Seminar: Abstraction

Question One

Discuss the metrics used to assess the features of an object oriented program.

Metrics for assessing OOP features focus on how well a program adheres to OOP principles. Misiko (2020) highlights that while OOP metrics remain in development and require standardisation, several widely used metrics offer valuable insights:

Cohesion:
Measures how closely related and focused the methods within a class are to fulfilling a single responsibility. High cohesion indicates well defined class responsibilities, improving readability and maintainability.

Coupling:
Evaluates the degree of dependency between classes. Lower coupling indicates better modularity, allowing independent updates or changes to classes without widespread impact.

Encapsulation:
Assesses how well data and methods are hidden and accessed through controlled interfaces. Strong encapsulation ensures robustness and prevents unintended interference from external code.

Polymorphism:
Examines the use of polymorphic behaviour to enable flexibility and code reuse. It reflects how effectively common behaviours are shared across diverse object types through method overriding and interfaces.

Inheritance Depth:
Measures the depth of the inheritance tree. While deeper hierarchies promote reuse, they can complicate understanding and debugging if not managed carefully.

Number of Methods and Attributes per Class:
Indicates class complexity by analysing the number of responsibilities. Overloaded classes with too many methods or attributes suggest poor design and a need for decomposition.

Reusability:
Determines how easily classes and components can be reused in different contexts or applications. High reusability reduces development effort for future systems.

Scalability:
Evaluates how well the design can adapt to growth in functionality or data without significant rework. Modular and loosely coupled designs often improve scalability.

Testability:
Measures how easily the code can be tested to ensure correctness and identify bugs. Testable designs typically feature clear responsibilities and low dependencies between components.

Question Two

Develop a Python program which has three abstract methods and one subclass which allows a user to perform banking operations.

```
from abc import ABC, abstractmethod

class BankAccount(ABC):
	"""
	Abstract base class for a bank account.
	"""

	@abstractmethod
	def deposit(self, amount):
		"""
		Abstract method to deposit money into the account.
		"""
		pass

	@abstractmethod
	def withdraw(self, amount):
		"""
		Abstract method to withdraw money from the account.
		"""
		pass

	@abstractmethod
	def check_balance(self):
		"""
		Abstract method to check the account balance.
		"""
		pass


class SavingsAccount(BankAccount):
	"""
	A subclass of BankAccount that implements banking operations.
	"""

	def __init__(self, account_holder, initial_balance=0):
		self.account_holder = account_holder
		self.balance = initial_balance

	def deposit(self, amount):
		if amount <= 0:
			print("Deposit amount must be greater than zero.")
		else:
			self.balance += amount
			print(f"Deposited £{amount:.2f}. New balance is £{self.balance:.2f}.")

	def withdraw(self, amount):
		if amount <= 0:
			print("Withdrawal amount must be greater than zero.")
		elif amount > self.balance:
			print("Insufficient funds.")
		else:
			self.balance -= amount
			print(f"Withdrew £{amount:.2f}. New balance is £{self.balance:.2f}.")

	def check_balance(self):
		print(f"Account balance for {self.account_holder}: £{self.balance:.2f}")


if __name__ == "__main__":
	account = SavingsAccount("John Doe", initial_balance=1000)

	while True:
		print("\nBanking Operations:")
		print("1. Deposit")
		print("2. Withdraw")
		print("3. Check Balance")
		print("4. Exit")

		try:
			choice = int(input("Choose an option (1-4): "))
		except ValueError:
			print("Invalid input. Please enter a number between 1 and 4.")
			continue

		if choice == 1:
			amount = float(input("Enter amount to deposit: "))
			account.deposit(amount)
		elif choice == 2:
			amount = float(input("Enter amount to withdraw: "))
			account.withdraw(amount)
		elif choice == 3:
			account.check_balance()
		elif choice == 4:
			print("Exiting banking application. Goodbye!")
			break
		else:
			print("Invalid choice. Please try again.")
```

Question Three

Read the article by Knox et al. (2018) and answer the following questions:

What is Component-based modelling?

Component-based modelling involves representing processes within an integrated model using "pluggable" model components. These models are often implemented as Environmental Modelling Frameworks (EMFs) and use standards or software systems to build and integrate models around a coherent structure. This approach supports modular development and enables models to adhere to a consistent framework for input and output parameters, enhancing reusability and integration.

Upon what do component-based modelling frameworks depend?

These frameworks depend on a consistent underlying structure that integrates all components. They may use standards to explicitly define input and output parameters or provide a common structure for models to comply with. This structure is essential for supporting multidisciplinary models and abstractions across various domains, such as water resources, energy and transport.

Within the context of the work presented in this paper, what is Pynsim?

Pynsim is a Python-based, object-oriented framework designed for networked resource system simulations. It supports the integration of multiple modules through a centralised network structure.

How does Pynsim achieve its goal when using object-oriented Python programming?

Pynsim uses object-oriented design principles by defining abstract classes that developers can extend to create domain-specific models. It leverages inheritance to build hierarchical relationships between objects. Developers define subclasses for network components, allowing for customisation while adhering to a shared structure. Pynsim's modular approach allows for adding and removing algorithms, supporting flexibility and maintainability.

Reference List

abc — Abstract Base Classes (no date). Available at: https://docs.python.org/3/library/abc.html (Accessed: December 18, 2024).

Chen, X. *et al.* (2006) *A Model of Component-Based Programming*. Available at: https://www.researchgate.net/publication/220843674_A_Model_of_Component-Based_Programming.

Milosz, M. and Borys, M. (2010) *Metrics of Object-Oriented Software*. Available at: https://www.researchgate.net/profile/Marek-Milosz/publication/235759663_Metrics_of_Object_Oriented_Software/links/0fcfd513b00ac7409f000000/Metrics-of-Object-Oriented-Software.pdf.

Misiko, N. (2020) “A Review of Metrics for Object-Oriented Design,” *Global Journal of Computer Science and Technology* [Preprint]. Available at: https://computerresearch.org/index.php/computer/article/view/1986/1-A-Review-of-Metrics-for-Object_JATS_NLM_xml.

Pasupathy, S. and Bhavani, R. (2013) “Object Oriented Metrics Evaluation,” *International Journal of Computer Applications*, 78(1), pp. 30–37. Available at: https://research.ijcaonline.org/volume78/number1/pxc3891137.pdf.

Stavrinoudis, D. and Xenos, M. (2000) *OBJECT-ORIENTED METRICS – A SURVEY*. Available at: https://www.researchgate.net/publication/2524641_Object-Oriented_Metrics_A_Survey.