1. Stages in a STI’s Life
   1. Undiagnosed defects (leaving here is the most important transition)
   2. Perceived possible defects
   3. Actual distance defect (“active bugs”)
   4. Triaged defects (defects judged worth fixing)
   5. Defects assigned to
      1. Different developers
      2. Temporary: need to be processed soon (PM, Testers)
         1. Need to be processed soon
   6. Defects believed Resolved/” fixed” but not closed by reporters or quality assurance team
      1. These are defects bugs that programmers believe are fixed but which have not yet been confirmed as fixed by original reporters (reported is still editable)
   7. Close defects
   8. Bug feedback
      1. Bugs caused by other bug fixes
   9. Defects revealed by fixing other defects
2. Bug Resolution Pipeline
   1. Bug reports often includes
      1. Duplicate defect reports
      2. Outdated defects
      3. Defects reports based on incorrect understanding
      4. Unsupported platform, off-based feature expectations
   2. Active bugs
      1. Defects judged too risky to fix
      2. Defects that are too minor to merit cost to fix
      3. Defects that don’t have time to fix
3. Common defect tracking attributes
   1. Title
   2. Priority
      1. Must
      2. Will
      3. Desirable, but improbable
      4. Extremely improbable
   3. Severity
      1. Data loss or security issue
      2. Major functionality doesn’t work as specified
      3. Minor functionality doesn’t work as specified
   4. Reproduction formula
   5. Person assigned to
   6. Area of the project
   7. Person opened by
   8. Status
      1. Active
      2. Fixed
      3. Resolved
      4. Closed
   9. Resolved as (fixed postponed, duplicate, won’t be fixed)
   10. Type (regular/regression)
   11. Triage (whether triaged, if so accepted/rejected)
4. Creating a good bug report
   1. Structured exploration
   2. Reproduction (test several times)
   3. Isolate (controlled tests on behavior)
   4. Generalize cases (figure out underlying rules)
   5. Compare with other versions, features
   6. Summarize bug
   7. Condense bug report
   8. Disambiguate
   9. Neutralize
   10. Review
5. Reproducing Bugs
   1. Make copy of any data that could be destroyed/modified before test
   2. If conducting exploratory testing, jot down what doing (roughly)
      1. Even external thing
      2. Allows easier reproduction when bug emerges
   3. Containerization (e.g. via docker) is a key potential asset in realizing reproducibility)
6. Reproducibly and restoration
   1. Prepare
   2. Prepare test procedure
   3. Execute test procedure
   4. Evaluate response
   5. Restore test environment 🡪 back to prepare
   6. Restore (containerization is an asset here)
7. Problems with reproduction
   1. Different dev and test environment (use containerization!)
   2. Race conditions
   3. Forgotten external details
   4. Did something unintended and forgot
   5. Certain amount or fragmentation of heap
   6. Only occurs 1st time that runs
   7. Cascading failure
   8. Time dependence
   9. Resource dependence
   10. Machine disrupted
   11. Delay of underlying outcome
   12. Hardware
8. Test reports for informing decision making
   1. Traceability
      1. Principle of traceability: Interlinked documents
         1. Within software engineering projects, we deal with a large number of disparate artifacts/processes
            1. Deign components
            2. Requirements documentation
            3. Test documentation
            4. Code
            5. Test cases
         2. These items are related to one another in structured ways
            1. Elements of a design may be needed due to certain requirements
            2. Certain test exercise particular requirements, or particular components of a design
9. Effects of change
   1. Frequently, the elements of the project evolve from what was originally intended
      1. Recorded user requirements may be changed due to changes in user needs, correction by the user
      2. Derived requirements may change due to changes in technology
      3. Design decisions are changed as tradeoffs become more apparent
      4. Testing focus evolves as sense of risk change
   2. These changes trigger “ripple-through” effects on project
      1. E.g. requirements change 🡺 design change 🡺test changes, code changes
   3. People are far less likely to make good use of documentation if they feel it is outdated
10. Traceability: reasoning and buffering from change
    1. Traceability helps identify gaps (e.g. test gaps, design capturing)
    2. Traceability helps buffer the project from change
       1. When a requirement change, we can go and quickly identify the part of the design, the tests, or the code that are affected
       2. When a design component changes, we can rapidly locate the affected test and code
    3. Disadvantages
       1. Book keeping overhead of keeping track of info
       2. Risk that oversights or failure to update may lead to overconfidence in traceability completeness, and overlooking the needed change