Embedding Perseverance in Your Computing Course

Perseverance is a disposition, when students keep trying to do something despite unforeseen obstacles. This disposition is critical both in college and career settings for several reasons:

Why to Embed

- **Overcoming challenges:** Both college and career paths are filled with challenges and obstacles. Perseverance helps people persist through difficulties, learn from failures, and continue working towards their goals.
- **Achieving long-term goals:** Many goals are long-term and require sustained effort. Perseverance ensures that individuals stay committed to these goals, even when immediate rewards are not evident.
- **Adapting to Change:** The ability to persevere is vital when adapting to change, whether it involves changing majors in college or shifting job roles in a career. It enables individuals to remain focused and productive amidst transitions.
- **Skill development:** Mastery of complex skills requires persistent practice and dedication. Perseverance is key to developing and refining these skills over time.
- **Building resilience:** Perseverance is a key factor in building resilience, an essential quality for coping with stress and setbacks.

How to Embed

To teach perseverance to college students, consider these strategies:

- **Integrate perseverance into the curriculum:** Include assignments and projects that require sustained effort over time and provide feedback that focuses on both the process and the outcome. See the examples and vignettes below for some ideas.
- **Provide support and resources:** Offer resources such as tutoring, counseling, and career advising. Knowing that support is available can help students persist through challenges.
- **Model perseverance**: As an educator or trainer, share your challenges and how you overcame them. Modeling perseverance can be a powerful teaching tool.
- **Encourage self-reflection:** Guide students in reflecting on their experiences, especially after setbacks. This can help them understand what went wrong and how to improve.
- **Create a supportive community:** Foster a community among students where they can share their struggles and successes, creating a sense of belonging and mutual support. Peer support can be a powerful motivator.
- **Set realistic goals:** Help students set achievable, incremental goals for their projects and academic goals. Achieving these smaller goals can build their confidence and commitment to larger objectives.
- **Teach time management and organization skills:** Equip students with skills to manage their time effectively and stay organized. This can reduce stress and help them persevere when faced with multiple demands.
- **Foster a growth mindset:** Encourage a mindset where challenges are viewed as opportunities to learn and grow, rather than insurmountable obstacles.

National Science Foundation

This material is based upon work supported by the National Science Foundation under Grants No. 2111097 and No. 2417492

Activity Ideas

Teaching perseverance in a college-level computing class involves designing activities that challenge students, encourage them to persist through difficulties, and help them develop resilience and problem-solving skills. Here are several examples of active learning activities tailored to foster perseverance:

- **Incremental coding challenges:** Start with very basic coding problems and gradually increase the complexity. These challenges should require students to apply logic and problem-solving skills, pushing them to persist and overcome obstacles.
- **Debugging simple programs:** Provide students with simple, pre-written code containing bugs. The task of identifying and fixing these bugs teaches patience and attention to detail.
- **Escalating difficulty tasks:** Design assignments or labs where the difficulty level gradually increases, requiring students to apply and extend their knowledge progressively and learn from prior experiences and their own mistakes.
- **Code refactoring exercises:** Give students poorly written or inefficient code and ask them to refactor it. This requires patience and persistence to improve the code while keeping its functionality.
- **Reverse engineering tasks:** Have students reverse-engineer a piece of software or hardware. This type of task can be very challenging and requires sustained effort and attention to detail.
- **Tech support simulation:** Create a role-play activity where students act as tech support staff, solving complex and frustrating user problems, teaching patience and perseverance in customer service scenarios.
- **Learning from failure discussions:** Hold regular sessions where students share their project failures or challenges and discuss what they learned, reinforcing the idea that persistence is key to overcoming obstacles.

Examples

Classroom Activities:

Vignette 1: First Year Computing Class - Incremental Coding Challenges and Peer Review

In a first-year introductory programming course, the instructor introduces incremental coding challenges. The course begins with fundamental problems, such as writing a simple "Hello, World!" program. As weeks progress, the complexity increases. Students move from basic syntax to implementing loops, then to creating functions, and finally to data structures. Each new set of problems builds on the concepts from the previous ones.
Along with the incremental coding challenges, the instructor integrates peer review sessions. After completing each coding challenge, students are paired up to review each other's code. They provide feedback on coding style, efficiency, and problem-solving approaches. Additionally, encouraging students to reflect on their learning, challenges, and how they persevered through the task can help them gain confidence and create a model for future learning.

> **Outcome:** Students initially struggle but gradually gain confidence. They learn to persist through increasingly complex tasks, understanding that each challenge is a stepping stone to more complicated coding. The peer review sessions foster a collaborative learning environment. Students not only learn from their coding efforts but also gain insights from their peers' approaches. This collaboration

This material is based upon work supported by the National Science Foundation under Grants No. 2111097 and No. 2417492

encourages them to persevere through challenges, knowing they have the support and perspectives of their classmates to draw upon.

Vignette 2: Second Year Computing Class - Debugging Simple Programs & Code Refactoring Exercise
In the second-year software development course, students are not only tasked with debugging simple, pre-written code but also with refactoring it for efficiency and readability. After the initial debugging phase, where students fix a basic application with various bugs, the instructor introduces a code refactoring exercise. The students are given the same application, which now functions correctly but is poorly written – the code is inefficient, lacks proper structure, and has poor readability. The students must refactor the code, applying best practices such as DRY (Don't Repeat Yourself) principles, proper naming conventions, and efficient algorithms. They also need to document their code adequately.

In another instance, students are given a basic calculator program. After debugging, they discovered that the program contains redundant code blocks and is structured inefficiently. The refactoring task involves restructuring the code to use functions for common operations, optimizing the logic used for calculations, and ensuring the code is well-documented and easy to understand.

Reflections on what worked and what did not help students better understand their mistakes and get back on track with learning.

> **Outcome:** Through the refactoring exercise, students gain a deeper understanding of not just how to make a program work, but how to make it work well. They learn that writing clean, efficient, and maintainable code is as important as fixing bugs. This exercise teaches them the value of revisiting and improving their work, a crucial skill in software development. By the end of the course, students are not only adept at debugging but also at writing code that is efficient, readable, and easier to maintain – all key aspects of professional software development.

Vignette 3: Third Year Computing Class - Escalating Difficulty Tasks with Reverse Engineering and Debugging
In this third-year advanced programming course, the escalating difficulty of assignments is complemented by reverse engineering and debugging tasks. In addition to the initial straightforward functions, such as creating a user authentication system, the instructor introduces a mid-semester project focused on reverse engineering. Students are given a compiled version of a simple software tool, such as a basic text editor or a file management system, without access to its source code. Their task is to analyze and understand its functionality, identify its architecture, and then recreate or replicate a similar application themselves. This reverse engineering process requires them to apply their understanding of programming concepts, problem-solving, and critical thinking.

As the semester progresses, students are assigned a complex project, like building a small-scale database management system, with an added requirement: they must integrate components of the reverse-engineered software into their new project. This integration inevitably leads to bugs and compatibility issues, requiring students to engage in intensive debugging. The debugging process involves not only identifying and fixing issues but also optimizing the integration for efficiency and performance.

> **Outcome:** Students face significant challenges in reverse engineering and debugging, pushing them to apply their knowledge in new and unexpected ways. The process of deconstructing existing software teaches them about different programming styles and architectures, enhancing their understanding of software design. The subsequent debugging and integration work challenges them to be patient, meticulous, and innovative. By the end of the course, students have not only learned to build and integrate complex systems but also gained valuable skills in problem-solving, critical analysis, and

National Science Foundation
This material is based upon work supported by the National Science Foundation under Grants No. 2111097 and No. 2417492

perseverance. They emerge with a comprehensive understanding of the software development lifecycle, better equipped to tackle real-world software engineering challenges.

Vignette 4: Fourth Year Computing Class - Tech Support Simulation and Learning from Failure Discussions
In a fourth-year computing class, the instructor combines tech support simulation with discussions on learning from failure. Students role-play as tech support staff, dealing with complex and frustrating user problems. These scenarios are interspersed with sessions where students discuss their past project failures or challenges and the lessons learned.

Outcome: The tech support simulation teaches students to be patient and persevere through user issues, while the failure discussions help them understand that setbacks are part of the learning process. They leave the course better prepared for real-world IT challenges.

National Science Foundation

This material is based upon work supported by the National Science Foundation under Grants No. 2111097 and No. 2417492