



Unit 3

Programming Languages: History, Concepts & Design

Programming Languages

“Since developers can always find some slight flaw in any existing language, we are bound to witness the creation of new languages until the end of time”. [Shane Markstrum, *Staking Claims: A History of Programming Language Design Claims and Evidence A Positional Work in Progress*]



History of Programming Languages



Course: Secure Software Develop

Programming Languages - A Brief History

M1 Data Science | Université Paris

my-course.co.uk/Computing/Computer%20Science/SSDCS/SSDCS%20Lecturecast%202/content/index.html#/lessons/EEk1RZHhRQTkYnN2ThGAXzw_M6...

Programming Languages

25% COMPLETE

▼ WHAT IS A PROGRAMMING LANGUAGE?

What is a Programming Language?

Compiled vs. Interpreted

Data Types

Programming Paradigms

▼ A BRIEF HISTORY

A Brief History - Part 1

A Brief History - Part 2

The second graphic is from Sestoft (2020) (below) which classifies languages into academic, old mainstream or modern mainstream. It also covers a much wider time period, almost up to the present day (2020). It also exhibits a more Eurocentric focus on the events covered. Nonetheless, it still omits a number of key events in the evolution of programming languages.

The diagram is a timeline of programming languages from 1956 to 2010. It is divided into three main categories by dashed blue lines and green callouts: 'Mostly-academic' (top left), 'Old mainstream' (bottom), and 'Modern mainstream' (top right). Languages in the 'Mostly-academic' category include LISP, SASL, ML, SCHEME, PROLOG, SMALLTALK, SIMULA, ALGOL 68, ALGOL, C++, VISUAL BASIC, and C#. Languages in the 'Old mainstream' category include FORTRAN, COBOL, BASIC, PASCAL, ADA, and FORTRAN77. Languages in the 'Modern mainstream' category include HASKELL, STANDARD ML, CAML LIGHT, OCAML, F#, Scala, C# 2, C# 4, Java 5, VB.NET 10, Go, and JAVASCRIPT. The timeline shows the evolution of these languages over time, with many languages being derivatives of others.

(Sestoft, 2020)

Introduction to Pr....pdf

Show all

Type here to search

28°C Mostly cloudy

11:58 20/11/2021

Course: Secure Software Develop

Programming Languages - Comp

my-course.co.uk/Computing/Computer%20Science/SSDCS/SSDCS%20Lecturecast%202/content/index.html#/lessons/LCQ2I5AEf4ztjM-mNUWtdB7ggRV...

Programming Languages

6% COMPLETE

▼ WHAT IS A PROGRAMMING LANGUAGE?

What is a Programming Language?

Compiled vs. Interpreted

Data Types

Programming Paradigms


▼ A BRIEF HISTORY

A Brief History - Part 1

A Brief History - Part 2

Lesson 2 of 16

Compiled vs. Interpreted



A programming language is an abstraction designed to make it easier for humans to create programs to run on computing devices. However, for the computer to understand these programs they must first be converted into native machine code.

Introduction to Pr...pdf

Show all

Type here to search

W

28°C Mostly cloudy

11:39 20/11/2021



- ▼ WHAT IS A PROGRAMMING LANGUAGE?
 - What is a Programming Language? ✓
 - Compiled vs. Interpreted ✓
 - Data Types ✓
 - Programming Paradigms ○
- ▼ A BRIEF HISTORY
 - A Brief History - Part 1 ○
 - A Brief History - Part 2 ○

Introduction to Pr....pdf ^

Lesson 4 of 16

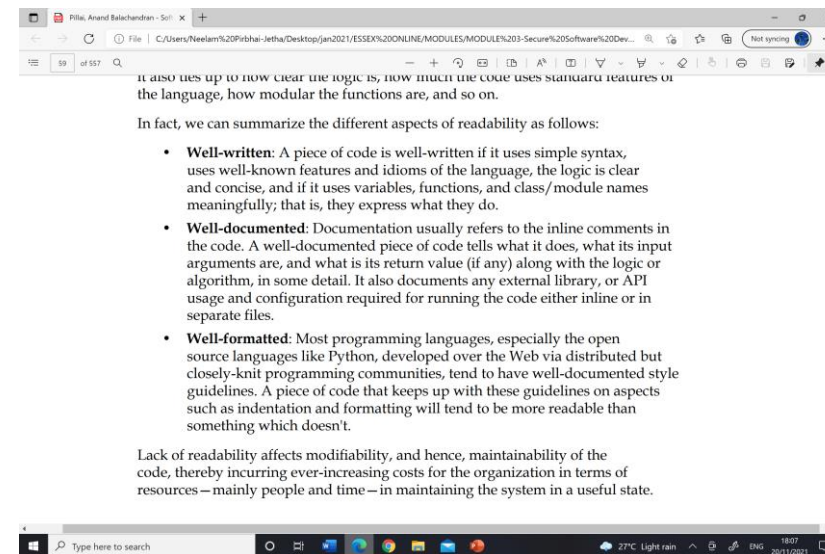
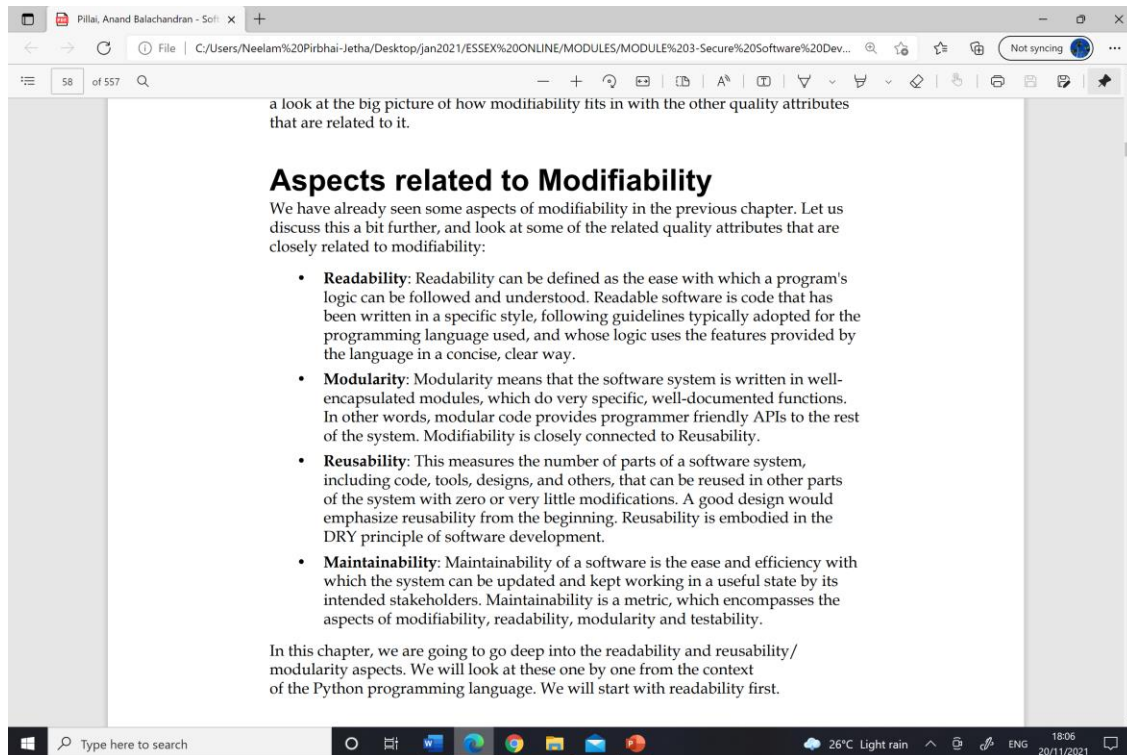
Programming Paradigms

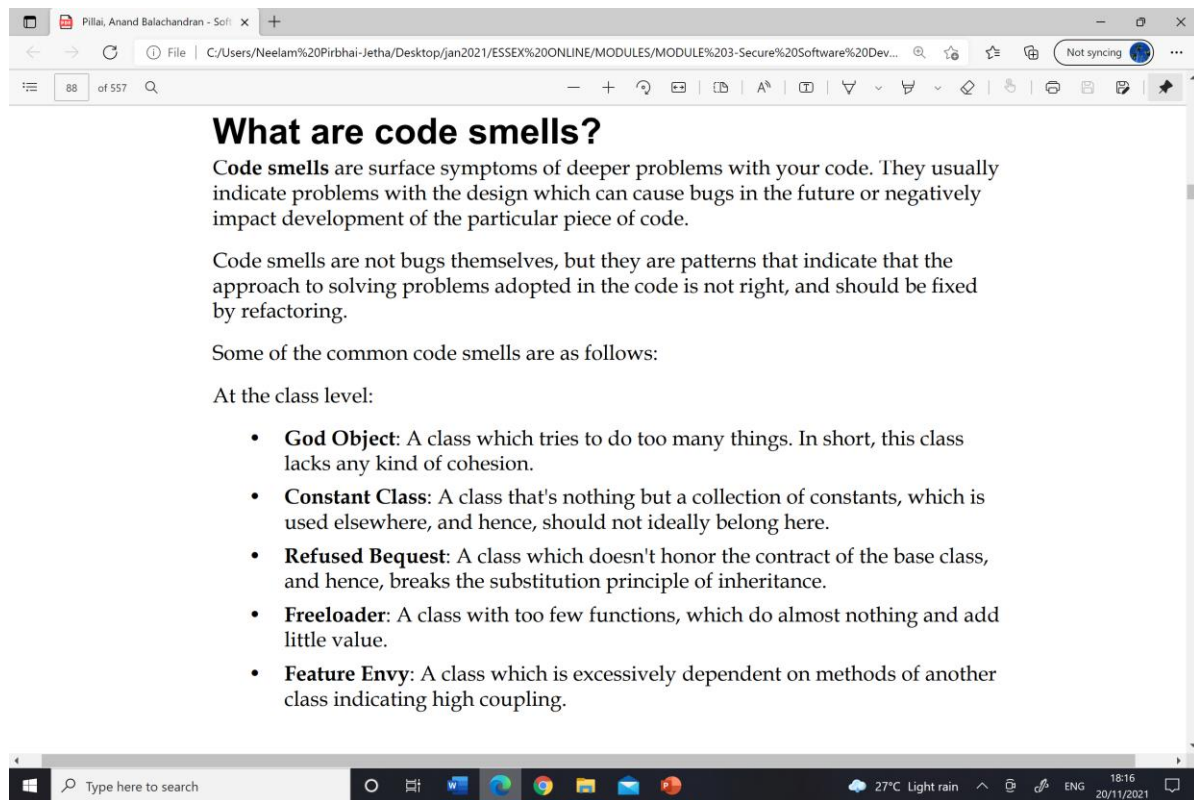
Van Roy (2009) defines a programming paradigm as 'an approach to programming a computer based on a mathematical theory or a coherent set of principles'.

Show all x

Chapter 2,6,7,8 of the course text (Pillai, 2017)

Writing Modifiable and Readable Code





What are code smells?

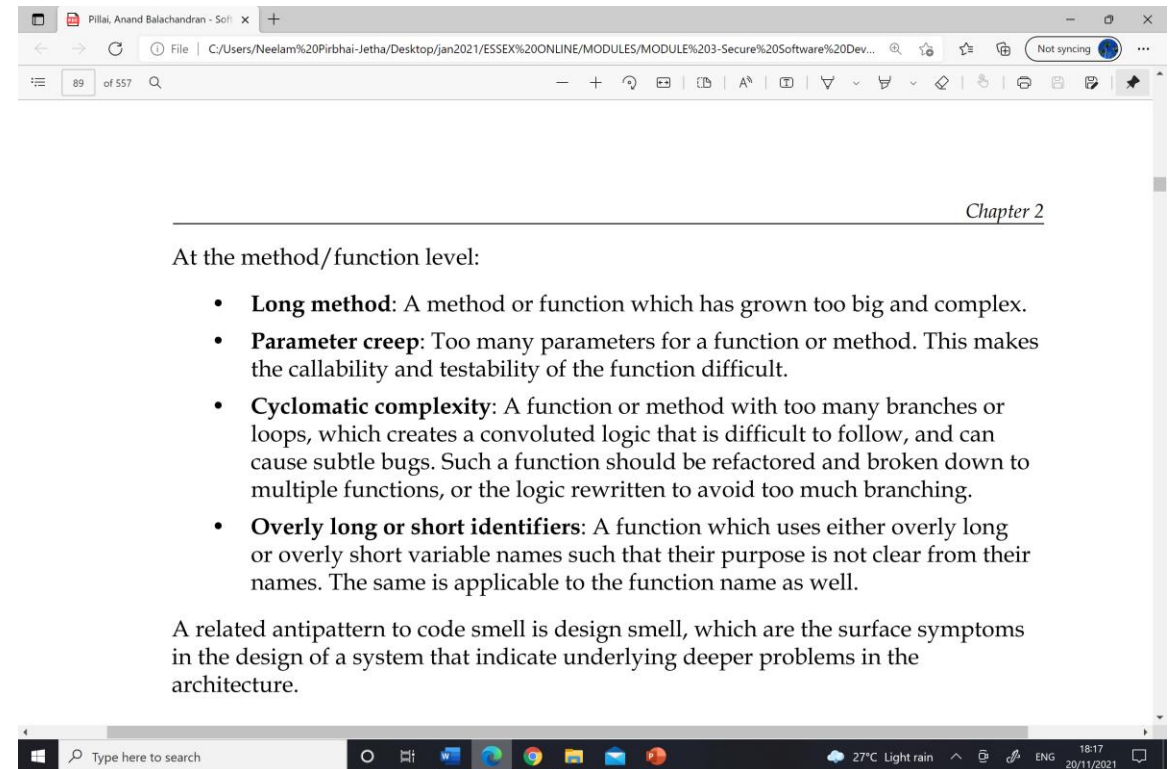
Code smells are surface symptoms of deeper problems with your code. They usually indicate problems with the design which can cause bugs in the future or negatively impact development of the particular piece of code.

Code smells are not bugs themselves, but they are patterns that indicate that the approach to solving problems adopted in the code is not right, and should be fixed by refactoring.

Some of the common code smells are as follows:

At the class level:

- **God Object:** A class which tries to do too many things. In short, this class lacks any kind of cohesion.
- **Constant Class:** A class that's nothing but a collection of constants, which is used elsewhere, and hence, should not ideally belong here.
- **Refused Bequest:** A class which doesn't honor the contract of the base class, and hence, breaks the substitution principle of inheritance.
- **Freeloader:** A class with too few functions, which do almost nothing and add little value.
- **Feature Envy:** A class which is excessively dependent on methods of another class indicating high coupling.



Chapter 2

At the method/function level:

- **Long method:** A method or function which has grown too big and complex.
- **Parameter creep:** Too many parameters for a function or method. This makes the callability and testability of the function difficult.
- **Cyclomatic complexity:** A function or method with too many branches or loops, which creates a convoluted logic that is difficult to follow, and can cause subtle bugs. Such a function should be refactored and broken down to multiple functions, or the logic rewritten to avoid too much branching.
- **Overly long or short identifiers:** A function which uses either overly long or overly short variable names such that their purpose is not clear from their names. The same is applicable to the function name as well.

A related antipattern to code smell is design smell, which are the surface symptoms in the design of a system that indicate underlying deeper problems in the architecture.

Pillai, Anand Balachandran - Sofi x

File | C:/Users/Neelam%20Pirbhai-Jetha/Desktop/jan2021/ESSEX%20ONLINE/MODULES/MODULE%203-Secure%20Software%20Dev... Not syncing

107 of 557

Summary

In this chapter, we looked at the architectural quality attribute of modifiability, and its various aspects. We discussed readability in some detail, including the readability antipatterns along with a few coding antipatterns.

We looked at various techniques for improving readability of code and understood the different aspects of commenting of code such as function, class and module docstrings. We also looked at PEP-8, the official coding convention guideline for Python.

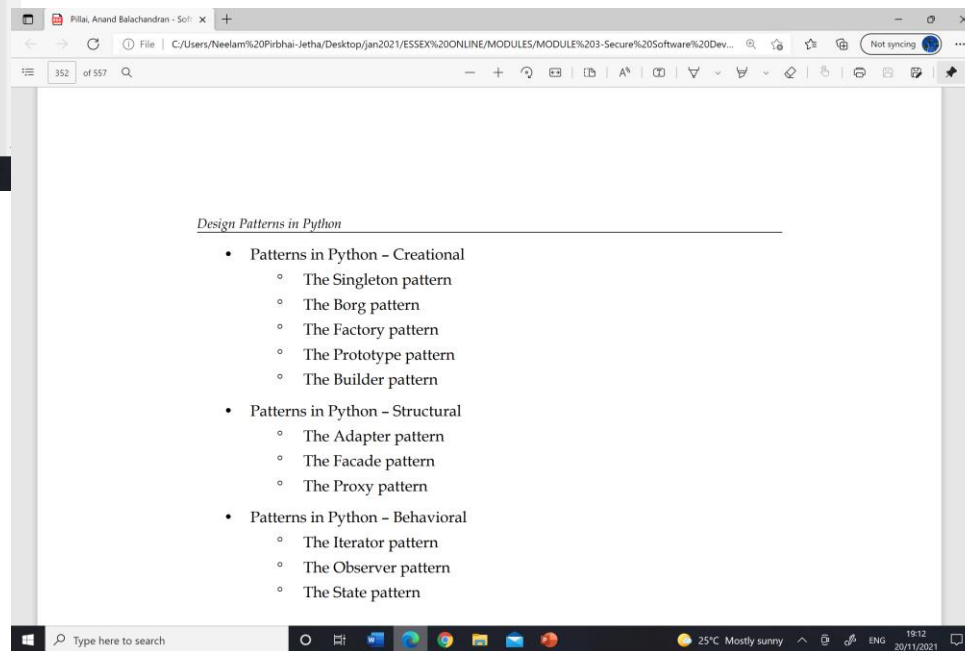
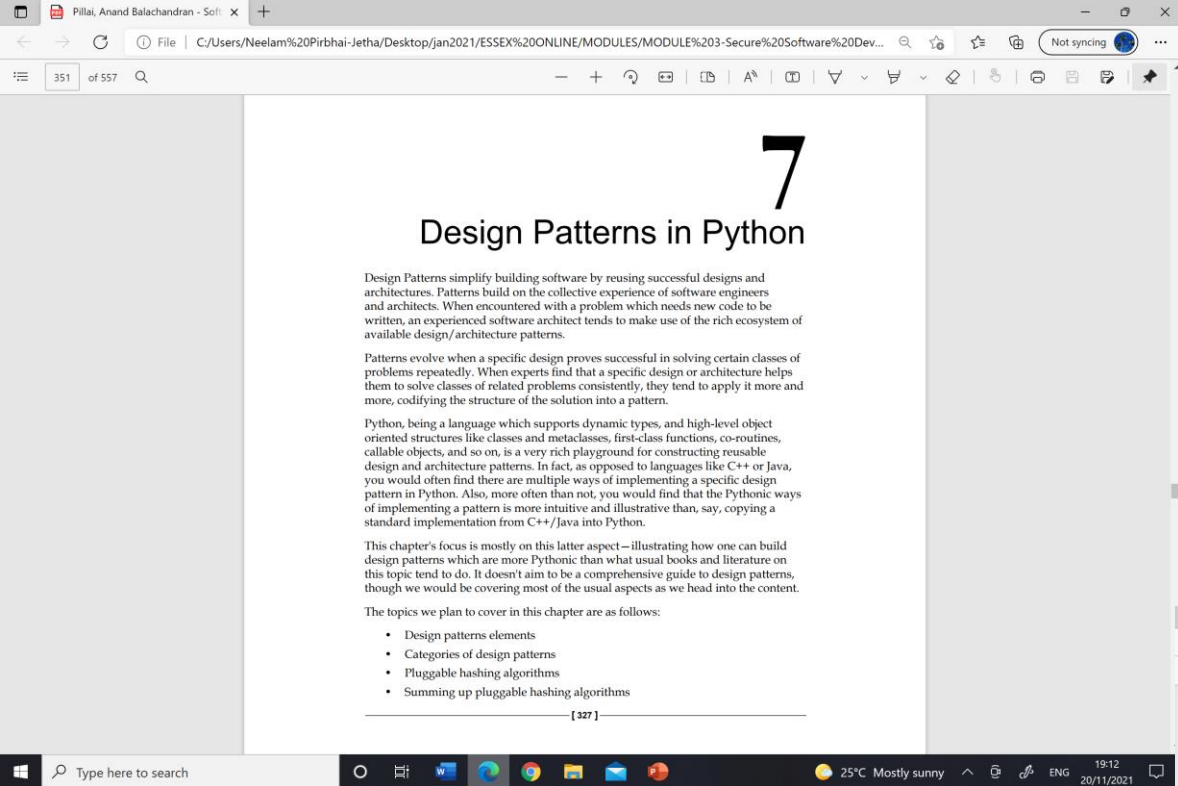
We then looked at some rules of thumb for code comments, and went on to discuss the fundamentals of modifiability, namely, coupling and cohesion of code. We looked at different cases of coupling and cohesion with a few examples. We then went on to discuss the strategies of improving modifiability of code such as providing explicit interfaces or APIs, avoiding two-way dependencies, abstracting common services to helper modules, and using inheritance techniques. We looked at an example where we refactored a class hierarchy via inheritance to abstract away common code and to improve the modifiability of the system.

Towards the end, we listed the different tools providing static code metrics in Python such as PyLint, Flake8, PyFlakes, and others. We learned about McCabe Cyclomatic complexity with the help of a few examples. We also learned what code smells are, and performed a refactoring exercise to improve the quality of the piece of code in stages.

In the next chapter, we'll discuss another important quality attribute of software architecture, namely, Testability.

Type here to search

27°C Light rain ENG 18:19 20/11/2021



Software Security and Secure Coding

Software security and secure coding has assumed more importance than ever due to the unprecedented amounts of data being shared across software and hardware systems—the explosion of smart personal technologies such as smart phones, smart watches, smart music players, and other smart systems has aided this immense traffic of data across the Internet in a big way. With the advent of IPV6 and expected large scale adoption of IoT devices (Internet of Things) in the next few years, the amount of data is only going to increase exponentially. P. 281