B,A} = K_A^{k_B} \mod p$ *(used by B).*
- *The protocol is based on the following theorem:*

Theorem

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

- *The key* $S_{A,B} = S_{B,A}$ *is used for communication between A and B.*

$(g^a \bmod p)^b \bmod p = g^{ab} \bmod p = (g^b \bmod p)^a \bmod p = g^{ba} \bmod p$.

Same computation therefore same shared key.

RSA protocol:

Definition

A number k is relatively prime to a number n if k has no factors in common with n.

Definition

The totient function $\Phi(n)$ is the number of positive integers less than n and *relatively prime* to n.

Algorithm (RSA Protocol)

- *Choose two large prime numbers* p *and* q.
- *Compute* $n = pq$. *Then* $\Phi(n) = (p-1)(q-1)$.
- *Choose* $e < n$ *such that* e *is* relatively prime *to* $\Phi(n)$.
- *Compute* d *such that* $ed \mod \Phi(n) = 1$.
- *PUBLIC KEY:* (e, n)
- *PRIVITE KEY:* d
- *ENCIPHER:* $c = m^e \mod n$ *(uses PUBLIC KEY* (e, n)*)*
- *DECIPHER:* $m = c^d \mod n$ *(uses PRIVATE KEY* d*)*

Prime numbers: only divisible by 1 and itself.

Checksums are mainly used to detect transmission errors, however, they can also indicate attacks during transmissions.

Key management:

Notation

$$X \to Y : \{Z\}k$$

The entity X sends entity Y a message Z enciphered with key k

- k_A *- key belongs to A*
- k *- secret key in classical cryptosystems*
- e *-public key (in public key cryptosystem)*
- d *- private key (in public key cryptosystem)*
- $a||b$ *- concatenation of bit sequences a and b.*

Key exchange enables A to communicate secretly with B and vice versa, using a shared cryptographic key.

Interchange key: cryptographic key associated with a principal to a communication.

Session key: a cryptographic key associated w/ the communication itself.

Key exchange: key cannot be transmitted in the clear, A and B must decide to trust a third part C, the cryptosystems and protocols are publicly known. The only secret data are keys.

Needham-Schröder Protocol

Assumptions:

- $rand_1, rand_2$ are two random numbers, they cannot repeat between different protocol changes.
- A and B trust C.

PROTOCOL:

1. $A \to C : \{A||B||rand_1\}$
2. $C \to A : \{\underbrace{A||B||rand_1||k_{session}||\{A||k_{session}\}k_B}_{message}\}\, k_A$
3. $A \to B : \{A||k_{session}\}k_B$
4. $B \to A : \{rand_2\}k_{session}$
5. $A \to B : \{rand_2 - 1\}k_{session}$

Now B can verify that it was A who sent the message.