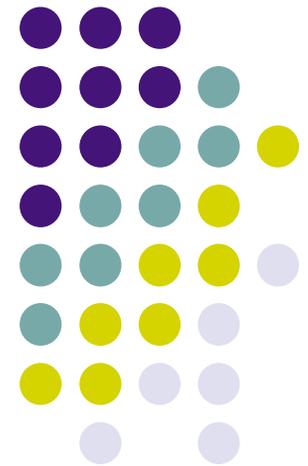


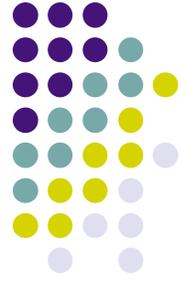
Better Software No Matter What

Rahman Syed, AMTI
SIVO Brown Bag Presentation
April 11, 2008

Shamelessly paraphrased from
SDE expo 2008 presentation entitled
“Better Software – No Matter What”
by Scott Meyers, Aristeia

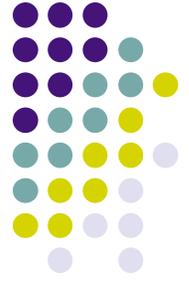


Better Software No Matter What



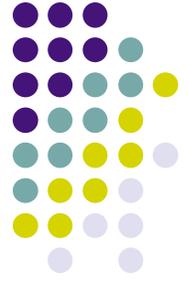
- Software quality is a global optimization problem based on both external and internal characteristics
 - External – does it do what it's supposed to?
 - Internal – characteristics of the code itself
 - Customers also care about internal quality!

Better Software No Matter What



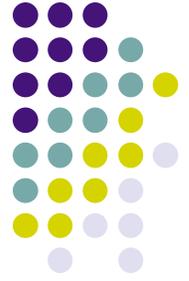
- General approach to optimize both external and internal quality
 - Insist on high quality, don't compromise
 - Defects are embarrassments
 - Offer guidelines that lead to higher quality

Better Software No Matter What



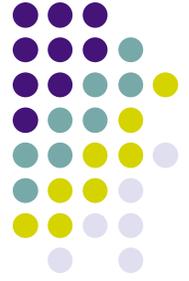
- Local vs. Global Optimization
 - Achieving quality for individual jobs/tasks does not guarantee quality for the project
 - Example: one team focuses on delivering a component that is fast, the other team focuses on portability. Resulting product: neither fast, nor portable
- Management of programmer discretion is critical to software quality

Better Software No Matter What



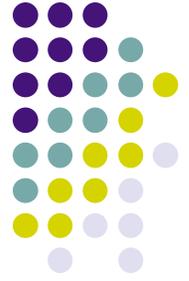
- Insist on a useful specification
 - Complete and detailed enough to answer questions during design and implementation
 - Unambiguous enough that different people are likely to interpret it the same way
 - Taken seriously by the people creating, approving, and using it

Better Software No Matter What



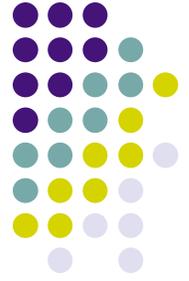
- Design by Contract
 - Preconditions: conditions that must be true upon entry to the function
 - Postconditions: conditions that must be true upon exit to the function
 - Invariants: relationships amongst parts that must be true between independent operations

Better Software No Matter What



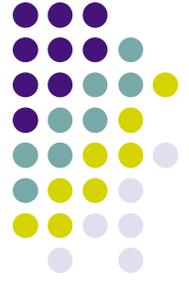
- Design by Contract is a design methodology that identifies and documents “contracts” between callers and callees
 - Callers must satisfy a function’s preconds
 - Callees must satisfy a function’s postconds
 - Everyone must maintain invariants

Better Software No Matter What



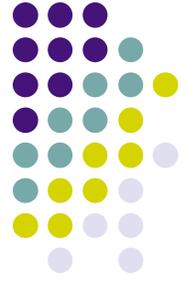
- Benefits of Design by Contract
 - Clearer thinking about requirements before, during, and after method execution
 - Documentation of these ideas
 - Focusing on interfaces rather than implementations

Better Software No Matter What



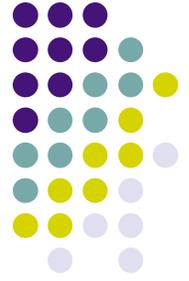
- Make interfaces easy to use correctly, hard to use incorrectly
 - People using interfaces are generally smart and motivated
 - They're experienced with software
 - They're willing to read some documentation
 - They want to use your software correctly
 - **Implication: if they use your software improperly, it's your fault**

Better Software No Matter What



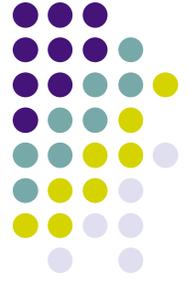
- Principle of Least Astonishment
 - Maximize the likelihood that user expectations about an interface are correct
 - Expectations can be just guesses: your job is to maximize the odds that guesses are correct
 - If users know what they want to do, they should be able to guess how to do it

Better Software No Matter What



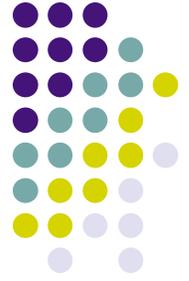
- Guideline
 - Adhere to the principle of least astonishment
 - Avoid gratuitous incompatibilities with the surrounding environment
 - Avoid undefined behavior
 - Choose good names
 - Shoot for “nice” classes
 - Be consistent
 - Document interfaces before implementing them
 - Introduce new types to prevent common errors
 - Types, not type synonyms
 - Consider explicitly defining all possible values for the type
 - Automatically generate the types, if possible
 - Minimize use of naked “string” as a type
 - Minimize potential resource leaks

Better Software No Matter What



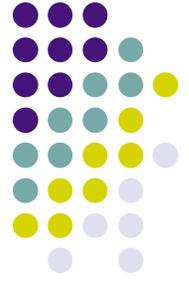
- Embrace static analysis
 - Woefully underused
 - Detectable problems:
 - Likely design violations
 - Likely omissions
 - Likely logic errors
 - Likely inefficiencies
 - Likely security issues
 - Likely typos
 - Likely violations of local coding standards

Better Software No Matter What



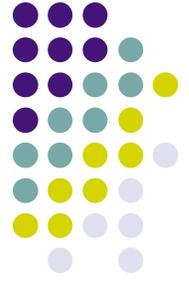
- Survey of static analysis tools/methods:
 - Compiler warnings – compilers probably know the language better than you do
 - Try to require clean compiles at max warning level
 - Don't become dependent on compiler warnings (vendors vary on what they report)
 - lint and lint-like utilities
 - Requires non-trivial up-front investment (signal/noise ratio is unacceptable out of the box)

Better Software No Matter What



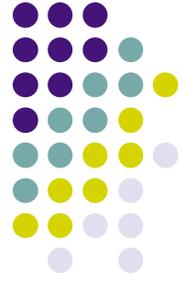
- Survey of static analysis tools/methods:
 - Code reviews (informal)
 - Ad hoc review: over-the-shoulder review of your code
 - Deskcheck: you review a printed copy of your own code
 - Peer deskcheck: someone else reviews a printed copy of your code
 - Passaround: several people perform independent deskchecks
 - Peer programming: two people program at the same computer
 - Walkthrough: you walk through and explain your code to others

Better Software No Matter What



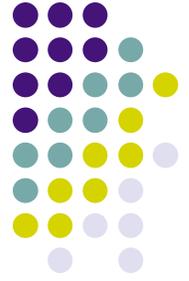
- Survey of static analysis tools/methods:
 - Code reviews (formal)
 - Team review: planned, somewhat structured multi-role review of your code, possibly including post-review followup activities
 - Inspection: fully planned, fully structured multi-role review of your code, followed by specific followup activities
 - Benefits of reviews
 - Less sloppy coding (more careful when you know others will read what you write)
 - Knowledge transfer among programmers
 - Earlier defect detection

Better Software No Matter What



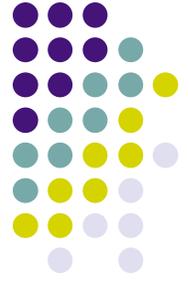
- Retrospectives - structured mechanism for learning from past work
 - Identify what worked well
 - Identify what should be done differently
 - Help participants achieve “closure”
- Why?
 - Improve the development process
 - Enable people to focus on the future
- When?
 - End-of-project
 - End-of-milestone
 - End-of-iteration
 - Longer project \diamond longer retrospective
 - Sooner after the project, the better

Better Software No Matter What



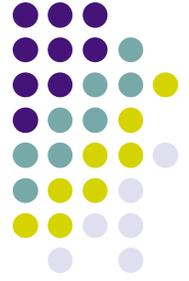
- Who? (nobody knows the full story)
 - Customers
 - Requirements analysts
 - Developers
 - Testers
 - Managers
 - A (skilled, neutral, trusted) facilitator
 - A scribe

Better Software No Matter What



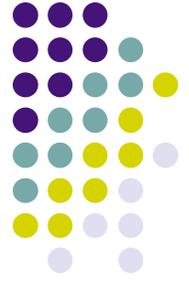
- Safety
 - Participants must feel safe
 - Express themselves without fear of repercussions
 - Facilitator and participants both must work on safety
 - **Kerth's Prime Directive: Regardless of what we discover, we understand and truly believe that everyone did the best job they could, given what they knew at the time, their skills and abilities, the resources available, and the situation at hand**
 - The goal is learning, not blaming
 - Without safety, a retrospective cannot succeed

Better Software No Matter What



- Preparation
 - Determine retrospective objectives
 - Gather project data and artifacts
- Meeting
 - Create and discuss project record
 - Prioritize topics and analyze most important
 - Develop action plans
 - Evaluate the retrospective
 - Interactive and participatory, not a presentation
- Post-meeting
 - Follow-through: ensure that action items are pursued

Better Software No Matter What



- Reminder that the content was taken **completely** from slides provided by Scott Meyers during his presentation “Better Software—No Matter What” at Software Development Expo 2008
- Will make copies of the full slide handout available if people are interested