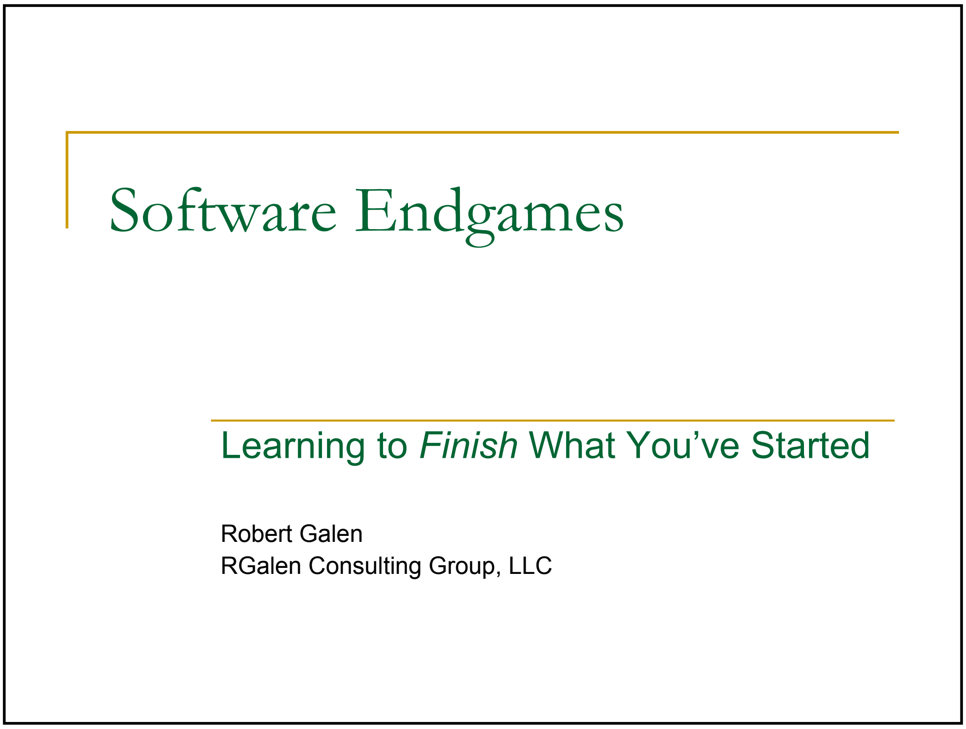
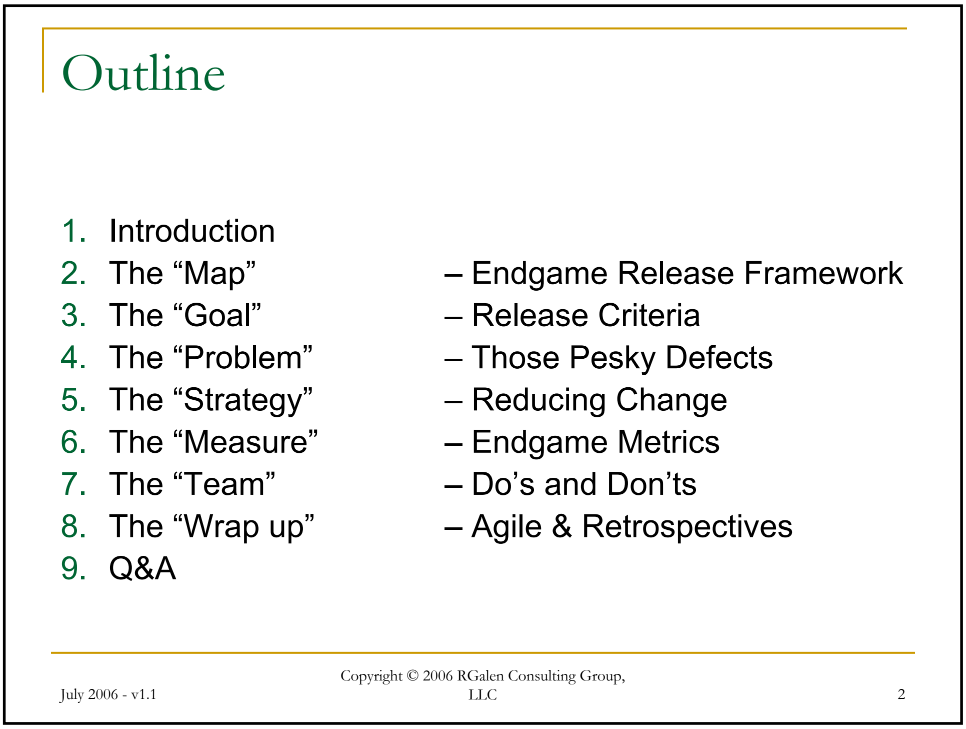


Software Endgames

Learning to *Finish* What You’ve Started

Robert Galen
RGalen Consulting Group, LLC



Outline

1. Introduction
2. The “Map” – Endgame Release Framework
3. The “Goal” – Release Criteria
4. The “Problem” – Those Pesky Defects
5. The “Strategy” – Reducing Change
6. The “Measure” – Endgame Metrics
7. The “Team” – Do’s and Don’ts
8. The “Wrap up” – Agile & Retrospectives
9. Q&A

Introduction

- Software Endgames are:
  - Period from 1’st release for testing outside of development thru to final customer release
  - Independent of methodology, although agility does introduce some “exceptions”
  - Primarily the domain of the testing team
  - Perhaps 40% of the overall project effort
  - Pressure packed towards – Release!

Introduction

- Let’s collect some data around *your* past endgames –
  - What are some of the core characteristics or patterns in a typical endgame?
  - What habits or patterns have you seen that work well to ensure a successful project outcome?
  - Other observations?

Introduction

- My Goals
  - Get you to think about your Endgames as a significant phase of your project and convince you that Endgames deserve your attention
  - Provide examples and tools to manage “change” activity during this time
  - Have you takeaway just ONE technique to apply in your next endgame

Endgame Release Framework

You can’t get to where you are going without some sort of MAP…

Map – ERF Defined

- High level plan for negotiating the Endgame, containing –
  - Historical Notes
  - Plan Details
  - Endgame Workflow
- Derived collaboratively with all major functional stakeholders (marketing/customer, development, testing, documentation, PM)
- Usually posted in a project wide locale and updated in real-time with Endgame adaptations

Map – ERF Components

- Historical Notes
  - Testing cycle characteristics
  - CCB meeting frequency, defect rates
  - Defect rates – find rates, repair rates, avg. TTR, deferral rates
  - Historical observations of
    - Code freeze maturity
    - Development team quality trends
    - Test team performance trends

Map – ERF Components

- Plan Details
  - Team organization (leadership, key roles & responsibility)
  - Number and type of testing passes or phases, for example:
    - Pass 1 - exploratory and familiarity testing
    - Pass 2 - re-work verification and regression testing
    - Pass 3 - re-work verification and regression testing
    - Pass 4 – final verification, regression and packaging testing
  - Handling of initial defect triage
  - Build and integration responsibilities
  - CCB logistics and attendee list
  - Alpha/Beta testing passes – focus and requirements

Map – ERF Components

- Endgame Workflow - For each release (iteration, pilot, focus group, Alpha/Beta, etc.) for testing define the following:
  - Testing focus or intent for this release
    - Initial, Regression, Beta pre-qualify, New features, Repair verification
  - Functionality % coverage targets
  - Other testing targets – performance, acceptance, etc.
  - Test duration, # days or weeks
  - Clear Entry & Exit criteria

Map – ERF
Define a Strategy

- As part of the Endgame Flow, you should define a strategy for estimating testing cycles and times, based on the history of previous endgames –
  - Testing cycle time
  - Defect trends & settling times
    - Zero bug bounce, S-curves
  - Cycles to achieve code freeze, code stability, final release quality levels
  - *Patterns* for: New feature introduction, Regression testing, Repair verification, Maturation phasing, Pareto defect trending

Map – ERF
Release Workflow Example

- Initial Release to Test - Pass #1: January 15, 2000
  - Focus: Initial testing – familiarity with product and level of quality
  - Functionality 80% complete, estimated test time 2 weeks @ 75% of delivered functionality
  - Entry criteria: release notes, unit test results and release review meeting for quality overview and candidate pass smoke test
  - Exit criteria: Testing @ 50% complete, 2 weeks elapsed time
- Release to Test - Pass #2: January 30, 2000
  - Focus: Testing new functionality, regression testing, verification testing and initial probing of system performance
  - Functionality 95% complete, estimated test time 2 weeks @ 100% of delivered functionality
  - Entry criteria: release notes, unit test results and release review meeting for quality overview and candidate passed smoke test
  - Exit criteria: Testing @ 80% complete, 2 weeks elapsed time
- …

Map – ERF
Release Plan Example

Build/Qual Milestone	Date	Notes
Pre-DVT Pkg 5.2.1	8/25/03	5.2.1.0
Pre-DVT Pkg 5.2.2	9/8/03	Original build had two P0 bugs. Respin still has issues that are being investigated.
Pre-DVT Pkg 5.2.3	9/22/03	Want to deliver new HW support in this pkg
RIT Pkg 5.2.4	9/29/03	1 week Release Integration Test beginning on 9/30
1st DVT Pkg 5.2.5	10/6/03	2 week qual beginning on 10/7 (re-spin MWare mid-cycle)
…	…	…
DVT Pkg 5.2.10	12/28/03	2 week qual beginning on 12/29 (re-spin MWare mid-cycle)
DVT Pkg 5.2.11	1/12/04	2 week qual beginning on 1/13 (re-spin MWare mid-cycle). Possible beta refresh candidate.
DVT Pkg 5.2.12	1/26/04	3 week qual beginning on 1/27
RTM Candidate Pkg 5.2.13	2/16/04	3 week qual beginning on 2/17. RTM target 3/9/04.

Map – ERF
Sabourin’s Test Cycle Rule of Thumb

*After code freeze, it minimally takes 3 testing cycles to complete testing. Due to the nature of testing, bugs and software, the first cycle takes twice the effort of the second. If the effort to complete the third cycle is A then the total test effort after code freeze should be (7 * A)*

Therefore, if you are allocated a testing budget of B hours, tests should be designed so that a complete cycle, without blocking bugs and with complete passes, will require (B / 7) person hours to complete!

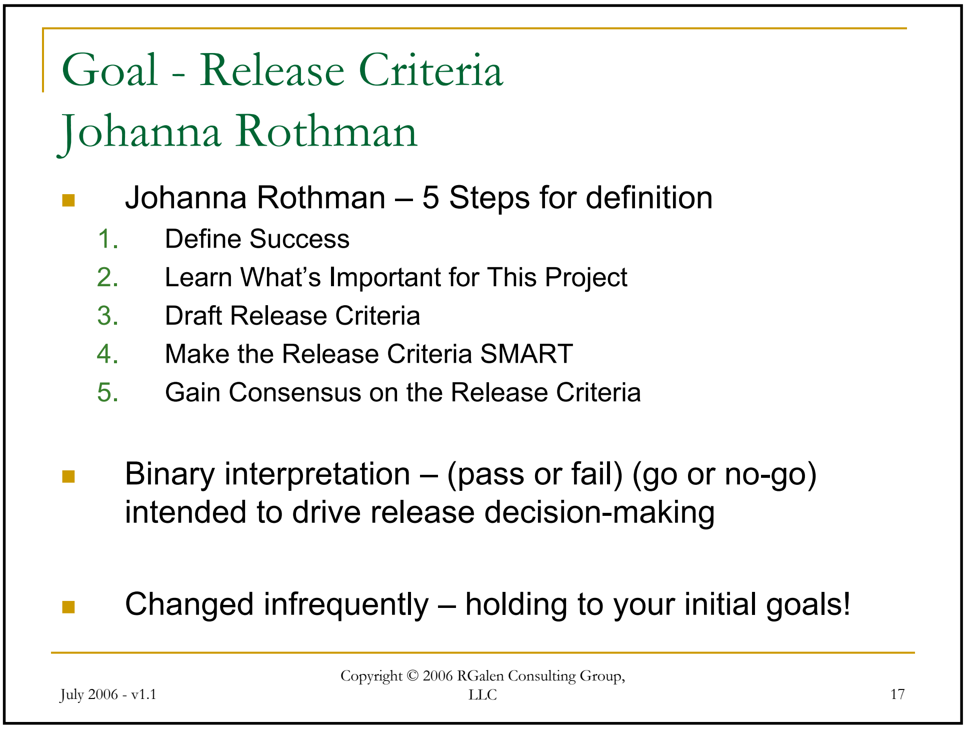
It is a sanity check for test estimation. If (B / 7) is unreasonable then you either need to increase B or scope down the project!

Release Criteria

You can't determine if you've ARRIVED unless you know what the GOAL is…

Goal - Release Criteria What Are They?

- From this perspective, criteria guiding ultimate product release decision-making
  - Linked to Requirements
  - Linked to the Customer & Business
  - Linked to the Product & Team
- Normally defined within Requirements phase of the project

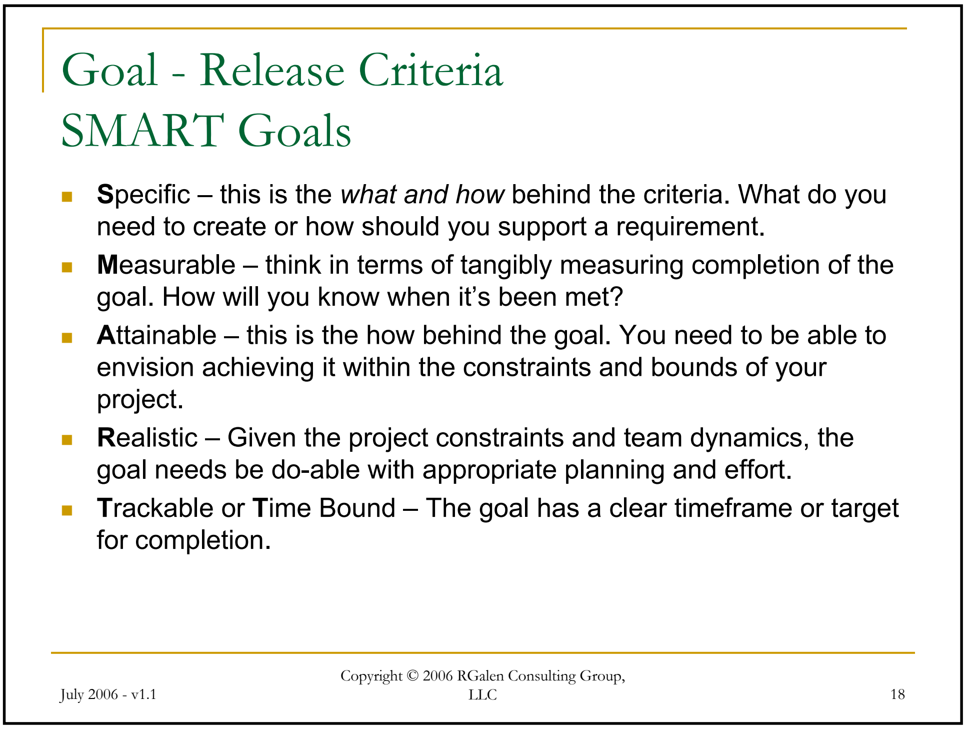


Goal - Release Criteria
Johanna Rothman

- Johanna Rothman – 5 Steps for definition
  1. Define Success
  2. Learn What’s Important for This Project
  3. Draft Release Criteria
  4. Make the Release Criteria SMART
  5. Gain Consensus on the Release Criteria

- Binary interpretation – (pass or fail) (go or no-go) intended to drive release decision-making

- Changed infrequently – holding to your initial goals!



Goal - Release Criteria
SMART Goals

- **S**pecific – this is the *what and how* behind the criteria. What do you need to create or how should you support a requirement.
- **M**easurable – think in terms of tangibly measuring completion of the goal. How will you know when it’s been met?
- **A**ttainable – this is the how behind the goal. You need to be able to envision achieving it within the constraints and bounds of your project.
- **R**ealistic – Given the project constraints and team dynamics, the goal needs be do-able with appropriate planning and effort.
- **T**rackable or **T**ime Bound – The goal has a clear timeframe or target for completion.

Goal - Release Criteria
Rothman Example

- Example Release Criteria –
  1. The code must support both Windows 2000 and Windows XP
  2. All priority P0, P1 and P6 defects will be repaired / addressed
  3. For all documented bugs – on-line help, release notes and formal documentation will contain relevant updates
  4. All QA tests run at 100% of expected coverage
  5. No new P0 – P3 defects found within the last 3 weeks
  6. Release target – GA of April 1, 2005

- To the left is a sample set of release criteria intended to guide activity for a given release. They speak to –
  - ❑ Platforms
  - ❑ Defects allowed
  - ❑ Defect actions
  - ❑ Coverage
  - ❑ Maturity
  - ❑ Performance
  - ❑ Release date

Goal - Release Criteria
Working View

- Working View – A different sort of Release Criteria

- Dimensional across – Scope, Quality, Time, People, Cost targets

- Graphical – To better illustrate the trade-offs and balance

- Hierarchical – Defined at a high level, then explored for each low priority dimension

- Explored, defined and changed with the team – Truly collaborative based on discovery and adjustment

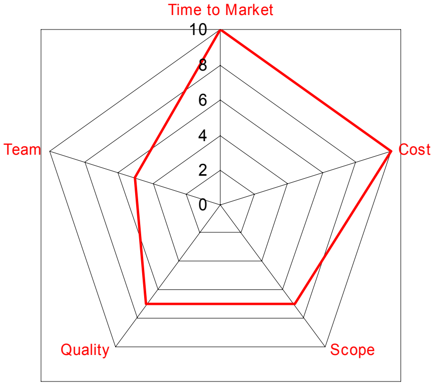
Goal - Release Criteria Working View

- Working View – 5 Steps
  1. Identify Project Stakeholders
  2. Set the Stage
  3. Establish Project Vision, Essence, Release Criteria
  4. Explore Project / Product Dimensions
  5. Iterate & Adapt

- Dynamic, team based with strong emphasis on trade-offs

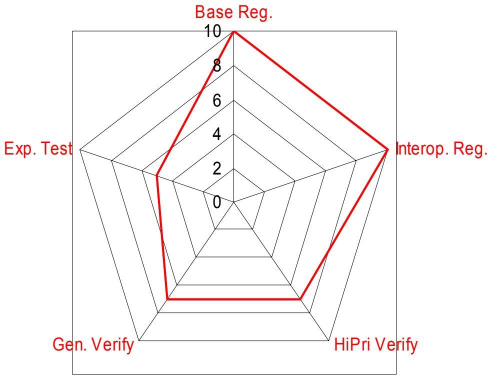
Goal – Release Criteria Working View Example

- Participants
  - Product Manager, Project Manager, VP of Engineering, Marketing and Testing/QA
- Dimensions of the problem
  - We are not able to acquire additional experienced resources
  - TTM is our ultimate priority – our customers and the business need the release
  - We must work some overtime – give extra effort
  - We can't regress deployed functionality – even in the interoperable cases – and we must include all field based severity 1-2 repairs
- Ranking
  - TTM = 10 & Cost = 10,
  - Scope = 7 & Quality = 7,
  - Team = 5



Goal – Release Criteria Working View Example

- Participants
  - Product Manager, Project Manager, Team leads from development and testing functions
- Dimensions of the problem
  - Insure we have full regression tests for deployed functionality – continue to run tests & report results
  - Extend existing regression tests to insure interoperability is covered
  - New additions to regression suite
  - Repair verifications may lag behind
  - We can't perform any ad-hoc testing
- Ranking
  - Previous version regression = 10,
  - Interoperable regression = 10,
  - 1 & 2 repair verification = 8,
  - General verifications = 6,
  - Exploratory testing = 2



Goal – Release Criteria Working View Example

An alternative approach is High / Low or Do / Don't Do Contrasting Charts

High Priority / Focus

- Existing functionality can not be affected by new changes - (functional regression testing)
- Existing performance may not be degraded by new changes - (performance regression testing, we also lacked a performance benchmark)
- New functionality must work
- Component interoperability w/o performance regression

Low Priority / Focus

- Interfaces beyond 10/100/1000 Ethernet and ATM are lower priority
- Existing performance may not be degraded by new changes - even when running multiple components, don't get hung up on improvement
- New functionality must work, except new reports that do not map to older reports
- Component interoperability across all permutations, we can identify (n) key

Goal – Release Criteria Working View Example

Will Cover (In Bounds)

- Will run 100% of the existing regression suite prior to final release, must have > 95% pass rate of tests
- Will test all new features associated with higher performance network interfaces (GE 10, 100, 1000)
- Of the reporting options, will test the customer highest priority (most used) 50 for correctness.
- Performance will be tested at the 10Gb., 50% throughput level, with filtered recording for 10 hours.
- Will work with customer to demonstrate & run 100% of the acceptance tests.

Will Not Cover (Out-of-Bounds)

- Regression will not contain features presented in the last iteration and these will only be tested on a risk (< 20% of the tests) basis.
- Will not regress nor feature test on slower interfaces – less than 10 Gb.
- Of the remaining 250 reporting options, will only sample a few via exploratory testing.
- All other interface performance will not be tested. Also, the requirement is for 24 hours of record time and, at best, we can extrapolate from 10.
- There will be no usability testing nor support of the planned pilot releases.

Those “Pesky” Defects

Defects are the core IMPEDIMENT to your reaching a successful release. How you DEAL with them makes all the difference…

Problem – Those "Pesky" Defects Add "Dimensions" to Repairs

- In my experience, organizations get too caught up in binary repairs decisions – fix or don't fix. They therefore miss tremendous flexibility in their decision-making
  - Good software project managers should guide their teams' defect triage towards multi-faceted repair decisions
  - Good testers and test teams should help remind their projects of these options

Problem – Those "Pesky" Defects Add "Dimensions" to Repairs

- Repair
- Make only a "partial" repair to buy some time
- Log it as "known" and move on
- Change the priority and/or severity to reset your "view" to the defect
- Ignore it and hope for the best
- Consider it anomalous behavior and wait for the next occurrence
- Defer the repair to a later release
- Negotiate with marketing or the customer for "relief"
- Add more system resources to reduce the impact
- Find a work around
- Remove the offending code
- Change or remove interacting software products

Problem – Those “Pesky” Defects Add “Dimensions” to Repairs

- Of course each one of these decisions has its own decision criteria and context and “depends” on –
  - Priority, customer impact, business impact
  - Team bandwidth and overall schedule
  - Previous defect trends, specific product domain
- The “Working View” can be particularly helpful here in making these decisions
- Remember – there is always next time…

Problem – Those “Pesky” Defects Directing Repairs

- I’ve seen too many teams approach defect repairs without any direction –
  - Simply picking what you want from the list
  - Little thought to repair efficiency, side effects, regression risks
  - Over-loaded and under-loaded resources
  - Developers making repairs to unfamiliar code
  - Little thought towards repair “packaging” – outside of priority
  - Little overall repair time estimates / predictability

Problem – Those "Pesky" Defects
Directing Repairs – Other Considerations

1. Reproducibility (ease, resources, time)
2. Overall Level of Difficulty (LOD)
3. Locality (proximity to other work)
4. Impact on the Testing team
5. Generally – potential workarounds
6. Handling "Gold" features
7. Your team capabilities (knowledge, familiarity, skills & current loading)

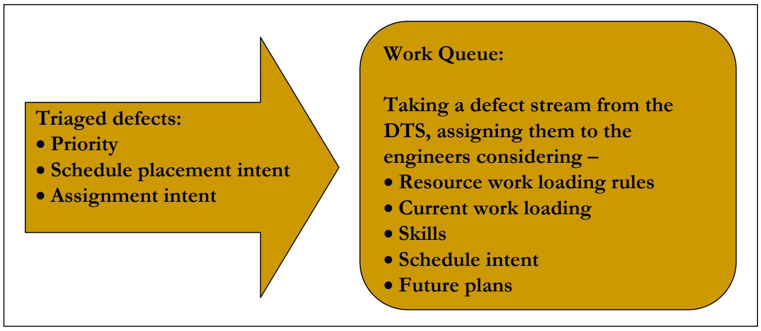
Problem – Those "Pesky" Defects
Consider Level of Difficulty

LOD	Typical Defect Characteristics	Approximate TimeToRepair
1	not well understood, pervasive problem that is fundamental to operation near-impossible or impossible to reproduce verification time unknown	> a week
2	pervasive problem and not so well understood difficult to reproduce verification time unknown	3–7 days
3	isolated and well understood, but broader scope of change relatively easy to reproduce verification effort moderate	1–3 days
4	isolated, cosmetic, well understood, minor code change easy to reproduce verification effort easy	< a day
5	trivial and cosmetic. Examples include spelling or grammatical errors on documentation, simple errors on UI screens or reports. easy to reproduce verification effort trivial	< an hour

Problem – Those “Pesky” Defects

Directing Repairs

- Concept of Work Queues implemented from within any Defect Tracking system



Triaged defects:
- Priority
- Schedule placement intent
- Assignment intent

Work Queue:

Taking a defect stream from the DTS, assigning them to the engineers considering –
- Resource work loading rules
- Current work loading
- Skills
- Schedule intent
- Future plans

Problem – Those “Pesky” Defects

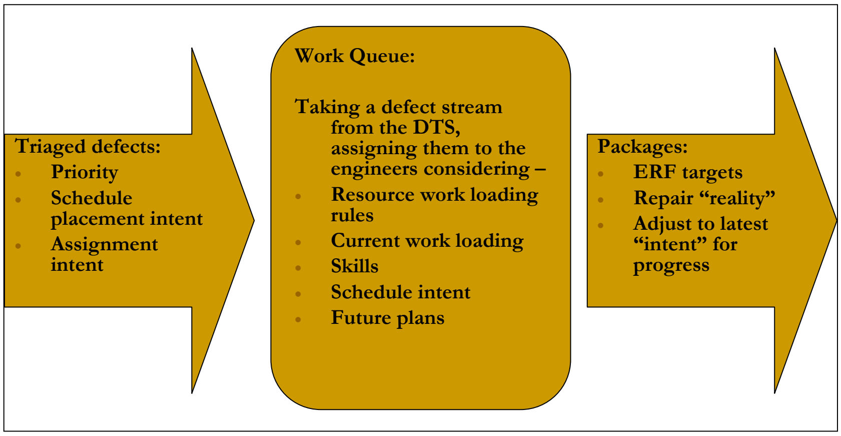
Directing Repairs

- **Engineer Name**
- **Time allocated (hours per day or week) for repairs**
- **What are their particular areas of expertise**
- **Bandwidth - this is defined by the team lead and the engineer. Things that come into play include –**
- **Are they providing leadership or mentoring to others**
- **Overall experience level**
- **Experience level with particular components**
- **Debugging abilities**
- **Ability to handle high severity / priority defects, how many at one time**
- **Ability to work on tasks in parallel**
- **Current loading - from the defect tracking system, assigned defects in severity order for planned releases 1…n**

- Resource Profile define for each developer
- Describes the abilities and constraints around defect repairs
- Helps to guide effective repair assignments and proper load balancing

Problem – Those “Pesky” Defects

Packaging Repairs



Triaged defects:

- Priority
- Schedule placement intent
- Assignment intent

Work Queue:

Taking a defect stream from the DTS, assigning them to the engineers considering –

- Resource work loading rules
- Current work loading
- Skills
- Schedule intent
- Future plans

Packages:

- ERF targets
- Repair “reality”
- Adjust to latest “intent” for progress

Problem – Those “Pesky” Defects

Packaging Themes

- Packages should be planned to have “Themes”, here are a few examples –
  - The very *FIRST* release to test
  - *Showstopper* package(s) that contains one to only a few repairs to remove obstacles that have slowed or halted testing
  - *Interim* package(s), for releasing repairs and new functionality
  - *Major* releases and *Minor* releases, each containing large and small amounts of changes respectively
  - There is the *Code freeze* package for a particular piece of functionality, component or the entire product
  - The *LAST* package, the one with ALL of the final agreed fixes in it that need verification

Problem – Those "Pesky" Defects Packaging Themes

- Themes help to –
  - Create a sense of release tempo throughout the Endgame
  - Focus the development and testing teams towards the primary goals for each iterative release
  - Help to drive discussion around appropriate loading (changes) given the theme for the release
    - To load balance the release with work that development can produce and testing can consume
  - Create hand-off expectations
- Should be developed as part of ERF creation
- Define your overall maturation strategy

Reducing Change

Independent of methodology, at some point change becomes the enemy and you need a STRATEGY to reduce and ultimately stop it…

Strategy – Reducing Change Drivers

- Reducing change is all about repair decision-making –
  - Given your release criteria
  - Given product requirements
  - Given business goals
  - Given defect trending and product quality levels
  - What needs to be repaired and when?
- At some point, there needs to be discussion and a decision

Strategy – Reducing Change LW-CCB Characteristics

- LW-CCB (Lightweight - Change Control Board)
  - Simply a periodic meeting
  - All stakeholders involved, crisply facilitated by a meeting Chair
  - Repair decisions (priority / impact based)
  - Guide team repair activity w/results logged in DTS
- Other dynamics
  - Defects can be triaged offline (preferred state and > 70% should be handled this way)
  - Lots of offline investigation and collaboration

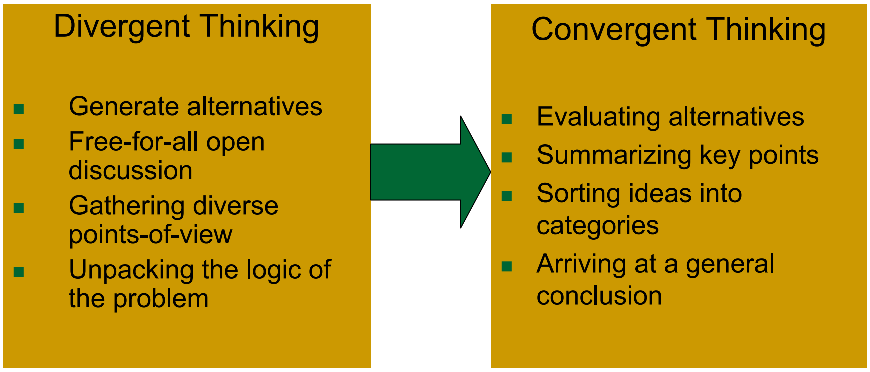
Strategy – Reducing Change
LW-CCB Agenda Structure

7 Phase **Meeting Structure**	**Notes**
1. Functional Roundtable	❑ Quick functional status
2. Old Work Review	❑ Review decisions or analysis
3. New Work Planning	❑ Majority of time (70%) spent here
4. Examine Trending	❑ Quick look at project trends
5. Release Criteria Adjustments	❑ Based on meeting, adjust?
6. Meeting Follow-up	❑ Minutes / DTS updates
7. Meeting Metrics	❑ Productivity/flow of LW-CCB

Strategy – Reducing Change
LW-CCB "Feelings"

- A "good" CCB meeting has the following attributes or *feelings* associated with it –
  - ❑ The meeting started and ended on time and it was *fully attended*.
  - ❑ The time was efficiently managed and well spent. There was a feeling of solid *accomplishment*.
  - ❑ The meeting was well led and facilitated, clearly someone was *in charge* of the meeting.
  - ❑ The team was properly *prepared*, doing the requisite off line "homework" required in defect analysis.
  - ❑ We didn't solve the problems, we *triaged* them.
  - ❑ Most of the defects reviewed *felt right*. That is, they were defects that required a group decision and you could understand why they were discussed at the CCB.
  - ❑ Meeting *follow-up occurs naturally*. Defects are updated with results, minutes are sent out quickly, plans are adjusted appropriately and actions are assigned.

Strategy – Reducing Change Triage Decision-Making



Divergent Thinking

- Generate alternatives
- Free-for-all open discussion
- Gathering diverse points-of-view
- Unpacking the logic of the problem

Convergent Thinking

- Evaluating alternatives
- Summarizing key points
- Sorting ideas into categories
- Arriving at a general conclusion

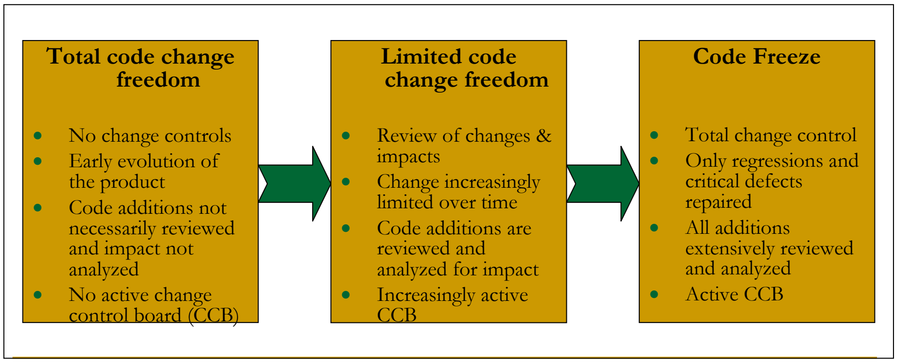
Strategy – Reducing Change Decision-Making

- Participatory Decision-Making: Core Values
  - Full Participation
  - Mutual Understanding
  - Inclusive Solutions
  - Shared Responsibility

- Decision-Making Models
  - Collaborative consensus *(hardest to achieve, "best" decision)*
  - Consensus with decision leader
  - Voting *(easiest to achieve, "worst" decision)*

Strategy – Reducing Change
Code Freeze

- Along with a well defined CCB structure, the team needs to have a notion of “code freeze”



Total code change freedom

- No change controls
- Early evolution of the product
- Code additions not necessarily reviewed and impact not analyzed
- No active change control board (CCB)

Limited code change freedom

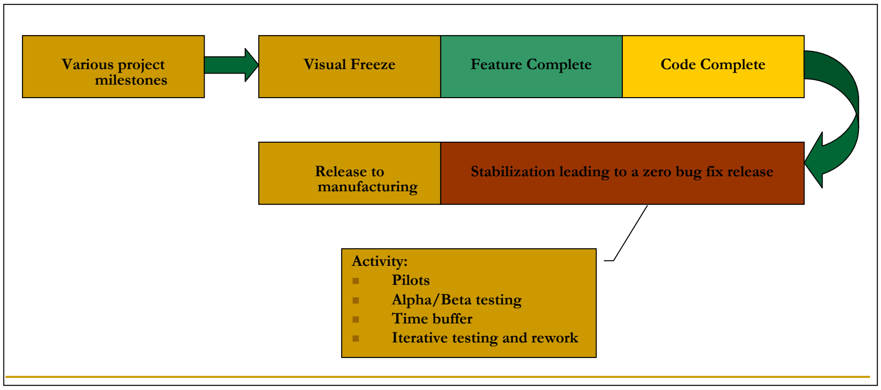
- Review of changes & impacts
- Change increasingly limited over time
- Code additions are reviewed and analyzed for impact
- Increasingly active CCB

Code Freeze

- Total change control
- Only regressions and critical defects repaired
- All additions extensively reviewed and analyzed
- Active CCB

Strategy – Reducing Change
Microsoft - Code Complete

- Microsoft employs a “code complete” strategy as defined below –



Various project milestones

Visual Freeze

Feature Complete

Code Complete

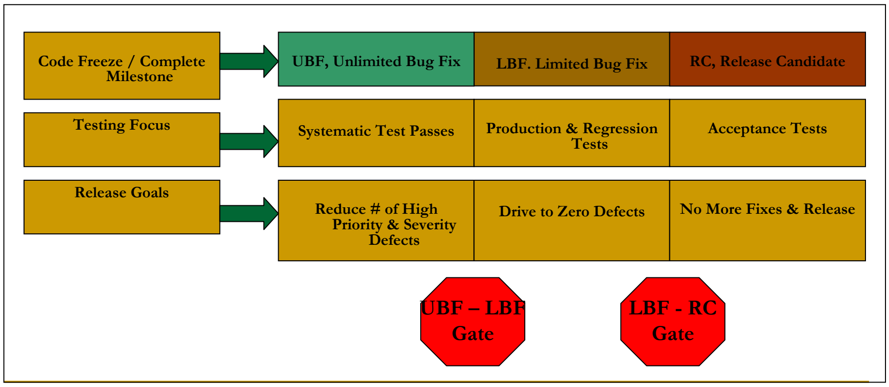
Release to manufacturing

Stabilization leading to a zero bug fix release

Activity:
- Pilots
- Alpha/Beta testing
- Time buffer
- Iterative testing and rework

Strategy – Reducing Change
Schoor – Bug Fix Gates

- Bruce Schoor has an extension to the Microsoft "code complete" strategy as defined below –



Code Freeze / Complete Milestone	UBF, Unlimited Bug Fix	LBF. Limited Bug Fix	RC, Release Candidate
Testing Focus	Systematic Test Passes	Production & Regression Tests	Acceptance Tests
Release Goals	Reduce # of High Priority & Severity Defects	Drive to Zero Defects	No More Fixes & Release

Strategy – Reducing Change
It Takes Courage

- It's easy to keep repairing defects…
  - It creates a sense of accomplishment
  - You're doing it for the customer
  - Quality is job #1
- It's hard to drive towards a release point
  - Effective prioritization, good judgment, and managing risks
  - Focusing on goals and making just the *right amount* of adjustments
- Next are some Anti-Patterns to watch out for…

Strategy – Reducing Change
Anti-Pattern: Delayed Code Freeze

- Also know as:
  - I don’t seem to be able to hold her Captain
  - Severe, unobstructed optimism
  - Thickening, but never frozen
- Corrective actions:
  - Perform component, architectural level, or other area partial freezes
  - Create additional “last chance” milestones – you’ve got to stop sometime
  - Improve CCB controls, insight into impacts
  - Re-examine QA planning & clarify impacts

Strategy – Reducing Change
Anti-Pattern: Inherent Instability

- Also know as:
  - If it ain’t broke, oops…
  - Oh, you mean it was supposed to work
  - Architecture, smarchitecture—if we build it, they will come…
- Corrective actions:
  - Leverage QA team to discover key problem areas. Real-time Pareto analysis can help here
  - Perform architectural review w/development team. Driven from the problem areas and behaviors
  - Situational assessment
  - Adjust release criteria and goals as appropriate

Strategy – Reducing Change
Anti-Pattern: Ad-hoc Testing Delays

- Also know as:
  - Testing without a compass or a map…
  - Spelunking for bugs…
  - Boldly going where no man has…

- Corrective actions:
  - Limit ad-hoc testing to more of an early release familiarization technique
  - Later on, replace with true exploratory testing (planned, goal based, results collected)
  - Re-emphasize planning, at the ERF level and focus on test strategy and scripted (traceable) results

Strategy – Reducing Change
Anti-Pattern: Never-ending Rework

- Also know as:
  - For every repair, 2 defect that takes its place
  - The never ending story
  - At least we have work to do

- Corrective actions:
  - Identify risky areas with Pareto
  - Increase planning surrounding analysis, estimating and assignment of repairs / enhancements
  - Emphasize CM system monitoring (check-ins) and institute process changes (acceptance reviews)
  - Via Release Criteria and CCB – take less on

Strategy – Reducing Change
Anti-Pattern: Volatile Release Criteria

- Also know as:
  - Which way do we go George?
  - Gosh, that was a painful CCB meeting…
  - I thought you were working on something else…
- Corrective actions:
  - The Working View is a good way to resolve this – by drilling down into the details of what’s truly important
  - Engage key stakeholders to ensure there is agreement and balance across the Criteria
  - Ensure team leads all consistently understand the goals
  - Check to ensure that the goals are *achievable*

Measuring Progress

To understand where you’re AT relative to where you NEED TO BE you need a STRATEGY for measuring progress and direction while continuously detecting the need for adjustment…

Measure – Progress
Role

- We (Testers) need to adjust from a
  - Detecting
  - Reporting
  - “Safety net” perspective or role
- To one including all of the above, but more so serving as a project *Guide*. Providing more -
  - Deep data insights and recommendations
  - Bottleneck and efficiency improvement guidance
  - Risk detection and resource application guidance

Measure – Progress
Origin of MAQ

- STQE May/June 2001 article by Anna Allison – Meaningful Metrics
- Looking for the meaning *behind* the metrics or project data
- Determining a list of questions to ask when viewing or observing project data trends
- Can simply ask. Increase your curiosity and give yourself a license to inquire.
- Always looking to provide higher level guidance to the project team

Measure – Progress
MAQ Examples

1. Has there been a change in QA personnel assignments that could account for fewer bugs being found in the last cycle?
   - Vacations, time-off, illness?
   - Another project starting up?

2. We've seen a drastic increase in priority 1 defects in this last release. Did the developers do something?
   - Add a new component, fix some related defects?
   - Are we testing something for the first time?

Measure – Progress
MAQ Examples

3. What kinds of bugs are being found?
   - Severity mix? Priority mix?
   - What is the trending? Did we expect this?
   - Are these only code bugs or other things? Relationships in counts and trending?

4. Is the software really stabilizing / maturing?
   - Where specifically – everywhere? Exceptions?
   - Same trends? Anything troubling or getting worse?

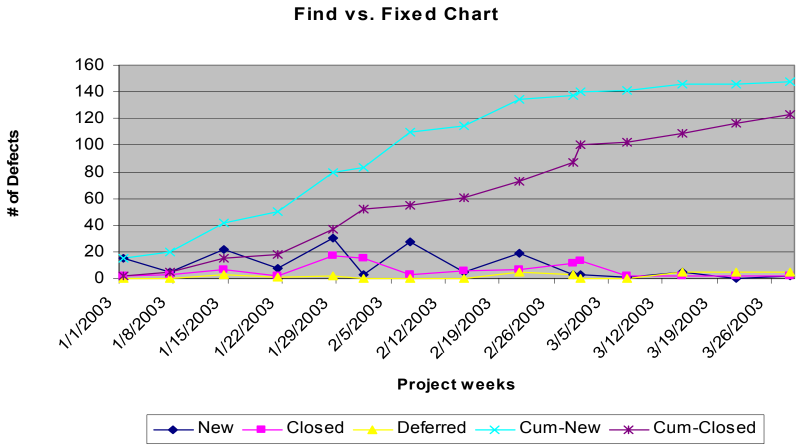
Measure – Progress
MAQ's Why?

- Why ask? Why probe?
  - Provide insight for improved management decision-making
  - Alert team to risks
  - Provide clarity and insight into the REAL state of the product & project
  - Convey the project context – scope, resources, quality, time
  - Because you have the ability to do so AND the functional breadth to do it well

Measure – Progress
5 Core Metrics

1. Found vs. Fixed
2. High Priority
3. Project Keywords
4. Defect Transition Progress
5. Product Functional Area Distribution

Measure – Progress
Defects Found vs. Fixed (Maturation)



Measure – Progress
Defects Found vs. Fixed (Maturation)

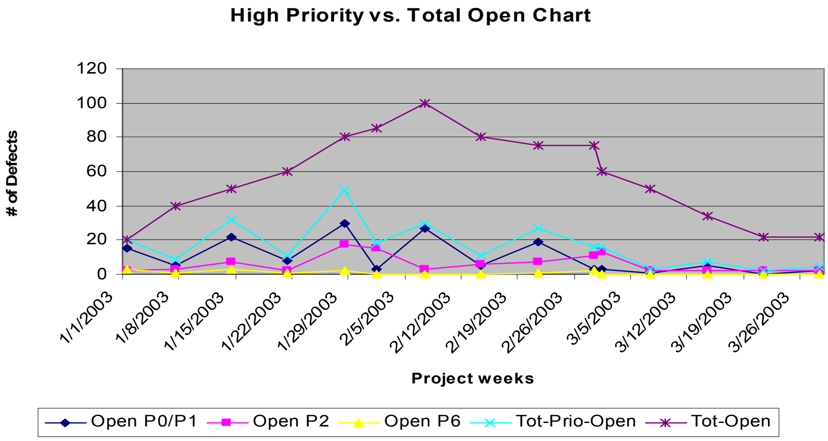
- Found – new, vs. Fixed – repaired, closed, duplicate, not-a-bug, works-as-designed, etc.

- MAQ
  - General trends? Found down, fix up general trending
  - Look for stabilization in trending – for a duration or test cycle
  - Release point – cumulative trends intersect, with period of low / no find and stability

Measure – Progress
Defects Found vs. Fixed (Maturation)

- Key Guide Points
  - Carefully view trending to take into consideration organizational or other anomalies
  - Looking for equilibrium or regression trends (progress & stability vs. backsliding)
- Feedback to Project Management
  - Release readiness predictions
  - Triage / closure concerns
  - Intermittent trend influences – identifying root causes

Measure – Progress
High Priority (Robustness & Risk)



Measure – Progress
High Priority (Robustness & Risk)

- Specifically defined set of defects indicating high priority (defects vs. enhancements) monitored as a significant group

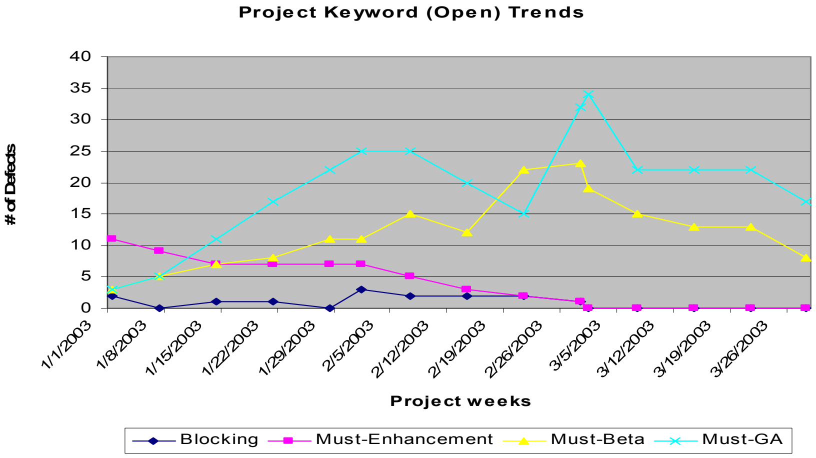
- MAQ
  - Overall trending – relative to position within the Endgame release cycle
  - Stable downward curve, no spiking and regression
  - Defects vs. Enhancements – scope creep

Measure – Progress
High Priority (Robustness & Risk)

- Key Guide Points
  - High priority defects should decline and stabilize early in testing iterations
  - Continuous and abrupt downward trending
  - Shouldn’t occur midway to late in Endgame

- Feedback to PM
  - Any “spiking” in trending usually indicates a systemic regression of some sort
  - Late in the Endgame implies lack of robustness and increasing risk

Measure – Progress
Project Keywords (PM Workflow Impact)



Measure – Progress
Project Keywords (PM Workflow Impact)

- Attaching keywords to defects, allowing for targeting repairs towards your iterative release planning scheme. Normally driven by triage and change control.

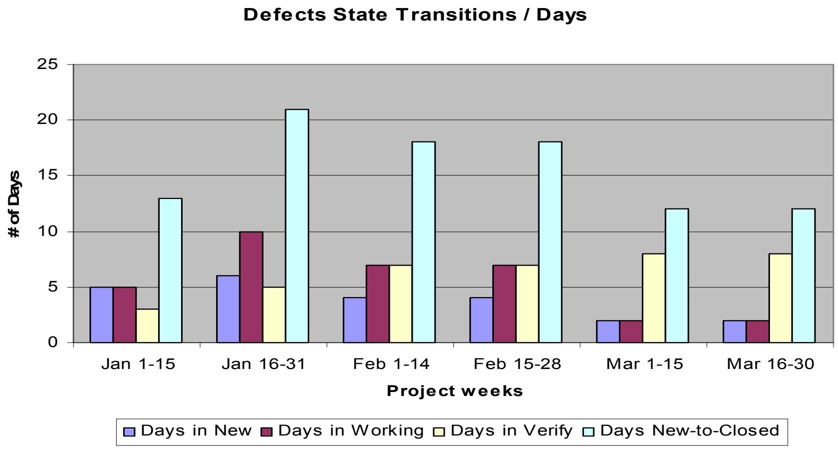
- MAQ
  - What is the trending of “must have” repairs for individual project milestones
  - What are the priorities within each targeted milestone release
  - Will we make it and what to defer

Measure – Progress
Project Keywords (PM Workflow Impact)

- Key Guide Points
  - Define meaningful keywords aligned with major project milestones
  - Target repairs towards these milestones

- Feedback to PM
  - Blocking issues
  - Trending relative to milestones, example Code Freeze
  - Deferral guidance (priority, impact)

Measure – Progress
Defect State Transitions (Team Capacity)



Measure – Progress
Defect State Transitions (Team Capacity)

- Reviewing defect activity state transitions to determine team capacity (efficiency & capacity)

- MAQ
  - What is typical triage / assigning time (Triage)?
  - What is the typical repair turnaround time (Dev)?
  - What is the typical verification time (Test)?
  - What are the trending and relationships amongst the team workflows? Improving or not? Performance problems, bottlenecks?

Measure – Progress
Defect State Transitions (Team Capacity)

- Key Guide Points
  - As the project progresses, triage should reduce from 5-7 days average to < day average
  - Development time should decrease as they move from construction iterations to maintenance
  - Verification time relates to overall cycle time, early feedback preferred.
    - Regression rates come into play
  - # of days moving from New -> Closed indicates overall team capacity and time to resolve defects. Should be factored into iteration release plans.

Measure – Progress
Defect State Transitions (Team Capacity)

- Feedback to PM
  - Triage backlog & workflow bottlenecks – tracking average time to New -> Close
  - Filter out anomalies
  - Development-to-testing equilibrium
    - Either function overloaded, suggest adjustments. More developers, more testers, help each other, adjust goals.
  - Factor functional and team capabilities into iterative release planning.

Measure – Progress
Pareto Principle

- Pareto Principle also known as the 80:20 Rule
- Simply stated – *20% of something will typically have 80% of the impact*
- Non-software example:
  - Toyota Prius warehouse, 20% of the parts make up 80% of the volume
  - 20% of the parts also make up 80% of the cost
- Software: 20% of the components, objects, LOC, whatever will produce 80% of the bugs!

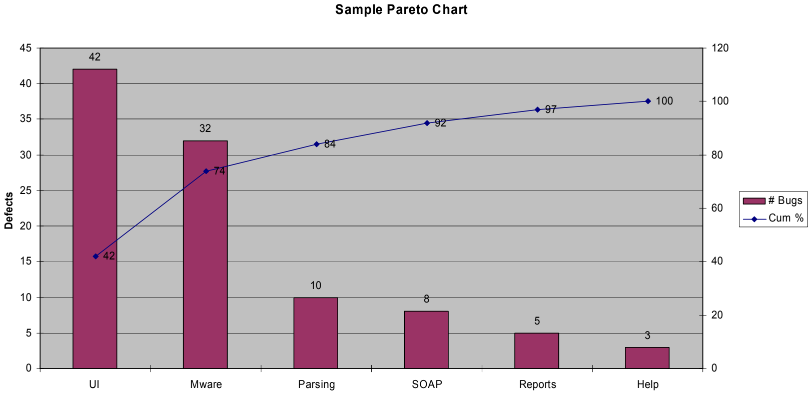
Measure – Progress
Pareto Principle

- Challenges are two-fold:

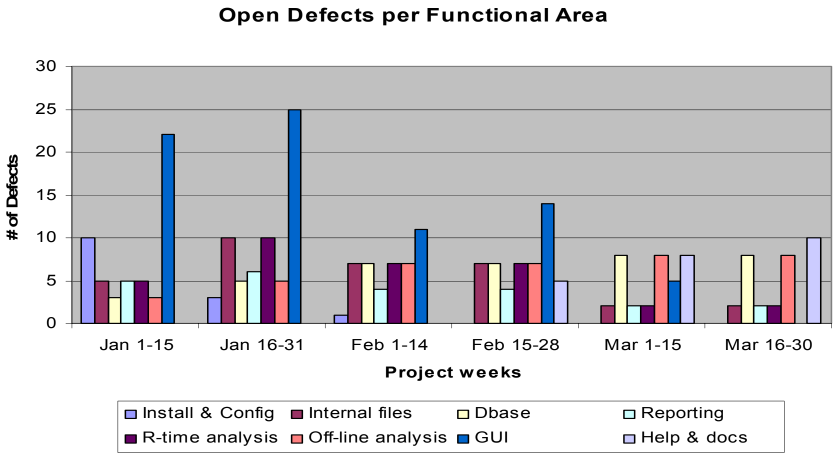
1. Partitioning
   - Partitioning the application into the right number of component areas, somewhere between 5 – 12
   - Roughly equivalent in size and complexity
   - Usually along architectural and/or contributing boundaries
     - *UI major features, middleware, DB & back-ends, API's, Services, external systems*

2. Modifying the DTS for locale isolation (not root cause!)
   - And having the experience / taking the time to isolate

Measure – Progress
Open Defects per Area (Pareto Guidance)

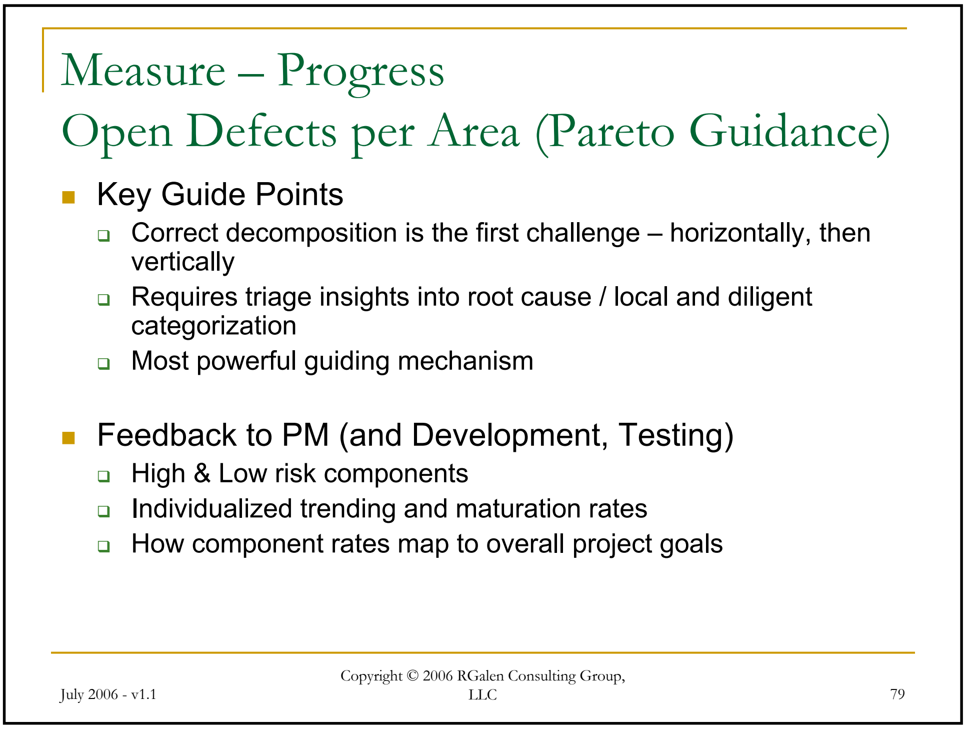


Measure – Progress
Open Defects per Area (Pareto Guidance)



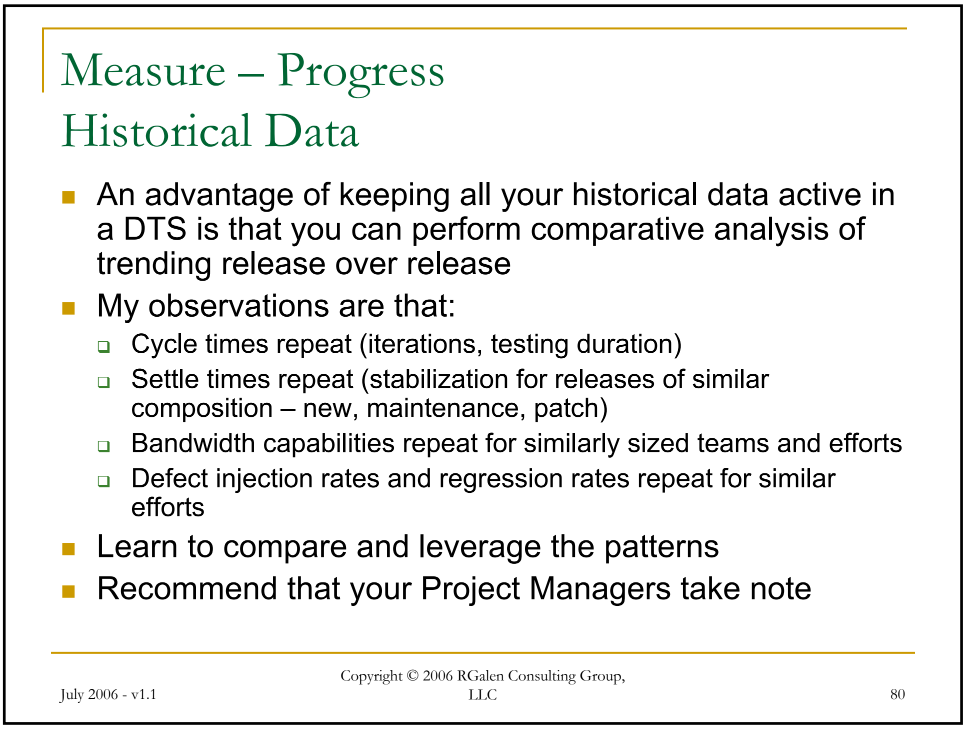
Measure – Progress
Open Defects per Area (Pareto Guidance)

- Partitioning the AUT into meaningful components, then diligently mapping defects to component yields valuable Pareto insights
- MAQ
  - Different trends within components? Is it expected behavior? (Late vs. early maturation)
  - Highest risk, lowest risk areas? By defect count, priority or regression?
  - Where to focus process (unit testing) and testing?



Measure – Progress
Open Defects per Area (Pareto Guidance)

- Key Guide Points
  - Correct decomposition is the first challenge – horizontally, then vertically
  - Requires triage insights into root cause / local and diligent categorization
  - Most powerful guiding mechanism
- Feedback to PM (and Development, Testing)
  - High & Low risk components
  - Individualized trending and maturation rates
  - How component rates map to overall project goals



Measure – Progress
Historical Data

- An advantage of keeping all your historical data active in a DTS is that you can perform comparative analysis of trending release over release
- My observations are that:
  - Cycle times repeat (iterations, testing duration)
  - Settle times repeat (stabilization for releases of similar composition – new, maintenance, patch)
  - Bandwidth capabilities repeat for similarly sized teams and efforts
  - Defect injection rates and regression rates repeat for similar efforts
- Learn to compare and leverage the patterns
- Recommend that your Project Managers take note

Teams – Do's & Don'ts

Endgames are the most pressure-packed times within your projects. If you don't handle your TEAM effectively, you will fail in the short and/or long term…

Team – Do's & Don'ts

Leadership

- DO:
  - Have a leadership perspective of "Servant Leadership" realizing you're there for the team
  - Bring a positive and energized attitude every day
  - Understand the power of leadership – It Matters!
  - Use overtime in short bursts - judicious, infrequent, selective and targeted
- DON'T:
  - Be too self or business centered as a leader
  - No faster way to screw up an Endgame than to dictate overtime

Team – Do's & Don'ts
Roles & Responsibilities

- DO:
  - Setup specific roles for team leadership and interfaces between functional groups (Build & Integration, Test PM)
  - Signup your leaders for the effort
  - Consider team size and composition (skill, personality type) when setting up your teams. Consider adding or reserving bench strength!
- DON'T:
  - Just throw folks together under a project directive and expect good results

Team – Do's & Don'ts
Team Lead Charter

- **Team leaders are:**
- **Individually signed up for the job. I'm a firm believer in the enrollment process on teams – so take some time, sit down with each leader and enroll them in the effort. Make sure they want to be a part of the team and equally important a leader on the team.**
- **Clearly aware of the responsibilities for the role. Insure that they understand all of the nuances for the role. Sometimes there are more subtle expectations and challenges. Make sure they are prepared for them.**
- **The "point person" for a particular function:**
  - **directs repair and other endgame activities**
  - **provides input and guidance into the overall Endgame plan and framework**
- **provides primary functional representation to all CCB/Triage meetings**
- **coordinates release packages, while insuring overall content quality and performing hand-off reporting**
- **provides first level recommendations on changing your release criteria or "Working View"**
- **Very competent technically and broadly familiar with the product in order to provide sufficient guidance and mentoring to their team.**
- **From a more technical perspective for development and testing functions, capable of mentoring team members, "pitching in" to help solve tough problems and directing overall team adherence to proper processes and practices.**

Team – Do’s & Don’ts
DTS Implementation

- DO:
  - Your DTS is central to your Endgame – ensure it’s in place, accessible and everyone know how to use it
  - Overload the DTS for work queues, Pareto analysis, estimating, features &
  - Leverage DTS driven historical data whenever possible for trending insights
    - From directly similar projects
    - Or from organizational / domain similar efforts
- DON’T:
  - Track bugs & feature work haphazardly, without consistent rules, or in multiple systems

Team – Do’s & Don’ts
Making Repairs

- DO:
  - Plan and estimate your repairs & packaging
  - Consider increasing repair process rigor (inspections, unit testing, integration steps, etc.) as you approach release
  - Consider level of difficulty in making assignments, repairs, and determining testing requirements
  - Use work queues for balanced repairs
- DON’T:
  - Just fix bugs indiscriminately, without a plan or assignment strategy

Team – Do’s & Don’ts
Resource Balancing – Pareto everywhere!

- DO:
  - Consider reserving some of your best developers and testers as “Bench Strength”
  - Find your best debuggers – component level and system wide
  - Understand your regression rates
  - Understand your Pareto components and the relationship to team members
- DON’T:
  - Thank that all work is the same and needs the same level of scrutiny and effort

Team – Do’s & Don’ts
Team Logistics

- DO:
  - Setup an Endgame War Room for posting project state – Information Radiators. It’s for ad-hoc group meetings (work & play) and be sure to stock it with food.
  - Hold daily standup Scrum-like Endgame meetings where status is shared (did, will do, impediments). Everyone talks – to each other (team collaboration and commitment).
- DON’T
  - Operate out of cubes or team separation or isolation

Team – Do's & Don'ts
Testing Team

- DO:
  - Realize that your testing team is your Endgame "Secret Weapon". They are intimate with the product maturity, stability and customer viability.
  - View the test team as your primary "Guide" in the Endgame – early warning of trends, Pareto sweet spots for efficiency, adjustments to release criteria, etc.
- DON'T
  - Ignore your test team or relegate them to a by-rote role within the Endgame

Team – Do's & Don'ts
Retrospectives

- DO:
  - Keep a personal log of your Endgame experience.
  - Conduct short Endgame retrospectives with your team. The book by Norm Kerth book is a wonderful guide here.
  - Document your core lessons and carry them forward to your next Endgame / ERF
  - Oh, and Celebrate a bit!!!
- DON'T
  - Just move on the next project without THINKING

Wrap-up

Wrap-Up

1. The “Map” – Endgame Release Framework
2. The “Goal” – Release Criteria
3. The “Problem” – Those Pesky Defects
4. The “Strategy” – Reducing Change
5. The “Measure” – Endgame Metrics
6. The “Team” – Do’s and Don’ts
7. The “Wrap up” – Agile & Retrospectives

The **“Result”** – A much higher probability of achieving ***Endgame Success***

Wrap-up
Endgame CM

- Effective Endgame configuration management practices include:

- Constructing an early CM plan
  - Then running a “trial” build through to testing to shake out process and tools issues

- Build & Release Practices
  - Continuous integration
  - Daily builds
  - Running smoke tests prior to SQA entry
  - Release “turnover” meeting w/release notes and

Wrap-up
Agile Endgames

- A daily, standup, Scrum-like meeting is central to the Agile Endgame

- Other key practices include –
  - Heavyweight defect entries
  - Release criteria
  - ERF
  - Collaborative work planning

- Agility in the Endgame sense is about key lightweight practices applied with good judgment and common sense

Wrap-up
Team Insight

- Endgames are the most stressful time within projects. As such, they provide a wonderful opportunity to discover –
  - The character of your team. Who the true leaders are. Who behaves badly and who behaves well under stress
  - Expose the conviction and quality of your own leadership
  - Ability to provide sound judgment and decision-making
  - Who are your best fire fighters and Endgame performers
- Don’t miss the opportunity to observe, learn and apply towards future efforts

Wrap-up
More on Retrospectives

- In the **Early Endgame Stages**, you are interested in:
  - Early high priority settling curves
  - Early defect rates for specific components
  - Overall settling curves, from initial high bursts to more normal defect rates
  - Development multi-tasking performance, finishing functionality vs. repairing early defects

Wrap-up
More on Retrospectives

- Whereas in the **Middle Endgame Stages** your interest should change to:
  - Ongoing component trends, start looking for Pareto distributions and defect clustering
  - Expected reductions in overall defect arrival rates, particularly higher priority defects
  - How long did it take to get to code freeze (weeks, cycles, releases, etc.)
  - The work balances between software development and testing. What are the trends to achieving a repair / test verification balance?

Wrap-up
More on Retrospectives

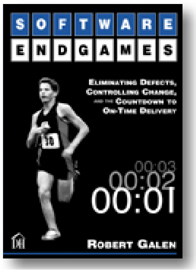
- Finally, in the latter or **Late Endgame Stages** the following is of interest:
  - Flat line rates for no P0/P1 defects, no “blocking” defects and further stability trends
  - Switch of work from focus development repairs to testing repair verifications
  - Time to get from code freeze to final test cycles (weeks, cycles, releases, etc.)
  - Non-repair defect handling rates (workarounds, deferrals, partial repairs, etc.)
  - Final Pareto analysis for targeting last bit of testing towards clustering or risk areas

Thank You!
For The Gift of Time

Questions?

References

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


SOFTWARE ENDGAMES
Eliminating Defects, Controlling Change, and the Countdown to On-Time Delivery
00:03
00:02
00:01
Robert Galen

Contact Information

Robert Galen
Principal Consultant

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