International Journal Of Humanities Education And Social Sciences ***(IJHESS)*** *E-ISSN:* **2808-1765**
Volume 4, Number 6, June 2025, *Page. 2544 – 2553*
Email : editorijhess@gmail.com

Tactical Object Monitoring System for Defense Operations Using the Concept of Object-Oriented Programming (OOP) with Python Programming Language

Ilvan Dino Rahmandhala [1)*], Asep Adang Supriyadi [2)], Yosef Prihanto [3)], Ahmad Farid[4)], Muhamad Samingan[5)]
[1,2,3,4)] Sensing Technology Study Programme, Faculty of Defense Science and Technology, Indonesia Defense University, Jakarta, Indonesia
[5)] Defense Economics Study Programme, Faculty of Defense Management, Indonesia Defense University, Jakarta, Indonesia

*Corresponding Author
Email: Ilvan.rahmandhala@tp.idu.ac.id

Abstract
Information and communication technology is currently very important in the defense field, including in the development of tactical object monitoring systems for defense operations. One approach used is to apply the concept of object-oriented programming OOP using the Python programming language. This research uses data processing methods using the Python programming language with the help of Google Colab. In its implementation, this research creates two main classes, namely the Sensor Class and the Tactical Map Class. The Sensor Class functions to represent physical sensors in code form, with attributes of identity, location, and sensor status. The Tactical Map class serves as a representation of a tactical map used to manage and visualize objects, such as sensors, within a certain area. The development of a defense operation tactical object monitoring system by applying OOP concepts using the Python programming language has several benefits, such as easier system setup and maintenance, as well as the development of new features without disrupting existing functions. The application of OOP concepts using the Python programming language can be one of the effective approaches in developing a tactical object monitoring system for defense operations. This approach can improve the efficiency and effectiveness of military operations by utilizing modern technology, as well as provide various benefits in terms of system setup, maintenance, and development.

Keywords: Object-Oriented Programming, Phyton, Tactical Object, Operation Defense.

INTRODUCTION

The development of information and communication technology today has had a significant impact on various fields, including defense. In the modern context, military operations no longer rely solely on physical force or heavy equipment, but also on technological capabilities to manage data, improve efficiency and respond quickly to threats. One important application of technology in military operations is tactical object monitoring systems, which are used to support strategic decision-making. Research shows that artificial intelligence (AI) technology has accelerated innovation in military theory, creating major changes in the form and mode of warfare (Wang et al., 2020) Therefore, research on the application of advanced technologies such as Object-Oriented Programming (OOP) in military tactical monitoring systems is urgent to explore in order to face modern challenges and improve the effectiveness of military operations.

Many studies have identified the important role of technology in military operations, but there is still a gap in the application of OOP-based approaches to improve system modularity and security. (Gao et al., 2019) (Ergenc et al., 2023) (Zhang et al., 2019) discussed defense strategies in three different scenarios, but they have not explored the application of OOP-based technologies to overcome these limitations. (Billing et al., 2021)(Friedl, 2018) (Stevanoski et al., 2016) Emphasized the need for research on military human performance capable of adapting to new technologies, but this study lacks highlighting the role of tactical monitoring systems in the context of OOP. Research by (Zhou et al., 2021) shows that OOP has great potential in software development, but its implementation in military systems is minimal. Therefore, there is an urgent

need to integrate OOP concepts such as encapsulation in military systems to address security and operational efficiency challenges.

Within the theoretical framework, OOP is considered the dominant paradigm for the development of large and complex information systems (Yelamali & Beelagi, 2021)(Elrad et al., 2001)(Korson & McGregor, 1990). Explains that key OOP concepts, such as classes and encapsulation, allow developers to hide implementation details, which in turn increases system modularity (Gomes et al., 2019) (Snyder, 1986) (Lieberherr & Holland, 1989). In addition, Python as one of the popular programming languages has been recognized for its ability to handle big data and support the development of analytical tools for machine learning (Butwall et al., 2019) (Stancin & Jovic, 2019). However, the application of Python in the development of OOP-based military systems is still rarely explored. (Reitenbach, 2022) (Chmielewski et al., 2019) (Reefe et al., 2022). Research also shows Python's potential in making math more interactive, which can be applied in the development of tactical monitoring algorithms.

Some previous research lacks an integrated solution for military tactical monitoring systems. For example, (Lian et al., 2022) discussed the importance of OOP in the modern software industry but did not highlight the defense aspect. The research of (Wang et al., 2020) focused more on AI in warfare without explaining how AI can be combined with OOP. The study by (Snyder, 1986) highlights the benefits of OOP in problem solving but does not present its practical application in military systems. These gaps indicate the need for research that integrates OOP theory, Python, and tactical monitoring technologies to provide concrete solutions that can be directly applied in the defense field. Previous studies have made important contributions to understanding the role of technology in military operations, but have focused less on the specific integration between OOP, Python, and military systems. Studies such as (Billing et al., 2021) emphasize the need for adaptive research development, but lack practical software-based solutions. (Zhou et al., 2021) pointed out the general benefits of OOP, but did not investigate how it can be used to strengthen defense systems. Thus, this research attempts to fill the void by offering a new framework that integrates OOP theory, Python capabilities, and military tactical needs. Optimal defense strategies in object protection scenarios, relevant to the context of tactical object monitoring (H. Wu et al., 2019). The application of virtual sensors and their integration in AI-based monitoring systems is closely related to the concept of Sensor Class (Listewnik et al., 2021). This research aims to develop a tactical object monitoring system in military operations by using the concept of encapsulation in OOP and Python programming language. How the application of Python-based OOP can improve system efficiency and reliability by utilizing features such as modularity, data security, and continuous development capabilities. The results of this research are expected to contribute to developing more adaptive and effective military technologies.

RESEARCH METHODS

With the aim of simulating the monitoring of objects in the field of defense operations using sensors and maps, this research uses data processing methods using the Python programming language with the help of Google Colab. Python is a programming language with simplicity and abstraction, which brings it to the top of the list (Ramon-Cortes et al., 2020) Google Colab is used for training and testing algorithm-focused models (Chang & Zhang, 2022). Python is known for its flexibility, supporting various programming paradigms including object-oriented, imperative, and functional programming. Python is also open-source, so it has a very active community that is constantly developing and updating various packages and libraries, which makes it a powerful tool for a wide range of applications, from simple scripting to complex application development (Carneiro et al., 2018) (J. & V., 2021).

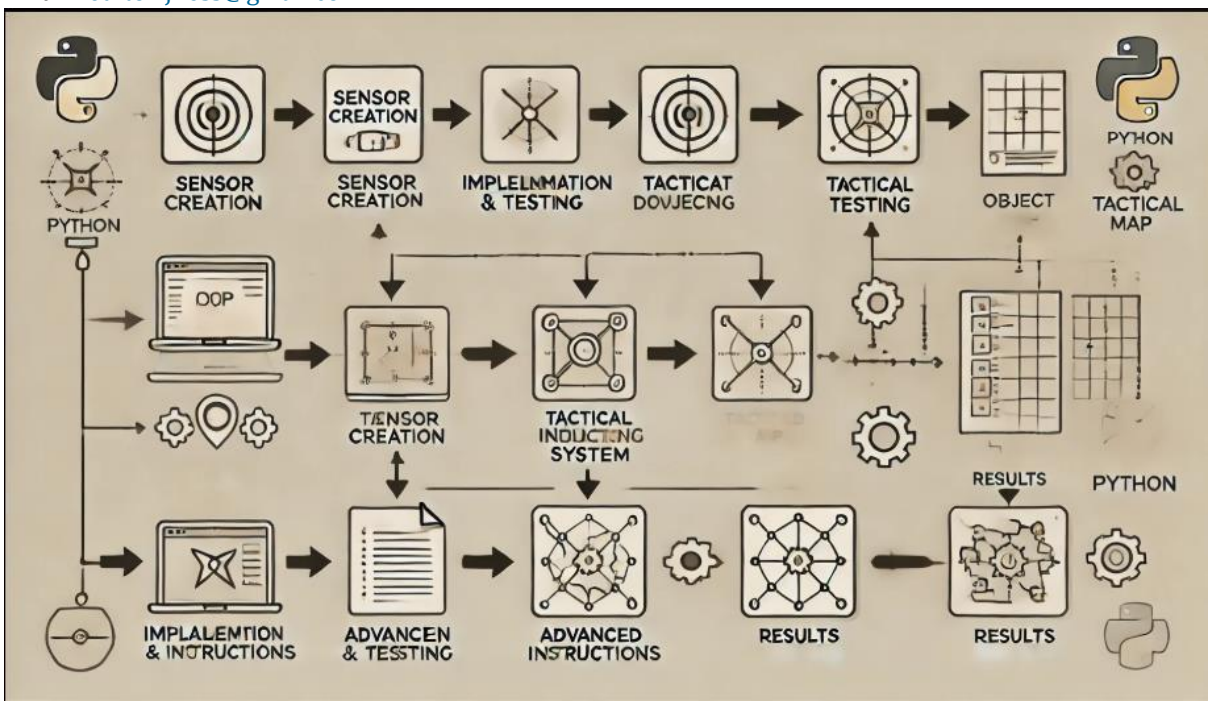

Figure 1. Framework Model

RESULT AND DISCUSSION

Creating a Sensor class in programming with Python is part of applying the concept of Object-Oriented Programming (OOP). The Sensor class is a representation of a physical sensor in code form. Virtual sensors can overcome limitations in physical sensor data integration, improve AI-based information systems and digital twinning (Martin et al., 2021). Object-oriented programming using Python involves the basic grammar of the language and basic OOP knowledge and skills (Y. Wu, 2019). The Sensor class serves as a blueprint or template for creating sensor objects that can be used in simulations or real applications, such as monitoring or surveillance systems.

```python
class Sensor:
    def __init__(self, id, location, status=True):
        self._id = id
        self._location = location
        self._status = status

    def activate(self):
        self._status = True

    def deactivate(self):
        self._status = False

    def get_status(self):
        return self._status

    def get_location(self):
        return self._location

    def get_id(self):
        return self._id

    def __str__(self):
        return f"Sensor(ID: {self._id}, Location: {self._location}, Status: {'Active' if self._status else 'Inactive'})"
```

Figure 2. Creating a Sensor

The code above defines a class called Sensor in Python that represents a sensor with identity (_id), location (_location), and status (_status) attributes. The __init__ constructor initializes these three attributes when the sensor object is created, with a default state of active (True). This class provides methods to activate and deactivate the sensor, as well as methods to

get the status (get_status), location (get_location), and sensor ID (get_id). The __str__ method overrides the string representation of the sensor object to display the sensor ID, location, and status information in an easy-to-read format. This class encapsulates sensor information and behavior, providing an efficient way to manage the status and location of sensors in a system.

Creating a Tactical Map class serves as a representation of a tactical map used to manage and visualize various objects, such as sensors, in a given area. The Tactical Map class is a blueprint for creating map objects used in simulation or tactical monitoring and analysis applications. Blueprint creation is a four-stage process that identifies key knowledge and skill domains, assesses objectives, determines assessment methods, and allocates emphasis to each domain (Raymond & Grande, 2019). Using the Tactical Map class, we can create dynamic and interactive tactical maps, which can be used in a variety of applications, from military monitoring to security surveillance systems. This class makes it possible to manage and visualize sensor data in a geographical context, supporting better analysis of the situation on the ground.

```python
class TacticalMap:
    def __init__(self, map_size):
        self._map_size = map_size
        self.sensors = []

    def add_sensor(self, sensor):
        self.sensors.append(sensor)

    def remove_sensor(self, sensor_id):
        self.sensors = [sensor for sensor in self.sensors if sensor.get_id() != sensor_id]

    def get_active_sensors(self):
        return [sensor for sensor in self.sensors if sensor.get_status()]

    def display_map(self):
        for sensor in self.sensors:
            print(sensor)
```

Figure 3. Creating a Tactical Map

The code above defines a class called TacticalMap in Python that manages sensors in a tactical map. The class has an __init__ constructor that initializes the map size (_map_size) and an empty list of sensors that will hold the sensor objects. The add_sensor method is used to add new sensors to this list, while remove_sensor removes sensors based on the given ID. The get_active_sensors method returns a list of currently active sensors by checking the status of each sensor, and the display_map method prints the information of all sensors on the map by iterating over the list of sensors. This class provides a basic structure for sensor management in the context of a tactical map, allowing users to add, remove, and monitor the status of existing sensors.

Implementation is the stage where an idea or plan is put into action, while testing is the process of ensuring that the results of the implementation are as expected and error-free before the product or system is widely used. Implementation strategies are defined and implemented to overcome barriers with a focus on testing their effectiveness in research trials. The test strategy focuses on formulating and executing a test plan that meets the established objectives, taking into account factors such as test priorities, objectives, and product identification and availability. The goal of implementation is to realize the technical and functional specifications that have been designed to make the product or system operate properly. Testing involves various techniques and methods to detect bugs, ensure system performance, test reliability, and evaluate overall product quality (Kirchner et al., 2020).

```
[ ]  # Membuat sensor
     sensor1 = Sensor(id=1, location=(5.888,95.317))
     sensor2 = Sensor(id=2, location=(-5.423,120.333), status=False)
     sensor3 = Sensor(id=3, location=(-8.467,140.399))

     # Membuat peta taktis
     tactical_map = TacticalMap(map_size=(100, 100))

     # Menambahkan sensor ke peta
     tactical_map.add_sensor(sensor1)
     tactical_map.add_sensor(sensor2)
     tactical_map.add_sensor(sensor3)

     # Menampilkan informasi semua sensor di peta
     print("Semua Sensor di Peta:")
     tactical_map.display_map()

     # Mengaktifkan sensor kedua
     sensor2.activate()
     sensor1.deactivate()
```

Figure 4. Implementation and Testing Sensor

```
     # Mendapatkan daftar sensor yang aktif
     print("\nSensor yang Aktif:")
     active_sensors = tactical_map.get_active_sensors()
     for sensor in active_sensors:
         print(sensor)

     # Menghapus sensor pertama
     #tactical_map.remove_sensor(sensor_id=1)

     # Menampilkan informasi semua sensor di peta setelah penghapusan
     print("\nSemua Sensor di Peta Setelah Penghapusan:")
     tactical_map.display_map()
```

Figure 5. Implementation and Testing Sensor and Tactical Map

The code above shows the implementation of using the `Sensor` and `TacticalMap` classes to manage sensors in a tactical map. First, three sensor objects are created using the `Sensor` class, each with an ID, geographic location, and status (active or inactive). The first and third sensors are initialized as active by default, while the second sensor is initialized as inactive. After that, a `TacticalMap` object is created with the specified map size. The three sensors are then added to the tactical map using the `add_sensor` method. Next, the information of all the sensors present on the map is displayed using the `display_map` method, which prints the details of each sensor. The initially inactive second sensor is then activated, while the first sensor is deactivated using the `activate` and `deactivate` methods respectively. The list of currently active sensors is retrieved using the `get_active_sensors` method, and the details of those sensors are displayed. At the end of the code, there is a commented section to remove the first sensor from the map using the `remove_sensor` method. After that, the information of all sensors on the map is again displayed to show the status of the map after deletion. This code. illustrates how sensors can be dynamically managed in a tactical map, including adding, removing, and managing sensor status.

Advanced Instruction refers to additional directions or commands given after basic instructions have been delivered. Instruction tuning models can achieve comparable performance to original instructions, even with simplified task definitions and misleading examples, which highlights the need for more reliable methods and evaluation [29] Typically, these advanced instructions are needed to clarify, expand, or detail the actions to be performed in order to achieve the desired results. Follow-up instructions are essential to ensure that all the steps are followed and that the task is completed efficiently and in accordance with the set objectives.

```
Semua Sensor di Peta:
Sensor(ID: 1, Location: (5.888, 95.317), Status: Active)
Sensor(ID: 2, Location: (-5.423, 120.333), Status: Inactive)
Sensor(ID: 3, Location: (-8.467, 140.399), Status: Active)

Sensor yang Aktif:
Sensor(ID: 2, Location: (-5.423, 120.333), Status: Active)
Sensor(ID: 3, Location: (-8.467, 140.399), Status: Active)

Semua Sensor di Peta Setelah Penghapusan:
Sensor(ID: 1, Location: (5.888, 95.317), Status: Inactive)
Sensor(ID: 2, Location: (-5.423, 120.333), Status: Active)
Sensor(ID: 3, Location: (-8.467, 140.399), Status: Active)
```

Figure 6. Advanced Instruction

The output generated from the code shows the various states and conditions of the sensors managed in a tactical map. At the initial stage, the information of all sensors on the map is displayed. Three sensors are displayed with their ID, geographic location (in coordinates), and status. The first and third sensors are in the active state, while the second sensor is in the inactive state. After that, the second sensor is activated, and the first sensor is deactivated through the methods available in the `Sensor` class. The list of active sensors is then displayed, showing that the second sensor is now active along with the third sensor. However, no sensor deletion occurs at this stage, so when all the sensors on the map are displayed again after the state change, the first sensor is still present but in an inactive state, while the second and third sensors remain active. This code illustrates how sensors on the map can be dynamically manipulated, including the activation and deactivation of sensors, and how these changes are reflected in the overall map display. The results shown by this code are a concrete example of how sensor information can be managed and monitored in a Python-based application.

```
[ ] import folium

    # Create a map centered at the middle of the tactical map
    map_center = [tactical_map._map_size[0] / 2, tactical_map._map_size[1] / 2]
    my_map = folium.Map(location=map_center, zoom_start=10)

    # Add markers for each sensor
    for sensor in tactical_map.sensors:
        location = sensor.get_location()
        tooltip = f"Sensor {sensor.get_id()}"
        icon_color = "green" if sensor.get_status() else "red"
        folium.Marker(location=location, tooltip=tooltip, icon=folium.Icon(color=icon_color)).add_to(my_map)

    # Display the map
    my_map
```

Figure 7. Result

Use the `folium` library to visualize the positions of sensors in the tactical map on an interactive map. The first step in the code is to create a map centered on the midpoint of the tactical map (`map_center`), which is calculated based on the map size (`_map_size`). This map is initialized with an initial zoom of 10, giving a fairly wide view of the map. Once the map is created, the code then adds markers to the map for each sensor in the `sensors` list of the `TacticalMap` object. The location of each sensor is retrieved using the `get_location()` method, and each marker is given a tooltip indicating the sensor ID. The color of the marker icon is set according to the sensor status: green (`green`) is used for active sensors, while red (`red`) is used for inactive sensors. These markers are then added to the map using `folium.Marker`. The end result is an interactive map that displays the position and status of each sensor on the tactical map. The user can see the location of the sensor on the map and easily identify its status based

on the color of the icon. This map is useful for providing a more intuitive and understandable visualization of the distribution and condition of sensors in an operational area, especially in the context of monitoring or tactical operations.

Figure 8. Interactive Map of Indonesian Territory

The image shows an interactive map covering Indonesia, with markers placed at strategic locations. The map was created using the `folium` library in Python, which allows interactive visualization of spatial data on top of a base map taken from OpenStreetMap. On the map, there are three markers indicating the location of sensors with different statuses. The first red marker, located in Banda Aceh, indicates that the sensor at that location is inactive. The other two green markers, located in Kendari and near the Papua New Guinea border, respectively, indicate that the sensors in those locations are active. This map provides a clear visual representation of the distribution and status of sensors over a wide area. The color of the icons gives an immediate indication of the operational status of each sensor: green for active sensors and red for inactive ones. With this interactive map, users can quickly understand field conditions and make decisions based on the information presented visually. This implementation is particularly useful in tactical or military monitoring scenarios, where sensor placement and status can affect operational response and strategy. The map makes it easy to identify areas that need more attention, such as areas with inactive sensors, so that corrective actions or strategy adjustments can be made immediately.

CONCLUSION

The application of the Object-Oriented Programming (OOP) concept using the Python programming language has proven effective in building a tactical object monitoring system to support defense operations. This system was developed by utilizing two main classes, namely Sensor and TacticalMap, which each represent a physical monitoring device and a tactical map for visualization and sensor management. The use of OOP allows the compilation of a modular system, easy to maintain, and flexible in developing additional features without disrupting existing functions. In its implementation, sensors can be dynamically activated, deactivated, added to, or removed from the tactical map. These functions are controlled through structured methods in each class, so that interactions between objects become more efficient.

Interactive map visualization using the folium library is an advantage in itself, as it presents spatial data in a visual form that is easy to understand. In the resulting map, sensor status is displayed using colored icons—green for active and red for inactive—which provide a clear picture of the monitoring situation in the operational area. Overall, this study shows that the

application of Python in an object-oriented approach has great potential to support modern defense systems, especially in the context of tactical object monitoring. This technology not only improves the efficiency of information management but also strengthens the real-time analysis and decision-making capabilities in military operations. With its flexibility and scalability, the system can be further developed to include advanced features, such as real-time sensor integration or artificial intelligence, to meet future defense challenges.

REFERENCES

Billing, D. C., Fordy, G. R., Friedl, K. E., & Hasselstrøm, H. (2021). The implications of emerging technology on military human performance research priorities. *Journal of Science and Medicine in Sport*, *24*(10), 947–953. https://doi.org/10.1016/j.jsams.2020.10.007

Butwall, M., Ranka, P., & Shah, S. (2019). Python in Field of Data Science: A Review. *International Journal of Computer Applications*, *178*(49), 20–24. https://doi.org/10.5120/ijca2019919404

Carneiro, T., Medeiros Da Nobrega, R. V., Nepomuceno, T., Bian, G.-B., De Albuquerque, V. H. C., & Filho, P. P. R. (2018). Performance Analysis of Google Colaboratory as a Tool for Accelerating Deep Learning Applications. *IEEE Access*, *6*, 61677–61685. https://doi.org/10.1109/ACCESS.2018.2874767

Chang, Y.-H., & Zhang, Y.-Y. (2022). Deep Learning for Clothing Style Recognition Using YOLOv5. *Micromachines*, *13*(10), 1678. https://doi.org/10.3390/mi13101678

Chmielewski, M., Sapiejewski, K., & Sobolewski, M. (2019). Application of Augmented Reality, Mobile Devices, and Sensors for a Combat Entity Quantitative Assessment Supporting Decisions and Situational Awareness Development. *Applied Sciences*, *9*(21), 4577. https://doi.org/10.3390/app9214577

Elrad, T., Filman, R. E., & Bader, A. (2001). Aspect-oriented programming. *Communications of the ACM*, *44*(10), 29–32. https://doi.org/10.1145/383845.383853

Ergenc, D., Schneider, F., Kling, P., & Fischer, M. (2023). Moving Target Defense for Service-Oriented Mission-Critical Networks. *2023 32nd International Conference on Computer Communications and Networks (ICCCN)*, 1–10. https://doi.org/10.1109/ICCCN58024.2023.10230175

Friedl, K. E. (2018). Military applications of soldier physiological monitoring. *Journal of Science and Medicine in Sport*, *21*(11), 1147–1153. https://doi.org/10.1016/j.jsams.2018.06.004

Gao, K., Yan, X., Liu, X., & Peng, R. (2019). Object defence of a single object with preventive strike of random effect. *Reliability Engineering & System Safety*, *186*, 209–219. https://doi.org/10.1016/j.ress.2019.02.023

Gomes, G., Delgado Neto, A. M., Bezerra, L. M., & Silva, R. (2019). An object-oriented approach to dual reciprocity boundary element method applied to 2D elastoplastic problems. *Multidiscipline Modeling in Materials and Structures*, *15*(5), 958–974. https://doi.org/10.1108/MMMS-05-2018-0095

J., P. G., & V., N. K. (2021). Google Colaboratory : Tool for Deep Learning and Machine Learning Applications. *Indian Journal of Computer Science*, *6*(3–4), 23. https://doi.org/10.17010/ijcs/2021/v6/i3-4/165408

Kirchner, J. E., Smith, J. L., Powell, B. J., Waltz, T. J., & Proctor, E. K. (2020). Getting a clinical innovation into practice: An introduction to implementation strategies. *Psychiatry Research*, *283*, 112467. https://doi.org/10.1016/j.psychres.2019.06.042

Korson, T., & McGregor, J. D. (1990). Understanding object-oriented: a unifying paradigm. *Communications of the ACM*, *33*(9), 40–60. https://doi.org/10.1145/83880.84459

Kung, P.-N., & Peng, N. (2023). Do Models Really Learn to Follow Instructions? An Empirical Study of Instruction Tuning. *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, 1317–1328. https://doi.org/10.18653/v1/2023.acl-short.113

Lian, V., Varoy, E., & Giacaman, N. (2022). Learning Object-Oriented Programming Concepts Through Visual Analogies. *IEEE Transactions on Learning Technologies*, *15*(1), 78–92. https://doi.org/10.1109/TLT.2022.3154805

Lieberherr, K. J., & Holland, I. M. (1989). Assuring good style for object-oriented programs. *IEEE Software*, *6*(5), 38–48. https://doi.org/10.1109/52.35588

Listewnik, P., Bechelany, M., Wierzba, P., & Szczerska, M. (2021). Optical-Fiber Microsphere-Based Temperature Sensors with ZnO ALD Coating—Comparative Study. *Sensors*, *21*(15), 4982. https://doi.org/10.3390/s21154982

Martin, D., Kühl, N., & Satzger, G. (2021). Virtual Sensors. *Business & Information Systems Engineering*, *63*(3), 315–323. https://doi.org/10.1007/s12599-021-00689-w

Ramon-Cortes, C., Amela, R., Ejarque, J., Clauss, P., & Badia, R. M. (2020). AutoParallel: Automatic parallelisation and distributed execution of affine loop nests in Python. *The International Journal of High Performance Computing Applications*, *34*(6), 659–675. https://doi.org/10.1177/1094342020937050

Raymond, M. R., & Grande, J. P. (2019). A practical guide to test blueprinting. *Medical Teacher*, *41*(8), 854–861. https://doi.org/10.1080/0142159X.2019.1595556

Reefe, M., Alfaro, O., Foster, S., Plavchan, P., Pepin, N., Banaji, V., Vidaurri, M., Webster, S., Banaji, S., Berberian, J., Bowen, M., Chimaladinne, S., Collins, K., Combs, D., Eastridge, K., Ellingsen, T., El Mufti, M., Helm, I., Jimenez, M., … Wittrock, J. (2022). Asynchronous object-oriented approach to the automation of the 0.8-meter George Mason University campus telescope in Python. *Journal of Astronomical Telescopes, Instruments, and Systems*, *8*(02). https://doi.org/10.1117/1.JATIS.8.2.027002

Reitenbach, M. (2022). The power of the snake: number theory with Python. *International Journal of Mathematical Education in Science and Technology*, *53*(12), 3484–3490. https://doi.org/10.1080/0020739X.2021.1998688

Snyder, A. (1986). Encapsulation and inheritance in object-oriented programming languages. *Conference Proceedings on Object-Oriented Programming Systems, Languages and Applications - OOPLSA '86*, 38–45. https://doi.org/10.1145/28697.28702

Stancin, I., & Jovic, A. (2019). An overview and comparison of free Python libraries for data mining and big data analysis. *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 977–982. https://doi.org/10.23919/MIPRO.2019.8757088

Stevanoski, G., Kocev, I., Achkoski, J., Koceski, S., & Temelkovski, B. (2016). Implementation of a System for Physiological Status Monitoring by using Tactical Military Networks. *Defence Science Journal*, *66*(5), 517. https://doi.org/10.14429/dsj.66.9920

Wang, W., Liu, H., Lin, W., Chen, Y., & Yang, J.-A. (2020). Investigation on Works and Military Applications of Artificial Intelligence. *IEEE Access*, *8*, 131614–131625. https://doi.org/10.1109/ACCESS.2020.3009840

Wu, H., Hu, F., & Fang, Q. (2019). A comparative study for the impact performance of shaped charge JET on UHPC targets. *Defence Technology*, *15*(4), 506–518. https://doi.org/10.1016/j.dt.2019.04.005

Wu, Y. (2019). Object-oriented Programming Course Reform Using Python Language in the Background of Artificial Intelligence. *Proceedings of the 2019 3rd International Conference on Education, Management Science and Economics (ICEMSE 2019)*. https://doi.org/10.2991/icemse-19.2019.20

Yelamali, V., & Beelagi, S. (2021). The Results of Classification of Lab Assignment in Object Oriented Programming and Database Management System Lab: A Case Study. *Journal of Engineering Education Transformations*, *34*(0), 277. https://doi.org/10.16920/jeet/2021/v34i0/157155

Zhang, H., Zheng, K., Wang, X., Luo, S., & Wu, B. (2019). Efficient Strategy Selection for Moving Target Defense Under Multiple Attacks. *IEEE Access*, *7*, 65982–65995. https://doi.org/10.1109/ACCESS.2019.2918319

Zhou, Y., Wang, X., Guo, S., Wen, Y., & He, J. (2021). A cost-effective adaptive random testing algorithm for object-oriented software testing. *Journal of Intelligent & Fuzzy Systems*, *41*(3), 4415–4423. https://doi.org/10.3233/JIFS-189701