Department of Computer

Engineering Academic Term: First

Term 2023-24

Class: T.E /Computer Sem – V / Software Engineering

Practical No:	4
Title:	Calculating Function Points of the Project in Software Engineering
Date of Performance:	24-08-23
Roll No:	9595
Team Members:	Atharva Dalvi

Rubrics for Evaluation:

Sr. No	Performance Indicator	Excellent	Good	Below Average	Total Score
1	On time Completion & Submission (01)	01 (On Time)	NA	00 (Not on Time)	
2	Theory Understanding(02)	02(Correct)	NA	01 (Tried)	
3	Content Quality (03)	03(All used)	02 (Partial)	01 (rarely followed)	
4	Post Lab Questions (04)	04(done well)	3 (Partially Correct)	2(submitted)	

Signature of the Teacher:

Department of Computer

Engineering Academic Term: First

Term 2022-23

Class: T.E /Computer Sem – V / Software Engineering

Signature of the Teacher:

Lab Experiment 04

Experiment Name: Calculating Function Points of the Project in Software Engineering

Objective: The objective of this lab experiment is to introduce students to the concept of Function Points and the Function Point Analysis (FPA) technique for measuring software size and complexity. Students will gain practical experience in calculating Function Points for a sample software project, enabling them to estimate development effort and assess project scope accurately.

Introduction: Function Points are a unit of measurement used in software engineering to quantify the functionality delivered by a software application. Function Point Analysis (FPA) is a widely used technique to assess the functional size of a software project based on its user requirements.

Lab Experiment Overview:

1. Introduction to Function Points: The lab session begins with an overview of Function Points, explaining the concept and their significance in software size measurement.
2. Defining the Sample Project: Students are provided with a sample software project along with its
user requirements. The project may involve modules, functionalities, and user interactions.
3. Identifying Functionalities: Students identify and categorize the functionalities in the sample project, such as data inputs, data outputs, inquiries, external interfaces, and internal logical files.
4. Assigning Complexity Weights: For each identified functionality, students assign complexity weights based on specific criteria provided in the FPA guidelines.
5. Calculating Unadjusted Function Points: Students calculate the Unadjusted Function Points (UFP) by summing up the weighted functionalities.
6. Adjusting Function Points: Students apply adjustment factors (e.g., complexity, performance, and
conversion) to the UFP to calculate the Adjusted Function Points (AFP).
7. Estimating Development Effort: Using historical data or industry benchmarks, students estimate
the development effort required for the project based on the calculated AFP.
8. Conclusion and Reflection: Students discuss the significance of Function Points in software estimation and reflect on their experience in calculating Function Points for the sample project. Learning Outcomes: By the end of this lab experiment, students are expected to:
Understand the concept of Function Points and their role in software size measurement.
Gain practical experience in applying the Function Point Analysis (FPA) technique to assess software functionality.
Learn to categorize functionalities and assign complexity weights in Function Point calculations.
Develop estimation skills to assess development effort based on calculated Function Points.
Appreciate the importance of accurate software size measurement in project planning and resource allocation.

Pre-Lab Preparations:

Before the lab session, students should familiarize themselves with the concept of Function Points and the guidelines for their calculation. They should review the categorization of functionalities and the complexity weighting factors used in Function Point Analysis.

Materials and Resources:

Project brief and details for the sample software project
Function Point Analysis guidelines and complexity weighting criteria
Calculators or spreadsheet software for performing calculations

Conclusion: The lab experiment on calculating Function Points for a software project provides students with a practical approach to estimating software size and complexity. By applying the Function Point Analysis (FPA) technique, students gain insights into the importance of objective software measurement for project planning and resource allocation. The hands-on experience in identifying functionalities and assigning complexity weights enhances their estimation skills and equips them with valuable techniques for effective project management. The lab experiment encourages students to apply Function Point Analysis in real-world scenarios, promoting accuracy and efficiency in software size measurement for successful software engineering projects.

External inputs:

1. **User Registration:**
   When a user registers on the website, they provide information such as their name, email address, and password. Each of these would be considered an external input.
2. **Pet Adoption Form:**
   When a user fills out a form to adopt a pet, they provide information about their preferences, living conditions, and contact details.
3. **User Login:**
   When users log into their accounts, the username and password they enter would be external inputs.
4. **Report Lost Pet:**
   If there's a feature for users to report a lost pet, the information they provide about the pet and its last known location would be external inputs.
5. **Search Filters:**
   If users can apply filters when searching for pets (e.g., by species, age, location), the filter criteria they enter would be considered external inputs.

External Outputs (EO):

1. **Pet Adoption Confirmation:** When a user successfully adopts a pet, a confirmation message is displayed. This could be a simple external output with low complexity.
2. **Pet Listing Display:** Displaying a list of available pets with their details (name, breed, age, etc.) based on user search criteria. This could be of average complexity.
3. **Pet Details Page:** Displaying detailed information about a selected pet, including its story, health status, and adoption requirements. This could be of average complexity.
4. **Donation Receipt:** After a user makes a donation, a receipt page is displayed, showing the donation amount and a thank-you message. This might be a simple external output with low complexity.

External Inquiries (EQ):

1. **Search for Pets:** Users can inquire about available pets based on their preferences. This involves input (search criteria) and output (list of matching pets) and could be of average complexity.
2. **Contacting Support:** Users can inquire about adoption processes or specific pet details through a contact form. This involves input (user inquiry) and output (support response) and might be of average complexity.
3. **Checking Adoption Status:** Users can inquire about the status of their adoption application. This could be a simple inquiry with low complexity.

Internal Logical Files (ILF):

1. **User Profile:** Storing and managing user profile information, including names, contact details, and adoption history. This would likely be of average complexity.
2. **Pet Database:** Storing and managing information about pets available for adoption, including details like breed, age, health status, and adoption status. This could be of average complexity.

External Interface Files (EIF):

1. **Pet Medical Records:** Interfacing with an external system or database that holds medical records for pets. This might be of average complexity if the integration is straightforward.
2. **Location Data:** Interfacing with an external source to provide location data for pets available for adoption. This could be of low complexity if the data exchange is simple.

- External Inputs (EI):
    - Low Complexity: 5
    - Average Complexity: 3
    - High Complexity: 2
- External Outputs (EO):
    - Low Complexity: 4
    - Average Complexity: 2
    - High Complexity: 1
- External Inquiries (EQ):
    - Low Complexity: 6
    - Average Complexity: 4
    - High Complexity: 2
- Internal Logical Files (ILF):
    - Low Complexity: 3
    - Average Complexity: 2
    - High Complexity: 1
- External Interface Files (EIF):
    - Low Complexity: 2
    - Average Complexity: 1
    - High Complexity: 1

Information	Weighting Factor			
Domain Value	Count	Simple	Average	Complex
External Inputs	5	5	3	2
External Outputs	4	4	2	1
External Inquiries	3	6	4	2
Internal Logical files	2	3	2	1
External Logical files	2	2	1	1

- For External Inputs:

  FP = 10 * (0.65 + (0.01 * 10)) = 10 * (0.65 + 0.1) = 10 * 0.75 = 7.5

- For External Outputs:

  FP = 7 * (0.65 + (0.01 * 7)) = 7 * (0.65 + 0.07) = 7 * 0.72 = 5.04

- For External Inquiries:

  FP = 12 * (0.65 + (0.01 * 12)) = 12 * (0.65 + 0.12) = 12 * 0.77 = 9.24

- For Internal Logical Files:

  FP = 6 * (0.65 + (0.01 * 6)) = 6 * (0.65 + 0.06) = 6 * 0.71 = 4.26

- For External Logical Files:

  FP = 4 * (0.65 + (0.01 * 4)) = 4 * (0.65 + 0.04) = 4 * 0.69 = 2.76

  Total FP = FP for EI + FP for EO + FP for EQ + FP for ILF + FP for

  EIF Total FP = 7.5 + 5.04 + 9.24 + 4.26 + 2.76 = 28.8

a) Critically evaluate the Function Point Analysis method as a technique for software sizing and estimation, discussing its strengths and weaknesses.

Strengths:

Technology Agnostic: FPA is technology-independent, focusing on the functionality and complexity of software systems rather than specific technologies or coding practices. This makes it applicable to a wide range of projects.

Focus on User Perspective: FPA assesses software from a user's perspective, considering how users interact with the system. This helps in aligning software development with user needs.

Sizing Metric: FPA provides a quantitative measure of software size, aiding in project planning, cost estimation, and resource allocation.

Mature and Proven: FPA is a well-established technique with decades of use and refinement. It has a solid foundation and well-documented rules.

Weaknesses:

Subjectivity: FPA relies on human judgment to assess complexity and functionality, which can introduce subjectivity and variability in estimations. Different analysts may assign different complexity values.

Learning Curve: FPA requires training and experience to be used effectively. Novice analysts may struggle to accurately assess complexity and functionality.

Not Suitable for Early Phases: FPA is typically applied during the later stages of project development when detailed requirements are available. It may not be suitable for estimating software in the early phases when requirements are still evolving.

Limited Scope: FPA doesn't consider non-functional aspects of software, such as performance, security, and usability, which are critical in many projects.

Influence of Previous Projects: Estimators may be influenced by their experience with similar projects, leading to biased estimations.

b) Apply the Function Point Analysis technique to a given software project and determine the function points based on complexity and functionalities.

Identify Functional Components:

Identify and document all the functional components of the software, such as user inputs, outputs, and inquiries.

Assign Complexity Weights:

Evaluate the complexity of each functional component and assign complexity weights (Low, Average, or High) based on established FPA rules and guidelines.

Count Components:

Count the total number of functional components for each complexity level (Low, Average, High) in each of the five categories (External Inputs, External Outputs, External Inquiries, Internal Logical Files, External Interface Files).

Calculate Function Points:

Calculate function points using the formula: Total count * (0.65 + (0.01 * F)), where F is the total count of complexity values for External Inputs and External Outputs.

c) Propose strategies to manage and mitigate uncertainties in function point estimation and how they can impact project planning and resource allocation.

Use Range Estimates:

Instead of providing a single function point estimate, provide a range (e.g., optimistic, pessimistic, and most likely estimates). This helps project managers account for uncertainty in their planning.

Leverage Historical Data:

Use historical data from past projects to refine estimations. Compare actual function points with estimated ones to improve the accuracy of future estimates.

Expert Review:

Conduct peer reviews or expert assessments of the function point estimates to reduce subjectivity and improve accuracy.

Iterative Estimation:

Continuously refine function point estimates as the project progresses and more information becomes available. Update estimates as requirements evolve.

Scenario Analysis:

Analyze different scenarios by varying assumptions to understand the potential impact of uncertainty on project outcomes.

Buffer in Resource Allocation:

Allocate additional resources or time buffers to account for uncertainty in function point estimates. This ensures that the project remains on track even if estimates are not precise.

Regularly Communicate Estimation Risks:

Maintain open communication with stakeholders about the uncertainties in function point estimation and the potential impact on project timelines and budgets.