Part III. Other QA Alternatives

- Inspection and defect prevention
- Formal verification
- Fault tolerance and safety
- Summary and comparison
QA Alternatives

- Defect and QA:
  - Defect: error/fault/failure.
  - Defect prevention/removal/containment.
  - Map to major QA activities

- Defect prevention:
  - Error source removal & error blocking

- Defect removal: Inspection/testing/etc.

- Defect tolerance:
  - Fault tolerance (failure↓)
  - Damage minimization (safety)
Ch. 13: Defect Prevention

- Error blocking
  - Error: missing/incorrect actions
  - Direct intervention
  - Error blocked
    - fault injections prevented
    - (or errors tolerated)
  - Rely on technology/tools/etc.

- Error source removal
  - Root cause analysis
    - identify error sources
  - Removal through education/training/etc.
Ch.13: Defect Prevention

- Education/training to eliminate error sources
  - product/domain knowledge,
  - development methodology,
  - development process, etc.

- Technology to prevent fault injection
  - process/product technology
  - tools: e.g., syntax directed editor

- Causal and risk analysis
  - Analyze pervasive defects
  - Cause identification and fixing
  - Risk analysis to focus/zoom-in
Ch.14: QA and Inspection

- QA alternatives/activities:
  - inspection,
  - formal verification,
  - testing,
  - fault tolerance, etc.

- Inspection as part of QA:
  - Throughout the software process
  - Many different software artifacts:
    - program code, typically
    - requirement/design/other documents
    - charts/models/diagrams/tables/etc.
  - People focus
  - Complementary to other QA activities
Ch.14: Inspections: Overview

- Generic process/steps:
  - Preparation (individual)
  - Collection (group/meeting)
  - Repair (followup)

- Inspection Process Variations:
  - Team size
  - Number/coordination of multiple sessions
  - Collection technique
  - Defect detection (and classification/analysis)
  - Use of post-collection feedback
Ch.14: Fagan Inspection

• General description
  ▶ Earliest, Fagan at IBM
  ▶ Lead to other variations
  ▶ Generic process and steps

• Six steps of Fagan inspection:
  ▶ Planning
  ▶ Overview (1-to-n meeting)
  ▶ Preparation (individual inspection)
  ▶ Inspection (n-to-n meeting)
  ▶ Rework
  ▶ Follow-up
Ch.14: Fagan Inspection

● Planning
  ▶ Entry criteria: what to inspect
  ▶ Team size: about 4 persons
  ▶ Developers/testers from similar projects
  ▶ Effectiveness concerns (assumptions)
  ▶ Inspectors not authors

● Overview
  ▶ Author-inspectors meeting
  ▶ General background information
    – functional/structural/info., intentions
  ▶ Assign individual tasks:
    – coverage of important areas
    – moderate overlap
Ch.14: Fagan Inspection

- Preparation or individual inspection
  - Independent analysis/examination
  - Code as well as other document
  - Individual results:
    - questions/guesses
    - potential defects

- Inspection (generic: collection)
  - Meeting to collect/consolidate individual inspection results
  - Team leader/meeting moderator (1)
  - Reader/presenter: summarize/paraphrase for individual pieces (assignment)
  - Defect identification, but not solutions, to ensure inspection effectiveness
  - No more than 2 hours
  - Inspection report
Ch.14: Fagan Inspection

- Rework
  - Author’s response
  - Defect fixing (solutions)

- Follow-up
  - Resolution verification by moderator
  - Re-inspection?

- Fagan inspection in practice
  - Widely used in industry
  - Evaluation studies
  - Variations and other inspections
Ch. 14: Other Inspection Methods

- Many inspection methods:
  Variations to Fagan inspection
  - Two-person inspection
  - Gilb inspection
  - Meetingless inspections
  - Active design reviews
  - Inspection for program correctness
  - Phased inspections
  - N-fold inspections
  - Code reading
  - Code reading with stepwise abstraction
Ch.14: Other Inspection Methods

- Two-person inspection
  - Fagan inspection simplified
  - Author-inspector pair
  - Smaller scale program

- Gilb inspection
  - Fagan inspection expanded
  - A “process brainstorming” meeting
    - root cause analysis
    - right after inspection meeting
    - aim at preventive actions/improvement
  - Checklist for preparation
  - 4-6 person teams
Ch.14: Other Inspection Methods

- Meetingless inspections
  - Importance of preparation (indiv. insp.)
    - (most defects found during preparation)
  - Empirical evidence
  - 1-on-1 instead of team meetings
    - (or other feedback mechanisms)

- Active design reviews (ADR)
  - Parnas and Weiss
  - Inspector active vs. passive
  - Author prepares questionnaires
  - More than one meeting
  - Scenario based (questionnaires)
  - 2-4 persons
Ch.14: Other Inspection Methods

- Inspection for program correctness
  - Enhanced ADR by Britcher
  - Correctness (vs. questionnaire) of:
    - topology (decomposition, hierarchy)
    - algebra (equivalence of refinements)
    - invariance (variable relations)
    - robustness (error handling)
  - Formal specifications
  - Verification technique

- Phased inspections
  - Expand Fagan inspection
  - Multiple phases/meetings
Ch.14: Other Inspection Methods

- N-fold inspections
  - Idea similar to NVP
  - N parallel inspections, 1 moderator
  - Duplications $\Rightarrow$ cost $\uparrow$

- Code reading
  - Focus on code
  - Optional meetings

- Code reading by stepwise abstraction
  - Variation to code reading
  - A formalized code reading technique
  - Top-down decomposition and bottom-up abstraction
Ch.14: Extending Inspection: Analysis

- Inspection as analysis
  - Program/document/etc. analysis
  - Inspection as statics analysis
  - Testing as dynamic analysis

- Other analyses
  - Static: algorithm, decision table, boundary value, control flow, data flow, etc.
  - Dynamic: symbolic execution, simulation, prototyping, timing, in-field execution, etc.
  - Reference: Wallace et al 1996 (NIST Special Publication 500-234) available online
Ch.15: QA and Formal Verification

- QA alternatives/activities:
  - inspection,
  - formal verification,
  - testing,
  - fault tolerance, etc.

- Formal verification as part of QA:
  - Formal methods = formal specification + formal verification
  - People intensive/focus
  - “Prove absence of fault”
Ch.15: Formal Specification: Ideas

- Formal specification:
  - Correctness focus
  - Different levels of details: ⇒ traceability requirement
  - 3Cs: complete, clear, consistent
  - Two types: description & behavioral

- Descriptive formal specs:
  - Logic, notations, & language support
    - Z, VDM, etc.
  - Math functions

- Behavioral formal specs:
  FSM, Petri-Net, etc.
Ch.15: Formal Verification: Ideas

• “Testing shows the presence of errors, not their absence.” — Dijkstra

• Formal verification: proof of correctness
  ▶ Formal specs: as pre/post-conditions
  ▶ Axioms for components or functional units
  ▶ Composition (bottom-up, chaining)
  ▶ Development and verification together

• Other related approaches:
  ▶ Semi-formal verification
  ▶ Model checking
  ▶ Inspection for correctness
Ch.15: Formal Verification Basics

- Basic approaches:
  - Floyd/Hoare axiomatic
  - Dijkstra/Gries weakest precond. (WP)
  - Mills’ prog calculus/functional approach

- Basis for verification:
  - logic (axiomatic and WP)
  - mathematical function (Mills)
  - other formalisms

- Procedures/steps used:
  - bottom-up (axiomatic)
  - backward chaining (WP)
  - forward composition (Mills), etc.
Ch.15: Formal Verification: Overview

- Basic block: statements
  - block (begin/end)
  - concatenation (S1; S2)
  - conditional (if-then/if-then-else)
  - loop (while)
  - assignment

- Formal verification
  - rules for above units
  - composition
  - connectors (logical consequences)
Ch.15: Axiomatic Approach

- Floyd axioms/flowchart
  - Annotation on flowchart
  - Logical relations
  - Verification using logic

- Hoare axioms/formalization
  - Pre/Post conditions
  - Composition (bottom-up)
  - Loops and functions/parameters
  - Invariants (loops, functions)
  - Basis for many later approaches
Ch.15: Axiomatic Correctness

- **Notations**
  - statements: $S_i$
  - logical conditions: $\{P\}$ etc.
  - rules/axioms:
    \[
    \{P\}S_1\{Q\}, \quad \{Q\}S_2\{R\} \\
    \{P\}S_1; S_2\{R\}
    \]

- **Axioms:**
  - similar to example above
  - schema for assignment
  - basic statement types
  - “connectors”
  - loop invariant
  - examples in (Tian 2001)
Ch.15: Axiomatic Proofs

- Program termination
  - $P$ positive within a loop
  - $P_i > P_{i+1}$

- Sample axiomatic proof
  - simple example in Tian
  - more complex example in Zelkowitz
  - observations: many steps involved

- Arrays and functions/procedures
  - more complicated
  - key: substitution
Ch.15: WP Approach

- Dijkstra/Gries approach

- Similarity to axiomatic approach:
  - Logic based, same annotations
  - Similar units (axioms)

- Different procedures:
  - Dijkstra model: Predicate transforms
  - Gries “science of programming” book
  - Weakest preconditions (WP)
  - Applies to guarded command and non-deterministic execution
  - Start with post-condition (output)
  - Backward chaining of WPs
Ch.15: Functional Approach

- Functional approach
  - Mills’ program calculus
  - Symbolic execution
  - Code reading/chunking/cognition

- Functional approach elements
  - Mills box notation
  - basic function associated with individual statements
  - compositional rules
  - larger programs
  - forward flow/symbolic execution
  - comparison with Dijkstra’s wp
Ch.15: FM: Applications

- What can be formally verified:
  - Program code
  - Formal design, documentation, etc.
  - Protocols: timing properties
    - deadlock/starvation/etc.
  - Hardware verification
  - Distributed program verification
  - Connected to software process

- Stepwise refinement/verification
  - UNITY system
  - Design and verification together
  - Distributed protocols
Ch.15: Application in Software Safety

- Leveson approach
  - Focused verification
  - Driven by hazard analysis
  - Distributed over development phases
  - Which FM? ad hoc

- Other applications
  - Model checking based on FSMs
  - Parnas’ tabular method
  - Formal inspection based
  - Cleanroom:
    - combination with statistical testing
  - Prospect of automation
  - Yih/Tian: PSC
Ch.15: Formal Verification: Limitations

• Seven myths
  ▶ FM guarantee that software is perfect
  ▶ They work by proving correctness
  ▶ Only highly critical system benefits
  ▶ FM involve complex mathematics
  ▶ FM increase cost of development
  ▶ They are incomprehensible to client
  ▶ Nobody uses them for real projects

• Assessment and evaluation:
  ▶ Refutation/discussion in Zelkowitz
  ▶ However, some validity/quantified
  ▶ King et al paper in our reference list
  ▶ Education/experience factor
Ch.15: FV and Defect Prevention

- Defect prevention:
  - Error source elimination
  - Error blocking

- FM in defect prevention
  - FS to eliminate error sources
  - FV to verify defect absence
  - Works in combination with other defect prevention techniques

- Other defect prevention techniques
  - Education and training
  - Technology based
  - Causal and risk analysis based
Ch.16: QA and Fault Tolerance

- QA alternatives/activities:
  - inspection,
  - formal verification,
  - testing,
  - fault tolerance, etc.

- Fault tolerance as part of QA:
  - Duplication: over time or components
  - High cost, high reliability
  - Run-time/dynamic focus
  - FT design and implementation
  - Techniques: recovery block, NVP, etc.
  - Complementary to other QA activities
Ch.16: Fault Tolerance

- General idea
  - Local faults not lead to system failure
  - Duplication/redundancy
  - redo \(\Rightarrow\) recovery block (RB) concept:
    - parallel redundancy
      \(\Rightarrow\) N version programming (NVP):

- Additional resources:
  - SMU expert: Dr. Suku Nair
    - CSE 8377: Fault Tolerant Computing
Ch.16: FT: Recovery Blocks

- General idea
  - Periodic checkpointing
  - Problem detection/acceptance test
  - Exceptions due to in/ex-ternal causes
  - Rollback (recovery)
  - Flow diagram for above

- Research/implementation issues
  - Checkpoint frequency:
    - too often: expensive checkpointing
    - too rare: expensive recovery
  - Smart/incremental checkpointing.
  - External disturbance: environment?
  - Internal faults: tolerate/correct?
Ch.16: FT: NVP

• NVP: N-Version Programming

• General idea
  ▶ Multiple *independent* versions
  ▶ Dynamic voting/decision rule
  ▶ Correction/recovery?
    – p-out-of-n reliability
    – in conjunction with RB
    – dynamic vs. off-line correction

• Research/implementation issues
  ▶ How to ensure independence?
  ▶ Support environment:
    – concurrent execution
    – voting/decision algorithms
Ch.16: FT/NVP: Ensure Independence

- Ways to ensure independence:
  - People diversity:
    - type, background, training, teams, etc.
  - Process variations
  - Technology: methods/tools/PL/etc.
  - End result/product:
    - design diversity: high potential
    - implementation diversity: limited

- Ways to ensure design diversity:
  - People/teams
  - Algorithm/language/data structure
  - Software development methods
  - Tools and environments
  - Testing methods and tools (!)
  - Formal/near-formal specifications
Ch.16: FT/NVP: Development Process

• Programming team independence
  ▶ Assumption: P-team independence
    ⇒ version independence
  ▶ Maximize P-team isolation/independence
  ▶ Mandatory rules (DOs & DON’Ts)
  ▶ Controlled communication (see below)

• Use of coordination team
  ▶ 1 C-team – n P-teams
  ▶ Communication via C-team
    – not P-team to P-team
    – protocols and overhead cost
  ▶ Special training for C-team

• NVP-specific process modifications
Ch.16: FT/NVP: Development Phases

- Pre-process training/organization

- Requirement/specification phases:
  - NVP process planning
  - Goals, constraints, and possibilities
  - Diversity as part of requirement
    - relation to and trade-off with others
    - achievable goals under constraints
  - Diversity specification
  - Fault detection/recovery algorithm?

- Design and coding phases:
  enforce NVP-process/rules/protocols
Ch.16: FT/NVP: Development Phases

- Testing phases:
  - Cross-checking by different versions — free oracle!
  - Focus on fault detection/removal
  - Focus on individual versions

- Evaluation/acceptance phases:
  - How N-versions work together?
  - Evidence of diversity/independence?
  - NVP system reliability/dependability?
  - Modeling/simulation/experiments

- Operational phase:
  - Monitoring and quality assurance
  - NVP-process for modification also
Ch.16: FT and Safety

- Extending FT idea for safety:
  - FT: tolerate fault
  - Extend: tolerate failure
  - Safety: accident free

- Safety related concepts:
  - Safety: accident free
  - Accident: failure w/ severe consequences
  - Hazard: precondition to accident

- Safety assurance:
  - Hazard elimination/reduction/control
  - Damage control
Ch.17: Summary and Comparison

- Applicability, effectiveness, and cost

- Inspection:
  - Good throughout dev. process
  - Works on many software artifacts
  - Conceptual/static faults
  - High fault density situations:
    - non-blocking
    - experience → efficiency ↑
  - Human intensive, varied cost

- Fault tolerance:
  - Dynamic problems (must be rare)
  - High cost & reliability (low defect)
  - Technique problems (independent NVP?)
  - Process/technology intensive
Ch.17: Summary and Comparison

• Testing:
  ▶ Important link in dev. process
  ▶ Activities spilt over to other phases
    – OP development, test preparation, etc.
    – (partial) code exist before testing
  ▶ Dynamic/run-time/interaction problems
  ▶ Medium/low defect situations
  ▶ Techniques and tools
  ▶ Coverage vs. reliability focus
  ▶ Cost: moderate

• Formal verification:
  ▶ Positive confirmation/correctness.
  ▶ On design/code with formal spec.
  ▶ Low/no defect situations
  ▶ Practicality: high cost → benefit?
  ▶ Human intensive, rigorous training
    (therefore, high up-front cost)

Jeff Tian, Wiley-IEEE/CS 2005
Ch.17: Comparison: Inspection vs. Others

- Inspection vs. preventive actions:
  - Often coupled with causal analysis
  - Together drive preventive actions
  - Key difference: error vs fault focus

- Inspection vs. formal verification
  - \( \approx \) formalized inspection
  - Focus: people vs. mathematical/logical
  - Applicability to design/code only?
  - Existence of formal specifications?
  - Tradeoff: formality vs. cost
  - Training and acceptability issues
Ch.17: Comparison: Inspection vs. Others

- Inspection vs. testing:
  - Existence of the implemented product
  - Levels of quality/defects
  - Static vs. dynamic defects
  - Localized vs. interconnected defects
  - Combined approaches:
    - phases and responsibilities
    - inspection of testing entities/processes
  - effectiveness vs. cost

- Inspection vs. fault tolerance
  - Complementary instead of competing (e.g., inspect individual versions)
  - Static vs. dynamic
  - Inspection of FT techniques/mechanisms
Ch.17: Other Comparisons

• Similar to inspection vs. others
  ▶ Applicability, effectiveness, and cost
  ▶ Fig.4 of Tian SQP 2001 paper
  ▶ Expanded: SQE book, Chapter 17.

• Applicability comparison:
  ▶ objects
  ▶ product/process type & activities
  ▶ expertise required

• Effectiveness: defect type/level
  improvement potential?

• Cost: mainly s/w professionals’ time
Ch. 17: Integration

- Competing or complementary?
  - Often complementary:
    - dealing with different problems
    - work under different phase/environments
    - combined effect
    ⇒ use multiple QA alternatives together
  - Shared resource and expertise

- Fit into produce/process environment
  - Adaptation/customization often needed
  - Past experience/organizational influence

- Concerted QA effort needed.