
Software Endgames

Learning to *Finish* What You've Started

Bob Galen

RGalen Consulting Group, LLC

www.rgalen.com bob@rgalen.com

Outline

- The “Map”
 - Endgame Release Framework
- The “Goal”
 - Release Criteria
- The “Problem”
 - Those Pesky Defects
- The “Strategy”
 - Reducing Change
- The “Team”
 - Do’s and Don’ts

The “Result” – Success by letting it go...

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
 -
 - 1. Historical Notes
 - 2. Plan Details
 - 3. 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

1. 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

2. 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

3. 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 – average length, number, exceptions
 - Defect trends & settling times: Zero bug bounce, S-curves
 - Cycles to achieve code freeze, code stability, final release quality levels; set milestones accordingly
 - *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
1 st 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
 - Specific, Measurable, Attainable, Realistic and Trackable
 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

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

http://www.rgalen.com/t_files/AspectsOfAProjectWorkingView_v2.doc

Goal – Release Criteria

Working View Example

■ Participants

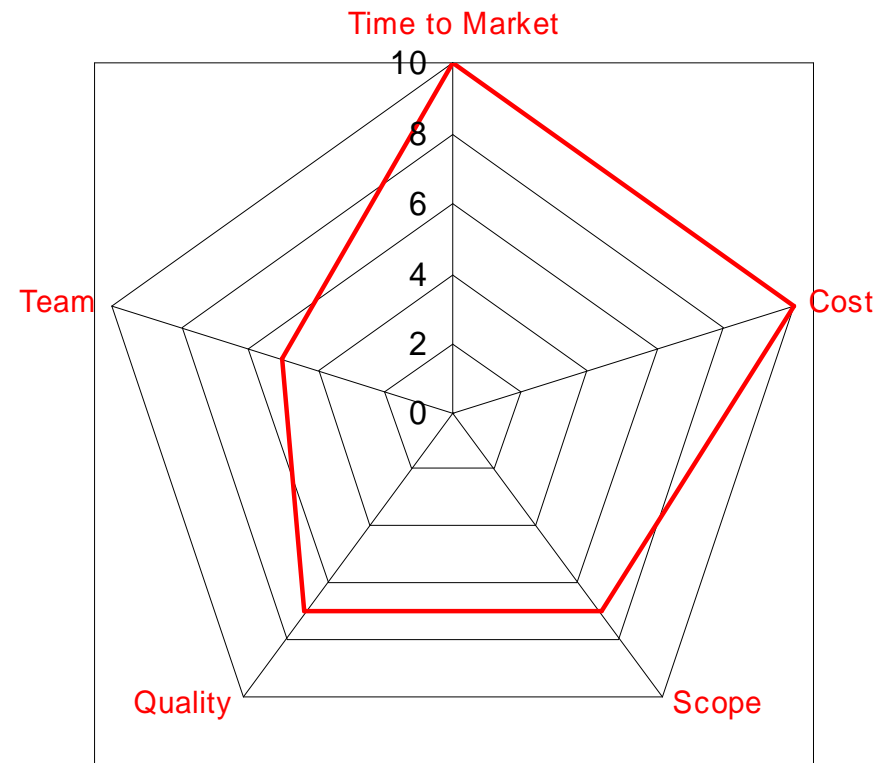
- Product Manager, Project Manager, VP of Engineering, Marketing and Testing/QA

■ 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

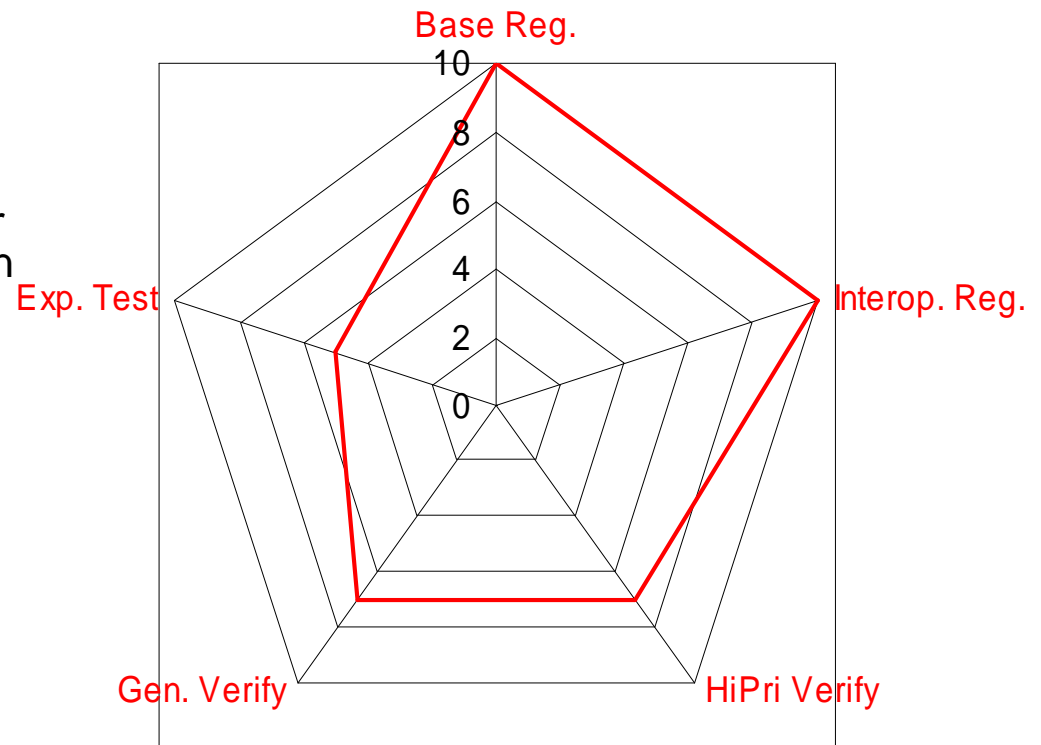
- Product Manager, Project Manager, Team leads from development and testing functions

■ 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

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

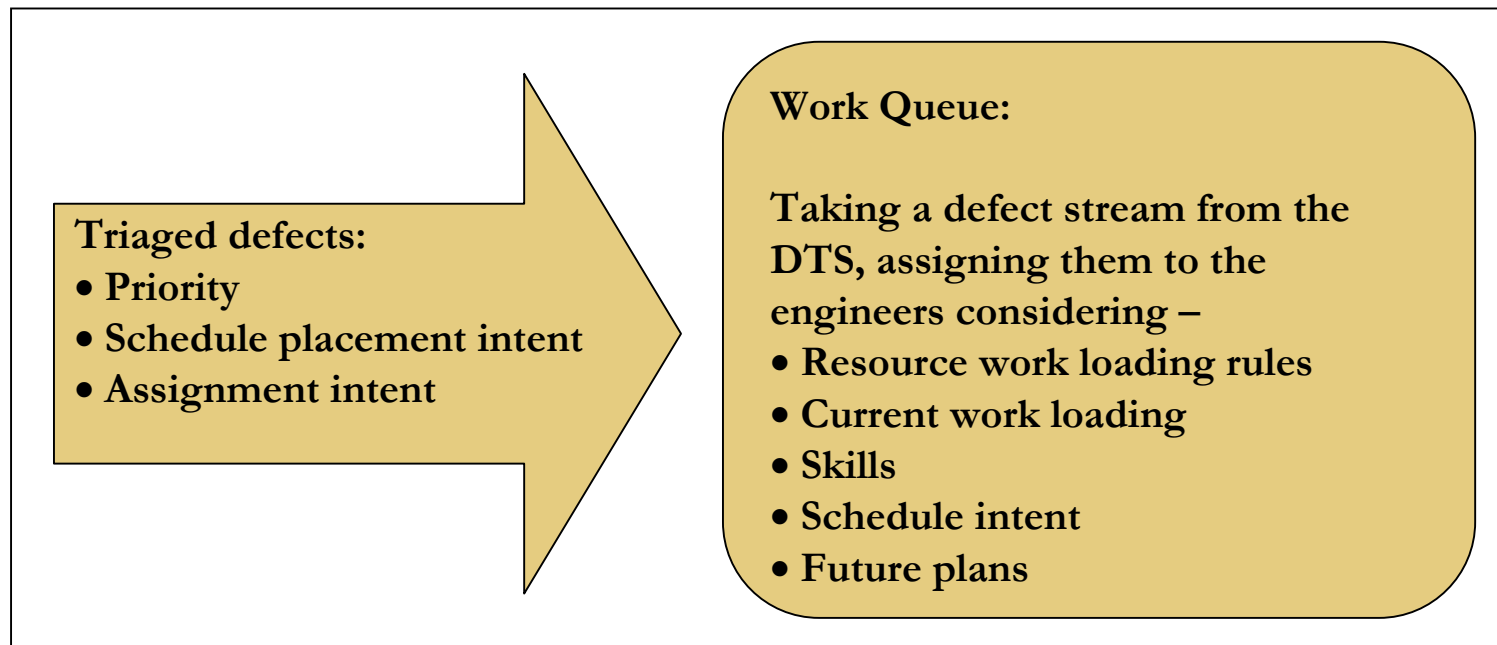
Directing Repairs

- 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

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



Problem – Those “Pesky” Defects

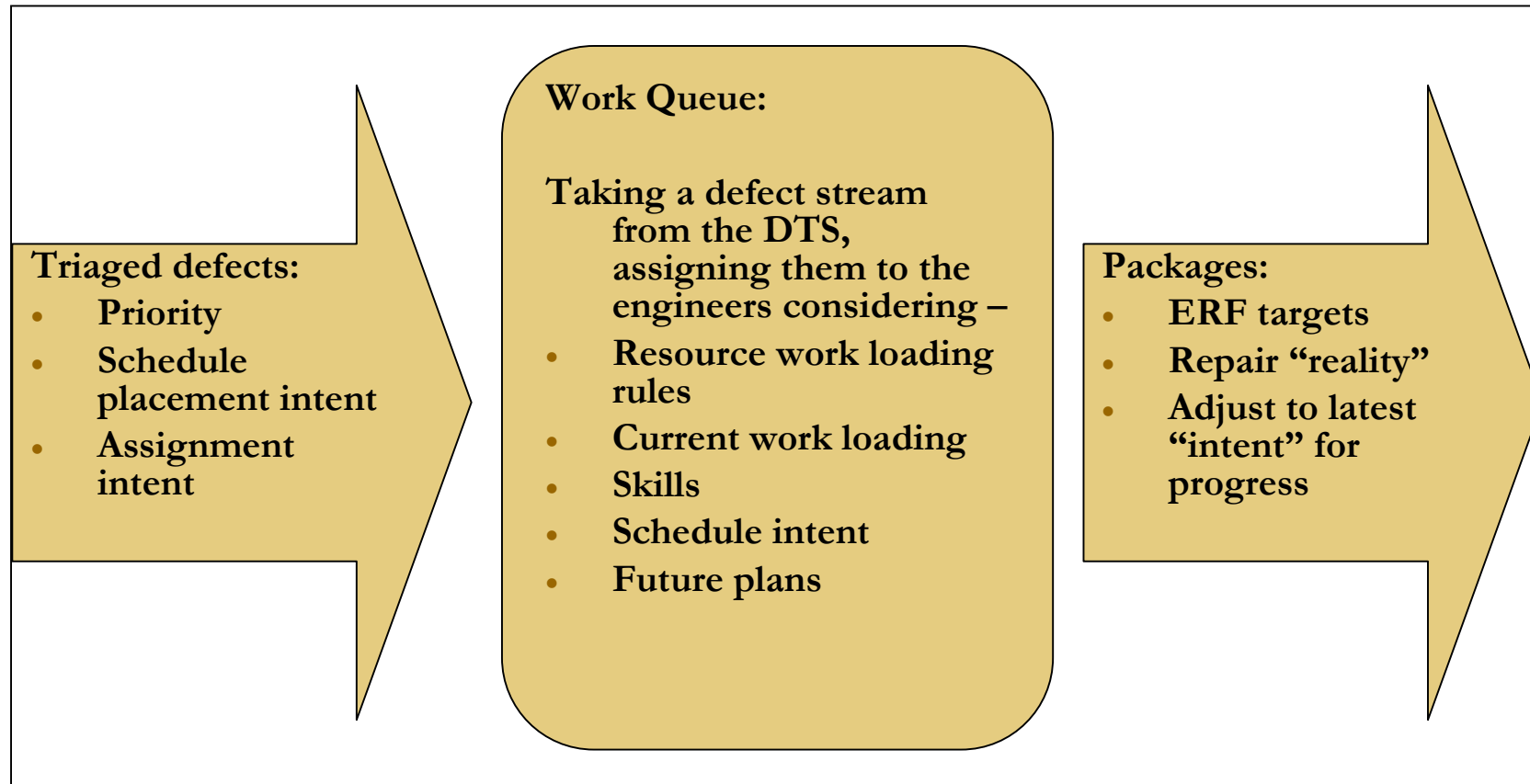
Directing Repairs

- **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



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...to release it!

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

- IEEE Std. 828-1998, IEEE Standard for Software Configuration Management Plans
- IEEE Std. 1042-1987 IEEE Guide to Software Configuration Management

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 |

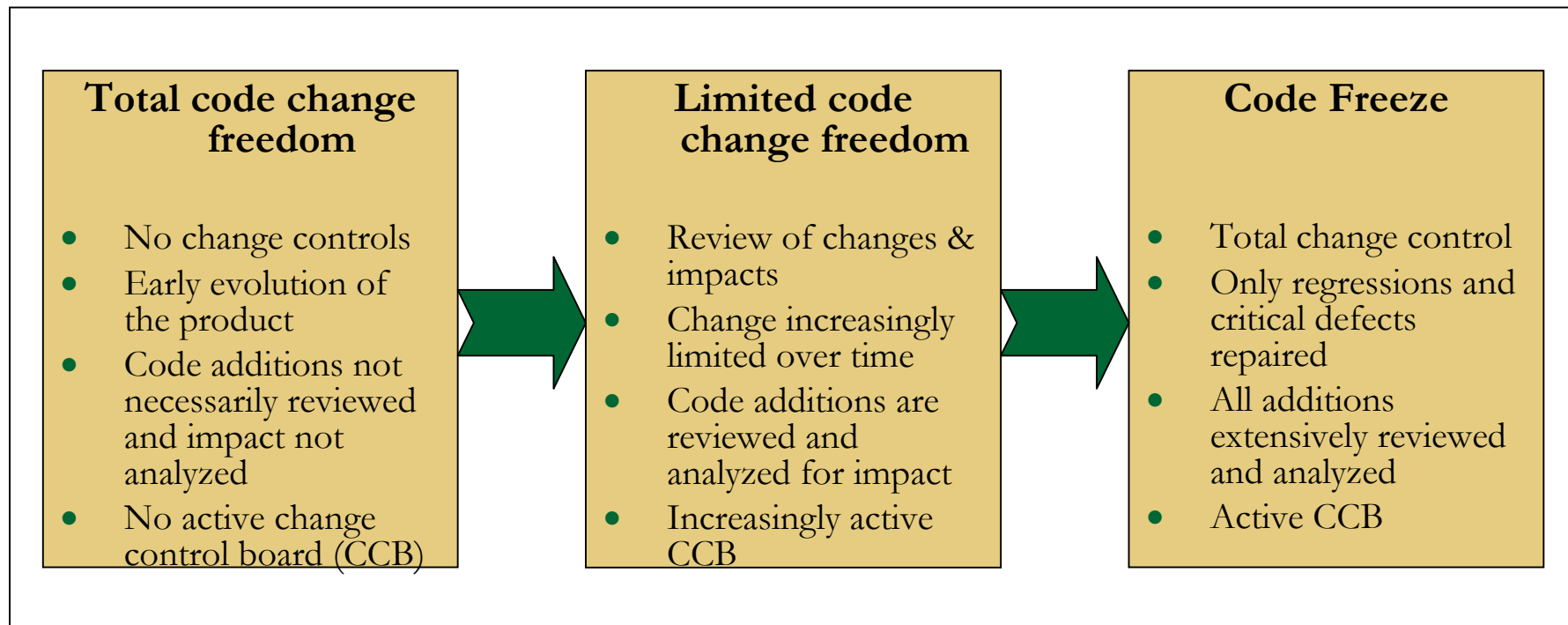
Strategy – Reducing Change

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 Code Freeze

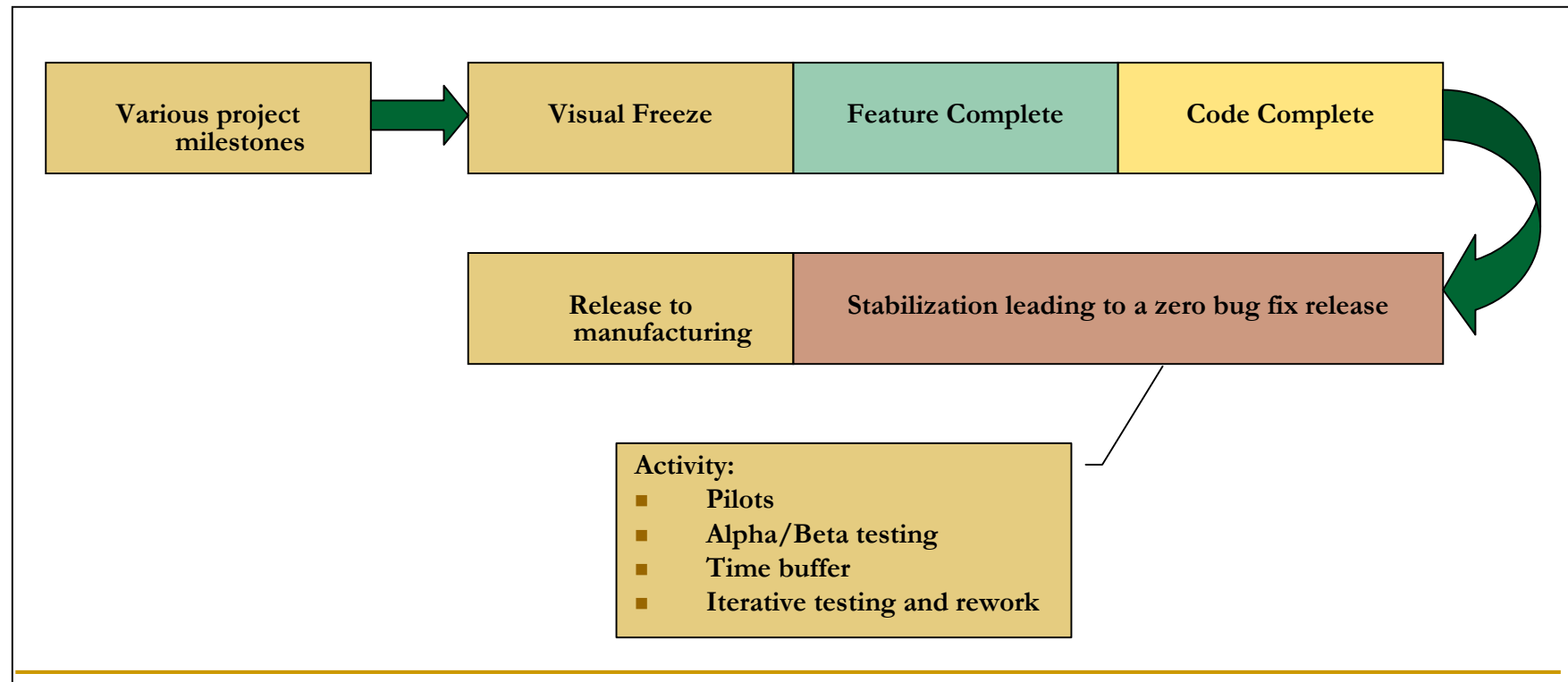
- Along with a well defined CCB structure, the team needs to have a notion of “code freeze”



Strategy – Reducing Change

Microsoft - Code Complete

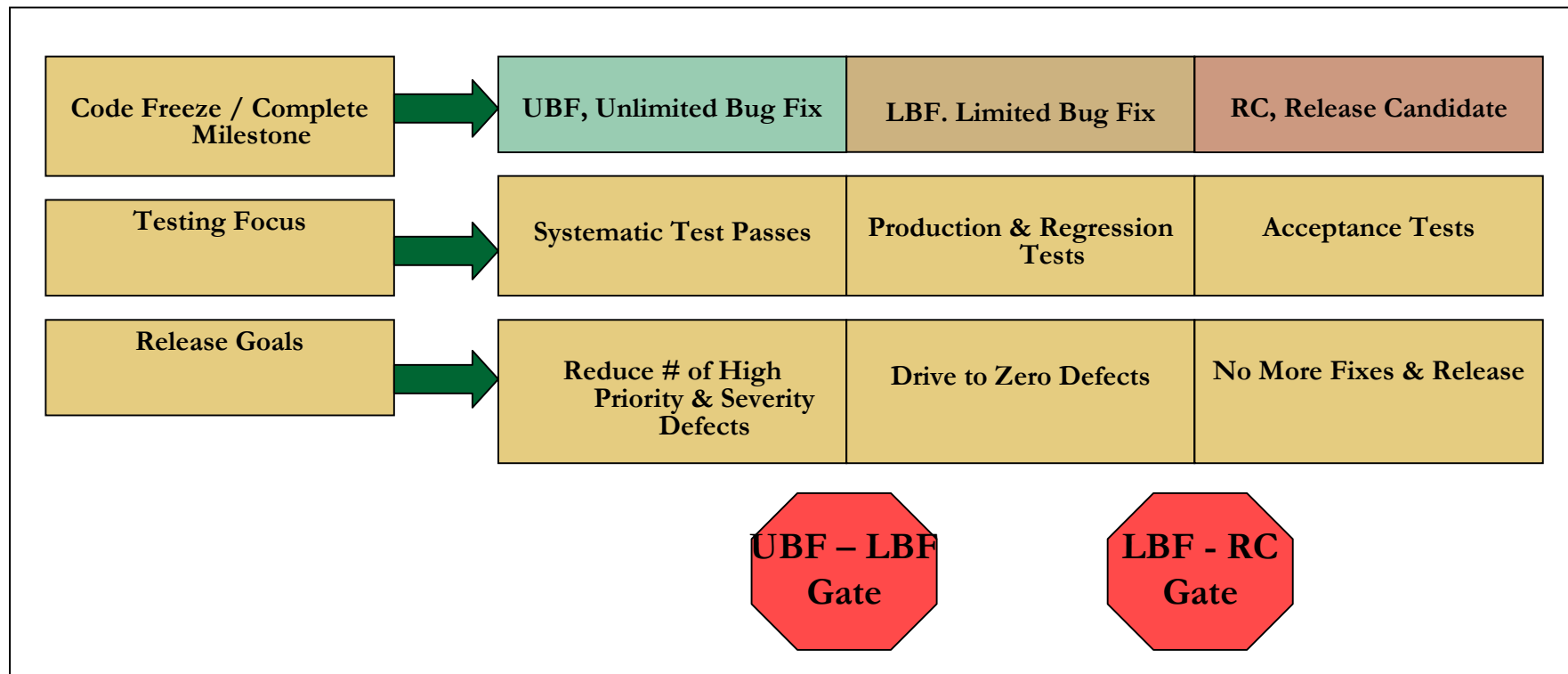
- Microsoft employs a “code complete” strategy as defined below –



Strategy – Reducing Change

Schoor – Bug Fix Gates

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



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

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

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

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-routine 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 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

- The “**Map**”
 - Endgame Release Framework
- The “**Goal**”
 - Release Criteria
- The “**Problem**”
 - Those Pesky Defects
- The “**Strategy**”
 - Reducing Change
- The “**Team**”
 - Do’s and Don’ts

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

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!

Questions?

Contact Info

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

*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.

