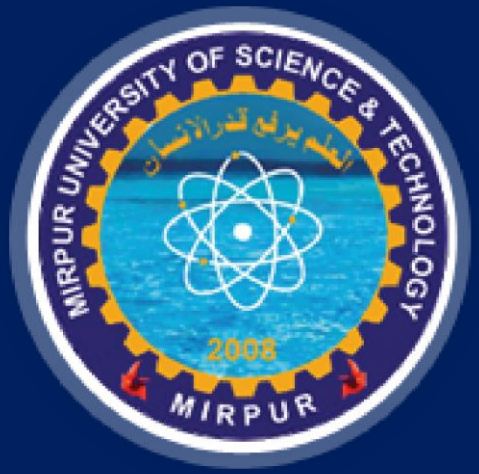

MUST

Wisdom & Virtue

MIRPUR UNIVERSITY OF SCIENCE AND TECHNOLOGY (MUST), MIRPUR

DEPARMENT OF COMPUTER SCIENCE & INFORMATION TECHNOLOGY

Engineering Economics
Lecture 4

Engr. Syeda Iqra Gillani
(Lecturer)

LECTURE DESCRIPTION

Lecture Title: Understanding Money and Its Management

Course: Software Engineering Economics

Credit Hours: 2 hours

CLO: Understand the fundamentals of money management in software project decision-making

Bloom's Level: Understanding / Applying

Mode: Interactive Lecture + Activities + Micro Quizzes

Today's Agenda

This lecture explores how money works within software projects — from funding and budgeting to cost-benefit evaluation and financial decision-making.

Students will connect *economic principles* to *real-world project scenarios.*

Module 1

The Concept of Money in Software Engineering

4.1.1 What is Money?

- **Medium of exchange, unit of account, store of value.**
- **In software context → money represents *budget, investment, revenue, cost,* and *value***

4.1.2 Why Money Matters in Software Projects

- Determines project feasibility.
- Guides decisions between alternative technologies, tools, and approaches.
- Impacts project scope, time, and quality.

4.1.3 Key Financial Terms for Engineers

Term	Meaning	Software Example
Capital	Total funds available	Initial funding for startup or project
Cost	Money spent to produce value	Developer salaries, cloud cost
Revenue	Income from product	App subscription income
ROI	Return on Investment	(Gain from project – Cost) / Cost

Class Activity 1: “Decode the Budget”

Scenario: A software startup has $50,000 to launch a new mobile app. The estimated development cost is $35,000, marketing $10,000, and maintenance $5,000.

Task:

Students work in pairs to list what financial risks this project might face and how to allocate money wisely.

Mini Quiz 1 (Quick 3-Minute Check)

- Define “money” in a project context.
- Why is budgeting crucial for a software project?
- What is ROI, and why is it important?

Module 2

Fundamentals of Money Management in Projects

4.2 The Money Flow in Software Projects

In a software project, money doesn't just *exist* —
it **moves through stages**:

- **Funding/Investment** – Seed money or client payment.
- **Allocation** – Dividing funds between design, development, testing, deployment.
- **Expenditure** – Salaries, tools, hosting, marketing, etc.
- **Revenue Return** – Payment from client or user subscriptions.
- **Profit/Reinvestment** – Used to upgrade, maintain, or expand

Example

A startup receives $100,000 from investors to build an AI-based attendance system.

- $60,000 → Developer salaries and tools
- $25,000 → Marketing & deployment
- $15,000 → Maintenance and reinvestment
- By managing this flow carefully, they ensure the project remains financially viable.

4.2.2 Core Principles of Money Management

4.2.2.1 Budgeting

- **Definition:** Estimating total project costs before execution begins.
  Goal: To ensure financial feasibility and allocate money efficiently.
- **Steps in Software Project Budgeting:**
- Identify all cost elements (personnel, hardware, software, licenses, infrastructure
- Estimate costs using historical data or expert judgment.
- Allocate budget to each phase (requirements, design, implementation, testing, maintenance).
- Add contingency (10–20%) for risks.

Real-Life Example: Budgeting in the Gmail Beta Project

Background

- In the early 2000s, Google's main products were Search and Ad Words.
  Email services like **Yahoo Mail** and **Hotmail** dominated the market — but they offered **limited storage (4–6 MB)** and slow performance.
- A small Google engineering team proposed building a new email service with **1 GB of storage per user** — revolutionary at that time.
  However, top management was hesitant because the **idea seemed risky and costly**, so they gave the team a **very limited initial budget**.

Real-Life Example: Budgeting in the Gmail Beta Project

- Google didn’t allocate a massive budget initially. Instead, the Gmail project started as part of their internal **“20% time initiative,”** where employees could spend 20% of their working hours on experimental ideas.
- **Budget Planning Strategy:**
  Rather than hiring a new team or infrastructure, they **used existing internal resources**:
- **People:** 2–3 engineers (including Paul Buchheit, the creator of Gmail) from within Google.
- **Hardware:** Reused existing Google servers.
- **Tools:** Internal development environment and data storage system.

Real-Life Example: Budgeting in the Gmail Beta Project

Component	Description	Estimated Cost
Developer salaries (partial time)	3 engineers × 20% time	$50,000
Server & hosting resources	Internal use	$20,000
Testing & bug fixing	Internal beta	$10,000
Marketing	None initially	$0
Miscellaneous (tools, utilities)	Internal shared	$5,000
Total Estimated Budget		≈ $85,000

Budget Allocation by Phases

Phase	Duration	Primary Expense	Cost Estimate	Description
Prototype Phase	3 months	Developer time	$25,000	Basic UI, storage test, internal functionality
Internal Testing	4 months	Debugging, server scaling	$20,000	Internal Google employees used Gmail to identify issues
Beta Launch	5 months	Cloud storage optimization	$30,000	Invited users via “invitation-only” model
Feedback & Improvement	3 months	Maintenance	$10,000	Adjustments based on feedback

Cost-Saving Techniques

- **Incremental Budgeting:**
  Instead of spending all at once, Google gradually increased funding as the project showed potential.
- **Resource Reuse:**
  They reused Google's existing server farms instead of purchasing new hardware.
- **No Marketing Costs:**
  Gmail's *"invitation-only"* beta launch created curiosity and **free word-of-mouth marketing**.
- **Internal Workforce:**
  Used existing employees rather than hiring new developers.

Monitoring and Adjustments

- Regular team meetings were held to track spending vs. progress.

- Since the team stayed **under budget** and performance metrics were promising, Google later approved **a larger investment** for full public release.

- The successful beta test justified increasing funding for scaling Gmail worldwide

OUTCOME

- By the time Gmail went public in 2007:
- The **initial low-budget experiment** had evolved into one of Google's most successful products.
- The budgeting approach proved that **strategic cost control + incremental funding** can lead to innovation with minimal financial risk.

Classroom Reflection Questions

- What budgeting strategy did Google use for Gmail's beta project?
- Why was incremental budgeting a good choice in this scenario?
- How did reusing internal resources impact cost efficiency?
- What lesson can software project managers learn from Gmail's budgeting story?

Take away

The Gmail beta project teaches us that budgeting isn’t just about cutting costs — it’s about **allocating resources intelligently**.

Smart budgeting enables experimentation, manages financial risk, and supports growth once the idea proves viable.

4.2.2.2 Forecasting (Predicting Future Financial Outcomes)

Definition: Using past data and trends to estimate future costs, profits, and risks.
Purpose: To avoid financial surprises during the project lifecycle.

- **Types of Forecasting in Software Projects:**
- **Cost Forecasting:** Predicting how much money will be needed in future sprints.
- **Revenue Forecasting:** Estimating expected returns from product releases.
- **Cash Flow Forecasting:** Predicting when and where money will be spent or received

Real-life Example

Background

- Slack, now one of the world's leading **team collaboration and messaging platforms**, started as a side project in 2013 when the company *Tiny Speck* pivoted from developing a failed game (*Glitch*) to building a communication tool their team had used internally.
- When they decided to turn that internal tool into a full-fledged product, **financial forecasting became critical** for determining whether the idea could become a viable business.

Step 1: The Need for Forecasting

- The founders had limited capital left from their previous gaming venture — approximately **$1.5 million**.
  Before committing to this new idea, they needed to **forecast:**
- How much it would cost to develop the product.
- When it would start generating revenue.
- Whether the remaining funds would sustain the company until breakeven.

Step 2: Creating the Financial Forecast

- Slack's team used **three forecasting models**:
- **A. Cost Forecasting**
- They estimated development, design, and infrastructure costs for the first 12 months.
- **B. Revenue Forecasting**
- The team forecasted income using a **user subscription model** at $8/user/month.
- **Cash Flow Forecasting**
- Cash flow forecasts helped the team predict **when money would enter and leave** their account.
- This allowed them to plan:
- Which months would have **negative cash flow** (expenses > income).
- When to seek **additional investment or reduce spending**.

Result

Forecasting revealed that Slack would **run out of cash by Month 10**

unless they raised new funding or delayed certain expenses.

Step 3: Decisions Based on Forecasting

- Based on forecasts, Slack's leadership made three critical financial decisions:
- **Reduce initial marketing costs by 30%** — relying on organic growth through referrals.
- **Stagger feature development** — delaying advanced functions to Phase 2, reducing upfront costs.
- **Pitch to investors early** — used forecasting data to secure $42.75M funding from Andreessen Horowitz and Accel before cash shortage.
- These decisions were all **data-driven outcomes of accurate forecasting.**

Step 4: Monitoring and Adjusting Forecasts

- Every quarter, Slack updated its forecasts with **actual results** to refine predictions:
- Actual user growth exceeded estimates by 40%.
- Infrastructure costs grew faster than planned.
- They revised their next 6-month forecast accordingly.
- This **rolling forecast model** allowed continuous control over financial direction.

Step 5: Outcome

- By mid-2014, Slack had **500,000 daily active users**.
- Their earlier **forecasting accuracy and adjustments** convinced investors of financial maturity.
- Within two years, Slack's **revenue model and cost control** turned it into one of the fastest-growing SaaS products globally.
- Today, Slack's success story is often cited in tech finance courses as a **textbook example of data-backed forecasting** in software project management.

Class Discussion Questions

- What financial risks did forecasting help Slack identify early on?
- How did accurate forecasting influence investor confidence?
- If the team had ignored cash flow forecasting, what could have gone wrong?
- How can forecasting help *your* future software projects avoid financial failure?

Take away

Forecasting in software engineering isn’t about predicting the future perfectly — it’s about preparing for it intelligently.

Slack’s story shows how data-driven forecasting prevents cash crises, supports smart decisions, and attracts investor trust

4.2.2.3 Cost Control (Tracking & Monitoring)

- **Definition:** The continuous process of comparing actual expenses with the planned budget and taking corrective actions if variances occur.
- **Techniques:**
- **Earned Value Analysis (EVA)** – Measures project performance against cost and schedule.
- **Variance Analysis** – Difference between actual cost and planned cost.
- **Cost Baseline Revision** – Adjusting budget based on project evolution.

Example

A project team building a hospital management system planned $40,000 for testing but ended up spending $60,000 due to integration bugs.

Action: They reviewed vendor contracts, optimized testing automation, and updated the cost baseline to reflect new realities.

4.2.2.4 Financial Evaluation (Decision Stage)

Definition: Using quantitative techniques to decide which projects or features are worth investment.

Common Evaluation Metrics

Metric	Description	Software Example
ROI (Return on Investment)	(Gain – Cost) / Cost	30% ROI on upgrading UI for higher sales
NPV (Net Present Value)	Value of future returns discounted to today's money	Evaluating long-term benefit of cloud migration
IRR (Internal Rate of Return)	Expected growth rate of project investment	Choosing between two SaaS product ideas
Payback Period	Time to recover initial investment	Recovering initial mobile app cost in 18 months

Example

Atlassian evaluated whether to migrate Jira Cloud infrastructure to AWS.

They calculated a **positive NPV** (long-term savings outweighed costs) and **ROI > 40%**, so they approved the migration.

Plan Budget → Allocate Funds → Track Spending → Adjust Forecasts → Evaluate ROI

Question:

Which phase do you think causes the most financial problems — budgeting, forecasting, or control? Why?

Class Activity 2: “Budget Balancers”

- **Scenario:**
- You are the **project manager** for a mobile health app. Initial funding: **$120,000**
- **Estimated Costs:**
- Development: $70,000
- Marketing: $30,000
- Server & Hosting: $10,000
- Maintenance: $10,000
- After 3 months, you discover:
- Developer overtime added $8,000 extra cost.
- Marketing campaign underperformed.
- Server costs increased due to high traffic.

Task

Task for Students:

- Identify which **money management area** was weak.
- Suggest actions to restore financial balance (cut costs, reforecast, or revise budget).
- ➞ Discuss answers in groups, then share key takeaways.

Summary

- Money management involves **planning, predicting, controlling, and evaluating** financial flows.
- Tools like ROI, NPV, and forecasting models guide *data-driven* software project decisions.
- Real-world success depends on **continuous financial visibility**, not one-time budgeting.

Module 3

Economic Decision-Making in Software Projects

4.3 Build vs Buy Decision

Scenario

- Microsoft needed a collaboration tool for enterprises after Slack gained popularity.
  They faced two options:
- **Option A:** Buy Slack (then valued at $8 billion).
- **Option B:** Build their own tool integrated with Office 365.

4.3.1 Cost Analysis

Factor	Option A: Buy Slack	Option B: Build Teams
Initial Investment	$8 billion	$1.5 billion development & marketing
Integration Effort	Low	Moderate
Control Over Product	Low (Slack IP remains separate)	Full control
Customization	Limited	High
Long-Term ROI	Medium	Very High (subscription-based model)

Decision:

- Microsoft **built Teams** internally — saving billions and generating **massive ROI** by bundling it into existing Office 365 subscriptions.
- Today, Teams has over 300 million monthly users, proving that a **long-term economic perspective** often outweighs short-term savings.

Evaluating Alternatives – Mini Case

- **Case: Cloud vs. On-Premise Hosting**
- You are managing a **university attendance management system**.

Option	Setup Cost	Annual Maintenance	Scalability	Control
Cloud (AWS)	$5,000	$2,000	High	Moderate
On-Premise	$12,000	$1,000	Low	High

Financial Analysis

- Cloud ROI (5 years): Higher because of scalability, reduced downtime, and pay-as-you-go pricing.

- On-Premise ROI: Lower but offers better control and data privacy.

Conclusion:

If project longevity > 5 years and data privacy is critical → choose **On-Premise**.

Otherwise, for startups or limited budgets → **Cloud** is economically wiser.

Activity: “The Feature Funding Dilemma”

Scenario:

- You’re managing a software project for a ride-sharing app. The development team proposes adding a **“Ride Safety Feature”** using real-time location tracking.

Details	Cost	Expected Benefit
Development	$20,000	Increased user trust
Marketing Boost	$5,000	10% rise in user base
Expected ROI	30% in 1 year	

TASK

(5–7 minutes, group discussion):

- Should you approve this feature now or delay it to the next release?
- Evaluate using ROI and Payback Period concepts.
- Each group gives 1-minute justification of their decision.

Summary of Module 3

- Economic decision-making ensures every project choice creates financial value.
- Use tools like **ROI**, **NPV**, and **Payback Period** to make rational choices.
- Real-world examples (Slack, Teams, AWS vs. On-prem) prove the importance of *financial foresight*.
- Smart decisions combine **technical excellence** with **economic prudence**.

Wrap-Up Note

Software engineering is not just about writing efficient code — it's about building sustainable, financially viable products.

A good engineer doesn't just solve problems; they invest resources wisely.