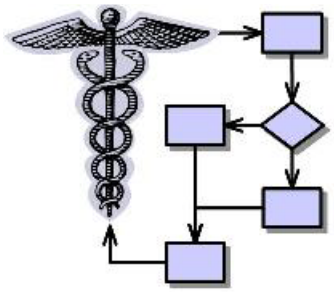
Software Validation Tools

Discovering Hidden Quality Problems

Brian Shoemaker

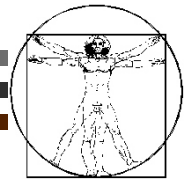
ShoeBar Associates



ShoeBar Associates

Credit

This presentation was originally developed by David James, of Precision V&V Services:



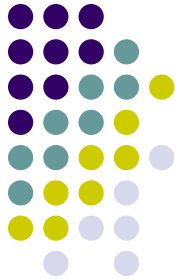
Precision V&V Services

Simplicity is the ultimate sophistication

- V&V planning, testing, and consulting
- *Specializing in Session Based Testing, manual and automated verification of medical device software*
- **David James, President:**
  - 20 years in medical device development and V&V
  - Wide variety of devices (in vitro diagnostic, software-only, therapeutic)
  - Has been leading V&V teams for 10 years
  - david.james@precisionv-v.com
    (970) 443-2402
    http://www.precisionv-v.com

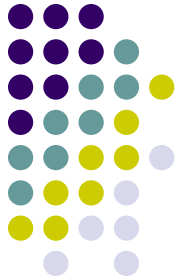
ShoeBar Associates

Discovering Hidden Quality Problems



- **Validation Economics: Find & Fix Bugs Early!**
- Classic Quality Issues Drive Up Cost
- Break the Cycle: Iterate Deliberately
- How Does Iteration Improve Quality?
- What Keys Make This Approach Work?

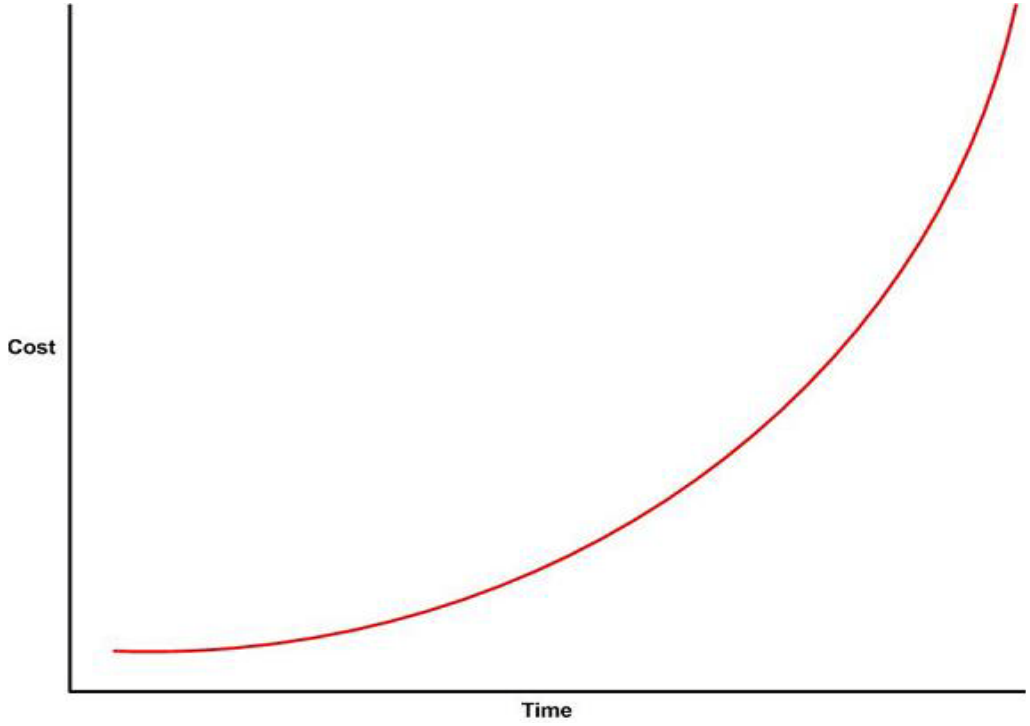
If Quality Costs, Poor Quality Costs More



- According to FDA guidance, SW validation is not just testing; it involves the entire development lifecycle
- Quality depends on many activities, not only testing (but testing is important)
- Primary goal:
  Improve quality = Remove bugs
  …*Cost-effectively*

Defect Removal Cost vs. Time

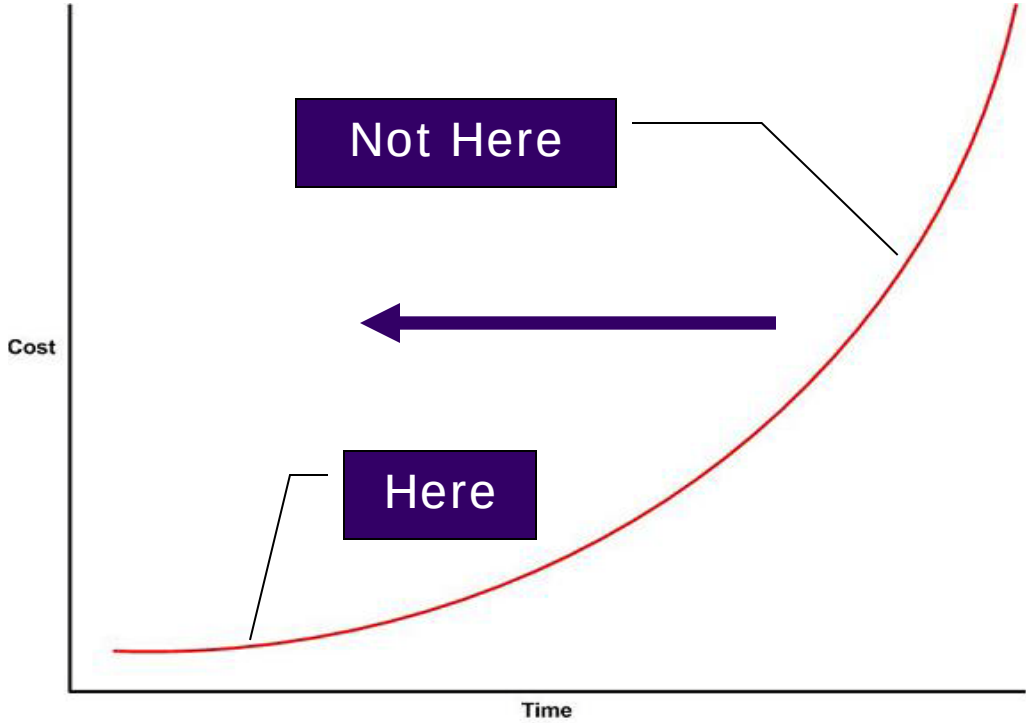
Everyone has seen some variation on this plot, the cost to fix a bug, based on when it was found:



ShoeBar Associates

Find Earlier, Fix at Lower Cost

Clearly we want to find bugs earlier in time, in order to save the most money:



ShoeBar Associates

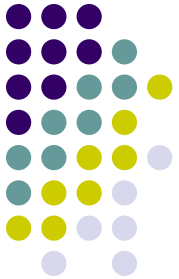
So How To Accomplish This?

If you apply certain techniques, you can simultaneously improve development efficiency, device safety, and regulatory compliance. We will focus on two areas:

- Eliminate bugs early (sooner on the curve)
- Reduce unnecessary rework (lower the curve)

ShoeBar Associates

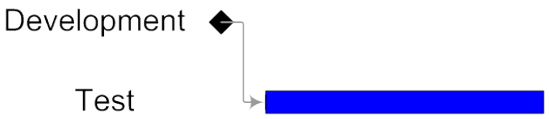
Discovering Hidden Quality Problems



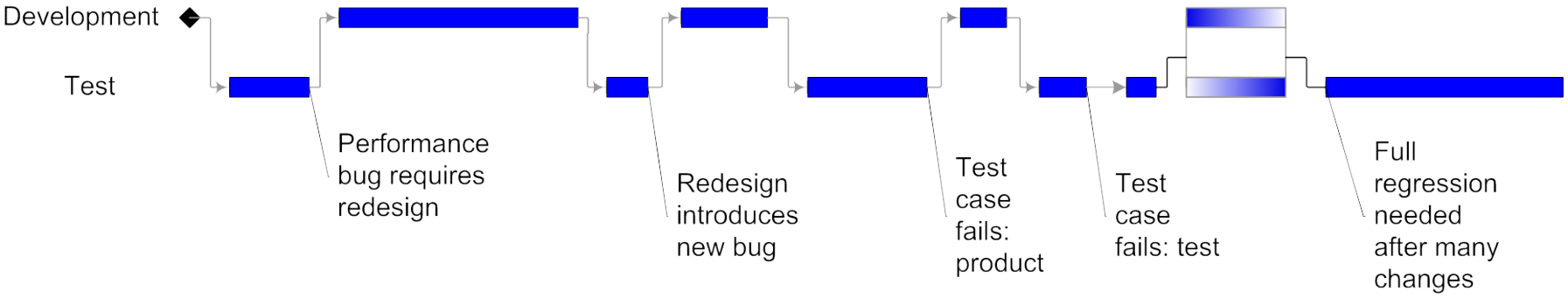
- *Validation Economics: Find & Fix Bugs Early!*
- **Classic Quality Issues Drive Up Cost**
- Break the Cycle: Iterate Deliberately
- How Does Iteration Improve Quality?
- What Keys Make This Approach Work?

Unexpected Iterations Stretch Schedule

- Ever had a release schedule planned like this



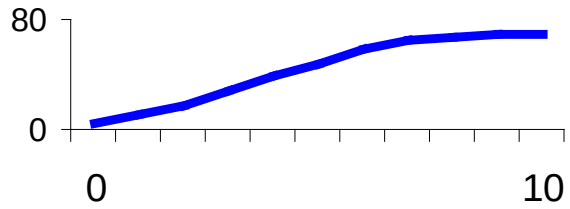
- But wind up looking like this?



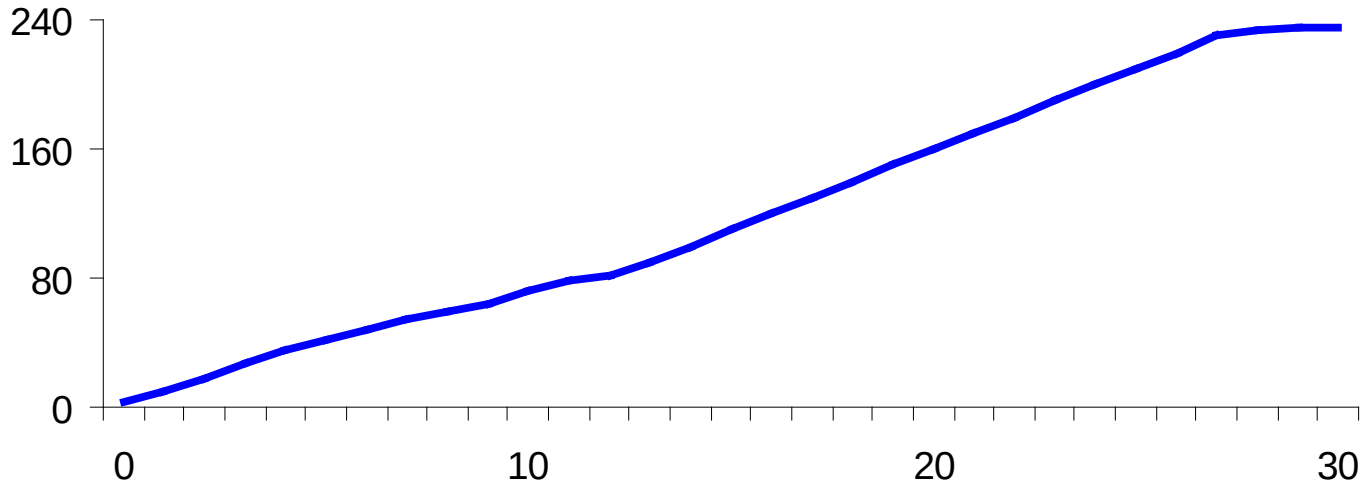
ShoeBar Associates

Unexpected Iterations Jack Up Budget

- Planned budget during testing



- Actual budget during testing



ShoeBar Associates

Big Idea: Eliminate Bugs Early

Source	Effect	Cost
Product Quality	Field Recall/Lawsuits	$5M+
Product Quality	Dev Schedule/Cost	$0.1M+
Process Quality	Dev Schedule/Cost	$0.1M+

- FDA 1996 estimate: field recall cost $5M on average
- 2005-present: Boston Scientific/Guidant recalled 109,000 implanted devices
  - $240M in civil settlement of 8000 lawsuits in 2007
  - $296M in criminal plea agreement—rejected by District Court 4/27/10
  - Untold losses in reputation, legal fees, stock price, and actual rework cost
- Baxter just recalled 200,000 Colleague infusion pumps, $400-600M
- Report: 1 in 3 medical devices containing SW recalled for SW-related problem! (Bliznakov et al.)

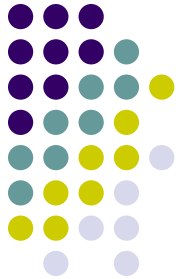
Bugs → Product Recalls

- Requirements don’t meet customer/user needs
- Design defect produces major problems
- Reliability issue arises in field
- Hazard is inadequately mitigated

Big Idea: Eliminate Bugs Early

Source	Effect	Cost
Product Quality	Field Recall/Lawsuits	$5M+
Product Quality	Dev Schedule/Cost	$0.1M+
Process Quality	Dev Schedule/Cost	$0.1M+

Bugs as Schedule Killers



- Requirements don’t meet stakeholder/customer/user needs (which can change over time)
- Device is functional but UI is not usable
- Architecture is inadequate for desired performance
- Technology is less mature or available than originally thought
- System-wide design dependencies not met
- Hazard is inadequately mitigated

Big Idea: Eliminate Bugs Early

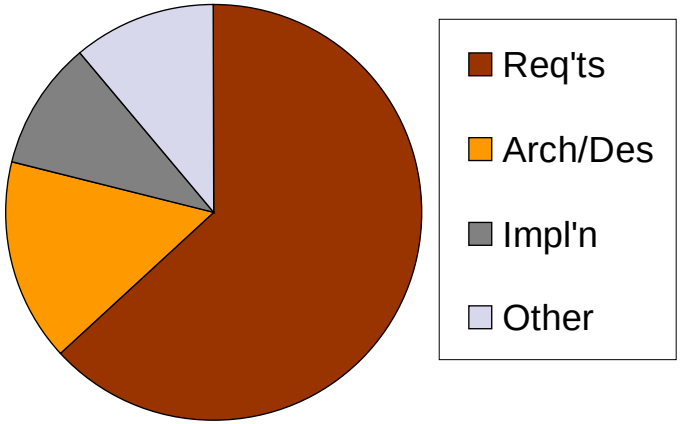
Source	Effect	Cost
Product Quality	Field Recall/Lawsuits	$5M+
Product Quality	Dev Schedule/Cost	$0.1M+
Process Quality	Dev Schedule/Cost	$0.1M+

Process-Related Schedule Killers

- Product defects found too late in the cycle
- Project estimation and budgeting process are poor
- Resources split time or are reassigned
- Configuration management breaks down
- Change-control process implemented too early
- Brittle automated tests require continuous maintenance
- Reliance on hero developers, *delivered ≠ finished*
- Integration inadequately planned or staffed
- Communication breaks down

Sources of Defects

- Requirements: 63%
- Architecture and Design: 16%
- Implementation: 10%
- Other: 11%

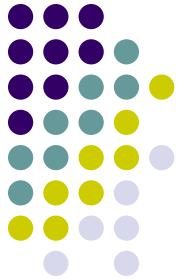


- Clearly, the code itself is only a small part of the problem. Solutions for quality improvement must address *more* than simply, “Does this code meet this requirement?”

ShoeBar Associates

Discovering Hidden Quality Problems

- *Validation Economics: Find & Fix Bugs Early!*
- *Classic Quality Issues Drive Up Cost*
- **Break the Cycle: Iterate Deliberately**
- How Does Iteration Improve Quality?
- What Keys Make This Approach Work?



Remember Our Goals

- Eliminate bugs early
- Reduce unnecessary rework

ShoeBar Associates

Suggested Solution

- Start with a “backlog” – general list of desired features
- Establish *team* ownership of all code, and *team* estimation of tasks
- Iterate, over defined time periods
- Start small: team takes on only those tasks they estimate they can complete in the time box
- Integrate continuously: make sure each iteration results in *working* code
- Test early: write and execute tests with each increment of code (*a feature isn’t done until it successfully passes its tests!*)
- Measure and track progress for all to see
- Formalize only when appropriate
- Get frequent feedback from customer / end user

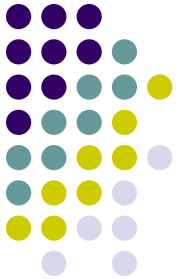
Iterate?!?

- *Didn't you say we're trying to REDUCE unnecessary rework?*

- We iterate because of unknowns.
- Users often don't know what they want until we show it to them. We must keep responding to their input.
- Engineers are inventing something new. Expect to throw away some early attempts.
- Technologies involved may not be as stable as we thought, and may need further development.
- Hazards, mitigations and failure modes affect the design and are affected by it.

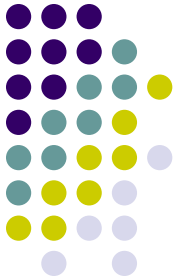
- *60% of MedDev companies are still using waterfall or stage-gate methodologies. The rest have adopted concurrent, spiral, or agile iterative methodologies.*

Backlog – the starting point



- We need to start with some target, even if it's not well defined
- Users often can't express exactly what they want, but can ask for overall features
- Team members must be able to interview users to refine the items they take in an iteration – "What would that look like?"
- The list can grow or shrink as a project goes along
- User input is vital, not only at the outset, but during development – priorities always shift, so plan for that
- User / customer may at some point decide that "enough" features have been developed and tested, to release a version

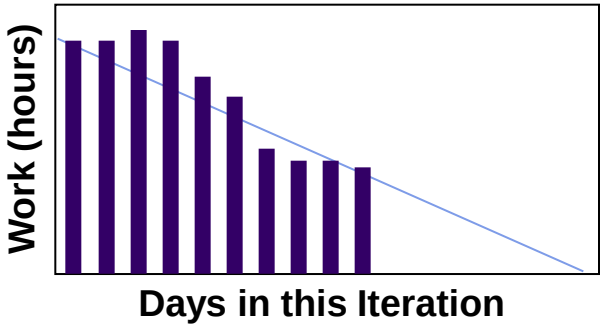
Team Ownership and Estimation



- Management's job is to ensure the team has the right members, the necessary tools, and the leeway to do their own work
- Also to avoid: prima donnas who "own" a particular function or area. Anyone can work on any section in any iteration.
- The team must learn to break down features into basic kernels (usually called "stories") – and estimate the effort to implement each story
- Everyone must be able to see what's complete, what's waiting, and where difficulties have arisen
- Everyone must also be able to see the measured progress

Total Transparency

Status reporting is not separate from team's own way of tracking their work



Each day:

- Team estimates hours remaining for each task
- All remaining hours are summed
- That total is today's data point on burn-down chart

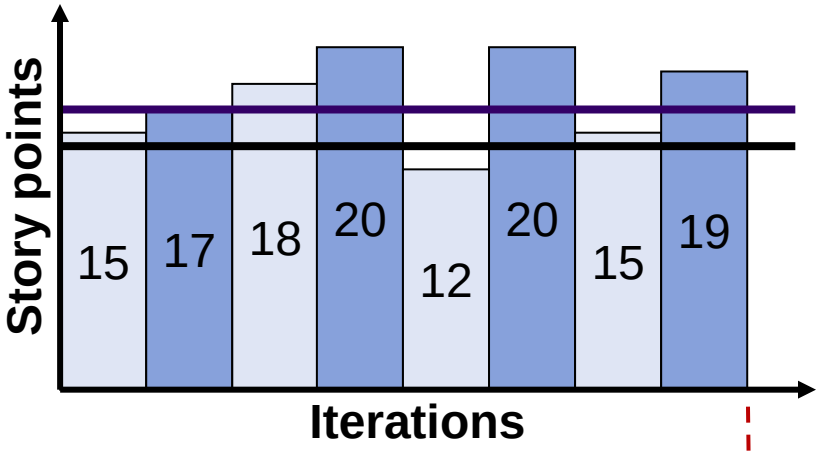
Source: Van Schooenderwoert and Shoemaker, 2010.

ShoeBar Associates

Time Boxes?

- The discipline of having a set time period forces team members to break down “stories” effectively
- The tasks taken on in a time box must be a *commitment*
- Teams must learn not to “push out” the difficult tasks, but take on limited numbers of them in each session
- After several iterations, the team’s estimates will become more accurate – allowing them to predict completion
- Use each time box to better understand:
  Previously unrecognized hazards
  True user needs (vs. the wishes they present)
  Unexpected UI confusion
- Build in time to evaluate hazards!
- *Each new iteration, respond to what was learned in the previous one/s*

Predictable Project Speed

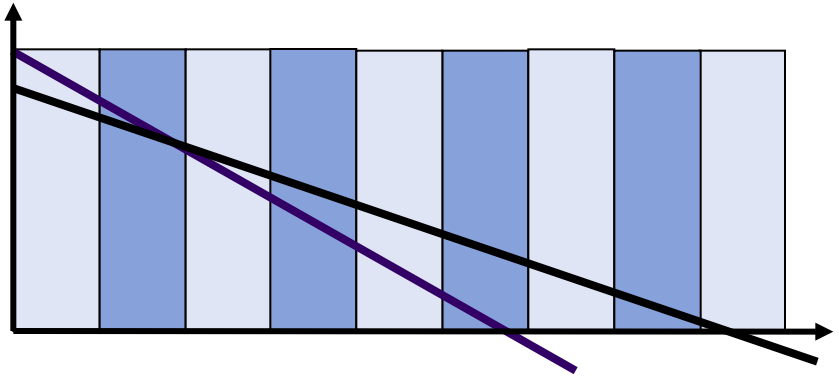


Mean (Last 8) = 17

Mean (Worst 3) = 14

Q. How long to finish project if 100 story points of work remains in product backlog?

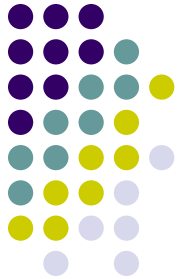
A. If it's 14 points/iter, then it takes 7.2 iterations. If it's 17 points/iter. then it will take 5.9 iterations. You can be conservative or not, as appropriate.



Source: Van Schooenderwoert and Shoemaker, 2010.

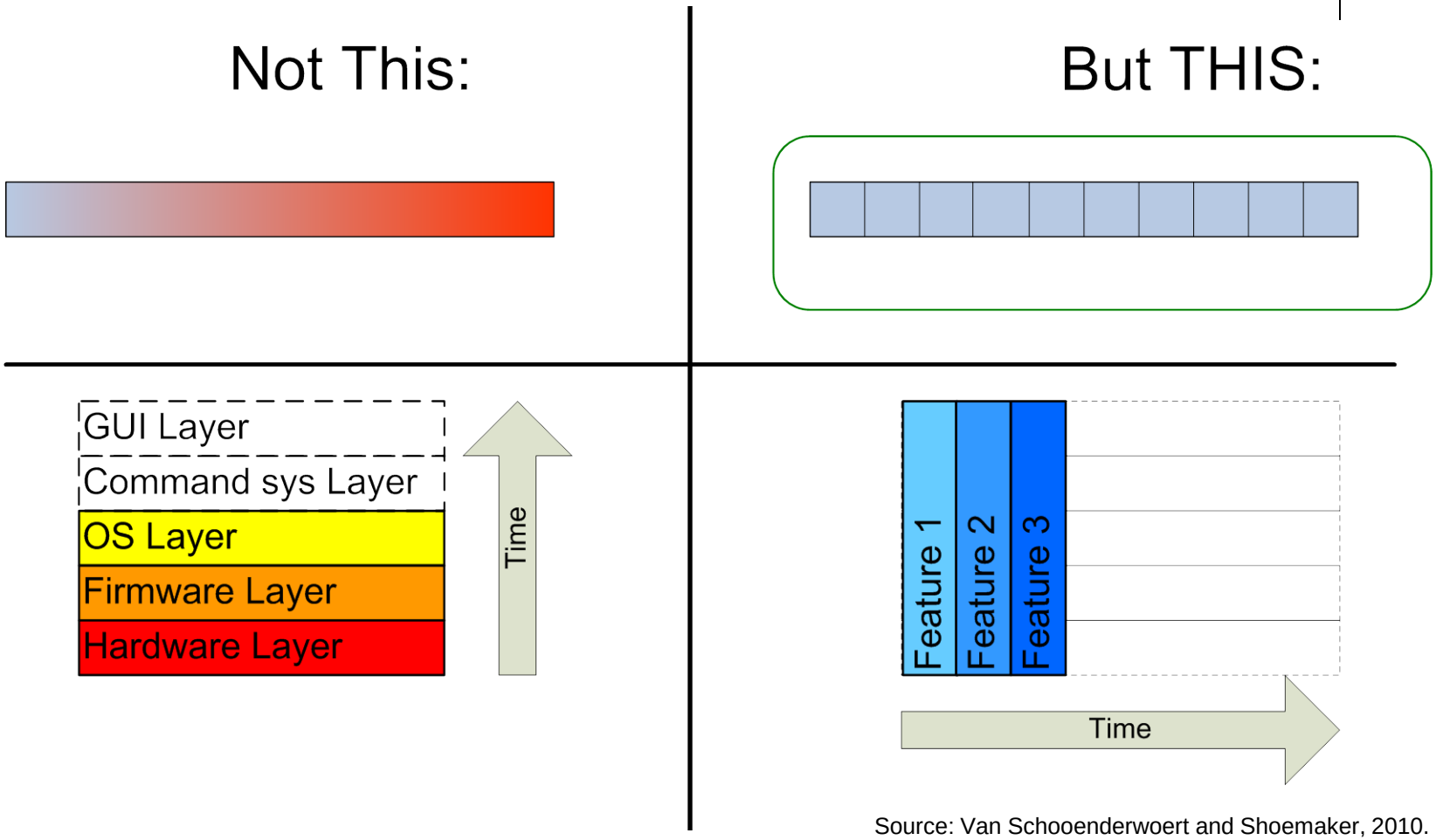
ShoeBar Associates

Integrate Continuously



- Avoid big-bang surprises
- Stub out non-working features
- Communicate testers ↔ developers about what is working and what isn't
- Test-Driven Development: create a unit test of functionality before creating the function

Deliver in Working Increments

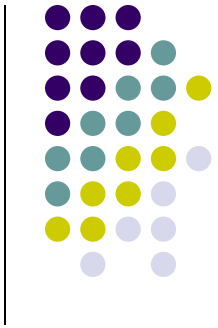


Source: Van Schooenderwoert and Shoemaker, 2010.

ShoeBar Associates

Develop Testing in Parallel

- Use unit tests to show structural integrity
- Use functional tests to check requirements and hazard mitigations
- Accumulate tests – automation framework is vital
- Leave space for exploratory / unscripted testing – to learn where to add tests
- Look at all relevant areas – interfaces, SQL, API calls, load, performance
- Involve the customer / user! Usability, missing requirements, missing / inadequate hazard mitigations
- ***A feature isn't complete until it has passed its tests.***



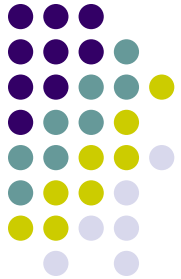
Formalize Only When Appropriate

- Requirements—fuzzy at first, sharpen when needed

  Mary and Tom Poppendieck:

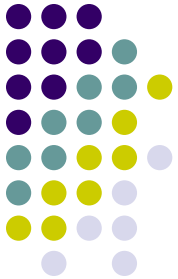
  > Concurrent software development means starting development when only partial requirements are known and developing in short iterations that provide the feedback that causes the system to emerge. Concurrent development makes it possible to **delay commitment until the last responsible moment, that is, the moment at which failing to make a decision eliminates an important alternative.**

Formalize Only When Appropriate



- Design—sketch at first, formalize at end
  - The point of design is to clarify developer intent, both for the implementation team and later maintenance.
  - How much design detail is necessary for those two groups?
  - How does the level of design detail change with the size of the team?

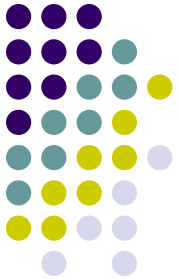
Formalize Only When Appropriate



- Develop tests early but don't spend time documenting results
- Use iterations to correct/improve earlier tests
- When final development iteration completes, test suite should also be ready – Final Acceptance can then be documented
- Actual experience: few or no surprises at final test

Experience: Vogel, 2009.

Discovering Hidden Quality Problems



- *Validation Economics: Find & Fix Bugs Early!*
- *Classic Quality Issues Drive Up Cost*
- *Break the Cycle: Iterate Deliberately*
- **How Does Iteration Improve Quality?**
- What Keys Make This Approach Work?

What Do Defect Outcomes Suggest?

Team	Defects/Function Point	
Follett Software[1]	0.0128	agile
BMC Software[1]	0.048	agile
Grain Mgmt System[2]	0.22	agile
Industry Best[3]	2.0	traditional
Industry average[3]	4.5	traditional

1 Computed from data reported in Cutter IT Journal, Vol. 9, No. 9 (Sept 2008), page 10
2 “Newbies” paper presented at Agile 2006. See “References” slide for full citation.
3 Capers Jones presentation for Boston SPIN, Oct., 2002

Source: Van Schooenderwoert and Shoemaker, 2010.

Compare Solution to Our Issues

- Defects
  - Product Quality, e.g., missing requirements, design defects, hazard mitigations
  - Process Quality, e.g., defects found too late, change control implemented too early
- Solutions
  - Start general, work toward specific
  - TEAM owns and estimates
  - Iterate over defined time
  - Integrate continuously
  - Test early, and accumulate
  - Measure and track progress
  - Formalize only when appropriate
  - Get frequent user feedback

Product Quality

Defect	Solutions
Requirements don't meet customer/user needs	Iterate, get user feedback
Design defect produces major problems	Test cumulatively, get user feedback
Reliability issue arises in field	Test cumulatively, get user feedback
Hazard is inadequately mitigated	Iterate, test cumulatively, get user feedback

Product Quality

Defect	Solutions
Device is functional but UI is not usable	Iterate, test early, get user feedback
Architecture is inadequate for desired performance	Iterate, integrate continuously, test cumulatively
Technology is less mature or available than originally thought	Iterate, test early, integrate continuously
System-wide design dependencies not met	Start small, integrate continuously, test cumulatively

Process Quality

Defect	Solutions
Product defects found too late in the cycle	Iterate, integrate continuously, test cumulatively, get user feedback
Change-control process implemented too early	Start general and work toward specific; formalize when appropriate
Brittle automated tests require continuous maintenance	Team owns processes / tools, formalize when appropriate

Process Quality

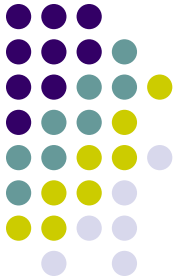
Defect	Solutions
Reliance on hero developers, *delivered ≠ finished*	TEAM owns entire product & estimates, measure & track progress, test cumulatively, integrate continuously
Integration inadequately planned or staffed	Iterate, integrate continuously

Discovering Hidden Quality Problems

- *Validation Economics: Find & Fix Bugs Early!*
- *Classic Quality Issues Drive Up Cost*
- *Break the Cycle: Iterate Deliberately*
- *How Does Iteration Improve Quality?*
- **What Keys Make This Approach Work?**

ShoeBar Associates

Know the objections & benefits



Points to counter:

- Lack of defined requirements
- Lack of structured review/release cycles
- Lack of documentation

Advantages to offer:

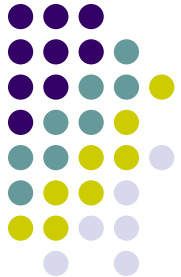
- Ability to resolve incomplete / conflicting requirements
- Ability to reprioritize requirements (mitigations) as system takes shape
- Many chances to identify hazards (controls not frozen too soon)

Source: Van Schooenderwoert and Shoemaker, 2010.

Delay Decisions

- Put off decisions and formalization until the *last responsible moment (LRM)*.
- Putting off decisions is cheaper and results in better decisions
  - Reduces rework
  - Utilizes even the most recent information
- LRM is “the moment at which failing to make a decision eliminates an important alternative.”

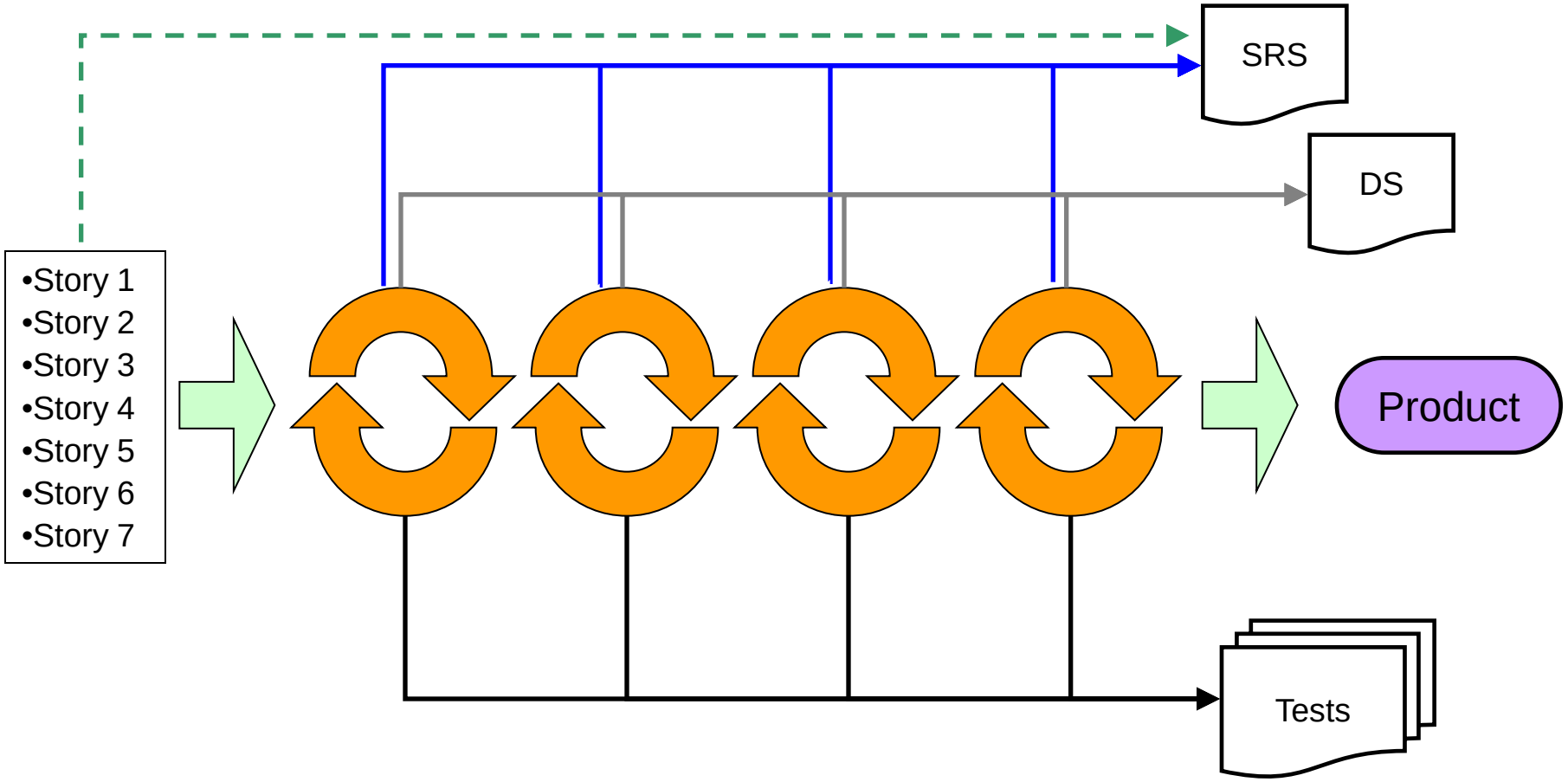
Document Effectively but Flexibly



- SOPs: focus on deliverables
  - Cover all required areas
  - Specify outputs, not strict order of completion
- Development outputs: focus on information
  - Requirements, architecture/design, hazard analysis
  - View as deliverables rather than process support
  - Consider nontraditional form if this makes information capture easier or more automatic

Source: Van Schooenderwoert and Shoemaker, 2010.

Capture knowledge as work proceeds



Source: Van Schooenderwoert and Shoemaker, 2010.

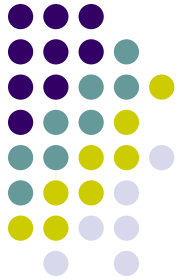
ShoeBar Associates

Use appropriate tools

- Initial user stories – may simply be index cards
- Requirements manager as they're elaborated
- Unit test harness
- Code-comment document extraction
- User-focused functional / system test engine – best if tied to requirements, e.g. FitNesse

Source: Van Schooenderwoert and Shoemaker, 2010.

Don't forget communication



- With customer / product owner – for input and ongoing feedback
  - Iter. end Demo; discussions during iteration
- Among team members – frequent but brief, to build team dynamics
  - Daily stand-up meeting; team room conversations
- With management – to show progress and build trust
  - Information radiators; iteration-end Demo

Source: Van Schooenderwoert and Shoemaker, 2010.

References

- Slide 11: FDA estimated field recall to cost $5M on average (in 1996, in support of QSR).
- Slide 11: http://www.businessweek.com/news/2010-04-27/boston-scientific-unit-s-defibrillator-plea-rejected-update1-.html
- Slide 11: Bliznakov, Zhivko, G. Mitalas and N. Pallikarakis, “Imaging the Future Medicine”, World Congress on Medical Physics and Biomedical Engineering 2006, Seoul, Korea
- Slide 17: Raimund L. Feldmann, “Software in Medical Devices: a Health Check”, IQPC 10th Conference on Software Design for Medical Devices, Philadelphia, 2008, based on a survey of software engineering technologies in medical device development, conducted by the Fraunhofer Center for Experimental Software Engineering, Maryland, in 2006.
- Slide 21: “60% waterfall/stage-gate” survey results reported in “Total Product Lifecycle Management: Lowering Costs While Increasing Quality”, by Axendia Inc. and Cambashi Inc., www.fdanews.com/total-product-lifecycle-management
- Material in slides 24, 26, 28, 34 from Van Schooenderwoert, Nancy, and B. Shoemaker, “Jump out of the Waterfall: Applying Lean Development Principles in Medical Device Software Development,” workshop presented at IQPC 13th Software Design for Medical Devices conference, San Diego CA, May 2010.

ShoeBar Associates

References

- Slides 30 and 42: Tom and Mary Poppendieck, Lean Software Development: An Agile Toolkit, 2003
- Slide 32: Vogel, M. (EMC Consulting), “What it takes to support a Validated Agile Project,” presented at 45th annual meeting of Drug Information Association, San Diego CA, June 2009.
- Slide 34: “Newbies” paper full citation: Nancy Van Schooenderwoert, “Embedded Agile Project By the Numbers with Newbies” presented at Agile 2006, Minneapolis MN, July 2006
- Slide 34: Michael Mah “How Agile Projects Measure Up, and What This Means to You”, Cutter IT Journal, Vol. 9, No. 9 (Sept 2008), for data on BMC and Follett Software projects.
- Slide 34: Capers Jones, “Software Quality in 2002” presentation for Boston SPIN, Oct 2002. See http://www.boston-spin.org/talks.html#yr2001

Acknowledgement

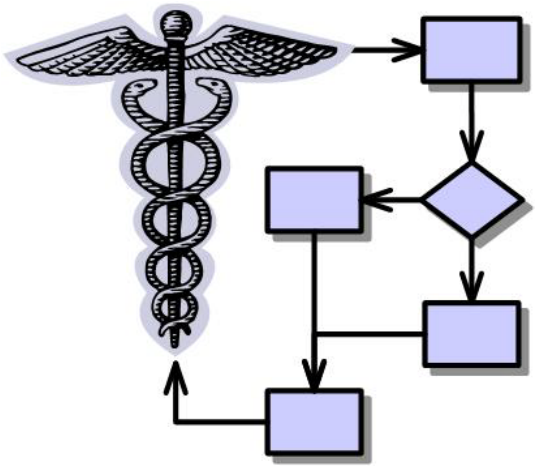
A number of these slides were developed by Nancy Van Schooenderwoert, Lean-Agile Partners Inc., and are based on her work in coaching teams in lean methods for high-quality software development.

Nancy Van Schooenderwoert
Lean-Agile Partners, Inc.
162 Marrett Rd., Lexington, MA 02421
781-860-0212
NancyV@leanagilepartners.com
http://www.leanagilepartners.com

Lean-Agile Partners

ShoeBar Associates

Questions? Comments?



Brian Shoemaker

Principal Consultant

ShoeBar Associates

781-929-5927

bshoemaker@shoebarassoc.com

ShoeBar Associates