System Design

- transforms the analysis model by
  - defining the design goals of the project
  - decomposing the system into smaller subsystems
  - selection of off-the-shelf and legacy components
  - mapping subsystems to hardware
  - selection of persistent data management infrastructure
  - selection of access control policy
  - selection of global control flow mechanism
  - handling of boundary conditions

System Design is

- a creative process
  - no cook book solutions
- goal driven
  - we create a design for solving some problem
- constraint driven
  - by the function to be served and the constructions which are possible
- good designs can be recognised
  - simple, coherent, adequately meets requirements, adaptable

System Design Activity Diagram
Identifying Subsystems

A first step in system design is to break down the solution domain into simpler parts. A **SUBSYSTEM** is a collection of classes, associations, operations, events and constraints that are inter-related. Identifying subsystems usually involves backtracking, evaluation and revision of various solutions. It is important to get the decomposition right:

- subsystems implemented by different teams
- bad decomposition can lead to unworkable designs

Heuristics to Identify Subsystems

- Consider the objects and classes in your requirements analysis models.
- Try grouping **objects** into **subsystems** by:
  - assigning objects in one use case into the same subsystem
  - create a dedicated subsystem for objects used for moving data among subsystems
  - minimizing the number of associations crossing subsystem boundaries
  - ensure all objects in the same subsystem are functionally related

Some further criteria

- **Primary Question**: what kind of service is provided by the subsystems?
- **Secondary Question**: Can the subsystems be hierarchically ordered (layers)?

- Criteria for selecting subsystems: most of the interaction should be within a subsystem and not across subsystem boundaries (we’ll return to this idea)
Modular design

- A design is **modular** when
  - each activity of the system is performed by exactly one component
  - inputs and outputs of each component are **well-defined**, in that every input and output is necessary for the function of that component
  - the idea is to minimise the impact of later changes by abstracting from implementation details

Coupling

- Coupling is the strength of dependencies **BETWEEN** two subsystems
- In general, the fewer dependencies between subsystems the better
- Why are fewer dependencies better?

![Diagram showing coupling types](image)
Coherence / Cohesion

- Coherence (or cohesion) is the strength of dependencies WITHIN a subsystem
- In general, the stronger the dependencies within a subsystem the better
- Why is (fairly) strong coherence best?

**FRIEND Objects - after Analysis**

- FieldOfficer (E)
- FieldOfficerStation (B)
- ReportEvent (C)
- EmergencyReportForm (B)
- EmergencyReport (E)
- Acknowledgement (B)
- Incident (E)
- IncidentForm (B)
- Dispatcher (E)
- DispatcherStation (B)

**Possible Subsystem Decompositions**

- Possible Separation Criteria
  - interface (boundary objects) from data (entity objects)
  - Field Officer activities from Dispatcher ones
  - use both the above splits
  - one subsystem per use case

- (High) Cohesion and (Loose) Coupling are one measure of the “quality” of each subsystem decomposition
Design Summary (so far)

- Requirements analysis focuses on the problem domain.
- Design focuses on the solution domain.
- In system design, objects identified during analysis are grouped into subsystems.
- The degree of cohesion within and coupling between subsystems can be used to guide subsystem decomposition.