A/L ICT Teacher Training 2023

Project Management / Programming



Pasindu Marasinghe
ppm@ucsc.cmb.ac.lk

Who am I?

- Pasindu Marasinghe – Lecturer UCSC
- ppm@ucsc.cmb.ac.lk
- Data Structures and Algorithms, Data Analytics, Machine Learning, Digital Signal Processing

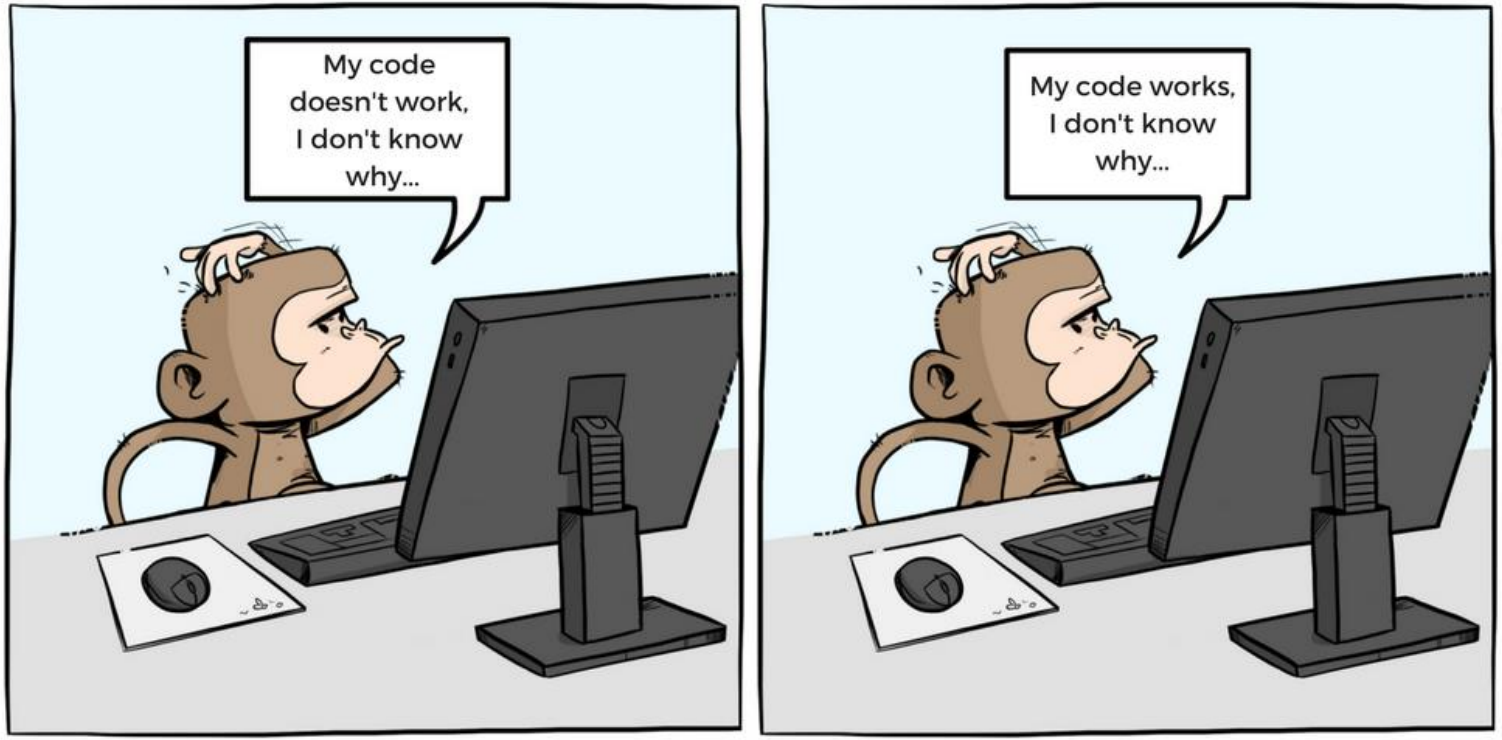
- Note: Not an A/L ICT Paper Setter

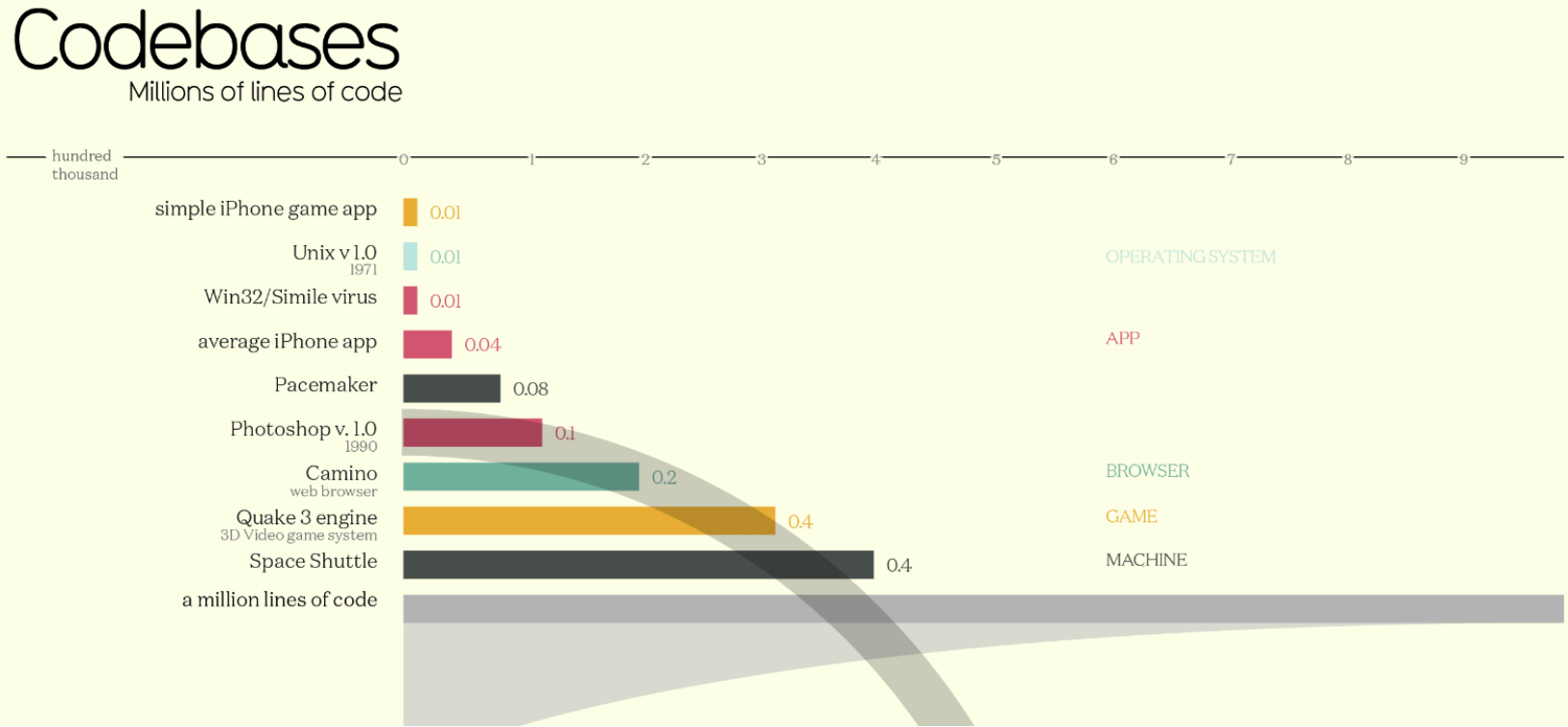


bit.ly/TT23-D3

Projects?

Why Can’t We Just Code?





https://www.informationisbeautiful.net/visualizations/million-lines-of-code/

Why “Engineer” Software?

- ‘Just coding’ does NOT scale up
- Longer time
- Might not do what users want
- Tougher to debug
- Murphy’s Law

Anything that can go wrong will go wrong

Software Project

Note: “just sitting and coding” and “hope it works” is NOT doing a Software Project!



How Is It Different from Other Projects?

- **Unconstrained** by physical laws
- **Many ways** to achieve same aim
- **Complexity**
- Software **evolves**
- **Reliability** and **stability**

Question: How many ways could you sort a pile of student tests by their last name?

Small Case Study

You are asked to build a bridge across a river, that is the only information you are given. You can probably assume most things;

- The bridge needs to be stable enough that many people/cars can cross safely.
- Must hold up in the weather. Build it using durable materials.

Small Case Study cont.

Another example. You are asked to make a cool app that helps teach UCSC students how to program in JavaScript.

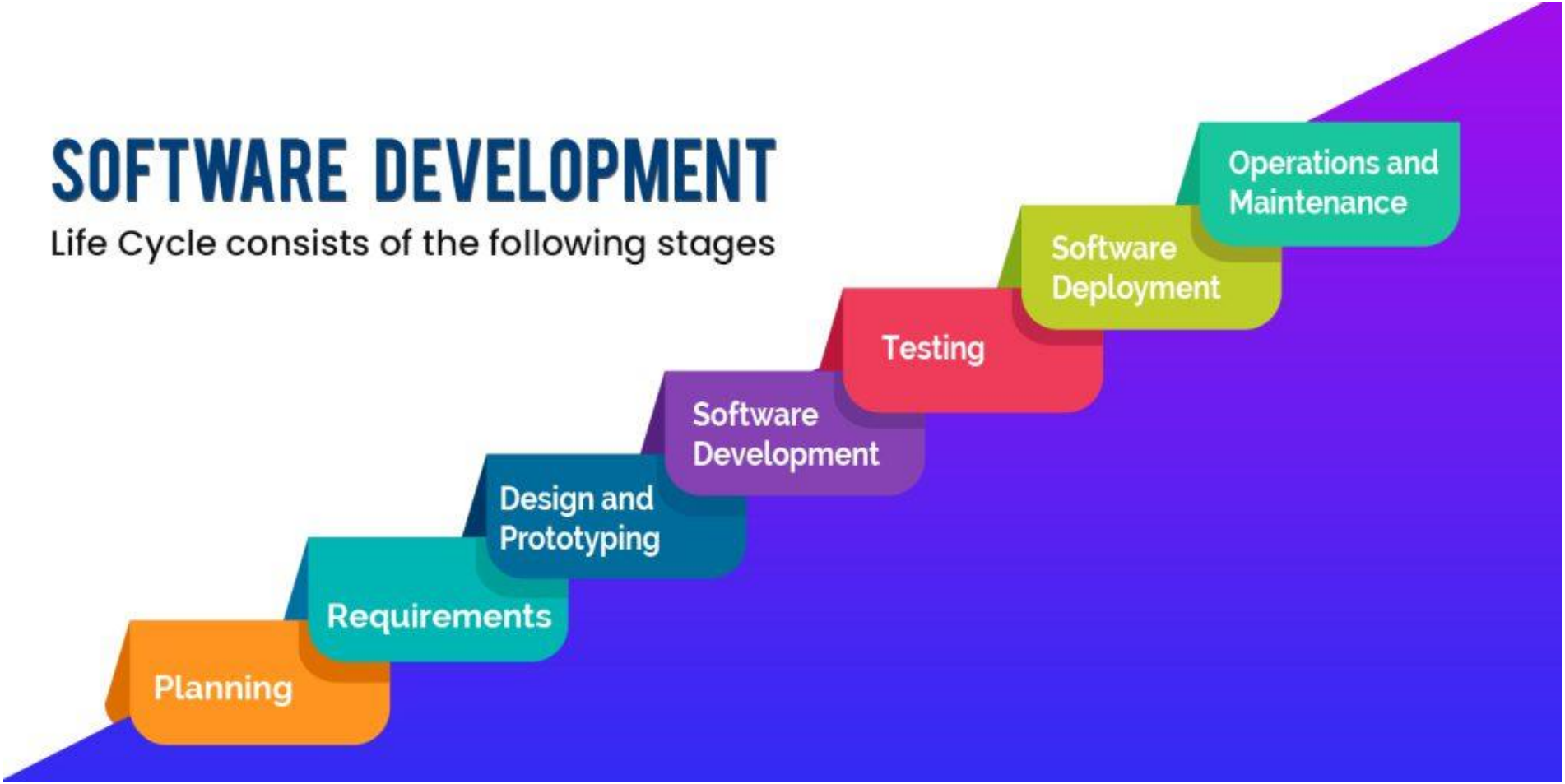
- What kinds of things need to be taught?
- How in-depth should the teaching go?
- What devices can they access the app on?
- Many more questions to ask.

Similar request in terms of how much detail we asked for, but the nature of Software Projects makes it more difficult to design, more info is needed.

Do We Start with SDLC Model?

SOFTWARE DEVELOPMENT

Life Cycle consists of the following stages



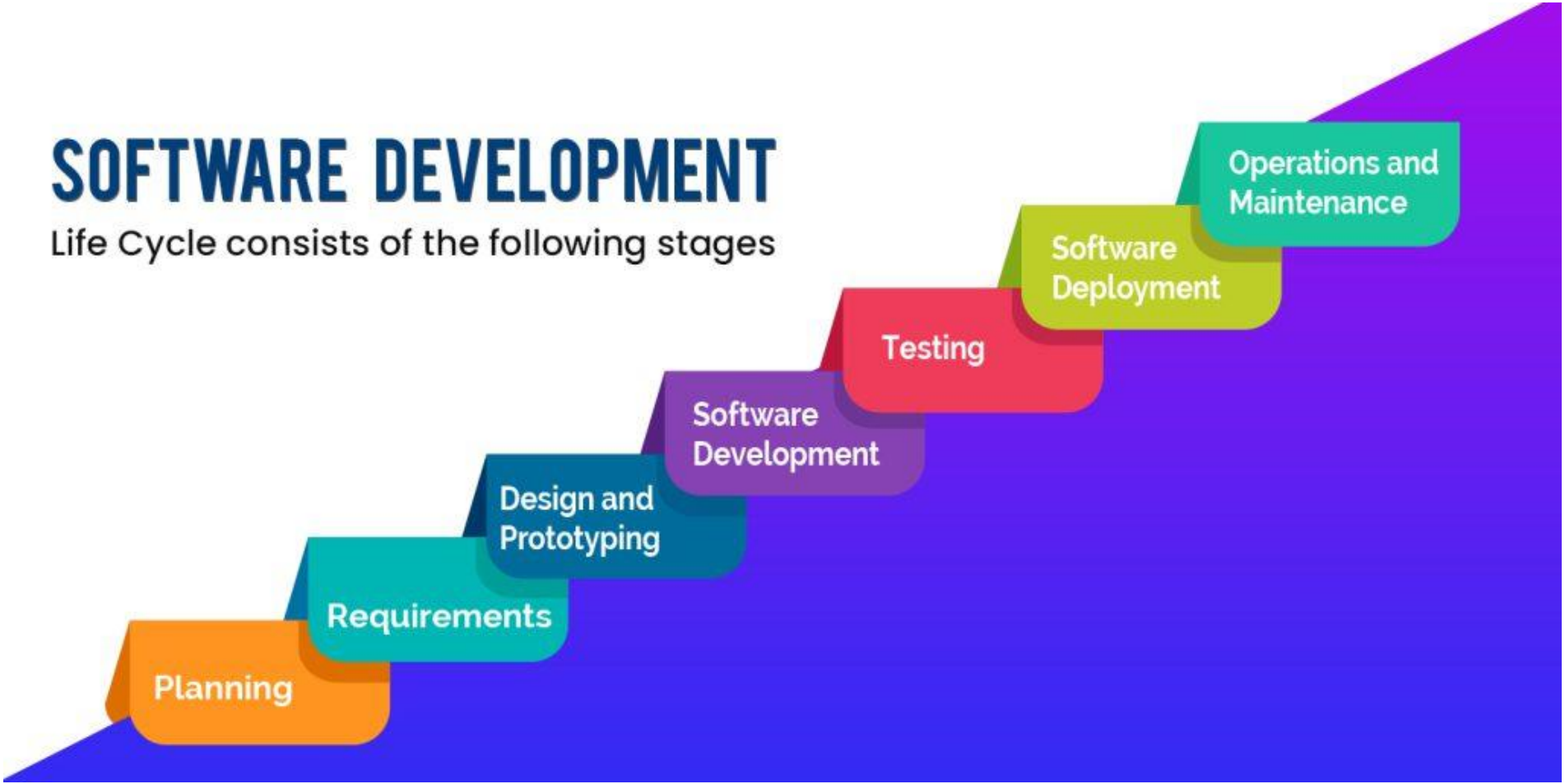
Project Planning

- Pre-Study
- Feasibility Study
- Project Management Plan



SOFTWARE DEVELOPMENT

Life Cycle consists of the following stages

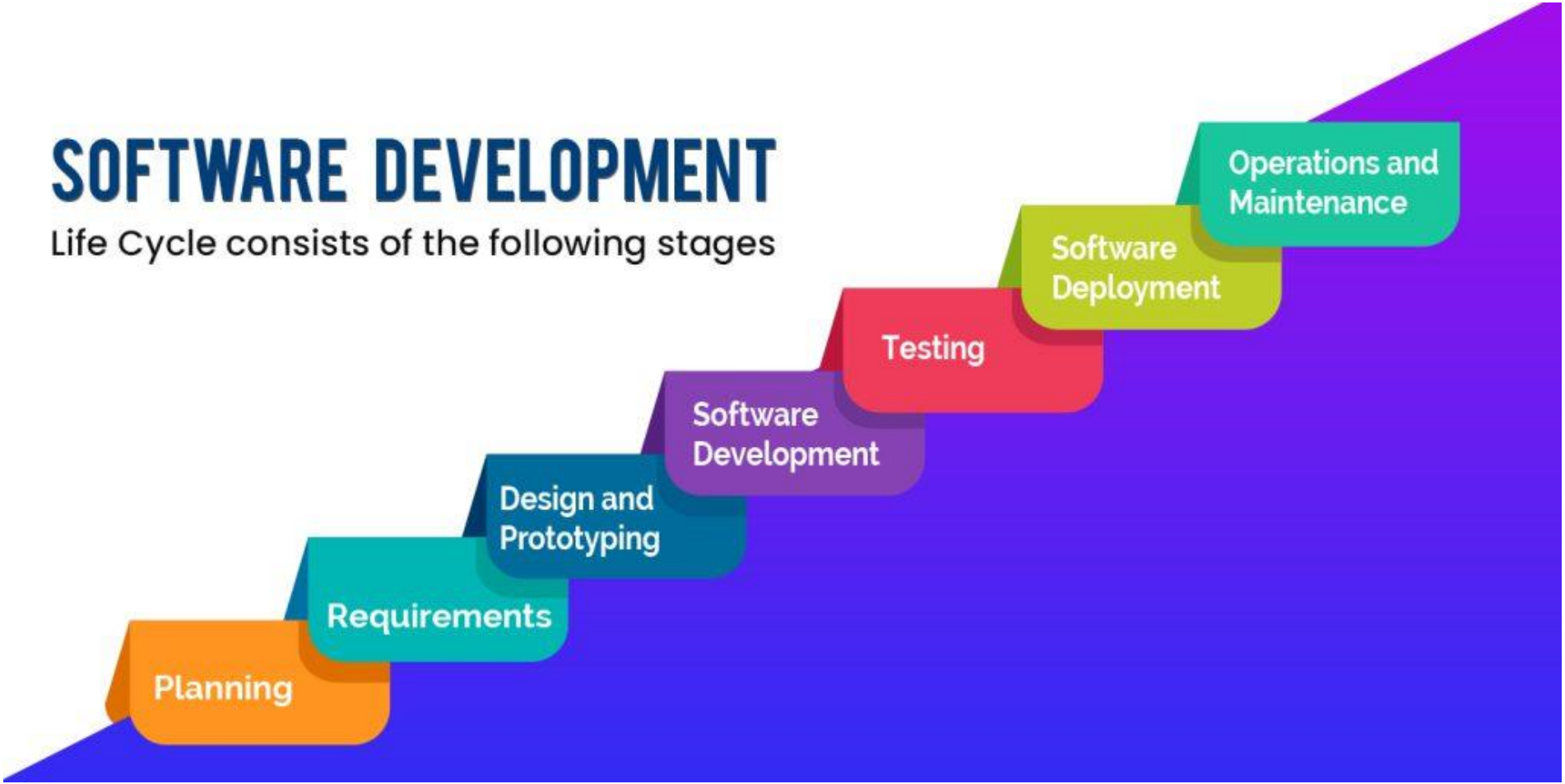


Requirements Gathering Techniques

- Interviews
- Questionnaires or Surveys
- User Observation
- Document Analysis
- Interface Analysis
- Workshops
- Brainstorming
- Role-Play
- Use Cases and Scenarios
- Focus Groups
- Prototyping

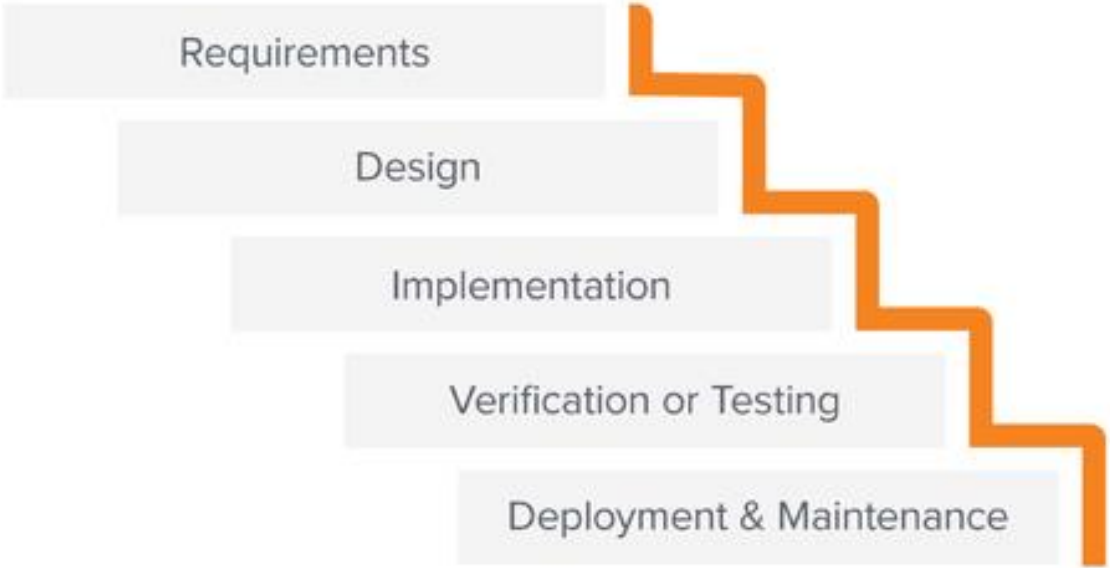
SOFTWARE DEVELOPMENT

Life Cycle consists of the following stages



SDLC Methodologies

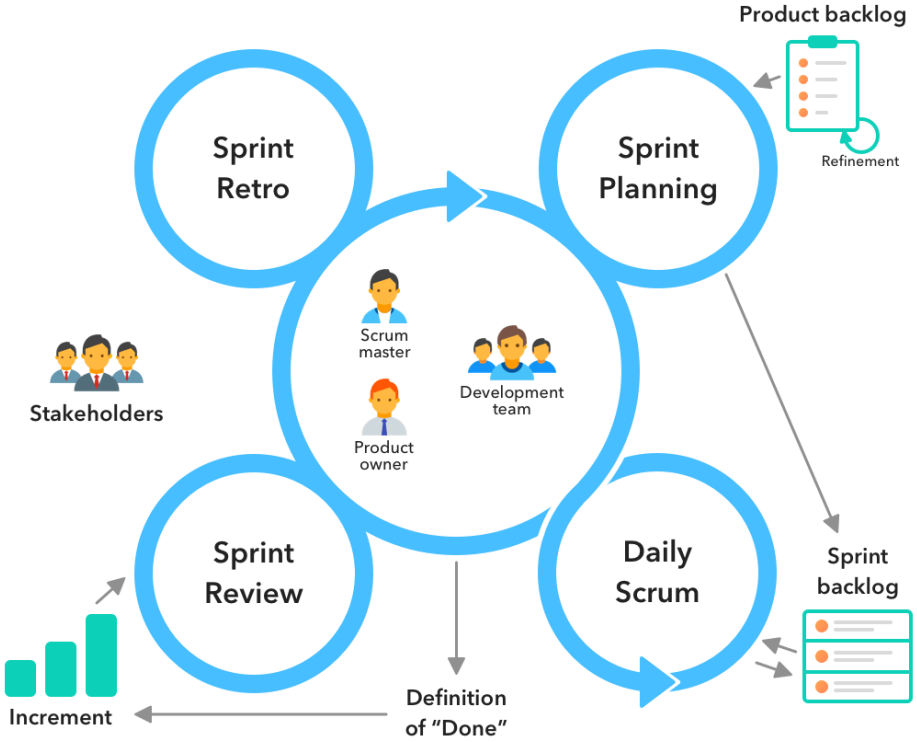
Waterfall


Requirements
Design
Implementation
Verification or Testing
Deployment & Maintenance

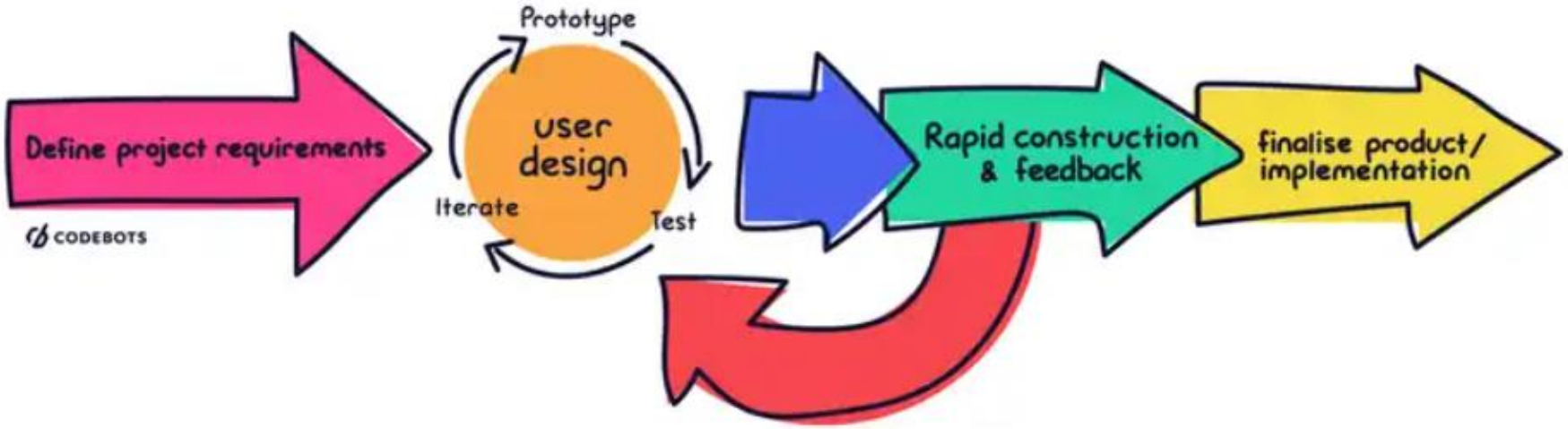
Spiral


Plan
Requirements Gathering
Risk Analysis
Risk Reduction Prototyping
Start
01
02
03
04
Release
Evaluate
Customer Evolution
Engineering
Coding Testing

Agile

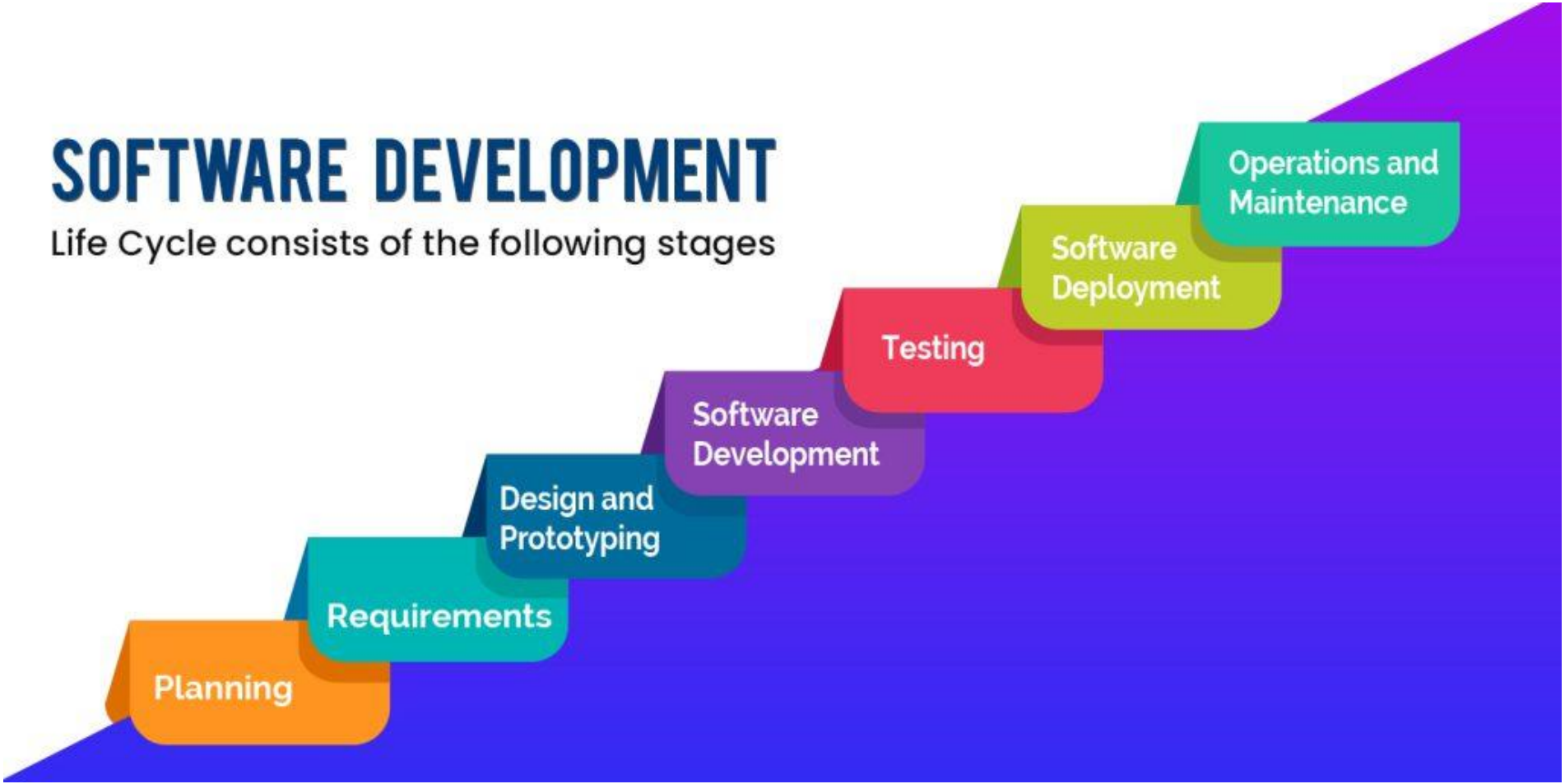

Product backlog
Refinement
Sprint Retro
Sprint Planning
Scrum master
Development team
Stakeholders
Product owner
Sprint Review
Daily Scrum
Sprint backlog
Increment
Definition of "Done"

Rapid Application Development



SOFTWARE DEVELOPMENT

Life Cycle consists of the following stages



Tests must be…

- Tests must be documented.
- Tests must be repeatable – Why?
- Tests must feature both EXPECTED and ACTUAL outcomes.

Test case

- A test case is a set of input condition (values) and expected output.
  - Ex: A program that takes 2 integers and prints their sum

Input		Expected Output
Number1	Number2	Sum
3	7	10
-5	5	0
-10	-6	-16
x	5	Not a valid number

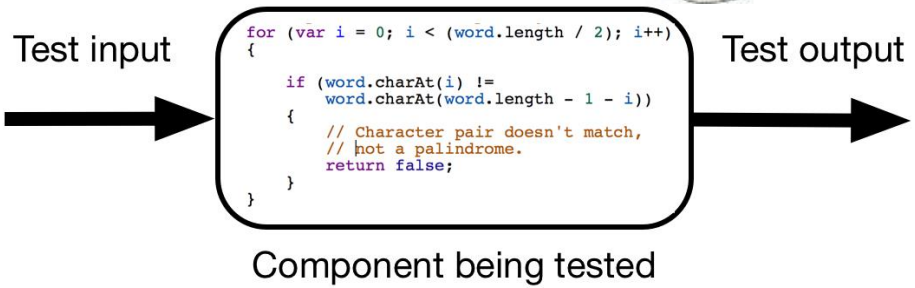
- A good test case is one that has a high probability of finding an as-yet undiscovered error
- A successful test is one that uncovers an as-yet undiscovered error”

Testing Approaches

White-Box Testing

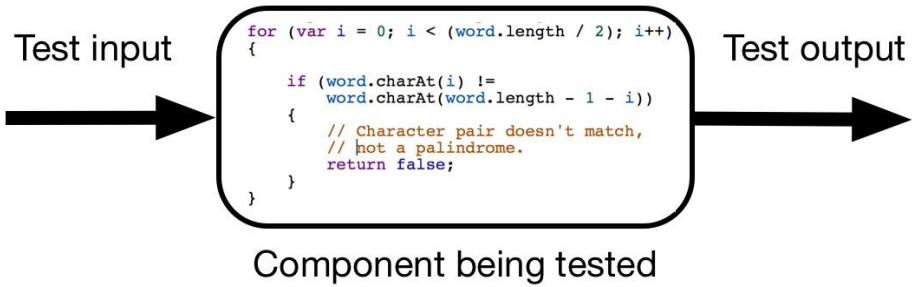
- White box testing (structural testing, program-based testing)
  - Here we test by looking through the code directly





White-Box Testing

- White box testing (structural testing, program-based testing)
  - Assumes that the program is available
    - Derives test cases from the program
    - Controls and observes the internal structure of the program
- Limitations
  - Time consuming
  - Cannot catch omission errors
    - Which requirements are missing in the program?

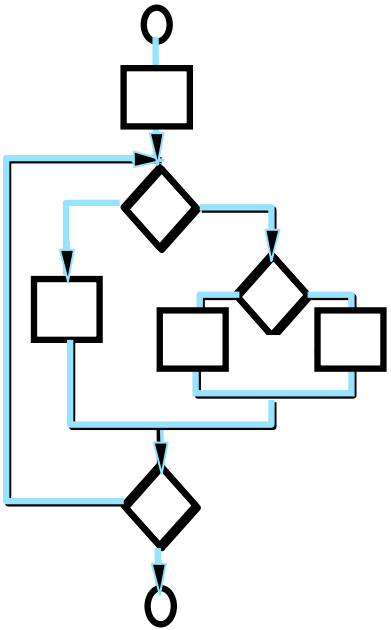


White-Box Testing

- Advantages
  - We can ensure ALL the code is tested if we spend enough time on it.
  - Is often done while development is still happening, so testing is done incrementally.

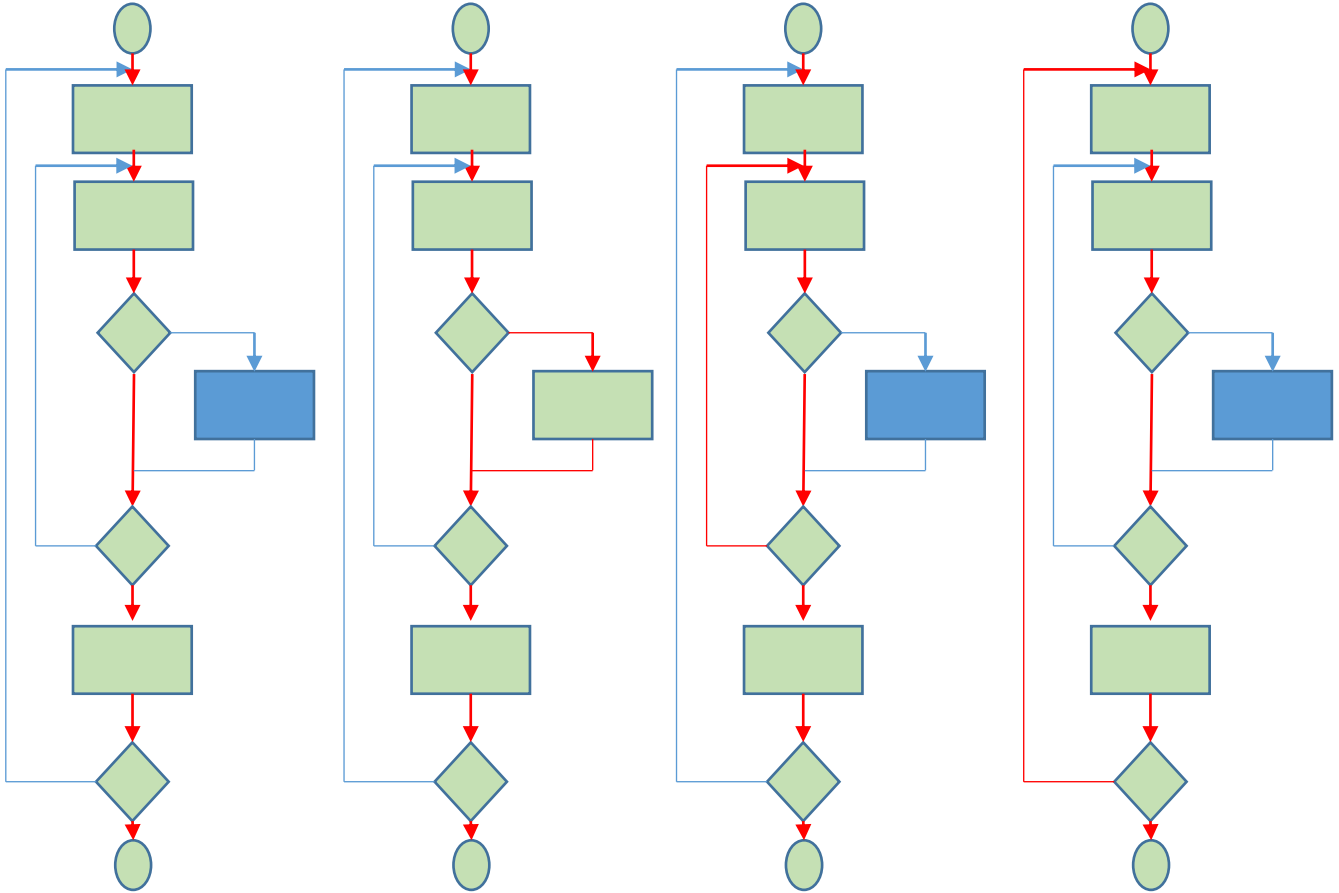
- Disadvantages
  - The tester may not test things that they think won’t occur.
  - Requires someone who understands code.

Basis Path Testing



Testing the different paths that the code could take.

We want to aim to check ALL the code.



Black-Box Testing

- Black box testing (functional testing, requirements-based testing)
  - We can't see the inside
  - Try some cases
  - See what happens
  - Cases depend on what it's supposed to do





Black-Box Testing

- Black box testing (functional testing, specification/requirements-based testing)
  - Assumes that the program is unavailable or testers do not want to look at the details of the program
    - Derives test cases from the requirements of the program
    - Controls and observes the program only through external interfaces
    - Testing without having an insight into the details of underlying code



Black-Box Testing

- Advantages
  - No need for source code
  - Can be used for unit as well as system level testing
- Disadvantages
  - May get correct output by chance
  - When does testing stop?

Example – Palindrome Function

- I have a function called “Palindrome”.
  - The function should take a word, and return a true if it is a palindrome, or a false if it isn’t.

- Testing the check Palindrome function
  - What cases should I try?
  - Remember that they just need to check if something does or doesn’t work.

Example

- Testing the check Palindrome function
  - Cases to confirm the last and first characters are checked.
  - Does it work with even and odd numbered strings?
  - Does it still work with different upper/lower cases?
  - Wrong types?
  - Symbols?
  - Etc.

Who Tests the Software?

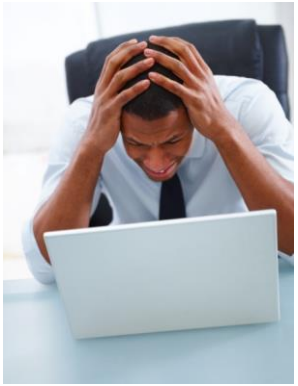


developer

Understands the system

but, will test "gently"

and, is driven by "delivery"



independent tester

Must learn about the system,

but, will attempt to break it

and, is driven by quality

Methods of Testing

Methods of Testing

- Static
  - Testing which is done on code that is NOT running.
  - These tests are more about understanding the code and trying to find errors before they occur.
- Dynamic
  - Testing which is done on running code.
  - These tests are more about testing that the code is working properly in a broader sense.

- Both categories have their own testing strategies.

Static Strategies

- Code Review
  - Another person looks through the code and tries to find errors. Other people are more likely to spot errors that you missed.
- Walkthroughs
  - A larger group activity where a designer or programmer leads team members through the software where participants are encouraged to question and comment on the code and possible errors.
- Static Analysis
  - A test which involves automatic analysis of the code usually performed continuously by your IDE (Brackets).

Dynamic Strategies

- 3 main levels of testing
  - Unit Testing
  - Integration Testing
  - System Testing
- Other dynamic tests
  - Coverage Testing
  - Regression Testing
  - User Acceptance Testing

Unit Testing

- Unit tests stress **small portions**, or units, of code. Each unit test generally tests a function, or a method of a class in the case of object-oriented code. Such tests are usually written by the programmer while writing that portion of the code base.

- We've just been doing this with the selection sort and palindrome functions

Integration Testing Strategies

- Sometimes bugs appear when functions and methods interact with each other
- Often these are issues of functions not calling on each other appropriately
- Typically this is done after unit testing

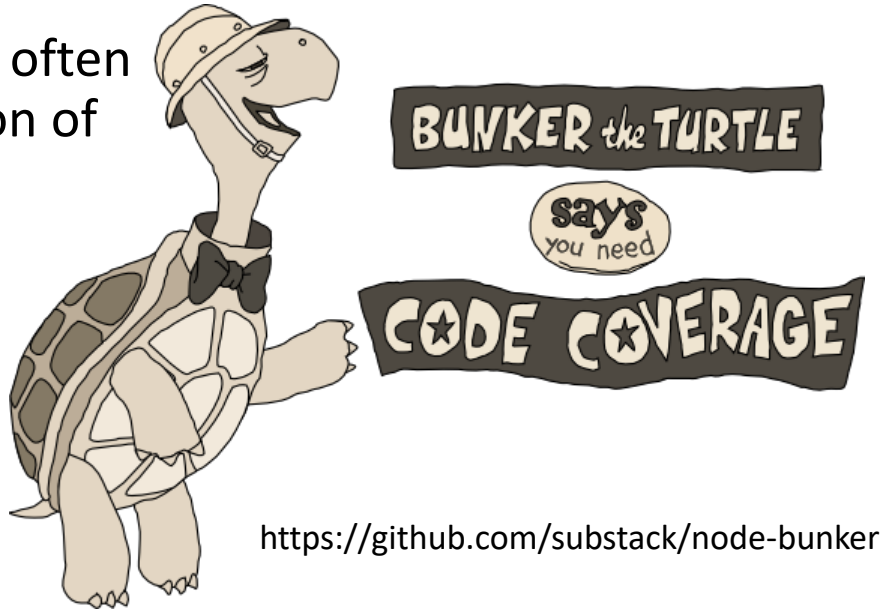


System Testing

- A series of tests to verify that all system elements have been properly integrated.
- Recovery Testing:
  - Does it act appropriately on crashing (think of MS word)
- Security Testing:
  - Is it sufficiently hard to crack?
- Stress Testing:
  - How does the system handle crazy resource use
- Performance Testing:
  - How well does it perform in the expected setup

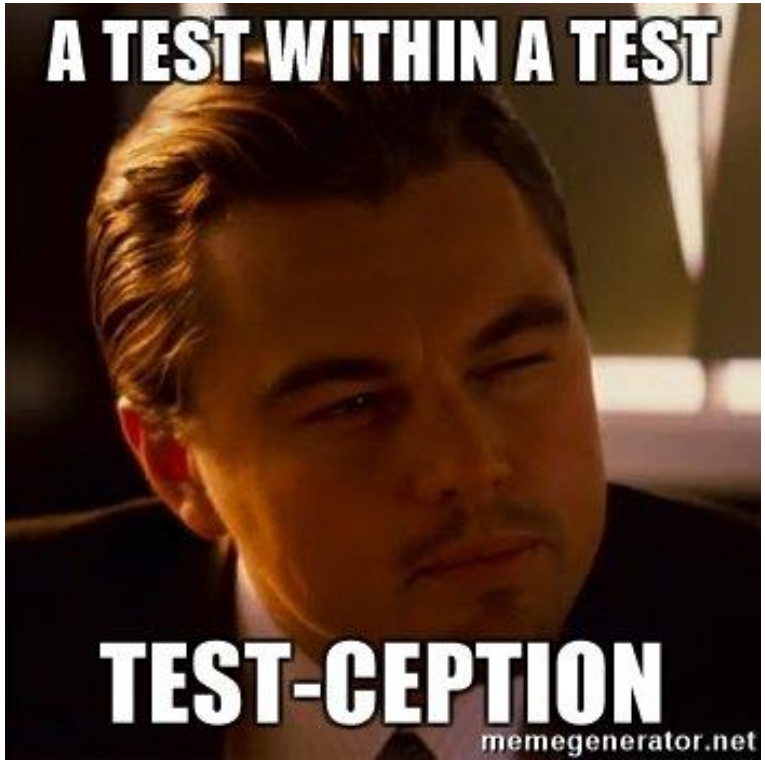
Coverage Testing

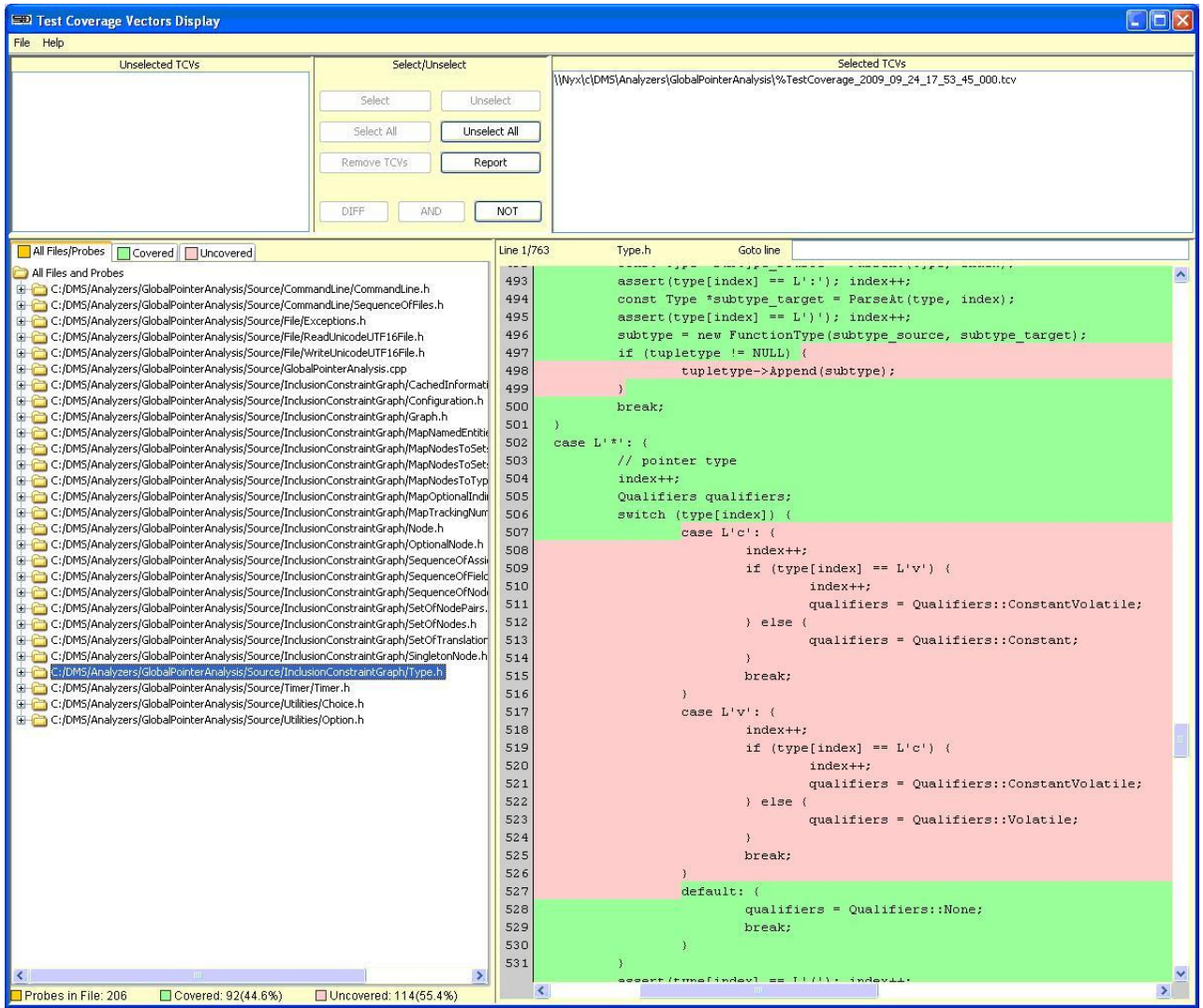
- There are special tools for this
  - They figure out how much of your code is tested by your test cases
  - Aim high! 100% is fantastic!
- Coverage testing is often done via a collection of Unit Tests.



https://github.com/substack/node-bunker

Coverage Testing




Test Coverage Vectors Display
File Help
Unselected TCVs
Select/Unselect
Select
Unselect
Select All
Unselect All
Remove TCVs
Report
DIFF
AND
NOT
Selected TCVs
\\Nyx\c\DMS\Analyzers\GlobalPointerAnalysis\%TestCoverage_2009_09_24_17_53_45_000.tcv
All Files/Probes
Covered
Uncovered
All Files and Probes
C:/DMS/Analyzers/GlobalPointerAnalysis/Source/CommandLine/CommandLine.h
C:/DMS/Analyzers/GlobalPointerAnalysis/Source/CommandLine/SequenceOfFiles.h
C:/DMS/Analyzers/GlobalPointerAnalysis/Source/File/Exceptions.h
C:/DMS/Analyzers/GlobalPointerAnalysis/Source/File/ReadUnicodeUTF16File.h
C:/DMS/Analyzers/GlobalPointerAnalysis/Source/File/WriteUnicodeUTF16File.h
C:/DMS/Analyzers/GlobalPointerAnalysis/Source/GlobalPointerAnalysis.cpp
C:/DMS/Analyzers/GlobalPointerAnalysis/Source/InclusionConstraintGraph/Configuration.h
C:/DMS/Analyzers/GlobalPointerAnalysis/Source/InclusionConstraintGraph/Graph.h
C:/DMS/Analyzers/GlobalPointerAnalysis/Source/InclusionConstraintGraph/Node.h
C:/DMS/Analyzers/GlobalPointerAnalysis/Source/InclusionConstraintGraph/OptionalNode.h
C:/DMS/Analyzers/GlobalPointerAnalysis/Source/InclusionConstraintGraph/SetOfNodes.h
C:/DMS/Analyzers/GlobalPointerAnalysis/Source/InclusionConstraintGraph/SingletonNode.h
C:/DMS/Analyzers/GlobalPointerAnalysis/Source/InclusionConstraintGraph/Type.h
C:/DMS/Analyzers/GlobalPointerAnalysis/Source/Timer/Timer.h
C:/DMS/Analyzers/GlobalPointerAnalysis/Source/Utilities/Choice.h
C:/DMS/Analyzers/GlobalPointerAnalysis/Source/Utilities/Option.h
Probes in File: 206
Covered: 92(44.6%)
Uncovered: 114(55.4%)
Line 1/763
Type.h
Goto line
493 assert(type[index] == L':'); index++;
494 const Type *subtype_target = ParseAt(type, index);
495 assert(type[index] == L')'); index++;
496 subtype = new FunctionType(subtype_source, subtype_target);
497 if (tupletype != NULL) {
498 tupletype->Append(subtype);
499 }
500 break;
501 }
502 case L'*': {
503 // pointer type
504 index++;
505 Qualifiers qualifiers;
506 switch (type[index]) {
507 case L'c': {
508 index++;
509 if (type[index] == L'v') {
510 index++;
511 qualifiers = Qualifiers::ConstantVolatile;
512 } else {
513 qualifiers = Qualifiers::Constant;
514 }
515 break;
516 }
517 case L'v': {
518 index++;
519 if (type[index] == L'c') {
520 index++;
521 qualifiers = Qualifiers::ConstantVolatile;
522 } else {
523 qualifiers = Qualifiers::Volatile;
524 }
525 break;
526 }
527 default: {
528 qualifiers = Qualifiers::None;
529 break;
530 }
531 }

Regression Testing

- Regression Testing
  - Re-execution of some subset of tests already conducted to ensure that the new changes do not have unintended side effects.
- The Regression test suite should contain three different classes of test cases :
  - A representative sample of tests that will exercise all software functions
  - Additional tests that focus on functions that are likely to be affected by the change.
  - Tests that focus on software components that have changed.

User Acceptance Testing

- It provides final assurance that software meets all functional, behavioral, and performance requirements.
  -- Exclusive use of Black-box testing techniques.
- After each validation test case either software conforms to specs or a deviation from specs is detected and a deficiency list needs to be worked.
- Alpha and Beta testing
  - Alpha test -- At developer's site by customer.
  - Beta test -- At customer's site in a "live" environment.

Testing Strategy

- We begin by ‘testing-in-the-small’ and move toward ‘testing-in-the-large’

- For conventional software
  - The module (component) is our initial focus
  - Integration of modules follows

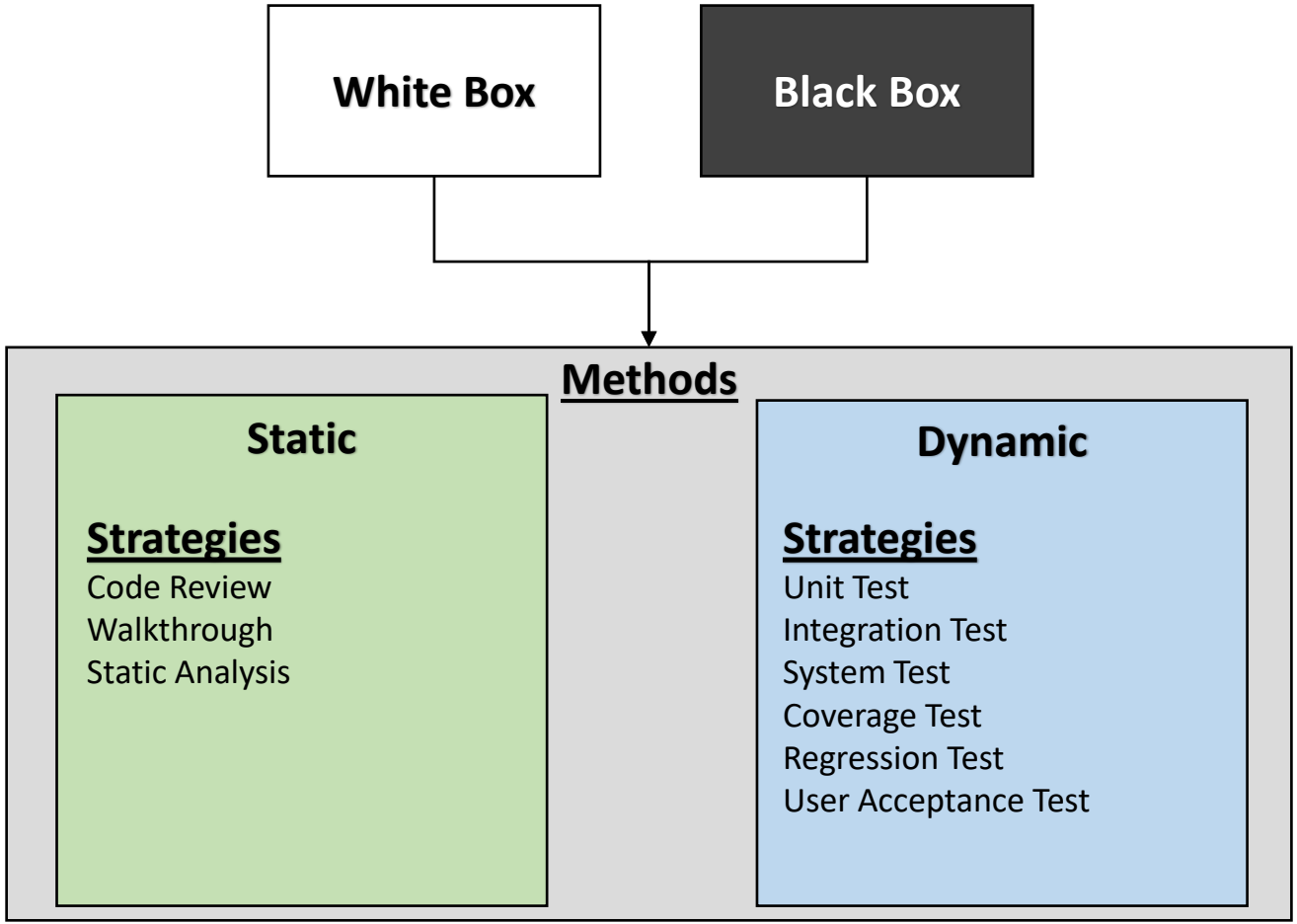
- For OO software
  - our focus when “testing in the small” changes from an individual module (the conventional view) to an OO class that encompasses attributes and operations and implies communication and collaboration

What do we use?

- Testing approaches are used with the various testing methods and strategies.
  - We could perform a black box unit test, or a white box integration test.

  - Generally the context surrounding the situation informs the **approach** and the thing that needs testing informs the **method & strategy**.
  - i.e We have hired an external tester to test the system, indicates Black box testing approach.
  - i.e I tried to implement a new function but it doesn't work, indicates Unit Testing strategy.

Testing Approaches



Learning Data Science

=

Learning Python?


MATLAB
JavaScript
R
sas
SQL
Scala
Kotlin
julia
Java

Why Python?

Python is an excellent choice for beginners in data science for several reasons:

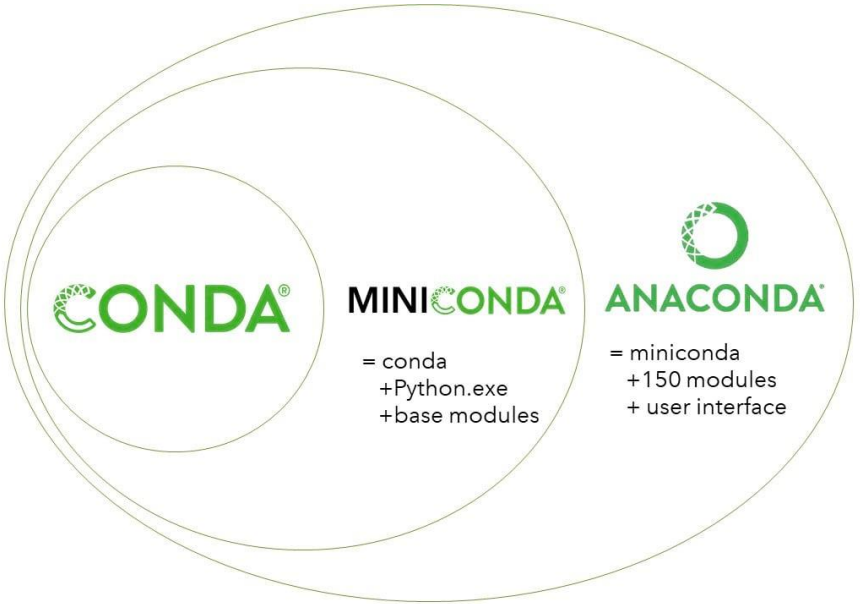
- **Easy to learn:** Python has a relatively simple and easy-to-learn syntax, making it accessible for beginners without a programming background.
- **Rich libraries and frameworks:** Python has a vast collection of libraries and frameworks designed specifically for data science, including NumPy, Pandas, Scikit-learn, Matplotlib, and many others.
- **Large community:** Python has a large and active community of developers and users, providing support, documentation, and resources for beginners in data science.

Okay !
I Visited Python Website and Installed Python. Now What?

Environment Manager

An environment manager like Conda is essential for Python development for several reasons:

- **Managing dependencies**
- **Reproducibility**
- **Flexibility**
- **Collaboration**
- **Experimentation**



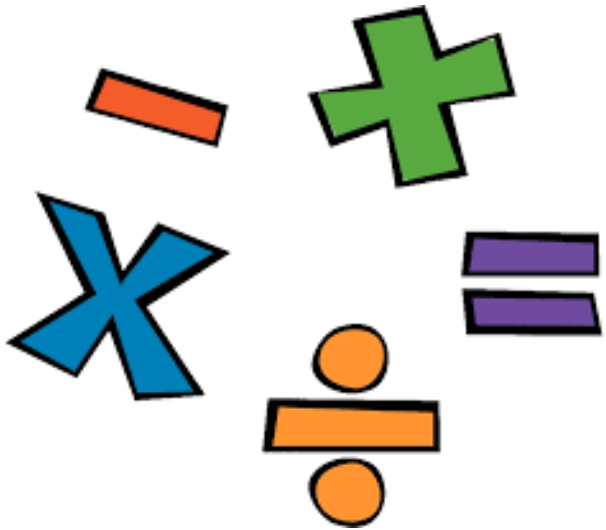
Let’s Get Back to Basic Python

Built-in Data Types

1. **Text Type:** str
2. **Numeric Types:** int, float, complex
3. **Sequence Types:** list, tuple, range
4. **Mapping Type:** dict
5. **Set Types:** set, frozenset
6. **Boolean Type:** bool
7. **Binary Types:** bytes, bytearray, memoryview
8. **None Type:** NoneType

Python Operators

1. Arithmetic operators
2. Assignment operators
3. Comparison operators
4. Logical operators
5. Identity operators
6. Membership operators
7. Bitwise operators



Python File Handling

open(filename, arguments)

- "r" - Read - Default value. Opens a file for reading, error if the file does not exist
- "a" - Append - Opens a file for appending, creates the file if it does not exist
- "w" - Write - Opens a file for writing, creates the file if it does not exist
- "x" - Create - Creates the specified file, returns an error if the file exists

- "t" - Text - Default value. Text mode
- "b" - Binary - Binary mode (e.g. images)

Python Lambda

lambda arguments : expression

- x = lambda a : a + 10

  print(x(5))

- x = lambda a, b : a * b

  print(x(5, 6))

pass by reference

```
cup =
```



```
fillCup(      )
```

*pass by **value***

```
cup =
```



```
fillCup(      )
```