Computational Thinking

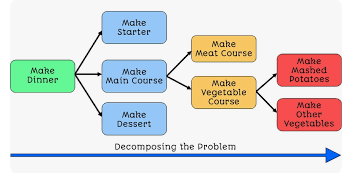
Computational thinking is a problem-solving methodology that involves thinking of a solution to a problem based on how the computer would solve it.

It involved in breaking down complex problems into simpler, more manageable parts, and devising solutions by using algorithmic and logical reasoning.

Technique of Computational Thinking

* Decomposition
* Pattern recognition
* Abstraction
* Algorithm Thinking

Decomposition

Decomposition in computational thinking is the process of taking a complex problem and breaking it down into smaller, more manageable and understandable components or steps. This involves identifying the key components and understanding of a problem and how they relate to each other so that it can be broken down a problem into smaller, more manageable sub-problems.

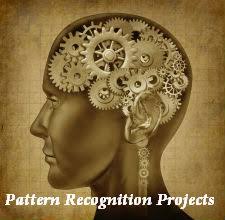
Each sub-problem is then solved independently, and the solutions are combined to solve the overall problem.

* Modularity: Each component addresses a specific aspect of the problem
* Parallel Development: different teams or individuals can work on solving different sub-problems simultaneously, allowing for faster overall and more efficient solution**.**
* Reusability: Solutions to sub-problems can often be reused, promoting code reuse and efficiency.
* Simplification: Decomposing a complex problem simplifies the problem-solving process by breaking it into more understandable and manageable

When decomposing a problem consideration should be taken with the following

* Interconnection of the decomposed parts. understanding how the decomposed parts are interconnected because changes in one part may affect others.
* Excessive decomposition can lead to unnecessary complexity. It's important to find the right balance between the granularity of parts and the manageability of the problem.
* After solving the individual parts, integrating the solutions to form a coherent whole can sometimes pose a challenge.

Pattern Recognition

Pattern recognition is a field of study that involves the identification and interpretation of patterns in data. Identifying patterns or trends within a problem, this involves observing patterns, trends, or regularities in data or behavior.

The goal is to use these observations to make educated guesses or to solve problems.

Pattern recognition involves classifying or grouping information based on similarities, which can simplify complex problems and facilitate decision-making.

**Importance of Pattern Recognition**

* Recognizing patterns enables individuals to predict and anticipate future events or behaviors based on historical data.
* improve the efficiency of problem-solving by allowing the application of the same solution to similar problems.
* can lead to the discovery of new insights by revealing hidden structures in data.
* adaptable to various domains, such as mathematics, science, language learning, and artificial intelligence.

Applications of Pattern Recognition

* Machine Learning: Algorithms use pattern recognition to learn from data and make decisions or predictions.
* Natural Language Processing (NLP): Identifying patterns in language data to understand syntax and semantics.
* Data Mining: Discovering patterns in large datasets to extract useful information and insights.
* Medical Diagnosis: Recognizing patterns in symptoms, test results, or patient histories to diagnose diseases.
* Market Analysis: Identifying patterns in consumer behavior, stock market trends, or sales data to optimize business strategies.

Challenges and Considerations

* In machine learning, overfitting occurs when a model is too closely tailored to the training data and fails to generalize to new data. *Overfitting occurs when a model learns the detail and noise in the training data to an extent that it negatively impacts the model's performance on new data. This means that the model becomes very well-suited to the specific data it was trained on, including any outliers or anomalies, but it may fail to generalize to other datasets or real-world scenarios*
* Pre-existing biases can influence pattern recognition, leading to incorrect conclusions or discrimination.
* Some patterns are complex and may require sophisticated methods to be recognized and understood.
* Patterns that change over time can be particularly challenging to recognize and respond to appropriately.

Abstraction

Abstraction is the process of reducing complexity by focusing on the essential features of an object or system, while ignoring more specific details that are not immediately relevant. It seeks to manage complexity by hiding lower-level details, making it possible to think about problems and systems in a more general and simplified way.

Advantages of abstraction

* Promotes modularity by breaking down a system into manageable components.
* Creates portable solutions. By hiding hardware-specific details behind abstraction layers, software can be developed to run on different platforms without significant modification, enhancing cross-platform compatibility.
* Facilitates communication among team members with different expertise. Allows individuals from different area to collaborate as they can focus on the shared abstraction rather than the details specific to their domains.
* Debugging is easier. It reduces the scope of potential issues. No in-depth understanding of the entire system, making it easier to identify and fix problems.

How Abstraction used in computing

* Abstraction is a key principle in OOP, where objects represent abstract data and methods.
* Functional Decomposition where it focuses on breaking down complex processes into simpler functions or sub-procedures.
* Representing data at a high level (e.g., a 'date' object instead of individual day, month, and year).

Example

Let's consider the example of an online e-commerce platform to illustrate the process of a customer purchasing an item from the e-commerce platform.

Information about the following is needed and stored in different table

* + Customer: The user who browses and purchases items. and would include attributes such as Customer ID, name
  + Shopping Cart: A temporary storage for items the customer intends to purchase that would include attributes such as List of Product IDs and quantities
  + Product: The items available for purchase that would include attributes Product ID, Name,
  + Order: A finalized transaction containing purchased items that would include attributes such as Order ID, List of Product IDs with quantities and total price
  + Payment System: The service that handles payment transactions such as Payment Status (details like the internals of how the payment system processes a transaction are not needed).

1. Processes/Functions:
   * Browse: Customer searches for and views products.
   * Add to Cart: Customer selects products and adds them to the shopping cart.
   * Checkout: Customer reviews items in the cart and proceeds to purchase.
   * Process Payment: Payment system validates and processes the customer's payment.
   * Confirm Order: The system confirms the order and notifies the customer upon successful payment.
2. Interactions:
   * Customer to Product: Customer browses and selects products.
   * Customer to Shopping Cart: Customer adds products to the cart.
   * Customer to Payment System: Customer provides payment during checkout.
   * Payment System to Order: Payment System confirms payment and triggers order creation.
3. Constraints:
   * Stock Availability: Products can only be purchased if they are in stock.
   * Payment Authorization: Orders can only be confirmed if payment is authorized.

Oline e-commerce platform, several complex issues were abstracted away to focus on the core purchasing process.

* The details of the customer's account, such as registration, authentication, password management, and user profile settings, are excluded. These are complex systems in their own right but are not necessary to understand the fundamental process of purchasing a product.
* Real-time inventory tracking, restocking processes, and supplier interactions are not included in the model above. While crucial for the actual operation of an e-commerce platform, they do not directly impact the high-level view of a customer transaction.
* The functionality for customers to leave reviews and rate products adds complexity to a product's representation but is not essential for the transactional flow from selection to purchase.
* Many e-commerce platforms use sophisticated algorithms to recommend products to customers based on their browsing and purchase history. This complexity is removed in the example given, as it does not affect the basic purchasing process.
* The logistics of shipping, including the calculation of shipping costs, packaging, delivery options, and tracking, are significant operational aspects. However, they are not necessary to understand the checkout and payment process; thus, they are abstracted away.
* The infrastructure for handling customer inquiries, returns, and support tickets is a detailed subsystem that is not depicted in the example focused on the purchasing aspect.
* While the example includes a "Payment System," it does not goes into the complexities of payment processing, such as fraud detection, payment gateways, and handling of different payment methods.
* Mechanisms for applying discounts, promotions, and loyalty rewards can be complex and involve several rules and calculations. These are not included in the abstract model to keep the focus on the essential transaction process.
* The complexities of the user interface design, such as page layouts, navigation, and responsive design, are significant for user experience but are not part of the core abstract model.

Stepwise Refinement

Stepwise refinement is a software development technique used to break down a complex problem or system into smaller, more manageable parts. It involves gradually refining and expanding the initial solution by adding more details and functionality at each step.

The main idea behind stepwise refinement is to start with a high-level overview of the problem or system and then iteratively decompose it into smaller subproblems. Each subproblem is further refined until it becomes small enough to be easily understood and implemented.