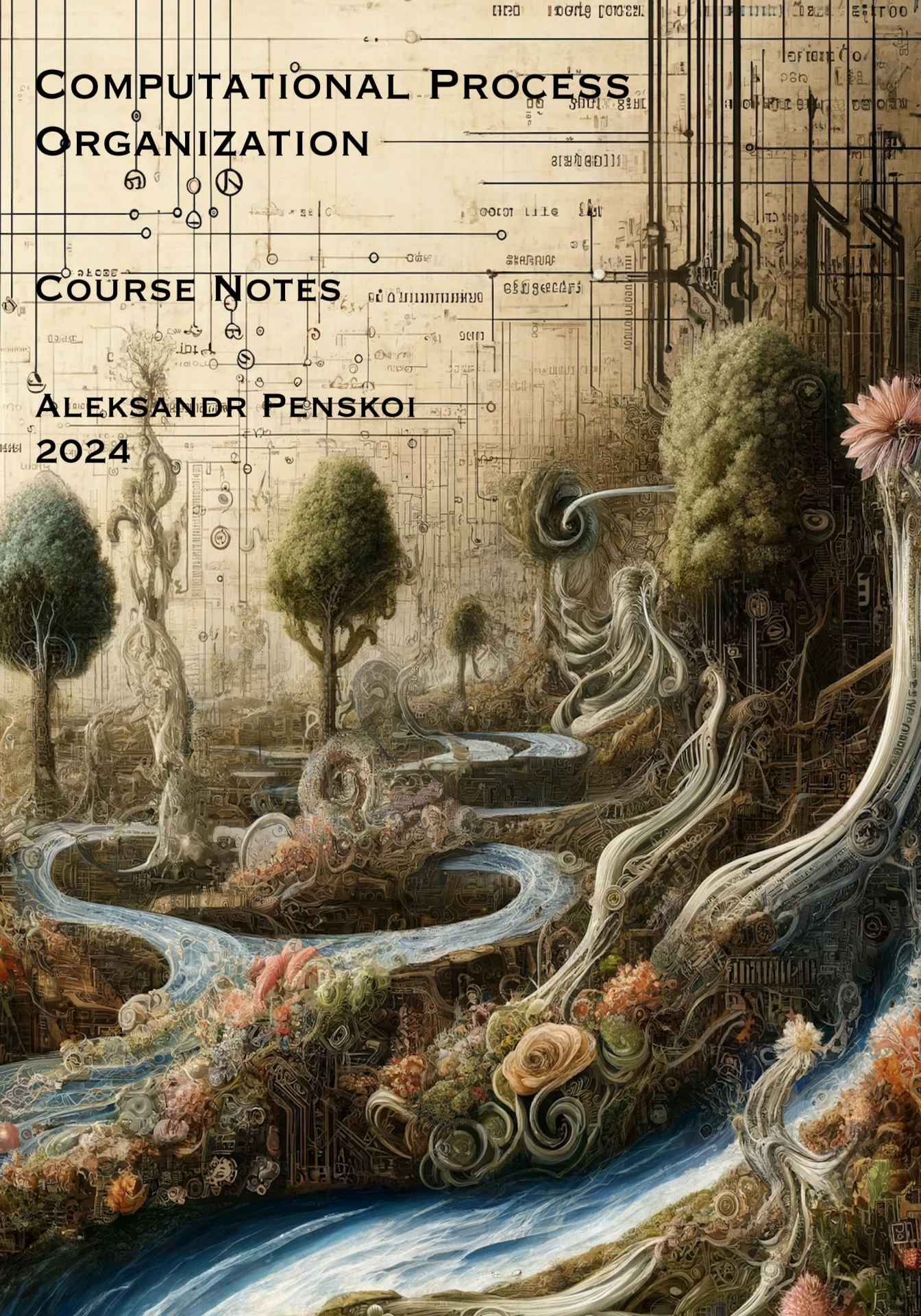
Computational Process Organization

Course Notes

Aleksandr Penskoi

2024



Computational Process Organization

Course Notes

Aleksandr Penskoi

This book is available at
http://leanpub.com/computational-process-organization

This version was published on 2025-02-05

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

© 2025 Aleksandr Penskoi

Contents

Chapter 1: Introduction 1
Rationale 2
Theory Structure 5

Part I: Computer and Process as a Whole 6

Chapter 2: System Concept and System Life Cycle 7
System Engineering 7
Internal Organization 7
Operational Environment 8
Life Cycle 10

Part II: Basic Computational Process Description 14

Part III: Computational Process Management 15

Part IV: Appendix. Grade System 16

Part V: Appendix. Laboratory Course 18

Chapter 3: Laboratory General Requirements 20
Workspace 20
Requirements for All Laboratory Works 20
Submission and Review Process 21
Evaluation 22

Chapter 4: Lab 1: Mutable Algorithms and Data Structure Implementation 23
Variants 25
Design and Develop Examples 28
Test Mutable Version 31
Analysis and Conclusion 34

Part VI: Appendix. Essay Task 35
Essay 1: The Future of Programming Tools 35
Essay 2: Modeling in Engineering and Computer Science 35
Essay 3: How Technology Loses Out in Companies, Countries & Continents 36
Evaluation 36

Part VII: References 37
Theory 37
Practice 38
Additional 38

Chapter 1: Introduction

> We are about to study the idea of a computational process. Computational processes are abstract beings that inhabit computers. As they evolve, processes manipulate other abstract things called data. The evolution of a process is directed by a pattern of rules called a program. People create programs to direct processes. In effect, we conjure the spirits of the computer with our spells.
>
> A computational process is indeed much like a sorcerer's idea of a spirit. It cannot be seen or touched. It is not composed of matter at all. However, it is very real. It can perform intellectual work. It can answer questions. It can affect the world by disbursing money at a bank or by controlling a robot arm in a factory. The programs we use to conjure processes are like a sorcerer's spells. They are carefully composed from symbolic expressions in arcane and esoteric programming languages that prescribe the tasks we want our processes to perform (Abelson 1996, SICP).

This course is designed for master students and aims to provide an overview and organize knowledge in computer science, computer engineering, software engineering, etc. The course's main topic is the computational process as a goal of computer system design, development, production, and utilization.

The course consists of lecture and laboratory parts. In the lectures, students receive computational organization problem statements, knowledge about different methods of computation process descriptions, and computational machine organizations, and analysis in different practical applications, their features, and disadvantages. The students gain practical experience with different technologies, tools, and approaches in laboratory work to get hands-on experience.

Course prerequisites: a basic understanding of computer organization, basic skills in structured or object-oriented programming, a common understanding of data structures and algorithms, and software developing tools and workflow.

This course is aimed at senior students who already have basic knowledge of modern computer system architecture and organization, have practical skills in software development, and optionally hardware.

The main course goal is to form a complete understanding of a whole computer system and its development and design process.

Engineers and businesses develop computer systems (from a text editor to a Smart Grid) for their run-time (utilization stage) and its computational process–some data transformations or processing under requirements. For example, processing data for its visualization on display, processing data from sensors to control the robot's servo motors, or collecting weather forecasting data.

The course places computational process organization into focus. It starts from the concept stage (how to formalize an application domain process) and ends on its actualization in its operational environment. The main course focuses on how computer system developers can describe a process, how they can benefit from that, and how computers perform it.

Students can use this course in two different ways:

1. To combine previously obtained fragmented knowledge obtained in specialized courses (for example, operating systems, peripheral device interfaces, processor architecture, etc.) into a whole view.
2. To form a comprehensive understanding of computer systems and their ways of organization for a specialist focused on specific issues before diving into a more narrow domain. To get a proper association chain.

Unfortunately, these course circumstances do not allow diving deeply into most of the issues raised. The course is more focused on setting parallels and relationships between different domains and knowledge. For this reason, students are recommended (or should be for some topics) to read reference materials for a detailed explanation and understanding. Be aware; reference material may have another point of view, glossary, or aims.

Rationale

The course was developed not due to the academic purpose of building a unified paradigm of information technology and computer science but to

the attendance of specific practical problems that many computer system developers face.

As we know, a significant part of projects in information technology exceed the stated deadlines and budgets, often by times. Also, we can remember two fundamental laws of computer engineering:

Murphy's law [1]
: Anything that can go wrong will go wrong.

Hofstadter's law [2]
: It always takes longer than you expect, even when you take into account Hofstadter's Law.

Real-world project statistics show that the main parts of mistakes (and the most expensive ones) are made at early project stages (requirements collection and architectural design) when the most significant design decisions are made.

Also, our experience allows us to formulate the following theses:

1. Humanity creates too much knowledge in the information technology domain that it does not fit into one person. We need specialization at the level of business, subject area specialist, technical specialist, and specific computer engineering issues. Specialists in user interfaces (web interface, desktop interface, mobile interface, etc.), server-side development (operational systems, programming languages, or sets of libraries or frameworks), hardware (digital/analog devices, FPGA/ASIC, HLS, etc.), security, operation systems, etc. The result is an increase in communication costs in a team and unbalanced project decisions. Such decisions usually cause delayed effects that are difficult to be fixed. A small example from the embedded system can select a twice cheaper processor for our small-scale controller and spend twice more time to program it (of course, it significantly increases the integral project budget).
2. Extra partitioning of responsibility among workers is caused by developer specializations or characteristics of the development, deployment, and support organization. Each specialist works on the system in a strictly assigned area and usually has no idea about other parts of it. Great examples include system administrators and system developers. Today, that partition has led to a new "hybrid" specialization—DevOps. This

[1] https://en.wikipedia.org/wiki/Murphy%27s_law
[2] https://en.wikipedia.org/wiki/Hofstadter%27s_law

partition results in no one being responsible for the most critical system-level issues. Another common case is a project where nobody has a full and holistic understanding of the system.

3. A significant amount of unexpected costs appears during integration with external (operational environment) or internal (modules or platforms) dependencies. They arise from incorrect expectations and assumptions about their functioning and behavior. In other words, the inability to collect information about the working environment fully before development starts.

The root of these problems is different people's backgrounds, concerns, and interests, which rigidify their attention and make many significant system success aspects invisible to them (see on Figure 1 from (Partridge, 1996)).

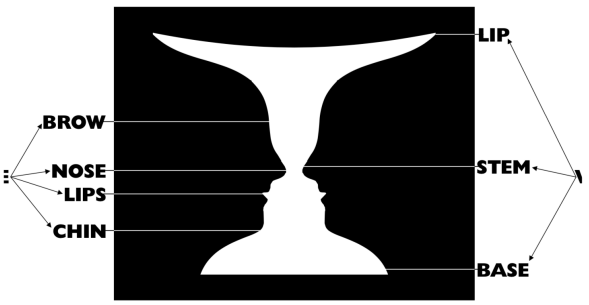


Figure 1. Two views of the same underlying pictures

The course aims to partly solve these problems by extending students' views, filling white spaces, and making bridges between IT concepts. It forms wide expertise of a range of technologies and techniques that are "widely used in narrow areas," which can play a significant role in your decision-making into future projects. It will allow you to understand a computer system life cycle, related tools, and project design space and see many alternative decisions. That knowledge is required to make correct holistic decisions, predicting their consequences, benefits, and disadvantages.

Some examples of such topics:

- Modern development technologies like Model-Driven Engineering, programming languages with non-mainstream elements, and advanced tools (from the web framework to High-Level Synthesis).

- The generalized point of view on a programmable system and programming mechanism.
- Verification and validation, certified programming, and property-based testing.
- Machine learning and specialized computational platforms. How to describe computational flow (neural network structure and learning method) independently from a computational description and computational machines?

Theory Structure

The course's theoretical framework is divided into three distinct parts, each building upon the last to provide a comprehensive understanding of computational processes within computer systems.

1. **Part I: Computer and Process as a Whole**

   This section delves into the holistic understanding of computing systems, including the computing process, life cycle, and various stages. It aims to provide a holistic representation of a computer system, discussing the qualities of a computer system and strategies for their management.
2. **Part II: Basic Computational Process Description**

   Here, we explore the nuts and bolts of computational process descriptions and performance. This includes models of computation, principles of modern computers, programming languages, and the use of reusable objects for the swift and efficient development of modern computer systems.
3. **Part III: Computational Process Management**

   The focus shifts to the automation of computational process organization. It discusses how developers can delegate direct operation of process elements to computers, streamlining the computational process management.

This text offers a brief overview of numerous topics related to "Computational Process Organization." It is important to note that many subtopics are summarized concisely. For a deeper understanding, students are encouraged to consult the referenced materials, which typically cover topics more thoroughly with detailed examples.

Part I: Computer and Process as a Whole

Chapter 2: System Concept and System Life Cycle

System Engineering

The system concept is one of the fundamental concepts in engineering and science. That concept is taken into account by the system engineering discipline, aiming to create a successful system.

Systems Engineering (SE) **(SEBoK, 2014)**

> is an interdisciplinary approach and means to enable the realization of successful systems. It focuses on holistically and concurrently understanding stakeholder needs; exploring opportunities; documenting requirements; and synthesizing, verifying, validating, and evolving solutions while considering the complete problem, from system concept exploration through system disposal.

To fully represent a computational process and a computer system, we should briefly overview the system concept from three points of view: internal organization, an operational environment, and a life cycle.

Internal Organization

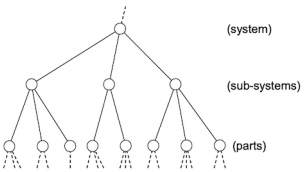
Classical system term definition with two notes can be found in (ISO 15288, 2008):

system

> a combination of interacting elements organized to achieve one or more stated purposes
>
> NOTE 1 A system may be considered as a product or as the services it provides.

> NOTE 2 In practice, the interpretation of its meaning is frequently clarified by the use of an associative noun, e.g. aircraft system. Alternatively the word system may be substituted simply by a context dependent synonym, e.g. aircraft, though this may then obscure a system principles perspective.

The key points of that definition say that a system is a construction from the interacting components (see the figure from (Gielingh, 2008)). For any computer system, those components are semiconductors, capacitors, resistors, triggers, etc., but that set of elements is useless, except you make an application-specific integrated circuit (ASIC). Also, it includes a lot of information components like programs, data, and so on. The nature of information components we will recognize in section .

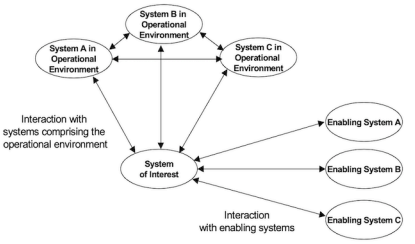


In theory, we can go deep inside a system organization to elemental particles, but it is not only useless, but it will also be harmful. The selection of component granularity, abstraction level, and computational platforms is critical in computer system development. That selection should be relevant to a computer system and business requirements. For example, if you make a web application—you should work on a web framework with web page components; if you make an embedded controller—you may work on a bare-metal level with the controller, its periphery. If developers build a system on a too low level, it will raise the complexity and cost of system development. If developers try to build a system on a high level, the system implementation will be surplus and may not fit requirements.

Operational Environment

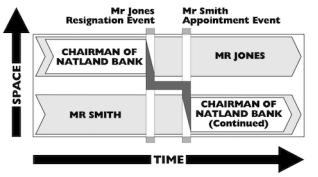
As we say ahead, a computer system was developed for its operation, run-time. The key property of a computer system is its functionalities. Functionalities also depend on operational environments, in which people, other systems, or

physical objects interact with the system-of-interest (see the figure, what is an enabling system will be described in the next paragraph).



(ISO 15288, 2008)

At this point, we can ask about system identification (What is the system? What is the boundary of the system? Where we restrict our responsibility for the system?). Let see an example in figure .



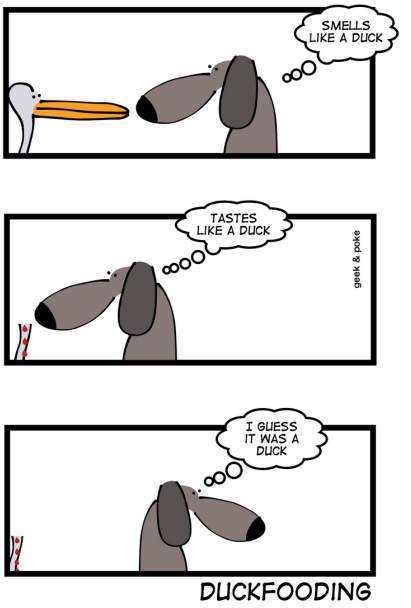
(Partridge, 1996)

We can see that at the start, Mr. Jones is a chairman of NatLandBank, but in the end, Mr. Smith replaces him. Question: "Is chairman of NatLandBank at the start and in the end the same?" The answer depends on our concerns and will be:

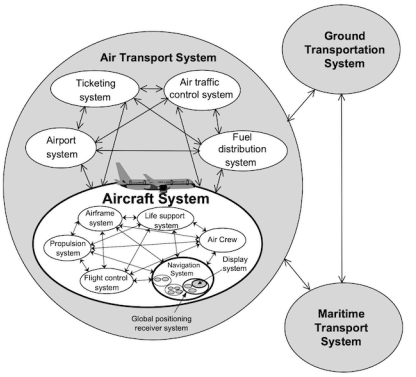
- "No" if we have a personal relationship with the chairman
- and "Yes" if we have a professional relationship with the chairman (or chairman functional place).

From the informational technologies view, "If it walks like a duck and it quacks like a duck, then it must be a duck". So, for a computer system, the operational environment is more important than the internal organization. If we save the interface to the operational environment, usually we can completely

change the internal organization, and nobody cares. For example, see modern mainframes, which execute source code from the seventies. That expects the punched cards but work with high-speed solid-state drive.



The main question of operational environment analysis is: "When should we stop?" Because we can extend it over and over again till it will be a Universe. A system engineer can find the answer only in a specific case. ISO 15288 gives us the following example for aircraft.



(ISO 15288, 2008)

Life Cycle

A system organization and its operational environment define the system in operation (or system-of-interest). But this is not enough for a successful system, which should be developed, deployed, and used. A holistic representation should include the system life cycle and its enabling systems.

system life cycle
: the evolution with time of a system-of-interest from conception through to retirement

enabling system
: a system that complements a system-of-interest during its life cycle stages but does not necessarily contribute directly to its function during operation

  NOTE 1 For example, when a system-of-interest enters the production stage, an enabling production system is required.

  NOTE 2 Each enabling system has a life cycle of its own. This International Standard is applicable to each enabling system when, in its own right, it is treated as a system-of-interest.

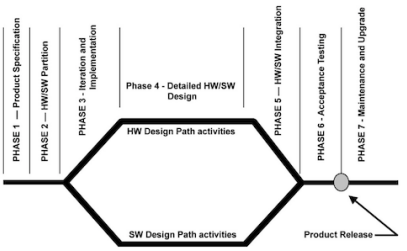
  (ISO 15288, 2008)

Let's briefly overview typical life cycle stages, which present for each system (for detail see (ISO 15288, 2008) Annex B), but not enough for any specific case:

1. The **Concept Stage** is executed to assess new business opportunities and develop preliminary system requirements and a feasible design solution. For a computer system, it is requirements analysis and architectural design.
2. The **Development Stage** is executed to develop a system-of-interest that meets acquirer requirements and can be produced, tested, evaluated, operated, supported, and retired. For a computer system, it is writing source code or computational process organization.
3. The **Production Stage** is executed to produce or manufacture the product, to test the product, and to produce related supporting and enabling systems as needed. For a computer system, it is hardware development, preparing executable artifacts, and making distributive.

4. The **Utilization Stage** is executed to operate the product, deliver services within intended environments, and ensure continued operational effectiveness.
5. The **Support Stage** is executed to provide logistics, maintenance, and support services that enable continued system-of-interest operation and sustainable service.
6. The **Retirement Stage** is executed to provide for removing a system-of-interest and related operational and support services and to operate and support the retirement system itself. Usually, for a computer system, that stage includes preparation for modernization or replacement.

For more detail, see an example of an embedded system life cycle in the figure, related to the hardware/software codesign approach.



> Hardware/software codesign investigates the concurrent design of hardware and software components of complex electronic systems. It tries to exploit the synergy of hardware and software with the goal to optimize and/or satisfy design constraints such as cost, performance, and power of the final product. At the same time, it targets to reduce the time-to-market frame considerably.

Complex representation of a system is shown on the figure (ISO 15288, 2008):

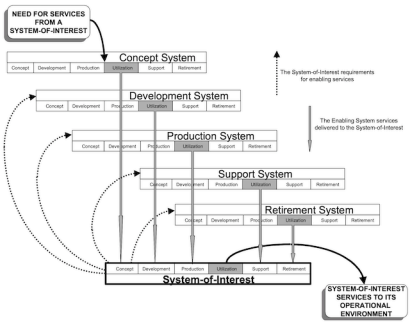


Figure 2. System interaction with Typical Enabling Systems

Part II: Basic Computational Process Description

Part III: Computational Process Management

Part IV: Appendix. Grade System

Below is the breakdown of grades for the course. The resulting grade of the course is made up of grades for laboratory works (60%), tests (15%), the final exam (16%), and essays (9%). Also, you can get additional points (5%) if you find mistakes in the course notes.

The description of the result mark is presented below:

- 91–100 score– excellent (pass– A)
- 84–90 score– very good (pass– B)
- 75–83 score– good (pass– C)
- 68–74 score– satisfactory (pass– D)
- 60–67 score– satisfactory (pass– E)
- 0–59 score– unsatisfactory (fail– F)

The course score is formed according to the table. The theoretical part of the course is evaluated by:

- tests
- writing essays on the defined topic
- final exam.

Figure 3. Score Point Distributions

Task	**optional**	**min**	**max**[1]
Test 1. “Computer system and computational process as a whole”		3	5
Test 2. “Basic computational process description”		3	5
Test 3. “Computational process management”		3	5
Essay 1	+	0	3
Essay 2	+	0	3
Essay 3	+	0	3
Laboratory work 1		14	20
Laboratory work 2		14	20
Laboratory work 3		14	20
Laboratory work 4 (optional and advanced)	+	0	20
Final test (whole course)		9	16
Finding mistakes in course notes[2]	+	0	5
Total		60	100 [+ 25]

[1]If several students find the same mistake, only first will be counted. The final score depends on the number of mistakes found by all students.

[2]The references used in the course materials can be divided into two groups: reference materials used as an argument or source of additional information and training materials that can simplify the learning process.

Part V: Appendix. Laboratory Course

The laboratory course aims to develop practical skills in the organization of computing processes, primarily through the development of tools for their description and evaluation.

It would be ideal if, during this practical course, students could use a wide variety of tools and programming languages. However, we have certain limitations. Consequently, we will only use Python 3 . The choice of this language represents a compromise due to its successful balance of complexity, capabilities, ubiquity, multi-paradigm nature, and practical value. Nevertheless, it should not be assumed that this technology lacks serious disadvantages or cannot be replaced. Some software engineers say, "Python is the second-best language in any domain." We hope this language will be beneficial for all students in their current or future endeavors.

If possible, we recommend repeating this laboratory course using some non-mainstream programming languages, such as Haskell, Smalltalk, Clojure, Rust, etc.

The laboratory course includes the following works:

- Lab 1: Mutable Algorithms and Data Structure Implementation

  This lab aims to assess students' knowledge and skills in programming. It also focuses on acquiring programming abilities that are often overlooked in many curricula.
- Lab 2: Immutable Algorithms and Data Structure Implementation

  This lab aims to shift students' perspectives on familiar concepts by transitioning from mutable to immutable ways of working with data, which is gaining popularity in real-world programming with the increase in the number of processors and computers.

- Lab 3: Basic Model of Computation

  It focuses on acquiring skills in developing programmable/configurable systems and developing embedded specialized programming languages. Computational processes can be organized differently from von Neumann processors and procedural programming.
- Lab 4: Computational Process Management

  It addresses questions about computational process management: How to manipulate computational process components and organize them in complex processes to solve practical tasks? This lab is optional and will only be available to the few best student groups in the course.

All laboratory works should be performed in groups of 1 to 2 members. The group size will be considered during the evaluation process.

Chapter 3: Laboratory General Requirements

To enhance the efficiency and quality of laboratory work evaluation, the results must adhere to the following requirements:

Workspace

Initially, you should prepare your workspace by installing the recommended software:

- Python 3.9 or newer, available at https://www.python.org/downloads
- Version control system git, available at https://git-scm.com/downloads

  Remember to configure git personally with the following commands (substitute with your actual email and name in Latin):

```
    ```shell
    git config --global user.name "Dr. Henry Jekyll"
    git config --global user.email jekyll@example.com
    ```
```

- IDE: PyCharm, Community Edition, available at https://www.jetbrains.com/pycharm/download (recommended)
- Library for Property-Based Testing (PBT): Hypothesis, available at https://github.com/HypothesisWorks/hypothesis

For detailed installation instructions tailored to your operating system and preferences, please visit the respective developers' websites. Should you encounter any problems, contact the instructor.

Requirements for All Laboratory Works

1. Use the template for each laboratory work available at edu-cpo-lab-template, which provides:

- The basic project structure.
- Setup for GitHub actions (see: .github/workflows/check.yml). It includes:
  - markdownlint for `README.md`
  - ruff for linting and formatting
  - pytest for running your tests
  - mypy for type checking
  - coverage for test coverage

2. In `README.md`, you should:

   - Update the title.
   - Provide a description of your laboratory work.
   - Describe the project structure.
   - Explain each group member's contribution in the contribution section, which should be verifiable by `git log` and `git blame`.

   - List the key features of your implementation, ensuring it's current with each revision of the work.
   - Update the changelog upon each submission of the laboratory work.
   - Include explanations for non-trivial design decisions or address inquiries about them in the `Design note` section.
   - For figures, use the `figures/...` path.

3. Use only the English language for code, readme, and comments.
4. Ensure your project is free of temporary files and unnecessary artifacts.

Submission and Review Process

Before submission:

- Ensure all **GitHub actions have successfully passed**. Otherwise, the laboratory work will not be evaluated.

- Update all sections of `README.md`.

Submit your work via email with the following steps:

1. Send a review request to the instructor at cpo-course@yandex.ru with the other group member CC'd.
    - The email subject should include your group name, laboratory work number, and variant (e.g., "`Dangerous Ducks - lab 1 - variant 2`").
    - Send one work per email. **Do not combine multiple works in a single email.**
    - For resubmissions, **use the "reply all"** function in your email client and include the entire message history.
2. The instructor will review your submission once a week. Afterwards, you will:
    - Receive feedback with a list of issues and additional tasks to improve your score.
    - Be able to check your score in the registry.
3. Upon addressing the feedback, resubmit your work to the instructor.
    - Ensure GitHub actions have successfully passed.
    - Update all sections of `README.md`, especially the changelog.
    - The number of resubmissions is limited.
4. For questions or more detailed information, consult with the instructor.

Evaluation

The evaluation of laboratory work focuses on overall performance, encompassing the practice part results, responses to questions about the results, and the outcomes of any additional tasks assigned post-practical part.

Grading scale:

- 20– for completing the task, accurately answering questions, and completing an optional part of the task.
- 18– for completing the task and accurately answering questions.
- 16– for completing the task.
- 14– for completing the mandatory part of the task.
- 0– for partially completing the task.

Chapter 4: Lab 1: Mutable Algorithms and Data Structure Implementation

Objectives:

- Use development tools: Python 3, IDE/source code editor, Git, GitHub Actions, and laboratory work process.
- Design algorithms and data structures in mutable styles.
- Develop unit and Property-Based Tests.

Students should meet with tools and the typical development workflow on the classical development task in the first laboratory work: developing a library for a specific data structure.

Students should implement the data structure selected by the variants as a mutable object (interaction with an object should modify it if applicable). All changes should be performed in place, by changing the initial structure. Let's see a few examples:

- Set list element by index.

```
x = [1, 2, 3]
x[1] = 5
print(x)  # => [1, 5, 3]
```

- Filter list by a specific predicate.

```python
def mut_filter(lst, p):
    i = 0
    while i < len(lst):
        if not p(lst[i]):
            del lst[i]
        else:
            i += 1


lst = [1, 2, 3, 4, 5]
mut_filter(lst, lambda e: e % 2 == 0)
print(lst)  # => [2,4]
```

For your data structure, you should implement the following features (in the brackets, you can see examples of API for a single-linked list):

Figure 4. Features and API Examples

Description	API
Add a new element	`lst.add(3)`
Set an element with a specific index/key (if applicable)	`lst.set(1, 3)`
Remove an element (key, index, or value)	`lst.remove(3)`
Size	`n = lst.size()`
Is a member	`b = lst.member(3)`
Reverse (if applicable)	`lst.reverse()`
From built-in `list`	`lst.from_list([12, 99, 37])`
To built-in `list`	`l = lst.to_list()`
Filter data structure by a specific predicate	`lst.filter(is_even)`
Map structure by a specific function	`lst.map(increment)`
Reduce process elements and build a value by the function	`x = lst.reduce(sum)`
Data structure should be an iterator	`i = iter(lst)` and `x = next(i)`
Data structure should be a monoid and implement `empty`	`x = List.empty()`

Description	API
Data structure should be a monoid and implement `concat`	`lst1.concat(lst2)`

Where:

- `map`– Built-in Functions[1]
- `reduce`– Higher-order functions and operations on callable objects from `functools`[2]
- Iterator– Python Iterator[3]

> Suppose that `S` is a set, and • is some binary operation `S × S → S`, then `S` with • (`concat`) is a monoid if it satisfies the following two axioms:
>
> **Associativity**
> For all `a`, `b` and `c` in `S`, the equation `(a • b) • c = a • (b • c)` holds.
> **Identity element**
> There exists an element `e` (`empty`) in `S` such that for every element `a` in `S`, the equations `e • a = a • e = a` hold.
>
> – from Wikipedia. Monoid [4]

Pay extra attention to return values and corner cases like:

- What should happen if a user puts a `None` value into the data structure?
- What should happen if a user puts elements of different types (e.g., strings and numbers)?

After that, you need to verify your library with unit tests and Property-Based Tests (see :).

[1] https://docs.python.org/3/library/functions.html#map
[2] https://docs.python.org/3/library/functools.html#functools.reduce
[3] https://wiki.python.org/moin/Iterator
[4] https://en.wikipedia.org/wiki/Monoid#Definition

Variants

(1) Unrolled linked list[5]

- You can use the built-in `list` inside nodes.
- You need to check that your implementation works correctly with the `None` value.
- A user should specify node size.
- You need to implement functions/methods for getting/setting value by index.

(2) Dynamic array [6]

- You can use the built-in `list` inside a node with a fixed size.
- You need to check that your implementation works correctly with the `None` value.
- You need to implement functions/methods for getting/setting value by index.
- A user should specify a growth factor.

The growth factor is all about efficient memory usage and performance. How it works:

1. You have a chunk of memory. The chunk has a capacity (how many elements it can contain) and length (how many elements it contains right now).
2. You need to add a new element, but what if `capacity == length`? You don't have space for a new element. What will you need to do?

    1. Allocate a new chunk of memory (in Python, usually, it looks like `[None]*(capacity*growth_factor)`).
    2. Copy data from the old chunk to the new chunk.
    3. Add a new element to the new chunk.

[5] https://en.wikipedia.org/wiki/Unrolled_linked_list
[6] https://en.wikipedia.org/wiki/Dynamic_array

(3) Set based on binary tree[7]

- You need to check that your implementation works correctly with the `None` value.

(4) Set based on hash map, separate chaining[8]

- You can use the built-in `list` for storing buckets and a bucket itself.
- You need to check that your implementation works correctly with the `None` value.

(5) Set based on hash map, open addressing[9]

- You can use the built-in `list` for storing buckets and a bucket itself.
- You need to check that your implementation works correctly with the `None` value.
- A user should specify a growth factor.

The growth factor is all about efficient memory usage and performance. How it works:

1. You have a chunk of memory. The chunk has a capacity (how many elements it can contain) and length (how many elements it contains right now).
2. You need to add a new element, but what if `capacity == length`? You don't have space for a new element. What will you need to do?

    - Allocate a new chunk of memory (in Python, usually, it looks like `[None]*(capacity*growth_factor)`).
    - Copy data from the old chunk to the new chunk.
    - Add a new element to the new chunk.

[7] https://en.wikipedia.org/wiki/Binary_search_tree
[8] https://en.wikipedia.org/wiki/Hash_table#Separate_chaining
[9] https://en.wikipedia.org/wiki/Hash_table#Open_addressing

(6) Dictionary based on binary tree[10]

- You need to check that your implementation works correctly with the `None` value.
- You need to implement functions/methods for getting/setting value by key.

(7) Dictionary based on hash map, separate chaining[11]

- You can use the built-in `list` for storing buckets and a bucket itself.
- You need to check that your implementation works correctly with the `None` value.
- You need to implement functions/methods for getting/setting value by key.

(8) Dictionary based on hash map, open addressing[12]

- You can use the built-in `list` for storing data.
- You need to check that your implementation works correctly with the `None` value.
- You need to implement functions/methods for getting/setting value by key.

Design and Develop Examples

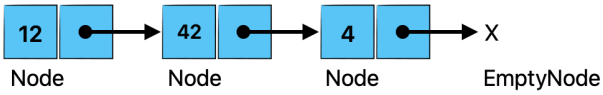
Let's see an example based on the single-linked list. First of all, we need to design how to represent the list data structure in Python. We have at least a few options:

1. Create a class for not-empty nodes and a class for the empty node.

   It can be relevant for the immutable version but not applicable for the mutable version. E.g., we have `lst = EmptyNode()`. If we mutably add a

[10] https://en.wikipedia.org/wiki/Binary_search_tree
[11] https://en.wikipedia.org/wiki/Hash_table#Separate_chaining
[12] https://en.wikipedia.org/wiki/Hash_table#Open_addressing

new element (`EmptyNode().add_to_tail('a')`), `lst` should change its type to `Node` type, which is not possible.

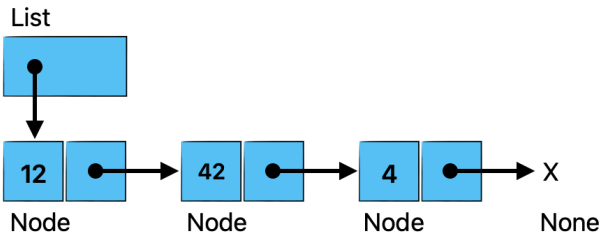



```
class EmptyNode:
    def __str__(self):
        return "Nil"


class Node:
    def __init__(self, value, next_):
        self.value = value
        self.next_ = next_


lst = Node(12, Node(99, Node(37, EmptyNode())))
```

2. Create a class for lists (a top-level node) and a class for a node.

   It is a good choice for the mutable version because:

   - It allows us to define methods.
   - It is not hard to implement.



Next, look at the fragment of the mutable library for list

```
class List:
    def __init__(self, root=None):
        self.root = root

    def __str__(self):
        """for str() implementation for printing"""
        return " : ".join(map(str, self.to_list()))

    def size(self):
        res = 0
        cur = self.root
        while cur is not None:
            res += 1
            cur = cur.next_
        return res

    def to_list(self):
        res = []
        cur = self.root
        while cur is not None:
            res.append(cur.value)
            cur = cur.next_
        return res

    def from_list(self, lst):
        if len(lst) == 0:
            self.root = None
            return
        root = None
        for e in reversed(lst):
            root = Node(e, root)
        self.root = root

    def add_to_head(self, value):
        self.root = Node(value, self.root)

    def _last_node(self):
        """helper function for find last node, should be private"""
        assert self.root is not None, "list should be not empty"
        cur = self.root
        while cur.next_ is not None:
            cur = cur.next_
        return cur

    def add_to_tail(self, value):
        if self.root is None:
            self.root = Node(value)
            return
        self._last_node().next_ = Node(value)

```

```
51      def map(self, f):
52          cur = self.root
53          while cur is not None:
54              cur.value = f(cur.value)
55              cur = cur.next_
56
57      def reduce(self, f, initial_state):
58          state = initial_state
59          cur = self.root
60          while cur is not None:
61              state = f(state, cur.value)
62              cur = cur.next_
63          return state
64
65      def __iter__(self):
66          return List(self.root)
67
68      def __next__(self):
69          if self.root is None:
70              raise StopIteration
71          tmp = self.root.value
72          self.root = self.root.next_
73          return tmp
74
75
76  class Node:
77      def __init__(self, value, next_=None):
78          """node constructor"""
79          self.value = value
80          self.next_ = next_
```

Here we implemented only the following features: `add`, `size`, conversion from and to Python list, `map`, `reduce`.

Test Mutable Version

For testing, you should use two approaches:

- Unit tests (for all features).
- Property-based tests (for features with specific properties, such as monoid properties and conversion with built-in `list`).

Next, look at the fragment of mutable library tests. The Property-Based Tests for `from_list` and `to_list` methods are `test_from_list_to_list_equality` and `test_python_len_and_list_size_equality`).

```python
import hypothesis.strategies as st
import pytest
from hypothesis import given
from mutable import List, Node


def test_size():
    assert List().size() == 0
    assert List(Node("a")).size() == 1
    assert List(Node("a", Node("b"))).size() == 2


def test_to_list():
    assert List().to_list() == []
    assert List(Node("a")).to_list() == ["a"]
    assert List(Node("a", Node("b"))).to_list() == ["a", "b"]


def test_from_list():
    test_data = [[], ["a"], ["a", "b"]]
    for e in test_data:
        lst = List()
        lst.from_list(e)
        assert lst.to_list() == e


def test_add_to_head():
    lst = List()
    assert lst.to_list() == []
    lst.add_to_head("a")
    assert lst.to_list() == ["a"]
    lst.add_to_head("b")
    assert lst.to_list() == ["b", "a"]


def test_add_to_tail():
    lst = List()
    assert lst.to_list() == []
    lst.add_to_tail("a")
    assert lst.to_list() == ["a"]
    lst.add_to_tail("b")
    assert lst.to_list() == ["a", "b"]


def test_map():
    lst = List()
    lst.map(str)
    assert lst.to_list() == []

    lst = List()
```

```
    lst.from_list([1, 2, 3])
    lst.map(str)
    assert lst.to_list() == ["1", "2", "3"]

    lst = List()
    lst.from_list([1, 2, 3])
    lst.map(lambda x: x + 1)
    assert lst.to_list() == [2, 3, 4]


def test_reduce():
    # sum of empty list
    lst = List()
    assert lst.reduce(lambda st, e: st + e, 0) == 0

    # sum of list
    lst = List()
    lst.from_list([1, 2, 3])
    assert lst.reduce(lambda st, e: st + e, 0) == 6

    # size
    test_data = [[], ["a"], ["a", "b"]]
    for e in test_data:
        lst = List()
        lst.from_list(e)
        assert lst.reduce(lambda st, _: st + 1, 0) == lst.size()


@given(st.lists(st.integers()))
def test_from_list_to_list_equality(a):
    lst = List()
    lst.from_list(a)
    b = lst.to_list()
    assert a == b


@given(st.lists(st.integers()))
def test_python_len_and_list_size_equality(a):
    lst = List()
    lst.from_list(a)
    assert lst.size() == len(a)


def test_iter():
    x = [1, 2, 3]
    lst = List()
    lst.from_list(x)
    tmp = []
    for e in lst:
        tmp.append(e)
```

```
    assert x == tmp
    assert lst.to_list() == tmp

    i = iter(List())
    with pytest.raises(StopIteration):
        next(i)
```

Execution results:

```
$ pytest . -v
================================== test session starts ==================================
platform darwin -- Python 3.12.2, pytest-7.4.4, pluggy-1.4.0 -- /Users/ryukzak/Library/Ca\
ches/pypoetry/virtualenvs/cpo-examples-tQ3HfS1q-py3.12/bin/python
cachedir: .pytest_cache
hypothesis profile 'default' -> database=DirectoryBasedExampleDatabase(PosixPath('/Users/\
ryukzak/edu/edu-cpo/manuscript/resources/src/lab1/.hypothesis/examples'))
rootdir: /Users/ryukzak/edu/edu-cpo/manuscript/resources/src
configfile: pyproject.toml
plugins: hypothesis-6.100.0, golden-0.2.2
collected 10 items

mutable_test.py::TestMutableList::test_add_to_head PASSED                          [ 10%]
mutable_test.py::TestMutableList::test_add_to_tail PASSED                          [ 20%]
mutable_test.py::TestMutableList::test_from_list PASSED                            [ 30%]
mutable_test.py::TestMutableList::test_from_list_to_list_equality PASSED           [ 40%]
mutable_test.py::TestMutableList::test_iter PASSED                                 [ 50%]
mutable_test.py::TestMutableList::test_map PASSED                                  [ 60%]
mutable_test.py::TestMutableList::test_python_len_and_list_size_equality PASSED    [ 70%]
mutable_test.py::TestMutableList::test_reduce PASSED                               [ 80%]
mutable_test.py::TestMutableList::test_size PASSED                                 [ 90%]
mutable_test.py::TestMutableList::test_to_list PASSED                              [100%]

================================== 10 passed in 2.57s ===================================
```

Analysis and Conclusion

After finishing software development and testing, you should:

- Note your implementation restrictions.
- Analyze advantages and disadvantages of unittest and PBT tests.

Part VI: Appendix. Essay Task

The essays should be small and specific, with a maximum size of 2048 symbols (including spaces). In essays, you should:

- demonstrate that you have learned the material (briefly describe its content)
- describe your thoughts on the material's topic.

This is an individual task.

Essay 1: The Future of Programming Tools

Before writing this essay, you should watch the two following presentations from the conference:

- The Future of Programming–Bret Victor–Dropbox's DBX conference, 2013
- The Worst Programming Language Ever–Mark Rendle–NDC Oslo 2021

If you have problems with access, let the teacher know.

In this essay, try to represent what you learned from the presentations listed above, and afterward, let's formulate your vision of the current state and trends in programming tools. Evaluate the thoughts that you got from the presentation.

Essay 2: Modeling in Engineering and Computer Science

Before writing this essay, you should watch the two following presentations from the conference:

- Science and Engineering for Cyber-Physical Systems–Edward Lee–2021
- Modeling in Engineering and Science–Edward Lee–Communications of the ACM–2019

If you have problems with access, let the teacher know.

In this essay, try to represent what you learned from the presentations and article listed above. After that, let's formulate your vision on the question: "Is it possible to design a computer system with predicted parameters and quality? How to do this?". Evaluate the thoughts that you got from the materials.

Essay 3: How Technology Loses Out in Companies, Countries & Continents

Before writing this essay, you should watch the two following presentations from the conference:

- How Technology Loses Out in Companies, Countries & Continents and What to Do About It–Bert Hubert–2021
- How Tech Loses Out over at Companies, Countries, and Continents–Bert Hubert–2021

In essays, you should:

- demonstrate that you have learned the material (in a short form, describe its content)
- describe your thoughts on the material's topic.

Evaluation

Grade points evaluation:

- 3–essay describes an integral and complete view of the topic, and some interesting/new ideas are presented, essay submitted on time
- 2–essay describes an integral and complete view of the topic, essay submitted on time
- 1–essay contains a limited topic description
- 0–the student skipped this task.

Part VII: References

Theory

Abelson 1996, SICP

H. Abelson, G. J. Sussman, A. J. Perlis, and J. Sussman, Structure and Interpretation of Computer Programs, 2nd ed. MIT Press, 1996.

Clements, 2010

Clements P. Documenting Software Architectures / P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, P. Merson, R. Nord, J. Stafford– Carnegie Mellon, 2010.

Dijkstra, 1989

Edsger W. Dijkstra.–The role of scientific thought."In selected works on computing: a personal perspective, 60-66. Springer-Verlag, 1982.

Gielingh, 2008

Gielingh, W., 2008. A theory for the modelling of complex and dynamic systems. Journal of Information Technology in Construction (ITcon), 13(27), pp.421-475.

ISO 15288, 2008

ISO/IEC 15288 Systems and software engineering–System life cycle processes. 2008.

ISO 42010, 2011

ISO/IEC/IEEE 42010 Systems and software engineering–Architecture description. 2011.

OMG Essence, 2018

Essence–Kernel and Language for Software Engineering Methods. Foundation for the Agile Creation and Enactment of Software Engineering Methods (FACESEM) RFP, 2018.

Ptolemaeus, 2014

Claudius Ptolemaeus, System Design, Modeling, and Simulation using

Ptolemy II, Ptolemy.org, 2014, https://ptolemy.berkeley.edu/books/Systems

Teich, 2012

Teich J. Hardware/Software Codesign: The Past, the Present, and Predicting the Future / J. Teich // Proc. IEEE–2012.–T. 100–№ Special Centennial Issue–1411-1430p.

SEBoK, 2014

The Guide to the Systems Engineering Body of Knowledge (SEBoK), V. 1.3. 2014.

Practice

Chacon, 2020

Scott Chacon, Ben Straub. Pro Git // apress–2020–second edition, https://git-scm.com/book/en/v2

Gruber, 2022

John Gruber, Markdown: Syntax, https://daringfireball.net/projects/markdown/syntax

Hypothesis Library

David R. MacIver, Hypothesis, https://github.com/HypothesisWorks/hypothesis

PEP 8, 2013

PEP 8–Style Guide for Python Code, https://www.python.org/dev/peps/pep-0008

Pilgrim, 2009

Pilgrim, Mark, and Simon Willison. Dive into python 3. Vol. 2. New York, NY, USA: Apress, 2009. https://diveintopython3.problemsolving.io

Additional

Atkinson, 2005

Atkinson C. A Generalized Notion of Platforms for Model-Driven Development Springer Berlin Heidelberg, 2005.–119-136c.

Baldwin, 2018

Baldwin C.Y. The Architecture of Platforms: A Unified View. / C. Y. Baldwin, C. J. Woodard // Work. Pap.–Harvard Bus. Sch. Div. Res.–2008.–31c.

Berger, 2001

Berger, Arnold. Embedded systems design: an introduction to processes, tools, and techniques. CRC Press, 2001.

Boehm, 2008

Boehm, B., Valerdi, R. and Honour, E. (2008), The ROI of systems engineering: Some quantitative results for software␣intensive systems. Syst. Engin., 11: 221-234. https://doi.org/10.1002/sys.20096

Booch, 2007

Grady Booch, Maksimchuk Robert A., Engle Michael W. et all. Object-Oriented Analysis and Design with Applications (3rd Edition). Addison-Wesley Professional; 3 edition, 2007

Dijkstra, 1968

Edgar Dijkstra, Go To Statement Considered Harmful, Aus: Communications of the ACM 11, 3 (March 1968). 147-148.

Ducasse, 2017

S. Ducasse, D. Zagidulin, N. Hess, D. Chloupis, "Pharo by Example 50," in Square Bracket Associates, 2017.

Evans, 2017

Josh Evans, Mastering Chaos - A Netflix Guide to Microservices, 2017 https://www.youtube.com/watch?v=CZ3wIuvmHeM

GeeksforGeeks,2018

8086 program to find the factorial of a number, GeeksforGeeks, https://www.geeksforgeeks.org/8086-program-find-factorial-number/

Gupta, 2022

Lokesh Gupta, What is REST, REST API Tutorial, https://restfulapi.net

Harrison, 1997

John Harrison, Introduction to Functional Programming, 1997, https://www.cl.cam.ac.uk/teaching/Lectures/funprog-jrh-1996/all.pdf

IEEE 1471, 2000

Group, IEEE Architecture Working. IEEE Std 1471-2000, Recommended Practice for Architectural Description of Software-intensive Systems. IEEE, 2000.

Jacobson, 2011

Ivar Jacobson, Discover the Essence of Software Engineering. CSI Communications, July 2011, http://semat.org/documents/20181/27952/Technical_Trends_3.pdf/f12257fc-81e9-4a25-8fef-c2076ebce105

Kennedy, 2018

William Kennedy, Scheduling In Go, Part I - OS Scheduler, Part II - Go

Scheduler, Part III - Concurrency, 2018, https://www.ardanlabs.com/blog/2018/08/scheduling-in-go-part1.html

Kerckhove, 2019

Tom Sydney Kerckhove, QuickCheck, Hedgehog, Validity. February 2019. available at: https://www.fpcomplete.com/blog/quickcheck-hedgehog-validity

Key, 2007

Alan Key, et al. STEPS Toward The Reinvention of Programming, VPRI Technical Report TR-2007-008

Kirpichov, 2009

E. Kirpichov, Elements of functional languages, "Practice of functional programming" Journal, vol. 3, 2009 (in Russian) / Евгений Кирпичёв, Элементы функциональных языков, Журнал «Практика функционального программирования», выпуск 3, 2009, http://fprog.ru/2009/issue3/practice-fp-3-screen.pdf

Lee, 2006

E. A. Lee, "The problem with threads," Computer (Long. Beach. Calif)., vol. 39, no. 5, pp. 33-42, 2006.

Lee, 203

Lee E.A. Actor-Oriented Design of Embedded Hardware and Software Systems / E. A. Lee, S. Neuendorffer, M. J. Wirthlin // J. Circuits, Syst. Comput. – 2003. – T. 12– 231–260c.

Levenchuk, 2015

Anatoly Levenchuk "Towards a Systems Engineering Essence." arXiv preprint arXiv:1502.00121 (2015).

Lundin, 2008

Kenneth Lundin, Inside the Erlang VM with focus on SMP, Erlang User Conference, Stockholm, November 13, 2008.

Mellor,2002

Mellor, Stephen J., and Marc J. Balcer. Executable UML: A Foundation for Model-driven Architecture. Boston: Addison-Wesley, 2002. Print.

Partridge, 1996

Partridge, C., Business Objects: Re-engineering for re-use. 1996.

Pierce, 2002

Benjamin C. Pierce, Types and Programming Languages, The MIT Press, Massachusetts Institute of Technology, 2002

Potok, 1999

Thomas E. Potok, Mladen Vouk, Andy Rindos. Productivity Analysis of

Object-Oriented Software Developed in a Commercial Environment // Software – Practice and Experience. – 1999. – Vol. 29, no. 10. – P. 833–847.

Sangiovanni-Vincentelli, 2001

Sangiovanni-Vincentelli A. Platform-based design and software design methodology for embedded systems / A. Sangiovanni-Vincentelli, G. Martin // IEEE Des. Test Comput.–2001.–T. 18–№ 6- 23-33c.

Sayfan, 2016

Gigi Sayfan, Deep Dive Into Python Decorators, 2016, https://code.tutsplus.com/tutorials/deep-dive-into-python-decorators--cms-29725

Seibel, 2005

P. Seibel, “Practical common lisp,” 2005.

Tellman, 2019

Z. Tellman, “Elements of Clojure,” 2019.

Terrell

Riccardo Terrell, Concurrency in .NET, Modern patterns of concurrent and parallel programming, https://livebook.manning.com/book/concurrency-in-dot-net/chapter-1/188

van Rossum, 2005

Guido van Rossum, Five-minute Multimethods in Python, 2005, https://www.artima.com/weblogs/viewpost.jsp?thread=101605

Weinberg, 1992

Weinberg, Gerald M. “Quality software management.” New York (1992).

Weinberg, 2017

Gerald Weinberg, What is Software?, 2017, http://secretsofconsulting.blogspot.com/2017/12/what-is-software.html

Zefirov, 2009

Zefirov Sergey, Too lazy to be afraid, “Practice of functional programming” Journal, vol. 1, 2009 (in Russian) / Сергей Зефиров, Лень бояться, Журнал «Практика функционального программирования», выпуск 1, 2009, http://www.fprog.ru/2009/issue1/serguey-zefirov-lazy-to-fear