Seminar: Design Patterns in a Python Program

1.1 Define the test cases for the following system development: Oracle. (n.d.) Cruise Ship Management and Cruise Software.

The Shipboard Property Management System (SPMS) simplifies guest and crew handling procedures for cruise operators. It stores guest information centrally, including loyalty program numbers, emergency contacts, and cabin numbers. The onboard identifier card is used for gangway security, mobile mustering, and point of sale. SPMS efficiently manages financial information and tracks critical crew information with customised cards. The system also facilitates visitor handling and ensures onboard safety by creating muster lists and tracking safety duties and attendance. As the ship’s central data hub, SPMS offers many modules to help manage all aspects of the cruise operation (Oracle, N.D.).

I have identified 4 test cases that can be used in the shipboard property management system:

1.1.1 **Test Case 1 – The system has Administrative and Check-in capabilities:** to shorten wait times for visitors, the system enables crew members to perform guest check-ins away from the desk. Crew members can access visitor information, profile updates, guardianship assignments, payment information, and terms & conditions. Operators can establish templates, create contracts, and specify the mandatory data for the profile screen using the Mobile Administration module. The technology can also speed check-in by compiling all relevant data into a single, centralised profile.

1.1.2 **Test Case 2 – The system has Security Capabilities:** with real-time counts for passengers, crew, and guests, the security system offers total security and management. Counts and movement logs are centrally accessible. The ability to specify shore leave rule sets is also provided. Photos and other information can be included for quick identification. The module can be used with magnetic cards, barcodes, and RFID technology. There is also a mobile choice.

1.1.3 **Test Case 3 – The system has Event Management capabilities:** this section is called the "Event Management" feature that helps organise shipboard events, meals, and equipment. Efficiently handles booking, posting, and printing tickets for recreational activities, including shore tours, spas, and cinemas. It also allows for easy cancellation and cost tracking, supporting multiple price levels and foreign languages. The system can import pre-paid and pre-booked excursions and offers a complete solution for managing onboard spas, including scheduling, appointment calendars, and retail item sales.

1.1.4 **Test Case 4 – The system has Staff Management capabilities:** this feature tracks crew working hours and integrates with Security. The time and Attendance function supports various login methods. The Maintenance module tracks requests, printed work orders, and links to guest reservations. The Housekeeping module generates cleaning tasks automatically, with a customisable colour status and linen change forecast. Staff can view work orders and schedule cleaning services, with direct messaging available.

1.2 Which design patterns do you consider compatible with others, and why?

The architectural pattern records the various systems and software's design structures so they can be reused. Developers face the same issues repeatedly throughout their careers, in projects, and while producing software code. Design patterns, which offer engineers a reusable technique to handle these issues and enable software engineers to accomplish the same structural outcome for a particular project, are one way to address this (Walker, 2022).

A few examples of possible design pattern pairings are provided below:

1.2.1. **Composite and Decorator Patterns:** The composite pattern allows treat objects as a group and defines a standard interface. The decorator pattern adds new functionality to an object without changing its structure. They both use composition and single responsibility but serve different purposes. Sometimes they are used together for flexibility but should be used carefully according to design goals and requirements. (Linkedin, N.D.).

1.2.2. **Strategy and Factory Pattern:** Factory and Strategy patterns are helpful application design patterns. The Factory pattern is often used in frameworks for creating different subtypes of objects, while the Strategy pattern is ideal for code with multiple branching statements (Verinext, 2021).

1.2.3 **Factory and Builder Pattern:** There are two types of design patterns: Builder and Factory Method. Builder is used to assemble complex items, while Factory Method constructs objects without knowing their exact type. These patterns can be used together for greater flexibility and easy addition of new features (Kralj, 2022).

1.3 Read Zhang & Budgen (2012). Which design patterns are used most commonly, and why?

According to Zhang & Budgen (2012), design patterns in software engineering are often advocated without concrete evidence to support their effectiveness. Much of the literature on patterns is focused on documenting them rather than analysing experiences with using them. To determine patterns' effectiveness as a knowledge transfer mechanism, they conducted a mapping study using Systematic Literature Review. This approach helped to systematically gather and analyse empirical data on a given topic, reducing bias in individual studies. However, one of the papers they examined has identified the following advantages as being claimed for design patterns:

1.3.1 Using patterns improves programmer productivity and program quality.
1.3.2 Novices can increase their design skills significantly by studying and applying patterns.
1.3.3 Patterns encourage best practices, even for experienced designers.
1.3.4 Design patterns improve communication, both among developers and from developers to maintainers.

The team identified essential topics that needed further investigation during the mapping study. However, they acknowledged that each pattern should be evaluated separately. Therefore, the team concentrated on examining which designs had been researched, the methods used, and the level of agreement or disagreement. Additionally, they conducted a quality assessment to evaluate the reliability of each study.

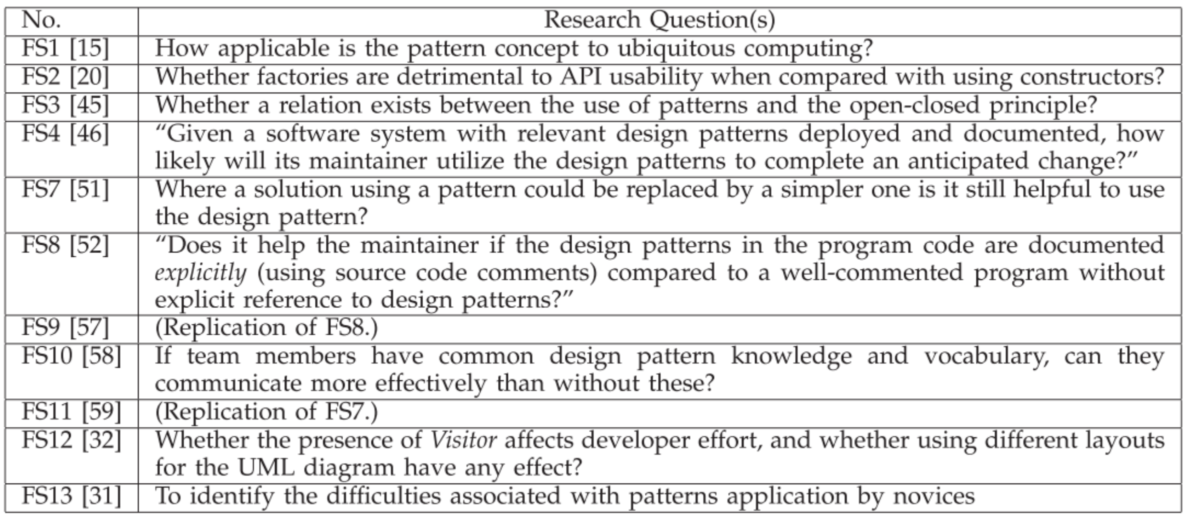


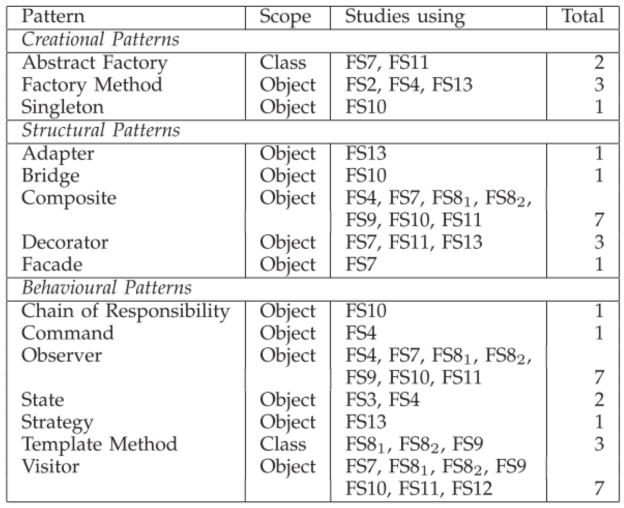
TABLE 4
Research Questions Addressed

No.	Research Question(s)
FS1 [15]	How applicable is the pattern concept to ubiquitous computing?
FS2 [20]	Whether factories are detrimental to API usability when compared with using constructors?
FS3 [45]	Whether a relation exists between the use of patterns and the open-closed principle?
FS4 [46]	"Given a software system with relevant design patterns deployed and documented, how likely will its maintainer utilize the design patterns to complete an anticipated change?"
FS7 [51]	Where a solution using a pattern could be replaced by a simpler one is it still helpful to use the design pattern?
FS8 [52]	"Does it help the maintainer if the design patterns in the program code are documented *explicitly* (using source code comments) compared to a well-commented program without explicit reference to design patterns?"
FS9 [57]	(Replication of FS8.)
FS10 [58]	If team members have common design pattern knowledge and vocabulary, can they communicate more effectively than without these?
FS11 [59]	(Replication of FS7.)
FS12 [32]	Whether the presence of *Visitor* affects developer effort, and whether using different layouts for the UML diagram have any effect?
FS13 [31]	To identify the difficulties associated with patterns application by novices

(Zhang & Budgen, 2012)

Object-Oriented studies mainly used patterns from the GoF catalogue, including five creational, seven structural, and 11 behavioural patterns. Composite, Observer, and Visitor were the most studied patterns, while several others were only used in one study. Only two studies focused on a specific pattern and its properties. The patterns not studied were Builder, Prototype, Flyweight, Proxy, Interpreter, Iterator, Mediator, and Memento.

Patterns Used in the Different Studies

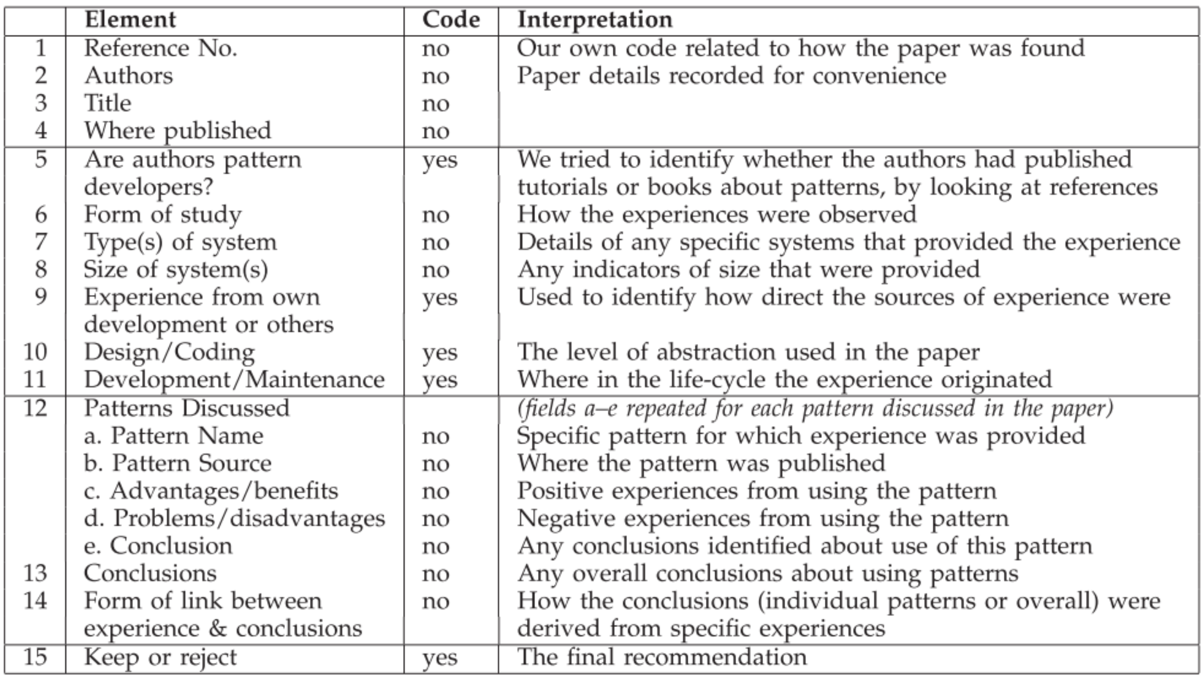


Pattern	Scope	Studies using	Total
Creational Patterns			
Abstract Factory	Class	FS7, FS11	2
Factory Method	Object	FS2, FS4, FS13	3
Singleton	Object	FS10	1
Structural Patterns			
Adapter	Object	FS13	1
Bridge	Object	FS10	1
Composite	Object	FS4, FS7, $FS8_1$, $FS8_2$, FS9, FS10, FS11	7
Decorator	Object	FS7, FS11, FS13	3
Facade	Object	FS7	1
Behavioural Patterns			
Chain of Responsibility	Object	FS10	1
Command	Object	FS4	1
Observer	Object	FS4, FS7, $FS8_1$, $FS8_2$, FS9, FS10, FS11	7
State	Object	FS3, FS4	2
Strategy	Object	FS13	1
Template Method	Class	$FS8_1$, $FS8_2$, FS9	3
Visitor	Object	FS7, $FS8_1$, $FS8_2$, FS9 FS10, FS11, FS12	7

(Zhang & Budgen, 2012)

They analysed the participants and organisation of studies on pattern usage. Most had programming experience, recognising the need for maturity in object-oriented design. Studies involved modification and coding, with only a few exceptions.

Data Extraction Form Used for Experience Papers



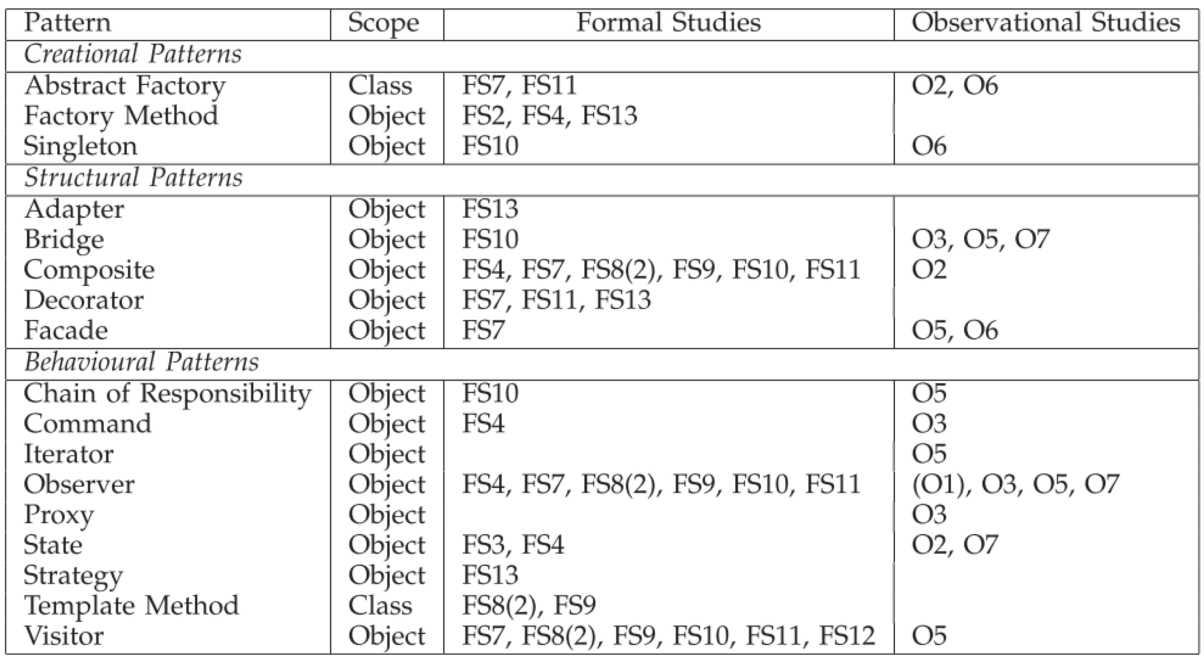
	Element	Code	Interpretation
1	Reference No.	no	Our own code related to how the paper was found
2	Authors	no	Paper details recorded for convenience
3	Title	no	
4	Where published	no	
5	Are authors pattern developers?	yes	We tried to identify whether the authors had published tutorials or books about patterns, by looking at references
6	Form of study	no	How the experiences were observed
7	Type(s) of system	no	Details of any specific systems that provided the experience
8	Size of system(s)	no	Any indicators of size that were provided
9	Experience from own development or others	yes	Used to identify how direct the sources of experience were
10	Design/Coding	yes	The level of abstraction used in the paper
11	Development/Maintenance	yes	Where in the life-cycle the experience originated
12	Patterns Discussed		*(fields a–e repeated for each pattern discussed in the paper)*
	a. Pattern Name	no	Specific pattern for which experience was provided
	b. Pattern Source	no	Where the pattern was published
	c. Advantages/benefits	no	Positive experiences from using the pattern
	d. Problems/disadvantages	no	Negative experiences from using the pattern
	e. Conclusion	no	Any conclusions identified about use of this pattern
13	Conclusions	no	Any overall conclusions about using patterns
14	Form of link between experience & conclusions	no	How the conclusions (individual patterns or overall) were derived from specific experiences
15	Keep or reject	yes	The final recommendation

(Zhang & Budgen, 2012)

To clarify discrepancies in design pattern usage, they reviewed 22 papers with "experience" data and selected seven for further analysis. These papers presented observations as explicit "lessons."

Fifteen out of twenty-three GoF patterns have been empirically evaluated, but more is needed about why specific patterns were chosen for study. Many studies focused on maintenance activities, leaving little knowledge about patterns' role in development. Observational studies suggest caution in misusing patterns, and case studies may be helpful. Using patterns improves communication between developers and maintainers, but there needs to be evidence that they help novices learn about design. More studies are required to understand patterns' effectiveness in productivity and quality.

Patterns Studied in Experiments and Experience Papers



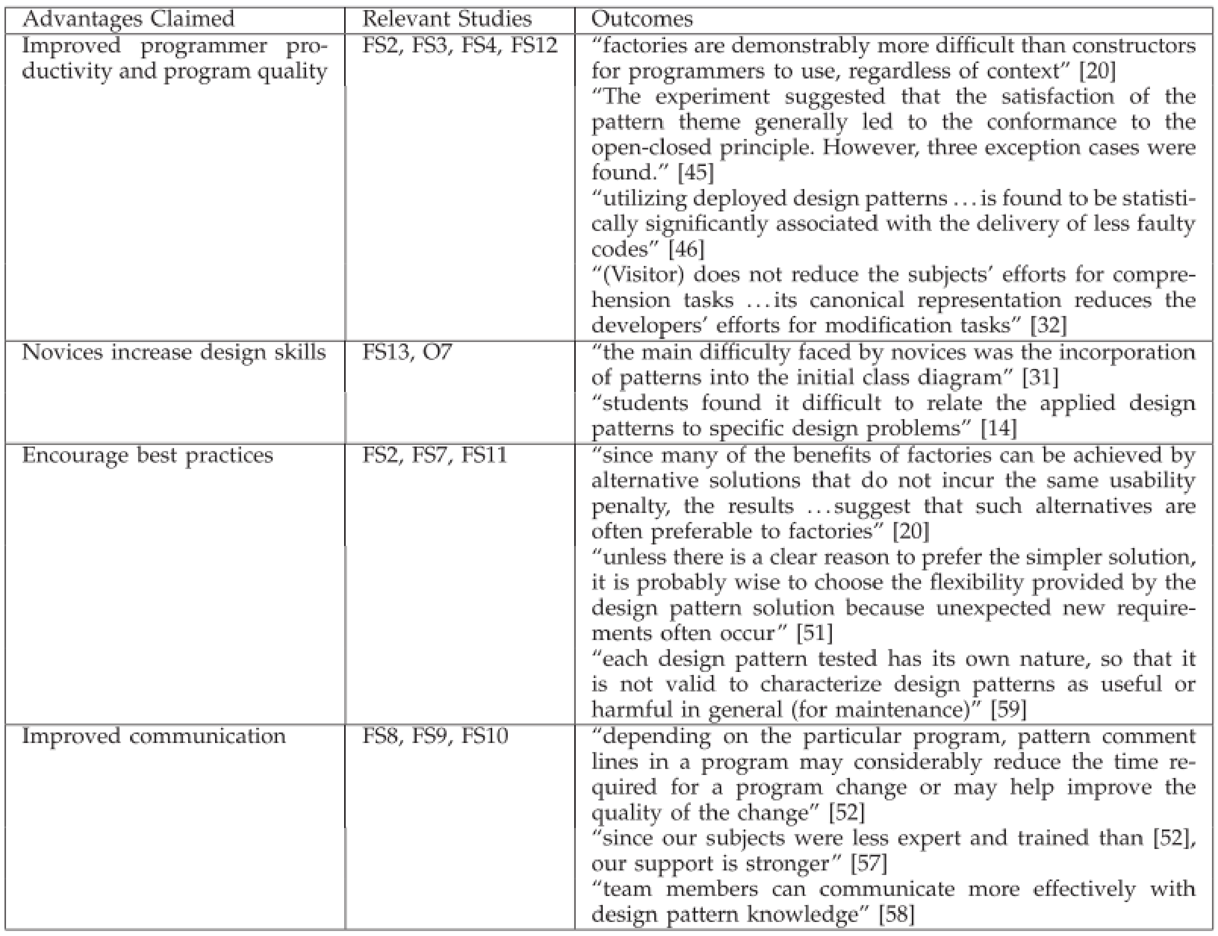
Pattern	Scope	Formal Studies	Observational Studies
Creational Patterns			
Abstract Factory	Class	FS7, FS11	O2, O6
Factory Method	Object	FS2, FS4, FS13	
Singleton	Object	FS10	O6
Structural Patterns			
Adapter	Object	FS13	
Bridge	Object	FS10	O3, O5, O7
Composite	Object	FS4, FS7, FS8(2), FS9, FS10, FS11	O2
Decorator	Object	FS7, FS11, FS13	
Facade	Object	FS7	O5, O6
Behavioural Patterns			
Chain of Responsibility	Object	FS10	O5
Command	Object	FS4	O3
Iterator	Object		O5
Observer	Object	FS4, FS7, FS8(2), FS9, FS10, FS11	(O1), O3, O5, O7
Proxy	Object		O3
State	Object	FS3, FS4	O2, O7
Strategy	Object	FS13	
Template Method	Class	FS8(2), FS9	
Visitor	Object	FS7, FS8(2), FS9, FS10, FS11, FS12	O5

(Zhang & Budgen, 2012)

Using patterns in software development can improve communication when adequately documented. However, there needs to be evidence that it helps novices learn design. The impact of patterns on productivity and quality needs to be clarified and requires further research.

According to the article, different design patterns are employed more frequently depending on the context and domain. Nevertheless, some design patterns are typically seen as being often used and well-liked in the software development industry. These include Singleton Pattern, Factory Pattern, Observer Pattern, Adapter Pattern, and Decorator Pattern because these patterns are frequently employed because they offer tested answers to recurrent design issues. They encourage software systems' modularity, adaptability, reuse, and upkeep. Due to their considerable documentation and study, developers recognise and understand these patterns well.

What the Studies Concluded about the Claimed Advantages



Advantages Claimed	Relevant Studies	Outcomes
Improved programmer productivity and program quality	FS2, FS3, FS4, FS12	"factories are demonstrably more difficult than constructors for programmers to use, regardless of context" [20] "The experiment suggested that the satisfaction of the pattern theme generally led to the conformance to the open-closed principle. However, three exception cases were found." [45] "utilizing deployed design patterns ... is found to be statistically significantly associated with the delivery of less faulty codes" [46] "(Visitor) does not reduce the subjects' efforts for comprehension tasks ... its canonical representation reduces the developers' efforts for modification tasks" [32]
Novices increase design skills	FS13, O7	"the main difficulty faced by novices was the incorporation of patterns into the initial class diagram" [31] "students found it difficult to relate the applied design patterns to specific design problems" [14]
Encourage best practices	FS2, FS7, FS11	"since many of the benefits of factories can be achieved by alternative solutions that do not incur the same usability penalty, the results ... suggest that such alternatives are often preferable to factories" [20] "unless there is a clear reason to prefer the simpler solution, it is probably wise to choose the flexibility provided by the design pattern solution because unexpected new requirements often occur" [51] "each design pattern tested has its own nature, so that it is not valid to characterize design patterns as useful or harmful in general (for maintenance)" [59]
Improved communication	FS8, FS9, FS10	"depending on the particular program, pattern comment lines in a program may considerably reduce the time required for a program change or may help improve the quality of the change" [52] "since our subjects were less expert and trained than [52], our support is stronger" [57] "team members can communicate more effectively with design pattern knowledge" [58]

(Zhang & Budgen, 2012)

References:

Oracle (N.D.). *Oracle Hospitality Cruise Shipboard Property Management System*. Available at: https://www.oracle.com/uk/a/ocom/docs/industries/hospitality/hosp-shipboard-property-sys-ds-4413369.pdf [Accessed 21 Jul. 2023].

Walker, V. (2022). *14 Software architecture design patterns to know*. [online] Enable Architect. Available at: https://www.redhat.com/architect/14-software-architecture-patterns.

LinkedIn (n.d.). *How do you choose between composite and decorator patterns for tree structures in Java?* [online] Available at: https://www.linkedin.com/advice/0/how-do-you-choose-between-composite-decorator.

Verinext (2021). *Strategy & Factory Patterns in Spring Boot Application: Part I*. [online] Verinext. Available at: https://verinext.com/strategy-factory-patterns-in-spring-boot-application-part-i/ [Accessed 21 Jul. 2023].

Kralj, K. (2022). *Builder vs Factory Method Pattern: An In-Depth Comparison*. [online] Available at: https://methodpoet.com/builder-vs-factory-method/#Factory_Method_vs_Builder_patterns_what_are_the_differences [Accessed 21 Jul. 2023].

Zhang, C. & Budgen, D. (2012). What Do We Know about the Effectiveness of Software Design Patterns? *IEEE Transactions on Software Engineering*, 38(5), pp.1213–1231. doi:https://doi.org/10.1109/tse.2011.79.