

Session 17

Reverse engineering the expense splitter of a bill-splitting app from its observed behaviour

Sessions 17 and 18 are one exercise split over two lab days. In this session you study an application you did not write, using only what it shows you on screen, and you write down what it must be doing inside. In Session 18 you build that portion yourself and compare. The application chosen here is the group expense splitter of a bill-splitting app (Splitwise on Android). The deliverable of this session is a study record: observed features, inferred inputs and outputs, an inferred data model, inferred rules, a requirements list, and a build plan. No code is written today.

Objectives

✕ Do not copy. Read for understanding and the viva

- Learn to recover requirements from the output of a running program (black-box reverse engineering).
- Separate what you observed from what you inferred, and mark each clearly.
- Choose a portion of an application small enough to rebuild in one lab session.
- Produce a requirements list that Session 18 can be checked against.

Problem Statement

■ Write in lab record

Sessions 17 and 18: Have a look at the output of any program that was not written by you. Preferably, look at an application that is not developed by you and write the program for the development of that application or a portion of that application.

Concept

✕ Do not copy. Read for understanding and the viva

Reverse engineering from behaviour

Reverse engineering means recovering a design or a specification from a finished product. When you have the source, you read it. When you have only the running program, you drive it with inputs, watch the outputs, and infer the rules that connect them. The manual says “have a look at the output”, so this session is the second kind: behaviour in, specification out. Nothing you write today is certain; it is the best explanation of what you saw.

Observation versus inference

Keep two columns in your head. An observation is something you saw on the screen: “when 100 is split three ways the app shows 33.34, 33.33, 33.33”. An inference is a rule you propose to explain it: “the app works in paise and gives the leftover paise to the first members in the list”. A viva examiner will ask which is which. Write “observed” and “inferred” next to every claim.

Picking the portion

The whole app has sign-up, friends, groups, currencies, receipts, charts, reminders and payments. You cannot rebuild that in a lab session, and the manual allows “a portion”. Choose one screen flow that has a clear input, a clear output, and at least one non-obvious rule. The expense splitter qualifies: inputs are members and expenses, outputs are balances and a settlement plan, and the interesting rules are rounding and debt simplification.

Why this belongs in software engineering

Most maintenance work starts with a program someone else wrote, often without documents. Being able to recover a specification from behaviour is the skill that makes such maintenance possible. It is also the basis of writing a compatible replacement, which is what Session 18 asks for.

Application Studied

■ Write in lab record

Item	Value
Application	Splitwise (Android app, free version), studied on 12 September 2026
Purpose	Groups of people record shared expenses and see who owes whom
Portion chosen	Group expense splitter: add members, add expenses, view balances, settle up
Portions excluded	Sign-up, friends outside a group, currencies, receipt photos, charts, reminders, online payment
Method	Created a test group “Goa trip” with four members, entered expenses, took notes on every screen

Observed Features

■ Write in lab record

No.	Feature	What was observed on screen
1	Create group	A group has a name and a list of members. Members are added by name or email
2	Add expense	Form with description, amount, "paid by" (one member by default) and "split" (equally by default)
3	Split equally	Each member's share appears; for 1000 among 4 every share is 250.00
4	Unequal split	Options: exact amounts, percentages, shares. With exact amounts, the form refuses to save until the shares add up to the total
5	Exclude a member	Unticking a member in the split leaves them out of that expense
6	Rounding	100 split equally among 3 shows 33.34, 33.33, 33.33 in list order
7	Balances screen	Each member is shown as "owes X" or "gets back X" in green or orange; a member with nothing to pay shows "settled up"
8	Settle up	Suggests who pays whom; with "simplify debts" switched on for the group, the number of payments drops
9	Expense list	All expenses in date order with description, amount, payer and share of the viewer
10	Edit and delete	Any expense can be edited or deleted and balances update at once

Features 1 to 4 and 6 to 8 are rebuilt in Session 18. Feature 5 is covered by giving a member a share of 0. Features 9 and 10 are only partly covered (list, no edit).

Inferred Inputs and Outputs

■ Write in lab record

Input	Type	Observed constraint
Member name	text	Non-empty, unique inside the group
Expense description	text	Optional; blank shows as “Expense”
Expense amount	money, two decimals	Positive; the app refuses 0
Paid by	one member of the group	Must be a member
Split mode	equal or exact amounts	Exact amounts must total the expense amount
Share per member	money	Zero is allowed (member excluded)

Output	Where seen	Content
Expense list	group screen	Description, amount, payer, shares
Balance per member	balances tab	Net amount, “owes”, “gets back” or “settled up”
Settlement plan	settle up screen	List of payments (from, to, amount)

Inferred Data Model

■ Write in lab record

```

1  Group(group_id, name)
2    |
3    | has 1..n
4    v
5  Member(member_id, group_id, name)
6    |
7    | paid 0..n                      Share(expense_id, member_id, amount)
8    v                                ^
9  Expense(expense_id, group_id, desc, | split into 1..n
10         amount, paid_by, date) -----+

```

Entity	Attributes	Evidence
Group	group_id, name	Group screen shows a name and its members
Member	member_id, group_id, name	Members are listed and picked in "paid by"
Expense	expense_id, group_id, desc, amount, paid_by, date	Expense form fields and list rows
Share	expense_id, member_id, amount	Each expense shows one share per included member

Balance is not stored; it is inferred to be computed as the sum of amounts paid minus the sum of shares owed, because editing an old expense changes the balances immediately.

Inferred Rules

■ Write in lab record

Id	Rule	Observation that supports it
R1	Money is handled in whole paise, never in fractions of a paisa	Shares are always shown to exactly two decimals and always add up to the total
R2	Equal split: each member gets the total divided by the count, rounded down to the paisa; leftover paise go one each to the first members in list order	100 among 3 gives 33.34, 33.33, 33.33
R3	Exact-amount split must sum to the expense amount	Save button stays disabled until the sum matches
R4	Net balance of a member = total paid by the member minus total of the member's shares	Balances change when any expense is edited
R5	Sum of all net balances in a group is zero	Every "owes" is matched by a "gets back"
R6	Settlement with "simplify debts" pairs the largest debtor with the largest creditor and repeats; a member either pays or receives, never both	Four members with two positive and two negative balances produced exactly two payments
R7	A member with net zero is shown as "settled up" and does not appear in the plan	Observed on the balances tab

R6 is the least certain inference. The app may use a different pairing order that produces the same number of payments. Session 18 uses the greedy largest-with-largest method and the comparison table there records whether the payment count matched.

Inferred Requirements

■ Write in lab record

Id	Requirement	Type
ES-FR-1	The system shall let the user add members to a group by name; names are unique within the group	Functional
ES-FR-2	The system shall let the user add an expense with description, positive amount, payer and split mode	Functional
ES-FR-3	The system shall split an expense equally among all members using rule R2	Functional
ES-FR-4	The system shall accept custom shares and reject them if they do not total the amount (R3)	Functional
ES-FR-5	The system shall list all expenses with amount, payer and shares	Functional
ES-FR-6	The system shall show each member's net balance as owes, gets back or settled up (R4, R7)	Functional
ES-FR-7	The system shall produce a settlement plan with the smallest number of payments it can find (R6)	Functional
ES-NFR-1	Amounts shall be shown to two decimal places and shall never lose a paisa	Non-functional
ES-NFR-2	The program shall run on the lab PC with Python 3 and no extra packages	Non-functional

Plan for Session 18

■ Write in lab record

1. One Python file `expense_splitter.py`, console menu, no persistence (the original stores data on a server; that is outside the chosen portion).
2. Store money as integer paise to satisfy R1 and ES-NFR-1.
3. Functions: `equal_split` (R2), `add_expense` with the custom-share check (R3, R4), `balances` (R4), `settle` using the greedy debtor and creditor method (R6), and menu functions for input

and display.

4. Test with the same numbers used on the real app: four members, 1000 split equally, 350 with custom shares, 800 split equally, and the 100 among 3 rounding case.
5. Fill a comparison table: original behaviour versus this program, one row per observed feature.

Viva Questions

✗ Do not copy. Read for understanding and the viva

- **Q:** What is reverse engineering? **A:** Recovering a design or specification from an existing product, here from its behaviour alone.
- **Q:** Why separate observations from inferences? **A:** Observations are facts; inferences may be wrong and must be tested in the rebuild.
- **Q:** Why work in paise instead of rupees with decimals? **A:** Floating point cannot represent 0.1 exactly, so shares would not add up; integers do.
- **Q:** How did you infer the rounding rule? **A:** 100 split among 3 gave 33.34, 33.33, 33.33, so the leftover paisa goes to the first member.
- **Q:** Why does the sum of balances have to be zero? **A:** Every rupee paid by someone is owed by someone; paid totals equal share totals.
- **Q:** What is debt simplification? **A:** Replacing many pairwise debts by fewer payments that leave every net balance the same.
- **Q:** Is the greedy settlement always minimal? **A:** No. It never pays or receives from a zero-balance member and usually gives few payments, but the true minimum is an NP-hard problem.
- **Q:** Why pick a portion instead of the whole app? **A:** The manual allows it, and a portion can be built and verified in one session.

Common Mistakes

✗ Do not copy. Read for understanding and the viva

- Describing the whole application in general terms instead of one portion in exact terms.
- Writing rules without saying what you saw that supports them.
- Forgetting the rounding case; a splitter that loses or invents a paisa is wrong.
- Listing screens and buttons but no data model, so Session 18 has nothing to build from.

- Choosing an application whose interesting behaviour is hidden (for example a search engine), which leaves nothing to infer.

Session Summary

■ Write in lab record

- Application, version, date of study, and the exact portion chosen.
- Observed features table with what was seen on screen.
- Inferred inputs, outputs and data model with evidence for each.
- Inferred rules R1 to R7 with supporting observations.
- Requirements ES-FR-1 to ES-FR-7 and ES-NFR-1 to ES-NFR-2.
- Build plan for Session 18.

 [Edit this page](#)

[< Previous](#)
Session 16

Session 18 [Next >](#)