Institute of Information Technology
University of Dhaka

Topic: Function Point Analysis of SPL-2 Project
Software Metrics (SE-611)

Submitted to

Dr. Emon Kumar Dey
Associate Professor
IIT, University of Dhaka

Submitted by

Md. Shakibul Islam Shakib - BSSE 1404
Nandan Bhowmick - BSSE 1436

1. Project Overview

The rental market in Dhaka city presents significant challenges for both renters and landlords. Renters struggle with high demand, limited trustworthy information, and difficulty finding properties that meet their budget, location, and amenity preferences. Landlords, on the other hand, face issues like effectively showcasing properties, establishing credibility, and managing tenant communication. Our application aims to resolve these problems by offering a streamlined, all-in-one platform that enhances the rental experience for both renters and landlords. Following are the primary features of our application:

1.1 User Registration & Login

Users create personalized accounts with basic details such as full name, email, phone number, and profile picture. If a password is forgotten, an OTP-based email verification system is used. There's no separate process for landlords—any user can become one by listing a property. Profiles may include optional document verification to build trust. All users log in via the same system, and dashboards are tailored to their activities (e.g., renting or hosting).

1.2 Property Listings & Management

Users can create and manage property listings with detailed information like location, price, property type, amenities, and media files. Listings can be edited, updated, or deleted through the dashboard. Renters maintain a wishlist and view rental history, while landlords categorize listings into active, reserved, or rented-out properties with tenant information.

1.3 Search by Area

An area-based search system allows users to find or list properties by specific Dhaka neighborhoods such as Dhanmondi, Gulshan, or Mohammadpur. A dropdown or interactive map enables filtering by landmarks, schools, or public transport, improving discoverability and precision.

1.4 Advanced Search Filters

Users can refine searches or improve listing visibility using filters like budget, property type, amenities, number of rooms, and location proximity. Tagged metadata ensures that relevant properties appear in targeted search results.

1.5 Detailed Property View

Every property listing includes comprehensive information, such as descriptions, amenities, house rules (e.g., no smoking), room count, availability, and the profile of the property owner or landlord with contact options.

1.5 Rent Calculator

A rent calculator allows users to estimate monthly rent by inputting details like location, room count, and amenities. Renters use it for budget planning, while landlords can price their properties competitively using market-based suggestions.

1.6 Visit Scheduling

Users can schedule property visits through a calendar and time picker. Scheduled visits trigger automatic notifications—renters can track appointments, and landlords stay informed about incoming visits.

1.7 Notifications

The platform sends real-time email and in-app notifications for key events, such as property wishlist additions, scheduled visits, or new messages. This ensures timely communication and engagement.

1.8 In-Platform Chat

A built-in messaging system enables direct communication between renters and landlords. Users can ask questions, schedule visits, negotiate terms, and share documents within the platform.

1.9 Integrated Map

Properties are displayed on an interactive map, helping users visually locate listings. When creating a listing, landlords can specify exact property locations to enhance clarity and visibility for potential renters.

2. Identifying Function Point Components

2.1 External Inputs (EI)

#	Feature	Reason	Complexity	FP
1	User Registration/Login/Reset	Basic fields like email, password; simple form validation	Low	3
2	Add/Update/Delete Property	Many fields, media, amenities, map input, high data complexity	High	6
3	Add/Remove from Wishlist	Simple link between renter and property	Low	3
4	Schedule Visit	Includes time/date selection, linked to property and landlord	Average	4
5	Send Message (Chat)	Single message input stored with metadata	Low	3
6	OTP Submission	OTP + timestamp, simple validation	Low	3
7	Rent Calculator Input (ML)	Simple fields, but model logic applied	Average	4
8	Make Reservation	Booking across multiple entities (user, property, time)	Average	4

Table 1: External Inputs

2.2 External Outputs (EO)

Outputs generated by the system to inform or confirm actions.

#	Feature	Reason	Complexity	FP
1	Notifications (visit/wishlist)	Simple text notifications	Low	4
2	Search Results	Filtered and sorted result set, multi-field display	High	7
3	Visit Confirmation/Cancellation	Standard confirmation messages	Low	4
4	Rent Prediction (ML Output)	Computed result, numeric estimate	Average	5
5	Booking Summary	Multi-entity data, confirmation of reservation	Average	5

Table 2: External Outputs

2.3 External Inquiries (EQ)

User requests to view system data without modifying it.

#	Feature	Reason	Complexity	FP
1	View Property Details	Multiple attributes, property overview	Average	4
2	View Wishlist	Simple list retrieval	Low	3
3	View Reservations	List-based, filtered view with basic data	Average	4
4	View Schedule	List of visits by date/time	Low	3
5	View Messages	Chat log display	Low	3

Table 3: External Inquiries

2.4 Internal Logical Files (ILF)

Logical groups of data maintained (created/updated/deleted) by the application.

#	ILF	Reason	Complexity	FP
1	Renter	Simple profile data	Low	7
2	Landlord	Same as Renter	Low	7
3	Property	Complex: media, amenities, coordinates, descriptions	High	15
4	Wishlist	Renter-property mapping	Low	7
5	Reservation List	Multi-entity records: renter, property, date, status	Average	10
6	Schedule	Stores date/time/landlord/renter	Average	10
7	Notification	Stores message, timestamp, status	Low	7
8	Chat	Message content, sender/receiver, timestamp	Average	10
9	OTP	Token, status, timestamp	Low	7

Table 4: Internal Logical Files

2.5 External Interface Files (EIF)

Data referenced from outside the system (not maintained internally).

#	EIF	Reason	Complexity	FP
1	Map API	Retrieves and displays geolocation info for listings	Average	7
2	Email Service	Sends OTP/notifications; simple message format	Low	5

Table 5: External Interface Files

3. Unadjusted Function Points (UFP)

Given Values:

Number of **External Inputs (EI)** = 30

Number of **External Outputs (EO)** = 25

Number of **External Inquiries (EQ)** = 17

Number of **Internal Logical Files (ILF)** = 80

Number of **External Interface Files (EIF)** = 12

Formula for Unadjusted Function Points (UFP):

$$UFP = EI + EO + EQ + IFL + EIF$$

UFP = 30 + 25 + 17 + 80 + 12 = 164

Therefore, **UFP** is **164**.

4. Technical Complexity Factor (TCF)

4.1 14 General System Characteristics (GSCs) with Justification

#	Characteristic	Rating	Justification
1	**Data Communications**	3	Frequent HTTP/API communication for chat, map, email
2	**Distributed Processing**	2	Backend-frontend and ML inference may run separately
3	**Performance**	3	Real-time responses needed for chat, booking, search
4	**Heavily Used Configuration**	3	Properties, search filters, etc. are configurable
5	**Transaction Rate**	2	Medium level of activity (search, booking, chat)
6	**On-line Data Entry**	4	Many forms for registration, booking, property entry
7	**End-User Efficiency**	4	Clean UI with chat, map, search, etc. improves user efficiency
8	**On-line Update**	3	Property updates, reservation updates
9	**Complex Processing**	3	ML model (XGBoost), filters, mapping, OTP verification
10	**Reusability**	2	Components like OTP, Chat can be reused
11	**Installation Ease**	1	Web-based deployment; no special client setup
12	**Operational Ease**	3	User-friendly design, notifications assist operations

13	**Multiple Sites**	2	App intended for wide use (landlords/renters), in dhaka city areas
14	**Facilitate Change**	3	Modular components make features extensible

Table 6: 14 General System Characteristics (GSCs) with Justification

4.2 Calculate TCF

To calculate TCF, we first sum up the ratings of the 14 general system characteristics:

Sum of Degree of Influence (DI):

DI = 3 + 2 + 3 + 3 + 2 + 4 + 4 + 3 + 3 + 2 + 1 + 3 + 2 + 3 = 38

TCF Formula:

$$TCF = 0.65 + (0.01 \times DI)$$

TCF = 0.65 + (0.01 × 38) = 1.03

Therefore, **TCF** is **1.03**.

5. Convert Function Points to LOC

Calculate Adjusted Function Points (AFP)

$$AFP = UFP \times TCF$$

AFP = 164 × 1.03 = 168.92

Therefore, **AFP** is **168.92**.

Assuming a development technology mix of **90% JavaScript (frontend and Node.js backend)** at approximately **50 lines of code per function point**, and **10% Python with XGBoost (ML backend)** at approximately **60 lines of code per function point**, the Weighted Language Factor is calculated as:

Weighted LOC/FP = (0.9*50)+(0.1*60) = 51

Estimated Lines of Code (LOC):

$$LOC = Function\ Points(FP) \times Language\ Factor$$

Estimated LOC = 168.92 × 51 = 8614.92 ≈ 8615

Comparison with real project:

Category	File Name	Lines of Code
Models	booking.js	41
	Listing.js	100
	message.js	46
	notification.js	40
	user.js	46
Routes	auth.js	176

	booking.js	36
	listing.js	319
	message.js	245
	notification.js	143
	rentprediction.js	54
	user.js	315
Main Server Entry	index.js	85
Flask ML Backend	backend.py	85
Frontend Pages	Category.jsx	168
	Creatlisting.jsx	560
	ForgetPasswordPage.jsx	112
	HomePage.jsx	20
	LandlordInboxPage.jsx	103
	ListingDetails.jsx	313
	LoginPage.jsx	71
	NotificationPage.jsx	204
	PropertyList.jsx	75
	RegisterPage.jsx	114
	RentPredictorPage.jsx	305
	ReservationList.jsx	68
	SearchPage.jsx	164
	TripList.jsx	60
	UpdatePage.jsx	663
	WishList.jsx	54

Frontend Components	Categories.jsx	33
	Footer.jsx	44
	ListingCard.jsx	241
	Listings.jsx	90
	Loader.jsx	12
	Navbar.jsx	176
	NotificationPopup.jsx	36
	chatPage.jsx	242
	instantDelete.jsx	36
	slide.jsx	17
Styles (SCSS)	Categories.scss	84
	CreateListing.scss	402
	Footer.scss	93
	variables.scss	40
	slide.scss	15
	chat.scss	395
	breakpoints.scss	44
	RentPredictorPage.scss	375
	Register.scss	173 (97 + 76)
	NotificationPage.scss	235
	Navbar.scss	165
	Login.scss	98
	Loader.scss	20
	ListingDetails.scss	180
	ListingCard.scss	144

	Listing.scss	51
	List.scss	48
	LandlordPage.scss	39
	LandlordInbox.scss	190
	ForgetPassword.scss	89
Redux & Other	state.js	46
	store.js	33
	frontend routing	60
	data.js	272
	index.js	15
	NotificationContext.jsx	71

Table 7: Comparison with real project

Therefore, **Actual LOC** = 8925.

Efficiency Formula:

$$Efficiency = (\frac{Estimated\ LOC}{Actual\ LOC}) \times 100$$

Efficiency = (8615/8925) × 100 ≈ 96.5%

Therefore, the efficiency of the project is 96.5%.

6. Estimate Project Metrics

The **COCOMO I (Constructive Cost Model I)** is used to estimate the required development effort, time, team size, and cost for software projects.

Software projects under COCOMO model strategies are classified into
3 categories, organic, semi-detached, and embedded

Organic: A software project is said to be an organic type if-

- Small, simple software projects.
- Teams are small and have good experience with similar projects.
- Requirements are well understood and flexible.
- Few constraints on hardware or operational environment.
- Examples: Simple business applications, payroll systems, small utilities.

Semi-Detached Mode: A software project is said to be a
Semi-Detached type if-

- Medium-sized projects with moderate complexity.
- Teams consist of both experienced and less-experienced members.
- Requirements may be partially flexible but have some constraints.
- Requires more management and coordination than organic mode.
- Examples: Transaction processing systems, database management systems, larger business applications.

Embedded Mode: A software project is said to be an Embedded mode type if-

- Large, complex projects with tight constraints.
- Strict requirements on reliability, performance, and resource usage.
- Often involves hardware, real-time systems, or safety-critical applications.
- Teams are usually large and highly skilled.
- Examples: Aerospace software, medical devices, defense systems, real-time embedded systems.

Based on the nature of this project—relatively small, straightforward, and developed by a small team—the **Organic mode** of COCOMO I is selected.

The constants used for Organic mode are as follows:

Parameter	**Value**
a	2.4
b	1.05
c	2.5
d	0.38

Table 8: Constants for Organic mode of COCOMO I

Input parameters:

Metric	**Value**
Adjusted Function Points (AFP)	168.92
Language Factor (LOC per FP)	55
Estimated LOC	8615

Table 9: Input parameters

Estimation Calculations:

Estimated Effort:

$$Effort = a \times (KLOC)^b$$

Effort = 2.4×(8.615)^1.05 ≈ 2.4 × 10.14 = **23.03 Person-Months**

Estimated Development Time:

$$Duration = c \times (Effort)^d$$

Duration = 2.5 × (23.03)^0.38 ≈ **8.23 Months**

Estimated Team Size:

$$Team\ Size = \frac{Effort}{Duration}$$

Size = 23.03/8.23 ≈ **3 people**

Estimated Cost:

Assuming a **monthly developer salary of ৳40,000**, the cost is estimated as:

Cost = 23.03 × 40000 = ৳**921200**

Based on the COCOMO I model using the Organic mode, the software project's adjusted function point count was calculated as **168.92**, with an estimated **8,615 lines of code** derived from a mixed tech stack (Python for the ML backend and MERN for the main web application). Using this as input, the total development **effort** was estimated to be **23.03 person-months**, with a **project duration** of approximately **8.23 months**. To meet this timeline, a team of **3 developers** would be required. Assuming a monthly developer salary of ৳**40,000**, the **total estimated development cost** comes to ৳**9,21,200**. These estimations provide a solid foundation for planning resources, budget, and timeline for successful project delivery.

References

[1]https://github.com/shakib1404/SPL2-reantalapp

[2] https://docs.google.com/document/d/1QkKz1MNEiP-IoZX1FEF5D-cevipHWZ73yFLsKAUGaiM/edit?tab=t.0