Software engineering education and training

PANAGIOTIS MELAS FRAGKISKOS

PANAGIOTIS TSELENTIS

PROFESSIONALISM IN SOFTWARE ENGINEERING

Professionalism in software engineering involves values, behaviors, and practices like **communication, ethics, and adherence to industry standards**, extending beyond writing functional code.

Following professional standards improves code quality, maintainability, and fosters trust and collaboration, enabling productive relationships.

Software engineering curricula often have project-based courses in which students are taught professionalism, but these courses, tend to be reserved for more advanced courses at the end of the degree.

PROFESSIONALISM IN SOFTWARE ENGINEERING

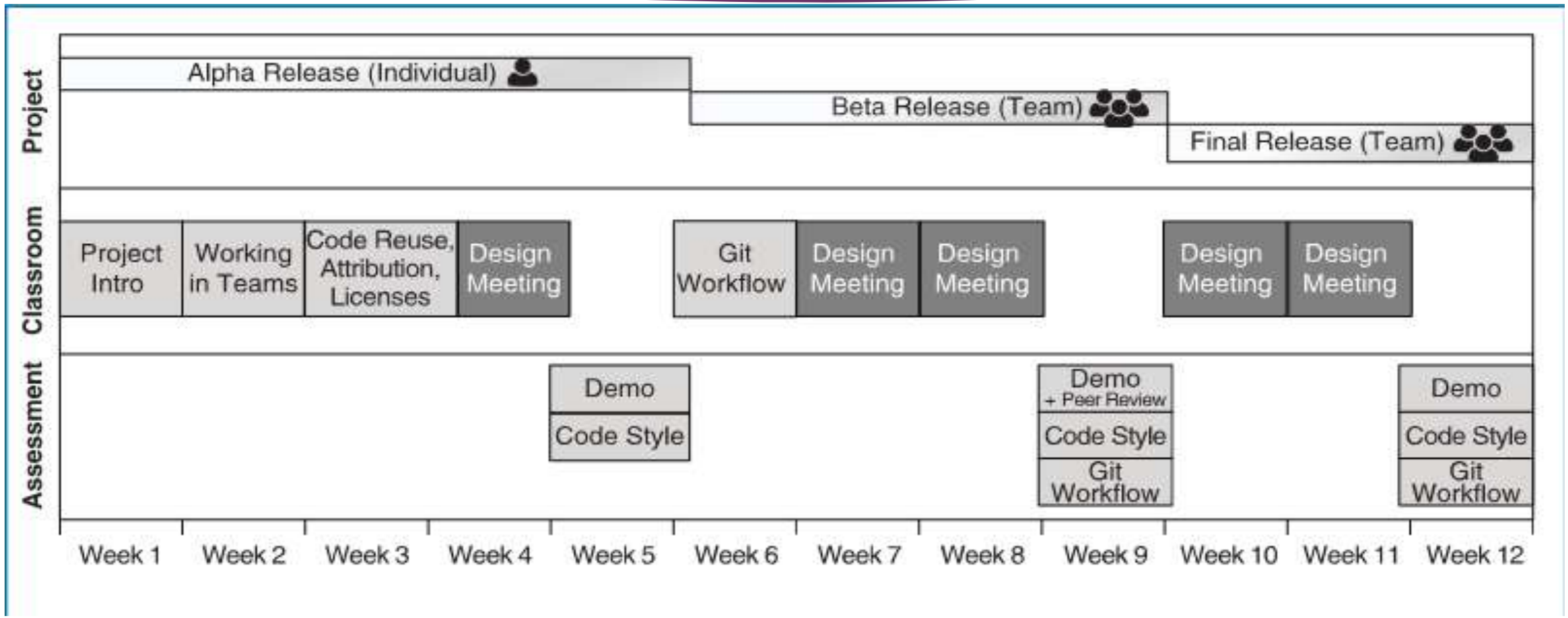
- But Terragni et al (2023) suggest that software engineering students should face professionalism early on in their careers, having a second-year project-based course where students work in teams to implement a project from scratch.

- The **benefits** of this early-exposure approach are:

1. The development of professionalism in software engineering requires repeated exposure and practice over time. So, having additional courses covering professionalism will be useful to better face the final-year projects.

2. With this early exposure, students will be able to better contextualize, understand and appreciate what they are being taught.

TIMELINE OF THE COURSE



DETAILS ABOUT THE COURSE (I)

- Like a real-world project, there is no exam or test. There is only one semester-long “project”, broken down into three deliverables (Alpha, Beta, Final) with incremental requirements.
- The **Alpha** deliverable is an individual assignment aimed to help students develop the confidence to develop in the relevant technologies before forming teams. For the **Beta** and **Final** releases, students are randomly allocated to groups of three students (the group formations are the same for both releases).
- The teams play the role of “professional software engineers” working for a software company and the course instructor plays the role of a “client” belonging to a hypothetical organization that has contracted this software company to build a custom application.

DETAILS ABOUT THE COURSE (II)

Working in Teams: students are taught how to ask questions to clients

Code Reuse, Attribution, Licenses: students learn the ethics of reusing other people's code and understand software licenses.

Git Workflow: students learn the best practices and workflows for Git, in particular *"GitHub Workflow"*

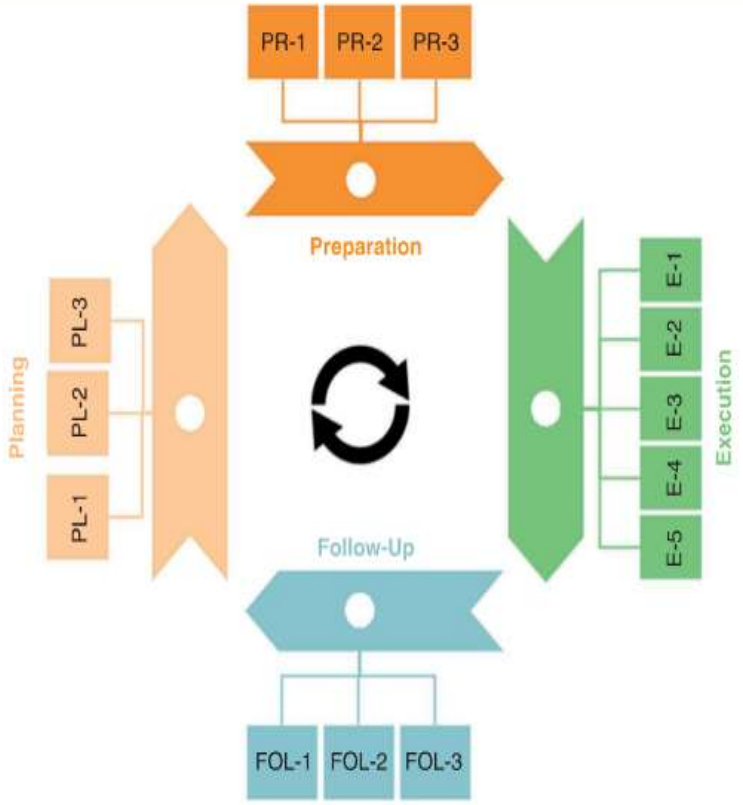
Design Meeting: This is the only opportunity engineers have to interact and seek clarifications from the client, as clients are not available outside of these meetings.

Assessment

- Demonstrations are the primary method used to assess students. For each of the three deliverables, students present their software product to the instructors. Some of the marks relate to professionalism such as the quality of the demonstration in terms of engagement, formality and politeness.

- Code style and Git workflows are evaluated using tools that automatically analyze source code and Git logs to report any violations.

Academia-industry collaborations



- The importance of academia-industry links is well understood. Several academic researchers have studied the phenomenon, with the aim to "close the loop" between what is taught in universities and what is requested by industry.

- The phases to boot and reboot academia-industry collaboration are divided in four major phases:

1. **Planning**
2. **Preparation**
3. **Execution**
4. **Follow-up**

PHASE 1: PLANNING

1. **Isolate the Learning Objectives (LO) that will be Part of the Industry-Academia Collaboration:** The course coordinator chooses which LOs will be developed as academic output, and which ones will be developed as industrial deliverables

2. **Interact With the Local Business Liaison Officers (BLO) and the Student Associations**: In this step, the course coordinator establishes a direct connection with the internal staff and student associations that facilitate the collaboration with industries.

3. **Create a Professional Organizational Structure Behind the Scheme**

PHASE 2: PREPARATION

1. **Create a Welcome Pack to Inform Industrial Partners and BLOs:** The course coordinator prepares a document detailing the purpose of the collaboration, the benefits of the students, and the industrial collaborators.

2. **Create a Proposal Form to Fill by Industrial Partners:** The course coordinator creates a form to collect software-based need and circulates it to industrial collaborators.

3. **Set Up a Web Portal to Make the Proposal Form, and Other Information, Available to Potential Collaborators**

Phase 3: execution

1. **Vet the Proposals Before the Start of the Course**

2. **Publish the Vetted Proposals for Students to Bid for, and Team Formation:** Ahead of the course, the course coordinator publishes the vetted proposals for students to evaluate.

3. **Introduce and Assess Soft Skills:** The course coordinator gives one lecture on soft skills, group work and professionalism.

4. **Manage Support and Monitoring of Groups:** This activity is delegated to teaching assistants. Two weekly meetings are organized: one between teaching assistant and the group and one between the industrial collaborator and the group.

5. **Assess Students on the Stated Learning Outcomes:** The course coordinator evaluates the groups according to University policies in a presentation day in which groups present their process and product.

PHASE 4: FOLLOW-UP

1. **Gather Feedback From Companies:** The course coordinator sets up questionnaire to evaluate how the companies perceived the scheme.

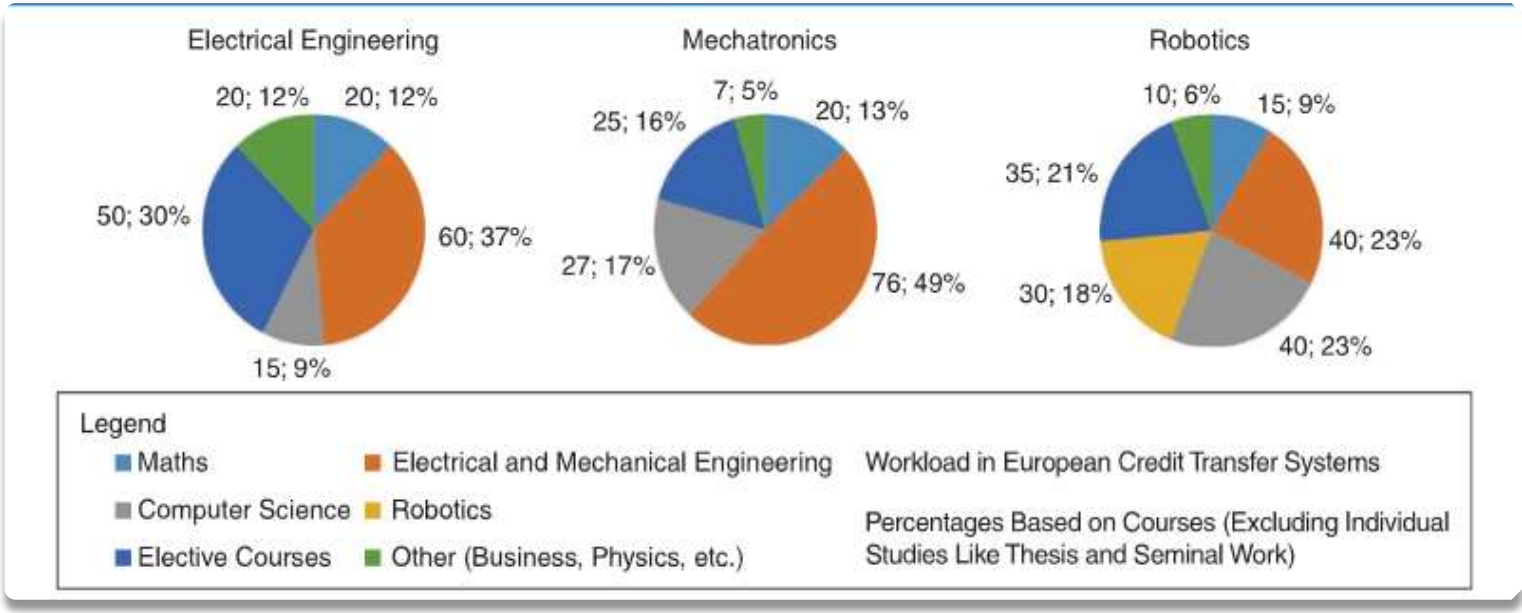
2. **Evaluate if Further Work is Needed on Completed Projects:** The course coordinator assesses whether the projects would benefit additional work, in terms of refactoring, missing tests, or further features.

3. **Solicit More Projects and for Different Courses:** The course coordinator decides whether the industrial partner could be interested in further projects within the same scheme and communication for the next round of the course is established.

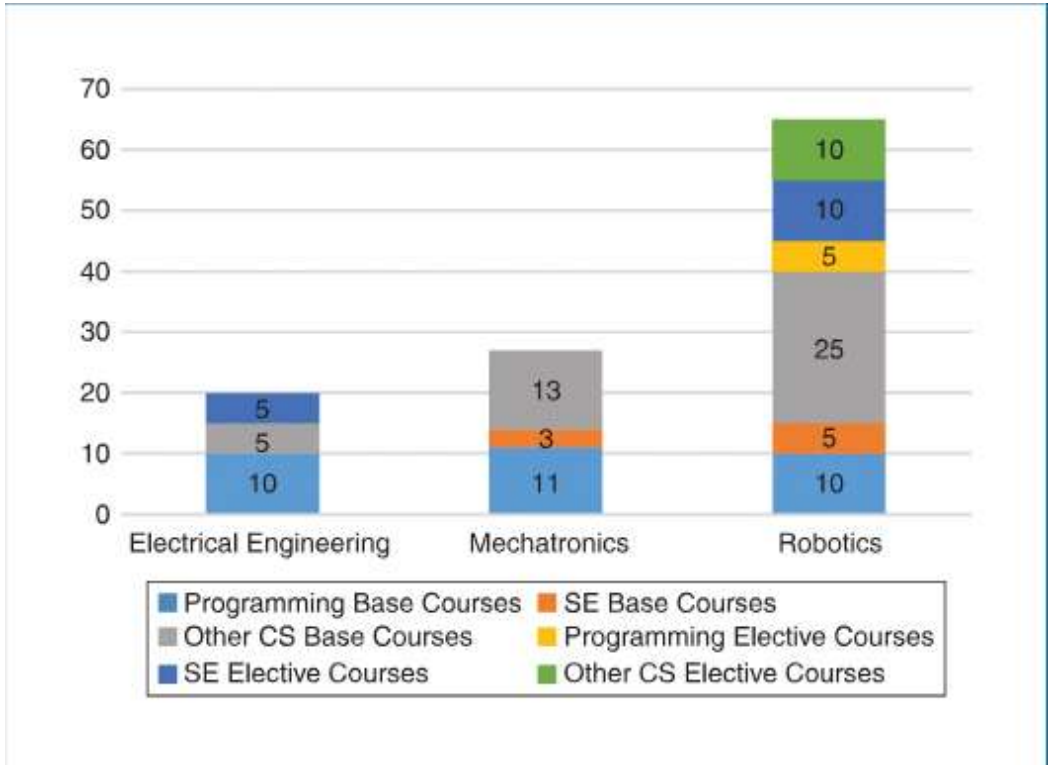
WHAT ABOUT OTHER TECHNICAL DEGREES?

- With the widespread networking of devices, embedded systems are increasingly connected and intertwined with information systems. Thus, technical engineering disciplines **also** require a strong education in software engineering.

- We compare the degree programs from the Technical University of Applied Sciences Würzburg-Schweinfurt.



WHAT ABOUT OTHER TECHNICAL DEGREES?



- Having a closer look at the elective courses in the different degree programs, we can see that the robotics degree offers considerably more courses on computer science courses.
- This can serve as a strong indication of the growing importance of computer science and especially software engineering in other than the classic computer science degree programs.

Software Engineering Body of Knowledge

- Software engineering body of knowledge (SWEBOK) divides the contents relevant for software engineers into different knowledge areas.

Knowledge area	2004	2016 (version 3)	In work (version 4)
Software requirements	X	X	X
Software architecture			X
Software design	X	X	X
Software construction	X	X	X
Software testing	X	X	X
Software engineering operations			X
Software maintenance	X	X	X
Software configuration management	X	X	X
Software engineering management	X	X	X
Software engineering process	X	X	X
Software engineering models and methods*	X	X	X
Software quality	X	X	X
Software security			X
Software engineering professional practice		X	X
Software engineering economics		X	X
Computing foundations		X	X
Mathematical foundations		X	X
Engineering foundations		X	X

*Called "Software engineering tools and methods" in the 2004 version.

CHALLENGES FOR SOFTWARE ENGINEERING EDUCATION

1. **Technical Engineering Degrees Require Specialized Software Engineering Education:**

   - Robotics students do not only need a good understanding of the electrical and mechanical parts of a robot but also have to work on advanced software development projects and realize complex business logics, connecting two worlds and multiple systems.

2. **Time Limits the Topics We Can Teach in Software Engineering Education:**

   - We must be selective about what to teach. It is impossible to cover all the important aspects of software engineering in just one course, if we want to reach a level of proficiency that allows students to apply their software engineering skills.

3. **What shall we teach them or, maybe an even better question: How shall we teach them?**

Solution to challenge 2

- There exist approaches for reducing the extent of software engineering practices and skills: we revisit the SWEBOK topics with the structure of robotics degree program in mind.
- We can identify topics that are less relevant for embedded systems and need not be part of the basic software engineering course. Thus, we end up with a list of topics to be included in the software engineering course itself.

Knowledge area	Course
Software requirements	Software engineering
Software architecture	Software engineering
Software design	Software engineering
Software construction	Programming
Software testing	Software engineering
Software engineering operations	-
Software maintenance	-
Software configuration management	-
Software engineering management	Project
Software engineering process	Software engineering
Software engineering models and methods*	Software engineering
Software quality	Software engineering
Software security	Software engineering
Software engineering professional practice	Project
Software engineering economics	Business
Computing foundations	Computer engineering and programmin
Mathematical foundations	mathematics
Engineering foundations	Electrical engineering

Solution to challenge 3

- Teaching software engineering to engineering students should place a greater emphasis on the crafts and skills involved in software development, such as **requirements elicitation**, **architecture design** and the **overall understanding of software engineering** as a discipline with multiple approaches to problem-solving.

- In software engineering, the degree of freedom for finding a solution is much greater and this concept is something that engineering students seem to struggle. Students need to identify and choose among multiple solutions that may vary in the degree of appropriateness and may be perceived as better or worse, depending on the specific situation.

Education with experience
Assessment of a Co-op model in undergraduate engineering programs in computing

This framework highlights the need to integrate practical experience into the software engineering curriculum.

While traditional lectures provide theoretical knowledge, hands-on projects, internships, and simulations bridge the gap to real-world development, helping students build coding, problem-solving, collaboration, and project management skills essential for the workforce.

Key concepts

- Real-world Exposure:
  - Students engage in projects that reflect real-world challenges, often sponsored by industry partners.
  - Through co-ops, internships, and collaborations with companies, students gain direct experience managing deliverables, responding to client feedback, and dealing with project constraints.
  - This exposure allows them to familiarize themselves with industry-standard tools, methodologies, and workflows

- Simulated Environments:
  - Classrooms are reimagined as simulated development labs where students work in teams on long-term projects that mimic professional environments.
  - Agile frameworks, sprints, and other project management practices are introduced to replicate real software lifecycle stages.
  - Students are assigned roles like project manager, lead developer, or quality assurance engineer to experience different facets of the development process.

- Soft Skill Development:

  - In addition to technical knowledge, teamwork, leadership, and communication skills are emphasized as part of project-based learning.

  - Peer code reviews, team presentations, and collaborative planning sessions help students develop interpersonal skills that are essential in software development teams.

Examples of Implementation

Partnerships:

Universities partner with tech firms, offering students real projects that involve meeting deadlines, refining requirements, and delivering software solutions.

Cross-disciplinary Teams:

Students from different academic backgrounds (e.g., computer science, business, and design) work together to foster diverse thinking and holistic project approaches.

Hackathons and Competitions:

Participation in competitions encourages quick, innovative thinking and exposes students to high-pressure development environments that simulate startup or tech industry scenarios.

Outcomes

- Graduates enter the workforce with robust portfolios of completed projects that demonstrate technical and collaborative abilities.
- Exposure to real-world environments enhances adaptability and prepares students to contribute effectively from day one.
- Employers benefit from hiring graduates with hands-on experience, reducing the need for extensive post-hire training.

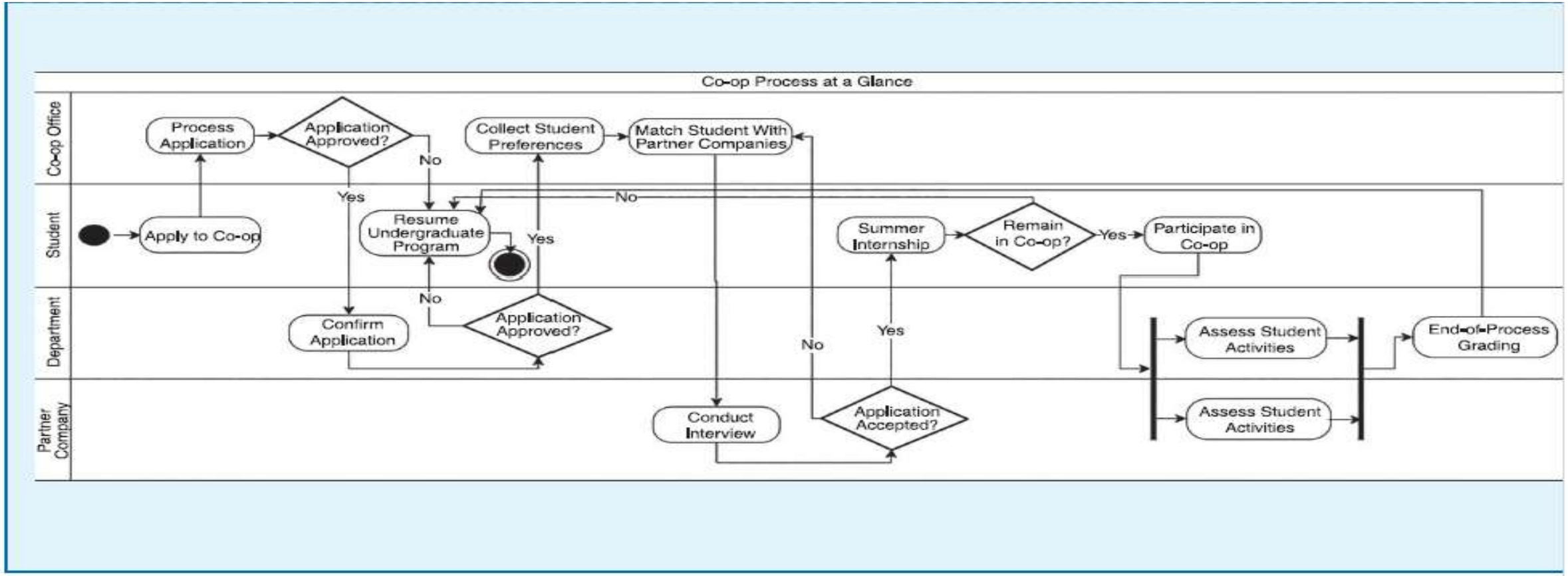


FIGURE 1. The co-op process swim lane diagram.

Learning by redoing

This model highlights the value of iterative learning through revisiting and refining work.

In software development, code rarely reaches perfection on the first try, and improvements come through rework and adjustments.

This approach encourages students to redo projects, enhance code quality, and refine solutions as their understanding deepens, mirroring industry practices where code evolves via continuous integration, deployment, and regular refactoring to maintain efficiency.

Major points

- Iteration as a Learning Tool:
  - Students are encouraged to treat projects as living documents, returning to them with new knowledge to improve and expand their functionality.
  - This aligns with agile methodologies where iterative refinement drives software development.
  - Reworking past assignments helps reinforce concepts and allows students to see their progress over time.

Feedback Loops:

- Regular, structured feedback from instructors and peers ensures students receive targeted advice on areas for improvement.

- Peer reviews and instructor evaluations simulate industry code review processes, teaching students how to critique and refine their work.

Growth Mindset

- Encourages students to view errors and feedback as opportunities for growth, fostering resilience and adaptability.
- Promotes experimentation and risk-taking by normalizing the need for rework as a natural part of development.

Practical Applications

- Phased Projects: Students work on large assignments over multiple stages, refining their solutions at each phase.
- Layered Grading: Grading systems reward students for improving and iterating on their code rather than simply producing perfect work on the first try.
- Portfolio Development: By continuously refining projects, students build portfolios showcasing both their technical skills and ability to evolve solutions over time.

Outcomes

- Enhances long-term retention of key concepts through repetition and active engagement.
- Fosters a culture of continuous improvement, reflecting real-world agile and DevOps practices.
- Encourages students to embrace failure as part of the learning process, enhancing better problem-solving skills and software quality.

Case Study:
A Replication and Expansion of the Original Code Mangler Study

- The Code Mangler study examines common coding errors, teaching methods, and code quality improvements, identifying student mistakes and evaluating strategies to help educators refine curricula and enhance coding skills.

- This section replicates and expands the Code Mangler study to assess if past error patterns persist in modern students and if advancements in educational tools have improved performance, providing insights into the effectiveness of current curricula.

Study Goals

- **Validation**:
  - The replication ensures that the findings of the original Code Mangler study remain relevant by recreating the initial experimental conditions.
  - It evaluates whether the same patterns of coding errors appear, offering a benchmark for evaluating curriculum changes over time.

Expansion:

- New technologies, programming languages, and IDEs are incorporated to reflect contemporary software development environments.

- Automated testing tools and collaborative platforms are integrated to see how they influence code quality and student performance.

- **Comparison:**
  - The results are compared with the original study, highlighting shifts in error trends and identifying new areas for improvement

- **Findings:**

  - Persistent error patterns indicate the need for stronger debugging and error management training in early coursework.

  - Modern tools reduce common coding errors with features like error detection and automation but introduce new challenges, such as increased complexity and tool management.

  - Consequently, the curriculum must adapt to teach students both how to use these tools effectively and navigate their complexities.

Case study:
Using Wireframes in a Capstone Software Engineering Class

- This practice explores using wireframes in capstone software engineering courses to visualize app structure and layout before coding, ensuring early alignment on design and functionality.

- Wireframes are basic visual layouts of an application or website, outlining structure and functionality without design details. They help align teams on content, navigation, and UI before development starts.

- They enhance communication, promote team collaboration, and reduce costly design changes later.

- The use of wireframes offer:
  - Visualization:
    - Provides a blueprint that guides development and allows teams to conceptualize user interfaces and workflows.
  - Alignment and Communication:
    - Wireframes serve as a communication tool between stakeholders, ensuring everyone is on the same page.
  - **Outcomes:**
    - Projects using wireframes reported clearer design goals and smoother development phases with students taking decisions early in the process and approaching the coding milestone with more confidence.

”

THANK YOU