Team Projects (Practice)

Learning Software Engineering
by Doing Together

ASE

ASE @ Northern Kentucky University

Notice

This is a story about how team leaders and members work together on a project throughout a semester, from beginning to end.

Before reading this story, **students must read and understand the basic concepts of ASE courses and project theory** covered in other stories.

In this story, we will focus on the **practical action items** for working as a team, including communication, tools, collaboration, ceremonies, and project management, with real-world examples.

Meet the team

Team Member Profiles

Member	Strengths	Growth Areas
Tim	Curious, eager to learn, asks good questions	First ASE course, needs process guidance
Jane	Problem-solver, independent developer	Only individual project experience
Ken	Prior team experience, understands process	First time as team leader

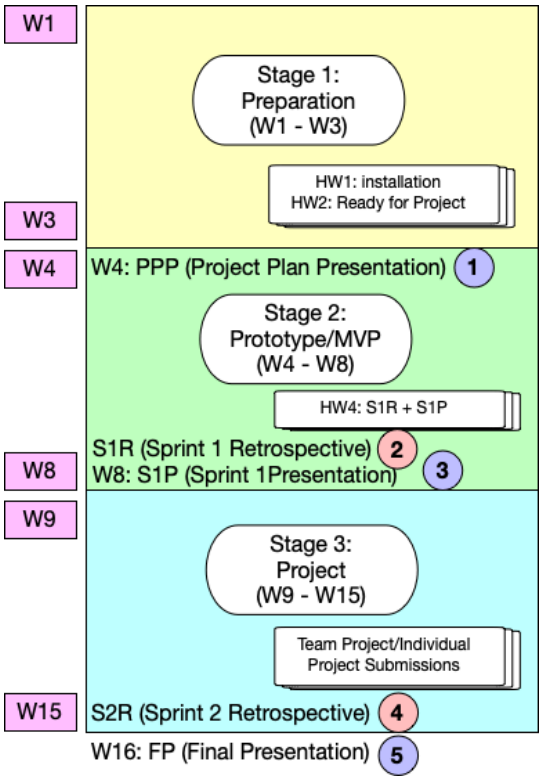
Why diverse skills matter: Teams succeed when members bring different perspectives. Tim's curiosity drives learning, Jane's problem-solving tackles technical challenges, and Ken's experience provides structure.

Roadmap of Team Project Stages

Five Ceremonies with Three Presentations

1. PPP (Project Plan **Presentation**)
2. S1R (Sprint 1 Retrospective)
3. S1P (Sprint 1 **Presentation**)
4. S2R (Sprint 2 Retrospective)
5. FP (Final **Presentation**)

Three Stages



- Stage 1: Preparation
  - Weeks: W1 - W3
  - Goal: **PPP (W4)**
  - Related Assignments
    - HW1: Team setup & installation
    - HW2: Ready for project

- Stage 2: Prototype/MVP
  - Weeks: W4 - W8
  - Goal: **S1P (W8)**
  - Related Assignment
    - HW4: S1R + S1P

- Stage 3: Project
  - Weeks: W9 - W15
  - Goal: **FP (W16)**
  - Related Assignments
    - Team project submission
    - Individual project submission

16 Weeks Schedule

This is an example schedule for a 16-week semester course with team projects. Adjustments, such as dates and deadlines, are needed based on your specific course timeline.

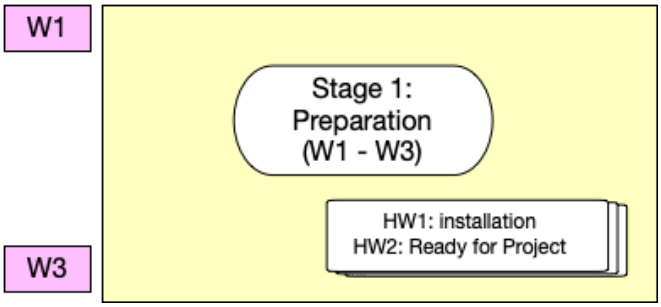
Stages	Weeks (M/W)	Sprints	Topics (Theory)	Modules	HW
Stage 1 (Preparation)	Week1 (1/12, 1/14)		Introduction		
	Week 2 (~~1/19~~, 1/21)	Project Discussion** Team Setup*	Project Preparations		HW1 (Installation) (1/25)*
	Week 3 (1/26, 1/28)		Project Preparations		HW2 (Project Plan) (2/1)*
Stage 2 (Prototype /MVP)	Week 4 (2/2, 2/4)	**Project Plan Presentation (2/2)** Sprint1 - 1	High-Level Languages	Module 1	
	Week 5 (2/9, 2/11)	Sprint1 - 2	High-Level Languages		
	Week 6 (~~2/16~~, 2/18)	Sprint1 - 3	High-Level Languages		
	Week 7 (2/23, 2/25)	Sprint1 - 4	Software Process		HW3 (Module 1) (2/27)**
	Week 8 (**3/2, 3/4**)	**Sprint 1 Presentation (3/2)**	**Midterm (3/4), on Canvas no class no office hours**		HW4 (Sprint 1 Retrospect + Sprint 2 Plan)(3/1)*
Stage 3 (Project)	Week 9		Spring Break (no class) **No classroom/office hours**		
	Week 10 (3/16, 3/18)	Sprint 2 – 1	Git	Module 2	
	Week 11 (3/23, 3/25)	Sprint 2 – 2	Git		
	Week 12 (3/30, 4/1)	Sprint 2 – 3	Security		
	Week 13 (4/6, 4/8)	Sprint 2 – 4	Security		
	Week 14 (4/13, ~~4/15~~)	Sprint 2 – 5	**Midterm (4/15, on Canvas no class/office hours)**	Module 3	HW5 (Module 2) (11/22)***
	Week 15 (4/20, 4/22)	Sprint 2 – 6 **Project submissions deadline (4/26)***	Deployment		
	Week 16 (4/27, 4/29)	**Presentation (4/27, 4/29)**			

Section

Stage 1: Preparation

Goal: PPP (Project Plan Presentation) @ W4

Weeks: W1 - W3



1. **PPP (Project Plan Presentation)** <- Goal
2. S1R (Sprint 1 Retrospective)
3. S1P (Sprint 1 Presentation)
4. S2R (Sprint 2 Retrospective)
5. FP (Final Presentation)

Week 1 - 2

This is a story about what students should do in the first week. Teams should be formed by the end of the week 2, so this story includes all the necessary steps in the week 2.

1. HW1
2. Team Formation
3. GitHub Setup
4. Canvas Pages

Stages	Weeks (MAM)	Sprints	Topics (Theory)	Modules	HW
Stage 1 (Preparation)	Week1 (1/12, 1/14)		Introduction		
	Week 2 (~~1/19~~, 1/21)	Project Discussion** Team Setup*	Project Preparations		HW1 (Installation) (1/25)*
	Week 3 (1/26, 1/28)		Project Preparations		HW2 (Project Plan) (2/1)*
Stage 2 (Prototype /MVP)	Week 4 (2/2, 2/4)	**Project Plan Presentation (2/2)** Sprint1 - 1	High-Level Languages	Module 1	
	Week 5 (2/9, 2/11)	Sprint1 - 2	High-Level Languages		
	Week 6 (~~2/16~~, 2/18)	Sprint1 - 3	High-Level Languages		
	Week 7 (2/23, 2/25)	Sprint1 - 4	Software Process		HW3 (Module 1) (2/27)**
	Week 8 (**3/2, 3/4**)	**Sprint 1 Presentation (3/2)**	**Midterm (3/4), on Canvas no class no office hours**		HW4 (Sprint 1 Retrospect + Sprint 2 Plan)(3/1)*
Stage 3 (Project)	Week 9		Spring Break (no class) **No classroom/office hours**		
	Week 10 (3/16, 3/18)	Sprint 2 – 1	Git	Module 2	
	Week 11 (3/23, 3/25)	Sprint 2 – 2	Git		
	Week 12 (3/30, 4/1)	Sprint 2 – 3	Security		
	Week 13 (4/6, 4/8)	Sprint 2 – 4	Security		
	Week 14 (4/13, **~~4/15~~**)	Sprint 2 – 5	**Midterm (4/15, on Canvas no class/office hours)**	Module 3	HW5 (Module 2) (11/22)***
	Week 15 (4/20, 4/22)	Sprint 2 – 6 **Project submissions deadline (4/26)***	Deployment		
	Week 16 (4/27, 4/29)	**Presentation (4/27, 4/29)**			

1. HW1

Tim and Jane meet after class to discuss HW1.

Tim

"Hi Jane, did you finish HW1 already?"

Jane

"Not yet. I'm still working on installing all the necessary tools and reading the materials. How about you?"

Marp, VSCode, and Git/GitHub Setup

Tim

"I just finished it a while ago. It took me some time to set up Marp and VSCode, but once I got the hang of it, it was pretty straightforward."

Jane

"Yes, actually, I have never used Marp before, so when I tried to read the markdown files, I was a bit confused at first. But after installing the Marp for VSCode extension[1], it became much easier to read the slides."

Tim

"Yeah, the Marp extension is really helpful. It allows us to make and view the slides directly in VSCode, which is convenient."

Jane

"I also had some trouble with Git and GitHub setup. I'm not very familiar with version control systems."

Tim

"I had some issues too, but I found a lot of helpful resources online, like YouTube tutorials and Stack Overflow posts."

Jane

"I think VSCode has built-in Git support, so once I understood how to use it, managing the repository became easier."

[1]Be sure to install the Marp for VSCode extension properly to start ASE courses.

Tim

"I was also impressed by the VSCode Git/GitHub integration features. It made committing and pushing changes to GitHub quite simple. In general, I think VSCode makes the development process smoother."

Reading Materials

Jane

"Did you read all the required materials about software engineering and ASE courses?"

Tim

"Yes, I did. The readings provided a good overview of what to expect in ASE courses and how team projects work. It helped me understand the importance of collaboration and communication in a team setting."

Jane

"I agree. The materials clarified a lot of my doubts about team projects. I feel more confident now about working in a team and contributing effectively."

Prototype Code and Tools Installation

Tim

"Also, exploring the prototype code was interesting. It gave me a glimpse of the project we'll be working on."

Jane

"True, also I installed all the project tools for the team and individual projects. Even though I don't fully understand the codebase and how to use the tools yet, I think we can figure it out as we progress."

Tim

"Definitely. In one of the required readings, it is mentioned that we should start from the 'Magic'. So, I'll just do and think everything is magic that works somehow. In the end, I'm sure I understand the 'Machine' behind the magic, and I believe the 'Making' will help."

Jane

"That's a good mindset. I'm sure we'll learn a lot as we work on the project together, and be the 'Master' sooner or later."

Classroom rules

Tim

"By the way, did you read the classroom rules for ASE courses? I think they are simple."

Jane

"Yeah. For students, two rules: (1) integrity and (2) two hats (professional & student). For the professor, also two rules: (1) fairness, (2) helping students to succeed. All the other rules are just details of these core rules."

Tim

"I like the integrity rule. It emphasizes the importance of honesty and ethical behavior in our work. As future software engineers, we need to uphold these values."

Jane

"Absolutely. Integrity is crucial in software engineering, as it builds trust among team members and stakeholders. I would not work with anyone who lacks integrity, even if they are highly skilled or not."

Deadline and Rules

Tim checks the deadline for HW1 submission.

Tim

"By the way, when is the HW1 submission deadline? I want to make sure I submit it on time."

Jane

"You can check the Canvas assignment page. We should submit it through the same Canvas page online."

Tim

"I'm especially concerned that there is no late submission allowed for the assignment. What if I miss the deadline due to some unexpected issues?"

Jane

"Actually, I understand that policy, because in real-world software engineering, meeting deadlines is crucial[2]. It teaches us to manage our time effectively and prioritize tasks."

[2]We are 'dead' when we miss the deadline.

Tim

"However, I read the rule that if I can request an extension of three days[3] before three days beofre the deadline with a valid reason, the professor may grant it to submit the assignment without penalty[4]. So, if I foresee any issues that might prevent me from submitting on time, I'll request an extension in advance. I mean three days before the deadline."

Jane

"That's a smart approach. Planning and communicating with the professor can help avoid any last-minute stress. I guess they're some of the core ideas of software engineering."

Tim

"What if I start late and find that I cannot make it to submit before the deadline? Is there any way to handle that?"

[3]This is default, but students can request longer extensions with valid reasons

[4]This is called three-three days late submission rule.

Jane

"Not exactly sure, but I think I can send an email to the professor explaining the situation with the date that I can submit the assignment. He may grant an exception, but I can't complain even if he doesn't, or even if he doesn't respond to my request. Anyways, it's always best to avoid such situations by staying organized and keeping track of deadlines."

Tim

"I see. Also, I read the rule that any requests after the deadline, assignment, midterm, or whatever, will not be considered, even if there is no response to the request. So, it's crucial to communicate any issues before the deadline[5]. Considering the professor is very busy,[6], it's better to reach out as early as possible to avoid missing the opportunity to get the permission before the deadline."

[5]This is to ensure fairness and consistency for all students.

[6]I believe the professor's #1 responsibility is to help students succeed, so the professor will do their best to respond as quickly as possible. But when a student requests a deadline extension hours before the deadline, it's practically impossible to respond to that kind of last-minute request.

Jane

"Yes, that's correct. It's important to respect deadlines and communicate proactively. Because in real-world software engineering, missing deadlines or no communication can have serious consequences for the entire team and project."

Tim

"Yeah, I agree. I also remember that a late submission, even with the professor's last-minute permission, will result in an automatic 20% penalty in the system.[7]. And there is no 2nd extension for the same assignment. So, it's best to avoid late submissions altogether by planning and managing our time effectively."

Jane

"I think it's following professor rule (1) fairness. Everyone should be treated equally when it comes to deadlines and penalties. Without this rule, everyone may request late submissions for various reasons, which could lead to chaos and unfairness for students who submit before the deadline."

[7]There is no exception. Under **any** circumstances, students will get a 20% deduction automatically

Tim

"That makes sense to me. It's unfair if some students are allowed to submit late without any penalties while others are not. At the same time, I can understand the rule that we request an extension three days before the deadline, because it shows that we already plan."

Jane

"Absolutely. Meeting deadlines is a crucial skill in software engineering, and practicing it now will benefit us in our future careers."

Tim

"Got it. I'll make sure to submit it before the deadline. If I know I can't do it, I'll contact the professor three days before the deadline to get permission to submit late. I sometimes forget about the deadlines, so it's good to double-check. I even think I need to make an application to remind me of important deadlines."

Jane

"That's a great idea. As a software engineering student, we should be able to create any applications that help us in our studies[8]."

[8]This is a good way to use the individual project opportunity.

Self Grading and Rubrics

Tim

"By the way, I'm not sure why do we need to grade ourselves for HW1. This is my first ASE course. Do you have any idea?"

Jane

"I heard that it reflects real-world software engineering practices. In the real world, developers should be responsible for their own work and contributions. Self-assessment encourages us to be honest about our progress and identify areas for improvement."

Tim

"That makes sense. But does the professor or the TA accept my grading as is? What if I overrate myself?"

Jane

"Actually, we grade based on the rubric provided in the instruction. So, as long as we follow the rubric and provide honest self-assessment, it should be fine.[9]"

[9] Of course, the professor or TA **will** check the self-grading for fairness and accuracy. If they find any discrepancies or anything wrong, they may adjust the grades accordingly.

Tim

"I see. I remember the professor mentioned that self-assessment is an important skill for software engineers, as it helps us reflect on our work and identify areas for growth."

Jane

"Exactly, also we know how many points we can get before the final grading.[10] This transparency helps us understand our performance and areas where we can improve."

Submission

They meet again after finishing HW1.

Tim

"Hi Jane, I just submitted my HW1 on Canvas. How about you?"

[10]Normally, professors grade and upload the results within one week after the submission deadline.

Jane

"I'm about to submit mine as well. I double-checked everything to ensure I followed the rubric and completed all the required tasks."

Tim

"Yes, make sure to check and grade yourself based on the rubric before submitting."

Jane

"Absolutely. I also made sure to review the submission guidelines to avoid any issues by checking all the checklist items."

Jane submits her HW1 on Canvas.

Jane

"Done! I just submitted my HW1. I earned 100% because I followed the rubric carefully and submitted on time. Also, I feel like I learned about software engineering to be ready for team and individual projects."

Tim

"Me too."

2. Team Formation & Team Name Selection

Common Mistake #1: Waiting Until Week 2:

❌ "I'll form a team later."

❌ "I'll think about the project later."

✅ Form teams as soon as possible

✅ Start brainstorming problems immediately

Why? Early start gives you more time to plan appropriately. Late teams may rush and make poor decisions[11].

Understanding Team Projects

Ken, Jane, and Tim form the team, and they need to understand what they need to do for the team project.

From the course discussion, they know that they need to build high-quality applications for their team and individual projects.

They also learned about Agile practices, such as Scrum and ceremonies like sprint planning, daily stand-ups, sprint reviews, and retrospectives, to manage their work effectively.

[11] In the agile process, we can adjust any decisions, but still, it's better to make a good decision from the beginning.

They checked the software prototype provided in the course materials. It has only basic features, so they need to enhance it significantly to meet user needs.

Ken

"So, we need to build a web app that helps students manage their assignments and deadlines effectively[12]. Did you read the prototype code?"

Tim

"Yes, I did. Even though the prototype is quite basic, but it gives us a starting point. But I'm really concerned because I'm new to web development; I don't have much experience with using JavaScript, Node.js, and MongoDB.[13]"

Jane

"Me too, but I think we can learn as we go. The important thing is that we understand the expectations and plan our work well."

[12]This is an example. The goal is different for each course.

[13]This is also an example of the technologies that the team is going to use for the team project. Each ASE course may use different technologies based on the course objectives.

Ken

"Exactly. Also, in the course discussions, we will learn all the necessary technologies and practices.[14] We just need to stay committed and work together as a team."

Jane

"OK. I think I can learn better when I work on real projects. So, I'm excited about this team project."

Team Name & Sending an Email

Ken

"First decision: team name. In industry, names describe function, not just sound cool."

Tim

"I thought about the team name 'Shooting Star'. Is it bad?"

Ken

"Not bad, just unclear. Compare these:"

[14]Because of time limitation, we may not be able to discuss all the technologies in depth during class discussions, but students will learn what they need to delve into to succeed in their projects.

Generic Name	Functional Name
"Shooting Star"	"Early Starter"
"Phoenix Rising"	"Assignment Tracker Team"
"Code Warriors"	"Student Deadline Manager"

Ken

"Which tells you what the team does?"

Jane

"The functional names. I get it—clarity over creativity.[15]"

Tim

"Yeah, 'Early Starter' sounds like a team that helps students begin assignments early. I remember Prof. Cho emphasized that we must start early to finish early and find unknown unknowns as quickly as possible, as that is the secret of successful software engineers."

Jane

"OK. I like the name 'Early Starter'".

[15]Of course, it's OK to combine both aspects if possible.

Ken

"Perfect. Now I'll email the instructor with proper formatting."

Professional Email Template

```
Subject: ASE 285 - Team 'Early Starter'
To: chos5@nku.edu

Hello, Dr. Cho.

This is the team information for the team project:

- Team name: Early Starter
- Team leader: Ken Johnson (johnsonk5@nku.edu)
- Team members:
  - Tim Brown (brownt5@nku.edu)
  - Jane Smith (smithj6@nku.edu)

Thanks,
Ken Johnson
```

templates/email.txt

*Use the **email.txt** template in the templates directory for your project.*

Why this format works:

- Subject includes course code (instructor teaches multiple courses)
- Clear, scannable bullet points

- Complete contact information
- Professional tone without being overly formal

Common Mistake 2: Vague Email Subjects

❌ Subject: “Team Info”

❌ Subject: “Question.”

✅ Subject: “ASE 285 - Team ‘Early Starter’”

Why? Instructors handle hundreds of emails. Help them help you! Be specific in the subject line, mainly including the course code to avoid confusion.

Prof. Cho receives the email, and he creates a Canvas page for the team named “Early Starter” with the provided members.

Prof. Cho also creates Canvas team project contribution pages for each team member.

Notice

There are 3-4 students per team, including the team leader. So, each team has one leader who manages the team and 2-3 team members who build the features.

A team of 4 is ideal, but a team of 3 is also acceptable[16], especially when a team member drops the course. A team of 5 is allowed only for a special reason.

Once the team is formed and the team leader is decided, it is unlikely to change the team composition unless there is a serious issue[17]. So, choose your team members and leader wisely.

Students who don't or can't join the team by week 1 will be assigned to a team by the instructor.[18]

[16] In the story, we intentionally have three members in a team for simplicity.

[17] For example, a team member drops the course or has a conflict with other members.

[18] Usually, these students are those who add the course late or have exceptional circumstances.

3. Set up GitHub Repo

Ken

"Next, we need to set up our GitHub repository and Canvas pages for collaboration and submission.

Ken

"I'll create a new GitHub repository named 'early-starter' under my account and add you both as collaborators."[19]

Ken creates a new GitHub repository named "early-starter" and adds Tim and Jane as collaborators.

He also makes the repository public so that the instructor or any stakeholders can access it easily.

Jane

"OK. I received the invitation and joined the repository."

[19]In real projects, it's better to create an organization account for the team and create repositories under that organization. But for simplicity, we use a personal account in this story.

Tim

"Me too. Now we can start working on the project together."

Got it — same Typst format, same function calls, just rewritten/cleaned so you can copy-paste directly. No Marp, no commentary, no format changes.

Below is a drop-in Typst rewrite of your content with more precise flow and minimal wording tweaks, preserving all structural constructs (`newpage_l2`, `kenchat`, `grid`, `bash_codeblock`, etc.).

Setup VSCode & Clone Repo

Ken

"Now that we have our repository, let's set up our local development environment. You already completed HW1, so VSCode and Git should already be installed."

Tim

"I already have VSCode, but how do I know Git is available?"

Ken

"In VSCode, look at the left sidebar. The third icon is called **Source Control**. If you see it, Git integration is already enabled by default.

If VSCode asks you to install Git, that means Git is not installed on your system yet."

Jane

"Okay, I see it. What's next?"

Ken

"Next, we'll clone the repository to your local machine. There are two ways to do this: (1) using the command line or (2) using VSCode's built-in Git features. Let's look at both."

Method 1: Clone via Command Line

Ken

"First, open a terminal and navigate to the directory where you want to store the project.[20] Then run the following commands:[21]"

[20] In VSCode, you can open the integrated terminal using View → Terminal.
[21] This is an example; replace it with your repository URL.

Bash

```
git clone https://github.com/ken-username/early-starter.git
cd early-starter
```

Tim

"I see a message saying 'Cloning into early-starter...' and it finished successfully."

Ken

"Great! The project is now on your computer. We are using HTTPS[22], so you will need to authenticate when pushing changes.

You can push your changes to my repository **only because** you have been added as a collaborator.

When authentication is required, VSCode will usually open a browser window and let you log in to GitHub directly."

Method 2: Clone via VSCode

Jane

"Instead of using the command line, can I do everything directly in VSCode?"

[22]You can use SSH later for better security and convenience.

Ken

"Absolutely.

In VSCode, open the Command Palette:

- Cmd + Shift + P on macOS
- Ctrl + Shift + P on Windows or Linux

Then type Git: Clone and press Enter."

Ken

"Paste the repository URL: `https://github.com/ken-username/early-starter.git`[23] and press Enter."

Choose a location on your computer to save the project."

Jane

"VSCode is asking if I want to open the cloned repository."

Ken

"Click Open. VSCode will open the project folder and automatically connect to Git."

"At this point, the repository is cloned and ready. You're now set up for development."

[23]This is an example; replace it with your repository URL.

Tim

"Awesome! I'm ready to start collaboration with you guys."

Ken

"Don't forget that VSCode is not an editor, it's a complicated IDE (Integrated Development Environment). So, don't open just one file to work on the project. Always open the entire project folder to make sure all the features work properly."

Jane

"Got it. I'll make sure to open the whole project folder in VSCode."

Verify the Setup

Ken

"Now, let's verify everything is set up correctly. Open the Source Control panel in VSCode by clicking the branch icon on the left sidebar."

Ken

"I see it says 'main' at the bottom left corner of VSCode. That's our main branch, right?[24]"

Ken

"Exactly! Now let's make sure we're all synchronized. In the terminal, run:"

Bash

```
git status
```

Jane

"It says 'On branch main' and 'Your branch is up to date with origin/main'. Looks good!"

Ken

"Perfect! One more thing—let's configure our Git identity so our commits are properly attributed. Run these commands with your own name and email:"

Bash

```
git config user.name "Your Name"
git config user.email "your.email@example.com"
```

[24]You will learn about branches in ASE 285. For now, just know that 'main' is the default branch where the stable code resides.

Tim

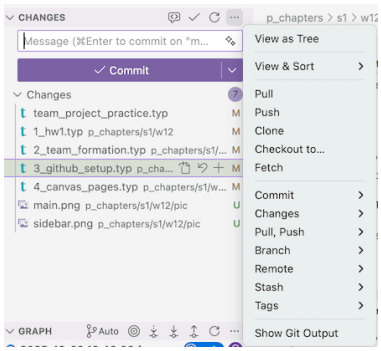
"Should I use my NKU email for this?"

Ken

"Yes, that's a good practice. Use your university email so the instructor can easily identify your contributions."

Jane

"All set! What's next?"



Ken

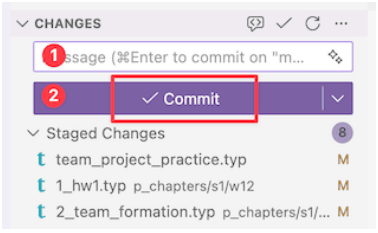
"Click the Git icon on the left sidebar to open the Source Control panel. If you click the three dots, you will see all the Git/GitHub you can use."

Tim

"Wow, there are so many options here! I see Commit, Pull, Push, Branches, and more."

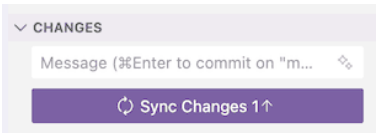
Ken

"Yes. Fortunately, you don't need to use all of them right away. Let's focus on the basic workflow that VSCode provides."



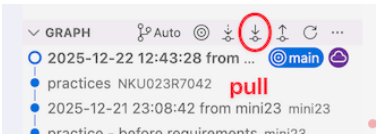
Ken

"But in most cases, all you need are Commit, Pull, and Push. Commit saves your changes locally. (1) Give your commit message and (2) click the checkmark to commit."



Ken

"Then, you can push, which uploads your committed changes to the remote repository on GitHub using the 'Sync Changes' button."



Ken

"Then, you can pull, which fetches and integrates changes from the remote repository on GitHub using the 'pull' button."

Jane

"Got it. So, the typical workflow is: make changes, commit them, push to GitHub, and pull any updates from others. I'm afraid that I'm in the 'Magic' state out of the M4[25]."

[25] Magic-Machine-Master-Make

Ken

“Don’t worry, Jane. You will learn all the details of Git and GitHub in ASE 285, including the algorithm and great ideas behind It. For now, just remember the basic workflow you mentioned.”

Tim

“Sounds good. I’m excited to start working on our project together!”

Ken

“Now we’re ready to start working on the project. But before we write any code, we should set up our Canvas pages for documentation and submission.”

4. Canvas Pages

Ken

"I checked Canvas, and the professor already created our team project, and individual contribution pages for the team project."

Ken

"Sometimes, we cannot edit the page due to edit privilege. In that case, you need to send an email to the professor to update the privilege so that we can edit the page."

Jane

"Got it, but for now, I can edit the team and individual contribution pages without any issue."

Tim

"I found that the team project Canvas page already has the individual contribution section set up.So, anyone can easily access individual contribution page from the team project page.[26]"

Ken

"Great! Now we are all set up for the team project. I'll take care of all the team project Canvas page and GitHub repository. But I ask you both to keep me updated on your progress by updating the individual contribution page regularly so I can track our overall progress.[27]"

Ken adds the GitHub repository link and the team project Canvas page link to the individual contribution pages for easy access.

Tim and Jane also add the GitHub repository directory link that they will be working on to their individual contribution pages.

[26]If thre is no link to the individual contribution page on the team project page, you can manually add the link by editing the team project page and inserting a hyperlink to the individual contribution page.

[27]There is no standard format for the team or individual contribution page. Teams should find the format that works best for them. But the format should be organized so that anyone can easily understand the progress of the team project and individual contributions.

Week 2 - 3

Now that the team 'Early Starter' has set up their GitHub repository and Canvas pages, it's time to focus on PPP in the week 4.

1. README.md
2. The Structure of GitHub Repository
3. The Structure of Canvas Pages
4. Team Rules
5. Making PPP Slides
6. HW2

Stages	Weeks (M/W)	Sprints	Topics (Theory)	Modules	HW
Stage 1 (Preparation)	Week1 (1/12, 1/14)		Introduction		
	Week 2 (~~1/19~~, 1/21)	Project Discussion** Team Setup*	Project Preparations		HW1 (Installation) (1/25)*
	Week 3 (1/26, 1/28)		Project Preparations		HW2 (Project Plan) (2/1)*
Stage 2 (Prototype /MVP)	Week 4 (2/2, 2/4)	**Project Plan Presentation (2/2)** Sprint1 - 1	High-Level Languages	Module 1	
	Week 5 (2/9, 2/11)	Sprint1 - 2	High-Level Languages		
	Week 6 (~~2/16~~, 2/18)	Sprint1 - 3	High-Level Languages		
	Week 7 (2/23, 2/25)	Sprint1 - 4	Software Process		HW3 (Module 1) (2/27)**
	Week 8 (**3/2, 3/4**)	**Sprint 1 Presentation (3/2)**	**Midterm (3/4), on Canvas no class no office hours**		HW4 (Sprint 1 Retrospect + Sprint 2 Plan)(3/1)*
Stage 3 (Project)	Week 9		Spring Break (no class) **No classroom/office hours**		
	Week 10 (3/16, 3/18)	Sprint 2 – 1	Git	Module 2	
	Week 11 (3/23, 3/25)	Sprint 2 – 2	Git		
	Week 12 (3/30, 4/1)	Sprint 2 – 3	Security		
	Week 13 (4/6, 4/8)	Sprint 2 – 4	Security		
	Week 14 (4/13, **~~4/15~~**)	Sprint 2 – 5	**Midterm (4/15, on Canvas no class/office hours)**	Module 3	HW5 (Module 2) (11/22)***
	Week 15 (4/20, 4/22)	Sprint 2 – 6 **Project submissions deadline (4/26)***	Deployment		
	Week 16 (4/27, 4/29)	**Presentation (4/27, 4/29)**			

1. README.md

After setting up GitHub and Canvas pages, and then completing the HW1, the team focuses on understanding the problem domain and defining features and requirements for the project.

Based on their discussion, they could understand the problem domain, define features and requirements, make data model and architecture diagram, and acceptance tests outlines[28].

Writing README.md

Ken

"OK. We have all the necessary information to write the README.md for our project."

Tim

"Why do we need a README.md file? Can't we just split the work and upload everything to GitHub?"

[28]Read the Project Theory story if you need to understand these concepts in detail.

Ken

"Good question. The README.md file serves as the main documentation for our project. In a way, this is like the summary of a book. It gives readers a quick overview and helps them decide if they want to read further."

Jane

"That makes sense. A well-written README.md can make our project look more professional and organized."

Ken

"Yes, it's also for stakeholders or clients. Stakeholders are busy, so they appreciate clear and concise documentation. A good README.md can also help us stay organized and focused on our goals."

Tim

"I see that it's also for me. This README.md will be like a summary of our project that anyone can read to understand what I've been doing. I think I can even use it for my resume."

Ken

"Also for the GitHub, the README.md is the first thing people see when they visit our repository. It should provide a clear overview of our project"

Ken shows an example README.md structure that he used in his previous projects:

```
1. Project title and description
2. Problem domain and motivation (what is the problem
and why it matters)
3. Features and requirements (what is our solution to
the problem)
4. Data model and architecture (overall structure and
design)
5. Tests (how we ensure quality)
6. Team members and roles
7. Links to code, tests, documentation, and
presentations
```

Jane

"Well, that sounds like a lot of information. How can we put everything in this one file?"

Ken

"Don't worry, we will have a dedicated `doc` directory that will have all the detailed documents, diagrams, and presentations. The README.md will provide an overview and link to those detailed files. Think of it as a table of contents for our project."

Tim

"I see that we should clearly define the problem domain and our solution in the README.md. That will help anyone understand what we're building and why. Then, they can dive into the details if they want to."

Ken

"Exactly! Otherwise, people might not understand **the context** of our project. If we just write down features and requirements without explaining the problem, people are not interested in it can be confusing."

Ken shares a README.md template with the team:

templates/README.md

Use the README.md template in the templates directory as a starting point for your README.md file.

Tim

“Wow, it covers pretty much everything we discussed.”

Ken

“The README.md is a living document. We should update it regularly as our project evolves. Whenever we make changes to features, architecture, or any other aspect, we should reflect those changes in the README.md. This way, it always stays relevant and accurate.”

Tim

“Now that makes more sense to me. We should keep adding more code, tests, and documentation as we progress. Then, we keep updating the README.md to reflect our current state.”

2. The Structure of GitHub Repository

Ken

"Now, we need to push the README.md to GitHub repository. Before we do that, let's talk about how to organize our GitHub repository. A well-structured repository makes collaboration much easier."

Tim

"What does a typical project structure look like?"

Ken

"Good question. Let me show you a standard structure we should follow for our Early Starter project."

Ken creates a diagram showing the repository structure:

templates/github_directory.txt

Use the github_directory.txt template in the templates directory as a starting point.

Directory Tree

```
early-starter/
├── README.md                 # Main project documentation
├── docs/                     # All documentation files
...
├── src/                      # Source code
...
├── tests/                    # Source code
...
├── individual/               # Individual contributions
...
└── .gitignore                # Git ignore file
```

Jane

"I see. So we have separate folders for documentation, source code, tets, and individual work. That makes sense."

Tim

"When stakeholders need to find the documentation or code, they can easily navigate to the right folder."

Ken

"Exactly. Let me explain the purpose of each main directory:"

Jane

"Sure, we already discussed README.md earlier. So, the first one is the docs/ folder."

Ken

"Before we start, remember this is just a guideline. We are doing this to solve problems by managing complexity, so if the github directory structure makes problem harder to solve, we are not solving problems effectively. So, each team should find the best structure that works for them, not following this structure blindly."

Tim

"Got it, but I think no matter what, the github structure should be easy to understand and navigate to anyone who visits the repository without any prior knowledge."

Jane

"Agreed. That's the reason why we need a well-defined structure like this one. So, what's the docs/ folder for?"

docs/

Tim

"The docs/ folder contains all the documentation files for our project."

Jane

"So, when we make any documents, diagrams, or presentations, we put them in this folder?"

Directory Tree

```
├── docs/                    # All documentation files
│   ├── presentation/        # Project Plan Presentation
│   │   ├── ppp_early_starter.md
│   │   ├── ppp_early_starter.PDF
...
│   ├── architecture/        # Architecture diagrams
│   │   ├── data_model.md
│   │   └── system_architecture.png
│   ├── features/         # Architecture diagrams
│   │   ├── overall.md
│   │   ├── frontend_features.md
│   │   └── database_features.png
│   └── requirements/        # Requirements documentation
...
│       └── acceptance_tests.md
```

Ken

"Yes, even our weekly presentations go in the docs/ presentation/ folder. This keeps all our documentation organized and easy to find."

Jane

“Do we need to store the markdown source files as well as the PDF versions of the presentations?”

Ken

“Yes. The idea is to keep everything so that we can easily update or modify them later.”

Tim

“So, the architecture/ folder contains diagrams like data models and system architecture, but we need to add the source files as well, right?”

Ken

“Exactly. This way, if we need to make changes, we can do that easily without searching for the original/source files.”

Jane

“It’s interesting to notice that features and requirements are part of documentation. I used to think of them as part of the code or design.”

Ken

"Good observation. Features and requirements define **what** we are building, so they belong in documentation. They guide our development process and ensure everyone is on the same page about the project's goals."

Tim

"I see, so we can refer back to these documents whenever we need to clarify what features we are implementing or what requirements we need to meet."

src/

Ken

"Next is the src/ folder, which contains all our source code."

Directory Tree

```
├── src/                        # Source code
│   ├── lib/                    # Library code
│   │   ├── models/             # Data models
│   │   ├── screens/            # UI screens
│   │   ├── services/           # Business logic
│   │   └── widgets/            # Reusable UI components
...
│   └── main.js                 # App entry point
]
```

Tim

"I think it's straitforward, and easy to organize. All we have to do it put all of our code results in here!"

Ken

"Yes, it's straitforward that all the code belong in src/. However, 'no' to your last sentence. We need to organize the code properly in sub-folders like this example; models/, screens/, services/, and widgets/."

Ken

"It's always a good idea to keep the library code (lib/) separate from the other parts of the code. This separation helps in managing complexity as our codebase grows."

Jane

"I think the code folder structure should be different per project. For example, if we were building a backend service, we might have different folders for controllers, routes, and database access."

Ken

"Exactly, I simplified and showed only the Flutter app structure here as an example. Depending on the project type, the src/ folder structure can vary significantly. The key is to keep it organized and intuitive for anyone working on the codebase."

Tim

"Got it. For our 3-Tier architecture, we may have three different main folders for frontend, backend, and database code."

Ken

"Yes, even in some cases, we may need to have separate repositories for different tiers. But for now, let's keep everything **simple** in one repository and organize the src/ folder accordingly. We can extend or modify the structure as needed later."

tests/

Ken

"The tests/ folder contains all our test code."

Directory Tree

```
├── tests/                    # Source code
│   ├── lib/                  # Common Library
│   ├── coverage/
│   ├── unit_tests/
│   ├── integration_tests/
│   ├── regression_tests/
│   └── acceptance_tests/
```

Jane

"Having a dedicated tests/ folder makes it easy to find and run tests."

Tim

"Yeah, especially for the regression tests, we know what tests to run. HaHa."

Ken

"Exactly, but this folder becomes larger and more complex as we add more tests. So, we need to organize the tests properly into sub-folders like unit_tests/, integration_tests/, regression_tests/, and acceptance_tests/ to keep things manageable."

Jane

"Do we need to keep all the test scripts, code coverage, and results in this folder?"

Ken

"Yes, we don't want to surprise anyone later. This should be the place where all test-related files are stored. This way, anyone can easily find and run the tests whenever needed."

Tim

"Also, when we need some common tools or libraries for testing, we can put them in the lib/ folder inside tests/. That way, we can reuse them across different test cases."

Ken

"Yes, but don't forget that this is also an example. Adapt and adjust the structure as needed for your specific project. The key is to keep it organized and easy to navigate."

individual/

Ken

"The individual/ folder contains each team member's contributions."

Jane

"I'm not sure, but isn't it better to use the Canvas individual project pages for that?"

Ken

"Good question. While we do have individual Canvas pages, having a dedicated individual/ folder in our GitHub repository allows us to keep track of our contributions in one place. This is especially useful for the instructor to see who did what in the team project."

Tim

"I see, and I also see the links.md files for each member. So, we can link our individual Canvas pages, code, tests, or documentation here as well, so anyone can understand each members's contributions easily."

Ken

"Yes, and it's also good for us to document our progress and contributions. Each member can maintain their own progress.md file to track what they've done, challenges faced, and next steps. This transparency helps with accountability and coordination within the team."

Jane

"That makes sense. So, basically the idea is that we use all the tools available to us to make our collaboration effective and transparent."

Ken

"Yes, that's the idea. If you think about it, it makes only sense to those who actively participate in the project. So, we need to make sure that our individual contributions are well-documented and easily accessible. This way, everyone knows who is responsible for what, and we can avoid any confusion later on."

Tim

"Got it. So, for example, I can document my testing efforts in my progress.md file, and link to my individual Canvas page in links.md. This way, the instructor can see my contributions clearly."

Jane

"And the team members and leaders too. This way, we can evalute each member's contributions effectively at the end of the class."

Directory Tree

```
├── individual/                # Individual contributions
│   ├── ken/
│   │   ├── s1_progress.md
...
│   │   └── links.md
...
│   └── tim/
│       ├── s1_progress.md
│       └── links.md
```

Other files

Ken

"Finally, we have all the other files."

Jane

"I know the .gitignore file. Without this file, I made my repositories really messy with unnecessary files."

Ken

"Haha Yes, I was there too. Also, any project related files and folders belong here. For example, if we need to make shell scripts for automation or tools for project management, we can put them in a tools/ folder."

Directory Tree

```
├── tools/                    # Tools for project management
│   ├── LoC/
...
│   └── scripts/
└── .gitignore                # Git ignore file
```

3. The Structure of Canvas Pages

Ken

"Now that we have our GitHub structure set up, let's organize our Canvas project page. This is where we'll submit our work and provide links to all our documentation, code, tests, and other project related materials."

Tim

"OK. When we need to find anything about our project, we can just go to the Canvas page and find the links there, right?"

Jane

"Also, for individual team member contributions, we can visit their individual Canvas pages from the team page. That way, everything is connected and easy to navigate."

Ken

"Yes, the Canvas project page is our submission hub. It should have clear links to all our work, making it easy for anyone, including ourselves, to review everything. Let me show you an example structure."

Ken shares his screen showing the Canvas page structure[29]:

templates/canvas.txt

Use the canvas.txt template in the templates directory as a starting point.

Team Information & Project overview

```
# Team Information
**Team Members:**
- Ken (Team Leader) - ken.student@nku.edu
  - [Individual Canvas Page](canvas-link-to-ken)
- Jane - jane.student@nku.edu
    - [Individual Canvas Page](canvas-link-to-ken)
- Tim - tim.student@nku.edu
    - [Individual Canvas Page](canvas-link-to-ken)

# Project overview

## Project Description
...
## Problem Domain
...

# Features and Requirements
...
```

[29]We use markdown format for the Canvas page content. You can use the Canvas extension to add a markdown format to the Canvas page

```
## Features & Requirements
...
```

Tim

"Well, it's basically the same as the README.md structure, but organized for easy navigation on Canvas."

Ken

"Exactly! The README.md has all the most important information, but it's in the GitHub repository. So, it's OK to store the same information in the Canvas page. If anyone needs to find deails about our project, they can visit the GitHub repository from the Canvas page."

Jane

"But what if we have too many features and requirements? Won't that make the Canvas page too long and hard to navigate?"

Ken

"Good point. This is a guideline, not a strict rule that everyone should follow. If we have too many features and requirements, we can summarize them on the Canvas page and link to the detailed documents in the GitHub repository. The key is to make it easy for anyone to find what they need without overwhelming them with too much information at once."

Tim

"That makes sense. So, we can have a brief overview on the Canvas page and link to the detailed documents in GitHub if needed."

GitHub Repository

Ken shows the next part of the Canvas page content:

```
# GitHub Repository

**Main Repository:** https://github.com/ken-username/
early-starter

## Quick Links to Key Directories

- [Documentation](https://github.com/ken-username/
```

```
early-starter/tree/main/docs)
- [Source Code](https://github.com/ken-username/
early-starter/tree/main/src)
...
```

Jane

"Yes, this section is about the GitHub repository links. This makes it easy for the instructor to find our code and documentation quickly."

Documents

Tim

"I think the Document section is really important. It has all the key documents links for our project, including the PPP files, diagrams, and acceptance tests."

Jane

"Yes, if anyone needs any documents about the project, they can find it all in one place here."

```
# Documents

## Requirements
- [Acceptance Tests](https://github.com/ken-username/
arly-starter/blob/main/docs/requirements/acceptance_
tests.md)

## Project Plan Presentation (PPP)
- [PPP Document](https://github.com/ken-username/
early-starter/lob/main/docs/ppp/ppp_document.md)
- [PPP Slides (Marp)](https://github.com/ken-
username/arly-starter/blob/main/docs/ppp/ppp_slides.
md)
- [PDF Version of Slides](link-to-pdf)

## Diagrams
- [Data Model](https://github.com/ken-username/early-
starter/lob/main/docs/architecture/data_model.png)
- [System Architecture](https://github.com/ken-
username/arly-starter/blob/main/docs/architecture/
system_architecture.png)

...
```

Ken

"Yes, having all these links in one place makes it easy for anyone to find everything they need to understand and evaluate our project. Now, let's look at how we can document individual contributions."

Progress

Ken

"And at the end of the team page, we usually have the progress section where we link our milestones and weekly progress. This shows how we've been advancing our project over time."

Jane

"That makes sense. It gives a clear timeline of our work in the form of milestonesand shows that we're making steady progress in the form of weekly presentation."

```
# Progress

## milestones

- Milestone 1: Team formation and GitHub setup -
Completed
- Milestone 2: Problem analysis and PPP preparation -
In Progress
- Milestone 3: Initial development sprint - Upcoming
...

## Weekly Progress
```

```
- Week 4 Presentation: [Link to the Github page]
- Week 3 Presentation: [Link to the GitHub page]
```

Individual Contribution Pages

Ken

"Let's talk about individual contribution pages next. Each team member should have their own Canvas page that details their specific goals, features, and contributions to the project. This helps anyone see who did what and how each person contributed to the overall effort."

templates/canvas_individual.txt

Use the canvas_individual.txt template in the templates directory as a starting point.

Jane

"The overall structure seems to be the same as the team page, but focused on individual contributions."

Ken

“Exactly! Each individual page includes:

1. Personal Information & Goals
2. Links to GitHub & Documents
3. Progress

Here’s an example structure for an individual Canvas page:“

Personal Information & Goals

```
## Member Information

- Ken (Team Leader) - <ken.student@nku.edu>
  - [Team Canvas Page](canvas-link-to-team)

# Features and Requirements

...

## Features & Requirements

...
```

Jane

"The team page has all the features andrequirements, but the individual page focuses on what that specific person is responsible for."

Tim

"Also, the link to the team Canvas page is important. It connects the individual page back to the team page, making navigation easy."

Links to GitHub & Documents

```
# GitHub Repository

**Main Repository:** <https://github.com/ken-
username/early-starter>

## Quick Links to Key Directories

- [Documentation](https://github.com/ken-username/
early-starter/tree/main/docs)
- [Source Code](https://github.com/ken-username/
early-starter/tree/main/src)
...

# Documents
```

```
## Requirements

- [Unit Tests](https://github.com/ken-username/arly-starter/blob/main/docs/requirements/acceptance_tests.md)

## Diagrams

- [Data Model](https://github.com/ken-username/early-starter/lob/main/docs/architecture/data_model.png)
```

Jane

"Compared to the team page, the individual page has fewer links, focusing on what that person worked on."

"Canvas for Managing Schedules"

Tim

"So, what's the main purpose of the Canvas team project page? Is it just for sharing documents and presentations?"

Ken

"Good question. While sharing documents and presentations is one purpose, the main purpose of the Canvas team project page is to manage our project schedule effectively. We need to set clear milestones and deadlines to ensure we stay on track throughout the project."

"Deadlines vs Milestones"

Tim

"I thought the deadlines provided in Canvas were enough. Why do we need to set additional milestones?"

Ken

"We have the **what** (features). Now we need the **when** (schedule)."

Tim

"Isn't the deadline from Canvas enough?"

Ken

"No. Think of it this way:"

Concept	Definition	Purpose
Deadline	Final due date (non-negotiable)	Hard constraint
Milestone	Intermediate checkpoint	Progress tracking & safety net
Sprint	Fixed time period (1-2 weeks)	Rhythm & planning

Jane

"So milestones are safety checkpoints before the deadline?"

Ken

"Exactly. If you miss multiple milestones, it's a red flag—time to adjust scope or get help."

Creating Your Milestone Schedule

Ken

"Here's the approach: work backwards from the deadline."

Example Schedule Calculation:

```
Final Presentation: Week 16 (Dec 10)
Submission Deadline: Week 16 (Dec 8, 11:59 PM)

Work backwards:
```

```
Week 16 (Dec 8):  Final submission & testing
Week 15 (Dec 1):  Integration testing complete
Week 14 (Nov 24): Feature 3 complete (Thanksgiving break)
Week 13 (Nov 17): Feature 2 complete
Week 12 (Nov 10): Feature 1 complete
Week 11 (Nov 3):  Sprint 1 MVP complete
Week 10 (Oct 27): Database & API routes working
Week 9 (Oct 20):  Project structure & setup complete
```

Tim

"But what if we can't estimate accurately? We've never done this before."

Ken

"That's the point of Agile. You adjust as you learn. But you need data to adjust."

Julie

"That's why we track **velocity**—how many story points we complete per sprint."

The team discusses and agrees on the milestone schedule based on their project plan and deadlines.

Progress

Ken

“Finally, the progress section on the individual page highlights that person’s milestones and weekly progress.”

Jane

“So, this is the section where each team member shows what they’ve accomplished individually each week.”

Tim

“And the progress should be aiming to finish the milestones we set as a team.”

```
# Progress

## Milestones

- Milestone 1: Individual Goal

...

## Weekly Progress

- Week 3 Progress: [Link to the GitHub page (if
necessary)]
  - LoC: ...
  - Tests: ...
```

Ken

"Exactly! I **need** this section for weekly presentation. So, make very sure you should update this section every week, by the end of each week. Otherwise, I can't make the presentation that includes your contribution."

Tim

"What if I forget to update it?"

Ken

"Then, it's totally your responsibility and problem. I will **not** remind you, and your contribution will be missing in the presentation."

Jane

"Isn't it too harsh? Isn't it your job to remind us to update our progress?"

Ken

"Not at all. Remember the student rule #2 wearing two hats. In the team project, you are a professional software engineer. In the real world, nobody reminds you to update your progress. You need to manage your own work and responsibilities. If you forget to update your progress, it's your problem, not mine."

Jane

“I see your point. It’s important to take responsibility for our own work. But I think we need to understand our roles and responsibilities clearly. I think we should have a team rule documented so that there should be no confusion later.”

Tim

“I agree with Jane. Let’s make a team rule document that clearly defines our roles, responsibilities, and expectations. That way, everyone is on the same page from the start. I’m sure we can avoid unnecessary conflicts when we evaulate peers and others.”

Ken

“I want you to understand team work. Even though you can’t make contributions every week, as long as you contact me and let me know your situation, I will understand and accomodate you. But if you just ignore and disappear without any notice, that’s unprofessional and unacceptable. So, please make sure to communicate with me if you have any issues or concerns.”

Tim

“Yes, ‘No surprises“ rule!

Ken

“Exactly! I’m not asking you to share your personal problems, but at least let me know if you are facing any challenges that might affect your work so that I can be ready and accommodate you. Communication is key in a team environment. Without this, we are not a team.”

Jane

“I totally agree. I don’t want to work with someone who just disappears without any notice, no matter how good that member might be. Communication is essential for effective teamwork.”

4. Team Rules

Ken

"OK. Let's talk about the team rules. It's important to set clear expectations and guidelines for how we will work together as a team. This will help us avoid misunderstandings and conflicts later on."

He stresses the importance of having clear team rules to ensure smooth collaboration.

Without Rules	With Rules
Expectations unclear	Everyone knows commitments
Violations go unaddressed	Violations have consequences
Unfair to reliable members	Fair evaluation for all
Team leader powerless	Team leader can escalate

Jane

"We already had a lot of discussions about team roles and responsibilities when we set up the Canvas page. So, in a way, it's just writing down what we already agreed on."

Ken

"Exactly. But rule becomes the rule only when it's documented and agreed upon by all team members. Also, make it very clear about the penalty when the rule is violated. We should know that we evaluate peers based on these team rules."

"Rules That Work"

Common Mistake #1: Rules Without Enforcement:

❌ Write rules but never follow through

❌ Enforce rules inconsistently (favorites get passes)

✅ Apply rules fairly to everyone (including team leader) ✅ Document violations and escalate as promised

Why? Rules without enforcement are worse than no rules. They breed resentment and unfairness.

Tim

"I agree with your idea. When a team member, or even team leaders, violated the rule, there should be a clear consequence. Otherwise, it can lead to resentment and frustration among team members."

Ken

"Right. This is what makes us professional software engineers. In the real world, if you don't follow the team rules, there are consequences, such as being removed from the project or even losing your job. So, we need to take this seriously."

Real-World Parallel:

Professional teams have "Working Agreements" or "Team Charters" that document:

- Communication norms
- Quality standards
- Accountability measures
- Escalation procedures

Your team rules are your working agreement.

Jane

"And show no emotions under any circumstances. We need to be professional and objective when evaluating peers based on the team rules."

templates/team_rules.txt

Use the team_rules.txt template in the templates directory as a starting point.

Definition of Done

Jane

"For each requirement, we should define what 'Done' means:"

A requirement is "Done" when:

✓ Code implemented and follows team conventions ✓ Unit tests written and passing ✓ Integration tests passing ✓ Acceptance criteria met (user story can be demonstrated) ✓ Code reviewed and approved by another team member ✓ Documentation updated (README, inline comments) ✓ No critical bugs remaining ✓ Deployed to staging environment

Jane

"This definition ensures we don't just write code—we deliver production-ready features."

Finalizing Team Rules

The team discusses the team rules and made a clear agreement on communication and responsibilities. Then, they made a team rule document based on their discussion.

Ken

“OK. Now we have our team rule, I’ll upload it to the GitHub repository and link it on the Canvas team page. This way, everyone can refer to it whenever needed.”

Team

“Sounds good.”

5. Making PPP Slides

Now, the team has all the information for their project. They have the problem domain, features, requirements, data model, architecture, and acceptance tests defined.

They have their GitHub repository and Canvas pages set up. They even have the team rules documented and agreed upon. They are ready to present their project plan next week.

Ken

"Well, it's time to create the PPP (Project Plan Presentation) files. As everything is almost ready, we just need to put everything together into the PPP document and slides."

Tim

"I'm new to any team ceremonies. What exactly is PPP?"

Julie joins the disscussion.

Julie

"PPP stands for Project Plan Presentation. It's an overview of our project plan, basically it's explaining what is already written in the READM.md to stakeholders."

Ken

“Exactly. In a way, we have all the discussions and documents ready for the PPP. We just need to organize everything into a presentation format.”

Jane

“So, we need to create PPP slides for the presentation, right?”

Ken

“Yes, I’ll make the skeleton of the PPP slides using Marp. Then, I want your feedback, and fill up your sections.”

Tim

“We already have all the information we need. It should be straightforward to create the slides.”

Julie

“Yes, but remember PPP is also about our storytelling skills. We need to make sure our slides are clear, concise, and engaging. We want to capture the audience’s attention and make them interested in our project.”

Jane

"So, in the PPP, do we all need to present our parts?"

Ken

"Yes, except for the weekly progress presentation, all of the presentation ceremonies are team presentations. So, we all need to be prepared to present our parts."

Julie

"Also, software engineers should be good communicators. We are solving problem by managing complexity. If we can't explain our ideas clearly, it's hard to convince others to support our solutions due to unnecessary complexity caused by confusion or pure communication."

Tim

"Wow. Everything is problem solving, even the presentation."

Ken

"Haha, yes. The good news is that solving problems is fun and rewarding. The better we communicate, the more effective we are as software engineers."

Tim

"One last question. We don't show a lot of contributions for now. Will that be a problem?"

Tim

"Not at all. PPP is about showing that teams are ready to start the team project togethe with some evidence of initial contributions. As long as we have a clear plan and defined roles and responsibilities, we are good to go."

Ken creates a skeleton of the PPP slides using Marp and shares it with the team for feedback and contributions.

They review the slides together, make necessary adjustments, and ensure that everyone is comfortable with their parts.

Ken

"I think the data model is OK, but the architecture diagram is not clear for the PPP. Jane, can you improve the architecture diagram to make it more understandable for the audience?"

Jane

"Sure, I can do that; I also think it's too complicated for the audience who may not be familiar with all the technical details. I'll make sure to simplify the diagram and highlight the key components and their interactions."

They keep on discussion and refinement until they are satisfied with the PPP slides that tell their project story clearly and effectively.

Now, they are ready to submit their preparation of their project as the HW2.

6. HW2

The team discusses the tasks and plans for completing HW2.

Tim

"The HW2 is a group submission[30], right? So, we need to make sure everyone contributes and effectively."

Jane

"Yes, but don't forget that we also upload the individual project presentation and schedule independently. So, we need to manage both team and individual projects well for the HW2[31]."

Ken

"Exactly, as a team leader, I'll make sure everything is correctly organized and submitted on time. But before I do that, I will share the HW2 before the submission deadline so that everyone can review it and provide feedback. This way, we can ensure that our submission is of high quality and meets all the requirements to earn 100%."

[30] Only the team leader uploads the team project submission, and all the team members earn the same points.

[31] In this story, we focus mainly on the team project, but individual projects are equally important.

Ken finishes the HW2 submission, and Tim and Jane give feedback. Ken uploads the final version before the deadline.

> **Ken**
>
> "Great job, team! We successfully completed and submitted our team project section of HW2 on time. Thank you all for your hard work and contributions. Let's keep up the good work for the rest of the course!"

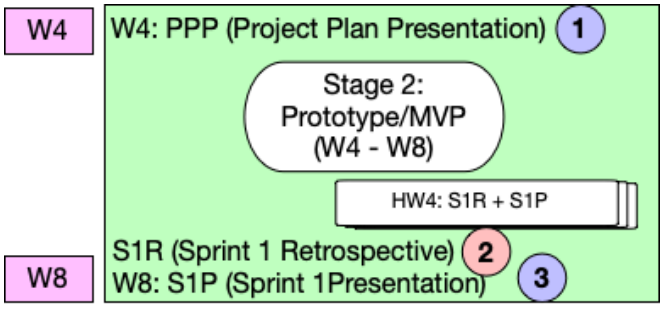
Now, all the team members work to finish their individual project presentation and schedule for the HW2 submission. As they already know the rules and expectations, they manage their individual projects effectively.

Section

Stage 2: Prototype/MVP

Goal: S1P (Project 1 Presentation) @ W8

Weeks: W4 - W8



1. PPP (Project Plan Presentation)
2. **S1R (Sprint 1 Retrospective)**
3. **S1P (Sprint 1 Presentation)** <- Goal
4. S2R (Sprint 2 Retrospective)
5. FP (Final Presentation)

1. PPP

This is Week 4, and we are conducting the Project Plan Presentation (PPP) ceremony in the first week of Stage 2.

PPP(Project Plan Presentation) Ceremony

Team ‘Start Early’ is ready to present their project plan for Stage 2: Prototype/MVP.

In the classroom meeting, Prof. Cho welcomes teams. He opens the Canvas team project page for ‘Start Early’ and shares the screen.

All the team members, Ken, Jane, and Tim, are at the podium, ready to present their projec to the class and stakeholders.

Prof. Cho

“Welcome, team ‘Start Early.’ Today is the PPP ceremony for starting Stage 2. Please proceed with your presentation.”

As all the information about the team project is on the Canvas team project page, he doesn’t need to explain anything else. He clicks the link to the GitHub repository where the team has uploaded their PPP slides in PDF format.

Ken

“Hello, everyone. We are team ‘Start Early.’ I’m Ken, the team leader. This is Jane and Tim, my team members. Today, we will present our project plan for our Task Management Application.”

Ken

"My team 'Start Early' is developing a Task Management Application that allows students to manage their assignment deadlines effectively.[32] We'll walk you through our understanding of the problem, the features we plan to build, and our sprint plan."

Step 1: Understanding Why - The Problem Domain

Jane

"Let me start by explaining the problem domain.[33]"

Jane displays a slide showing students struggling with deadlines.

Jane

"We observed that students often have multiple assignments with various deadlines across different courses. Without a centralized system, they frequently forget about upcoming deadlines and start working on assignments too late."

[32]It's a good to start with what we are going to do, because this is what stakeholdrs want to know.

[33]This is an example of possible format of PPP, teams can choose any format that works best for them. As long as they have a good storytelling to convince stakeholders the problems are worth solving and their solutions are appropriate, and their plans are practical and feasible, any format of presentation is OK.

Jane

"Through our research and interviews with fellow students, we identified two main problems:"

Problem 1: No proactive notifications for deadlines ⟶ Students don't get reminders until it's too late

Problem 2: Limited browser-based accessibility ⟶ Students can't easily check assignments from anywhere

...

Step 2: From Problems to Features

Ken

"Now, let me explain how we translated these problems into features."

Problem 1 ⟶ Feature 1: Email Notification System "App notifies the deadline N days before by email"

Problem 2 ⟶ Feature 2: Web-based CRUD Operations "Students can do CRUD operations via web interface"

...

Ken

"These features directly address our users' needs without specifying HOW we'll implement them. We're keeping our options open for the best technical solution."

Jane is going to implement Feature 1, and Tim is going to implement Feature 2. They will collaborate on architecture and integration.

Feature and Requirements Breakdown

Jane

"For Feature 1, we broke it down into specific user stories using the Actor-Goal format:"

Instead of explaining the requirements detail, Jane shows the requirements on the slide. The stakeholders already know where to find these requirements on the Canvas team project page, they don't need to explain everything in detail.

Instead, she focuses on explaining the structural aspects of the requirements, especially the metrics, the test counts and number of features/requirements that will be used to show their progress through burndown charts later.

Jane

"You can find all the requirements on our Canvas team project page. Let me explain the structure of the requirements we created for Feature 1."

RQ1: As a student, I want to get a notification email from the app N days before the schedule so that I do not miss any deadlines.

RQ1-1: As a student, I want to specify my email so that the system knows where to send notifications.

RQ1-2: As a student, I want to specify N days so that I choose when to receive emails.

RQ2: As a student, I want to click the checkmark of the schedule so that I can confirm that I am correctly notified and ready to finish the task.

Jane

"Notice that RQ1 is the most innovative requirment for the first feature, because this feature solves the main problem of no proactive notifications.

She explains the plan for the implementation.

Jane

"I am going to implement the first two features on the backend side during the first sprint, and the second two features in the second sprint."

Jane

"I have total 4 features and 12 requirements to implement. I have setup all the acceptance tests criteria for each requirement so that I can verify if the requirement is correctly implemented later."

Jane

"You can find all the weekly plans, features, requirements, acceptance tests, and other details on the Canvas team project page.[34]"

Feature 2: Web-based CRUD Operations - Requirements

Tim

"For Feature 2, we have four requirements corresponding to the CRUD operations:"

[34]Or individual contribution page.

RQ1: As a student, I want to create schedules via form/JSON so that I can add schedules online.

RQ2: As a student, I want to read schedules in JSON/HTML so that I can view schedules online.

RQ3: As a student, I want to update schedules via form/JSON so that I can modify schedules online.

RQ4: As a student, I want to delete schedules via form so that I can remove schedules online.

Tim

"Each requirement follows the Actor-Goal format: WHO (student), WHAT (the action), and WHY (the benefit). This makes them easy to test—if a student can successfully create a schedule via the form, we know RQ1 is met."

Then, Tim explains his implementation plan too in almost the same format.

Sprint Plan and Timeline

Now, Ken takes over to explain the sprint plan and timeline for Stage 2 & 3.

Tim

"Let me walk you through our sprint plan.

Everything is aloredy uploaded as the part of HW2, so he opens the Canvas team project page to show the sprint plan slide.

Sprint 1 (Weeks 4-6):
- *Feature 1: RQ1-1, RQ1-2 (Email setup, N-day specification)*
- *Feature 2: RQ1, RQ2 (Create and Read operations)*
- *Goal: Basic working prototype*

Sprint 2 (Weeks 6-8):
- *Feature 1: RQ2 (Checkmark confirmation)*
- *Feature 2: RQ3, RQ4 (Update and Delete operations)*
- *Goal: Complete CRUD with notification confirmation*

Tim

"By the end of Sprint 1, students will be able to create schedules and receive notification emails. By the end of Sprint 2, they'll have full CRUD capabilities and can mark tasks as acknowledged."

Closing Remarks

Ken

"Thank you all for your attention. We're excited to build this Task Management Application, 'Early Starter'. Do you have any questions or feedback for us?"

Some students raise their hands to ask questions about the team project, and Ken, Jane, and Tim answer them without any problems.

Prof. Cho

"Well done, team. I'm looking forward to seeing your implementation in the coming weeks."

Prof. Cho

"Remember: requirements are your contract with the users. Keep them visible, keep them testable, and keep them aligned with the problems you're solving. Good luck with Sprint 1!"

Prof. Cho introduces the next team for their PPP presentation until all teams finish their presentations for Stage 2.

After the Presentation

The team gathers after the presentation.

Tim

"That went well! I'm glad we spent time really understanding the requirements chapter before preparing this presentation."

Jane

"Yeah, Julie's question about N days was great. I'm happy we thought about making it configurable—that shows we learned from her mistake of making assumptions."

Ken

"The key takeaway for me is that this presentation forced us to think deeply about our requirements. We didn't just list features—we connected them back to real problems and forward to testable criteria."

Tim

"I also liked how we defined 'Done' clearly as a part of team rules."

Jane

"Exactly. Now we have a clear roadmap. Each requirement has acceptance criteria, each sprint has goals, and we know exactly what success looks like."

Looking Ahead to Sprint 1

Ken

"Okay, team. PPP is done. Now the real work begins. Let's start Sprint 1 on Monday."

Jane

"I'll draft the database schema based on our requirements for RQ12. We need tables for users, schedules, and notification preferences."

Tim

"I'll start designing the UI mockups for the Create and Read forms for RQ1 and RQ2. Once Jane has the API endpoints defined, I can start frontend integration."

Ken

"I'll research email service options and set up the development environment. Let's aim to have a good MVP as quickly as possoble."

Jane

"Perfect. Small steps, frequent integration, constant testing—just like we learned."

Tim

"I'm excited. We're not just coding—we're engineering a solution to a real problem."

Ken

"That's the difference between amateurs and professionals. Amateurs code. Professionals solve problems. Let's get to work!"

The team heads to their computers, ready to turn their well-defined requirements into reality.

2. Weekly Progress Reporting & Presentation

In the Stage 2, weekly progress reporting and presentations help the team stay aligned and keep stakeholders informed.

Weekly Progress Reporting & Presentation

After the PPP ceremony, team ‘Start Early’ meets to discuss the weekly presentations.[35]

Weekly Individual Contribution Format

> **Prof. Cho**
>
> “To make weekly progress reporting effective, each team member should follow a simple format when updating their individual contribution pages on Canvas.”

He writes the modified questions on the whiteboard:

> ***Weekly Standup Questions:***
>
> *Each team member answers:*
>
> 1. *Milestones/goals*
> 2. *Are you on track for milestones?*
> 3. *What did you accomplish this week?*
> 4. *What will you work on next week?*
> 5. *Any blockers or help needed? Recovery Plan?*

[35]This meeting can be in any format at any time; online biweekly meeting after or before the class can be one of the options that works well for many teams. Email based information sharing can be another option if synchronous meetings are hard to schedule, but this works effectvely when team members know each other well and have built trust.

6. Metrics: Burndown rate, LoC, tests added/removed.
7. Links to GitHub.

Jane

"So, basically, we report (1) the goal (1 and 2), (2) what we did, will do, and issues (3, 4, and 5), (3) metrics (6), and any links to GitHub (7)."

Prof. Cho

"Exactly. Use the item 5 to report any issues; but at the same time recovery plan and proposed solutions. If the plan needs to be updated, also the revised plan should be included. This shows proactiveness."

Jane

"How detail the item 7 should be?"

Prof. Cho

"It's up to the team rule and each team member. The key is to provide enough information for (1) team leaders check the progress and (2) make the decision what to share with stakeholders to understand the progress effectively."

Ken

“Yes, I sometimes need that information to make the weekly team progress presentation. So, I think providing links to the relevant GitHub pages (code, tests, requirements) is enough for most cases.”

Jane

“Makes sense. If you need more details, we can provide them for you.”

Tim

“Do we, team members, need to make the presentation each week?”

Prof. Cho

“No. Team members make individual updates on their Canvas contribution pages. Team leaders compile these into a team progress presentation for the weekly meeting.”

Prof. Cho

“Here’s an example format:”

templates/canvas_contribution.txt

*Use the **canvas_contribution.txt** template in the templates directory for your project.*

Example:

1. *Goal: Milestone 2 - Backend APIs for Task Management*
2. *This week: Completed Task model and API endpoints (Feature 3, RQ 4).*
3. *Next week: Build frontend task creation form (Feature 3, RQ 5).*
4. *Blockers: None. On track for Milestone 2.“*
5. *Right on track for Milestone 2.*
6. *Metrics:*
   - *Features burn down rate (3 finished out of 4, 75%)*
   - *Requirements burn down rate (2 finished out of 4, 40%)*
   - *LoC: +400 (Total 1500 LoC)*
   - *Tests: +8 unit tests (total 20 unit tests).*
7. *Links:*
   - *Requirements <...>*
   - *Code <...>*
   - *Tests <...>*

Prof. Cho

“Again, this is an example. Each team member customizes their updates based on their work. The key is to keep it concise and focused on progress and blockers.”

Jane

"Our team already has a rule to update weekly progress on our Canvas contribution pages. So, I think this will work well for us."

Prof. Cho

"Good. This is pretty much all the information stakeholders need to understand project status quickly."

Prof. Cho

"Remember, the goal is to keep everyone informed and aligned. Regular updates help catch issues early and keep the project on track."

Delegating Weekly Team Presentation

Team 'Start Early' discusses the weekly progress reporting process.

Tim

"So, team leaders should present team progress at the weekly progress report meeting[36]. It's your job to make the presentation, right?"

[36]The first classroom meeting for each week.

Ken

"Yes, that's correct. As the team leader, I compile individual updates from team members into a concise team progress presentation. I'll make the weekly presentation, but if I cannot join the class, I'll ask Julie or Tim to present it on my behalf.[37]"

Ken

"You must upload your **weekly progress** to your Canvas contribution page according to the team rules. You already have the weekly plan, so check if you finished it. If not, report why and provide a recovery plan with a revised schedule."

Jane

"Yeah, the focus is to keep making progress according to the plan. Even if I cannot make any progress, I should report it with a recovery plan."

Handling Non-Updates Professionally

[37]Weekly presentations are critical for keeping stakeholders informed. Under any circumstances. It is the team leader's responsibility to ensure the presentation happens, even if they cannot attend themselves. Missing this responsibilities will end up in (very) negative evaluations.

Tim

"What if a team member doesn't update their weekly progress on time?"

Ken

"In the real world, it's not a thinkable situation. But in this classroom, team leaders don't play the role of managers, but tech leads. So I simply report the issue to the manager—in this case, the professor.[38]"

Tim

"I understand many students are busy, but I believe team members should update weekly progress because it proves they can make planned progress."

Ken

"You'll be surprised when you experience all kinds of unexpectedness. Human factors in software engineering are one of the worst factors that add unnecessary complexity."

[38]Sending a simple email is enough. Team leaders do not waste their time chasing team members for updates. They focus on leading the technical aspects of the project. If a team member consistently fails to provide updates, the team leader escalates the issue to the professor for resolution.

Tim

"I still believe that under any circumstances, the weekly update is a must. Even if the team member cannot make progress, they should report it with revised plans. This is teamwork, isn't it?"

Ken

"I have no doubt that you guys will make progress report, and won't surprise me."

Professional Response to Problems

Julie joins the discussion.

Julie

"Haha, Tim, I like your attitude as a team member! But in the real world, there will be all kinds of chaos. That's why we should show **no emotion** in the workplace—showing emotion solves no problems."

Situation	Emotional Response	Professional Response
Teammate misses deadline	"Why didn't you tell me?!"	"What's the blocker? How can we adjust?"

Requirements change	"This is so unfair!"	"What's the new scope? How do we replan?"
Code review critique	"You're too critical"	"Thanks for the feedback, I'll fix it"
Technical debt	"Not my fault!"	"Let's document and plan to fix it"

Ken

"Yes. We should focus on **solving problems**, not adding unnecessary complexity by showing emotion.

Team Progress Presentations

Tim

"OK, so team members update weekly progress on their contribution pages, and team leaders use those for the team's weekly progress presentation."

Julie

"I need to add: in weekly meetings, team leaders should make discussions as short as possible and to the point."

Jane

"Haha, nobody wants to listen to whatever they're not interested in."

What to Present

Ken

"Yes. So basically, we share the following information."

Ken uses the whiteboard to stress the points:

templates/weekly_presentation.txt

*Use the **weekly_presentation.txt** template in the templates directory for your project.*

Weekly Progress Presentation:

1. *The Milestones (goals) for the week*
   - *The finished milestones, features, & requirements*
2. *The progress*
   - *Burndown rates*
     - *Features: finished/total*
     - *Requirements: finished/total*

- Tests counts
  - Acceptance tests
  - Integration tests
  - Unit tests
- Total LoC (lines of code)

3. If a plan change is needed
   - The reason
   - The impact & recovery plan
4. (Optional) Any issues to discuss
   - Ask for help

Jane

"Wait, it's pretty much the same, but not exactly the same as the individual updates. Right?"

Ken

"Yes, team leaders summarize individual updates into a concise team presentation. The focus is on overall progress and any changes needed."

Julie

"Remember that stakeholders may not need the technical details or individual contributions. Focus on the big picture."

Tim

"Makes sense. They already know the plans, both sprint and weekly. So, maybe what they need to know is that everything is OK or not; if it's not OK, they may want to know why and how to recover. Not the details, in most cases."

Julie

"Exactly. Even if nothing goes well, they should know that early with a recovery plan. No surprises!"

Ken

"Julie, do I need to report individual contributions in the weekly presentation?"

Julie

"Well, it depends, but in most case, my answer is 'No.' Stakeholders care about overall progress, not who did what.It's the team responsibilities, not the individual ones."

Tim

"Got it. But if a team member consistently fails to deliver, I think that should be reported separately. Right?"

Julie

"Yes, that's a different issue. Consistent underperformance should be escalated to the professor separately. But in weekly presentations, focus on team progress."

Jane

"Understood. So, in weekly presentations, we focus on team progress, milestones, burndown rates, tests, LoC, and any changes needed to share the progress with stakeholders, not finding faults with individuals."

Now, the team 'Start Early' understand the goal and format of weekly progress reporting and presentations.

Each team member updates their individual contribution pages on Canvas weekly. The team leader compiles these updates into a concise team progress presentation for the weekly meeting.

They also have a regular meeting after the class to discuss any issues and ensure everyone is aligned. Especially, if anything goes wrong, they discuss recovery plans proactively online or offline as needed.

In a way, they learn how to solve problems by managing complexity in a team to be the professional software engineers.

The Power of No Surprises:

By reporting the issue during weekly presentation, the team:

- Gave stakeholders an early notification
- Proposed a solution (move to Sprint 2)
- Maintained credibility
- Avoided last-minute panic

Lesson: Early communication of problems is professional. Hiding problems until the deadline is not.

Common Mistake #4: Vague Progress Reports:

❌ "We're working on the feature."

❌ "It's almost done."

❌ "We had some issues."

✅ "Feature 2 is 80% complete (3/4 requirements done)."

✅ "Estimate completion by Friday."

✅ "SMTP config issue took two extra hours, now resolved."

Why? Vague reports don't give stakeholders actionable information, and even can be interpreted as hiding something. Be specific and show your team has the situation under control.

3. S1R

Now, it's time to finalize the first sprint. S1R (Sprint 1 Retrospective) is when we reflect on our first sprint and discuss what went well, what could be improved, and how we can do better in the next sprint.

Sprint 1 Retrospective Ceremony

Next week, it is planned to have the Sprint 1 Presentation (S1P) ceremony. Team 'Start Early' holds a retrospective meeting to reflect on their first sprint.

This is an internal meeting for the development team only.

Ken presides over the meeting.

Ken

"Alright, team, it's time for our retrospective meeting for the first sprint."

Jane

"Great! Let's start by discussing what went well during the first sprint."

What Went Well

Tim

"OK. I'll go first. I think our team communication was excellent. We had regular meetings and kept each other updated on our progress."

Jane

"I agree with Tim. I also think our time management was good. We met our deadlines and completed our tasks on time. But I don't feel comfortable with the testing process."

They continue discussing what went well during the sprint.

What Could Be Improved

Tim

"Now, let's talk about what could be improved. I think we could have done better with our code reviews. Sometimes, we rushed through them and missed important issues."

Ken

"Yes, I know your concern. I think we can find the issues easier if we could improve our testing process."

Tim

"I agree. I also think we should consider schedule changes more carefully next time. We had to adjust our schedule a few times during the iteration, and it caused some confusion."

Ken

"Good point, Tim. I think it's OK to make changes, but we should have communicated earlier and announced the change to stakeholders as soon as we found the need for the change."

Jane

"Yes, when my feature was delayed, we should have decided to move one of Tim's features to the next iteration quickly to notify stakeholders earlier."

Retrospective Summary & Action Items

templates/retrospective.txt

*Use the **retrospective.txt** template in the templates directory for your project.*

The goal of the retrospective meeting is to identify actionable items for improvement in the next sprint, so the team can continuously improve their process.

Retrospective Meeting Rules:

- No blame or personal attacks
- Focus on process, not people
- Everyone participates

- Document all action items

They keep discussing what went well, what didn't, and what can be improved for the next iteration.

Ken

"Alright, team, I think we had a productive retrospective meeting. I'll summarize our discussion and upload it so that we can keep these points in mind for the next iteration."

Action Items Example:

From Sprint 1 Retrospective:

1. **Write tests first** (Jane) - Try TDD for next iteration
2. **Communicate schedule changes within 24 hours** (All) - Don't wait for the weekly meeting
3. **Standup at fixed time** (Ken) - 3 PM 20 minutes before class meetings, 15 minutes max

Why This Works: Specific, measurable, assigned, achievable.

Challenging Realities with 'Dogfooding'

Ken

"Before we end the meeting, I'd like to suggest that we can use our own software during the next sprint."

Jane

"That's a great idea! Using our own software will help us identify usability issues and bugs that we might not have noticed otherwise."

Tim

"Yes, I agree. We already have a working version of our software, so we can start using it right away. This will also give us a better understanding of how our users will interact with the software."

Ken

"OK. Then, let's do the 'dogfooding' during the next sprint. I'm sure this is a good opportunity to improve our software and our development process by being the real users.[39]"

Even though they will not discuss using their own software during the S1P, they decide to use it during the next sprint to improve their development process.

If everything goes well, they can demonstrate this during the Final Presentation later.

[39]It is highly recommended to use your own software during development. This practice is called 'eating your own dog food' or 'dogfooding.' It helps identify usability issues, missing features, and bugs that only appear with real usage.

4. HW4

Before their S1P (Sprint 1 Presentation) ceremony, the team prepares their presentations and finalizes their work. This includes the submission of HW4.

HW4: Sprint 1 Finalization & Sprint 2 Preparation

To be ready for the Sprint 1 Presentation (S1P) ceremony, Team 'Start Early' meets to disucss the HW4 and S1P preparation.

Finalizing Sprint 1

Ken

"Alright team, we finished S1R meeting, and it's time to finalize the first sprint. We need to prepare for the Sprint 1 Presentation (S1P) ceremony next week. Let's make sure we have everything ready."

Jane

"We already accomplished a lot during Sprint 1. The burndown rate will not be 100% due to the test issues, but we already notificied this issue to stakeholders during the weekly presentation. Other than that, we completed all our tasks for me and Tim. We have all the unit tests for modules, 20 integration tests, acceptance tests for all requirements, and documentation is up to date."

Ken

"Yes, I'm still working on the user manual, but it should be done by the end of today. I'm revising the REST API, but it's minor changes. We don't change the architecture, so I'll just reuse the architecture from the PPP.[40]"

Tim

"Do you need my design documents for the presentation?"

Ken

"Well, I don't think stakeholders will be interested in design documents. They care more about features and usability. But it's good to have them ready in case they ask for it. Make sure your design documents are up to date and organized in the project repository, and easily accessible from the Canvas team page or individual contribution pages.[41]"

[40] The most important documentation is the user manual, architecture/design documents, and REST API documentation. These documents help users understand how to use the software, how to interact with the API, and how the application is designed and working.

[41] Any project related information should be organized and easily accessible for team members and stakeholders. This includes design documents, user manuals, API documentation, test cases, and any other relevant materials.

Jane

"Got it. We already did PPP when we started the sprint 1, so it's much easier this time, all we have to do is just update it with the new information from this sprint."

Tim

"Yeah, HW4 is mostly about finalizing what we have done in Sprint 1 and preparing for the presentation. We should focus on presenting the features that we promised to deliver and promise new features for the next sprint to stakeholders."

Demo Preparation and Presentation Slides

Ken

"True, but we should show some demos during the presentation this time. Let's prepare a demo script to showcase the key features we implemented in this sprint."

Jane

"Do we need to do a live demo, or can we use pre-recorded videos?"

Ken

"It really doesn't matter, but I think pre-recorded videos are safer[42]. Live demos can be risky if something goes wrong. We can record the demo videos in advance and play them during the presentation."

Tim

"Good idea. I have my own YouTube channel, so I will upload the video clips there. It can be a good proof that I could solve software engineering problems in a team successfully."

Jane

"That's great! Tim, can you help me with recording the demo videos? I'm not very experienced with video editing."

[42]It's OK to do a live demo, but it is recommended to prepare pre-recorded videos as a backup even in this case.

Tim

"Sure, fortunately, it's not that hard to edit videos nowadays. I can help you with that. We can use free video editing software any video editing software you are comfortable with.[43] I think it doesn't have to be professional for now, in most cases, screen recording is enough. I can also help you with uploading the videos to YouTube and sharing the links with the team."

Ken

"Great! Make sure you have the links ready for the presentation. Also, let's prepare some slides to summarize our work, challenges we faced, and how we overcame them."

The team works together to finalize their documentation, prepare demo videos, and create slides for the S1P (Sprint 1 Presentation). They ensure everything is organized and easily accessible for stakeholders.

Preparing for Sprint 2

Ken

"Now that we have finalized Sprint 1, let's start preparing for Sprint 2. We need to plan our tasks and set clear goals for the next sprint."

[43] like DaVinci Resolve or HitFilm Express.

Jane

"But when we start the project, we already did the Sprint 2 planning in advance. We have a good idea of what we want to achieve in Sprint 2."

Ken

"True, but we had to adjust our plans based on what we learned in Sprint 1. Let's review our backlog and prioritize tasks for Sprint 2. We should also consider any feedback we received from stakeholders during the S1R meeting."

Tim

"Yeah, I learned from the class that we should be flexible and adapt our plans as needed. Let's make sure we have a clear understanding of the requirements and expectations for Sprint 2."

The team reviews their backlog, prioritizes tasks for Sprint 2, and discusses any adjustments needed based on their experiences from Sprint 1.

Then, they update their plans on GitHub and Canvas pages to reflect their goals for Sprint 2. They are easy to access for all team members and stakeholders.

"Submission of HW4"

Jane

“Then, what we should do for HW4?”

Ken

“HW4 is mostly about finalizing our work for Sprint 1 and preparing for the Sprint 1 Presentation. We need to ensure all our documentation is up to date, including the user manual, architecture/design documents, and REST API documentation. We also need to prepare our demo videos and slides for the presentation.”

Tim

“Lucky us, we already finished most of the work. We just need to polish things up and get ready for the presentation.”

Ken

“Yes, just like HW2, I’ll make HW4 today, and before submitting it, I will ask both of you to review it. We need to ensure everything is accurate and well-organized before submission.”

Team

"Sounds good. Just let me when it's ready for review. But I trust you, Ken. You did a great job on HW2, so I have no doubt about HW4."

Ken finishes preparing HW4, and the team reviews it online to ensure everything is accurate and well-organized before submission.

Finally, Ken submits HW4 on behalf of the team, ensuring all required materials are included and accessible for stakeholders.

They are all set for the Sprint 1 Presentation ceremony next week.

5. S1P

S1P (Sprint 1 Presentation) is the ceremony where the team presents their completed work from the first sprint to stakeholders. In this ceremony, the team delivers the high-quality working software that they promised during Sprint 1 planning.

There will be 'no surprises' for stakeholders, as the team has already communicated their progress and any changes during the sprint. The presentation will be more like a celebration, not checking for arguments.

Sprint 1 Presentation Ceremony

The first sprint is over, and it's time for the review ceremony through presentation. Team 'Start Early' gathers for their first sprint presentation ceremony.

All team leaders prepare Marp slides for the review ceremony before class, convert them to PDFs, and upload them to the GitHub repository. Then, they share the links on the Canvas team project page.

Prof. Cho opens the team's 'Start Early' project page on Canvas and shares the screen.

> **Prof. Cho**
>
> "Alright. This is the sprint 1 presentation ceremony. Team 'Start Early,' please go ahead with your presentation."

> **Prof. Cho**
>
> "All the artifacts and schedule information about the team project can be found on the Canvas team project page so that stakeholders can find all the information from one place."

Prof. Cho

"So, please focus on presenting what you promised and have accomplished during the first sprint, and demonstrate the working software. Also, please explain any challenges you faced during the sprint, and how you overcame them. Finally, share your plans for the next sprint based on your retrospective meeting."

Presentation Structure Guideline

This is the suggested structure for the sprint review presentation. Each team has about 15 minutes for their presentation, and five minutes for Q&A.

All the links should be easily accessible from the Canvas team project page, so teams should focus on presenting what stakeholders care about: working software (features implemented), challenges, and learning.

Review Ceremony Presentation (15 minutes):

1. Iteration Summary (4 min)

- Planned features vs. completed
- Metrics
  - Burndown rate (features & requirements)
  - Test metrics

2. Demo of team members' features (4 min)

- Show working software (live or video)

- Highlight key features
- Explain user flow

3. Challenges & Solutions (4 min)

- What was difficult?
- How did you solve it?
- What did you learn?

4. Next Iteration Plan (3 min)

- What's changing based on the retrospective?
- What features are planned for Sprint 2?

Presentation

Ken starts the presentation.

> **Ken**
>
> "Hello, everyone. We are team 'Start Early.' I'm Ken, the team leader. This is Jane and Tim, my team members. Today, we will present our review for the first sprint."

> **Ken**
>
> "During the first sprint, we focused on developing the core features of our application, including user registration, course schedule input, and reminder notifications."

Ken shows the features pages on the S1P slides.

Ken

"For the first sprint, we successfully implemented 3 out of the four planned features, and completed out of the 15 requirements. Burndown rate for features is 80%, and for requirements, it's also 80%."

Ken

"In the previous meeting, we already discussed the reason why the completion rate is 80%, as one of the features and 3 of the requirements should be implemented in the 2nd iteration due to constraints."

Ken

"We also created the acceptance tests for all requirements to ensure that our application meets the requirements and functions as expected."

Ken

"Also, we have written unit tests for all modules, and created 20 integration tests to verify the interactions between different components of our application.[44]"

[44]This is a guidline, teams may have different test strategies based on their project nature and requirements.

Ken

"The test coverage for our codebase is currently at 85%, which we believe is a good starting point for our application. We will continue to improve the test coverage in the next sprint.[45]"

Tim shows the test pages on the S1P slides.

The Demo

Jane takes over the presentation and demonstrates the working software through a recorded video demo.

Jane

"Thank you, Ken. Now, let me demonstrate the working software we developed during the first iteration. You can find the video demo on our GitHub repository, and I will play it for you now."

She plays the recorded demo video that shows the application's main features.

[45]High test coverage is important to ensure the quality and reliability of the software. It helps to identify bugs and issues early in the development process, and ensures that the software meets the requirements and functions as expected, but in general 80% coverage is enough because accomplishing 100% coverage is too expensive.

Demo Best Practices:

Do:

- Record a backup video (live demos can fail)
- Show realistic use cases (not just "hello world")
- Highlight interesting technical features
- Keep it under 4 minutes
- Give proper explanations of what you're showing

Don't:

- Apologize for incomplete features
- Show every single button
- Get lost in technical details
- Spend time on setup/config

Pro Tip: Practice your demo at least twice before presenting.

Then, Tim takes over the presentation, and demonstrates the email notification feature that he planned and implemented during the first sprint.

Challenges & Next Steps

> **Ken**
>
> "Now, let me discuss the challenges we faced during the first sprint, and our plans for the next sprint."

Ken

"One of the main challenges we faced was integrating the email notification feature with the rest of the application. We encountered some technical issues when we tested the feature, and it delayed the implementation of this feature."

Jane

"We reported this issue to stakeholders during our weekly presentation, and we worked together as a team to find a solution."

Tim

"Even though, we had to postpone the email notification feature to the next sprint, we learned a lot from this challenge. We improved our understanding of the email protocols and libraries, and we are confident that we can successfully implement this feature in the next sprint.[46]"

Tim explains how they overcame the challenges.

[46]It's OK to face challenges and failures during the sprint. The important thing is to learn from them and improve. Stakeholders appreciate honesty and learning.

Ken

"Based on our retrospective meeting, we plan to improve our communication and collaboration during the next iteration. We will also focus on implementing the remaining features and requirements in the next iteration."

Ken

"Thank you for your attention. Do you have any questions?"

Some students raise their hands to ask questions about the team project, and Ken, Jane, and Tim answer them without any problems.

They finished the presentation successfully.

Prof. Cho

"Thank you, team 'Start Early,' for your presentation. You did a great job explaining your accomplishments during the first iteration and demonstrating your working software. I appreciate your effort in preparing for this presentation. Now, let's move on to the next team."

Common Mistake #1: Focusing Only on Success:

❌ "Everything went great! No problems!"

❌ Hide challenges or failures

✅ “We faced X challenge. Here’s how we solved it.”

✅ “We learned Y lesson that will help in Sprint 2.”

Why? Stakeholders respect honesty and learning. Pretending everything is perfect lacks credibility.

Ready for the 2nd sprint

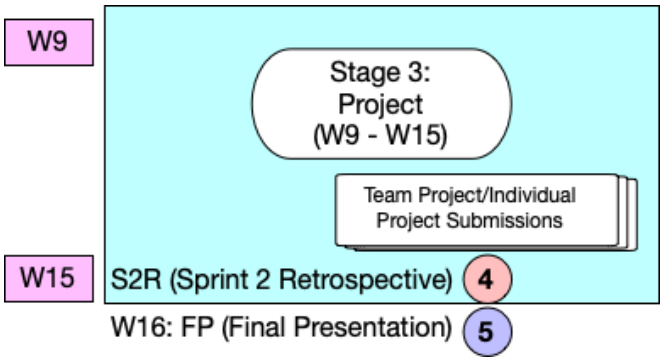
The team feels accomplished after the successful Sprint 1 Presentation (S1P) ceremony. They are motivated to start the second sprint and continue their progress.

Section

Stage 3: Project

Goal: FP (Final Presentation) @ W16

Weeks: W9 - W16



1. PPP (Project Plan Presentation)
2. S1R (Sprint 1 Retrospective)
3. S1P (Sprint 1 Presentation)

4. S2R (Sprint 2 Retrospective)

5. FP (Final Presentation) <- S3 Goal

1. Project

*After completing the first sprint, the team now there is no longer any **unknown unknowns** of the project. They also understand the technical challenges and risks involved, and how to address them effectively.*

With this knowledge, the team is now ready to execute the rest of the project successfully. This time, they can focus mainly on delivering high-quality working software that meets stakeholders' needs and expectations.

*They are finally in the stage of **Project**.*

Project Stage Overview

Ken

"Alright team, now that we've completed our first sprint and have a better understanding of the project, it's time to move into the Project stage."

Jane

"Yes, I feel much more confident now that we know what to expect. We can focus on delivering high-quality working software that meets our stakeholders' needs."

Tim

"Exactly. When I first started the team project, I didn't know what to do, and how to do it. But now, I have a clear understanding of the technical challenges and risks involved, and how to address them effectively.[47]"

Team 'Start Early' is now in the **Project** stage, where they can focus on delivering high-quality working software that meets stakeholders' needs and expectations.

[47] This is a common experience for many team members when starting a new project. The initial uncertainty can be daunting, but as the team progresses through the first sprint, they gain clarity and confidence in their ability to deliver. But still, there are many challenges ahead, and we need to stay focused and work together to overcome them.

This is possible because they have all the plans, designs, and strategies in place from the previous stages, and they have a clear understanding of the project requirements and goals.

> **Ken**
>
> "Welcome to real-world software engineering! I know you now understand the meaning of managing complexity[48]. Also, you now know how the software engineering tools and rules can help us solve problems effectively.[49] Let's keep up the good work and deliver a successful project!"

[48] They could sort out all unknown unknowns to minimize complexity.

[49] They could use various software engineering tools and practices such as version control, continuous integration, automated testing, code reviews, and agile methodologies to manage the project efficiently.

3. S2R

Now, it's time to finalize the project. S2R (Sprint 2 Retrospective) is when we reflect on our second sprint, and ultimately the whole project.

Sprint 2 Retrospective Ceremony

It is time to finalize the course and team project. It is planned to have the Final Presentation (FP) ceremony next week. Team 'Start Early' holds the 2nd retrospective meeting to reflect on their second sprint and the overall project.

Like the S1R, this is an internal meeting for the development team only.

Ken

"Team, it's time to finalize the project.

Jane

"Wow! Time flies so fast. It feels like we just started the project yesterday, but now we are at the end of the course."

Tim

"Yes, I agree. Let's start by discussing what went well during the second sprint and the overall project just like we did before."

S2R

Ken

“Alright team, let’s begin our Sprint 2 Retrospective. What do you think went well during this sprint?”

Jane

“I think our communication improved a lot. We were much better at keeping everyone updated on our progress through daily standups.”

Tim

“Agreed! And our testing process was much more systematic this time. We caught several bugs early before they became major issues.”

Tim

“Great points! Now, what about areas where we could improve?”

Ken

“Honestly, I think our time management could still be better. We had a bit of a rush at the end to finish everything.”

Jane

"True. And maybe we could have done more code reviews. Some parts of the codebase could have been cleaner if we'd reviewed each other's work more carefully."

Tim

"I think we still underestimated the complexity of some features. Better estimation would help us plan more realistically."

Ken

"Those are all valuable insights. How can we apply these lessons to future projects?"

Jane

"We should definitely maintain the communication habits we've built. Online discussion is OK, but regular face-to-face meetings really help."

Tim

"And we should build in buffer time for unexpected issues. Things always take longer than we think!"

Ken

"Plus, making code reviews a mandatory part of our process would help maintain code quality."

Tim

"Well, but realistically, everyone is busy with other course work, so I really don't think we can do the code reviews seriously just like real-world software engineers do."

Jane

"I agree with Tim. But I think the real problem is to maintain high quality work even under tight deadlines. We should focus on prioritizing important tasks and cutting down unnecessary work to meet deadlines without sacrificing quality."

Ken

"Excellent reflections, everyone. Now, this is also our last chance to make any final adjustments before the Final Presentation next week. Any concerns or items we need to address?"

Jane

"I feel more confident this time. Our demo scripts are ready, and our slides are polished."

Ken

"Perfect. Let's schedule a rehearsal for Thursday and do our final checks. This has been a great learning experience for all of us. I'm proud of what we've accomplished together!"

Jane

"Me too! It's been challenging but really rewarding. I feel like I've grown a lot as a problem solver, I mean software engineer."

Tim

"Same here. Can't wait to show everything we've built at the Final Presentation!"

Reflecting on 'Eating Your Own Dog Food'

They decided to use their software 'Start Early' to manage their tasks and collaboration during the project.

Tim

"I also want to mention how much I've learned about the importance of using our own software during development."

Tim

"Absolutely! Using 'Start Early' not only helped us manage our tasks but also gave us firsthand experience with the software we developed."

Ken

"That's a great point, Tim. Using our own software provided valuable insights into its usability and effectiveness."

The Power of 'Eating Your Own Dog Food':

Using your own software (called dogfooding) reveals:

- Usability issues you didn't anticipate
- Missing features that would be helpful
- Bugs that only appear with real usage
- Whether your solution actually solves the problem

Best practice: Always use your own software during development.

Jane

"I'll add that using our own software made us more empathetic towards our users. We experienced the same frustrations and challenges they might face."

Ken

"Alright, team, I think we had another productive retrospective meeting. I'll summarize our discussion and upload it so that we can keep these points in mind for future projects."

The retrospective discussion helps the team learn from their experiences and prepare for future projects. It's also their last opportunity to make final adjustments before celebrating their achievements at the Final Presentation ceremony.

3. Evaluation

Before the final presentation, the last step to finalize the project is evaluation.

When we start the project, we set goals, rules, and plans. In software engineering, just finishing work is not that important; what is truly important is to finish the work as planned without surprises. Also, it's important to deliver the work in high quality - the software should be maintainable and updatable effectively and easily.

Evaluation is the process of checking whether we have met those goals and plans by following the rules, and whether the quality of our work is satisfactory.

Project Evaluations

Ken

"Now, it's time to finalize our project before the final presentation."

Jane

"Yes, we need to submit team project results on GitHub and Canvas."

Tim

"Ken, just like HW2 and HW4, it's your responsibility to submit the team project results on time. Right?"

Ken

"Yes. But you also need to complete your peer and team leader evaluations to finalize the project."

Jane

"Also, you need to evaluate us, team members, as the team leader."

Ken

"Yes. As a team leader, I upload (1) the project results and (2) my evaluation of team mebers."

Ken

"You guys also need to submit your peer evaluations of each other and me as the team leader."

Tim

"Please explain more about peer evaluations and other evaluations."

Software Engineers as Professional Evaluators

Julie joins the discussion.

Julie

"As software engineers, we should be the professionals who can evaluate ourselves and our teammates fairly and honestly. We should not be biased or lenient just because we are friends or teammates. In the real world, our performance and contributions will be evaluated by our managers, peers, clients, and users, so we should practice being professional evaluators now."

Julie

"This is an example that shows the difference between poor evaluations and professional evaluations."

Evaluation Aspect	Poor Evaluation	Professional Evaluation
Communication	"Good"	"Responded to messages within 2 hours, attended all meetings"
Code Quality	"OK"	"Wrote clean code, 85% test coverage, thorough code reviews"
Reliability	"Fine"	"Delivered all features on time, reported issues proactively"
Teamwork	"Helpful"	"Helped Tim debug SMTP issue, pair programmed on complex features"

Jane

"Wow, the professional evaluation is so specific and detailed! It really shows what the person did and how well they did it."

Julie

"Yes, that's the point. Professional evaluations should be based on observable behaviors and measurable outcomes, not vague impressions or personal feelings."

Tim

"It's really serious work. I need to think carefully about how to evaluate my teammates and the team leader."

Evaluations based on Documents and Artifacts

Ken

"Also, we should base our evaluations on actual documents and artifacts, not just memories or hearsay."

Jane

"I see, that's why we needed to keep our progress updated and share artifacts and schedule with everybody."

Ken

"Exactly. In the real world, if the software engineers make excuses like 'I was busy' or 'I forgot to update the status,' they will be considered unprofessional and unreliable."

Tim

"I understand. Professional software engineers always make plans, and follow them, and document their work properly."

Jane

"No surprises."

Ken smiles.

Ken

"Yes. We all know how dangerous and breaks the team work, when a team member didn't do anything, but only rushes at the last minute."

Jane

"Right. Fortunately, nobody in our team did that. We all followed the plans and updated our progress regularly. But, I saw multiple times that other teams had such problems."

Tim

"So, basically, our evaluations should be based on actual work and contributions, not just feelings or memories."

Jane

"Yes. No personal feelings. Just facts and evidence."

Tim

"I feel much better now. I will be more objective and professional in my evaluations."

Three Evaluations by Team Members

Julie

"Each team member should complete three evaluations for the team project."

Julie

"First, each team member evaluates themselves honestly. The focus is (1) make a good plan and follow the plan, and (2) deliver high-quality work that can be effectively maintaiable and updatable."

Ken

"If you think you did a good job, give yourself a high score. If you think you could have done better, give yourself a lower score and explain why. Be honest and objective."

Jane

"I know it's hard to evaluate yourself, but it's important to be truthful. This is a learning opportunity."

Tim

"What if some other team members are not honest with their self-evaluations? In that case, I will get lower scores because of being honest."

Julie

"That's a good point, Tim. But you should know that software engineers are not dumb, on the contrary, they are very smart. They can easily find out who is honest and who is not."

Ken

"Yes. Everybody already knows who did a good job and who didn't. So, if you are honest, you will be respected and trusted by your teammates."

Julie

"Also, remember that other team members and leaders will evaluate you as well. So, being honest and professional is the best strategy."

Tim

"I see. It should be really awkward and even embarrassing if someone gives themselves a perfect score, but other team members and team leaders give them low scores.[50]"

Julie

"Exactly, but if you are honest and professional, you will get fair evaluations from others as well. And that will be the best reward."

Ken

"The second evaluation is peer evaluations. Each team member evaluates other team members honestly and professionally. The focus is on actual contributions and behaviors, not personal feelings or friendships."

Julie

"The third evaluation is the team leader evaluation. Each team member evaluates the team leader honestly and professionally. The focus is on how well the team leader managed the project, communicated with the team, and ensured high-quality deliverables."

[50]In real-world software engineering, inflated self-evaluations are often seen as a red flag. Managers and peers can usually tell when someone is not being truthful about their contributions. This can lead to loss of trust and credibility. ASE courses will reflect this reality to help students develop professional evaluation skills.

Two Evaluations by Team leader

Ken

"As the team leader, I also need to complete two evaluations."

Julie

"Yes. Just like the team membesr, team leaders also need to evaluate themselves."

Julie

"Then, team leaders evaluate each team member."

Jane

"I believe the same rule applies here as well. The evaluations should be honest and professional, based on actual contributions and behaviors."

Julie

"Exactly. That is why keep track of all work, progress, and artifacts is so important. It provides the evidence needed for fair evaluations.[51]"

[51]In real-world software engineering, team leaders and managers rely heavily on documented evidence to evaluate team members. This includes code commits, pull

Common Mistake #6: Lenient Peer Evaluations:

❌ Give everyone perfect scores "to be nice."

❌ Avoid mentioning any weaknesses

✅ Provide specific, honest feedback (positive and negative)

✅ Focus on behaviors, not personality

Why? Inflated evaluations hurt learning. Honest feedback helps people grow.

Project Grading Criteria

Tim

"How is the team project actually graded?"

Ken

"The evaluation focuses on three areas:"

Category	What It Measures	Weight
Product Quality	Effectiveness of solution, code quality, architecture, and test metrics	40%
Process Execution	Following Agile/Scrum, meeting milestones, documentation	30%

requests, issue tracking, meeting notes, and other artifacts that demonstrate contributions and behaviors.

Team Dynamics	Collaboration, communication, individual contribution	30%

1. Product Quality Metrics

Jane

"What does 'product quality' actually mean? Is it about features?"

Julie

"No. Quality is about **maintainability and extensibility**. Can you:"

- Fix bugs quickly?
- Add features easily?
- Onboard new developers efficiently?

Ken

"But in a way, features matter too. Users care about features."

Julie

"Sure. There is no doubt about it, but what I wanted to say was that quality is not about **quantity** of features or flashy UI. It's about **how well** the software is built under the hood."

Tim

"I understand. If the software is well designed and coded, adding new features or fixing bugs later will be much easier."

Jane

"I see. Also, if the software is just made in an ad-hoc way with beautiful UI but messy code, it will be a nightmare to maintain and update it later."

Julie

"Exactly. In real-world software engineering, maintainability and extensibility are key indicators of quality."

Ken

"That is why the tests metric is so important. As it shows the maintainability of the code through test counts and test coverage."

Julie

"Right. I must add that it is true that a software product can be in low quality even with many tests and high coverage, but it is impossible that a software product in high quality with low test metrics."

2. *Process Execution Metrics*

Julie

"Also, we have another metrics, process execution metrics."

Tim

"What is that?"

Julie

"Process execution measures **how well** the team followed Agile/Scrum practices and project plans. Key aspects include:"

Ken

"Process quality is measured by artifacts and adherence:"

Artifact	What It Shows
GitHub Commits	Regular progress, meaningful messages
Pull Requests	Code review culture, quality gates
Issue Tracking	Work organization, progress transparency
Sprint Boards	Agile process adherence
Meeting Notes	Team communication, decision tracking
Canvas Updates	Stakeholder communication

Jane

"So it's not just about the final product—it's about **how** we built it."

Ken

"Right. Because in real jobs, the process matters as much as the product."

3. Team Dynamics Evaluation

Julie

"Team dynamics get evaluated too. Here's what matters:"

Aspect	Good Practice	Red Flag
Communication	Daily updates, proactive	Silent until deadline
Contribution	Balanced commits, reviews	One person does everything
Conflict	Addressed constructively	Ignored or escalated badly
Accountability	Owns mistakes, proposes fixes	Blames others, makes excuses

Tim

"I see, but I'm not sure how the professor can evaluate these aspects fairly."

Julie

“Good question. The evaluations are based on multiple sources”

Julie

“Firstly, we have weekly presentations and retrospectives where team dynamics are visible. Professors can observe communication, collaboration, and problem-solving in action over one semester. So, they know pretty well how the team worked together.”

Julie

“Secondly, the peer evaluations provide direct insights from team members about each other’s contributions and behaviors. This helps identify any discrepancies or issues that may not be apparent from outside observation.”

Ken

“Lastly, the project artifacts like commit histories, issue tracking, and meeting notes also provide evidence of individual contributions and team collaboration. Professors can analyze these artifacts to assess how well the team worked together throughout the project.”

Jane

"That makes sense. So, the evaluation is comprehensive, considering multiple perspectives and evidence."

Julie

"Exactly. That is how professional evaluations work in real-world software engineering."

Ken

"Also, don't forget that if you think anything is unfair, you can always discuss it with the professor. He will listen to you and consider your feedback seriously."

Jane

"OK. As long as we are honest and professional, I think we will be fine."

Ken

"Yes. Even if we may not get great results out of the project, we can learn a lot from this experience. That is the most important thing."

Tim

“I understand the meaning of problem solving from different perspectives now. We solve our problems not only by coding, but also by communication, collaboration, and evaluation.”

Julie

“And getting better and better to be the professional software engineers step by step.”

Common Mistake #1: The 'Hero' Anti-Pattern:

❌ One developer does 90% of the work (looks good for them!)

✅ Work is distributed, everyone contributes meaningfully

Why? The point is learning. If one person does everything, the team fails even if the product succeeds.

4. Final Presentation

It's Week 16, the time for our final presentations.

The Final Presentation (FP) is when we showcase our completed team project to stakeholders, demonstrating the working software and highlighting our journey throughout the course.

The format is almost the same as the Sprint 1 Presentation (S1P), but this time, we present the entire semester's work, covering both sprints and all major features implemented.

Week 16: Final Project Preparation

At the end of the course, each team holds a final presentation to showcase their completed team project.

All the project related artifacts, including the final presentation slides in PDF format, are already uploaded to the GitHub repository and linked on the Canvas team project page before the class.

> **Prof. Cho**
>
> "Now that the course is coming to an end, it's time for your final presentations. Each team will present their completed team project and demonstrate the working software."

The team 'Start Early' gathers for their final presentation.

> **Ken**
>
> "Hello, everyone. We are team 'Start Early.' I'm Ken, the team leader. This is Jane and Tim, my team members. Today, we will present our final project."

Ken opens the PDF file of the Marp slides for the final presentation from the Canvas team project page.

The presentation structure is similar to the Sprint 1 Presentation (S1P), but this time, they will cover the entire semester's work, including both sprints.

Ken

"You may remember our Sprint 1 Presentation (S1P) from earlier in the course. We couldn't complete all the features we planned in Sprint 1, but we learned a lot from that experience. This time, we are excited to showcase the final version of our Task Management Application, which includes all the features we promised and more."

Jane

"Yes, we have accomplished 100% burndown rate. Make all the acceptance tests and passed them successfully. We have high-quality documents, including the user manual, API documentation, and design documents. We also improved our testing process and have comprehensive test coverage over 80%."

Tim

"And even we could use our own software to manage our assignment deadlines effectively throughout the course. It really helped us stay organized and on track with our work."

Students and stakeholders watch the final presentation of team 'Start Early,' learning about their journey, challenges, and achievements throughout the course.

The team demonstrates the working software, highlighting key features and improvements made since the Sprint 1 Presentation (S1P).

They could demonstrate the software using pre-recorded demo videos uploaded to YouTube, ensuring a smooth presentation without technical glitches.

Ken

"Thank you for your attention. We are proud of what we have accomplished as a team, and we hope our Task Management Application will be useful for students in managing their assignments effectively. We appreciate all the support and feedback from our stakeholders throughout the course."

Prof. Cho

"Great job, team 'Start Early.' Your final presentation was impressive, and your Task Management Application looks like a valuable tool for students. Thank you for your hard work and dedication throughout the course."