Vidyavardhini's College of Engineering & Technology

Department of Computer Engineering

Academic Year : 2024-25

Experiment No. 8
Apply FP based estimation technique for a case study
Name: Krisha Chikka
Std/Div: TE/1 Roll no. : 30
Date of Performance: 23/09/2024
Date of Submission: 30/09/2024

Aim: To apply FP based estimation technique for a case study.

Objective: To analyse and apply FP based estimation technique for Currency Detector for the Visually Impaired.

Theory:

Function Point (FP) is an element of software development which helps to approximate the cost of development early in the process. It may measure functionality from user's point of view.

Objectives of FPA:

- The objective of FPA is to measure the functionality that the user requests and receives.
- The objective of FPA is to measure software development and maintenance independently of the technology used for implementation.
- It should be simple enough to minimize the overhead of the measurement process.
- It should be a consistent measure among various projects and organizations.

Types of FPA:

- **Transactional Functional Type –**
    - **External Input (EI):** EI processes data or control information that comes from outside the application's boundary. The EI is an elementary process.
    - **External Output (EO):** EO is an elementary process that generates data or control information sent outside the application's boundary.
    - **External Inquiries (EQ):** EQ is an elementary process made up of an input-output combination that results in data retrieval.
- **Data Functional Type –**
    - **Internal Logical File (ILF):** A user identifiable group of logically related data or control information maintained within the boundary of the application.
    - **External Interface File (EIF):** A group of users recognizable logically related data allusion to the software but maintained within the boundary of another software.

STEPS:

- **Step-1:**
  F = 14 * scale

  Scale varies from 0 to 5 according to character of Complexity Adjustment Factor (CAF). Below table shows scale:

  0 - No Influence

  1 - Incidental

  2 - Moderate

  3 - Average

  4 - Significant

  5 - Essential

- **Step-2:** Calculate Complexity Adjustment Factor (CAF).
  CAF = 0.65 + (0.01 * F)
- **Step-3:** Calculate Unadjusted Function Point (UFP).
  Multiply each individual function point to corresponding values in TABLE.
- **Step-4:** Calculate Function Point.
  FP = UFP * CAF

Example:

Given the following values, compute function point when all complexity adjustment factor (CAF) and weighting factors are average.

User Input = 50

User Output = 40

User Inquiries = 35

User Files = 6

External Interface = 4

Explanation:

- **Step-1:** As complexity adjustment factor is average (given in question), hence,
- scale = 3.

  F = 14 * 3 = 42

- **Step-2:**
  CAF = 0.65 + (0.01 * 42) = 1.07

- **Step-3:** As weighting factors are also average (given in question) hence we will multiply each individual function point to corresponding values in TABLE.
  UFP = (50*4) + (40*5) + (35*4) + (6*10) + (4*7) = 628
- **Step-4:**
  Function Point = 628 * 1.07 = **671.96**

Solution:

Code:

```
#include <iostream>


class FunctionPoint {

private:

    int inputs;

    int outputs;

    int inquiries;

    int internalFiles;

    int externalInterfaces;


public:

    FunctionPoint()

        : inputs(0), outputs(0), inquiries(0),

          internalFiles(0), externalInterfaces(0) {}


    void addInputs(int count) {

        inputs += count;

    }

    void addOutputs(int count) {

        outputs += count;
```

```
    }

    void addInquiries(int count) {
        inquiries += count;
    }
    void addInternalFiles(int count) {
        internalFiles += count;
    }
    void addExternalInterfaces(int count) {
        externalInterfaces += count;
    }

    int calculateFunctionPoints() {
        // Weighting factors for different components (simplified)
        const int weights[5] = {4, 5, 4, 7, 5};

        return (inputs * weights[0] +
                outputs * weights[1] +
                inquiries * weights[2] +
                internalFiles * weights[3] +
                externalInterfaces * weights[4]);
    }
};

int main() {
```

```
    FunctionPoint fp;


    std::cout << "Enter counts for the following function types:" <<
std::endl;


    int count;


    std::cout << "Number of User Inputs: ";
    std::cin >> count;
    fp.addInputs(count);


    std::cout << "Number of User Outputs: ";
    std::cin >> count;
    fp.addOutputs(count);


    std::cout << "Number of User Inquiries: ";
    std::cin >> count;
    fp.addInquiries(count);


    std::cout << "Number of Internal Logical Files: ";
    std::cin >> count;
    fp.addInternalFiles(count);


    std::cout << "Number of External Interface Files: ";
    std::cin >> count;
```

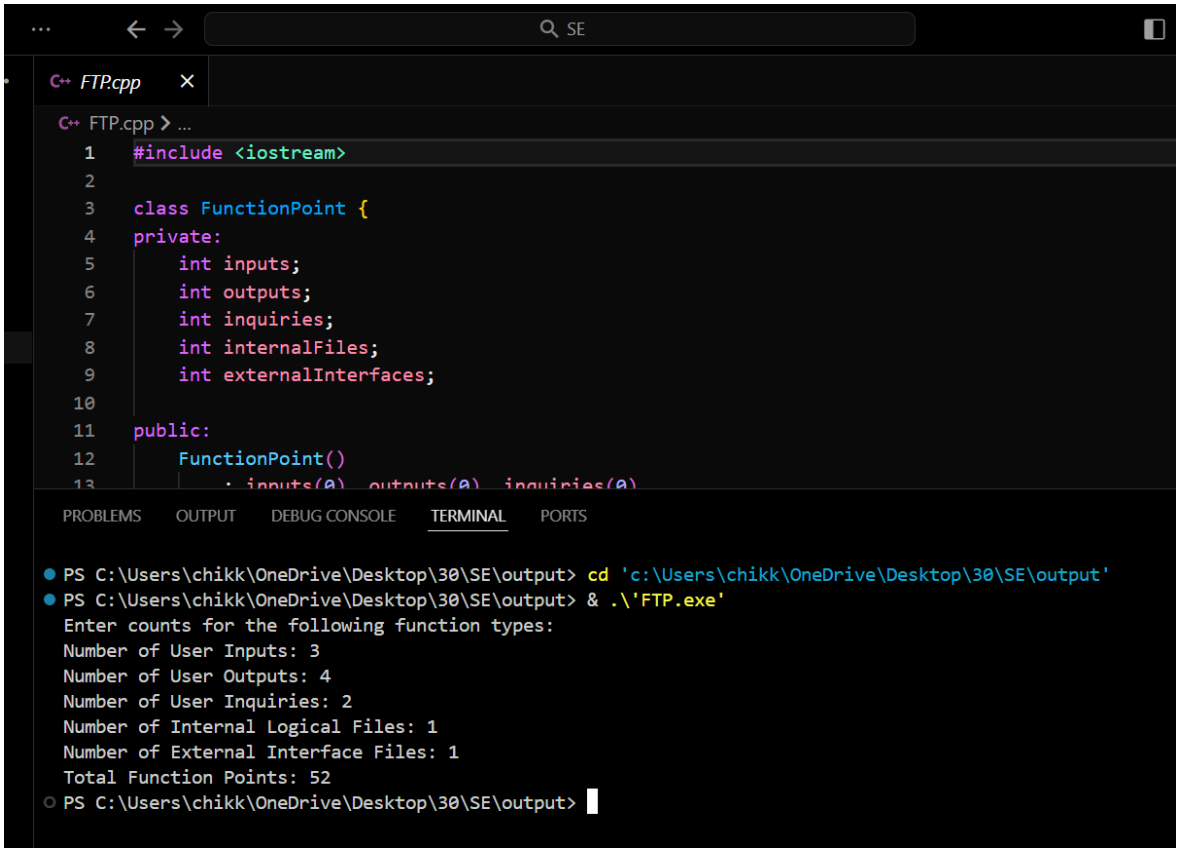
```
    fp.addExternalInterfaces(count);


    int totalFP = fp.calculateFunctionPoints();


    std::cout << "Total Function Points: " << totalFP << std::endl;


    return 0;
}
```

Output:



Conclusion: In this experiment, we successfully applied the Function Point (FP) estimation technique to the Currency Detector for the Visually Impaired. By evaluating various functional components such as user inputs, outputs, inquiries, internal files, and external interfaces, we calculated the unadjusted function points and applied the Complexity Adjustment Factor (CAF). This method provided a reliable estimation of the software size and development effort, showcasing the effectiveness of FP analysis in early-stage project planning and cost estimation.