Welcome to CSE 4407: System Analysis and Design

- Why
  - Core for systems analysts, developers, and managers
  - Business Problems $\leftrightarrow$ Technical Solutions
  - Leads to clearer requirements, defined scope, fewer reworks
- Your Role: Systems “Architect”
  - Plan before you build: know users and environment
  - Define requirements precisely
  - Ensure solutions fit real-world needs
- Course Roadmap
  - Systems Analysis Fundamentals
  - Information Requirement Analysis
  - The Analysis Process
  - The Essentials of Design

Course Logistics

- Google Classroom Code
  - Theory: `caqarlqw`
  - Lab: `ad22vdpj`
- Communication
  - Discussion in Google Classroom
  - Email: cse.bakhtiarhasan@iut-dhaka.edu
- Book: Kendall and Kendall, Systems Analysis and Design, 10th Edition
- Grading Policy
  - Attendance (10%)
  - Quiz (15%)
  - Mid Semester (25%)
  - Semester Final (50%)
- Academic Integrity Policy: Do not submit others' work as yours

Systems, Roles, and Development Methodologies

CSE 4407

Md. Bakhtiar Hasan

Assistant Professor
Department of Computer Science and Engineering
Islamic University of Technology

May 7, 2025

Introduction

- Opening thought
    - Information is now a **key organizational resource**, just like people or materials.
    - It needs *management*
- The Information Explosion: Networked computers and the web have amplified the amount and complexity of information we handle
- Chapter Goals: We will explore
    - Fundamentals of Information Systems (IS)
    - Roles of Systems Analysts
    - Development Lifecycles (SDLC, Agile, O-O)
    - Open Source and CASE Tools

Why Bother with Systems Analysis and Design (SAD)?

- What: A systematic way to
    - Understand user **needs**
    - Analyze data **input/flow**
    - Analyze data **processing/transformation**
    - Analyze data **storage**
    - Analyze information **output**
    - *(All within an organizational context)*
- Primary Goal
    - Identify and solve the **RIGHT** problem
    - Analyze, design, and implement **improvements** to computerized information systems
- Analogy: Doctor diagnosing a patient

The Cost of Chaos vs. The Value of Structure

- Why SAD is CRITICAL
  - Avoid Disaster: Prevents user dissatisfaction and systems falling into disuse (*Shelfware*)
  - Manages Complexity and Cost: Provides structure to what can be a very expensive and complicated endeavor
  - Business Improvement: Aims to enhance the business using *computerized* information systems
  - Collaboration is Key: User involvement throughout the project is non-negotiable for success!
  - Global Teams: Emphasis on user interaction is even *more* critical with international development teams

Security and Privacy: Not Just an Add-On!

- The Reality
  - Security is **critical** but challenging
  - Multiple vulnerabilities exist in any system
  - “Perfect” security is unrealistic – it involves **trade-offs** (value of data vs. risk vs. cost)
- The SAD Approach: Security by Design
  - Build security and privacy controls in from the VERY BEGINNING
  - Much more effective and desirable than trying to patch security onto older systems (legacy systems)
  - Always assess and improve security when updating existing systems
  - User training on security is also crucial

The Systems Analyst: Bridging Worlds

- Who: Someone who **systematically assesses**
  - How users interact with technology
  - How businesses function
  - *(By examining data input, processing, output, and information flow)*
- Intent: To **improve** organizational processes, often using computerized IS
- Key Characteristics
  - Works with diverse **people**
  - Experienced with **computers** and technology
  - Often balances **multiple roles** simultaneously

The Analysts Wears Many Hats...

- Primary Role 1: The Consultant
  - Often an **external** hire
  - Pros: Brings a fresh, objective perspective
  - Cons: Lacks deep knowledge of internal organizational culture/politics
  - Relies on: Systematic methods (like those in this course!) and user input
- Primary Role 2: The Supporting Expert
  - Usually an **internal** employee (part of the organization)
  - Draws on **specific technical expertise** (HW, SW, databases, etc.)
  - Often involved in smaller modifications, decisions, or specific parts of a larger project
  - Acts as a **resource** for project managers, not *the* manager

…Agent of Change and Essential Qualities

- Primary Role 3: The Agent of Change
  - The most **comprehensive** and responsible role
  - Can be internal or external
  - Involved throughout the **Systems Development Life Cycle (SDLC)** (often long-term: weeks to years)
  - Their mere **presence** and activities **change** the business
  - Must work closely with users/management from Day 1
- What Makes a GREAT Systems Analyst?
  - Problem Solver: Enjoys challenges, finds workable solutions (The Core!)
  - Communicator: Relates meaningfully to diverse people over time. Understands human needs with tech (HCI). Bridges gap to developers
  - Self-Disciplined and Motivated: Manages self, coordinates people, handles resources
  - Ethical: Acts with integrity, especially with sensitive info and client relationships
  - Technically Adept and Business Savvy: Understands both sides

The Traditional Blueprint: Systems Development Life Cycle (SDLC)

- What: A **phased** approach to systems analysis and design
- Assumption: Systems are best developed using a specific cycle of analyst and user activities
- Common Analogy: The "Waterfall" Method
    - Phases flow sequentially downwards, like water over falls
    - (PMI calls this a "Predictive" life cycle) [1]

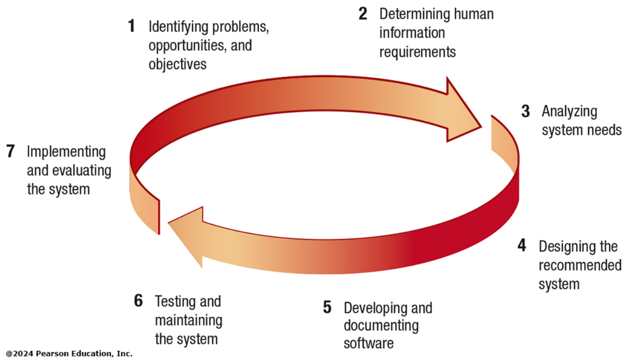

1 Identifying problems, opportunities, and objectives
2 Determining human information requirements
3 Analyzing system needs
4 Designing the recommended system
5 Developing and documenting software
6 Testing and maintaining the system
7 Implementing and evaluating the system
@2024 Pearson Education, Inc.

Figure. The 7 Phases (Kendall and Kendall Model)

SDLC Phase 1: Where Do We Start?

- Main Goal: Correctly identify the core **Problems**, **Opportunities**, and **Objectives**
- Why: Getting this wrong means wasting significant time and resources solving the *wrong issue*!
- Activities
  - Honestly assess the current business situation
  - Pinpoint specific **problems** (often the reason the analyst was called in)
  - Identify **opportunities** (situations where IT can provide improvement or competitive advantage)
  - Discover and define clear business **objectives**
- Who's Involved: Users (especially management), Analysts, Systems Managers
- Key Output: Feasibility Report (Defines problem, summarizes objectives, initial scope estimate, recommends Go/No-Go)

SDLC Phase 2: Understanding Users and Their Needs

- Main Goal: Determine **human needs** regarding the information system. How do users *need* to interact?
- Methods for Gathering Info
  - Interactive: Interviews, Questionnaires, Sampling data
  - Unobtrusive: Observing users work, analyzing existing documents (“hard data”)
  - All-encompassing: Prototyping (building preliminary versions)
- Focus: Human-Computer Interaction (HCI)
  - Physical aspects? (Legible, audible, safe?)
  - Cognitive aspects? (Easy to learn, use, remember?)
  - Affective aspects? (Pleasing, engaging, fun?)
  - Productivity? (Support tasks, enable new capabilities?)
- Key Framework: Understand the **Who**, **What**, **Where**, **When**, **How** of current processes, and Critically: **WHY?**
- Who’s Involved: Analysts, Users (Operational level often)

SDLC Phase 3: Analyzing System Needs

- Main Goal: Analyze the information gathered in Phase 2 to determine system requirements
- Key Tools and Technologies
  - Process Modeling: *How data moves and transforms*
    - Data Flow Diagrams
    - Activity Diagrams/Sequence Diagrams
  - Data Modeling: *What data is needed?*
    - Data Dictionary
  - Logic/Decision Modeling: *How are decisions made?*
    - Structured English
    - Decision Tables
    - Decision Trees
- Key Output: Systems Proposal (Summarized findings from Phase 1 and 2, includes Cost-Benefit Analysis, provides specific recommendations for the new system)
- Who's Involved: Analyst

SDLC Phase 4: Blueprinting the Solution - The Logical Design

- Main Goal: Use the requirements from Phase 3 to design the *logical* structure of the new system
- Key Design Activities
  - Input Design: Designing procedures, forms, screens for accurate and efficient data entry
  - Interface Design: Defining *how* the user interacts with the system (menu, GUI, navigation). Focus on usability, accessibility (audible, legible, safe), aesthetics. **User involvement** is crucial here.
  - Database Design: Designing the logical structure of the database to store data effectively and intuitively
  - Output Design: Designing reports, screen displays, etc. to deliver the needed information effectively
- Who's Involved: Analyst (leading design), often working closely with Users (especially for UI/Output)

SDLC Phases 5 and 6: Construction and Quality Checks

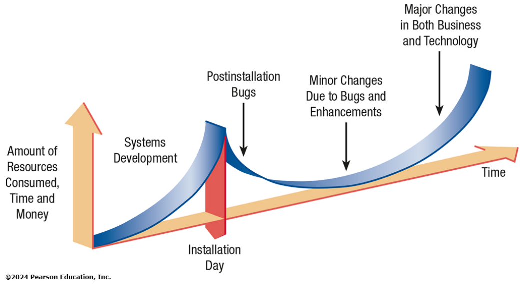
- **Phase 5:** Developing and Documenting Software
    - Analyst works with coders/developers
    - Activities
        - Develop/Code the actual software components
        - Create **Documentation** (User Manuals, Online Help, FAQs, Technical Docs) - *Should reflect user needs identified earlier!*
    - Quality: Code walkthroughs/reviews
- **Phase 6:** Testing and Maintaining the System
    - Goal: Find errors **BEFORE** the system goes live (much cheaper!)
    - Activities
        - Conduct various tests (unit, integration, system)
        - Use test data, then actual (anonymized) data
        - Follow pre-defined **Test Plans**
        - Begin **System Maintenance** activities and documentation updates
    - Who's Involved: Analysts, Coders/Developers, Testers, Users (for User Acceptance Testing - UAT)

SDLC Phase 7: Go Live and Look Back!

- Implementation Activities
  - User Training: Teaching users how to operate the new system (Analyst oversees)
  - System Conversion: Planning and executing the switch from the old system to the new one (e.g., parallel run, direct cutover, phased)
  - Data Conversion: Migrating data from old formats/systems
  - Installation: Setting up hardware, software
  - Go Live: The new system is officially launched!
- Evaluation
  - Occurs **throughout** the SDLC, not just a final step
  - Continues **after** implementation
  - Key Question: Is the system meeting objectives? Are users using it **effectively**?
- Reality Check: SDLC is often **Cyclical**. Discoveries might force revisiting earlier phases

After Launch: The Ongoing Cost and Effort of Maintenance

- The Reality: Maintenance consumes a **significant** portion of IT time and budget (Estimates range widely, but often $>$ 50%!)
- Why Maintain Systems?
  - Correct Errors: Fixing bugs missed during testing
  - Enhancements: Adapting the software due to:
    - New User Requests (after using the system)
    - Changes in the Business Environment
    - Changes in Technology (Hardware/Software updates)


Major Changes in Both Business and Technology
Postinstallation Bugs
Minor Changes Due to Bugs and Enhancements
Systems Development
Amount of Resources Consumed, Time and Money
Time
Installation Day
©2024 Pearson Education, Inc.

- Implication: Eventually, the cost of maintaining an old system outweighs the cost of developing a new one

Boosting Analyst Productivity: CASE Tools

- What: **C**omputer-**A**ided **S**oftware **E**ngineering Tools
- Purpose: To improve and automate tasks performed by Systems Analysts, especially those using structured methods like SDLC
- Key Benefits
  - Increased Productivity: Automates repetitive tasks (drawing diagrams, checking consistency)
  - Improved Communication: Visual models are easier for users to understand than text. Facilitates feedback.
  - Integration: Links different phases and outputs of the SDLC together
- Core Concept: The **CASE Repository**
  - A central **encyclopedia** storing *all* project information (diagrams, data definitions, screen/report layouts, project management details, etc.)
  - Enables consistency checks and report generation
- Example: Visible Analyst (Full CASE) vs. Microsoft Visio/OmniGraffle (Primarily Diagramming Tools)

Beyond the Waterfall: The Agile Approach

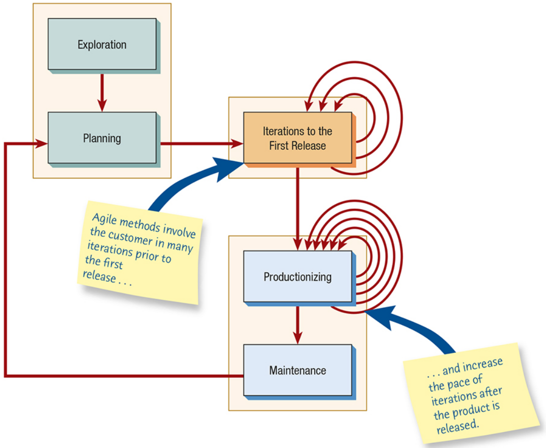
- What: A software development methodology based on:
    - Values: (Communication, Simplicity, Feedback, Courage) - *Good advice for ANY project!*
    - Principles
    - Core Practices
- Underlying Idea: An outgrowth of Object-Oriented Approaches
- Alternative/Supplement to SDLC: Used when flexibility is key, or traditional methods haven't fit
- PMI Terminology: An "Adaptive" Lifecycle (vs. SDLC's "Predictive") [1]
- Key Characteristics: Highly **Interactive** (constant communication) and **Incremental** (building in small pieces)

Agile in Practice: Flexibility and Teamwork

- Dynamic Resource Balancing: Agile methods often adjust **Time**, **Cost**, **Quality**, and **Scope** to meet goals
- Core Practices
  - Short Release Cycles
  - Sustainable Pace (e.g., 40-hour work week)
  - Onsite Customer (Direct, continuous user involvement)
  - Pair Programming
- Popular Agile Framework: Scrum
  - Named after a rugby formation (emphasizes teamwork)
  - Sprints: Short, fixed-duration work cycles (typically 2-4 weeks)
  - Goal per Sprint: Deliver a **potentially releasable** increment of the product
  - Team Empowerment: Teams often choose *how* to accomplish the work for a sprint

The Agile Journey: 5 Key Stages

- Overall Process: Characterized by frequent **iterations** and **feedback loops**
- Interesting Loops!
    - Iterations Loops and Maintenance to Planning Loops → Adaptive, incremental nature
    - Productionizing → Increased pace of iteration after the initial release


Exploration
Planning
Iterations to the First Release
Productionizing
Maintenance
Agile methods involve the customer in many iterations prior to the first release . . .
. . . and increase the pace of iterations after the product is released.

Getting Started with Agile: Exploration and Planning

- **Stage 1:** Exploration (Duration: Weeks to Months)
  - Goal: Understand the landscape; Confirm Agile is suitable
  - Activities: Assemble team and assess skills, Investigate potential technologies, Practice estimating task effort, **Customers** practice **writing** User Stories (informal feature descriptions)
  - Mindset: Be Playful, Curious!
- **Stage 2:** Planning (Duration: Typically Days!)
  - Goal: Agree on initial *scope* for the first major release and *target date* (e.g., 2-6 months out)
  - Focus: Tackle the **smallest** set of the **most** valuable features first
  - Uses “Planning Game” Metaphor (from Extreme Programming [2])
    - Goal: Maximize delivered Business Value
    - Strategy: Limit risk (simple design, early feedback, adapt quickly)
    - Pieces: User Story cards (features, notes, tracking)
    - Players: Development Team + Business Customer (sets priorities!)

The Cycle: Iterate, Produce, Maintain

- **Stage 3:** Iterations to First Releases (Cycles: ~3 weeks each)
  - Activities: Build selected features, Run customer tests *frequently*, Get feedback and adapt, Sketch out/refine system architecture incrementally
  - Important: **Celebrate** successful iterations (motivates!)
- **Stage 4:** Productionizing (Cycles: Faster, e.g., ~1 week)
  - Activities: **Release** the product! Implement faster feedback cycles (daily meeting?), Continue adding features based on feedback
  - Important: **Celebrate** the release (Development should be fun!)
- **Stage 5:** Maintenance
  - Activities: Keep the released system running smoothly, Continue adding new features (possibly riskier ones now), Adapt to changing needs, Rotate team members as needed
  - Mindset: Shifts to more **Conservative** ("Keeper of the flame") while still evolving

Thinking in Objects: Object-Oriented Analysis and Design

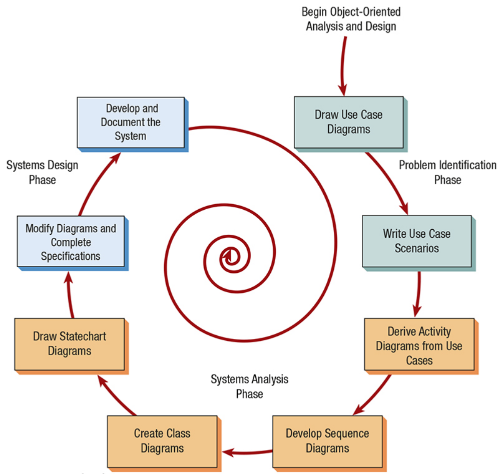
- What: An approach focused on **Objects** (representing real things/events like Customers, Orders, Products) grouped into **Classes** (defining shared attributes/behaviors)
    - Attributes: Characteristics
    - Methods: Things they do
- Contrast: Differs from traditional *procedural* programming (which focused on functions/procedures)
- Goal: Facilitate development and maintenance of systems needing **rapid change** and continuous adaptation, especially complex ones
- Key Benefit: Promotes **reuse** of components and improves system **maintainability**
- Standard Tool: Uses **Unified Modeling Language (UML)** for modeling O-O systems (e.g., Use Case Models)

O-O: Similar Phases, Different Rhythm

- Similarities to SDLC
  - Follows similar high-level phases (Problem Identification, Analysis, Design)
  - Uses rigorous, detailed modeling (UML)
- Pace: Because of the detailed modeling, the pace is often more deliberate than Agile (similar to SDLC in rigor)
- Key Difference: Iterative Nature within Phases
  - Often uses a **Spiral Model** approach:
    - Analyze $\rightarrow$ Design $\rightarrow$ Implement a *small part* of the system (e.g., a key feature)
    - Repeat (spiral outwards) for the next part
  - **Reworking** diagrams and components based on learning is **expected** and **normal**

The O-O Workflow with UML

- Understand user interactions
- Model the dynamic behavior
- Define the static structure
- Handle objects with complex lifecycle
- Refine and create detailed specifications
- Develop the system



Begin Object-Oriented Analysis and Design
Draw Use Case Diagrams
Problem Identification Phase
Write Use Case Scenarios
Derive Activity Diagrams from Use Cases
Develop Sequence Diagrams
Systems Analysis Phase
Create Class Diagrams
Draw Statechart Diagrams
Modify Diagrams and Complete Specifications
Systems Design Phase
Develop and Document the System

O-O Steps 1 and 2: Understanding Interactions

- **Step 1:** Define Use Case Model (Problem ID/Analysis)
  - Identify **Actors** (Users or other systems interacting with our system)
  - Identify Major Events/**Use Cases** (What actors accomplish, e..g, “Place Order,” “Register User”)
  - Draw **Use Case Diagrams**: Visual map of actors and their use cases
  - Write **Use Case Scenarios**: Textual step-by-step description of a typical interaction
- **Step 2:** Draw Initial UML Diagrams (Analysis)
  - Draw **Activity Diagrams**: Show workflow/steps within a single use case
  - Draw **Sequence Diagrams**: Show how different objects interact over time to fulfill a use case
- Iterative Nature: Creating Activity/Sequence diagrams often reveals need to refine/clarify Use Cases

O-O Steps 3 and 4: Classes and States

- **Step 3:** Develop Class Diagrams (Analysis)
  - Identify potential **Classes** (Tip: Look for important *nouns* in Use Cases and requirements
  - Define **Attributes** (data held by objects of the class) and **Methods** (actions objects can perform)
  - Show **Relationships** between classes (e.g., A Customer *has* Orders)
  - *This defines the static structure (blueprint) of the system*
- **Step 4:** Draw Statechart Diagrams (Analysis)
  - Model objects with complex lifecycles or **states**
  - Show possible **states** an object can be in (e.g., Order: Pending, Paid, Shipped)
  - Show **transitions** between states and the events triggering them
  - Helps refine understanding of object behavior and Class diagrams
- Iterative nature: Discoveries here can lead to refinement of other diagrams

O-O Steps 5 and 6: Refining Design and Building

- **Step 5:** Modify Diagrams and Complete Specifications (Design)
  - Refine UML diagrams (Class, Sequence, etc.) based on specific design decisions (technology choices, patterns, optimizations)
  - Write detailed **Class Specifications**: Precise descriptions of attributes, methods
  - Write detailed **Method Specifications**: Define input/output, internal processing logic for each method
- **Step 6:** Develop and Document the System (Development)
  - Translate the detailed models and specifications into working code
  - Documentation is CRITICAL: UML diagrams + Specifications are vital guides for the development team
  - *Good models and docs lead to faster development and more robust system*

Common Ground: More Similar Than Different?

- Shared Foundations (ALL Approaches Require)
  - Understanding the Organization: Business context, goals, problems → Chapter 2
  - Project Planning: Budgeting time and resources, Project Proposal → Chapter 3
  - Detailed Data Gathering: Interviews, Questionnaires, Observation, Sampling → Chapter 4 and 5
- Overlapping Characteristics
  - SDLC and O-O: Emphasis on detailed planning and diagramming
  - Agile and O-O: Facilitate building systems incrementally or subsystem-by-subsystem
  - Agile and SDLC: Consider the logical flow of data
- Key Takeaway: Strong foundational analysis skills (understanding needs, gathering data, planning) are **essential** regardless of the methodology used!

Making the Choice (1/3): When to Use SDLC?

- Consider the SDLC (Waterfall/Predictive) when
  - The organization has a history of using SDLC; existing systems used it (consistency)
  - Rigorous, step-by-step **documentation** is a high priority or requirement (e.g., compliance, regulation)
  - Upper management strongly **prefers** or feels more comfortable with a detailed, upfront **plan** and predictable phases
  - Sufficient **time** and **resources** are confidently available to complete the full cycle thoroughly
  - Formal **documentation** is the primary means required for communicating how the new system operates to stakeholders

Making the Choice (2/3): When to Use Agile?

- Consider Agile Methodologies (Adaptive/Iterative) when
  - An influential **project champion** who understands and advocates for Agile exists within the organization
  - The business environment is **dynamic** and requirements are likely to change or evolve rapidly. Need for **speed**
  - It's a **project rescue** situation – need to deliver value quickly, less focus on analyzing past failures in detail
  - The customer/users understand and value receiving working software in **small** and **incremental** improvements
  - Executives, analysts, and the development team are aligned with and support **Agile principles** (collaboration, adaptation, etc.)

Making the Choice (3/3): When to Use O-O?

- Consider Object-Oriented Analysis and Design when
  - The problem domain naturally lends itself to modeling with **Classes** and **Objects** (e.g., complex entities, simulations)
  - The organization supports the team in learning and effectively using **UML** modeling tools and techniques
  - The system can be realistically developed **incrementally**, perhaps one subsystem or major feature set at a time
  - There is a strong possibility or goal of achieving significant **code/component** reuse
  - It's acceptable (or desirable, e.g., for risk mitigation) to tackle the **most difficult** or **complex** parts first (aligns with spiral thinking)

Beyond Proprietary: Open Source Software (OSS)

- What: Software where the source code is publicly available for anyone to **study**, **share**, and **modify**
- Contrast: Opposite to **proprietary** software (where source code is hidden/secret)
- Core Principles: Collaboration, Shared contribution and benefits, Modifications typically shared back, Adherence to specific **open source licenses**
- Philosophy: Often viewed as a **communal** process and product, sometimes aimed at broader societal benefit
- Examples: Linux, Android, Apache Web Server, Mozilla Firefox Browser
- Growing Importance In: Cloud Computing, Big Data Analytics, AI/Machine Learning, Internet of Things (IoT), Cybersecurity, Blockchain

The OSS Ecosystem: Communities and Platforms

- Diverse Communities: OSS isn't one entity; communities vary (e.g., based on goals, structure: ad hoc, standardized, organized, commercial)
- Supporting Foundations: Provide crucial infrastructure, governance, legal support, event organization
  - Example: The Apache Software Foundation, The Linux Foundation (hosts 1000s of projects)
- Key Development Platform: GitHub (Owned by Microsoft)
  - Provides: Code hosting (using **Git**), Collaboration tools, Issue tracking, Developer profiles and networking
- Underlying Technology: Git
  - The most widely used open source **version control system**. Essential for tracking changes to code files over time, especially in collaborative projects

Bridging Worlds: Corporations and OSS

- Historical Divide
  - Corporations $\rightarrow$ **Proprietary** code (Guarded for competitive advantage)
  - OSS Communities $\rightarrow$ **Shared** code and community values
- The Modern Reality: Collaboration
  - Companies increasingly participate in and contribute to OSS
- "The Third Design Space" [3]
  - A **metaphorical space** where corporate developers and community developers collaborate
  - Creates: New shared design environments, resources, associations
  - Results In: Innovative shared software, new development processes (blending corporate needs and community practices), developers skilled in both worlds
  - Enables: Innovations potentially not achievable in purely commercial or purely community settings alone

Corporate Motivations: Why Participate in OSS?

- Research identified multiple drivers [4]

- Rational Reasons (Business Logic)
  - Save Money/Reduce Development Cost (Leverage existing code)
  - Less Maintenance Burden (Shared effort)
  - Contribute Within Limits (Influence project direction strategically)
  - Reduce Long-Term Costs
  - Marketing Benefits (Enhanced reputation, attract talent)
  - make the First Move (Gain strategic advantage)

- Emotional Reasons (Cultural/Intrinsic)
  - Accept Responsibility/Give Back to community
  - Improve Shared Software (Make tools they rely on better)
  - Gain Community Influence/Respect
  - Relinquish Traditional Gatekeeper Role (Embrace openness)
  - Improve Developer's Skills and Morale (Exposure, learning)
  - Extend Life/Relevance of Internal Projects

The Rules of the Road: OSS Licensing and Compliance

- Licenses are CRITICAL: They define permissions, obligations, and restrictions for using, modifying, and distributing OSS code
- Community Consensus: Licenses arise from and are chosen by specific OSS communities
- Popular Examples: Apache Licence 2.0, GNU General Public License (GPL - v2, v3, LGPL), MIT License, BSD Licenses, Mozilla Public License 2.0
- Your Responsibility: When *using* OSS components, you **must** identify their licenses and **ensure compliance**. Failure can have legal consequences!
- Compliance Tools and Standards
    - Software Package Data Exchange (SPDX): A standard format for communicating license information clearly (Linux Foundation WG)
    - FOSSology: An open-source tool/toolkit (Linux Foundation Project) to scan codebases, identify licenses and copyrights, and help manage compliance workflows

OSS on Your Resume: Skills and Opportunities

- Highly Valued: Experience contributing to or effectively using OSS is a **marketable** skill
- Talent Shortage: Over 90% of hiring managers report difficulty finding sufficient open-source talent [5]
- Employer Interest
  - Many actively seek *certified* OSS professionals
  - Many are willing to *pay for employees* to get relevant OSS certifications
- How to Get Involved
  - Create a **GitHub** profile; contribute (even documentation!) to projects you use
  - Explore projects hosted by **The Linux Foundation** or others
  - Inquire about **employer's** open source programs or contribution policies
- Benefits: Demonstrates technical ability, collaboration skills, continuous learning, and engagement with modern development

The Analyst and Open Source

- Why Might You Be Involved?
  - Your employer may **request participation** in relevant OSS communities ($>$ 90% org reliance on OSS creates need for awareness/engagement)
  - Curiosity/“Bandwagon Effect”: To understand what competitors are doing or explore potential OSS benefits
- Strategic Involvement: “Responsive Design”
  - Analyst acts as a bridge between OSS community and employer
  - Activities
    - Participate in relevant OSS projects
    - Understand community designs, practices, directions
    - Identify opportunities to **incorporate** or **adapt** valuable OSS ideas, components, or practices into the company’s *proprietary* systems or products
  - Goal: Blend external innovation with internal strategic objectives
- Example: NASA hosts open source projects, demonstrating OSS use even in highly specialized domains [6]

References I

[1] Project Management Institute, *A Guide to the Project Management Body of Knowledge (PMBOK® Guide) – Seventh Edition and The Standard for Project Management*, 7th. Newtown Square, PA, USA: Project Management Institute, 2021 (cit. on pp. 11, 20).

[2] K. Beck and C. Andres, *Extreme Programming Explained: Embrace Change*, 2nd. Boston, MA: Addison-Wesley Professional, 2004 (cit. on p. 23).

[3] K. E. Kendall, J. E. Kendall, M. Germonprez, and L. Mathiassen, “The Third Design Space: A postcolonial perspective on corporate engagement with open source software communities,” *Information Systems Journal*, vol. 30, no. 2, pp. 369–402, 2020. DOI: 10.1111/isj.12270. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/isj.12270 (cit. on p. 37).

[4] J. E. Kendall, K. E. Kendall, and M. Germonprez, “Game theory and open source contribution: Rationale behind corporate participation in open source software development,” *Journal of Organizational Computing and Electronic Commerce*, vol. 26, no. 4, pp. 323–343, 2016. DOI: 10.1080/10919392.2016.1228360. [Online]. Available: https://www.tandfonline.com/doi/full/10.1080/10919392.2016.1228360 (cit. on p. 38).

References II

[5] The Linux Foundation Research Team, "The 10th Annual Open Source Jobs Report," The Linux Foundation Research Team, Tech. Rep. 10, 2022. DOI: 10.70828/RZRE1873. [Online]. Available: https://www.linuxfoundation.org/research/the-10th-annual-open-source-jobs-report (cit. on p. 40).

[6] NASA, *NASA Open Source Software*, https://code.nasa.gov/, A catalog of open source software released by NASA, 2012 (cit. on p. 41).