Software Development in the FDA-Regulated World: Why Test Only When We're "Done"?

Ron Morsicato
Agile Rules

Brian Shoemaker
ShoeBar Associates

Agile Rules

ShoeBar Associates

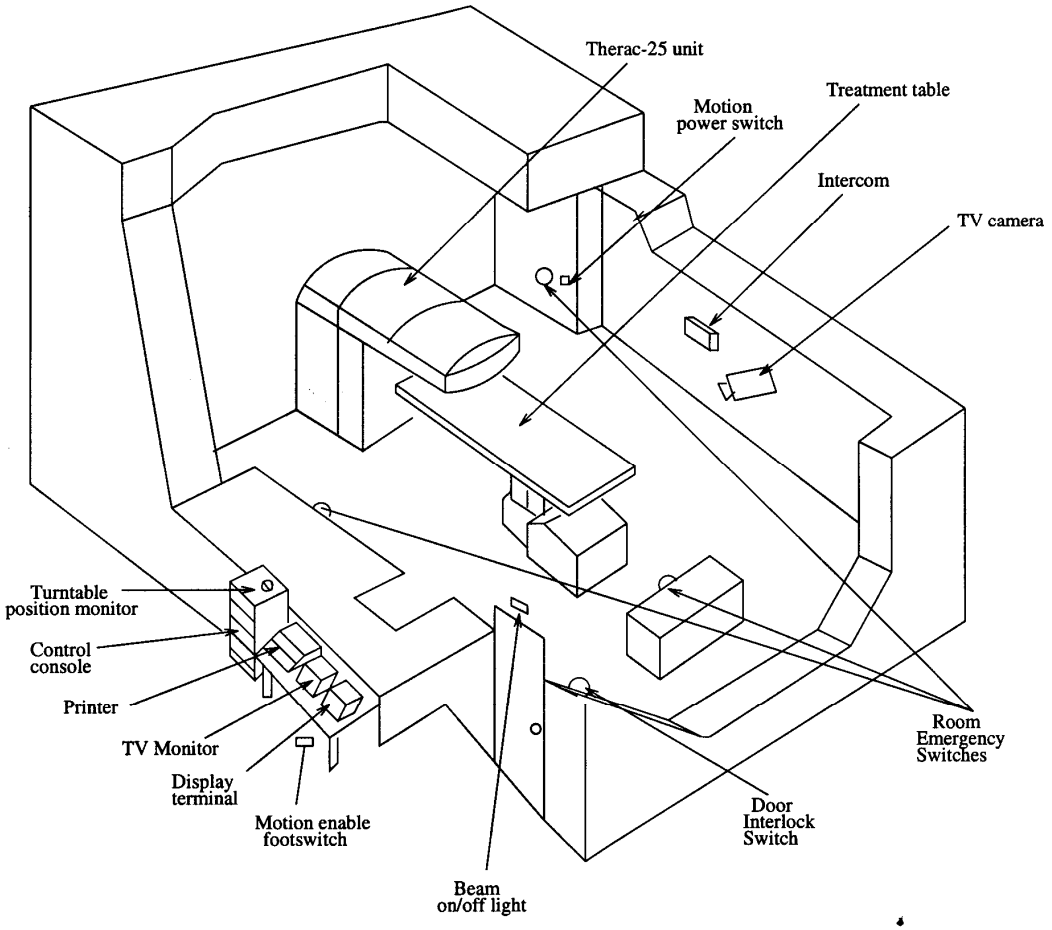
Goal of this presentation

We aim to show an approach which holds promise for software development in the FDA-regulated environment. We have provided material that gives the background for using this approach, but in order to focus on a demonstration of the approach, we may not review all the background material in detail.

Software in the FDA World: Why Test Only When We're "Done?"

- **Past history, and current bloopers, teach the need for robust software quality**
- Traditional SQA often leads to "testing squeeze"
- Hazard response: difficult unless done early
- Capturing thorough requirements is the key – but what if we don't know them all to start?
- Collaborative environment allows testing directly from requirements even as we refine them
- Payoff for the effort: validation becomes integral to development

Therac-25: The lessons are still valid

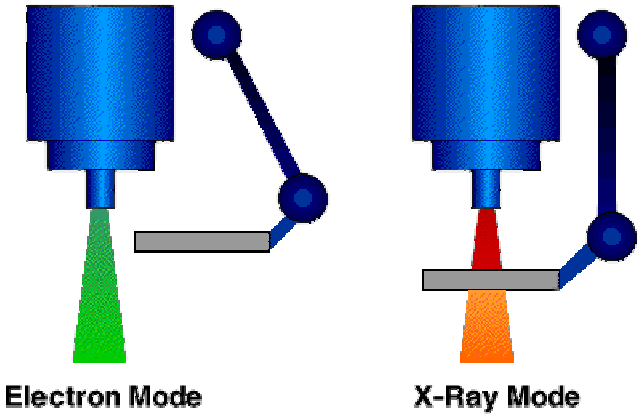


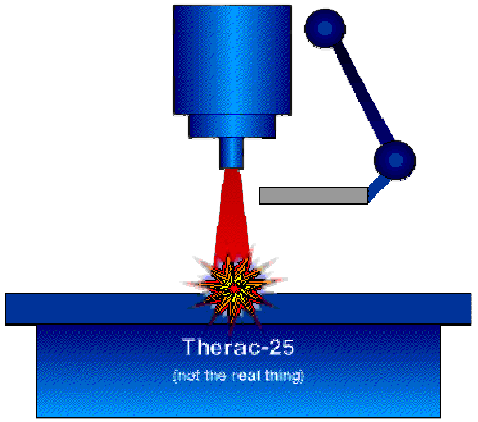
Therac-25: brief summary

- Linear accelerator system built for cancer therapy
- Instrument was further advancement of earlier models (controlled entirely through software)
- 11 Units installed in US / Canada; hundreds of patients treated (thousands of treatments)
- Mechanism: radiation beam destroys cancer tissue
  - Electron beam treats shallow tissue
  - X-rays penetrate deeper, minimal damage to overlying area
  - X-rays produced by hitting metal target with high-energy electrons
- Six overdose accidents (3 fatal): June 1985, July 1985, December 1985, March 1986, April 1986, January 1987
- Overdoses (~100x intended dose, ~20x lethal whole-body dose) traced to two specific software errors

The Therac-25 safety issue

- Single electron gun produces both modes
- In x-ray mode, electron energy must be ~100x higher (target is a good attenuator)
- Low energy + target = underdose
  High energy + no target = huge overdose





Therac-25: selected issues

System:

- Cryptic error messages; errors and malfunctions common
- Operators could, and often did, override Treatment Pause
- Did not produce audit trail that could help diagnose problems
- Relied entirely on software – removed electromechanical safety interlocks that were in previous models

Quality Practices:

- Little documentation during development; no problem tracking after release
- Minimal unit and integration testing
- QA was primarily 2700 hours of use as integrated system
- Failed to look for root causes - seized on each issue as ***the*** problem
- Treated requirements / design / directed testing as troublesome afterthought

Dangerous S/W Errors: not just history

- June 6, 2006: Ventilators recalled
  http://www.fda.gov/oc/po/firmrecalls/hamilton06_06.html
  Ventilators with older generation software, under specific conditions following oxygen cell calibration without compressed air supply, can be put in a state where no visible or audible alarms are triggered because of a software algorithm designed to suppress false positive alarms. Note that oxygen cell calibration is intended to be performed while the ventilator is connected to both air and oxygen high pressure gas sources (i.e. error occurred when not following instructed procedure).
- March 6, 2006: Dialysis device recalled
  http://www.fda.gov/cdrh/recalls/recall-081605.html
  Class I recall of dialysis device (11 injuries, 9 deaths): excessive fluid loss may result if caregiver overrides device's "incorrect weight change detected" alarm. (Device used for continuous solute and/or fluid removal in patients with acute renal failure.)
- March 15, 2005: Infusion pump fails if receives data
  Serial port on back of infusion pump allows nurse call system or hospital information system to monitor pump. ***If external system sends data to pump***, however, failure condition 16:336 may result; if this occurs during infusion, pump must be powered off and restarted.

FDA ***is*** interested in your SQA process!

Software developer hit with FDA warning letter

March 17, 2006 (http://www.fda.gov/foi/warning_letters/g5783d.htm)

The FDA issued a … warning letter to … Agile Radiological Technologies because [its] Radiation Analyzer (RAy) Film Dosimetry software does not meet good manufacturing practices (GMP) standards and deviates from quality system (QS) regulations.

Issues :

Corrective and Preventive Actions

- Software errors ("bugs") not used to identify existing and potential quality problems

Design Controls

- No complete procedures to control design of the device
- Design changes not documented, validated, reviewed, approved
- No Design History File (DHF) established or maintained

Production and Process Controls

- No validation of software used as part of the quality system

Management Controls

- Management failed to ensure that an effective quality system has been established and implemented
- No adequate procedures established for quality audits, nor were audits conducted

Software in the FDA World: Why Test Only When We're "Done?"

- *Past history, and current bloopers, teach the need for robust software quality*
- **Traditional SQA often leads to "testing squeeze"**
- Hazard response: difficult unless done early
- Capturing thorough requirements is the key – but what if we don't know them all to start?
- Collaborative environment allows testing directly from requirements even as we refine them
- Payoff for the effort: validation becomes integral to development

S/W Covered by Design Control

Medical device QSR (21 CFR pt 820) mandates ISO-9000 style design control activities for any system design:

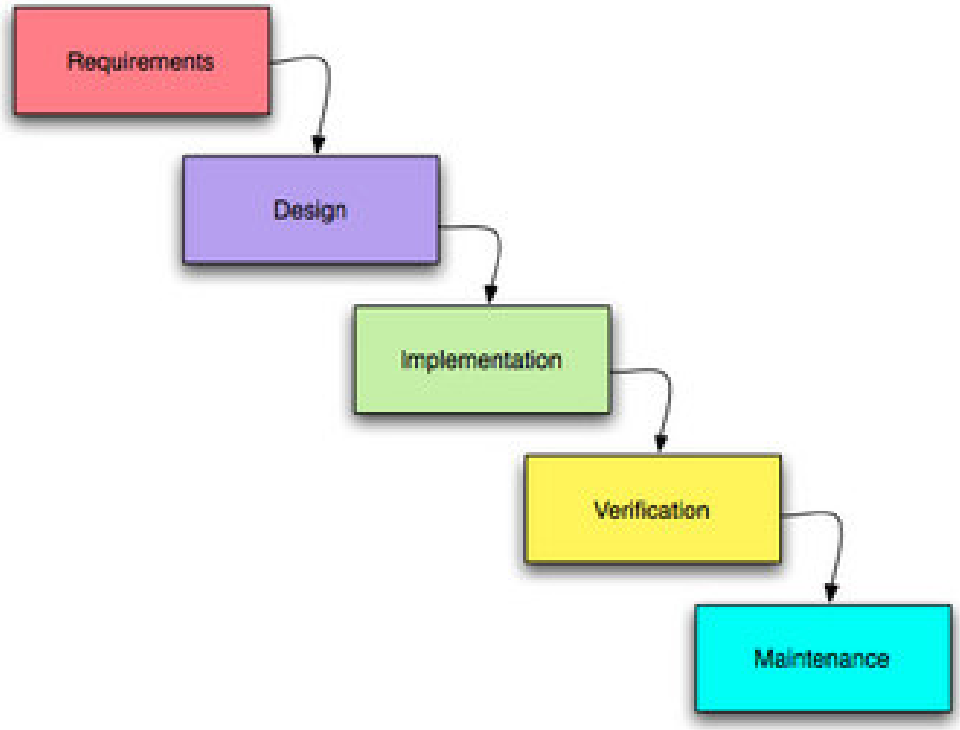
- Design / development planning
- Design input
- Design output
- Design review
- Design verification
- Design validation
- Design transfer
- Design change control
- Design History File

For software, **validation** often proves to be a struggle.

S/W Validation Expected – What is it?

- ISO definitions:
  verify = show that design outputs satisfy design inputs
  validate = show that user needs were met
- FDA *General Principles of Software Validation*:
  "confirmation by examination and provision of objective evidence that software specifications conform to user needs and intended uses, and that the particular requirements implemented through software can be consistently fulfilled"
- To validate is not just to test, but also to capture the user needs and the specifications, and to confirm by whatever means that they were met

Classical Models Assume Linear Progression – “Waterfall”



Assumes each phase is ***complete*** before starting the next!

Even with feedback, waterfall requires lots of knowledge up front

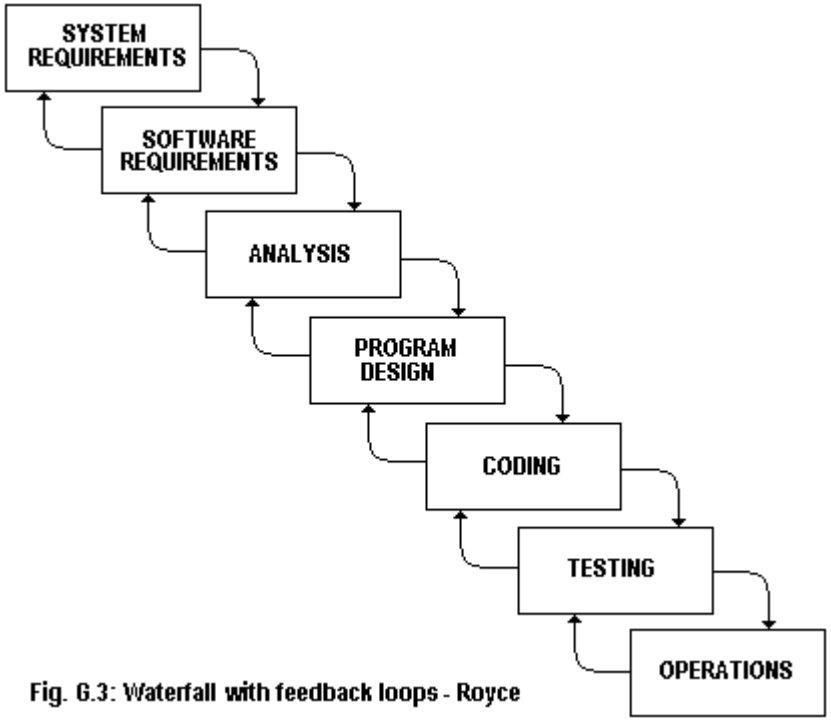
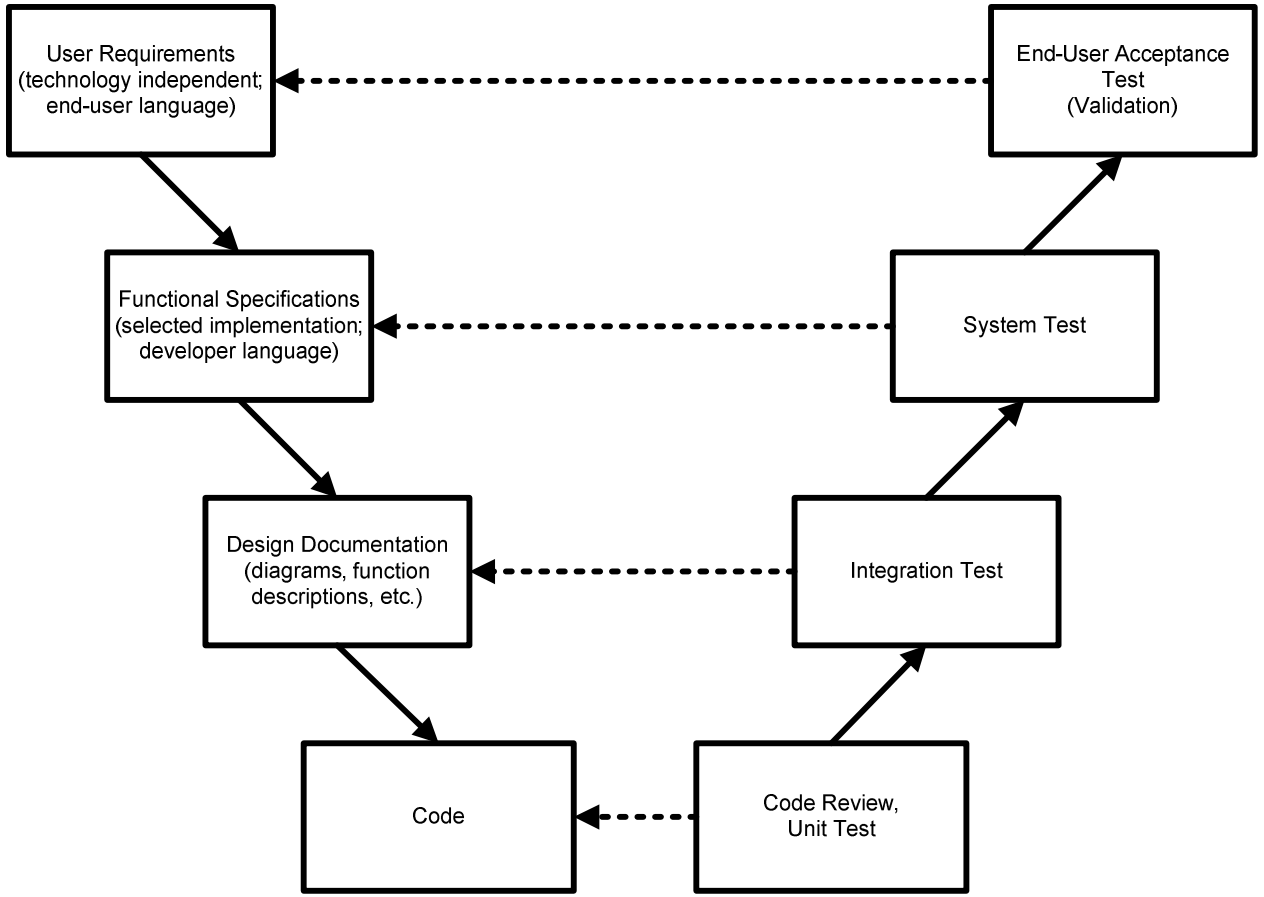


Fig. G.3: Waterfall with feedback loops - Royce

The “V” model makes little change in the order of operations



Classical Models - Notes

- All requirement gathering and design are at the beginning; all acceptance testing is late in the process
- Even where interaction / iteration are included, this is only between neighboring steps

What *really* happens in planning?

- Set the end date
- Work backward to assign times for phases
- Though phases slip, end date doesn't change
- Because testing traditionally comes at the end, time for testing is cut

This is the "testing squeeze"

Software in the FDA World: Why Test Only When We're "Done?"

- *Past history, and current bloopers, teach the need for robust software quality*
- *Traditional SQA often leads to "testing squeeze"*
- **Hazard response: difficult unless done early**
- Capturing thorough requirements is the key – but what if we don't know them all to start?
- Collaborative environment allows testing directly from requirements even as we refine them
- Payoff for the effort: validation becomes integral to development

S/W Hazards: not difficult to find

- September 2006: Infusion pump
  http://www.fda.gov/cdrh/recalls/recall-081006.html
  Touch-sensitive keypad used to program the pump sometimes registers a number twice when it has been pressed only once ("key bounce"). Thus the pump would deliver more than the intended amount of medication, leading to over-infusion and serious harm or death to the patient.
- June 2005: Glucose Meter
  http://www.fda.gov/cdrh/recalls/recall-060705.html
  If dropped or operated in battery-low condition for extended time, meter may set readout to different units.
- August 2004: Software card for programming infusion pumps
  http://www.fda.gov/cdrh/recalls/recall-082404b.html
  Software Application Card was used with an external programming device to set up parameters on specific models of implantable infusion pumps. Users could mistakenly enter a periodic bolus interval into the minutes field, rather than the hours field, resulting in deaths and injuries due to drug overdose. Numerous adverse events occurred, including two patient deaths and seven serious injuries.

S/W Hazards: not difficult to find

- June 2001: Radiation treatment planning software
  http://www-pub.iaea.org/MTCD/publications/PDF/Pub1114_scr.pdf
  Software used to calculate dose duration for radiation treatment of cancer would allow use of no more than four protective blocks (stated in the user guide). Physicians at Natl. Cancer Institute in Panama devised a way to "fool" the software into using five blocks, by entering data as if they were a single shape. If coordinates entered a specific way, the calculated dose would be as much as twice that intended. Users did not confirm calculated results; at least five patients died as a direct result of overexposure to radiation.

Hazard Analysis: Classically Up Front

- Hazards identified through systematic methods (FMEA, FMECA, or FTA)
- Hazard mitigation: changing or adding to requirements
- Requirements: supposed to be finished early in the project
- Hazards: evaluate repeatedly throughout project
- Systematic analysis: best if we know the design

Hazard Types: Often Caught in Context

- Direct failure
  Software flaw in normal, correct use of system causes or permits incorrect dosage or energy to be delivered to patient.
- Permitted misuse
  Software does not reject or prevent entry of data in a way that (a) is incorrect according to user instructions, and (b) can result in incorrect calculation or logic, and consequent life-threatening or damaging therapeutic action.
- User Complacency
  Although software or system clearly notes that users must verify results, common use leads to over-reliance on software output and failure to cross-check calculations or results.

Hazard Types: Often Caught in Context

- User Interface confusion

  Software instructions, prompts, input labels, or other information is frequently confusing or misleading, and can result in incorrect user actions with potentially harmful or fatal outcome.

- Security vulnerability

  Attack by malicious code causes device to transmit incorrect information, control therapy incorrectly, or cease operating. No examples in medical-device software known at this time, but experience in personal computers and "smart" cellular phones suggests this is a serious possibility.

Late Learned - Not Addressed?

DC-10 Aircraft:

- Cargo door latch tricky, hydraulic lines for controls under cabin floor, pressure difference if door blows open
- Issue pointed out in 1969 FMEA, again when Dutch bought aircraft in 1971; floor collapsed in 1970 static test
- M-D focused on user errors, and later on latch rather than cabin floor fail-safe
- 1972: Door blew open over Ontario; pilot landed safely only by extreme skill and foresight
- 1974: Turkish airline DC-10 cargo door blew open after takeoff from Paris; crash killed 346.

Late Learned - Not Addressed?

Immunoassay Instrument:

Late Learned - Not Addressed?

Immunoassay Instrument:

- Instrument transmits measurements to external computer (via RS232)
- Sandwich assay for hCG is known to “hook” (extremely high concentrations give signal as if low concentration)
- Test is therefore run at several dilutions – but the measurement is meaningless unless accompanied by dilution factor
- Ver. 5 software – transmitted results had concentration but not dilution factor

Software in the FDA World: Why Test Only When We're "Done?"

- *Past history, and current bloopers, teach the need for robust software quality*
- *Traditional SQA often leads to "testing squeeze"*
- *Hazard response: difficult unless done early*
- **Capturing thorough requirements is the key – but what if we don't know them all to start?**
- Collaborative environment allows testing directly from requirements even as we refine them
- Payoff for the effort: validation becomes integral to development

Requirements gathering is crucial

- Users or marketing find expressing requirements to be difficult
- Some (many?) requirements are assumed
- Language is often vague, ambiguous, or imprecise
- Sometimes, “requirements” are *too* explicit, and limit choice of technology

Test those requirements - individually

Each Requirement:

- Traceable: uniquely identifiable, tagged to all parts of the system where used
- Clear, unambiguous
- Measurable / objectively verifiable
- Completely expressed (normal and all exception situations)
- Correct and relevant to this system
- Solution neutral: what to accomplish, not how to do it
- Stakeholder value defined (includes safety implications)

Test those requirements as a whole

Requirements ensemble:

- Coherent: Definition of every essential subject matter term
- Consistent: Requirements don't contradict one another; every use of a defined term consistent with definition
- Complete: Wide enough context of the requirements; included conscious, unconscious, undreamed of requirements
- Fault tolerance, safety, security adequately addressed

We often learn requirements in context

- What other items does an element pre-require? (e.g. limited list of users – way to add users)
- Special case operations – execute quickly, or “bury” in menus?
- Input defaults – acceptable or possibly dangerous?
- What should happen if not all information is filled in?
- Where are “nonsense inputs” likely?
- Could the user interface be interpreted in an unintended way?

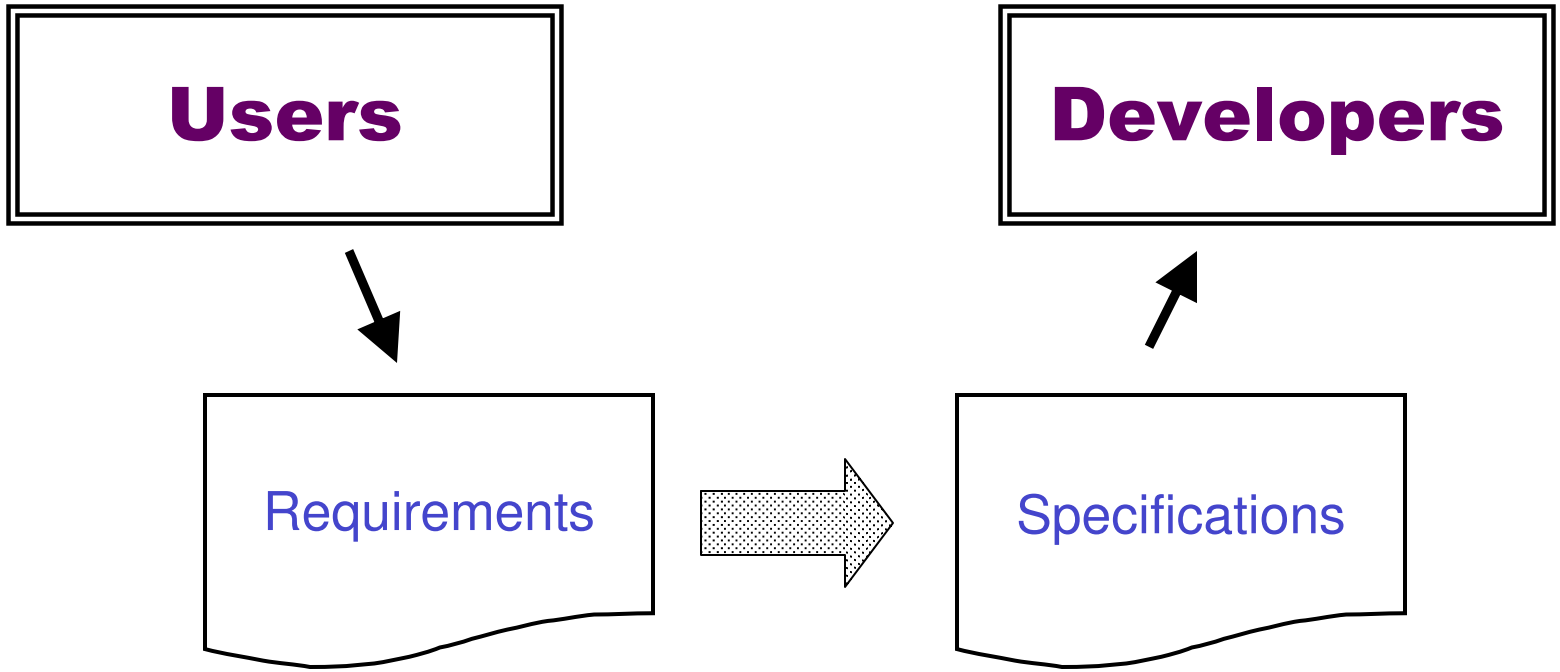
What we ought to do

- Create a learning environment
- Use it as a collaboration tool with the customer
- Continuously re-evaluate features
- Discover new requirements / specifications

Software in the FDA World: Why Test Only When We're "Done?"

- *Past history, and current bloopers, teach the need for robust software quality*
- *Traditional SQA often leads to "testing squeeze"*
- *Hazard response: difficult unless done early*
- *Capturing thorough requirements is the key – but what if we don't know them all to start?*
- **Collaborative environment allows testing directly from requirements even as we refine them**
- Payoff for the effort: validation becomes integral to development

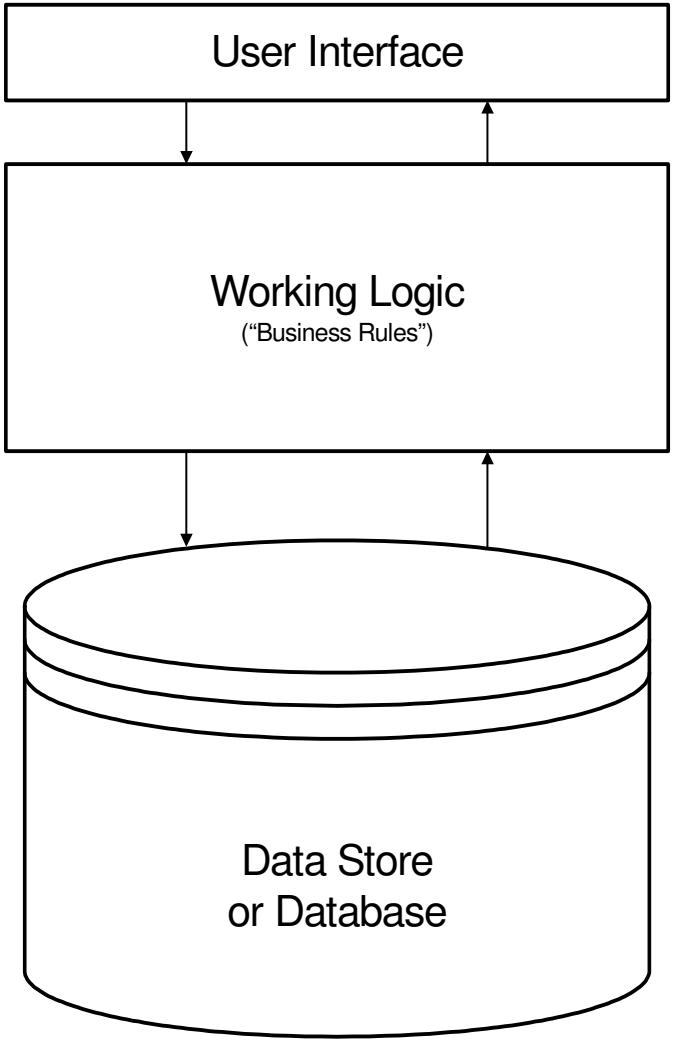
Requirements vs. Specification



Why not test directly from functional specification?

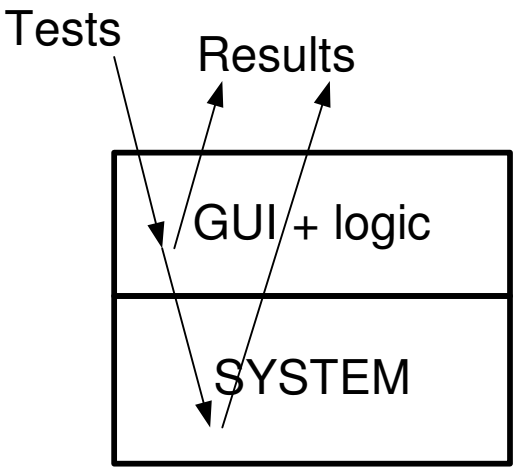
- Requirements must be testable
- We make our implementation choices
- Functional spec / use case / story, same intent: known inputs → expected output
  (We only <u>understand</u> the requirements when they're written that way!)
- To make this possible requires discipline in layered design

Layered Design: Three Pieces

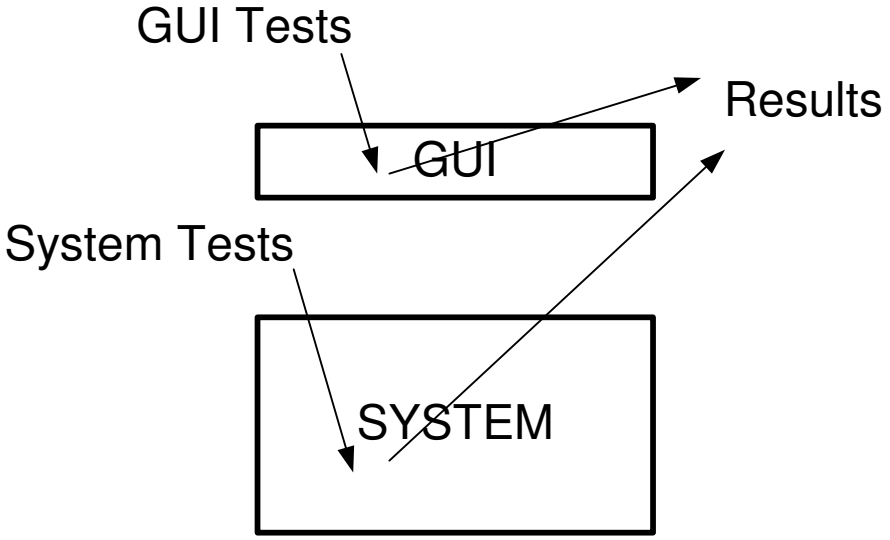


Our Goal: to test the logic directly

- Don't:



- Do:



Agile Rules

ShoeBar Associates

Environment: Write Tests, then Code

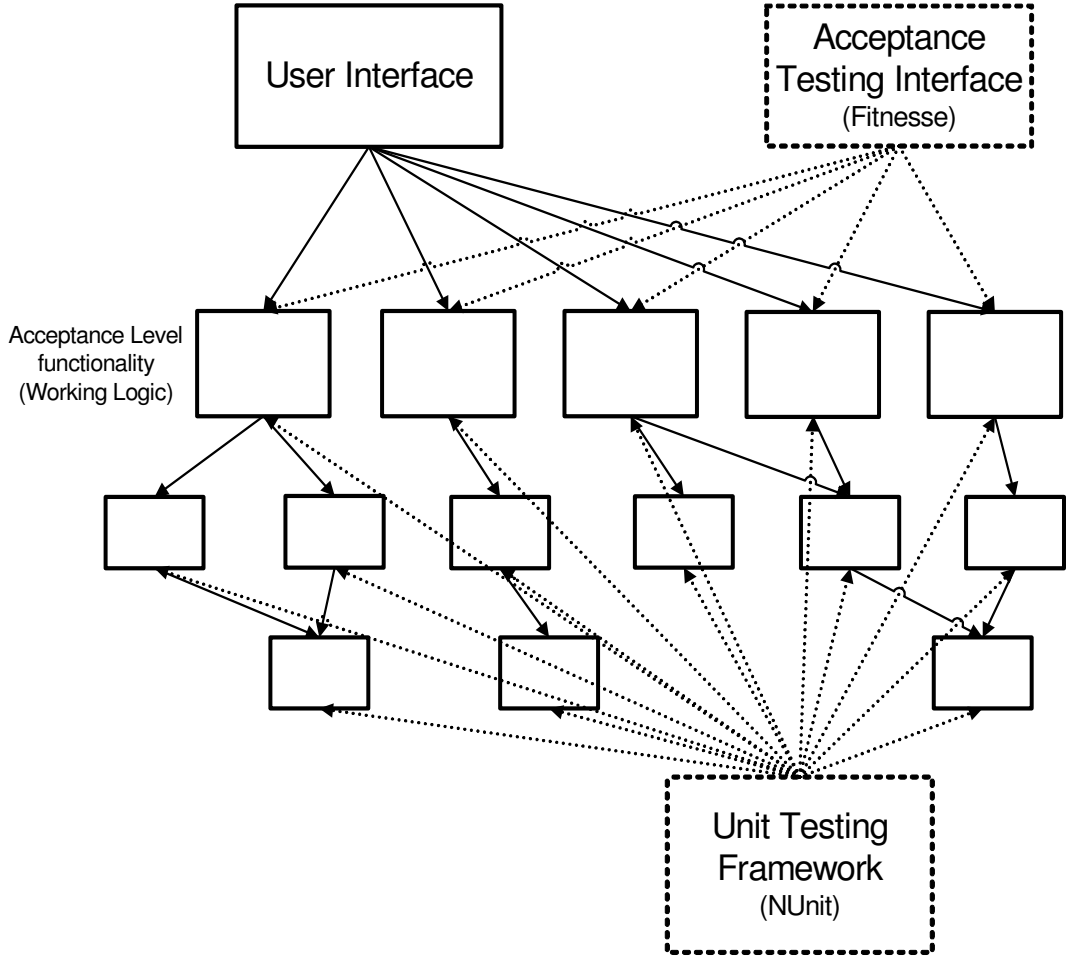
a) Develop high-level design at block-diagram level.

b) Write functional specification as executable tests.

c) Write unit tests for intended functions

d) Write code and debug until tests pass

e) Review, repeat as necessary



Every level is tested!

- Requirements must be refined to make functional specs possible
- Functional specs are elaborated; we ask questions that reveal the what-if cases
- Acceptance tests are specified to express functional specs, and corrected as requirements are revised
- Code is tested, and re-tested, with each addition or correction
- Process moves ahead *iteratively*: implement a selected group of features / functions in each cycle

Consider a practical example

- Implement key features of a gauge calibration tracking program
- Starting point: text user requirements
- We started with these use cases (“stories”):
  User log in
  Define new gauge
  Calibrate gauge
  Manage gauge status
- Working in sections, expressed each requirement in functional-specification form and created the corresponding acceptance test(s)
- Objects: users, gauges, calibration data, and history data
- Shoemaker wrote tests; Morsicato wrote code and unit tests

Sample User requirements

- The application shall allow only authorized users to open it or enter data.
- The application shall store a list of all gauges on the calibration program.
- The application shall permit defining new gauges.
- The application shall assign a unique ID to each newly-defined gauge.
- For each gauge, the application shall store the identifier(s) of one or more Master Gauges through which the subject gauge is traceable.
- The application shall prevent entering calibration data for a gauge if any of its master gauges is past due for calibration.
- Once a gauge which is due for calibration has been checked in, the application shall not permit the gauge to be checked out until calibration has been performed and has passed.
- Each time a gauge is checked in, checked out, sent out for repair, returned from repair, sent out for calibration, returned from a calibration vendor, or calibrated, the application shall record that event in the gauge's event history.

Test includes functional spec, inputs, expected results

Req 1: The system shall verify gauge data and detect if requests are made for information on non-existing gauges

Test 1.1: Test for gauges by name
Initial condition: start with Gauge1, Gauge2, Gauge3 defined, with IDs 0,1,2 respectively

AccTestCalTrack.GaugeTestFixture	
gaugeName	IsGauge?
Gauge1	true
Gauge2	true
Gauge3	true
Gauge4	false
gauge1	false
Gauge0	false
NotAGauge	false

Tests created in simple text form

```
Test capability of Calibration Tracking System:

Gauge names are case sensitive


!define COMMAND_PATTERN {%m %p}
!define TEST_RUNNER {dotnet\FitServer.exe}
!define PATH_SEPARATOR {;}

!path dotnet\*.dll
!path C:\Agile
Rules\CSharp_NUnitFitNesse_Solns\CalTrack\AccTestCalTrack\bin\Debug\AccTestCalTrack.dll

Req 1: The system shall verify gauge data and detect if requests are made for information
on non-existing gauges

Test 1.1: Test for gauges by name
Initial condition: start with Gaugel, Gauge2, Gauge3 defined, with IDs 0,1,2 respectively

!|AccTestCalTrack.GaugeTestFixture|
|gaugeName | IsGauge? |
|Gaugel | true |
|Gauge2 | true |
|Gauge3 | true |
|Gauge4 | false |
|gaugel | false |
|Gauge0 | false |
|NotAGauge | false |
```

Run test directly from this page

Req 1: The system shall verify gauge data and detect if requests are made for information on non-existing gauges

Test 1.1: Test for gauges by name
Initial condition: start with Gauge1, Gauge2, Gauge3 defined, with IDs 0,1,2 respectively

AccTestCalTrack.GaugeTestFixture	
gaugeName	IsGauge?
Gauge1	true
Gauge2	true
Gauge3	true
Gauge4	false
gauge1	false
Gauge0	false
NotAGauge	false

Cells green = expected result obtained

We readily see when actual ≠ expected

Test 2.2: Remove a gauge, check list of gauges

AccTestCalTrack.AddRemoveGaugeFixture		
gaugeName	Action	IsRemoved?
Gauge3	Remove	true
NotAGauge	Remove	false

AccTestCalTrack.GaugeTestFixture	
gaugeName	GaugeID?
Gauge1	0
Gauge2	1
Gauge3	20
NotAGauge	20
Gauge0	20
Gauge4	3
Gauge5	4 *expected* 20 *actual*

Cells green = results OK

In this case, instead of expected ID, result was a "doesn't exist" value – issue with code!

Fitnesse Demonstration

Software in the FDA World: Why Test Only When We're "Done?"

- *Past history, and current bloopers, teach the need for robust software quality*
- *Traditional SQA often leads to "testing squeeze"*
- *Hazard response: difficult unless done early*
- *Capturing thorough requirements is the key – but what if we don't know them all to start?*
- *Collaborative environment allows testing directly from requirements even as we refine them*
- **Payoff for the effort: validation becomes integral to development**

Pieces we need for this approach

- User requirements
- Close collaboration with users
- Acceptance testing framework
- Unit testing tool
- Compatible development environment
  (As shown here, the approach is running in a .NET environment, with code development in C#)

Other Points to consider

- Automation is essential, for frequent and repeated testing. (Open-source and vendor tools have allowed this integration of testing with development.)
- We do not ignore the user interface; rather, it is better challenged through other means.
- Similarly, performance will be tested by different tools.
- Electronic signature capability can be added to test-results output

Goal: not just generate code, but deliver quality product

- Translating requirements into functional specifications and tests enforces clarity, reveals implied requirements
- Traceability is built in
- Evolution of requirements / functional spec is woven into the development process
- Continual demonstration and evaluation help overcome “requirements creep” and gold plating
- Test framework permits repeating tests frequently (ongoing regression test)
- Testing remains independent: non-developers can maintain the test structure, adding / refining tests as development proceeds

Approach can enhance both quality and efficiency

- Documentation becomes truly lean, since in a displayable document we simultaneously:
  - Capture user requirements and resulting functional specification
  - Trace UR/FS to acceptance tests
  - Apply acceptance tests to show that FS is satisfied
- Requirements, functional spec, tests are all updated concurrently

Will this approach be acceptable?

- Practices similar to those shown here were found acceptable in a project reviewed against GAMP (http://www.poppendieck.com/safety.htm)
- A major medical device manufacturer regularly uses test-first approach (see references)

Satisfying FDA Expectations

FDA wants to see:	**Covered here**:
Defined development process	(*Spell out in quality docs*)
Written validation plan	(*Describe integrated method*)
Documented requirements	Built into acceptance framework
Documented functional spec	Built into acceptance framework
Documented design spec	Created at outset of project
Documented test protocols	Built into acceptance framework
Evidence of test results, reviews	(*Show review of run outputs*)
Installation protocols / test results	Run Fitnesse at end-user installation
Validation report	(*Readily created when iterations complete*)
Complete traceability	Built into acceptance framework
Sign-offs documented	Can build e-sig into test framework
Maintenance & change control	(*Spell out in quality docs*)
Changes, impact assessment	Handled in revision of acceptance framework, test re-execution
Security	(*Part of firm's IT practices*)

References

EduQuest, Inc., "FDA Auditing of Computerized Systems and Part 11," notes from course given July 2005.

FDA, General Principles of Software Validation (January 11, 2002), http://www.fda.gov/cdrh/comp/guidance/938.html

FDA, Design Control Guidance For Medical Device Manufacturers (March 11, 1997), http://www.fda.gov/cdrh/comp/designgd.html

Institute of Electrical and Electronics Engineers, "Recommended Practice for Software Requirements Specifications" (IEEE 830-1998), 01-May-1998.

Leveson, N.G., *Safeware - System safety and Computers. 1 ed*. 1995: Addison-Wesley Publishing Company Inc. [Revised version at: http://sunnyday.mit.edu/papers/therac.pdf] *(Therac-25 investigation)*

Royce, Winston W., “Managing the development of large software systems: Concepts and techniques,” in: *Proceedings, IEEE WESCON* (August 1970). *(Waterfall method)*

Robertson, James and Suzanne, *Mastering the Requirements Process*, Harlow (England), Addison-Wesley / ACM Press, 1999.

Schneider, Geri, and Jason P. Winters, *Applying Use Cases: A Practical Guide*, Harlow (England), Addison-Wesley / ACM Press, 1998.

Weyrauch, Kelly, “Safety-Critical. XP Rules.”, *Better Software*, July/August 2004.

Contact Information

Ron Morsicato
Managing Partner, Agile Rules
162 Marrett Rd., Lexington, MA 02421
978-973-1603
ronm@agilerules.com
http://www.agilerules.com

Brian Shoemaker, Ph.D.
Principal Consultant, ShoeBar Associates
199 Needham St, Dedham MA 02026
781-929-5927
bshoemaker@shoebarassoc.com