Architecture Design

Instructor: Mehroze Khan

Achieving Quality Attributes

- Reflect the characteristics that users want to see in the products we build.
- Architectural styles provide general beneficial properties. Quality attributes also need to be supported:
  - Modifiability
  - Performance
  - Security
  - Reliability
  - Robustness
  - Usability
  - Business goals

Modifiability

- Design must be easy to change
- More than half of the full life-cycle cost of a system—including development, problem fixing, enhancement, and evolution—is spent after the first version of the software is developed and released, modifiability is essential.
- Two classifications of affected software units:
  - Directly affected
  - Indirectly affected

Modifiability

- Directly affected units' **responsibilities change** to accommodate a system modification
  - Anticipate expected changes
  - Value cohesion
  - Maintain generality
- Indirectly affected units' responsibilities do not change, but **implementations must be revised**
  - Lower coupling
  - Interact through interfaces
  - Employ multiple interfaces

Performance

- Performance attributes describe constraints on system speed and capacity:
  - **Response time:** How fast does our software respond to requests?
  - **Throughput:** How many requests can it process per minute?
  - **Load:** How many users can it support before response time and throughput start to suffer?

Performance

- Tactics for improving performance include:
  - Improve utilization of resources
    - Concurrency
    - Replicate and distribute shared data
  - Manage resource allocation more effectively
    - First-come/first-served: Requests are processed in the order in which they are received
    - Explicit priority: Requests are processed in order of their assigned priorities
    - Earliest deadline first: Requests are processed in order of their impending deadlines
    - Pre-emptive Scheduling: Lower-priority request is preempted in favor of a higher-priority request.
    - Round Robin: Allocates resources to requests for fixed time interval.
  - Reduce demand for resources

Security

- Two key architectural characteristics particularly relevant to security: immunity and resilience
- **Immunity**: ability to thwart an attempted attack
  - The architecture encourages immunity by:
    - Ensuring all security features are included in the design
    - Minimizing exploitable security weaknesses
- **Resilience**: ability to recover quickly and easily from an attack
  - The architecture encourages resilience by:
    - Segmenting functionality to contain attack
    - Enabling the system to quickly restore functionality

Reliability

- A software system is reliable if it correctly performs its required functions under assumed conditions
  - Is the software internally free of errors?
- A **fault** is the result of human error, compared to a **failure**, which is an observable departure from required behaviour
  - Software is made more reliable by preventing or tolerating faults

Reliability

- **Passive fault detection**: wait until fault occurs during execution
- **Active fault detection**: periodically check for symptoms or try to anticipate when failures will occur
- **Exceptions**: situations that cause the system to deviate from its desired behaviour
- Include **exception handling** in design to handle exception and return system to acceptable state
- Typical exceptions include:
  - Failing to provide a service
  - Providing the wrong service
  - Corrupting data
  - Violating a system invariant (e.g.; security property)
  - Deadlocking

Reliability

- **Fault recovery**: handling fault immediately to limit damage
- Fault recovery tactics:
  - **Undoing transactions:** manage a series of actions as a single transaction that are easily undone if a fault occurs midway through the transaction
  - **Checkpoint/rollback:** software records a checkpoint of current state; rolls back to that point if system gets in trouble
  - **Backup:** system automatically substitutes faulty unit with backup unit
  - **Degraded service:** returns to previous state, offers degraded version of the service
  - **Correct and continue:** detects the problem (data consistency, stalled process) and treats the symptoms
  - **Report:** system returns to its previous state and reports the problem to an exception-handling unit

Robustness

- A system is **robust** if it includes mechanisms for accommodating or recovering from problems in the environment or in another unit
- **Mutual suspicion:** each software unit assumes that the other units contain faults
- Robustness tactics differ from reliability tactics because the source of problems is different
- Recovery tactics are similar:
  - Rollback to checkpoint state
  - Abort a transaction
  - Initiate a backup unit
  - Provide reduced service
  - Correct symptoms and continue processing
  - Trigger an exception

Usability

- Usability reflects the ease with which a user is able to operate the system
  - User interface should reside in its own software unit
  - Some user-initiated commands require architectural support
  - There are some system-initiated activities for which the system should maintain a model of its environment

Business Goals

- Business Goals are quality attributes the system is expected to exhibit (e.g., minimizing the cost of development and time to market)
  - **Buy vs. Build**
    - Save development time, money
    - More reliable
    - Existing components create constraints; vulnerable to supplier
  - **Initial development vs. maintenance costs**
    - Save money by making system modifiable
    - Increased complexity may delay release; lose market to competitors
  - **New vs. known technologies**
    - Acquiring expertise costs money, delays product release
    - Either learn how to use the new technology or hire new personnel
    - Eventually, we must develop the expertise ourselves

Architecture Evaluation and Refinement

- Design is **iterative**: we propose design decisions, assess, make adjustments, and propose more decisions
- Many techniques to evaluate the design:
  - Trade-off analysis
  - Cost-benefit analysis

Trade-off Analysis

- Often, there are several alternative designs to consider:
  - As professionals, it is our duty to explore design **alternatives** and not simply implement the first design that comes to mind.
  - It may not be immediately obvious which **architectural styles** to use as the basis for a design if the design is expected to achieve quality attributes that conflict with one another.
  - Different members of our design team may promote **competing designs**, and it is our responsibility to decide which one to pursue.
  - Need a **measurement-based method** for comparing design alternatives, so that we can make informed decisions and can justify our decisions to others.

Trade-off Analysis

One Specification, Many Designs

- To see how different architecture styles can be used to solve the same problem.
- The **[key word in context] KWIC** system:
  - Accepts an ordered set of lines.
  - Each line is an ordered set of words.
  - Each word is an ordered set of characters.
  - Any line may be “circularly shifted” by repeatedly removing the first word and appending it at the end of the line.
  - The KWIC index system outputs a list of all circular shifts of all lines in alphabetical order.
- Such systems are used to **index text**, supporting rapid **searching for keywords**. For example, KWIC is used in electronic **library catalogues** (e.g., find all titles that contain the name “Martin Luther King Jr.”) and in **online help systems** (e.g., find all index entries that contain the word “customize”).

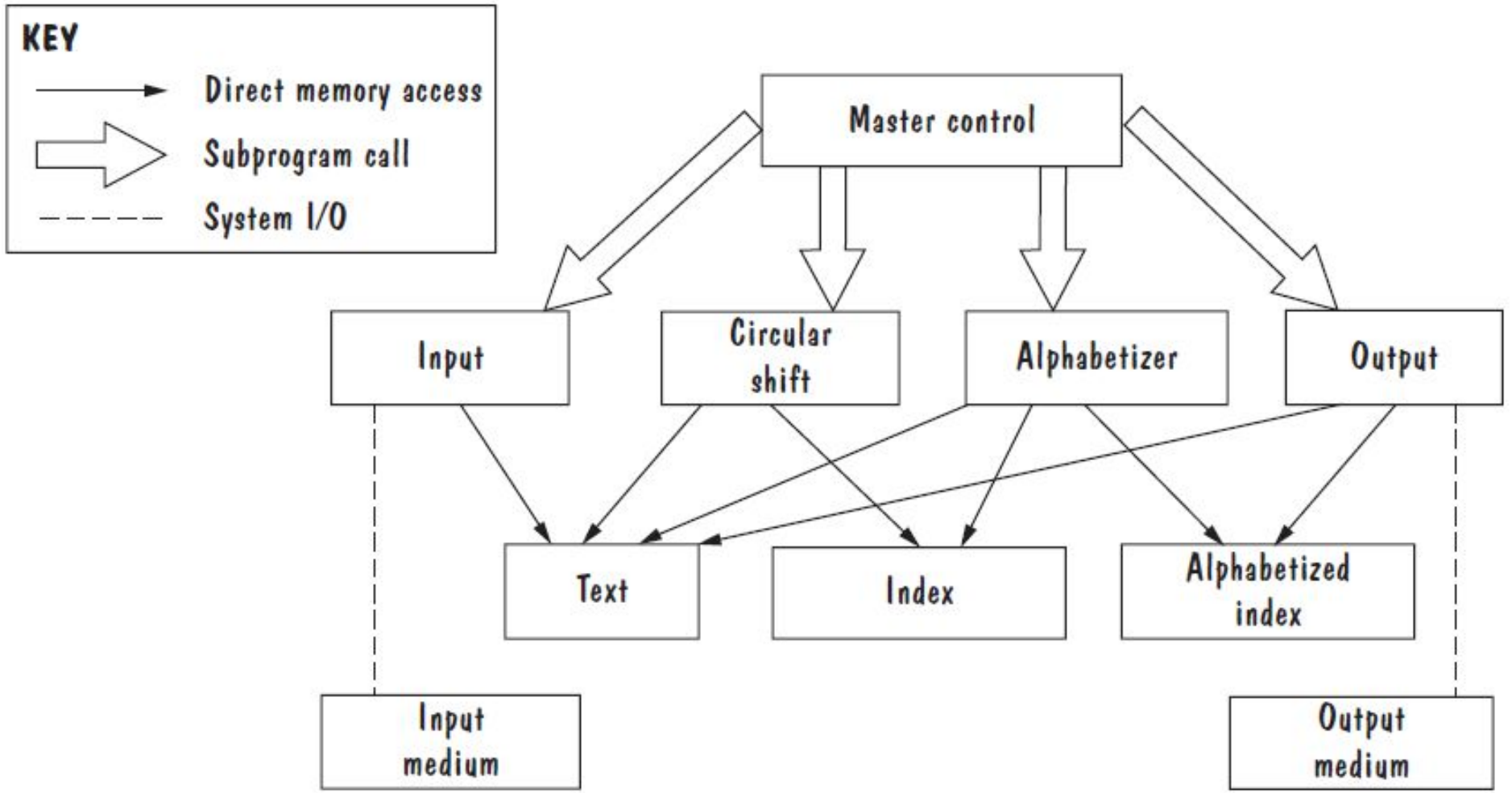
Trade-off Analysis

One Specification, Many Designs

- Shaw and Garlan (1996) present four different architectural designs to implement KWIC:
  - Repository
  - Data abstraction
  - Implicit invocation (a type of publish-subscribe),
  - Pipe-and-filter.

Trade-off Analysis

One Specification, Many Designs



Shared data solution for KWIC

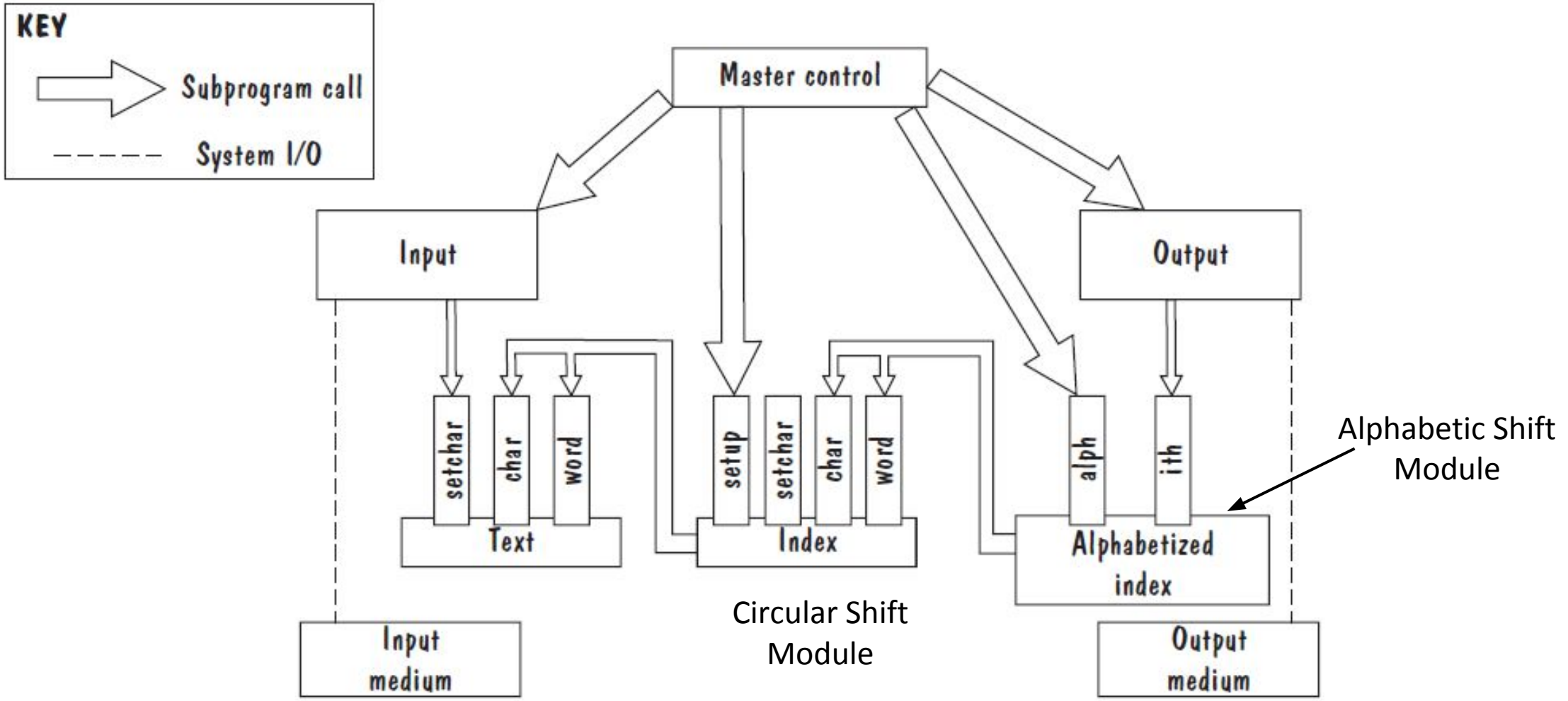
Trade-off Analysis

One Specification, Many Designs

- **Repository Solution.**
- Breaks the problem into its four **primary functions**: input, circular shift, alphabetizer, and output.
- Four modules are coordinated by a **master program** that calls them in sequence.
- Data are localized in their own modules, and not replicated or passed among the computational modules, the design is efficient.
- Design is difficult to change because the computational modules access and manipulate the data directly, via read and write operations, any change to the data and data format will affect all modules.
- None of the elements in the design are particularly reusable.

Trade-off Analysis

One Specification, Many Designs



Data-module solution for KWIC

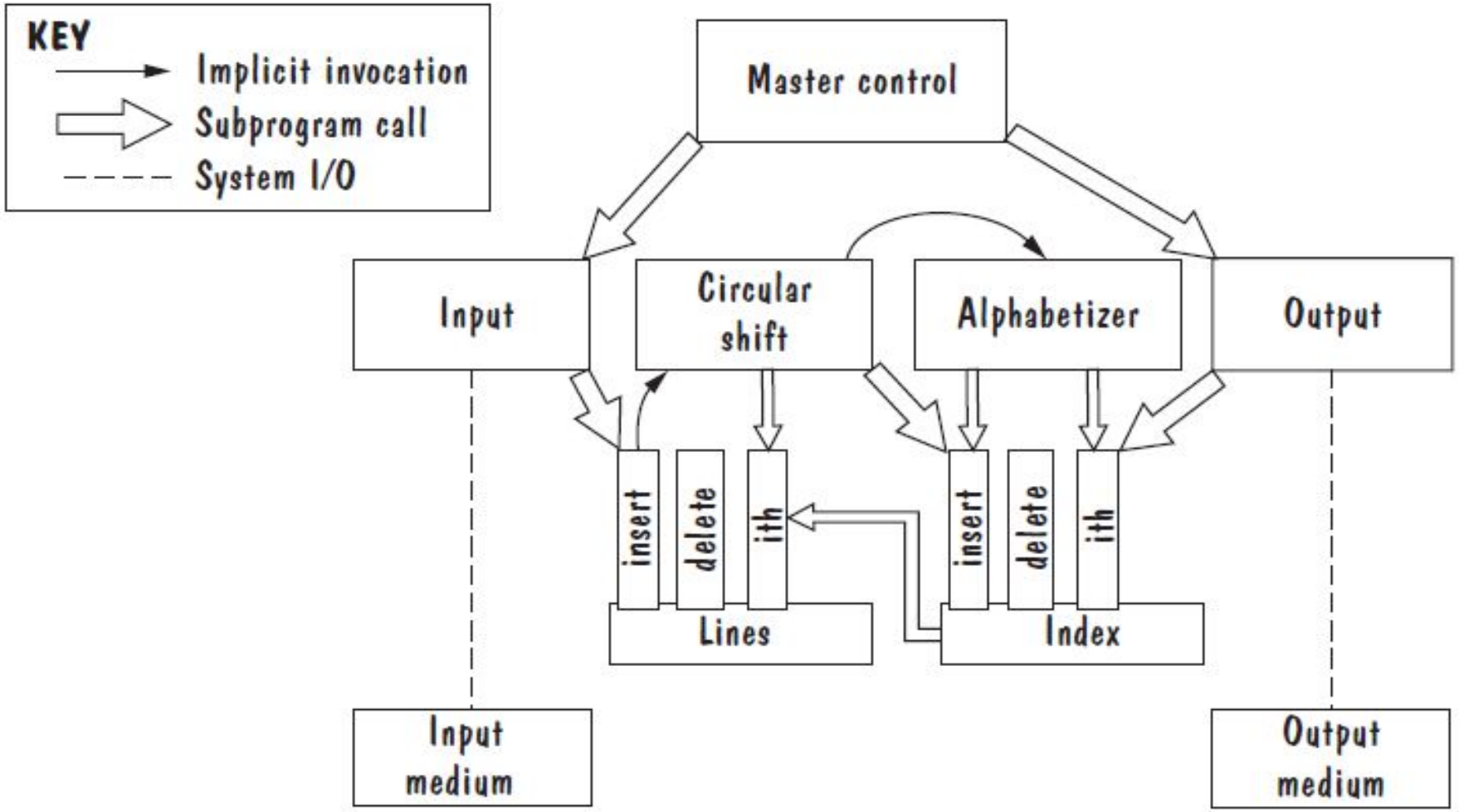
Trade-off Analysis

One Specification, Many Designs

- **Data Abstraction Solution.**
- Second design has a similar decomposition of functionality into sequentially called modules.
- In this design, the data computed by each computational module is stored in that module.
- **Circular-shift** module maintains the index to keywords in the text.
- **Alphabetic-shift** module maintains a sorted (alphabetized) version of this index.
- Each module's data are accessible via access methods, rather than directly. Thus, the modules form ***data abstractions*.**
- Data-abstraction modules encompass both the data to be maintained and the operations for maintaining the data, these modules are easier to **reuse** in other applications than modules from our first design.
- On the downside, changes to the system's functionality may not be so easy, because the functionality is so tightly coupled with the data.

Trade-off Analysis

One Specification, Many Designs



ADT solution for KWIC

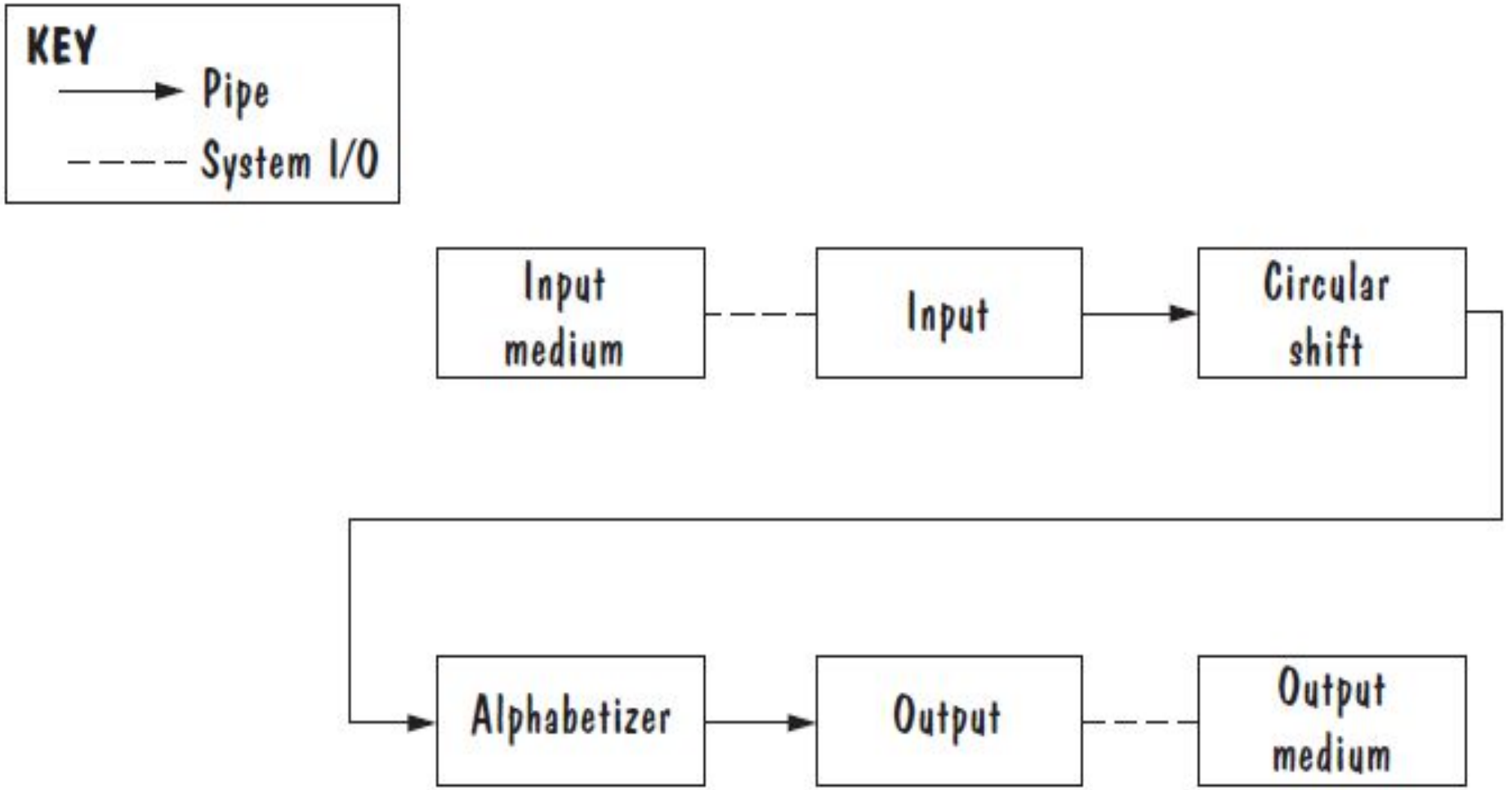
Trade-off Analysis

One Specification, Many Designs

- **Implicit Invocation Solution.**
- Data are stored in **ADTs** that manage generic data types, such as **lines of text** and **sets of indices**, rather than in KWIC-based data abstractions.
- Many of the computational modules are triggered by the occurrence of events rather than by explicit procedure invocation. For example, the circular-shift module is activated whenever a new line of text is input.
- ADTs are generic, they are even more **reusable** than data abstractions.
- Data and operations on data are encapsulated in the ADT modules, so changes to data representation are confined to these modules.
- The design can be easily **extended**, via new computational modules whose methods are triggered by system events; existing modules do not need to be modified to integrate the new modules into the system.
- One complication with an implicit-invocation design is that multiple computational methods may be triggered by the same event. If that happens, all the triggered methods will execute, but in what order?

Trade-off Analysis

One Specification, Many Designs



Pipe-and-Filter solution for KWIC

Trade-off Analysis

One Specification, Many Designs

- **Pipe-and-Filter Solution.**
- Sequence of processing is controlled by the sequence of the filter modules.
- This design is easily extended to include new features, in that we can simply insert additional filters into the sequence.
- The filters may execute in parallel, processing inputs as they are received; this concurrent processing can enhance performance.
- There are some space inefficiencies: circular shifts can no longer be represented as indices into the original text and instead are permuted copies of the original lines of text.
- Data item is copied each time a filter outputs its results to the pipe leading to the next filter.

Trade-off Analysis

One Specification, Many Designs

- Each design has its positive and negative aspects.
- Need a method for comparing different designs that allows us to choose the best one for our purpose.

Trade-off Analysis

One Specification, Many Designs (Comparison Tables)

TABLE 5.2 Comparison of Proposed KWIC Solutions

Attribute	Shared Data	Data Abstraction	Implicit Invocation	Pipe and Filter
Easy to change algorithm	–	–	+	+
Easy to change data representation	–	+	–	–
Easy to add functionality	+	–	+	+
Good performance	–	–	+	+
Efficient data representation	+	+	+	–
Easy to reuse	–	+	–	+

- A plus means that the design has the attribute
- A minus means that the design does not have the attribute

Trade-off Analysis

One Specification, Many Designs (Comparison Tables)

TABLE 5.3 Weighted Comparison of Proposed KWIC Solutions

Attribute	Priority	Shared Data	Data Abstraction	Implicit Invocation	Pipe and Filter
Easy to change algorithm	1	1	2	4	5
Easy to change data representation	4	1	5	4	1
Easy to add functionality	3	4	1	3	5
Good performance	3	4	3	3	5
Efficient data representation	3	5	5	5	1
Easy to reuse	5	1	4	5	4
Total		49	69	78	62

Highest Priority of an attribute = 5, Lowest priority = 1. For example, Reusability could be the most desirable attribute

Numbers in column 3 to 6 represent the extent to which a design satisfies the characteristic of the corresponding attribute. 1 being the lowest, 5 being the highest

Trade-off Analysis

One Specification, Many Designs (Comparison Tables)

TABLE 5.3 Weighted Comparison of Proposed KWIC Solutions

Attribute	Priority	Shared Data	Data Abstraction	Implicit Invocation	Pipe and Filter
Easy to change algorithm	1	1	2	4	5
Easy to change data representation	4	1	5	4	1
Easy to add functionality	3	4	1	3	5
Good performance	3	4	3	3	5
Efficient data representation	3	5	5	5	1
Easy to reuse	5	1	4	5	4
Total		49	69	78	62

Multiply the priorities with ratings and sum over the design to get score for each design
For example, score for pipe and filter = 1x5 + 4x1 + 3x5 + 3x5 + 3x1 + 5x4 = 62.
Pick the design with the highest score. In this case, Implicit-Invocation design is chosen

Trade-off Analysis

One Specification, Many Designs

- Other attributes to consider
  - Modularity
  - Testability
  - Security
  - Ease of use
  - Ease of understanding
  - Ease of integration

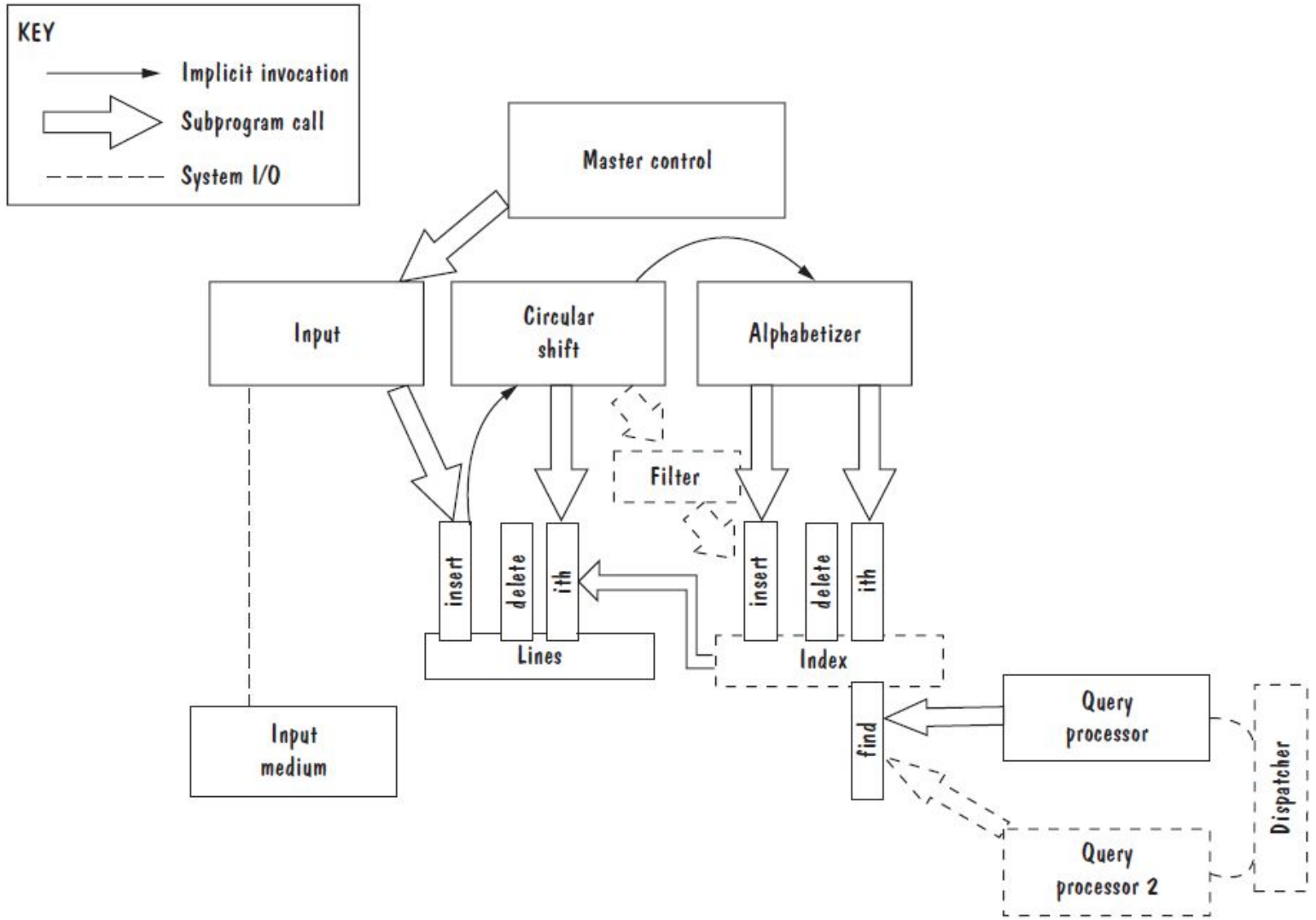
Cost-Benefit Analysis

- Will the benefits of a system based on a particular design outweigh the costs of its implementation?
- If there are competing designs, which one will give us the highest return on investment?

Cost-Benefit Analysis

- Consider a proposal to improve KWIC performance because the number of KWIC indices have increased
  - Eliminate noise word indices?
  - Change representation of indices to bin of indices?
  - Increase server capacity by adding another computer?
- A cost–benefit analysis is a widely used business tool for estimating and comparing the costs and benefits of a proposed change

Cost-Benefit Analysis
(Proposed Changes to KWIC)



Cost-Benefit Analysis
(Computing Benefits)

- A cost-benefit analysis contrasts financial benefits with financial costs, both in Dollars
  - Costs
    - Development
    - Operational
  - Benefits
    - Reduced Operational Costs
    - Increased Earnings

Cost-Benefit Analysis
(Computing Benefits)

- Suppose that the current catalogue contains 70,000 entries, and that the average size of an entry (i.e., the number of words per record, including the video's title, the director's name, the actors' names, and so on) is 70 words, for a total of almost five million circular shifts.
- On average, it currently takes the system 0.016 seconds to find and output all entries that contain two keywords, which means that the system accommodates about 60 such requests per second.
- However, at peak times, the system will receive up to 100 queries per second, and the system is eventually expected to handle 200 queries per second.

Cost-Benefit Analysis

TABLE 5.4 Cost–Benefit Analysis of Design Proposals

	Eliminate Noise Words	Store Indices in Bins	Add Second Server
Benefits			
Search time	0.015 sec	0.002 sec	0.008 sec
Throughput	72 requests/sec	500 requests/sec	115 requests/sec
Added value	$24,000/yr	$280,000/yr	$110,000/yr
Costs			
Hardware			$5,000
Software	$50,000	$300,000	$200,000
Business losses	$28,000+/yr		
Total costs first year	$78,000	$300,000	$205,000

Cost-Benefit Analysis

- For simplicity, suppose that every additional request per second that the system can process, up to 200 requests/second, would save the company $2000 per year, based on retained customers and reduced calls to technical support.
- Given this value function, eliminating noise words would save the company $24,000 per year, calculated as:

  (72 requests/second - 60 requests/second) * $2000/year = $24,000/year

- Adding a second server would save the company $110,000 per year, calculated as:

  (115 requests/second - 60 requests/second) * $2000/year = $110,000/year

- The second design option would improve the system's throughput beyond what will be needed (the system will receive at most 200 requests per second).Therefore, the value added by changing to bin-based indexing is the maximum possible value:

  (200 requests/second - 60 requests/second) * $2000/year = $280,000/year

Cost-Benefit Analysis

- Payback period
  - length of time before accumulative benefits recover the costs of implementation
- In this example, the payback period for restructuring the sorted-index module (design 2) is:

  $300,000/$280,000 = 1.07 of a year = approximately 13 months

Cost-Benefit Analysis

- Return on Investment (ROI)
- An ROI of 1 or greater means that the design's benefits outweigh its costs. The higher the ROI value, the more effective the design.
  - ROI = Benefits/Cost, ROI > 1 is desired
  - %age gain ROI = (Benefits – Cost) / Cost x 100
  - **Example**
    - After five years
      - Cost = $ 2249559
      - Benefits = $ 6122893
      - ROI = 2.72 i.e. Total benefits are approx. 3 times the total costs
      - %age gain ROI = 172.19 % i.e. Your earnings are 172.19 % of the total costs

Trade-off Analysis (Exercise)

One Specification, Many Designs (Comparison Tables)

Attribute	Priority	Design A	Design B	Design C	Design D
Good Performance	3	3	2	5	4
High Security	5	3	5	4	4
Modularity	4	5	1	3	1
Ease of Integration	1	1	3	5	5
Easy to Reuse	2	2	1	4	3
Total		?	?	?	?

- Highest Priority of an attribute = 5, Lowest priority = 1
- Calculate the score of each design.
- Which Design will be chosen?

Trade-off Analysis (Exercise Solution)

One Specification, Many Designs (Comparison Tables)

Attribute	Priority	Design A	Design B	Design C	Design D
Good Performance	3	3	2	5	4
High Security	5	3	5	4	4
Modularity	4	5	1	3	1
Ease of Integration	1	1	3	5	5
Easy to Reuse	2	2	1	4	3
Total		49	40	60	47

- Which Design will be chosen?
- Answer: Design C