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:	**14-08-2023**
Roll No:	**9600**
Team Members:	**Cloyster DSouza**

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:

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.

Waste Disposal Detection: This involves the functionality to detect when waste is disposed of into the smart bin using RPi sensors, object detection model, and pose detection model. This can be considered a complex functionality because it involves multiple components and technologies.

Rewards System: This includes the functionality to reward users based on the successful detection of waste disposal. It might also involve managing user accounts, points calculation, and notifications. This can be considered a medium to complex functionality, depending on the complexity of the reward rules.

Redemption System: The functionality to allow users to redeem their rewards. This may include cataloguing rewards, managing user requests, and updating user accounts. This can be considered a medium complexity functionality.

Tracking Points: Users need to be able to see their accumulated points, which would involve displaying user-specific data. This is relatively straightforward and can be considered a simple functionality.

Location Tracking of Bins: This functionality involves showing the locations of smart bins on a map. It may require integration with location services or APIs, but it's not inherently complex.

To calculate function points for the project:

1. External Inputs (EI):

Detecting waste disposal events, Managing user accounts

- Assign complexity weights based on your assessment:
- Waste Disposal Detection (Assume Average Complexity): 4 - User Account Management (if complex, assume High Complexity): 6
- Count the number of External Inputs:
- You might have multiple instances of waste disposal detection but consider it as one function for simplicity.
- Let's assume you have 1 instance of User Account Management. - Calculate the total weighted External Inputs:
- Total Weighted EI = (1 * 4) + (1 * 6) = 10

2. External Outputs (EO):

Notifying users of rewards, Displaying user-specific data (accumulated points).

- Assign complexity weights based on your assessment:
- Notifications (Assume Average Complexity): 5 - Displaying User Data (Assume Low Complexity): 4 - Count the number of External Outputs:
- You might have multiple instances of notifications and user data display, but consider them as one function each for simplicity.
- Let's assume you have 1 instance of each.

- Calculate the total weighted External Outputs:

- Total Weighted EO = (1 * 5) + (1 * 4) = 9

3. External Inquiries (EQ):

Checking the status of rewards, Checking the location of smart bins.

- Assign complexity weights based on your assessment:
- Rewards Status (Assume Average Complexity): 4
- Location of Smart Bins (Assume Low Complexity): 3
- Count the number of External Inquiries:
- Assume you have 1 instance of each.
- Calculate the total weighted External Inquiries:
- Total Weighted EQ = (1 * 4) + (1 * 3) = 7

4. Internal Logical Files (ILF):

User account data, Reward catalog data.

- Assign complexity weights based on your assessment:
- User Account Data (Assume Average Complexity): 10
- Reward Catalog Data (Assume Low Complexity): 7 - Count the number of Internal Logical Files:
- Assume you have 1 instance of each.
- Calculate the total weighted Internal Logical Files:
- Total Weighted ILF = (1 * 10) + (1 * 7) = 17

5. External Interface Files (EIF):

Integration with location services or APIs.

- Assign complexity weights based on your assessment:
- Integration with Location Services (Assume Average Complexity): 7 - Count the number of External Interface Files:
- Assume you have 1 instance.
- Calculate the total weighted External Interface Files:
- Total Weighted EIF = (1 * 7) = 7

6. Calculate Unadjusted Function Points (UFP):

Sum the weighted counts of all five function types:

UFP = Total Weighted EI + Total Weighted EO + Total Weighted EQ + Total Weighted ILF + Total Weighted EIF

UFP = 10 + 9 + 7 + 17 + 7

= 50 (Unadjusted Function Points)

The Fi (i=1 to 14) are value adjustment factors (VAF) based on responses to the following questions

1. **Does the system require reliable backup and recovery?**

Response: 3

2. **Are specialised data communications required to transfer information to or from the application? Response: 5**

3. **Are there distributed processing functions? Response: 3**

4. **Is performance critical? Response: 5**

5. **Will the system run in an existing, heavily utilised operational environment?**

Response: 2

6. **Does the system require online data entry?**

Response: 5

7. **Does the online data entry require the input transaction to be built over multiple screens or operations? Response: 3**

8. **Are the ILFs updated online?**

Response: 5

9. **Are the inputs, outputs, files, or inquiries complex? Response: 4**

10. **Is the internal processing complex?**

Response: 4

11. **Is the code designed to be reusable?**

Response: 4

12. **Are conversion and installation included in the design?**

Response: 3

13. **Is the system designed for multiple installations in different organizations?**

Response: 0

14. **Is the application designed to facilitate change and ease of use by the user?**

Response: 4

Calculations: Given: $\sum (Fi) = 50$

To Calculate: FP

Formula: 1. FP= UFP*CAF

2. CAF= 0.65 +0.01 * $\sum (Fi)$

Soln:
Soln: CAF = 0.65+0.01* ∑ (Fi)
CAF = 0.65+0.5
CAF = 1.15

FP = UFP*CAF
FP = 50*1.15
FP = 57.5

The Function Point for SmartBin+ is 57.5

Conclusion:

The Function Point (FP) value for the SmartBin+ is 57.5. This metric reflects the system's complexity and size, by taking the factors stated above into consideration. The Function Point metric offers insightful data on the time and resources needed for platform development, testing, and maintenance.

POSTLAB

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

Function Point Analysis (FPA) is a widely used method for software sizing and estimation. It was developed by Allan Albrecht in the 1970s and is based on the idea of quantifying the functionality provided by a software application. FPA measures software size in terms of "function points," which represent the user-visible functionality of a system. While FPA has been a valuable technique in software development for many years, it also has its strengths and weaknesses, which should be critically evaluated.

Strengths of Function Point Analysis:

1. Vendor and Technology Agnostic: FPA is technology-neutral and doesn't depend on a specific programming language, platform, or development methodology. This makes it applicable to a wide range of software projects.

2. User-Centric: FPA focuses on the functionality that the end users will see and interact with. This user-centric approach helps in understanding the software's true value and aligns development efforts with user needs.

3. Objective Measurement: FPA provides an objective way to measure software size. It breaks down functionality into standard categories (External Inputs, External Outputs, External Inquiries, Internal Logical Files, and External Interface Files) and assigns complexity weights to each, leading to a consistent size measurement.

4. Estimation Accuracy: When applied correctly, FPA can provide reasonably accurate estimates of software development effort, cost, and duration. It helps in setting realistic expectations for project stakeholders.

5. Benchmarking: Organizations can use FPA data to benchmark their projects against industry averages and historical data, aiding in performance evaluation and process improvement.

6. Early Estimation: FPA can be used early in the project lifecycle, even before coding begins. This helps in project planning and resource allocation.

Weaknesses of Function Point Analysis:

1. Subjectivity in Counting: Despite its claims of objectivity, FPA can be susceptible to subjectivity. Different analysts may interpret and count functions differently, leading to variations in estimates.

2. Learning Curve: FPA requires skilled analysts who have undergone training and certification. Learning FPA can be time-consuming, and errors in counting can lead to inaccurate estimates.

3. Complexity Weights: Assigning complexity weights to functions can be challenging. The weights are somewhat arbitrary and can lead to disagreements among analysts.

4. Inadequate for Non-Functional Requirements: FPA primarily focuses on functional requirements and doesn't address non-functional aspects like performance, security, and scalability. These factors are crucial but need to be estimated separately.

5. Limited in Agile Environments: FPA was designed for traditional waterfall development models. It may not align well with Agile methodologies, where requirements change frequently, making it harder to maintain accurate function point counts.

6. Limited for Very Small or Very Large Projects: FPA may not be practical for very small projects with minimal functionality or extremely large projects with numerous functions, as the counting effort may not be justified.

7. Lack of Industry-Wide Consistency: While FPA has a standard framework, its application can still vary from one organization to another, reducing the consistency of measurements and comparisons.

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

Function Point Analysis (FPA) is a systematic process that involves counting and categorizing various functional components of a software project. To apply FPA, you need detailed knowledge of the project's requirements and functionality. Below, I'll provide a simplified example of how FPA can be applied to a hypothetical software project.

Scenario:

Let's consider a simple e-commerce website as our software project. We'll break down the functionality into different categories and assign complexity weights based on a standard FPA table. Please note that in a real-world scenario, the analysis would be more detailed, and actual function point counts may vary.

Functional Categories:

1. External Inputs (EI): user registration, login, and product search.

2. External Outputs (EO): displaying product details, order confirmation, and generating invoices.

3. External Inquiries (EQ): checking order status.

4. Internal Logical Files (ILF): customer database.

5. External Interface Files (EIF): a product catalog provided by a third-party supplier.

Complexity Weights:

Each function point category (EI, EO, EQ, ILF, EIF) has different complexity levels: low, average, and high. Below are some example complexity weights, but these can vary depending on the organization's specific FPA guidelines.

- Low Complexity (3 points): Simple functions with few data elements and straightforward processing.
- Average Complexity (4 points): Moderately complex functions with more data elements or some degree of complexity.
- High Complexity (6 points): Highly complex functions with many data elements or intricate processing.

Function Point Calculation:

Now, let's count the function points based on the functionality and complexity:

1. External Inputs (EI):
   - User registration (Average Complexity): 4 points
   - User login (Low Complexity): 3 points

   - Product search (Average Complexity): 4 points

2. External Outputs (EO):
   - Display product details (Average Complexity): 4 points
   - Order confirmation (Low Complexity): 3 points

- Generate invoices (High Complexity): 6 points

3. External Inquiries (EQ):
   - Check order status (Average Complexity): 4 points

4. Internal Logical Files (ILF):
   - Customer database (High Complexity): 6 points

5. External Interface Files (EIF):
   - Product catalog (Average Complexity): 4 points

Total Function Points: Summing up the points for all categories:

Total Function Points = 4 + 3 + 4 + 4 + 3 + 6 + 4 + 6 = 34 Function Points

In this simplified example, the estimated function points for the e-commerce website project are 34. Please note that this is a basic illustration, and in a real-world scenario, you would perform a more detailed analysis with the involvement of domain experts and FPA specialists to arrive at a more accurate function point count.

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

Managing and mitigating uncertainties in function point estimation is critical for effective project planning and resource allocation. Uncertainties can arise from various factors, including changing requirements, complexity, and the subjectivity of the estimation process. Here are some strategies to manage and mitigate these uncertainties:

1. Use Historical Data: Collect and analyze historical data from previous projects to identify patterns and trends. This data can provide insights into the accuracy of past function point estimates and help in making more informed estimates for the current project.

2. Sensitivity Analysis: Conduct sensitivity analysis to identify which function points or components have the greatest impact on project outcomes. This allows you to focus your efforts on accurately estimating the most critical parts of the project.

3. Expert Involvement: Engage experienced analysts who are well-trained in Function Point Analysis. Their expertise can help reduce estimation uncertainties, as they are more likely to make accurate assessments of complexity and functionality.

4. Prototyping: Create prototypes or proof-of-concept models for complex or uncertain parts of the project. This allows you to gather more concrete information about requirements and functionality before finalizing estimates.

5. Iterative Estimation: Instead of relying solely on a single estimate, use an iterative approach. Continuously refine and update the estimates as more information becomes available throughout the project lifecycle. This helps in adapting to changing requirements and reducing uncertainty.

6. Risk Management: Implement a robust risk management process. Identify potential risks that could impact function point estimation and develop mitigation strategies for each risk. This proactive approach can help in handling uncertainties as they arise.

7. Buffering: Add contingency or buffer to the estimates to account for uncertainties. This can be in the form of extra time, resources, or budget. However, it's essential to communicate clearly with stakeholders about the purpose of these buffers and the potential consequences of using them.

8. Benchmarking: Compare your function point estimates with industry benchmarks and data from similar projects. This external benchmarking can provide additional perspective and help validate your estimates.

9. Documentation: Maintain detailed documentation of the function point analysis process, including assumptions made during estimation. This documentation helps in tracking changes and justifying estimates to stakeholders.

10. Regular Review: Periodically review and update function point estimates as the project progresses. This ensures that estimates remain accurate as requirements evolve and uncertainties are resolved.

11. Stakeholder Communication: Establish open and transparent communication with project stakeholders. Keep them informed about estimation uncertainties and their potential impact on project outcomes. Discuss trade-offs and decisions related to resource allocation.

12. Training and Certification: Ensure that analysts involved in function point estimation are well-trained and certified in FPA methodologies. This can improve the consistency and reliability of estimates.