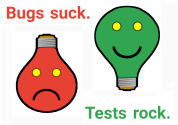
Testing on the Toilet Presents... ***Healthy Code on the Commode***



// Let Code Speak for Itself

by Shiva Garg and Francois Aube

Comments can be invaluable for understanding and maintaining a code base. But *excessive* comments in code can become unhelpful clutter full of extraneous and/or outdated detail. **Comments that offer useless (or worse, obsolete) information hurt readability.** Here are some tips to let your code speak for itself:

- **Write comments to explain the "why"** behind a certain approach in code. The comment below has two good reasons to exist: documenting non-obvious behavior and answering a question that a reader is likely to have (i.e. *why doesn't this code render directly on the screen?*):

```
// Eliminate flickering by rendering the next frame off-screen and swapping into the
// visible buffer.
RenderOffScreen();
SwapBuffers();
```

- **Use well-named identifiers** to guide the reader and reduce the need for comments:

```
// Payout should not happen if the user is
// in an ineligible country.
std::unordered_set<std::string> ineligible =
  {"Atlantis", "Utopia"};
if (!ineligible.contains(country)) {
  Payout(user.user_id);
}
```

```
if (IsCountryEligibleForPayout(country)) {
  Payout(user.user_id);
}
```

- **Write function comments (a.k.a. API documentation) that describe intended meaning and purpose, not implementation details.** Choose unambiguous function signatures that callers can use without reading any documentation. Don't explain inner details that could change without affecting the contract with the caller:

```
// Reads an input string containing either a
// number of milliseconds since epoch or an
// ISO 8601 date and time. Invokes the
// Sole, Laces, and ToeCap APIs, then
// returns an object representing the Shoe
// available then or nullptr if none were.
Shoe* ModelAvailableAt(char* time);
```

```
// Returns the Shoe that was available for
// purchase at `time`. If no model was
// available, throws a runtime_error.
Shoe ModelAvailableAt(time_t time);
```

- **Omit comments that state the obvious**. Superfluous comments increase code maintenance when code gets refactored and don't add value, only overhead to keep these comments current:

```
// Increment counter by 1.
counter++;
```

Learn more about writing good comments: To Comment or Not to Comment?, Best practices for writing code comments

More information and archives: `testing.googleblog.com`