Week 6: Inspections

- What do we inspect?
- When do we inspect it?
- Who does the inspection?
- How do we resolve disagreements?

Goals of code inspection

a. Enforcing rules, standards
b. Spotting errors, bugs
c. Suggesting improvements

Which is most important?
When do we focus on each?
What else (besides code) do we inspect?

Levels of programming standard

- Mandatory standard:
  - Deviation requires prior approval.
  - Request to deviate must compare cost of compliance with cost of allowing the deviation.
  - Deliberate unapproved violation is a serious offense.

- Convention:
  - Comply unless you have a good reason not to.
  - Be prepared to state convincing explanation.
  - Habitual deviations raise questions.

- Guideline:
  - Recommended practice, helpful suggestion.
  - Programmers are free to follow or not.

Purpose of mandatory standards

- Deviation may have impact beyond the accountability of the development team:
  - global impact
    - effect on other applications
    - effect on computer operations
  - future impact
    - effect on maintainability
    - effect on flexibility

- An enlightened organization has few mandatory standards.
  Why? Examples?
Two styles of programming standards for an organization

<table>
<thead>
<tr>
<th>Level:</th>
<th>Military style</th>
<th>Professional style</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nearly everything is mandatory; rigid rules.</td>
<td>Mostly conventions &amp; guidelines; flexible.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Source:</th>
<th>Expert(s)</th>
<th>Staff participation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scope:</td>
<td>Everything that can be standardized.</td>
<td>Things that are cost justified</td>
</tr>
<tr>
<td>Enforcement</td>
<td>Adversarial / confrontational</td>
<td>Cooperative; mutual respect &amp; trust</td>
</tr>
</tbody>
</table>

What's the third choice?

Walkthrough presentations
- Popularized as part of the "structured revolution" in the late 1970s
- May occur:
  - Before coding (design walkthrough)
  - After development
  - Any time the programmer seeks help/advice
- Aims are:
  - to spot errors
  - to suggest improvements
  - not a standards enforcement forum
- Depends upon "egoless programming" (Weinberg)

Egoless programming (Gerald Weinberg)
- Programmer doesn't feel obliged to defend his or her work.
  - Suggestions for improvement are not taken as criticisms of the programmer's competence or judgment.
    (unless it's really, really bad! *In what ways?*)
  - Programmers at all levels should welcome such suggestions
    - as valuable learning experiences.
    - as contributions to product quality.
- Some programmers can't handle that. *What kind of programmers?*

Walkthrough participation
- Developer distributes copies of module(s) to be reviewed a few days in advance
- Just preparing the presentation often helps the programmer to spot problems without any group participation.
- Review group consists of
  - 2, 3, or 4 peers
  - (optional) a methodology expert
  - no supervisors or user representatives  *Why not?*
- Emphasis on spotting issues
  - Don't try to solve complicated problems as a group.
A dilemma?

"Inspecting software still under construction may waste inspection effort on elements that are likely to change, but waiting until after integration and system test wastes testing effort on faults that could have been more cost-effectively removed by inspection."

- Pezze & Young, p. 344

Who should decide? When?

Spotting errors

Q: What's the number 1 contributor to both spotting bugs and avoiding bugs in the first place?

A: Readability and simplicity in the code.

- prefer obvious solutions
- if it's not obvious, explain it clearly

What about efficiency?

- prefer existing modules to custom ones

With a well-designed and carefully-coded MUT unit-testing should rarely take as long as design & coding

Is that our experience?

A fresh view

Sometimes a programmer encounters an extremely frustrating bug.

- The programmer may have a "mental block",
- knows what the MUT was supposed to do, and the code looks as if that's just what it does!
- It's often helpful to bring in another programmer.
- The act of explaining it to someone else often yields insight.
- Or just take a break to clear your mind.

Pair programming is a popular variation.

- One programmer prepares and runs the test code.
- The other acts as a reviewer, trying to spot problems
- They exchange roles from time to time.

Should a programmer test his or her own programs?

Unit test:

- Usually, yes, because of unforeseeable interleaving of activity. It's too hard to separate coding, testing, correcting, & revising.

System test (and beyond):

- Usually no, because the programmer, often subconsciously, wants the program to work. Remember Glenford Myers's advice.

A possible compromise (for both): pair programming! see http://en.wikipedia.org/wiki/Pair_programming
A pair programming session

- Starts with unit test!
  - One programmer enters new code (driver or MUT).
  - The other one checks for errors, oversights, bugs, opportunities for improvement, etc.
- One is senior – highly experienced; the other may be an apprentice.
  - Which one does which function?
  - They may exchange roles from time to time
  - Can be a valuable learning experience

- Possible complications:
  - Fatigue factor
  - Brilliant inspiration factor

Should it be optional or mandatory?

Discovering & reporting a bug

- Type:
  - Wrong results
    - According to whom?
  - Loss of control
    - Crash, infinite loop, etc.

- Severity:
  - Fatal
    - Application can't be used at all, or
    - May expose users to serious intolerable consequence
  - Serious
    - Users may experience loss or embarrassment
  - Tolerable
    - Users may be irritated or may waste time

Emergency repair

- Sometimes a fatal or serious bug is discovered in a production application, i.e. one that was thought to have been thoroughly tested and is now in essential operation.

- Enormous pressure to "fix" the problem immediately (e.g. in the middle of the night) so we can get work done and avoid penalties or embarrassment.
  - Must we re-run all the original tests?
  - What short-cuts are acceptable?
  - Who should authorize them?

Emergency repair drawbacks

- Results may look as if the problem was fixed, but the lack of thorough testing may lead to subtle but serious future problems.

  "Quick & dirty" fix, intended to be temporary, may become permanent under the pressure of other commitments.

What's wrong with that if it works?