Software Engineering and ASE Courses

How to Help Students Becoming Professional Problem Solver

ASE

ASE @ Northern Kentucky University

Meet Tim and Jane

Tim is a student at NKU who loves building and creating things. He's curious about how things work but hasn't built any software applications yet. He's excited to learn but a bit nervous about starting his software engineering journey.

Tim

"Hello, I'm Tim! I enjoy making things and figuring out how they work. I've always been curious about building software, but I'm just starting out."

Jane is also a student at NKU, but she has a different background. She loves solving puzzles and has already created several small software tools to help solve problems. For her, writing software is like solving an interesting puzzle.

Jane

"Hi, I'm Jane! I love math and solving puzzles—it's my favorite thing to do. When I discovered programming, I realized that building software is just another kind of puzzle, which makes it really fun for me!"

Throughout this story, you'll follow Tim and Jane as they learn about software engineering together. Tim represents students who are just starting their journey, while Jane represents those who already have

some programming experience. Whether you're more like Tim or Jane, you'll find valuable lessons in their experiences.

Chapter 1:
Welcome to Software Engineering

- We learn best by actually doing things and making mistakes along the way—mistakes help us grow.
- Software Engineering is about solving real problems in the real world using proven methods (rules), helpful technologies (tools), and working with others (teams).
- Software Engineers are professional problem solvers who create quality products that real people use and value.
- Success starts with believing in yourself: you can achieve anything if you commit to it and put in the necessary effort.

The First Day

Tim nervously walks into his first ASE course. He's excited but not sure what to expect. Prof. Cho starts the class.

Prof. Cho

"Welcome to Software Engineering! Let me begin with an important truth: the best way to learn is by doing."

Tim

"That makes sense. Like the saying 'practice makes perfect,' right?"

Prof. Cho

"Actually, it's more than just practice. We learn by doing things, and even more importantly, by making mistakes. Smart people also learn from watching others make mistakes, which helps them avoid the same pitfalls."

Tim thinks

"Wait, so making mistakes is actually a good thing? That takes some pressure off!"

Prof. Cho

"Think of your brain like a muscle at the gym. If you don't exercise it, it gets weaker. The more you use it to solve problems, the stronger and better it becomes."

Tim thinks

"Maybe that's what **Software Engineering** *is all about—learning by actually doing things, not just reading about them."*

Prof. Cho

"Here's an important mindset: As long as we learn from our mistakes and create systems to prevent making the same mistakes again, there are no real failures—only learning experiences."

What is Software Engineering?

Tim

"Professor, what exactly is Software Engineering? Is it the same as coding or programming?"

Prof. Cho

"Great question, Tim! Software Engineering is about solving problems in effective and efficient ways. The important thing to understand is that the problems we solve are real-world problems, not just computer puzzles."

Tim

"So there's more to it than just writing code?"

Prof. Cho

"Exactly! In real-world Software Engineering, we solve problems using three essential elements: Rules, Tools, and Teams. Let me explain each one."

Tim pulls out his notebook and starts writing carefully.

Prof. Cho

"Rules are the principles and patterns—these are proven approaches to solving problems that others have discovered over many years."

Prof. Cho

"Tools are anything that help us solve problems. They include design methods, programming languages, frameworks, IDEs, and testing software."

Prof. Cho

"Teams means working with other people, communicating well, and collaborating effectively."

Jane

"Oh, I get it! It's like having a toolbox full of useful tools, an instruction manual with proven methods, and teammates to work with!"

Prof. Cho

"That's a perfect analogy, Jane! And here's what makes software engineers special: Software Engineers are professional problem solvers, not just people who write code or programmers."

Tim thinks

"So being a software engineer is bigger than just knowing how to code. It's about solving real problems professionally."

Managing Complexity

Tim

"Professor, I've noticed many software applications have lots of bugs and problems. Why does this happen?"

Jane

"Yes, I've heard about many software projects that failed or became disasters. It seems to happen a lot."

Prof. Cho

"The main reason is the inherent complexity that comes with software engineering. It's one of the biggest challenges we face."

Tim

"What do you mean by complexity?"

Prof. Cho

"Software products are among the most complex things humans have ever created. They have millions of parts that all need to work together perfectly."

Jane

"So, software engineering is about solving problems, but software engineers also have to deal with this huge complexity?"

Prof. Cho

"Exactly right! Software engineering is about managing this complexity while delivering software products. And software engineers are problem solvers who succeed by learning to manage this complexity effectively."

Tim thinks

"I never thought about it that way. No wonder it's challenging—but it also sounds exciting!"

Professionals vs Amateurs

Prof. Cho

"Before we continue, let me ask you an interesting question: 'We are software professionals because we love software.' Is this statement True or False?"

Tim

"That must be True! I mean, if you don't love software, why would you become a software engineer?"

Jane

"Hmm, it seems too obvious. I think it's a trick question, so I'll guess False."

Prof. Cho

"Good thinking, Jane! The answer is actually NO. Here's why: by definition, when we do something because we love it, we're called 'amateurs.' The word 'amateur' comes from Latin 'amator,' which means 'lover.'"

Tim

"WHAT?! That's surprising! Then what's the real definition of a professional?"

Prof. Cho

"A professional is someone who makes money from doing something. Think about it: professional baseball players get paid to play, while amateur players actually spend their own money to play baseball games. That's the key difference."

The Key Distinction

Jane

"So it doesn't actually matter whether I like software or not to be called a professional?"

Prof. Cho

"To be called a professional, the key requirement is that you earn money from it. But here's the important part—to earn money consistently and successfully, you must become an expert at what you do."

He writes on the whiteboard.

AMATEUR SOFTWARE DEVELOPER:

- Writes code for fun on weekends
- Works on problems that interest them personally
- Can abandon projects anytime without consequences
- No strict deadlines or pressure from clients

→ Does it for LOVE and ENJOYMENT

PROFESSIONAL SOFTWARE ENGINEER:

- Writes code to solve clients' real problems
- Works on required projects (whether fun or not)
- Must finish and support what they build
- Meets deadlines and stakeholder expectations

→ Does it for MONEY and RESPONSIBILITY

Tim thinks

"I see! Being a professional means taking on more responsibility, not just doing what you enjoy."

Professional Standards in Practice

Jane

"So being a professional means more responsibility and commitment?"

Prof. Cho

"Exactly! Let me give you a concrete example from the real world:"

THE BORING CRUD APPLICATION EXAMPLE:

*Amateur Thinking: "CRUD operations are boring and repetitive. I want to build something exciting with AI instead!" → Quits the boring project and starts a new **"exciting"** one*

*Professional Thinking: "This CRUD app processes $2 million in daily transactions. Even though it's not glamorous, I'll make it reliable, secure, and maintainable for our users." → Delivers excellent work even on **"boring"** projects*

Prof. Cho

"Professionals deliver value by solving problems well, regardless of whether the work personally excites them. That's a crucial difference."

Jane

"But Professor, I really do want to be a professional software engineer because I love problem-solving!"

Prof. Cho

"That's wonderful, Jane! We should aim to be professionals who love what we do. But to make that possible, we must also become truly competent and skilled at our work—and that requires consistent effort and training."

Tim

"I understand now. It's hard to genuinely love work that we're not good at. We need the skills first."

Prof. Cho

"Exactly, Tim! Competence and passion work together to create a fulfilling career."

Professional Problem Solvers

Prof. Cho

"The first step to becoming a competent problem solver, I believe, is identifying ourselves as software engineers, not just coders."

Jane

"What's the real difference between a coder and a ***Software Engineer***?"

Prof. Cho

"A coder writes code to make things work. A Software Engineer solves problems systematically, works effectively in teams, and delivers real, high-quality products that people actually buy to use."

Tim

"I'm so tired of using buggy, low-quality software! I once lost days of work because of a silly bug in a tool I was using. It was devastating."

Prof. Cho

"That's exactly the point! High-quality products are ones that people actually use, need, and are willing to pay for. True quality isn't just about code that works—it's about solving real problems reliably."

Tim nods thoughtfully, beginning to see the bigger picture of what software engineering really means.

Prof. Cho

"Becoming a competent problem solver is a huge challenge, but we can definitely do it once we understand the secrets that successful problem solvers use."

Tim thinks

"Secrets of successful problem solvers? Now I'm really curious!"

Henry Ford - The Great Problem Solver

Prof. Cho displays a famous quote on the screen.

Prof. Cho

"Henry Ford, who revolutionized car manufacturing, once said: 'Whether you think you can, or you think you can't—you're right.'"

Prof. Cho

"Henry Ford, who revolutionized car manufacturing, once said: *'Whether you think you can or you cannot—you're right.'*"

Tim

"That's really inspiring! So it's all about having the right mindset?"

Prof. Cho

"Success begins with believing you can do it, then taking action. Do it now and continuously make it better. You can accomplish anything—if you decide to do it and are willing to pay the price, I mean, put in the necessary effort."

Tim thinks

"What's my decision? Am I really willing to pay the price? Honestly, I'm not completely sure yet..."

Prof. Cho

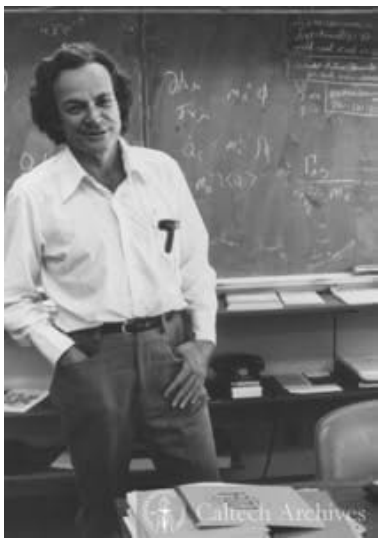
"Ford also taught us about systems and processes. His advice: Don't rely on random habits or whims. Instead, (1) use a systematic approach, (2) continuously improve your methods, (3) simplify and automate everything possible, and (4) make decisions based on clear rules and solid data.[1]"

Tim thinks

"Wow! It's amazing that he figured out all these principles more than a hundred years ago! These ideas still apply today."

[1]Henry Ford revolutionized manufacturing by creating the assembly line—a systematic process that made cars affordable for ordinary people. His focus on continuous improvement and simplification transformed not just car manufacturing, but how we think about production systems in every industry.

Feynman's Secret to Problem Solving



Prof. Cho

"Now let me share another powerful quote, this time from physicist Richard Feynman: '*What I cannot create, I do not understand.*'"

Jane

"So just reading about something or watching videos isn't enough to truly understand it?"

Prof. Cho

"Exactly! Making things with your own hands is the only path to **true, deep understanding**, especially in Software Engineering. The sooner you realize this fundamental truth, the sooner you'll succeed."

Jane

"How did Feynman approach problem-solving?"

Prof. Cho

"He had a deceptively simple three-step technique: First, write down the problem clearly. Second, think very hard about it. Third, write down the solution."

Tim

"Wait, that seems… almost like a joke!"

Prof. Cho

"It might sound simple, but his point is actually profound: (1) If you can't clearly define and write down the problem, you can't solve it effectively. And (2) thinking hard is nearly impossible when you don't clearly understand what you're supposed to solve."

Prof. Cho

"Richard Feynman was a Nobel Prize-winning physicist famous for his ability to explain incredibly complex concepts in simple, understandable terms."

Prof. Cho

"He believed that if you couldn't explain something simply to five-years old, you didn't truly understand it yourself."

Prof. Cho

"This principle applies perfectly to software engineering—if you can't clearly define a problem, your solution will be equally unclear."

Jane

"I think I'm starting to understand why both Henry Ford and Richard Feynman were such great problem solvers."

Tim

"Me too! System thinking, learning by actually doing things, clearly defining problems before solving them... these are such important ideas."

Tim's Realization

After class ends, Tim sits quietly and thinks about everything he learned.

> ***Tim thinks***
>
> *"So I need to actually build things myself to truly understand them. Just reading tutorials or watching YouTube clips isn't enough."*

> ***Tim thinks***
>
> *"And I need to start early, because I'll only discover what I don't know by actually doing it and running into problems."*

> ***Tim thinks***
>
> *"Prof. Cho said that real learning begins when I build something myself. I guess that means I should stop waiting for the 'perfect' time to start—because the perfect time is now."*

The next day, Tim arrives at class with a completely different attitude.

Tim

"Professor, you said we learn by making mistakes. So when should I actually start my project?"

Prof. Cho

"Start now. Begin immediately. Learning takes time, and you need to discover what you don't know as soon as possible. That's one of the real secrets to success."

Tim

"But what if I fail or make mistakes?"

Prof. Cho

"Remember Henry Ford's wisdom: Whether you think you can or can't, you're right—so choose to believe you can! And remember Feynman's approach: Start by clearly defining the problem. Once you understand the problem well, the solution will naturally follow."

Tim smiles broadly. He's finally beginning to understand what it really means to be a Software Engineer.

Tim thinks

"I'm ready to start my journey. Time to build something!"

Tim thinks

"I know my first apps may not that good in quality, but I'll learn from my mistakes, and keep building. And I beleive I can make high-quality software products that solve people's problems and people are willing to pay to use!"

Chapter 2:
The ASE Program Architecture

- ASE students learn to build high-quality applications that solve real software engineering problems.
- The 4D process provides a clear roadmap for problem-solving: Define, Design, Develop, and Deploy.
- ASE students master three key areas (APT): building Applications, applying Processes, and using Tools effectively.
- Each ASE course focuses on different aspects while working together to build comprehensive and high-quality applications in a team and independently.

The Big Picture

In the next lecture, Prof. Cho begins to show students how all the ASE courses connect and work together.

Prof. Cho

"Today I want to show you the big picture of the entire ASE program. Think of it as a roadmap for your journey from beginner to professional software engineer."

Jane

"So there's actually a master plan behind all the courses?"

Prof. Cho

"Exactly! The goal of the ASE program is clear but ambitious: you will learn to build high-quality, professional-level software applications that can solve any given software engineering problem."

Jane

"Any problem? That sounds... really challenging."

Prof. Cho

"It is ambitious, you're right. But we have a systematic, step-by-step approach to help you get there. Let me introduce you to what we call the 4D process."

Tim thinks

"A process to solve any problem? This sounds interesting!"

The 4D Process

Prof. Cho draws four boxes on the whiteboard.

Prof. Cho

"4D is our main process for software engineering problem-solving. It has four clear steps: **Define, Design, Develop**, and **Deploy**."

Tim

"Is it like following a recipe when cooking?"

Prof. Cho

"It's similar, but more flexible! First, you **Define** the problem you're trying to solve and propose a solution. Don't worry about getting it perfect—you can revise your solution multiple times. The important part is understanding exactly what problem you're solving."

Jane

"Then we **Design** the architecture of our application?"

Prof. Cho

"Yes! Second, you **Design** the architecture of your application. This means planning how all the pieces will fit together and work with each other."

Prof. Cho

"Third, you develop, I mean build, the application based on your design—this is where you actually create it."

Prof. Cho

"Finally, you deploy the application so real people can actually use it."

Jane thinks

"Define → Design → Develop → Deploy. When you break it down like this, it sounds logical and manageable."

Tim

"To be honest, I'm not exactly sure how to do all of that yet."

Prof. Cho

"Don't worry at all! You'll learn everything necessary by actually doing projects step by step. After working through projects using this process, you'll become a master of the 4D approach sooner than you think."

Jane thinks

"Okay, once again it comes back to learning by doing. I'm starting to see a pattern here!"

Jane

"I'm getting excited about the projects I'm going to do!"

Tim

"Me too! This actually sounds fun."

APT: What We Learn to Master

Prof. Cho

"Now, let me tell you about the three things ASE students will master by the end of the program. We call it APT, which is easy to remember."

Tim

"APT? What does each letter stand for?"

Prof. Cho

"A stands for **Applications**—you'll learn how to build high-quality applications that work well and solve real problems."

Prof. Cho

"P stands for **Processes**—you'll learn how to apply software engineering processes effectively."

Prof. Cho

"T stands for **Tools**—you'll learn how to use modern software development tools like a professional."

Jane

"So we learn to build **Applications** using good **Processes** and the right **Tools**?"

Prof. Cho

"Exactly right! And here's an important part: ASE students can complete tasks both independently on their own and collaboratively with teams. You'll be ready and confident in any work situation."

Tim thinks

"Applications, Processes, Tools... A-P-T. I need to remember this!"

The Journey Through ASE Courses

Prof. Cho

"Let me show you how each course builds on the previous ones. The journey starts with ASE 220."

Tim

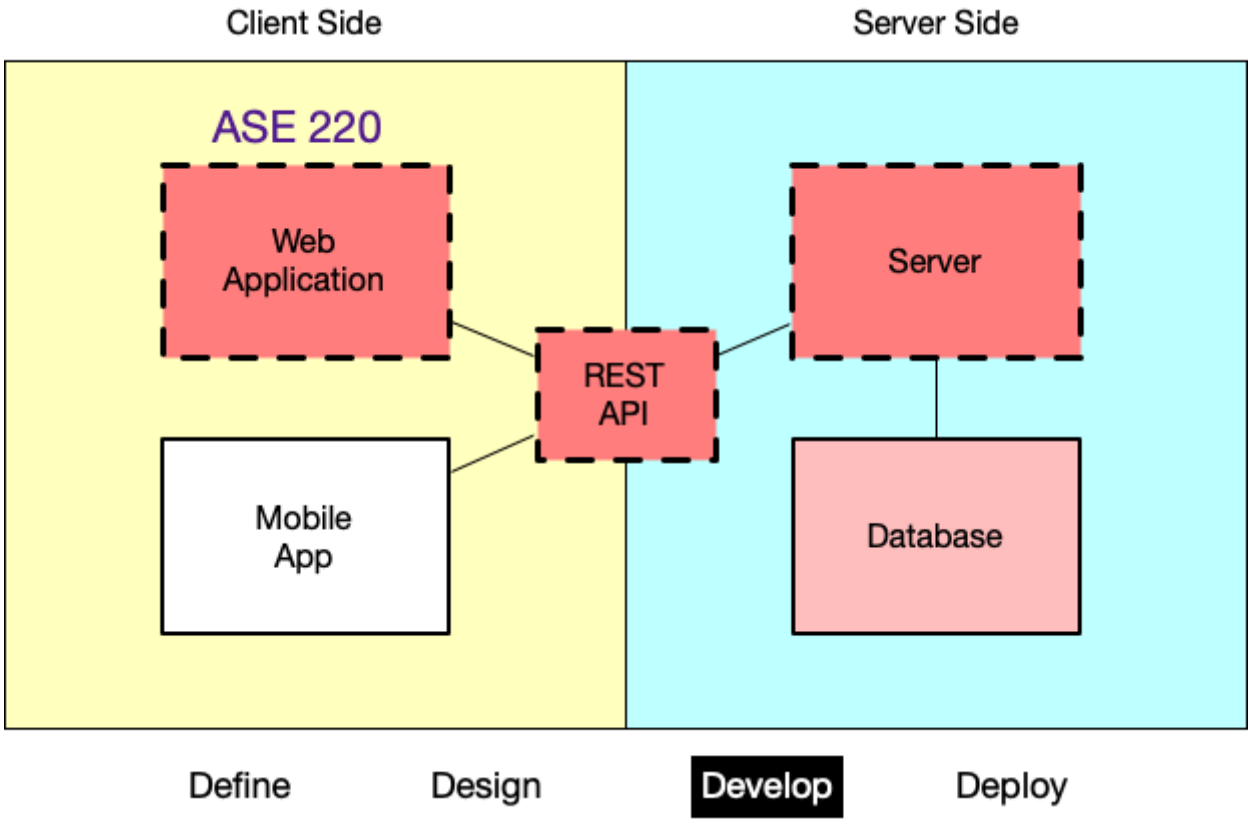
"What do we learn in ASE 220?"

Prof. Cho

"ASE 220 is Full Stack Application Development. You'll learn to build complete web applications with both the frontend (what users see) and backend (the server and database). It's all about understanding the entire picture of how web applications work."[2]

Prof. Cho shows the diagram on the screen.

[2]Depending on the instructors, the ASE course content might be different, but the overall structure should be the same.



Prof. Cho

"ASE 220 focuses on the **Develop** process, and students build applications using web, REST API, and server technologies."

Tim

"I don't really know what a REST API or a server is—it all sounds unfamiliar to me."

Prof. Cho

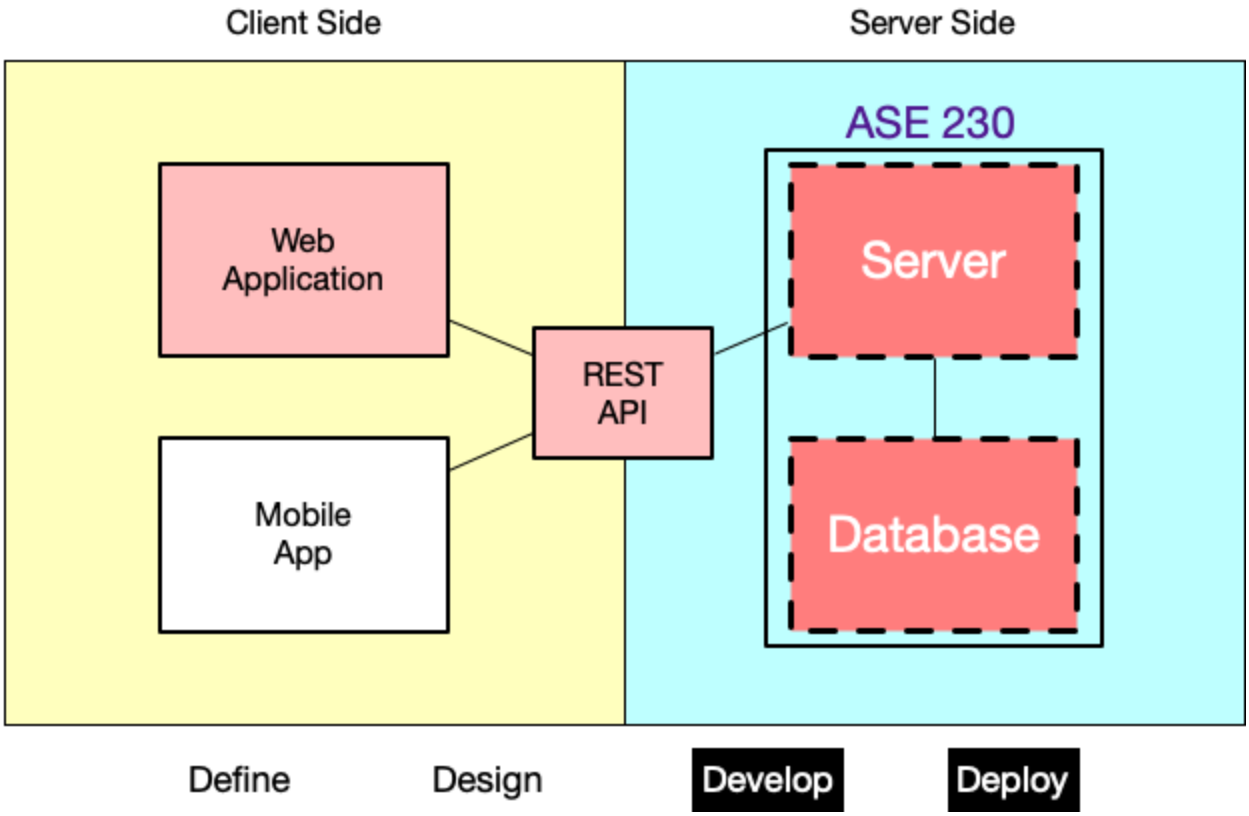
"That's completely okay. You don't need to worry about the technical details right now. With time and practice, you'll become comfortable with—and eventually master—these technologies."

Jane

"What comes after ASE 220?"

Prof. Cho

"ASE 230 focuses specifically on Server-Side Application Development. You'll dive much deeper into the backend—working with databases, building APIs, and creating server logic. You'll learn how to make the powerful engine that runs behind applications."



Tim thinks

"So we start with the full picture in 220, then specialize in specific areas afterward. That makes sense!"

Jane

"I notice that this course focuses not only 'Develop' but also 'Deploy'"

Prof. Cho

"Good catch! Yes, in this course we also learn how server applications are deployed automatically and systematically on the cloud or on real servers."

Corner Stone: ASE 285

Prof. Cho

"Then comes ASE 285: Introduction to Software Engineering and Security. This course has three important aspects: team projects, software engineering tools, and security."

Jane

"Is this our first time working on a real team project?"

Prof. Cho

"Yes! Even though you complete projects in other courses, those are mainly individual projects where you work alone. ASE 285 is where you truly learn how to work effectively on a team project with other students."

Tim

"That's exciting! Also, security is really important these days, right?"

Prof. Cho

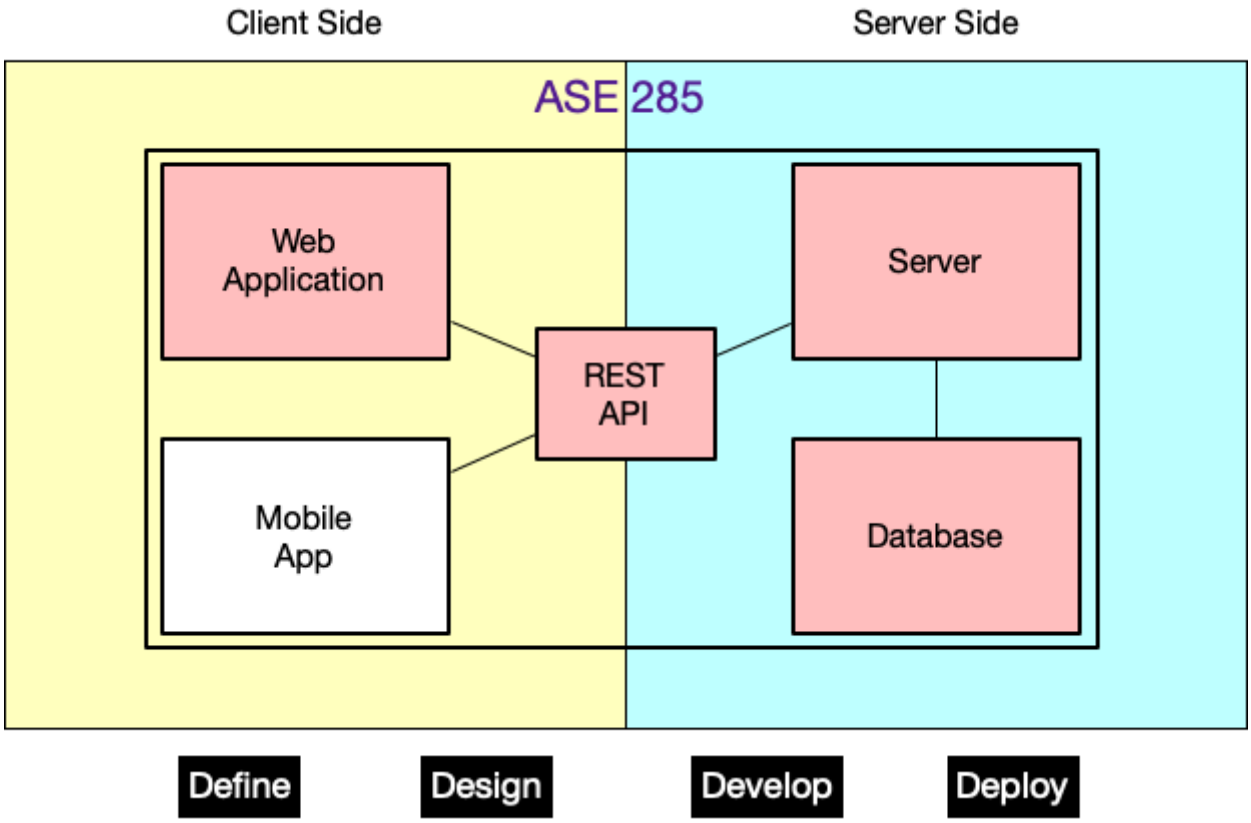
"Absolutely! Without proper security, your application is vulnerable to attacks and problems. ASE 285 teaches you to think like both a builder who creates applications and a protector who keeps them safe."

Jane

"I can see why ASE 285 is considered the cornerstone course of the entire ASE program."

Prof. Cho

"Yes, because it teaches most of the core software engineering topics—teams, tools, and security."



Tim thinks

"A cornerstone is the most important stone in a building. So this must be a really crucial course!"

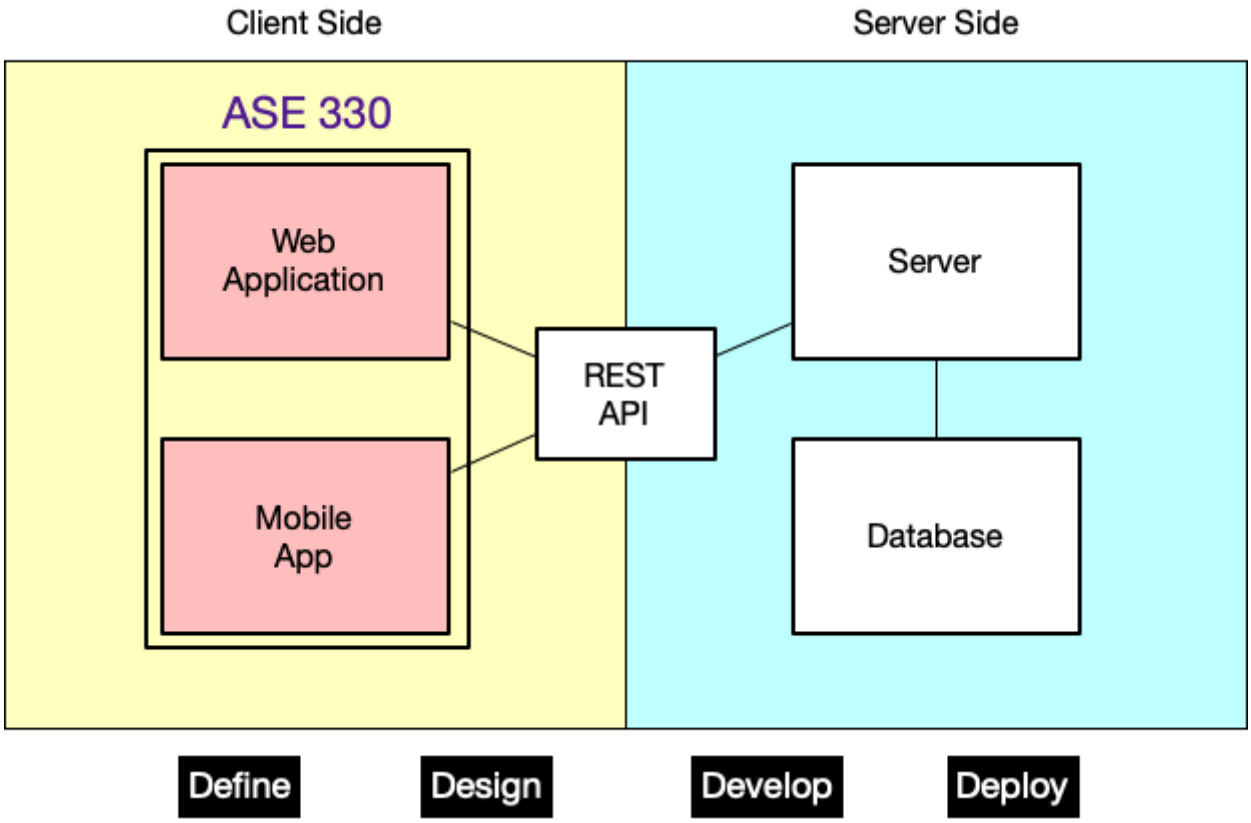
User Interface/User Experience

Jane

"What about learning to make applications that users actually enjoy using?"

Prof. Cho

"Great question, Jane! ASE 330, Human Computer Interaction (HCI), covers UI/UX Design in depth. You'll learn that effective, intuitive interfaces aren't just 'nice to have'—they're absolutely essential. Users simply won't use applications they can't understand or that frustrate them."



Tim thinks

"I've definitely stopped using apps that were confusing or hard to use. Now I'll learn how to make apps that people actually want to use!"

Software Design and Architecture

Prof. Cho

"ASE 420 is dedicated to Software Design. This is where you learn the principles of good software design—things like design patterns, SOLID principles, and how to structure code that lasts for years."

Jane thinks

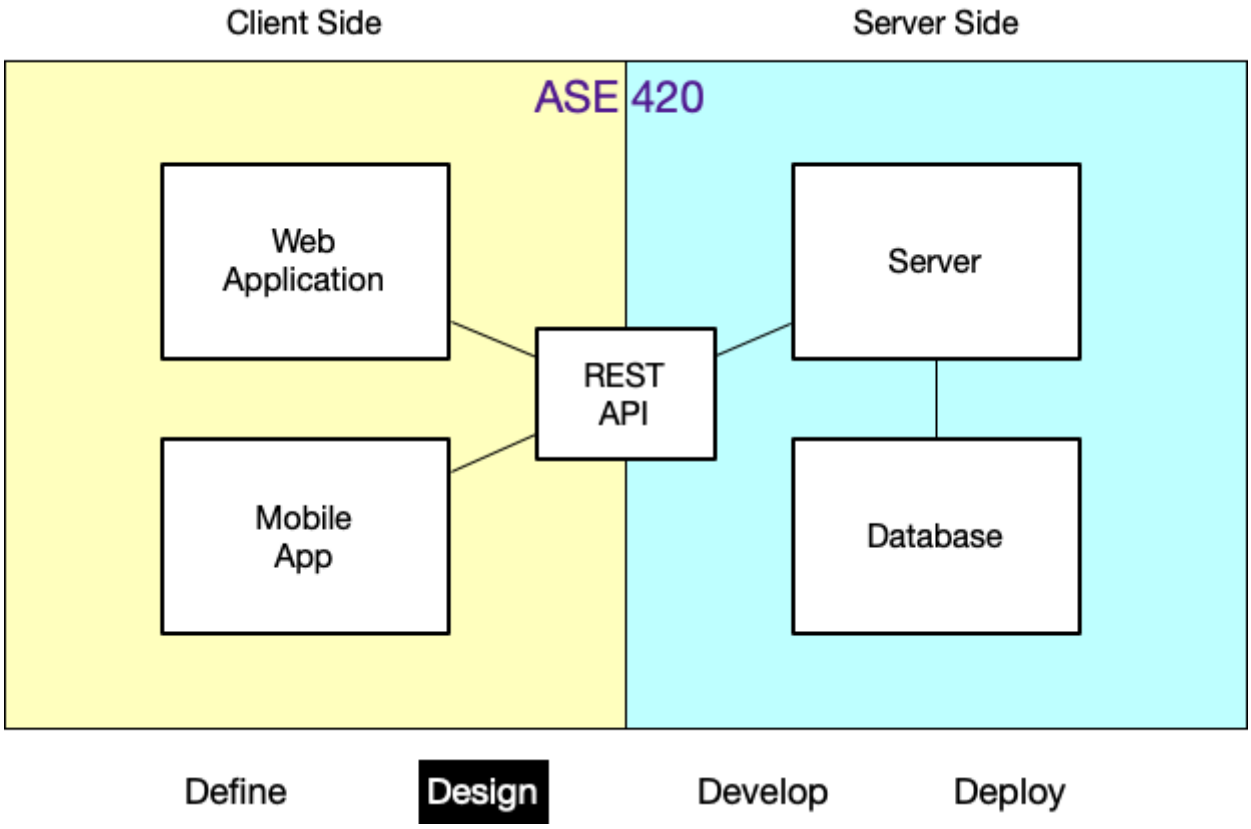
"Wow, there's an entire course just for software design! I guess software design must be incredibly important!"

Prof. Cho

"After successfully finishing this course, you'll be able to recognize good or bad software design when you see it."

Prof. Cho

"You can also read and write professional design diagrams called UML. In other words, you'll truly become a professional-level software engineer."



Tim

"What about software architecture? Is that different from design?"

Prof. Cho

"Yes, and ASE 456: Cross-Platform Development is where you learn about software architecture."

Prof. Cho

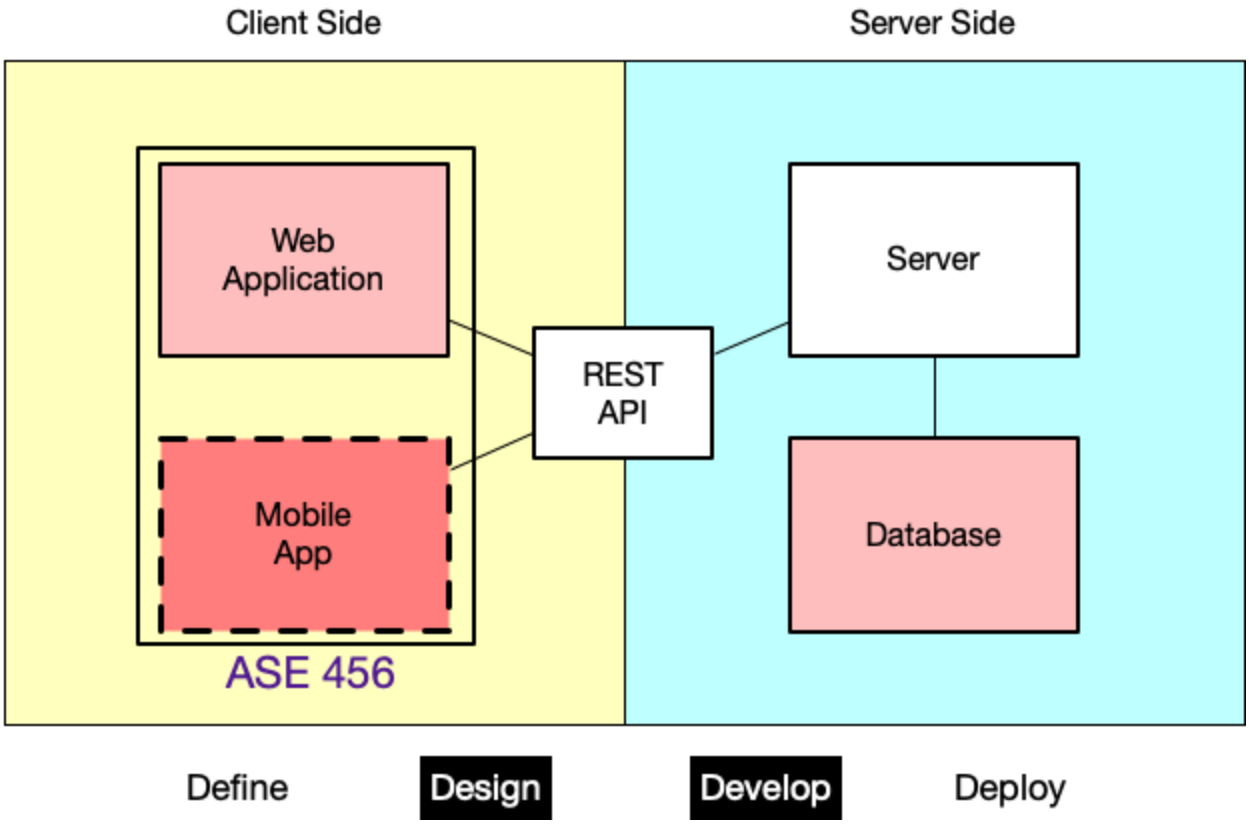
"You'll learn software architecture by building cross-platform applications—applications that work on phones, tablets, desktop computers, and web browsers, all from a single codebase."

Jane

"That sounds super efficient! Writing code once and having it work everywhere!"

Tim

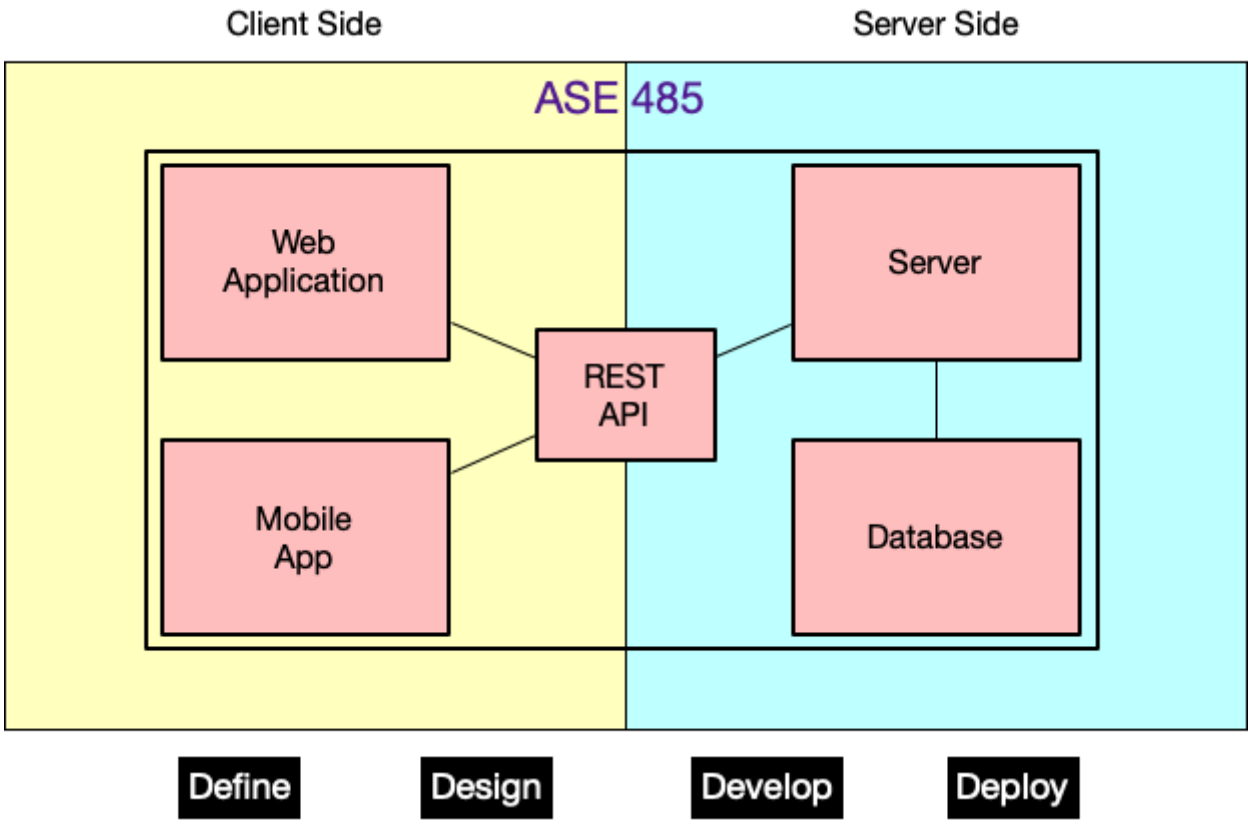
"And really fun! I've always wanted to build Android apps, and now I can build apps for iOS too!"


Client Side
Server Side
Web Application
Mobile App
ASE 456
REST API
Server
Database
Define
Design
Develop
Deploy

Capstone: ASE 485

Prof. Cho

"It is fun and also rewarding! And finally, ASE 485 is your Capstone course. This is where you pull everything you've learned together into one major project."



Tim's and Jane's Vision

After class, Tim and Jane discuss what they learned.

Tim

"So each course teaches us a different piece of the puzzle?"

Jane

"I think so. ASE 220 gives us the complete picture of full-stack development, then each following course lets us go deeper and specialize in different important areas."

Tim

"But they all connect back to the 4D process (Define, Design, Develop, Deploy) and APT (Applications, Processes, Tools)?"

Jane

"Yeah. Whether we're building mobile apps, designing databases, or creating user interfaces, we're always using the same fundamental approach and framework."

Tim thinks

"I can finally see the path ahead clearly. Each course is a stepping stone that builds on the previous one, not separate, disconnected topics."

Tim

"I'm starting to really understand the big picture. It's not just about learning random tools—it's about becoming a complete, well-rounded problem solver."

Jane

"I think so too. And that's actually really exciting!"

Tim thinks

"The journey is long, but now I can see where I'm going and how each step helps me get there."

Chapter 3:
P4M4: The Course Structure

- Dr. Cho's ASE courses build real applications through an integrated curriculum where you discover theory through hands-on practice.
- The M4 learning levels represent your journey: Magic (confused) → Machine (understanding) → Master (skilled) → Make (creating).
- The P4 activities form the core of learning: Principles, Patterns, Practices, and Projects.
- Each course follows three phases: Introduction & Setup, Learning Tools for the Project, and Completing the Project.

Four Key Characteristics

Prof. Cho

"Now, let me explain how my ASE courses are specifically structured. There are four key characteristics that make them different."

Tim

"What makes these courses special or different from other courses?"

Prof. Cho

"First, you build real, working applications—not just toy examples or simple demos, but actual software that works."

Prof. Cho

"Second, the curriculum is integrated, meaning core important ideas are revisited across different courses from different angles. This helps you understand them more deeply each time."

Prof. Cho

"Third, theory emerges from practice—you discover principles and patterns through hands-on experience, not just by memorizing textbook definitions."

Tim

"So we learn by doing first, then understand the theory behind it?"

Prof. Cho

"Exactly! And fourth, we solve problems using proven methods. You'll use time-tested, reliable tools and techniques that professionals actually use, which will make you a systematic and confident problem solver."

Tim thinks

"Real applications, integrated learning, practice first, proven methods. These all sound really practical!"

M4 Learning Journey

Prof. Cho

"Learning happens naturally in four stages. We call them M4, which stands for: Magic, Machine, Master, and Make."

Tim

"Magic? That's an interesting way to describe learning!"

Prof. Cho

"M1 is the Magic stage. At first, when you're a complete beginner, everything feels like magic—mysterious, confusing, and incomprehensible. You run some code, and it works, but you have no idea why or how it actually works."

Jane

"That's exactly how I felt when I first started learning to code! Everything seemed magical and confusing."

Prof. Cho

"M2 is the Machine stage. As you learn the theory and practice more, things transform from magic into systems you actually understand. The mysterious magic fades away and is replaced by real comprehension of how things actually work."

Tim

"So we go from being confused to having clarity... that makes sense!"

Jane

"It's like that 'aha!' moment when everything suddenly clicks!"

Prof. Cho

"Exactly! And here's an important mindset: We should never be afraid of anything in software engineering. Just start by making something—anything! Begin by thinking 'for now, it's magic and I don't understand it, but I know that it's really just a machine that I can learn to understand.'"

Tim thinks

"That's actually really reassuring. I don't need to understand everything before I start!"

Mastery and Creation

Prof. Cho

"M3 is the Master stage. With lots of practice and experience, you gain real mastery. You can solve problems efficiently and elegantly. You know not just how to do something, but also why it works that way."

Tim

"And what's M4?"

Prof. Cho

"M4 is the Make stage. By creating your own projects from scratch, you truly internalize and deeply understand what you've learned."

Prof. Cho

"Making things yourself is where the deepest, most lasting understanding happens.[3]"

[3]This echoes Feynman's principle: 'What I cannot create, I do not understand.' Creation is the ultimate test and proof of understanding.

Jane

"So the complete journey is: confused → understanding → skilled → creating your own things?"

Prof. Cho

"That's a perfect summary! Magic → Machine → Master → Make. These four stages describe how you'll grow as a software engineer."

Prof. Cho

"But don't misunderstand—Make doesn't just come at the end! Making things happens from the very beginning all the way to the end. Without continuously making things, true understanding simply cannot develop!"

Tim thinks

"Got it! So I should be making things at every stage, not waiting until I feel 'ready.'"

The P4 Framework

Prof. Cho

"Now, let me introduce P4, which describes what we actually do in classes. P4 stands for four concepts that lead to true understanding of software engineering: Principles, Patterns, Practices, and Projects."

Tim

"What's the difference between all of these?"

Prof. Cho

"P1 is Principles—these are the timeless, fundamental truths that explain how things work. These are like laws of nature that don't change."

Prof. Cho

"P2 is Patterns—these are reusable, proven solutions that master software engineers have discovered over many years, sometimes even decades of experience."

Jane

"So Principles and Patterns together make up the theory or knowledge part?"

Prof. Cho

"Exactly right! Together, they form your theoretical foundation—the knowledge you need."

Prof. Cho

"P3 is Practices—these are the repeated actions and exercises you do to develop real mastery and skill."

Prof. Cho

"And P4 is Projects—these are the things you make to truly learn and integrate everything together."

Tim thinks

"Theory is Principles and Patterns. Application is Practices and Projects. That's a helpful way to organize it!"

Map(Theory) and Journey (Application)

Prof. Cho

"Think of it this way: theories are like a map that guides you through unfamiliar terrain. Application is the actual journey—walking the path yourself, learning by doing, experiencing challenges firsthand."

Tim

"So we need both the map and the actual journey to really learn?"

Prof. Cho

"Yes! And here's what research shows: starting with the application—actually taking the journey—leads to a much deeper understanding of theories than just studying the map."

Prof. Cho

"That's why we build and do things first, then discover the underlying principles."

Jane

"Wait, you mean we don't start with long theory lectures?"

Prof. Cho

"Correct! We start by actually doing and building. In other words, it's OK to start with Magic, without understanding the Machine."

Prof. Cho

"The theory emerges naturally when you need it to solve real problems. This is the P4M4 model in action: combining the four activities (P4) with the four learning levels (M4)."

P4	M4
P4: Project	M4: Make
P3: Practice	M3: Master
P2: Pattern	M2: Machine
P1: Principle	M1: Magic

Tim thinks

"This is somewhat different from how I learned things before. But it makes sense—I remember things better when I use them!"

Three Course Stages

Prof. Cho makes three rows on the whiteboard. Then, writes three stages that start with the P.

Stage	Description
1	Preparation
2	Prototype / MVP
3	Project

1. Preparation Stage

Prof. Cho

"Every ASE course follows three clear stages. Stage 1 happens during the first few weeks: **Preparation**, also called Introduction and Setup."

Prof. Cho

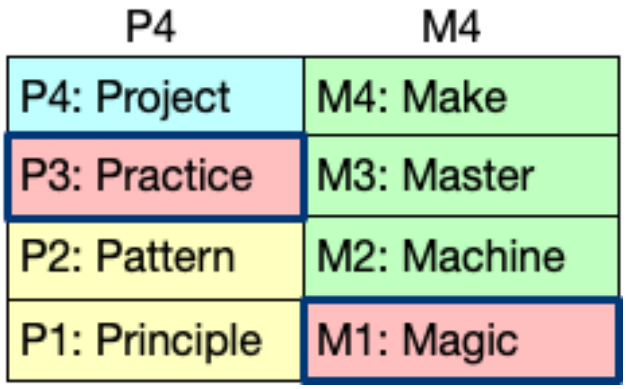
"During this stage, you'll set up your development tools, read important documents, and understand basic concepts to be ready for projects."

Tim

"Is that the Magic stage of M4?"

Prof. Cho

"Yes, exactly! This is P3M1—Practice combined with Magic. You're doing things that might feel mysterious and confusing at first, and that's completely normal and expected."



Tim thinks

"So it's okay to feel confused at first. I just need to keep practicing and it will start making sense."

2. Prototype/MVP Stage

Prof. Cho

"Then Stage 2, called Prototype/MVP, starts[4]: This is Skill Building through Practice."

Tim

"I know what a prototype is, but what exactly is an MVP?"

Prof. Cho

"MVP stands for Minimum Viable Product. It's the simplest version of your product that actually works and can be used. It's the starting point of a real software engineering project."

Tim

"If we already have a working prototype, why do we need an MVP?"

[4]It begins with the first iteration of the project and typically ends around the time of the first midterm. Students should check the specific schedule for each course."

Prof. Cho

"Good question! When we make a prototype, we focus only on checking feasibility[5], which means we skip many important activities like proper requirements, thorough testing, and good documentation. We just make code that barely works to see if our idea is possible."

Jane

"Okay, so with a prototype, we only have code that barely works. But by making the prototype, we discover things we couldn't see or think about before we started?"

Tim

"I get it now! We can't know what we don't know until we actually try building something. So we need a prototype to quickly discover what was unclear or completely unknown to us at the start."

[5] "Feasibility means checking if we can actually solve the problem or not by answering the question 'is it even possible?'"

Prof. Cho

"Exactly! Especially when starting a project you've never tried before, you don't know what you don't know in the beginning[6]. So you need to find out what you don't know[7] as quickly as possible to finish the project successfully."

Tim

"That sounds like a really important transition—from **unknown unknowns** to **known unknowns**."

Jane

"I agree! But on top of that, we need much more than barely working code that a prototype provides to make high-quality software."

Jane

"We need requirements, good design, thorough tests, and clear documentation to make a complete, professional project."

[6] "We call this **unknown unknowns**—things you don't even know that you don't know."

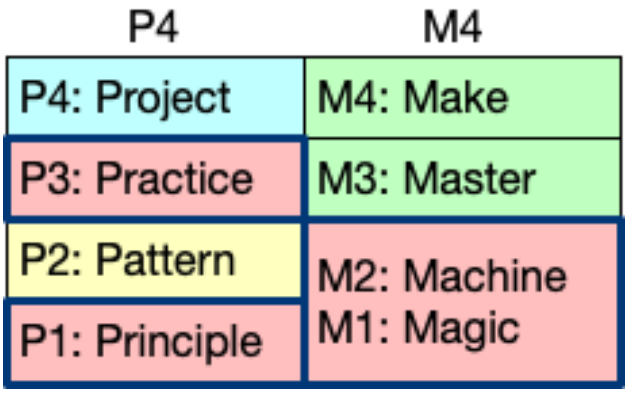
[7] "We call this **known unknowns**—things you now know that you don't know"

Tim

"Aha! That's exactly why we need the MVP stage!"

Prof. Cho

"In other words, this stage is about P13/M12—you're shifting focus from Magic to Machine (M1 to M2), and from just Practice to understanding Principles (P3 to P1)."



Jane

"So basically, you're saying we build our understanding of software engineering through repeated hands-on practice with prototypes and MVPs in this stage."

Prof. Cho

"Yes. Remember we learn by making things."

Tim

"By the way, do we have to make all the prototypes and MVPs completely on our own?"

Prof. Cho

"For most team projects[8], I'll provide you with a working prototype to start from. Together with that, we'll discuss all the necessary technologies and important ideas for understanding how the prototype works.[9]"

Tim

"But it's still our job to build a working MVP, right?"

Prof. Cho

"Good catch, Tim! Yes, prototype can be given, but teams should make an MVP in this stage."

Jane

"So this stage, teams focuses more on building an MVP from what they learn from the prototype.[10]"

---

[8] "For individual projects, students may need to create their own prototypes."

[9] "For the capstone course (ASE 485), students should create their own prototypes unless they're extending their previous projects."

[10] Or making a prototype before building an MVP.

Prof. Cho

"Yes! We learn and use software engineering rules and tools, such as development processes, high-level languages, high-level tools, in this stage to successfully finish the project."

Tim

"That makes a lot of sense to me now."

3. The Project Stage

Prof. Cho

"Finally, the stage 3 starts.[11]."

Prof. Cho

" In this stage, most of your unknown unknowns have been identified and discovered, so you can focus entirely on delivering the features you promised to build."

Tim

"So that's where everything we've learned comes together?"

[11]This phase typically starts after the first midterm and continues until week 16.

Prof. Cho

"Yes! You apply everything you've learned in class, and sometimes even beyond what we covered, to successfully finish the project. This is the stage where you prove you can solve real problems by implementing the features you promised to deliver."

Jane

"What happens if I discover that I missed something important? What if I know that I can't deliver the software features I promised at the beginning after making prototypes and MVPs?"

Prof. Cho

"This is why we use the agile development process! We can change anything as soon as we discover the necessary changes. That flexibility is built into the process."

Jane

"So each stage builds on the previous one, but we can still change anything whenever we discover it's necessary?"

Prof. Cho

"Yes, exactly! That's how we manage the complexity that comes from changes in real-world software engineering."

Prof. Cho

"So don't be afraid of starting and doing. Start early, make mistakes early, and correct those mistakes early while there's still time."

Tim

"I see! As problem solvers, we're actively exploring and discovering unknown unknowns."

Prof. Cho

"Exactly right! You start with exploration and discovery, build practical skills through practice, and finish by creating something real. That's the complete P4M4 journey."

Jane

"Also, as a software engineer, I need to build all the necessary tests and documentation based on my promises, not just code, right?[12]"

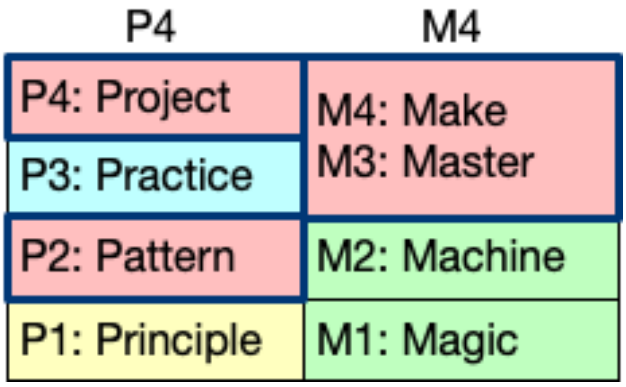
[12] "We call all these things—code, tests, documentation, etc.—artifacts."

Prof. Cho

"Yes! Code is actually a small part of real-world software engineering. We need to build a complete system that's easy to revise and extend."

Prof. Cho

"Software is high quality when we can easily fix bugs and add new features. That's our goal."



Tim's and Jane's Realization

After class, Tim and Jane discuss what they've learned.

Tim

"So learning isn't a straight line. It's Magic → Machine → Master → Make, and we go through all these stages."

Jane

"And the activities we do are Principles, Patterns, Practices, and Projects—the P4."

Tim

"Stage 1 is about Preparation: I might feel lost and confused, but that's completely okay—it's just the Magic stage. I'll understand the 'trick' behind it soon."

Jane

"Stage 2 is about Prototype/MVP: I'll practice a lot and gradually understand how things work—moving from Machine to Master. By the end, I'll have an MVP."

Tim

"Stage 3 is about the Project: I'll be building and delivering the features I promised, actually creating what I said I would make."

Jane

Also, the software product is easy to fix bugs and extend features; what we call 'high-quality software'.

Tim

"Jane, I think I finally understand. The ASE courses aren't trying to make learning easy or comfortable—they're designed to make learning real and effective by actually doing things and making high-quality software products."

Jane

"Exactly! And the P4M4 model gives us a clear roadmap for that journey. We know where we are and where we're going."

Tim thinks

"I feel so much more confident now. I understand the journey ahead, even if I don't know everything yet."

Chapter 4:
Course Grading and Success

- Grades come from assignments, projects, midterms, and daily quizzes (only for face-to-face courses).
- Starting early is absolutely crucial—the real secret to success is discovering what you don't know as soon as possible.
- Projects should be your top priority for your career; aim to add at least one impressive project to your portfolio after every course.
- Remember to: divide and conquer with practice, begin with magic and don't be afraid, and master many problem-solving patterns.

Understanding the Grade Breakdown

Prof. Cho

"Now let's discuss how grades work in this course. The total possible points is 1000."

Tim

"How are those 1000 points divided up?"

Prof. Cho writes down a table on the whiteboard.

Component	Points	Percentage
Assignments (5)	250	25%
Projects	350/400*	35%/40%*
Midterm Exams (2)	350	35%
Daily Quizzes	50/0*	5%/0%*

- * indicates online course
- Can be changed for each course.

Prof. Cho

"Five assignments are worth a total of 250 points—that's 25% of your grade. Projects are worth 350 points for face-to-face courses or 400 points for online courses—that's 35% to 40%. Two midterm exams are worth 350 points—that's 35%. And daily quizzes are worth 50 points for face-to-face courses—that's 5%.[13]

Jane

"So projects and exams together make up the biggest portions of our grade?"

Prof. Cho

"Yes, they're the most important parts. And there are also bonus points available—up to 50 points, which is an extra 5%—for helping to improve the course."

Tim thinks

"Let me write this down for face-to-face course: Assignments 25%, Projects 35%, Exams 35%, Quizzes 5%, Bonus up to 5%. Got it!"

[13] "This is a general guideline and can be adjusted depending on specific courses and situations."

Homework: The Foundation

Prof. Cho

"Let's talk about homework assignments. There are five assignments, each worth 50 points, totaling 250 points. Deadlines are firm and must be respected."

Tim

"What happens if we can't finish an assignment on time?"

Prof. Cho

"Extensions are possible if you request one at least 72 hours (3 days) before the due date. No penalty applies in this case."

Prof. Cho

"Extension requests made within 72 hours of the due date incur a 20% late penalty on that assignment. No exceptions."

Jane

"So it's much better to plan ahead and finish on time?"

Tim

"So if I start early and realize I can't finish on time, I can request an extension at least three days before the deadline without any penalty."

Prof. Cho

"Yes. When students request an extension, they must also submit a plan showing when they will finish the assignment. If they miss the new deadline, no second extension will be granted."

Prof. Cho

"The assignments are designed to systematically build your skills step by step. Each assignment prepares you for the projects that come later. Don't wait until the last minute to start—that's a recipe for stress and poor work."

Tim thinks

"Plan ahead, start early, avoid the penalty and the stress. Makes sense!"

Projects: Where You Shine

Prof. Cho

"Projects are where you really demonstrate and prove your abilities. For face-to-face courses, there's a team project worth 200 points and an individual project worth 150 points."

Tim

"What about for online courses?"

Prof. Cho

"For online courses, you'll complete two individual projects, each worth 200 points. For both formats, you'll give regular status updates to communicate your progress to me and the class."

Jane

"Why are projects considered so important compared to assignments?"

Prof. Cho

"Because projects are how you build your professional portfolio. Every course should add at least one impressive project that you can show to potential employers. Projects prove you can solve real, complex problems—not just answer homework questions."

Tim thinks

"Projects aren't just for getting grades—they're building blocks for my future career. I need to take them seriously!"

Exams and Daily Quizzes

Prof. Cho

"There are two midterm exams, worth 150 and 200 points respectively. The second exam is comprehensive, meaning it covers everything you've learned. Both exams are taken online through Canvas."

Tim

"Are we allowed to use notes during the exams?"

Prof. Cho

"Yes, you can use one double-sided cheat sheet that you prepare. But there are strict time limits to prevent cheating—you need to really understand the material deeply, not just frantically look things up during the exam."

Jane

"What about the daily quizzes you mentioned?"

Prof. Cho

"For face-to-face courses only, quizzes are given at the start of each class to review key ideas from previous classes. All the points per class will be accumulated and normalized to 50 points. Here's the good news: you get points for all quiz submissions, even blank ones."

Tim

"Wait, even blank submissions get points?"

Prof. Cho

"Yes! The goal is to encourage class participation and to check the students' understanding of the course, not to punish you for not knowing something. Online courses don't have daily quizzes because of the different format."

Jane

"How do we submit our quiz answers?"

Prof. Cho

"Send me or a TA, when we have a TA, your answers during the class meeting time. Even if you arrive late to class, you can still send an empty email to earn points for that day."

Prof. Cho

"However, if students don't send an email, the student can't earn points. Also, if students send an email outside classroom meeting time, they also earn 0 points."

Tim

"What if something comes up and I can't attend class for a personal reason?"

Prof. Cho

"In that case, contact me before class starts. You can answer the quiz questions later when they're uploaded to Canvas after class."

Tim thinks

"So daily quizzes reward showing up and participating. That's fair and encourages good habits!"

Earning Bonus Points

Tim

"How exactly can we earn those bonus points you mentioned?"

Prof. Cho

"You can earn up to 50 bonus points maximum for helping improve the course. This includes finding bugs or errors in course materials or programs, providing valuable feedback on how to make the course better."

Jane

"So you're encouraging us to be active participants who help improve the course?"

Prof. Cho

"Yes! My courses are my products, and you students are my clients. Teaching is my software engineering and my problem-solving. I need honest feedback from you to make my courses better, just like any software product needs user feedback."

Prof. Cho

"My goal is to make high-quality products, I mean, my ASE courses to benefit my clients, I mean, my students."

Tim thinks

"That's a cool perspective—treating students as clients and the course as a product that keeps improving! Then, what is my product and who is my client?"

Four Key Recommendations

Prof. Cho

"Let me give you four recommendations for success in this course and in your career."

Prof. Cho writes down on the whiteboard.

	Recommendations
1	Start early, finish early
2	Take action - don't overthink
3	Be proactive
4	Ask for help when needed

Prof. Cho

"First: Start early, finish early. The real secret to success is identifying what you don't know as soon as possible, while there's still time to learn it."

Tim thinks

"Don't wait to discover my knowledge gaps when it's too late to fix them..."

Prof. Cho

"Second: Take action. Just do it. Don't overthink. Solutions and understanding will appear once you start doing things. Don't wait for perfect conditions or perfect understanding—act now and learn as you go."

Jane

"So taking action comes before having complete clarity?"

Prof. Cho

"Exactly! Clarity comes through action, not before it. Third: Be proactive. Take ownership of your own decisions and actively drive your own progress forward. You know deep down that you can do this."

Prof. Cho

"Fourth, and this is really important: Ask for help when you need it. Don't waste time struggling alone with a problem. People—me, TAs, classmates—are ready and willing to help if you just reach out."

Tim

"But isn't asking for help a sign of weakness or not being smart enough?"

Prof. Cho

"No! It's actually a sign of wisdom and professionalism. In the real world of work, knowing when to ask for help is a crucial professional skill. Struggling silently for hours wastes everyone's time and delays the project."

Jane

"So we should ask early, not after we've been stuck and frustrated for hours?"

Prof. Cho

"Yes! If you've been stuck on something for a while with no progress at all, that's the perfect time to ask for help. That's the smart, professional approach."

Tim thinks

"Asking for help is smart, not weak. I need to remember that!"

Applying P4M4 to Success

Prof. Cho

"Remember the P4M4 model we discussed? You can apply it to your success in this course."

Prof. Cho

"First: Make projects your top priority. Build your portfolio—these projects will help you get jobs. Second: Divide and conquer with practice. Make progress step-by-step through focused, manageable practices rather than trying to do everything at once."

Tim

"What about the M4 part of P4M4?"

Prof. Cho

"Begin with the magic—start without overthinking everything. Don't be paralyzed by not understanding. Then grow with principles—discover how the underlying principles transform that initial 'magic' into a 'machine' you understand. And master many patterns—learning patterns is your secret weapon for solving real-world problems efficiently."

Tim thinks

"Projects first, practice consistently, don't fear the unknown, learn the principles, master the patterns. I can do this!"

Tim's and Jane's Resolution

After class, Tim has an honest conversation with Jane.

Tim

"I'll be honest—I tend to procrastinate. But I'm going to force myself to start early this time by just doing something, literally anything, to get started!"

Jane

"Me too! Also, I'm going to ask for help when I'm stuck, instead of suffering in silence for hours thinking I should figure it out alone."

Tim

"I know now that projects are my top priority for my future career as a software engineer. After each course, I'm going to add something I can be genuinely proud of to my portfolio."

Jane

"Absolutely! And I'll take action instead of waiting around for perfect understanding before I start."

Tim

"Jane, I'm going to start the first assignment this weekend, even though it's not due for two whole weeks."

Jane

"That's exactly the right spirit! Starting early means you'll discover what you don't know while there's still plenty of time to learn it and get help."

Tim

"Exactly. I'm done waiting for the 'perfect moment' to start. The perfect moment is right now."

Tim thinks

"No more excuses. No more procrastination. I'm starting early and asking for help when I need it. This is how I'll succeed!"

Chapter 5:
Course Rules

- Students follow two essential rules: Integrity First (protect your integrity above all) and Two Hats Rule (be both a learner and a problem solver).
- Professors follow two essential rules: Be Fair (treat all students equally without exception by following rules) and Help Students Succeed (guide them to become skilled, confident problem solvers).
- All additional rules in the course come from these four core principles.
- When you're unsure about any rule or situation, always ask immediately rather than guessing.

The Foundation of Rules

Prof. Cho

"Today we'll discuss the rules that guide how this course works. They're intentionally simple but absolutely fundamental."

Tim

"How many rules do we need to remember?"

Prof. Cho

"There are four core rules."

Prof. Cho writes down the rules on the whiteboard.

Role	Rule	Description
Students	1	Integrity First - protect your integrity above all
	2	Two Hats Rule - be both a learner and a problem solver
Professors/TA	1	Be Fair - treat all students equally by following rules
	2	Help Students Succeed - guide them to become skilled, confident problem solvers

Prof. Cho

"Two rules for students to follow, and two rules for professors and TAs to follow. Everything else in the course derives from these four basic principles."

Jane

"Only four rules? That actually seems very manageable!"

Prof. Cho

"Simplicity is completely intentional. When rules are complicated and numerous, people get confused and make mistakes. When rules are clear and few, everyone knows exactly where they stand and what's expected."

Tim thinks

"Four simple rules instead of a huge rulebook. I like this approach!"

Prof. Cho

"Of course, we need more detailed rules to clarify specific situations, but these four core principles are our foundation that we all agree upon and follow."

Student Rule 1: Integrity First

Prof. Cho

"The first and most important student rule is 'Integrity First.' Your integrity isn't just about academics—it's a core part of who you are as a person."

Tim

"What does 'Integrity First' mean in actual practice?"

Prof. Cho

"It means avoiding all forms of cheating. Don't copy code from others. Don't share your solutions with classmates. Don't claim work as yours that isn't."

Prof. Cho

"But most importantly: be completely truthful with yourself about your learning."

Jane

"What do you mean by being truthful with ourselves?"

Prof. Cho

"If you take shortcuts or cheat, you're only cheating yourself in the end. You might get a good grade temporarily, but you won't get the real learning and skills. And ultimately, you'll fail when you need that knowledge and those skills in the real world."

Tim thinks

"So integrity isn't really about avoiding punishment—it's about becoming a real, capable software engineer who actually knows what they're doing."

Prof. Cho

"Let me ask you this: Would you want to work with people who cheat or take dishonest shortcuts?"

Jane

"Absolutely not! Cheaters would ruin any team they're on."

Tim

"I would never want to work with someone who cheats or intentionally cuts corners! I couldn't trust them."

Prof. Cho

“Exactly! So maintain your integrity, not just for grades, but for your future career and for becoming someone others want to work with.”

Student Rule 2: Two Hats

Prof. Cho

"The second student rule is called 'Two Hats.' You wear two different hats in this course, and both are important."

Tim

"Two hats? What does that mean?"

Prof. Cho

"First, you are a professional problem solver. In ASE courses, you focus on solving problems effectively and efficiently. You deliver real results and working solutions, just like a professional would."

Jane

"And what's the second hat?"

Prof. Cho

"You are also a student. Your job as a student is to learn, explore new things, and continuously improve. You're allowed—even expected—to make mistakes."

Prof. Cho

"In fact, it's much better to make mistakes here in class. When you learn from classroom mistakes, you won't repeat them in the real world where the cost is much higher."

Tim

"So we're supposed to be both professional and beginner at the same time?"

Prof. Cho

"Haha, you could say that! You solve real, challenging problems like a professional would, but you learn and grow like a student should. Both roles are equally essential to your development."

Tim thinks

"Professional problem solver + eager learner. I need to remember to wear both hats!"

Professor Rule 1: Be Fair

Prof. Cho

"Now let me tell you about the rules I commit to following as your professor. First and most important: Be Fair. I have a serious obligation to treat all students equally without exception."

Tim

"What should we do if we feel something is unfair?"

Prof. Cho

"Any perception of unfairness must be addressed immediately. If you ever feel I'm treating you or another student unfairly, tell me right away—don't wait. I promise you that I'll correct anything unfair right away."

Jane

"So fairness isn't just a nice goal—it's an actual requirement you hold yourself to?"

Prof. Cho

"Absolutely! Fairness is completely non-negotiable. Every single student deserves equal treatment and equal opportunity to succeed, no matter who they are or where they come from."

Tim thinks

"It's good to know the professor takes fairness this seriously. That makes me feel more confident."

Professor Rule 2: Help Students Succeed

Prof. Cho

"The second professor rule: Help Students Become Skilled Problem Solvers. My primary goal is to guide/help you to become competent, confident, and ethical problem solvers who can succeed in the real world."

Tim

"So it's not just about helping us pass the course?"

Prof. Cho

"No, it's about much more than grades. It's about developing genuine competence and real skills. I want you to leave this course truly able to solve real problems in the real world of software engineering."

Jane

"And should the learning process be enjoyable too?"

Prof. Cho

"Absolutely! Learning should be enjoyable and engaging, and solving problems should feel rewarding and satisfying. If you're feeling that something is wrong with how the course is going, please let me know, and I'll do my best to fix it."

Tim

"I want to grow and learn, but sometimes I feel afraid, especially with all the complicated software engineering technologies and concepts."

Prof. Cho

"That's a completely normal feeling that every software engineer experiences, even experienced ones!"

Prof. Cho

"But don't let fear stop you. Just start something, make something, don't be afraid of making mistakes, and learn from any mistakes you do make. Suddenly, you'll discover that you've become a capable problem solver without even realizing it."

Tim thinks

"Start, make, learn from mistakes, grow. I can do this!"

All Rules Derive from These Four

Prof. Cho

"Here's the important part to understand: all additional, specific rules in the course will be derived from these four core principles. Detailed rules and guidelines will be available on Canvas for you to reference."

Tim

"So if we really understand these four core rules, we can figure out everything else?"

Prof. Cho

"Exactly! When you encounter a situation and aren't sure what to do, ask yourself these questions: What does the 'Integrity First' suggest I should do? What would the 'Two Hats approach' recommend? What would be the fair thing to do? What would help me become a better problem solver?"

Jane

"And if we're still not sure after thinking about those questions?"

Prof. Cho

"Then ask me directly! When in doubt, always ask. I'd much rather answer a hundred questions than have even one student confused, misled, or making wrong decisions because they were afraid to ask."

Tim thinks

"So the four rules are like a compass that helps me navigate any situation!"

The Spirit Behind the Rules

Prof. Cho

"Here's why we use these four simple rules instead of having too many detailed rules: Too many specific rules sometimes create confusion and loopholes. People start looking for what they can get away with instead of focusing on what's the right thing to do."

Prof. Cho

"These four rules establish clear principles, not technical loopholes. In the professional software engineering world, people expect you to think about intentions and doing the right thing, not just following the letter of rules while violating their spirit."

Jane

"So it's really about developing good judgment and ethical thinking?"

Prof. Cho

"Exactly right! In the real world of work, you won't have a detailed rulebook for every single situation you encounter. You'll need to exercise good judgment based on principles like integrity, fairness, continuous learning, and growth."

Tim thinks

"This is about becoming a professional who makes good decisions, not just someone who follows rules mechanically."

Tim and Jane Discuss

After class ends, Tim and Jane talk about what they learned about the course rules.

Tim

"You know, I honestly expected there to be a lot more complicated rules."

Jane

"Me too! But these four simple rules are actually really powerful when you think about them."

Tim

"Integrity First means I can't deceive myself or take shortcuts. Two Hats means I need to balance being a professional problem solver and being a learner who's allowed to make mistakes."

Jane

"And knowing that Prof. Cho is committed to fairness and to helping us succeed makes me feel supported and confident."

Tim thinks

"Simple rules, but they actually cover everything that matters. And if I'm ever unsure, I can just ask."

Tim

"I think these rules are designed to make us think carefully and make good decisions, not just blindly comply with lots of detailed requirements."

Jane

"That's exactly right! And that's probably the most important lesson of all—learning to think and make good decisions ourselves."

Tim thinks

"I like this approach. It treats us like adults who can think, not children who need constant supervision."

Chapter 6:
Software Development Tools

- ASE students master essential development tools: VSCode (code editor), Git/GitHub (version control), Linux CLI (command line), and documentation formats like Markdown and JSON.
- Studens will learn multiple programming languages: Java, Python, and JavaScript—each serves different important purposes.
- Learning to independently install and use tools is an essential skill for a successful software engineering career.
- Each ASE course introduces new tools and technologies, progressively building toward comprehensive professional competence.

Essential Skills for Modern Development

Prof. Cho

"Today we'll discuss the important tools you'll use throughout the ASE program."

Prof. Cho

"These aren't just random tools for coursework—they're industry standards that professional software engineers use every day."

Tim

"How many different tools do we need to learn?"

Prof. Cho

"In 200-level courses, you'll gain basic familiarity with three main categories of tools: SWE Tools (Software Engineering Tools), Documentation Formats, and Programming Languages."

Jane

"That sounds like quite a lot to learn!"

Prof. Cho

"It is a significant amount, that's true. But remember: these are state-of-the-art problem-solving tools that have passed the test of time."

Prof. Cho

"They're proven to work well. You'll master them faster than you think through consistent hands-on practice."

Tim

"But honestly, I really prefer using GUI tools with buttons and menus."

Prof. Cho

"Of course! GUI tools are easy and intuitive to use, which is great. But it's almost impossible to automate work sequences with GUI tools."

Prof. Cho

"The focus shouldn't be 'GUI or command line'—it should be finding and using the right tool for solving your specific problems."

Tim thinks

"So sometimes command line is better, sometimes GUI is better. Use the right tool for the job!"

Different Levels of Tools

Prof. Cho

"We need two different levels of software engineering tools."

Prof. Cho writes on the whiteboard.

Level	Focus	Description
Level 1	Essential Developer Tools	Individual development environment and workflow
Level 2	Team and Project Tools	Collaboration and project management

Prof. Cho

"Level 1 is about mastering your individual development environment. Level 2 is about working effectively with your team. You need both to be a professional software engineer."

Level 1: Essential Developer Tools

Prof. Cho

"First, let's talk about the essential developer tools you'll use constantly."

Prof. Cho writes on the whiteboard.

Tool	Purpose
VSCode	Code editor with powerful extensions
Linux CLI	Command-line efficiency and automation
Git and GitHub	Version control and team collaboration

VSCode

Prof. Cho

"VSCode is the de-facto standard IDE for modern software development."

Jane

"Isn't VSCode just a text editor?"

Prof. Cho

"VSCode is much more than a text editor. It's a full Integrated Development Environment (IDE). Using it as just an editor means you're missing out on its real power."

Prof. Cho

"With the right extensions, VSCode becomes a complete development environment for any programming language or framework."

Linux CLI

Prof. Cho

"The next tool is Linux command line interface tools, called CLI."

Tim

"Why do we need to learn Linux commands? Can't we just use graphical interfaces for everything?[14]"

[14]"Windows users should install WSL2 (Windows Subsystem for Linux) to get a Linux environment."

Prof. Cho

"Command-line tools are actually much faster and more powerful than graphical tools for many software engineering related tasks."

Prof. Cho

"They're scriptable, meaning you can automate them, and automatable, meaning you can chain commands together. Every professional software engineer needs solid CLI (command-line interface) skills.[15]"

Jane

"So we really can't avoid learning to use the terminal?"

Prof. Cho

"No, and you shouldn't want to avoid it! Once you master CLI tools, you'll actually wonder how you ever worked efficiently without them. They become incredibly powerful."

[15]Linux command-line skills aren't just for system administrators anymore. Modern software development requires automating repetitive tasks, managing cloud services and servers, and understanding how systems operate behind the scenes. These CLI skills will multiply your effectiveness as a developer significantly.

Tim thinks

"I guess I need to give command line a real chance. It might be hard at first, but it sounds worth it."

Git and GitHub

Prof. Cho

"Git is the tool for version control."

Tim

"I'm not sure why we need version control in a project."

Prof. Cho

"Git saves every version of your code. When you break something, you can go back to when it worked. It's like having unlimited undo for your entire project."

Jane

"What about GitHub?"

Prof. Cho

“GitHub is where you store your Git repositories online. It lets you collaborate with teammates, backup your code, and share your work with others.”

Tim

“Then, it sounds like GitHub is the bridge between Level 1 and Level 2 tools.”

Prof. Cho

“Exactly right! Git is individual, GitHub makes it collaborative. Perfect connection.”

Prof. Cho

“OK. Let’s talk about the Level 2, team and project tools.”

Level 2: Team and Project Tools

Prof. Cho

"In 300 and 400 level courses, you'll add Team and Project Tools to your toolkit."

Prof. Cho writes on the whiteboard.

Category	Examples	Required?
Version Control	Git and GitHub	Yes
CI/CD	GitHub Actions	Yes
Deployment	Docker	Yes
Project Management	Jira, Trello, Asana	No
Communication	Slack, Teams, Discord	No
Advanced Deployment	Kubernetes, Cloud platforms	No

- Required means that we use the tools in the course projects.

Prof. Cho

"Three tools are actually used in course prljects: GitHub for version control and collaboration, GitHub Actions for automation, and Docker for deployment. The rest are optional—you choose what works best for your team."

Tim

"What if we're not familiar with these tools yet when we get to those courses?"

Prof. Cho

"That's completely okay and expected! Just jump in and start using them. With regular practice, you'll master these tools faster than you think."

Jane

"Do we need to learn all these tools on our own?"

Prof. Cho

"Yes, to a large extent. Learning to independently install and use new tools is an essential software engineering skill. ASE courses provide basic instructions and support, but you'll need to drive your own learning."

Tim thinks

"So part of the learning is learning HOW to learn new tools independently. That makes sense for a career where technology is always changing."

Prof. Cho

“Exactly. Learning new tools on your own is a big part of software engineering. For information sharing for projects, we’ll use Canvas and GitHub. For everything else, choose tools that solve your team’s problems effectively.”

Core Fundamentals

Prof. Cho

"Beyond tools, you need to master two core fundamentals."

Prof. Cho writes on the whiteboard.

#	Fundamental	Purpose
1	Documentation Formats	Write and read structured information
2	Programming Languages	Solve problems and build software

Documentation Formats

Prof. Cho

"First, let's talk about documentation formats. Markdown is for lightweight text formatting—you'll use it constantly for README files, documentation, and even for creating presentations with a tool called Marp."

Tim

"What about JSON?"

Prof. Cho

"JSON is for structured data—things like configuration files and API responses. You'll also learn YAML and TOML, which are for human-readable configuration files."

Jane

"Why do we need so many different formats? Isn't that confusing?"

Prof. Cho

"Each format serves a specific purpose and is best for certain tasks."

Prof. Cho writes a table.

Format	Best For
Markdown	Human reading and writing
JSON	Machines to read and write
YAML	Configuration files that humans need to edit

Prof. Cho

"Markdown is best for human reading and writing. JSON is best for machines to read and write. YAML is best for configuration files that humans need to edit. Using the right format for the right job is what professionals do."

Tim thinks

"Different tools for different jobs. I'm starting to see the pattern here."

Programming Languages

Prof. Cho

"Finally, let's talk about programming languages."

Prof. Cho writes another table.

Language	Best For
Java	Standard object-oriented software engineering
Python	AI and rapid automation
JavaScript	Web development and interactivity

Prof. Cho

"You'll learn Java for object-oriented software engineering[16], Python for AI and automation, and JavaScript for web development.[17]"

Tim

"Do we really need to learn three different programming languages?"

Prof. Cho

"Yes, you really do. Each language is designed to solve different types of problems well."

Prof. Cho

"Java teaches you strong typing and solid object-oriented design principles. Python teaches you rapid prototyping, scripting, and AI integration. JavaScript teaches you web interactivity and modern web development."

[16] Or C#/C++ depending on your path

[17] This is a simplified guideline. In reality, these languages can be used for many other purposes too.

Jane

"So they're complementary—they work together to give us different capabilities?"

Prof. Cho

"Exactly right! Together, these three languages give you a complete, versatile toolkit for modern software development. You'll be able to solve almost any problem you encounter."

Tim thinks

"Three languages sounds intimidating, but if they're each good for different things, I can see why we need all of them."

Learning Resources

Prof. Cho

"Here's some good news: there are many high-quality YouTube tutorials available for almost any tool you need to learn."

Prof. Cho

"AI tools like ChatGPT can also help you—though keep in mind they sometimes give incorrect information due to hallucinations."

Tim

"So AI tools aren't perfect and we need to be careful?"

Prof. Cho

"That's right! But they're still very useful. When you get error messages, share them with AI tools—they can often guide you toward solutions or at least help you understand what's going wrong. Just remember to verify important information rather than blindly trusting everything."

Jane

"What should we do if we get really stuck with a tool?"

Prof. Cho

"Ask for help! Share your error messages in the class discussion forum. Your classmates, TAs, or I will help you. Remember what we discussed before: asking for help is professional and smart, not a sign of weakness."

Tim thinks

"YouTube, AI tools, class discussion, asking for help. I have lots of resources available!"

Tools Across ASE Courses

Prof. Cho

"Let me show you how tools build progressively across different ASE courses."

Prof. Cho

"While specific course details may vary by instructor, the core tools remain consistent. Each course introduces new tools while building on what you've already learned."

Prof. Cho

"For example, ASE 220 uses JavaScript, Node.js, and MongoDB for full-stack web development."

Tim

"What about ASE 230?"

Prof. Cho

"ASE 230 uses PHP and MySQL for server-side development, plus REST API tools like curl for testing APIs. Students also learn how to use deployment and CI/CD tools such as Docker and GitHub actions."

Prof. Cho

"ASE 285 adds Node.js, React, TypeScript for modern web development, and security tools like PGP and SSH."

Jane

"What does ASE 330 cover?"

Prof. Cho

"ASE 330 covers product design tools and methods: customer personas, the KANO Model, SCAMPER technique, and TRIZ framework. You'll learn to design products that users actually want and love to use."

Tim thinks

"Each course adds new tools and skills. It's like leveling up in a game!"

Advanced Course Tools

Prof. Cho

"ASE 420 dives deep into professional Python development—virtual environments, PIP package management, and PyTest for automated testing. You'll master Python development at a professional level."

Tim

"What about mobile development?"

Prof. Cho

"ASE 456 teaches Flutter and Dart for cross-platform development, along with working with a variety of databases. You'll build real applications that work on phones, tablets, desktop computers, and web browsers—all from a single codebase."

Jane

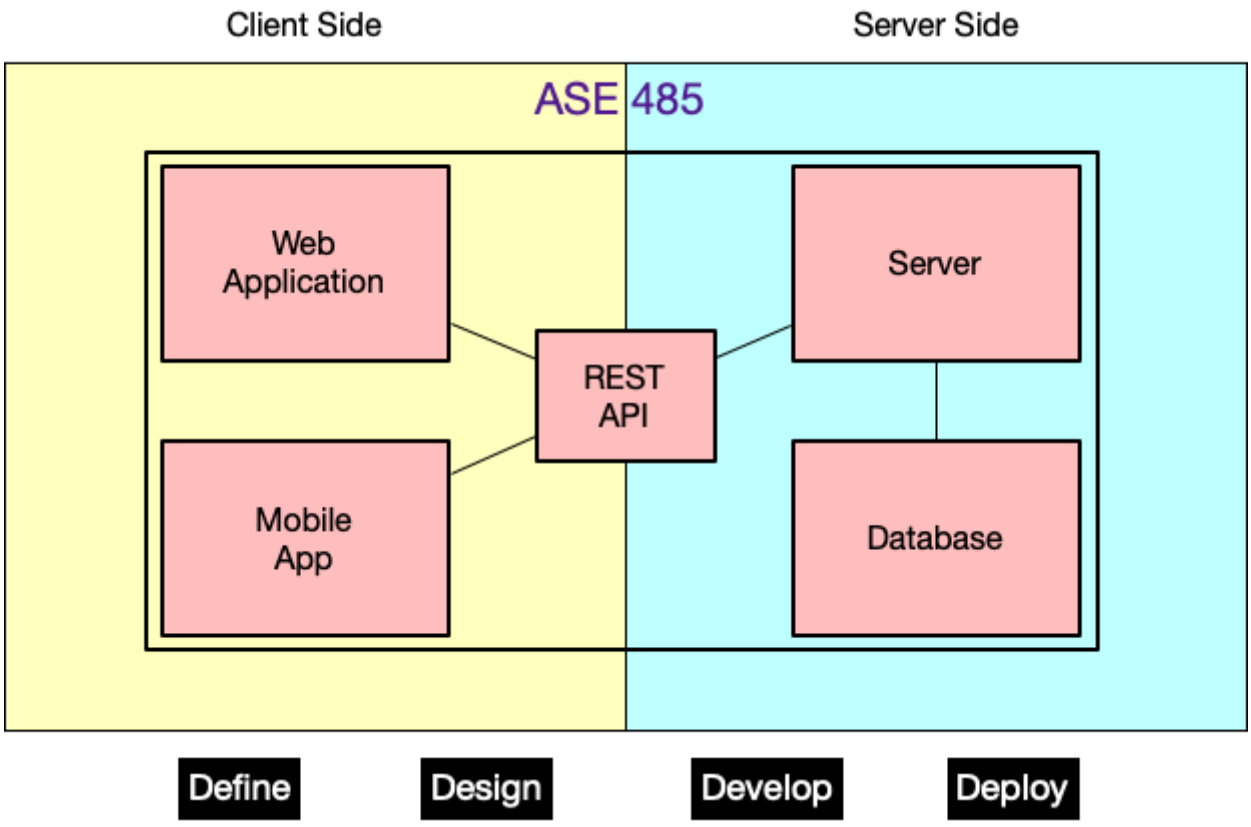
"That sounds incredibly efficient and practical!"

Tim

"And really fun! I've always wanted to build mobile apps."

Prof. Cho

"It is fun! And ASE 485, your capstone course, is where everything comes together. You'll choose and use the right tools for your specific capstone project based on everything you've learned."



Tim thinks

"By the end, I'll have experience with so many different tools and technologies!"

Remember This

Prof. Cho

"Here's what's really important to understand: this list of tools is a guideline, not a complete list. You'll actually use even more tools in ASE courses and beyond. You're expected to master these core tools—and more—before entering the real world of software engineering."

Tim

"So we need to be continuous, lifelong learners?"

Prof. Cho

"Always! Technology evolves and changes constantly. Your ability to quickly learn new tools and adapt to new technologies is actually more valuable than knowing any single specific tool perfectly."

Jane

"So it's not about memorizing specific tools—it's about learning how to learn new tools quickly to solve whatever problems come up?"

Prof. Cho

"Exactly right! That's the real, lasting skill. We software engineers should continuously learn how to solve problems better and better for the rest of our careers. The specific tools will change, but the learning process stays the same."

Tim thinks

"Learning to learn. That's the meta-skill I need to develop!"

Tim's and Jane's Revelation

After class, Tim feels both overwhelmed and excited about all the tools.

Tim

"So many tools to learn… VSCode, Git, Linux CLI, Markdown, JSON, Java, Python, JavaScript… my head is spinning a bit!"

Jane

"But Prof. Cho is right—they're just tools, like learning to use a hammer, saw, and drill. Once I understand one tool well, learning similar tools will become easier and faster."

Tim

"Yeah, the goal is to build a house effectively and as planned, not using the tools."

Jane

"Tim, are you worried or nervous about learning all these tools?"

Tim

"Not really, actually. I see them as power-ups in a video game. Each new tool makes me more capable and powerful as a developer. It's actually exciting!"

Jane

"That's a really good way to think about it! I like that perspective."

Jane

"Plus, we'll learn them gradually over time, one course at a time. By the time we finish ASE 485, we'll have a complete professional toolkit and be ready for real jobs."

Tim thinks

"A complete professional toolkit... I really like that idea. I'm not just learning to code—I'm becoming a professional with professional tools and skills."

The Journey Begins

Tim opens his laptop that evening, ready to start his journey.

> ***Tim thinks***
>
> *"Okay, first step: install VSCode. Then Git. Then practice some basic Linux commands."*

> ***Tim thinks***
>
> *"I'll start with the absolute basics and build from there, step by step."*

He downloads VSCode and carefully follows the installation guide.

> ***Tim thinks***
>
> *"It's working! Okay, next is installing Git..."*

Several hours later, Tim has VSCode, Git, and a working terminal all properly configured on his computer.

> ***Tim thinks***
>
> *"I did it! It wasn't as scary or difficult as I thought it would be. Each tool is just one step in the journey."*

He thinks back to Prof. Cho's encouraging words: "They're just problem-solving tools. You'll master them faster than you think."

> ***Tim thinks***
>
> *"He was completely right. I was intimidated for no good reason. Now I'm ready to start building real things. The journey has truly begun!"*

Tim smiles as he creates his first Git repository and makes his first commit.

> ***Tim thinks***
>
> *"One small step, but it feels like the beginning of something big. I'm on my way to becoming a real software engineer!"*