PART 2:
Getting Going (chapter 10)

- Gradual adoption
- Current practice is changed little in each step.
- First step: use coverage.
- If coverage is low: forget about coverage while you improve testing.

Improve test specifications

- Introduce test requirements? Not yet. First gradually improve your tests
  - enlist chance and variety
    - don't test features independently
    - choose unforced values randomly
  - Use the design checklists (appendix F)
  - Move towards test requirements

Improve test requirements

- Not too many test requirements to start
- Use user manual. If subsystem has a user interface, use the test requirements catalog's parsing and syntax section. Read chapter 16 first.
- Next derive test requirements from individual routines. Look at code, if no external description of each method.

Getting Good

- Become more efficient and more effective
- more efficient: write tests faster
- more effective: miss fewer clues and requirements, use requirements to produce better tests and use coverage information correctly.
- How to improve effectiveness: attention to feedback.

Getting Good

- Sources of feedback: coverage, bug reports, reflection on decisions
- Coverage: point to missed test requirements
- Bug reports: how could it have been caught?
- Reflection: 174/175

Subsystem Testing in Practice

- Chapter 12: Using more typical specifications. Clues in part one: operations, variables, preconditions, postconditions
- Real specifications o don’t contain preconditions and postconditions: they need to be discovered.
- What to do if no specification at all?

Subsystem Testing in Practice

- What to do if no specification at all?
  - Derive specification from code. Still useful.

Working with large subsystems

- Can get too many test requirements. Should not be more than 5 pages. Varies according to complexity of requirements. Choose subsets as follows.
  - Derive all test requirements from the subsystem's external interface.
  - Derive the test requirements from all internal routines.

Working with large subsystems

- Select a manageable group of test requirements. One alf to 3/4 from internal routines.
- Build test specifications. Stop when you find yourself unable to satisfy more than one or two unsatisfied requirements per test. Merge them into new group.

Working with large subsystems

- Implement tests, check coverage. Is anything obvious missed that can be included in next batch of tests?
- Repeat until all requirements are covered.
- Measure coverage of entire suite and check for missed requirements

Working with large subsystems

- The external interface requirements should be used more often than the internal requirements. They are more important.
- Routine Requirements: when generating clues from a routine, tag them with the routine's name.

Error Requirements in Larger Subsystems

- From Part 1
  - Every test satisfies exactly one ERROR test requirement.
  - Only the ERROR requirement's count is incremented. Non-ERROR requirements are never considered satisfied by an error test specification.
  - (two assumptions: error checking leads to an immediate exit. Error checking is first.)

Error Requirements in Larger Subsystems

- What if there is a lot of processing before the error is detected.
- If error handling is high risk, each error requirement should be covered more than once.

For each distinct error effect

- In a separate checklist, list each of the error requirements that can cause that error effect
- Select all non-error requirements that may be relevant
- Design error handling tests from the new checklist. Every test spec. should satisfy only one of the error reqs. and as many as possible of the others.

For each distinct error effect

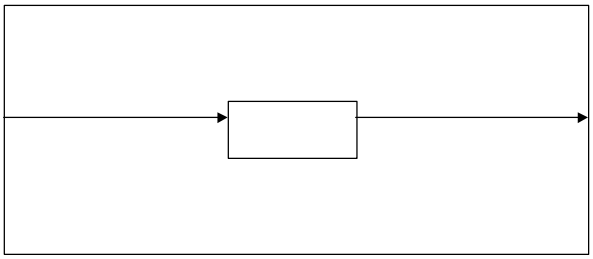
- Finish when all of the requirements (of both kinds) have been satisfied at least once.
- When error checklist is used up, the error requirements can be marked as satisfied in original checklist. Other requirements are left unchanged. Must be satisfied in non-error test

Example

- Error checking for hw 2
  - left recursion error
  - requires cycle checking code
  - There could be many errors in the cycle checking code

Dangers of large subsystems

- Routine test requirement. Sensitize; make result visible.



Options

- Test routine in isolation
  - is expensive
- Ignore test requirement
  - sometimes ok, can be dangerous if later routine is assumed to be tested.
- Use test requirement in code inspection
- Preferred: modify subsystem to make it testable. Use minidrivers