Understanding General Software Development

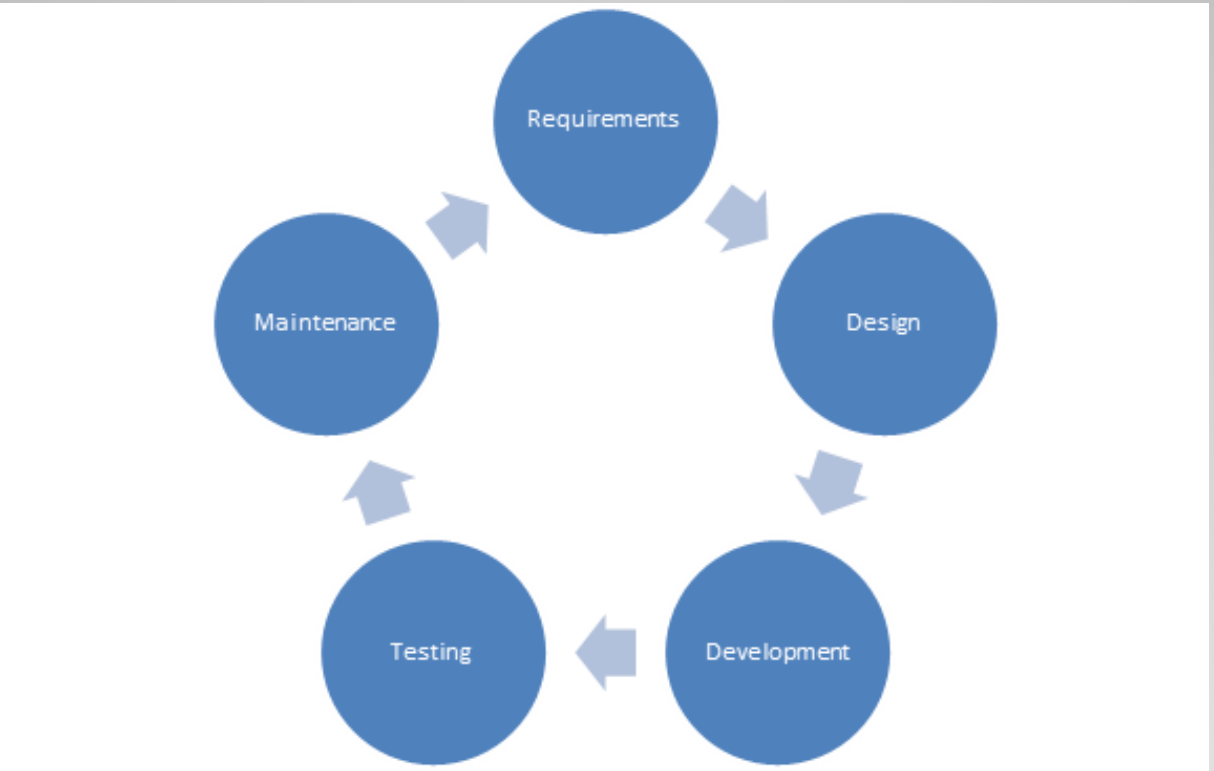
Lesson 3

Objective Domain Matrix

Skills/Concepts	MTA Exam Objectives
Understanding Application Lifecycle Management	Understand application lifecycle management (3.1)
Understanding Testing	Understand application lifecycle management (3.1)
Understanding Data Structures	Understand algorithms and data structures (3.3)
Understanding Sorting Algorithms	Understand algorithms and data structures (3.3)

Application Lifecycle Management (ALM)

- Application lifecycle management (ALM) is the set of activities that revolve around a new software product, from its inception to when the product matures.



Requirements

- Requirements analysis is the process of determining the detailed business requirements for a new software system.
- A business analyst is responsible for analyzing business needs and converting them into requirements that can be executed by the development team.

Design

- The design activity is used to create plans, models, and architecture for how the software will be implemented.
- Participants
  - Architect
  - User-experience Designer

Development

- The software development activity involves implementing design by creating software code, databases, and other related content.
- Participants
  - Developers
  - Database Administrators (DBAs)
  - Technical Writers
  - Content Developers

Testing

- Testing is used to assure the quality of the final product.
- Identifies possible gaps between the system expectations described in the requirements document and actual system behavior.
- Participants
  - Testers

Understanding Testing

- Software testing is the process of verifying software against its requirements.
- Software testing can only help find defects—it cannot guarantee the absence of defects.
- It is much more cost-effective to find defects earlier (rather than later) in the product development cycle.

Testing Methods

- Black-box Testing
  - Focusing solely on inputs and outputs.
  - Any knowledge of internal system workings is not used for testing.
  - Is used to make sure a software application covers all its requirements.
- White-box Testing
  - Testers use their knowledge of system internals when testing the system.
  - Is used to make sure that each method or function has proper test cases available.

Testing Levels

- Unit Testing
  - Verifies the functionality of a unit of code.
- Integration Testing
  - Assesses the interface between software components.
- System Testing
  - Overall testing of the software system.
- Regression Testing
  - Makes sure that each new fix doesn't break anything that was previously working.

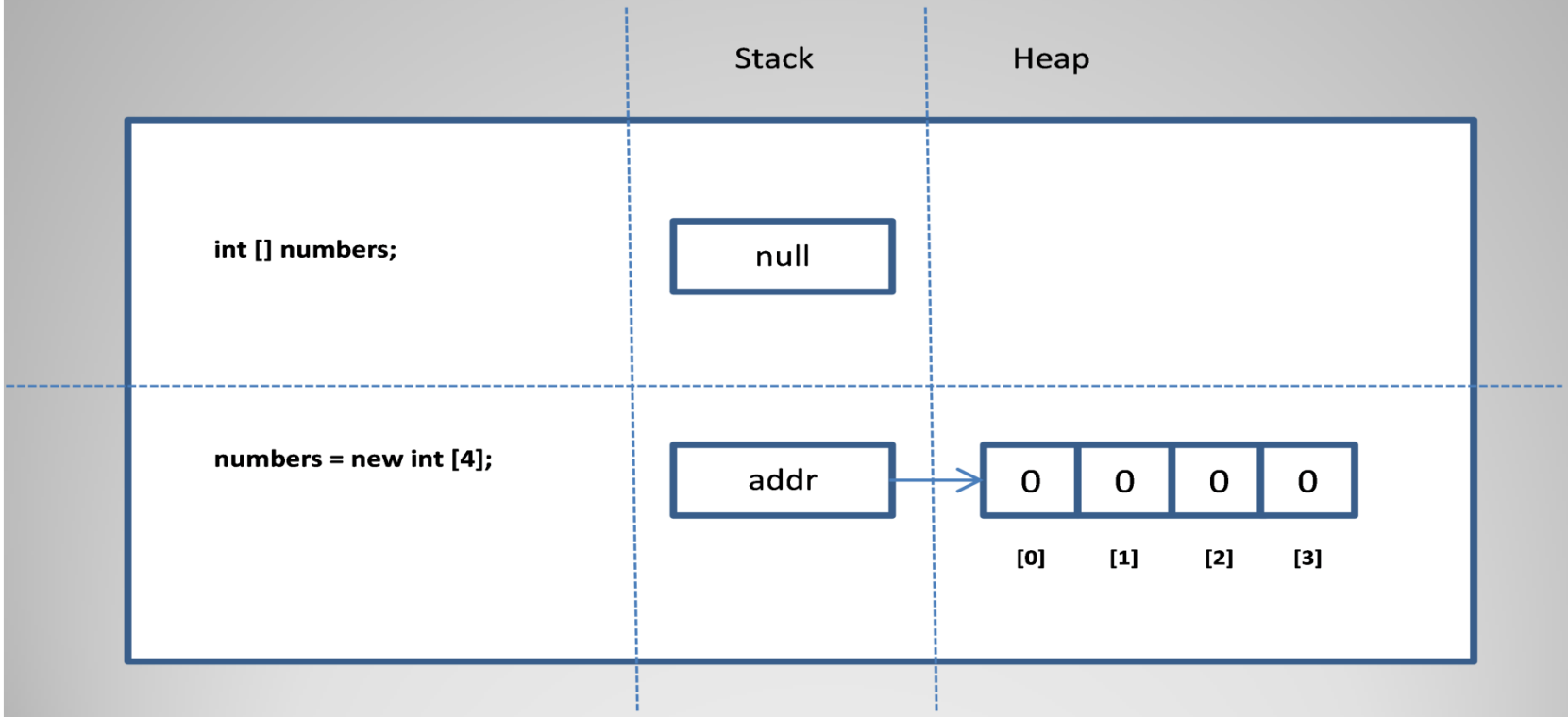
Understanding Data Structures

- Data structures are techniques for organizing and storing data in computer memory.
- Understanding a data structure involves
  - Understanding the storage pattern,
  - Understanding what methods are used to create, access, and manipulate the data structure.
- Common data structures
  - Arrays
  - Queues
  - Stacks
  - Linked Lists

Arrays

- An array is a collection of items of the same type.
- The items in an array are stored in contiguous memory locations.
- Capacity of an array is predefined and fixed.
- Any array item can be directly accessed by using an index.
- C# array indexes are zero-based.

Array - Internal Representation



Array – Common Operations

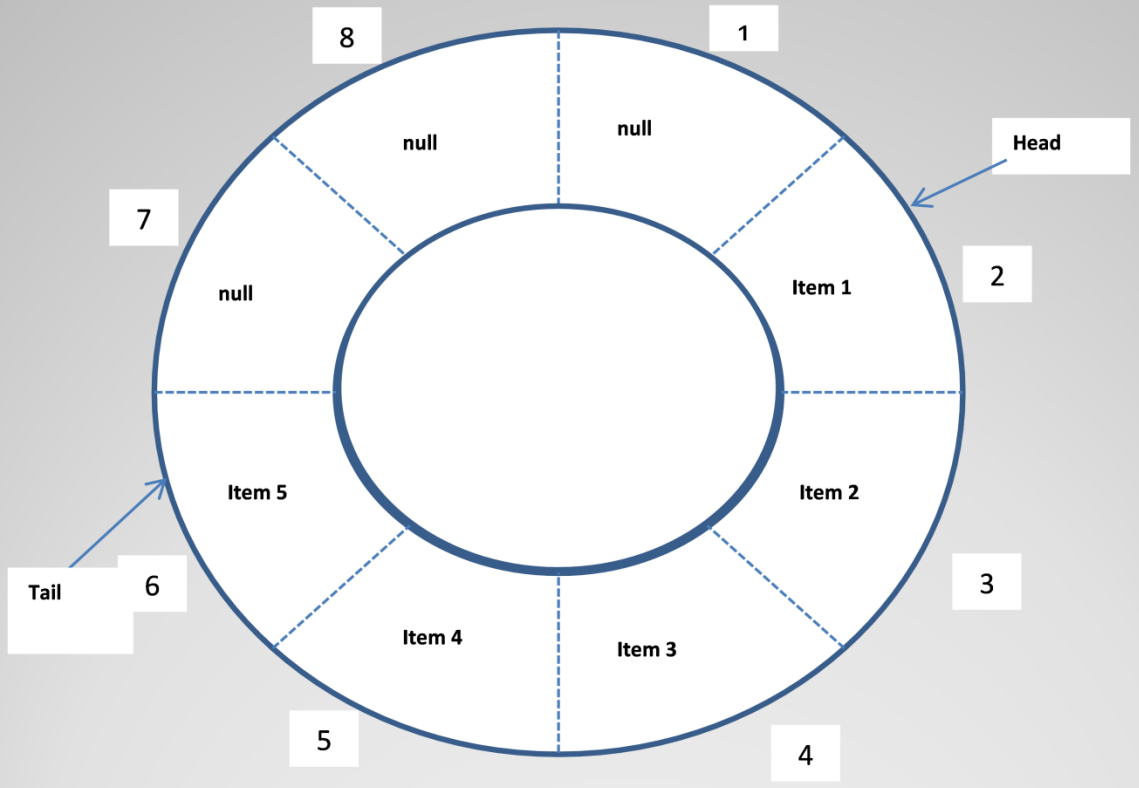
- Arrays support the following operations:
  - Allocation
  - Access
- Following code assigns a value of 10 to the fourth item of the array, and twice that value is then assigned to the variable calc:

```
number[3] = 10;
int calc = number[3] * 2;
```

Queues

- A collection of items in which the first item added to the collection is the first one to be removed.
- First In First Out (FIFO).
- Queue is a heterogeneous data structure.
- Capacity of a queue is the number of items the queue can hold.
- As elements are added to the queue, the capacity can be automatically increased.

Queues – Internal Representation



Queues – Common Operations

- **Enqueue:** Adds an item to the tail end of the queue.
- **Dequeue:** Removes the current element at the head of the queue.
- **Peek:** Access the current item at the head position without actually removing it from the queue.
- **Contains:** Determines whether a particular item exists in the queue.

Stacks

- A collection of items in which last item added to the collection is the first one to be removed.
- Last In First Out (LIFO).
- Stack is a heterogeneous data structure.
- Capacity of a stack is the number of items the queue can hold.
- As elements are added to the stack, the capacity can be automatically increased.

Stacks – Internal Representation

- A stack can be visualized just like the queue, except that the tail is called the top of the stack and the head is called the bottom of the stack.
- New items are always added to the top of a stack; when this happens, the top of the stack starts pointing to the newly added element.
- Items are also removed from the top of the stack, and when that happens, the top of the stack is adjusted to point to the next item in the stack.

Stack – Common Operations

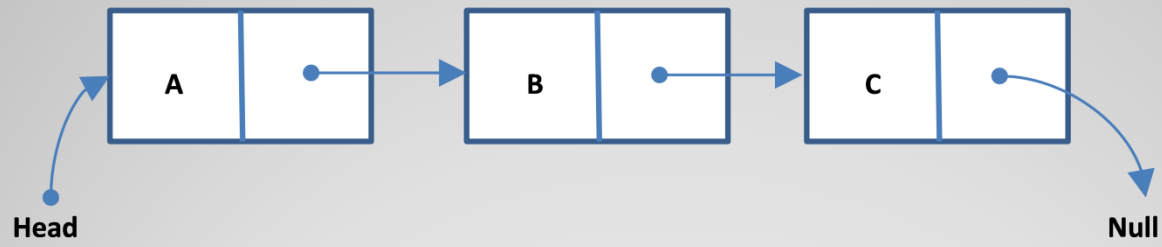
- **Push:** Adds item to the top of the stack.
- **Pop:** Removes the element at the top of the stack.
- **Peek:** Access the current item at the top of the stack without actually removing it from the stack.
- **Contains:** Determines whether a particular item exists in the stack.

Linked Lists

- A linked list is a collection of nodes arranged so that each node contains a link to the next node in the sequence.
- Each node in a linked list contains of two pieces of information:
  - the data corresponding to the node
  - the link to the next node

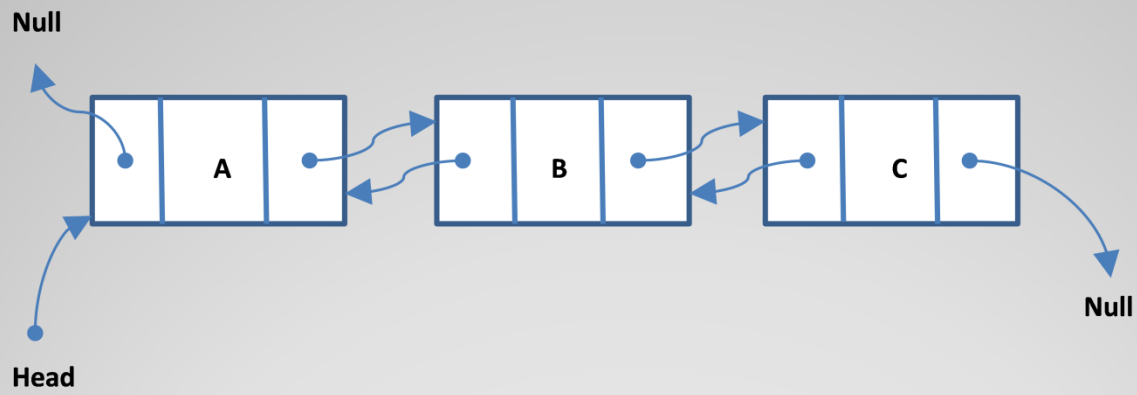
Linked Lists – Internal Representation

- A singly linked list



Linked Lists – Internal Representation

- A doubly linked list

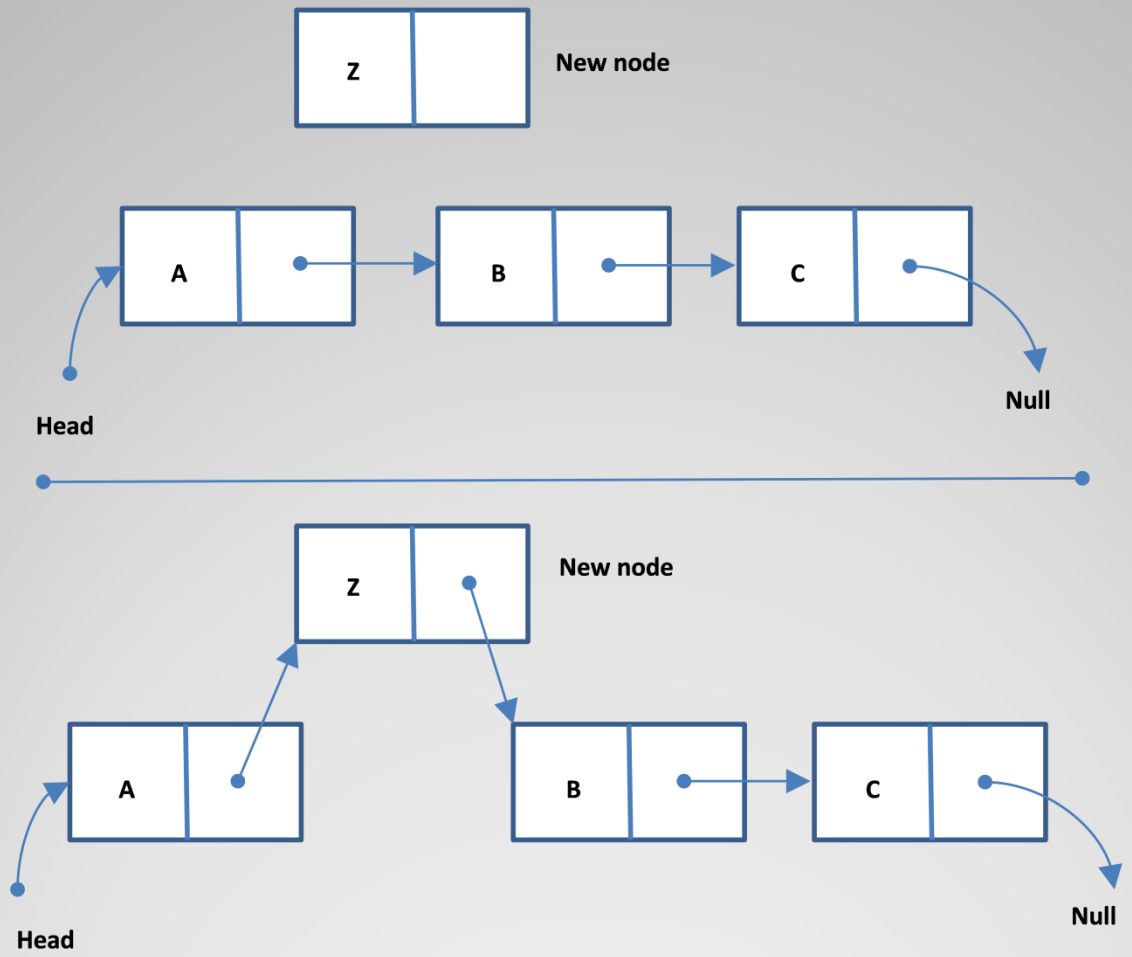


Linked Lists – Common Operations

- **Add:** Adds an item to a linked list.
- **Remove:** Removes a given node from the linked list.
- **Find:** Finds a node with a given value in the linked list.

Linked Lists – Visualizing the add Operation

- Adding an item to a linked list is a matter of changing links.



Understanding Sorting Algorithms

- Sorting algorithms are algorithms that arrange the items in a list in a certain order.
- Understanding sorting algorithms can help you understand, analyze, and compare different methods of problem solving.
- **BubbleSort** and **QuickSort** are two of many sorting algorithms .

BubbleSort

- **BubbleSort** works by comparing two elements to check whether they are out of order; if they are, it swaps them. The algorithm continues to do this until the entire list is in the desired order.
- **BubbleSort** gets its name from the way the algorithm works: As the algorithm progresses, the smaller items are “bubbled” up.

Visualizing BubbleSort – Pass 1

Step	*Before*	*After*	*Comments*
1	**20**, **30**, 10, 40	**20**, **30**, 10, 40	The algorithm compares the first two elements (20 and 30); because they are in the correct order, no swap is needed.
2	20, **30**, **10**, 40	20, **10**, **30**, 40	The algorithm compares the next two elements (30 and 10); because they are out of order, the elements are swapped.
3	20, 10, **30**, **40**	20, 10, **30**, **40**	The algorithm compares the next two elements (30 and 40); because they are in the correct order, no swap is needed.

Visualizing BubbleSort – Pass 2

Step	*Before*	*After*	*Comments*
1	**20**, **10**, 30, 40	**10**, **20**, 30, 40	The algorithm compares the first two elements (20 and 10); because they are out of order, the elements are swapped.
2	10, **20**, **30**, 40	10, **20**, **30**, 40	The algorithm compares the next two elements (20 and 30); because they are in the correct order, no swap is needed.
3	10, 20, **30**, **40**	10, 20, **30**, **40**	The algorithm compares the next two elements (30 and 40); because they are in the correct order, no swap is needed.

Visualizing BubbleSort – Pass 3

Step	*Before*	*After*	*Comments*
1	**10**, **20**, 30, 40	**10**, **20**, 30, 40	The algorithm compares the first two elements (10 and 20); because they are in the correct order, no swap is needed.
2	10, **20**, **30**, 40	10, **20**, **30**, 40	The algorithm compares the next two elements (20 and 30); because they are in the correct order, no swap is needed.
3	10, 20, **30**, **40**	10, 20, **30**, **40**	The algorithm compares the next two elements (30 and 40); because they are in the correct order, no swap is needed.

QuickSort

- The **QuickSort** algorithm uses the divide-and-conquer technique to continually partition a list until the size of the problem is small and doesn't require any sorting.
- **QuickSort** algorithm works much faster than **BubbleSort**.

Visualizing QuickSort

Step	*Data to be sorted*							*Comments*
1	50, 10, **30**, 20, 40							Start with an unsorted list and pick a pivot element—in this case 30.
2	20, **10**			30	50, **40**			Partition the list, with items less than the pivot going to the left list and items greater than the pivot going to the right list. Then, to sort the left list, pick a pivot (here, 10). Similarly, to sort the right list, pick a pivot (here, 40) for that list.
3	-	10	20	30	-	40	50	In the left list, 20 is greater than 10, and in the right list, 50 is greater than 40; therefore, both 20 and 50 go into the right list. This yields lists of only one number, which are all by definition sorted.
4	10, 20, 30, 40, 50							All the small sorted lists are merged to create the final complete sorted list.

Recap

- Application Lifecycle Management
  - Requirement Analysis
  - Software Development
  - Testing
  - Release Management
- Testing Methods
  - Black-box Testing
  - White-box Testing

Recap

- Testing Levels
  - Unit testing, integration testing, system testing, acceptance testing
- Data Structures
  - Arrays
  - Queues
  - Stacks
  - Linked Lists
- Sorting Algorithms: BubbleSort, QuickSort