Project Manager "Primer" for Software Testing & Quality

Bob Galen
RGalen Consulting Group, LLC
www.rgalen.com bob@rgalen.com

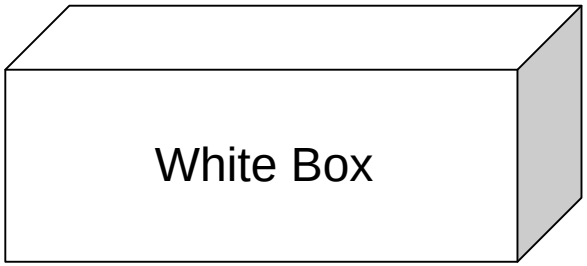
Outline

1. Different Types of Testing
2. Setting Meaningful Testing Milestones
3. Risk from a Testing Perspective
4. Overview of Risk-Based Testing
5. Managing Test Automation
6. SQA Outsourcing
7. Wrap-up

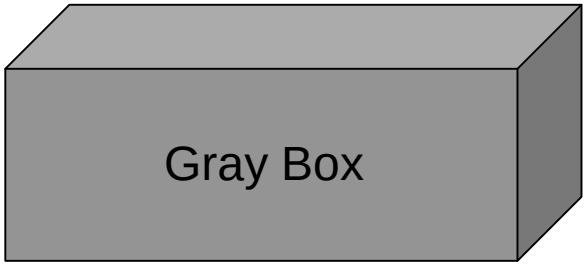
Different Types of Testing

- Black vs. White Box
- Various testing efforts
- 4 Schools of Testing

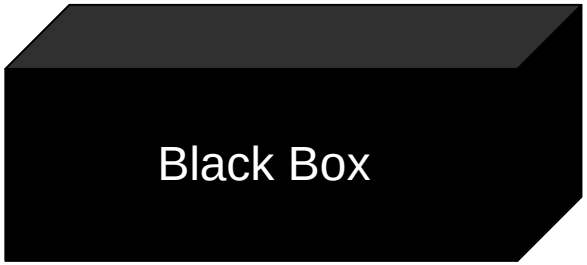
Types of Testing
The "Boxes"



- Writing tests with a strong view to the internal workings of the code.
  - Unit Tests & Component Tests



- Still knowing about the code, but also considering external requirements and functional behavior
  - Integration or API Tests



- Totally driven by requirement-driven functional behavior.
  - Functional & User Acceptance Tests

Various Types of Testing

- Developer-Driven
  - Unit
  - Automated Builds
  - Smoke Tests
  - API & Low Level integration
  - Data integrity & object interaction
- Early QA-Driven
  - Functional Testing
  - Integration Testing
  - Feature Testing
  - Agile – Customer Acceptance Testing
  - Exploratory Testing
- Later Cycle QA-Driven
  - Functional (feature-driven) Testing
  - System Testing, end-to-end testing
  - Regression Testing
  - User Acceptance Testing
  - Load & Performance Testing
  - Non-functional Requirement Testing
  - Security Testing
  - Usage scenario Testing

Non-Functional Requirements
Quality Attributes or "ilities"

- Availability
- Efficiency
- Flexibility
- Integrity
- Interoperability
- Maintainability
- Portability
- Reliability
- Reusability
- Robustness
- Testability
- Usability

- Performance
- Security

- Non-functional requirements are usually more challenging to test
  - Clarity of the requirement
  - Testing skills & effort
- Security, Performance & Load are usually added as non-functional requirements
- Availability, Reliability, Interoperability & Usability are frequently examined in modern applications
- Security is becoming more and more relevant

Context–Based 4 Schools

Analytic School	Quality School
Technique-Driven	***Process-Driven***
Sees testing as rigorous and technical with many proponents in academia	***Emphasizes process, policing developers and acting as a gatekeeper***
Factory School	**Context-Driven School**
Plan-Driven	
Sees testing as a way to measure progress with emphasis on cost and repeatable standards	***Emphasizes people, setting out to find the bugs that will be most important to stakeholders***

Context–Based
7 Basic Principles of the Context–Driven School

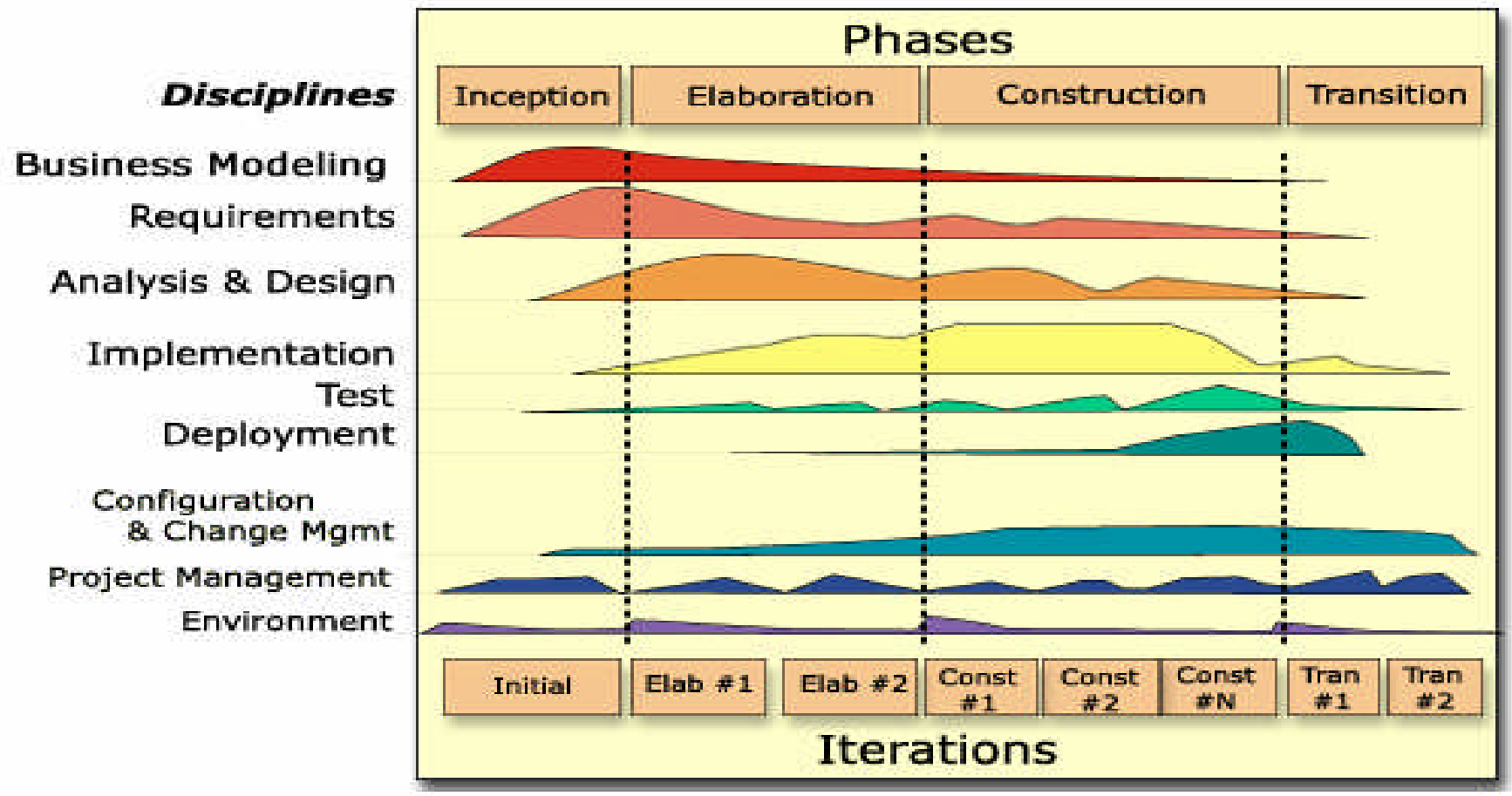
1. The value of any practice depends on its context.
2. There are good practices in context, but there are no best practices.
3. People, working together, are the most important part of any project's context.
4. Projects unfold over time in ways that are often not predictable.
5. The product is a solution. If the problem isn't solved, the product doesn't work.
6. Good software testing is a challenging intellectual process.
7. Only through judgment and skill, exercised cooperatively throughout the entire project, are we able to do the right things at the right times to effectively test our products.

http://www.context-driven-testing.com/

Setting Meaningful Testing Milestones

- Testing is iterative
- The SDLC is the primary influence point
- Key to Success: Entry & Exit Criteria
- Smoke Testing, etc.

Rational Unified Process
An "Example" of Testing Iterations



Source: http://en.wikipedia.org/wiki/Rational_Unified_Process

Methodology Implications for Testing

Model	Test Setup	Test Planning	Test Execution	Test Automation
Waterfall	Large scale, early on, dedicated equipment	Traditional System Test view	Single pass w/ limited rework	Executed but rarely developed till the next release
RUP	Enterprise scale, early on, often shared equipment	Incremental Test view	Iterative passes, moderate - heavy rework	Executed but rarely developed till the next release
Agile	Small scale, often shared environments until later iterations	TDD model, planned within development iterations	Within development iteration – unit & acceptance focused	Automated unit, smoke, and acceptance tests; minimal regression

Milestones should follow the SDLC

- Team formation
- Skills readiness
- Lab preparation
- Tools configuration
- Testing preparation
  - Planning
  - Test case design
  - Automation development
- Iterative test execution
  - Manual, Exploratory, & Automated testing
  - Progress towards maturation
  - Exit criteria

- Basic Recommendations –
  - Manage at the activity level, focused on testing iterations
  - Manage at the Test Suite level via pass / fail criteria
  - Manage defect trending – towards incremental goals
- Don't micro-manage at a test case (task) level

Coverage

- Notion of requirement traceability or coverage
  - Ability to trace test case(s) back to the originating requirement(s)
  - Feature coverage from a compliance and completeness perspective
  - If required, usually a 100% requirement. For example, FDA or SOX environments

- Notion of code coverage
  - More internally focused
  - Branch, conditional, path, statement – raw code
  - Rarely can/should achieve 100% coverage levels
  - Tools capture coverage as code is exercised via testing

Entry & Exit Criteria

- Entry Criteria
  - Conditions that must be met prior to QA beginning their testing efforts
  - Usually some sort of change log, content position
  - Smoke Testing (manual and/or automated) is a form of entry criteria – tied to execution / passing of focused testing
- Exit Criteria
  - Conditions that must be met prior to SQA completing testing on a specific deliverable
  - Normally includes coverage (test cases run, features completed)
  - Also includes quality attributes (pass rates, acceptable defect levels)

Smoke Testing

- A set of tests that are run prior to SQA “accepting” a release for testing
- Typically automated and “connected” to the build system
- Intended to prevent wasted effort by SQA on broken releases (basic operations and core features)
- Focus can / should change release over release
- Programmatic form of release criteria
- Usually defined collaboratively with and owned by the development team

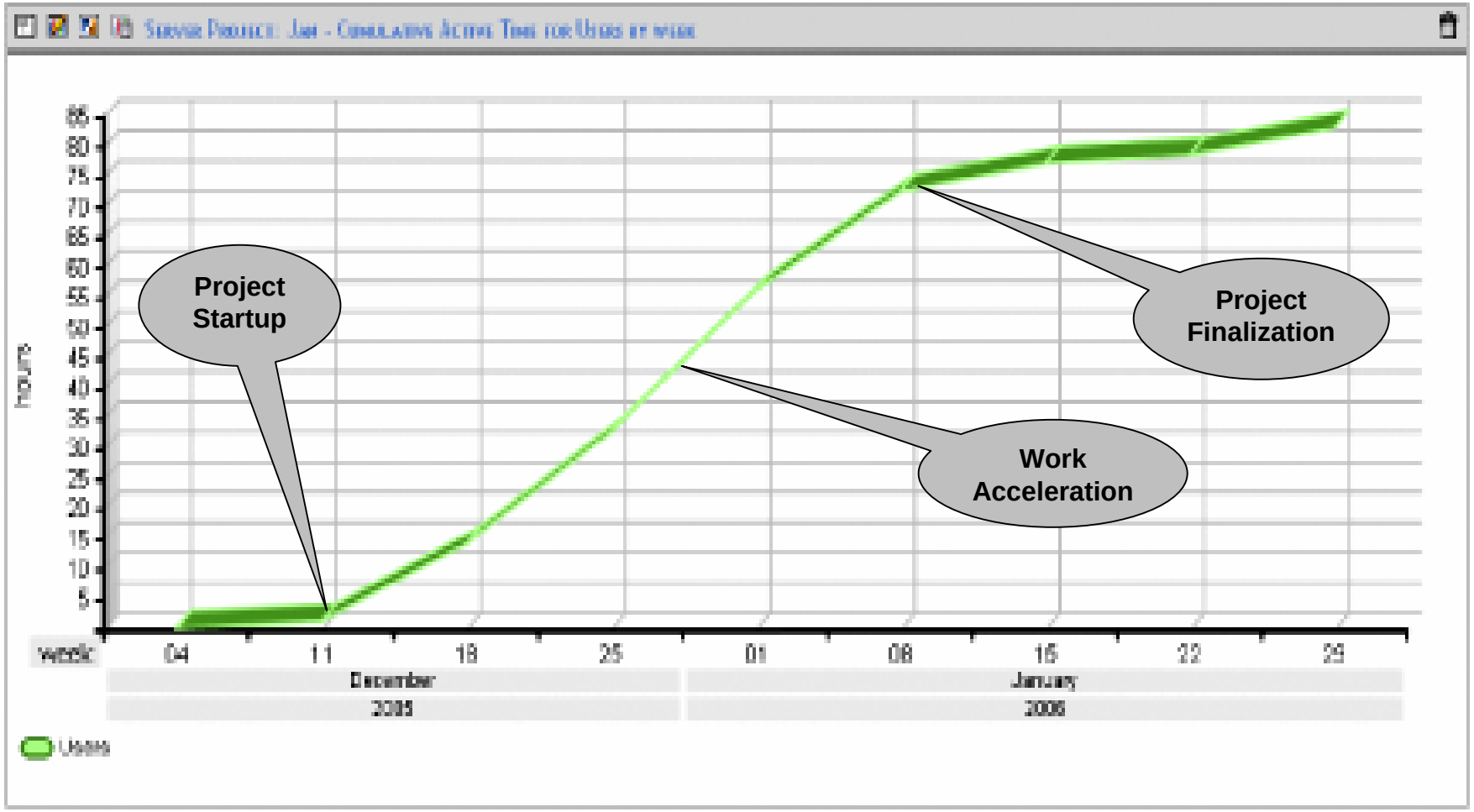
Challenges from a Testing Perspective

- Coverage, execution & blocking
- Defect density & trending
- Ratio balance

Your Ally
Historical Trending

- Since testing is essentially an iterative or cyclical activity, you can benefit by paying attention to historical trends –
  - Test case design & execution rates
  - Coverage attainment rates
  - Raw defect rates
  - Cyclical quality (maturation) rates (higher priority defects & blocking defects)
  - Regression levels
  - Observing patterns – for example deterministic S-curves and Zero Bug Bounce trends

S-Curves
Cumulative "Work" Over Time



Blocking & Adjustment

- SQA should not be forced to plan in great detail because…things change…daily!

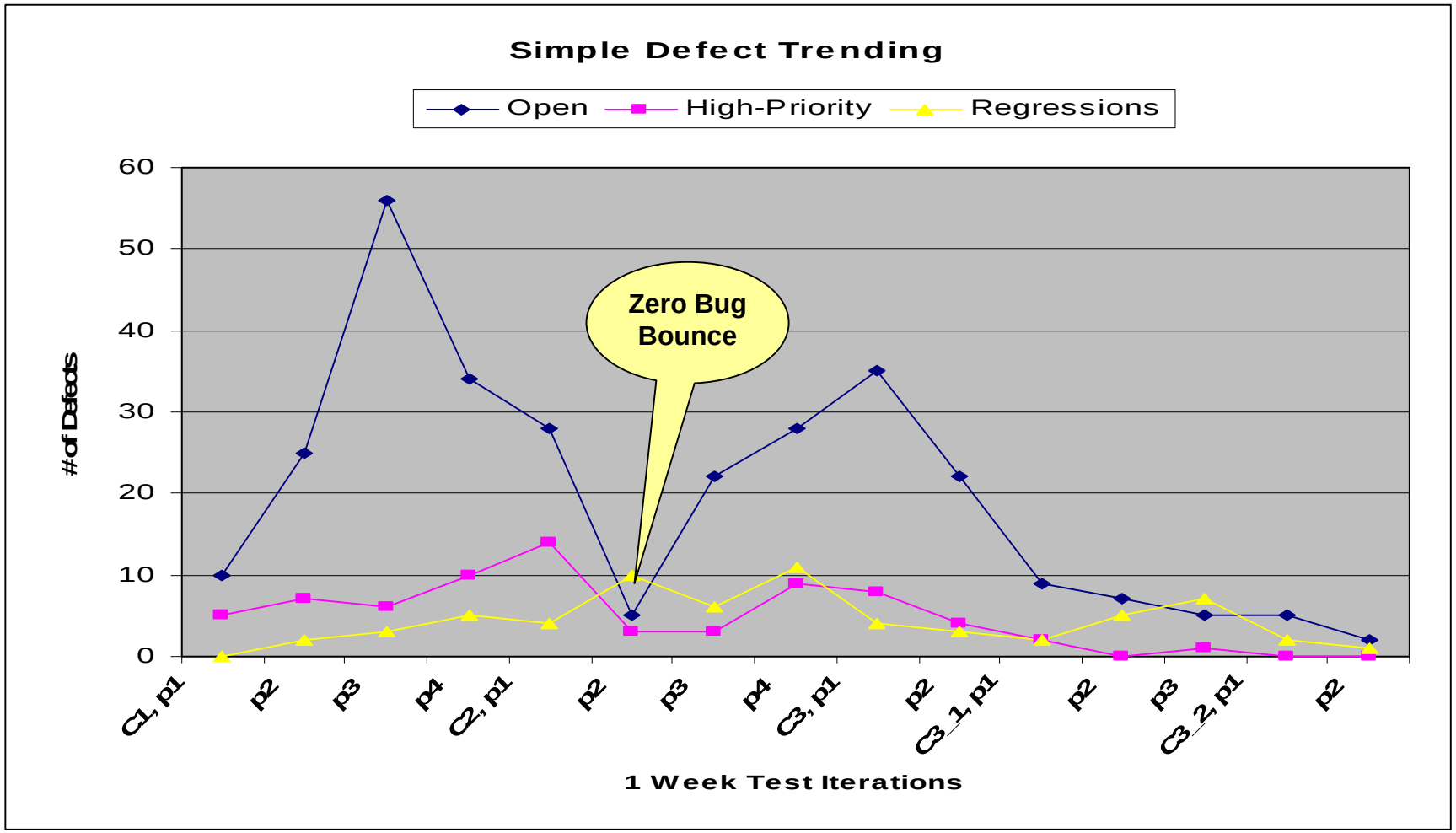
- Instead, continuously adjust based upon –
  - Blocking bugs
  - Feature immaturity
  - Blocking execution (for example: data, equipment, and other dependencies)
  - Exploration & discovery
  - Priority changes
  - Overall context shifting

  are common and expected…

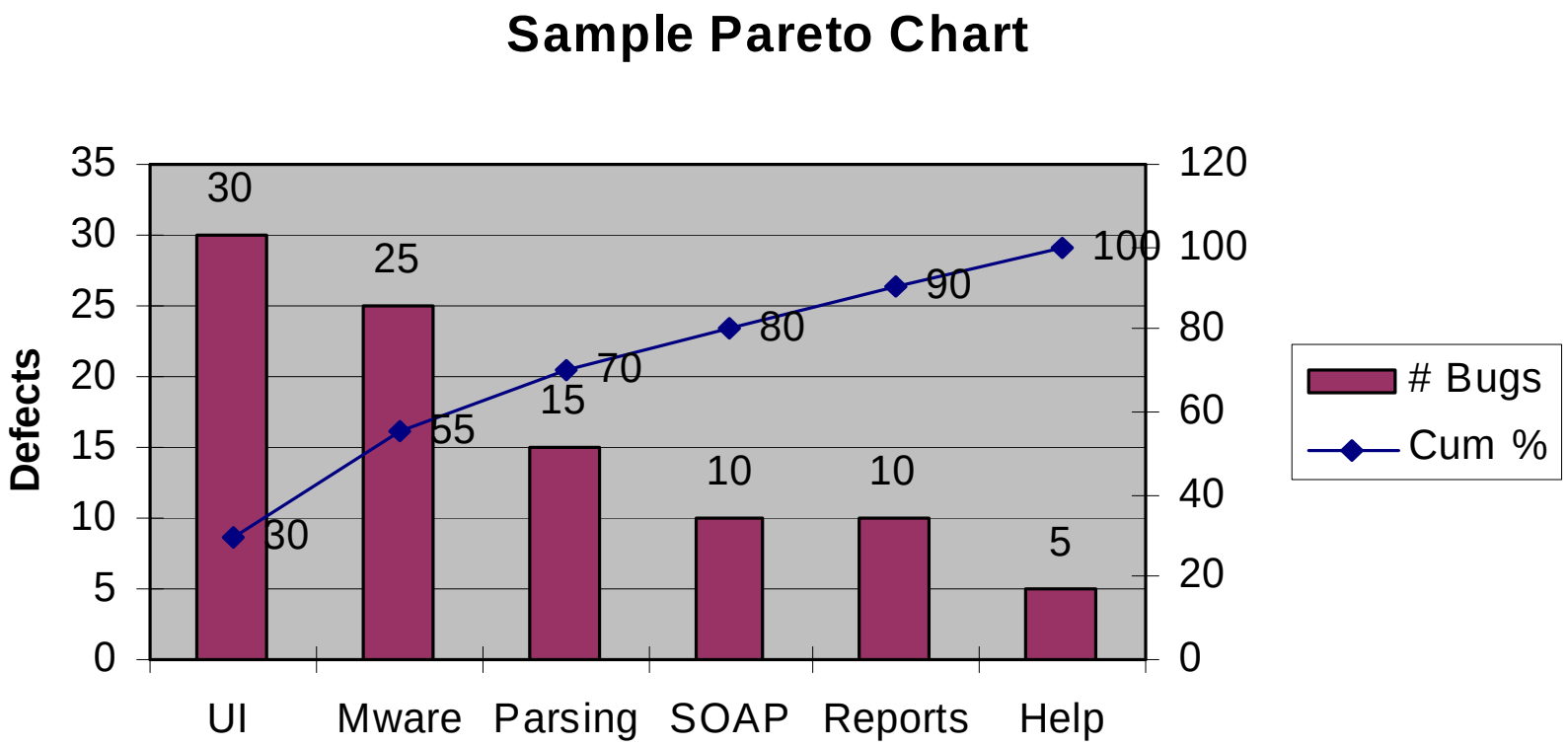
Defect Density & Trending

- You want to pay attention to defect density
  - Pareto analysis determining the 80:20 risk areas
  - Wrap risk management & focus around these areas
  - It changes, so monitor it release over release
- And other trends, comparing
  - Open vs. closed – approaching a release point?
  - High priority – how is the software maturing?
  - Regressions and rework times – at expected levels?
  - Test plan vs. actual progress (coverage)?

Open Defect Trending



Open Defects per Functional Area Trending – Pareto (80:20 Rule) Chart



Developer to Testers Ratio

It turns out that ratio's Matter!

- Developer-to-Tester
- Ensure you count ALL the developers (Producers) then factor in testers (Consumers)
- Types of applications really matter, for example:
  - Regulatory
  - Platform & component interoperability
  - Localization
- Automation has a strong impact
- Achieve a *balance* that meets your *clearly defined* quality goals!

Developer to Tester Ratio's – An "Informal" Study *29 companies responded* *R. Rice, 2000 QAI conference*
Maximum ratio: 1 developer to 0 testers
Minimum ratio: 30 developers to 1 tester
Most common ratio: 3 developers to 1 tester
Average ratio: 7 developers to 1 tester
Median ratio: 5 developers to 1 tester

Some local examples of developer to tester ratios M. Eason, 2004 Local RTP, NC study
Large statistical software company = 1:1
Huge software conglomerate = 1:1
Healthcare IT firm = 4:1
Analytical CRM company = 4:1
Technology consulting firm = 8:1

Overview of Risk-Based Testing

- Realization of coverage
- Comparing methods
- Exploratory testing

Risk–Based Testing Background

- It starts with the realization that you can't test everything – ever!

  100% coverage being a long held myth in software development

- There are essentially 5 steps in most of the models
  1. Decompose the application under test into areas of focus
  2. Analyze the risk associated with individual areas – technical, quality, business, schedule
  3. Assign a risk level to each component
  4. Plan test execution, based on your SDLC, to maximize risk coverage
  5. Reassess risk at the end of each testing cycle and adjust

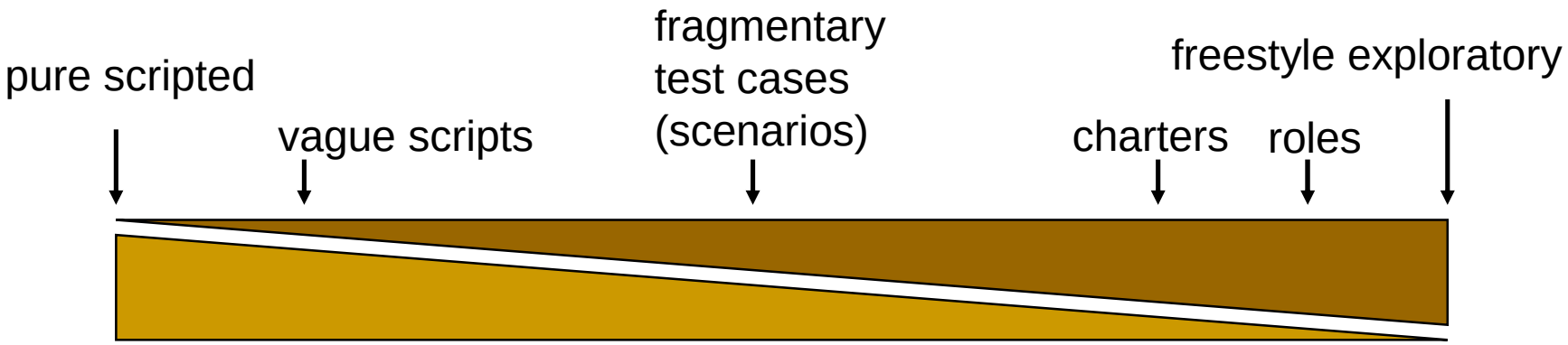
Risk–Based Testing
Risk Prioritization

- Traditional - Collaborative Risk Planning Workshop
  - Include stakeholders and interested parties – BA, Architects, Developers, Testers, PM's, Stakeholders, Management, etc.
  - Prepare by reading product requirement and other artifacts
  - Test team leads discussion for each suite – default intentions, concerns or issues, questions
  - Q&A
  - Constituents feedback their views to –
    - Business Importance, Probability of Instability, Overall Complexity, Coverage & Timing
  - Test team constructs a working view towards
    - Individual suite risk handling
    - Overall project risk handling

Risk–Based Testing
Test Suite – Execution Plan

Test Suites	# of Test Cases	Average Size	Average Complexity	Opening - Priority	Middle Game Priority	End Game Priority	Design Time	Setup Time	Execution Time		Initial Release	C1	C2	C3	Final Release
Accept - Smoke	25	S	M	High	High	Med	4	1	2		3	2	2	1	2
Func - Database Meta-data Integrity	45	M	H	Low	Med	High	5	3	3		4	2	2	3	3
Func - Mware, Business rules	75	L	H	Low	High	High	10	2	5		3	5	5	5	5
Func - Real-time data	30	M	M	High	Low	High	5	1	2		3	1	1	2	2
Func - Intelligent Searching	45	L	M	High	High	High	5	5	3		8	3	3	3	3
Func - Area 3	25	S	T	Med	Med	Med	5	1	2		2	1	1	1	2
Func - Area 4	40	S	T	Med	Med	Med	5	1	2		2	1	1	1	2
Func - Area 5	45	S	T	Med	Med	Med	5	1	2		2	1	1	1	2
Func - Common UI Features	150	S	T	Med	Med	High	15	2	10		7	5	5	10	10
Comp - Operating systems	30	S	T	Low	Low	High	2	3	3		4	1	1	3	3
Comp - Browsers & databases	130	S	M	Low	Low	High	3	10	5		11	1	1	5	5
Perf - 5 sources, 5 user scenarios	25	L	H	Low	Med	High	15	3	5		4	3	3	5	5
Defect Verifications	N/A	N/A	N/A	Low	High	Low	5	1	5		2	5	5	1	5
Regression	N/A	N/A	N/A	High	Low	High	0	1	15		16	4	4	15	15
Automation	N/A	N/A	N/A	Low	Low	Low	10	1	5		2	1	1	1	5
Total Test Cases	**665.0**					**Totals:**	**94**	**36**	**69**						
Average / time per test case	**0.30**								**Total Time**		**74**	**35**	**35**	**58**	**69**
Test team members	**3.5**								**Team/Person Days**		**21**	**10**	**10**	**16**	**20**

Exploratory Testing
Scripted vs. Exploratory Continuum



To know where a test falls on this scale, ask yourself: *"to what extent am I in control of the test, and from where did the idea originate?"*

Exploratory Testing
Session Strategy

- Exploratory Testing proceeds in a series of interconnected 60-120 minute sessions that are focused on a specific testing project (application)
- Planning the project encompasses establishing a set of time-boxed session charters and defined roles
- Establishing roles and focus areas for the sessions or groups of sessions
- Establishing the session execution dynamics
  - Starting, Stopping, Re-Chartering, Reporting (logging)
- Reporting progress to stakeholders & re-establishing the overall test strategy / charter

Risk–Based Testing
Risk Scheduling & Tracking

- Once you have your overall risk assessment and cyclical feedback, you need to create a plan & schedule that reflects the tempo and cyclical testing requirements of your SDLC

- Iterative or agile methodologies require more testing cycles
  - They also increase the complexity of your planning to sensibly handle rework (re-testing, regression, integration, and repair verifications)
  - Ensure you don't over-test, by testing too soon or too often

Risk–Based Testing Methods Comparison

Types of Context-Based Testing	Risk Based	Exploratory Testing	Just-In-Time Testing
Management & Tracking level	N/A	N/A	Test Usage Scenarios
	Test Suites	Charter-driven sessions	Ideas “chunked” into Test Objectives
Team Collaboration & Execution	Test Cases	Test Ideas, heuristics-driven testing	Test Ideas
	Test Steps	N/A	N/A

Managing Test Automation

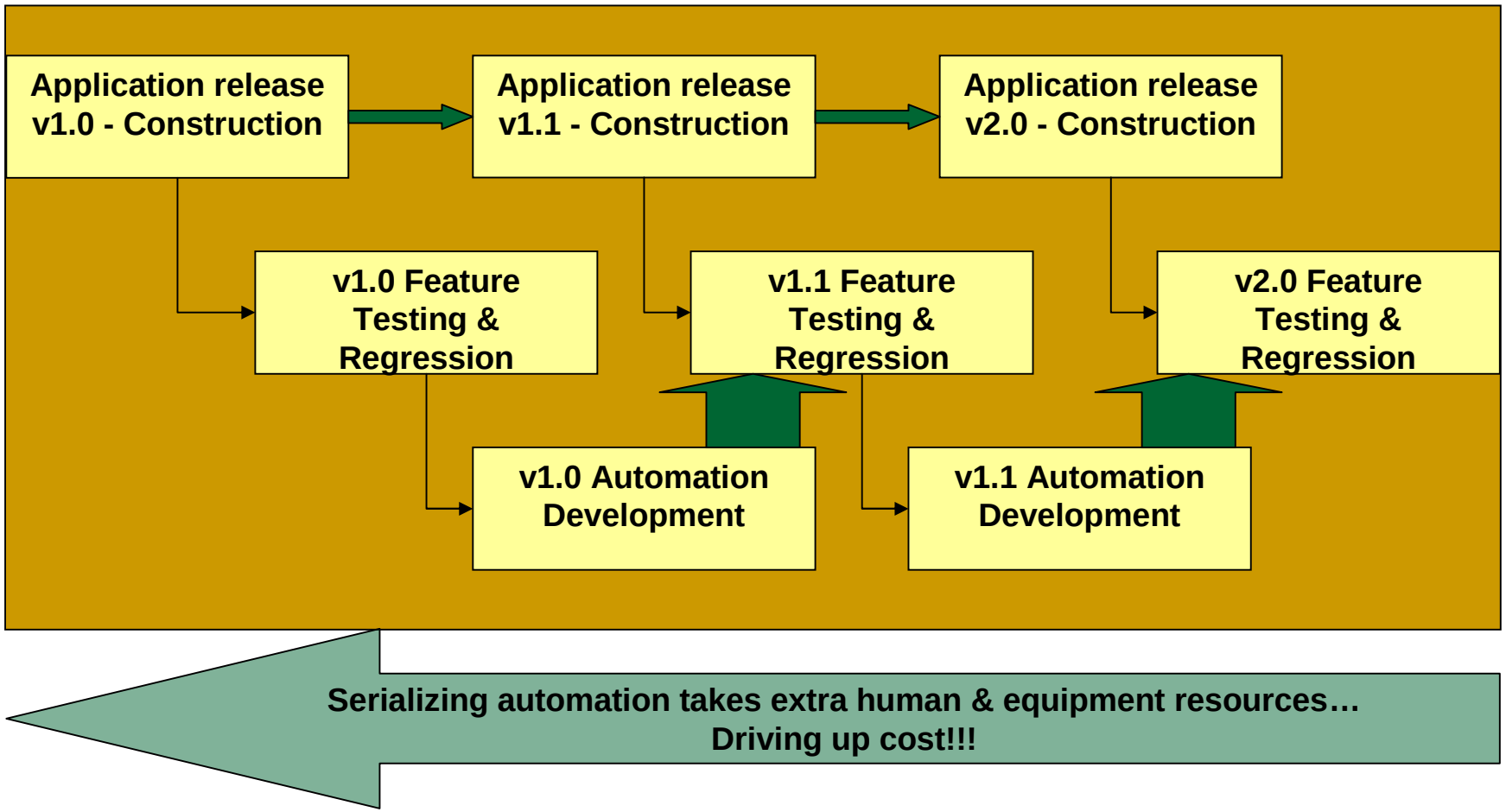
- First of all, it's another development project!
  - With all that implies: requirements, project management, risks, internal & external dependencies, bugs, etc.
  - Notions of architecture and design; heavy collaboration with the development team
  - Now you have two, parallel projects to coordinate
- Beyond start-up costs, there are –
  - Training, consulting, and skill-set upgrade costs
  - Tool costs
  - Maintenance burden & associated costs

Test Automation
Mitigating Risk

- Automation is often viewed as an automatic risk mitigation strategy. For example –
  - You simply run it as a regression safety net of sorts
  - Continuous (7x24) execution

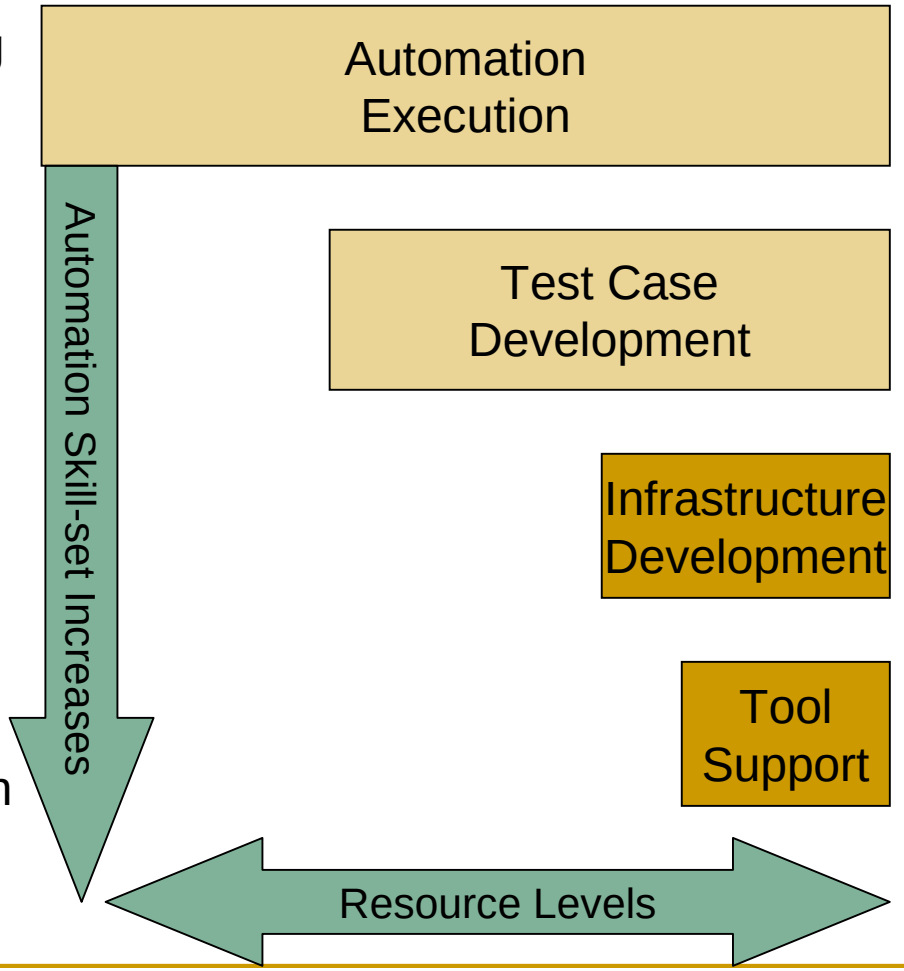
- While automation IS a viable strategy and required to truly achieve your speed goals, be careful to –
  - Acquire a proper environment and toolset
  - Understand your maintenance and support needs & costs
  - Realize your current skill set and expertise needs
  - Understand the connection to your product SDLC and the tension between it and your automation efforts

Risk–Based Testing
Typical Automation SDLC – "Skew"

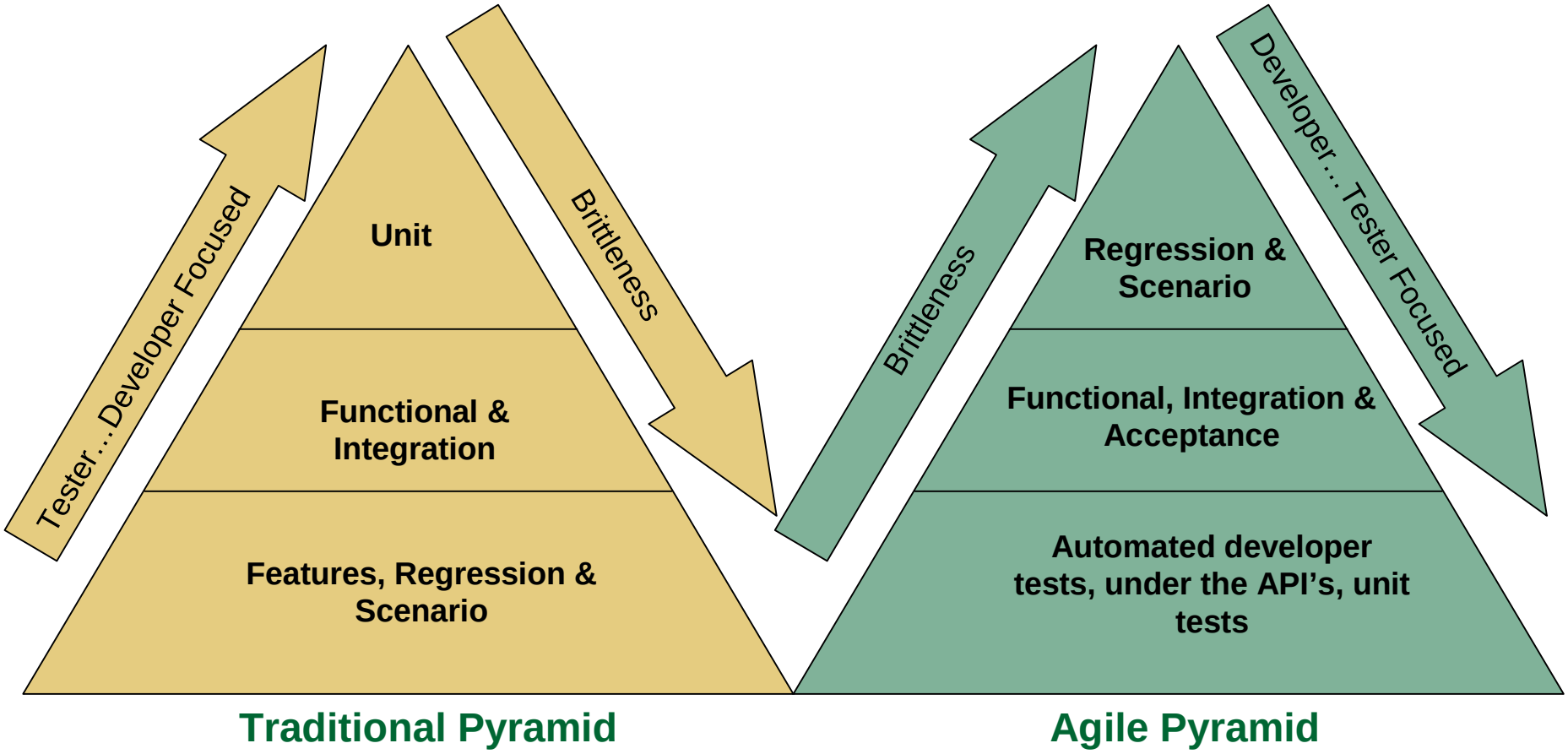


Test Automation Skill set

- Usually team struggle in acquiring automation skills –
  - Architecture & Design
  - Development
  - Toolset capabilities
- So they adopt a tiered model that aligns with resource skill-sets capabilities and automation construction effort balance
- Tools & Infrastructure are centralized; while test case design & execution are often distributed



Testing "Food" Pyramids
Another View to Maintenance



SQA Outsourcing

- Many firms are tempted to outsource their testing. The rational follows –
  - It's not a core competency nor does it generate IP
  - It's attractive from a cost perspective
  - Vendors seem to be at high CMMi levels and competent
  - A view towards testing as a commodity
- I agree that SQA is an attractive outsource target AND it should be part of your thinking…
- But,

Outsourcing
Candidate Selection

- Realize that not every application is a good candidate
- Good candidates:
  - Stable interfaces
  - Legacy products
  - Non-core products, low IP
- Bad candidates:
  - New, evolving products & interfaces
  - High IP content
  - You flagship products

Outsourcing
Culture & Capabilities

- Believe it or not, all cultures aren't created equal when it comes to testing

- Some issues:
  - Getting open & honest feedback on product quality
  - Raising issues for resolution
  - Tendencies for by-rote testing, starting at 1 and working to *n*
  - Domain experience gaps, attrition rates, and (re)training costs
  - Connection challenges between disparate process models (PM, SDLC, and Quality)
  - Accountability and motivation

Outsourcing
Sweet Spots

- Increasing bandwidth, speed, and coverage
  - Executing manual testing and increasing coverage
  - Regression testing
  - Legacy product coverage; product retirement risk mitigation
  - Automation execution
  - Types of testing requiring 7x24 coverage
- Technology & process sharing – improving local capabilities
- Raising your overall capacity bar
- Risk mitigation (holiday coverage, bench strength)

Wrap-up

Largest Challenges facing PM's & Test Teams

- Schedule compression
  - *We're behind 4 weeks in development – so we'll make it up in testing*
- Trivialization of testing effort
  - *Why can't we get the Boy Scouts to help us test?*
- Thoughtless compromises
  - *See "Schedule compression"*
- Agile testing
  - *Quality is entirely a development responsibility; i.e., we don't need no stinkin' testers*
- Traditional Mindsets
  - *100% testing; Zero defects; Quality & Process Gatekeeper*

Questions?

Thank you!

References & Backup

Basic Agile Principles

- Deliver working code in time boxed, small iterations
- Continuous integration, automated unit & acceptance testing
- Embrace change; lower the Cost of Change
- Customer collaboration; focus on delivering value & business acceptance
- Deliver just what's needed and no more
- Small teams; conversation & collaboration
- Trust teams judgment & capabilities

- Reference the Agile Manifesto

Agile Tester Profile
Disruptive to Traditional Views

Traditional Views

- Static Requirements
- Regression Testing
- Programming is for Programmers
- Detailed Test Planning

- Change Control
- Dedicated Phases for Testing
- Being in a Position of Authority or Gatekeeper
- Value by Testing

Agile Views

- Emerging requirements
- Iteration testing, continuous integration
- And for testers, increased technical skills, pairing
- Exploratory, collaborative, experience & trust based

- Embrace change
- Parallel work, small increments
- Quality as a team

- Value by delivering to customers and adding your own within the team!

Testing References

- **Black, Rex, "Managing the Testing Process – 2'nd Edition", Wiley, (2002)**
- **Black, Rex, "Critical Testing Processes: Plan, Prepare, Perform, Perfect", Addison Wesley, (2004)**
- **Copeland, Lee, A Practitioner's Guide to Software Test Design", Arctech House, (2004)**
- **Crispin, Lisa and House, Tip, "Testing Extreme Programming", Addison Wesley, (2002)**
- **Dustin, Elfriede, "Effective Software Testing: 50 Specific Ways to Improve Your Testing", Addison Wesley, (2003)**
- **Galen, Bob "Software Endgames – Controlling Mastering the Software Project Endgame", Dorset House Publishing, (late 2003 – early 2004)**
- **Kaner, Cem, Bach, James, and Pettichord, Bret, "Lessons Learned in Software Testing – A Context Driven Approach", Wiley, (2002)**
- **Kaner, Cem, Falk, Jack, and Nguyen, Hung Quoc, "Testing Computer Software", Wiley, (1999)**
- **Petschenik, Nathan, "System Testing with an Attitude: An Approach That Nurtures Front-Loaded Software Quality", Dorset House, (2005)**

Web References

- **Software Quality Engineering – www.sqe.com, and their portal www.stickyminds.com**
- **Software Testing & Performance magazine (free) – www.stpmag.com**
- **Florida Institute of Technology, Testing education site (Cem Kaner)– www.testingeducation.org**
- **Testing Blog site – www.testingreflections.com**
- **Elisabeth Hendrickson site – www.testobsessed.com**
- **Test Driven Development – www.testdriven.com**
- **Agile Journal – www.agilejournal.com**
- **CM Crossroads – www.cmcrossroads.com**
- **Association for Software Testing (Join!!!) - http://www.associationforsoftwaretesting.org/**

Contact Info

Software Endgames: Eliminating Defects, Controlling Change, and the Countdown to On-Time Delivery published by Dorset House in Spring 2005. www.rgalen.com for order info, misc. related presentations, and papers.

Robert Galen
RGalen Consulting Group, L.L.C.
PO Box 865, Cary, NC 27512
919-272-0719
www.rgalen.com
bob@rgalen.com