27.3 Strategic vs Tactical Programming

Tactical Programming

The goal is to get something working quickly, often using workarounds. Consider code that has many if statements to handle many separate cases to pass autograder tests that is challenging to update and explain.

Prototypes, proof-of-concepts often leverage tactical programming, to show that something could theoretically work.

However:

- There's no time spent on overall design
- Code is complicated
- Refactoring takes time and potentially means restarting
  - If you didn't plan for Project 2 runtime requirements, you would have to redo the constructor and the entire project
- Proof of concepts are sometimes deployed in the real world due to lack of time

Strategic Programming

The goal is to write code that works elegantly - at the cost of planning time, to reduce coding time. This emphasizes long term strategy.

Code should be:

- Maintainable to fix bugs
- Simple to understand
- Future-proof to add new functionality
  - 61B projects have deadlines; afterwards, you can throw it away

If the strategy is insufficient, go back to the drawing board before continuing work.

Helper method strategy is key to leverage throughout projects, especially when we have written comprehensive tests to ensure that these methods are correct.

Previous
27.2 Complexity

Next
27.4 Real World Examples

Last updated 7 months ago