Productive Programming

CS-370: Software Design and Development

Robert Bruce

What is “productive programming”?

- Productive programming are techniques for improving your efficiency throughout the software development lifecycle.
- Productive programming is implemented through:
  - Tools
  - Algorithms
  - Top-down design
  - Programmming tips and tricks

Source: pages 246-255, *Beginning Software Engineering* (2nd edition) by Rod Stephens

Productive programming: Tools (part 1 of 2)

- **Hardware:** a slow computer impacts build-times (compilation).
- **Network:** Internet access is vital to download open-source development libraries, apply operating system updates (infrastructure), view development library documentation, etc.
- **Development environment:** The development environment includes the compiler or intrepreter necessary to build applications. Frequently programmers use an Integrated Development Environment (IDE) which includes an editor, source-level debugger, and compiler (or interpreter) to develop, debug, build, and run applications.
- **Source code control:** A revision control system that allows a software development team to manage a software repository (code-base) in an organized manner.
- **Profiler:** Measures an application's performance in areas such as amount of memory used, execution time, areas in the application that are executed the most, etc. Profilers are very useful to determine bottlenecks in an application (where is performance being impacted the most).
- **Static analysis tools:** Static analysis tools measure aspects of an application's source code such as for how well a program is documented (number of comments) or how complex a program is (e.g. deeply-nested calls to other functions could indicate an overly fragmented application).

Source: pages 246-250, *Beginning Software Engineering* (2nd edition) by Rod Stephens

Productive programming: Tools (part 2 of 2)

- **Testing tools:** Automated tools to detect software defects. Rob note: one type of testing tool is a fuzzer to determine if there are any potential software defects like buffer overflows or a pointer accessing non-existent memory (general protection fault).
- **Source code formatters:** Automatically formatting programs (presuming the source code is written in a format-free language) with statement-block indention. This is critical to making a code-base consistent in readability. Rob note: formating source code consistently makes it easy to visually identify errors such as a “dangling else” in a nested series of if-then-else statements.
- **Refactoring tools:** These tools are useful for merging various code revisions from various software developers into the master source code repository.
- **Training:** Training resources can demystify how to program a complicated application programmer interface or how to effectively use a complex integrated development environment, etc.
- **Collaboration tools:** These usually online tools promote collaboration, organization, and communication (both synchronous and asynchronous) among the software project developers. Examples include: Chanty, Discord, Mattermost, Microsoft Teams, Ryver, Slack, Trello, etc.

Source: pages 249-251 , *Beginning Software Engineering* (2nd edition) by Rod Stephens

Productive programming: Algorithms

- “Algorithms are like a recipe for solving a hard programming problem.”
- Rob note: There are many excellent algorithms books out there. I suggest building your collection of algorithms books as you need them, preferably in hardcopy format. Some algorithms are specific to particular areas of interest such as computer vision or compiler construction while other algorithms are more general-purpose. I also recommend a mathematical handbook and a general math book on statistical methods targeted for engineers and scientists.
- General-purpose algorithm books:
  - *Numerical Recipes* (3rd edition) by William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery
  - *Algorithms* (4th Edition) by Robert Sedgewick and Kevin Wayne.
  - *The Art of Computer Programming* by Donald Ervin Knuth.
- Math books I recommend (depending on your point-of-need):
  - *Physics for Game Developers* by David M. Bourg and Bryan Bywalec.
  - *3D Math Primer for Graphics and Game Development* (2nd Edition) by Fletcher Dunn and Ian Parberry.

Source: pages 251, *Beginning Software Engineering* (2nd edition) by Rod Stephens

Productive programming: Top-down design

- *Top-down design* also known as *stepwise refinement* is the process of breaking a high-level problem into smaller, more detailed components.
- Rob note: The top-down design technique promotes modular programming. It is easy to identify a code section that repeats over-and-over. Consequently, such repeated patterns should be implemented as a callable function or procedure.

Source: pages 252-254, *Beginning Software Engineering* (2nd edition) by Rod Stephens

Productive programming: Tips and tricks (part 1 of 3)

- **Be alert:** Avoid writing software when you are tired. Take frequent breaks. Rest.
- **Write for people, not the computer:** Choose meaningful names for variables, functions, procedures, and classes. Preferably, write software in a high-level programming language (as opposed to assembler).
- **Comment first:** Comments are like a future note to yourself – what you were thinking or intended to produce for an unfinished software project. Comments could also describe what a code section is supposed to achieve (e.g. expected inputs and resulting outputs).
- **Write self-documenting code:** Use enumerated types with descriptive names. Use descriptive constants instead of hard-coded values.
- **Keep it small:** If possible, avoid long statement-blocks inside an interation statement. This makes it easier to follow the flow-of-control. If a long statement-block is required, put comments at the close of the statement block to indicate what this loop is iterating on.
- **Stay focused:** Keep object-oriented classes and methods small. If the class or method can't be described in one sentence, that class or method is too complex.

Source: pages 255-261, *Beginning Software Engineering* (2nd edition) by Rod Stephens

Productive programming: Tips and tricks (part 2 of 2)

- **Avoid side effects:** “A side effect is an unexpected result of a method call.” Rob note: side effects can also result from manipulating a global variable inside a local function or procedure. Side effects can also occur when a pointer’s value is manipulated through pass-by-reference to a function or procedure.
- **Validate results:** Add validation to your inputs within a function or procedure. Also validate inputs to a method. For object-oriented programming, use assertions to throw an exeception error for invalid input. For not object-oriented programming, you can return an error code or NULL to indicate failure.
- **Practice offensive programming:** Make your program work for all possible inputs and take appropriate action. In doing so, your program maintains control for all inputs (regardless of validity) without experiencing unexpected results. Rob note: This is especially important to alleviate zero-day exploits.
- **Use exceptions:** If the programming language supports exception handlers, it is preferable to throw an exception error rather than return an error code to the calling program since the calling program may ignore error code return results.

Source: pages 261-266, *Beginning Software Engineering* (2nd edition) by Rod Stephens

Productive programming: Tips and tricks (part 3 of 3)

- **Write exception handlers first:** When an exception occurs due to an assertion failure, the infrastructure is in place to process the exception!
- **Don't repeat code:** Repeated code should be implemented in a function or procedure to avoid code redundancy. Rob note: this technique can significantly reduce a compiled application's storage and memory footprint and result in faster application load-times as well.
- **Defer optimization:** "First make it work. Then make it faster if necessary."

Source: pages 266-267, *Beginning Software Engineering* (2nd edition) by Rod Stephens