A Presentation for the
Rose City Software Process Improvement Network (SPIN)

A Defect-Free Mentality to Software Development

Rick Anderson

rick.d.anderson@tektronix.com

Senior Software Engineering Manager, Tektronix, Inc.

February 2007

Agenda

- Getting better each day
- Defect-free mentality
- Some Tektronix best practices

Being Competitive

- This past year:
  - Softball Team
    - 25th in the Western US
  - Two Basketball Teams
    - 38-10 and 9-1
  - Volleyball Team
    - Co-League Champions
- I keep the bar very high!

Youth Sports Philosophy

- Two rules:
  - Have fun
  - Get better every day
- So what the heck does this have to do with software development?
- Sports Psychology applies to Software Engineering

A Challenge

- Do not waste your time
  - Get better each day
- Find one thing from this presentation
- Embrace it with a passion
- Modify it to fit your culture
- Implement it in your organization
  - Start tomorrow morning at 8am

No Excuses

- Let’s get past these first
- On my teams, there are no excuses:
  - The sun was in my eyes
  - I didn’t know the play
  - I left my equipment at home
  - She was taller than me

Have you heard these?

- We've always done it that way…
- We don't have the time…
- That's not my area…
- Management won't let us…
- We know that's best practice but…
- We must accept responsibility

Tektronix PPRs

- 1980's era Post Project Review quotes:
  - The project was originally estimated to be 10KLOC turned out to be over 18KLOC
  - We sacrificed discipline to preserve schedule
  - There must be a process for changing requirements
  - We should base estimates on actual work content, not show dates or artificial deadlines
  - Estimated to take 42PM; turned out to be 80PM
  - The team realized they did not have a clear understanding of what the project should be
  - People have a tendency to conduct projects in the same way they conducted previous projects. Both successes and mistakes tend to be repeated.

Cobb's Paradox

"We know why projects fail, we know how to prevent their failure
--
so why do they still fail?"

Martin Cobb
Treasury Board of Canada Secretariat

Software Hall of Shame?

- Standish Group 2001 Extreme CHAOS Report
  - Only 28% of projects finish on-time and on-budget
  - 45% projects are over budget
  - 63% projects are over schedule
  - 23% projects deliver no code
  - 67% features are delivered
- This is not winning
- This is not getting better each day
- This is not acceptable

Insanity

- *"The definition of insanity is doing the same thing over and over and expecting different results"*

Benjamin Franklin

- We must all be insane!


I DON'T SUFFER
FROM INSANITY...
I ENJOY EVERY
MINUTE OF IT!

“Somebody has to do something, and it’s just incredibly pathetic that it has to be us.”

Jerry Garcia

Defect-Free Mentality

- Search Google and you will not find this phrase
  - That is actually sad
- Closest you will find is zero-defect mentality and defect-free software
- Not many people even talking about this:
  - Terry Colligan from Tenberry Software wrote an article, *"Nine Steps to Delivering Defect-Free Software"*
  - Niels Malotaux has the right mindset in this quote: *"The **goal** of testing should **never** be fault finding. It should be finding the **absence** of faults."*

Definition

- Defect-Free Mentality – It is a mindset or belief that we will remove all defects from our product at every phase of the lifecycle
  - This implies confidence
  - This translates to actions
- This is not a process
  - It is different
  - We don't normally think this way
  - It will take some getting use too

Defect-Free Mentality Characteristics

- Doing it right the first time
  - If you haven't got time to do it right, when are you going to find time to do it again?
- It is an expectation of high quality
- It is a sense of urgency that quality must come early and often
- It is every phase of your software development lifecycle being as good as it can be
- It should drive your decision making

Defect-Free Mentality

- Most software engineering organizations don't think or talk this way
  - Most haven't even had a discussion on this topic
- As you advance in sports, you get more mental training and less physical
- In Software Engineering, this is the mental side of the game
  - It is an attitude: You must believe it
  - It is communication: You must talk it
  - It is action: You must walk it
- You must believe defect-free software is possible

Defect-Free Mentality

- In sports, you cannot control your opponent, but you can control yourself and how you play
- It's not necessarily perfect software
  - Not that anyone knows what that is
- There will be defects, but you should find and fix them sooner
  - No defect slip between phases

Example #1

- Softball – last line of defense on ground balls
  - Infielders
  - Outfielders
- Software – last line of defense for defects
  - Software Engineering
  - Software Quality
  - Use DRE or Defects Found by SWE vs. SQE as metric

Example #2

- 30 years ago, when you “compiled” your code, you spent a lot of time making sure it was right
- Today, we use the compiler as a crutch
  - We expect it will find the defects for us
  - (If so, why do so many turn off warnings?)
- Maybe a silly example?

Defect-Free Mentality

- The Importance of Belief
- Goal Setting
- Visualization

Importance of Belief

- Defect-free mentality is believing you will remove defects from your product
  - Tiger Woods on the 18th green: "*I wonder if I'm going to sink this putt*"
  - Michael Jordan doesn't view missing a free throw as normal
- The typical engineer views defects as normal
  - But this is wrong!
  - Modest expectations tend to produce modest results

Importance of Belief

- The more you expect from a situation, the more you will achieve
- In sports, we talk about the Power of Positive Talk
- 1. UCSF researchers instructed depressed patients to change posture – stand tall, lift chin, keep eyes forward, smile, deepen breathing
  - Depression lifted

Importance of Belief

- 2. Mayo Clinic study shows optimists live longer than pessimists
- 3. Univ of Penn tracked 120 men who had their first heart attack
  - After 8 years of research, 80% of the pessimists had died of a second heart attack compared with 33% of the optimists

Goal Setting

- Many studies on goal setting
  - Deciding what is important
  - Separating important from irrelevant
  - Motivating yourself to achievement
  - Building self-confidence based on measured achievement of goals
- According to Kenneth Baum, a sports psychologist, on average goal setting improves performance by 8-16% just by setting goals
  - Can we not apply this to business?
- What is our common goal? Together we can achieve it

Visualization

- In athletics, we use visualization to see ourselves succeeding
- Visualization helps put the mind and body into a situation that hasn’t yet occurred
  - Basically thinking ahead
- What would a defect-free software environment look like?

One More Pitch

- Soviet Union study prior to 1980

  Olympic Games with elite athletes
  - Four training programs
    - #1 – 100% physical training
    - #2 – 75% physical, 25% mental
    - #3 – 50% physical, 50% mental
    - #4 – 25% physical, 75% mental
  - Greatest improvement was made by group #4 (followed by #3, then #2 and finally #1)
- Many other studies show similar results
- *"90% of the game is half mental"*, Yogi Berra

Implementation

- Ask challenging questions:
  - What does a defect-free environment look and feel like?
  - How does a defect-free mentality change the way we work?
- The time to shift to this mentality is the same as any process improvement
  - Introduce -> find champions -> reinforce -> bring along laggards -> culture

Leverage

- Attack those parts of the SDLC that have the most cost-benefit:
  - Unit Testing can find 10-50% of the defects (Jones 1986)
  - System Testing can find 20-60% of the defects (Jones 1986)
  - Requirements and Design reviews can find 30-70% of the defects (Myers 1979, Boehm 1987, Yourdon 1989)
  - Inspections can find 60-90% of the defects, and save 10-30% of the schedule (Gilb / Graham 1993 and numerous other studies)
    - Some have found this to be 20x-30x more efficient than testing (Russell 1991, Doolan 1992)

Leverage

- Attack those parts of the SDLC that have the most cost-benefit:
  - About 40-50% of the effort is avoidable rework (Boehm/Basili)
  - 80% of the defects comes from 20% of the code (see Boehm and Basili paper for seven different studies that found the same results)
  - Harlan Mills' Cleanroom software development process can reduce defect rates by 25-75%
  - Watts Humphrey's Personal Software Process (PSP) can reduce defects by 10:1
- Are we ignoring this data when we decide our continuous improvement projects each year?

Some Defect-Free Mentality Best Practices

- Schedule accuracy
- Mini-milestones
- Evaluation Checklist testing
- Automated testing by SQA group

About This List

- Books have been written on each of these topics
- In the time remaining, will share a few of my beliefs on these items
- Based on actual experiences at Tektronix
- Will attempt to highlight some keys to success
- Not in any particular order

Tektronix

- A few things you should know:
  - Leading maker of test and measurement equipment
    - Over $1 billion in revenues
    - 4,500 employees (2,200 in Beaverton)
    - Founded in 1946
    - Headquarters in Beaverton, Oregon
    - Engineering centers in Beaverton, Texas, England, Germany, Italy, Japan, China and India
  - Embedded hardware / software products
    - Very complex (5 million lines of code)
    - Not uncommon to have 15-25 programmable parts
    - Many custom ASICs
    - Many are Windows-based

Tektronix

- A few things you should know:
  - Each product line does things slightly differently
    - Performance Oscilloscopes Product Line
  - Most groups use an iterative model that has pieces of waterfall, spiral and extreme programming built-in
    - Tom Gilb / Niels Malotaux Evolutionary Development or Evo
  - More and more groups have a separate Software Quality Engineering (SQE) group
    - Responsible for testing product and software requirements
  - We don’t have it all figured out yet!

Schedule Accuracy

- Estimation – have a documented process and follow it
- Capture estimates in a document so you can:
  - Review these estimates at various phases of your lifecycle
  - Learn from your mistakes
  - Capture assumptions that were made
    - If these are violated, must re-schedule

Schedule Accuracy

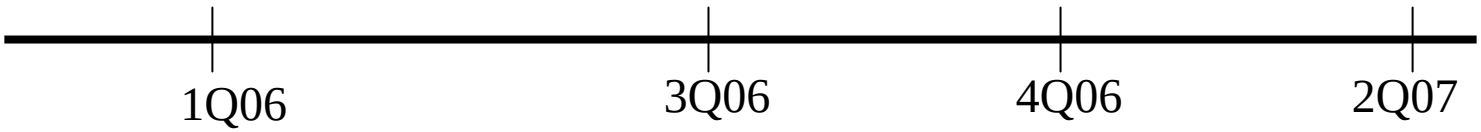
- 3Q2006 Standish Group research report showed that 64% of those interviewed felt they were moderately to poorly skilled in the area of estimating time and cost
  - Message: Train your teams!

Schedule Accuracy

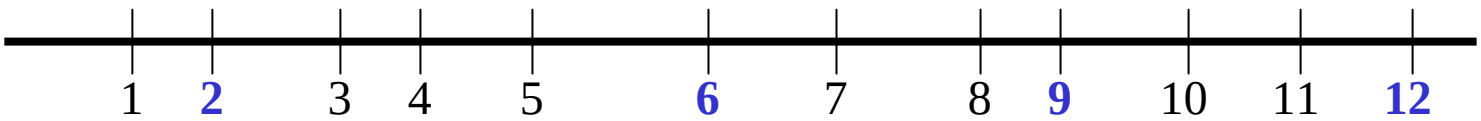
- Do:
  - Have best case and expected case schedules
    - We standardized on risk adjusted and non-risk adjusted
  - Do Risk Management and add risk tasks to schedule
  - Had stretch goal requirements/features that can be cut if you get behind
  - Schedule every task (holidays, vacation, reviews, testing and defect fixing, distractions, etc.)
  - Load engineers at 70%
    - Assume 30% of their time is email, meetings, helping others
    - This was a big change for us
    - Estimate assuming no interruptions and load in MS-Project @ 70%
  - Manage requirements and defects (CCB)

Mini-Milestones

- Most development methodologies have major milestones
  - Managerial review and approval gates
  - If you work on long projects, these might be every quarter or two



- Mini-milestones occur much more frequently
  - Goal is every month



Mini-Milestones

- Each milestone has a clear set of deliverables
  - Combination of SWE and Software Quality tasks
  - Can have stretch goals
  - Captured electronically and shared
  - Progress is tracked against this
  - Generally each team member affected by one deliverable
  - Goal is a “customer ready release” at the end of each milestone
    - Includes evaluation cycle by SWE and Software Quality
- Should have two mini-milestones planned in detail at a time; remaining mini-milestones have less detail, but should contain critical known tasks

Mini-Milestones

- Completion
  - Date driven (time-boxed; out of time)
  - Task driven (all tasks done)
- Our guideline is date driven
  - Exception – if dependent task outside of engineering development is holding-up critical engineering tasks
- Mini-milestone metrics
  - Days early or late
  - % tasks completed
- Celebration of accomplishments

Mini-Milestones

Project YP, FeatureZ Mini-Milestone Planned Release: 2/12/07	Assigned		Requirement	Customer	Notes
New probe cal procedure	Rick / Debbie	3	YP0051	Marketing	Doesn't need to pass
FeatureZ - embedded	Frank	1	YP0007 (part)	Marketing	
FeatureZ - UI	Doug	3	YP0007 (part)	Marketing	Excludes new control
FeatureA - UI	Rick	3	YP0010	Marketing	
Support UI operation of FeatureB	Mary	1	YP0003 (part)	UI team	
Probe tip selection	Fred	2	YP0004 (part)	Marketing	Interaction with group Z
Critical Field Defect 12345	Lisa	1	YP0042	Marketing	
MPK assignments	Lisa	3	YP0041 (part)	Marketing	Demux risk reduction
Horizontal model - constraints	Joe	3	YP0005 (part)	Marketing	
Stress - 1 million commands on SR, TS, HH	All	1		SPL	Average is 250K
Checklist testing	All	1		SPL	Target 2/8
Zero P1/P2 defects (id'd 2 wks prior to release)	All	2		SPL	2/2/07: 15 P1/P2s

Not complete	1
Partially complete	2
Complete	3
N/A	

Total Features:	14
Complete	6
% Complete	43%
Complete/Partial	8
% Complete/Partial	57%

Mini-Milestones

- Advantages of mini-milestones:
  - Focus us on getting most important things done first
  - Emphasis on finding and fixing defects early
    - Integrate early -> find defects early
  - More flexibility around prototype or beta releases
  - More time to react if off course
  - Puts a little more pressure on the team to deliver (more delivery points; easier to tell what's working and what is not)
- Challenges:
  - Needs full team involvement (ownership)
  - Must have measurable deliverables (done or not done decision is clear)
  - Tie short term mini-milestone tasks to long term project deliverables

Testing

- In sports, the more ways you have to score, the better
- In software engineering, the more different ways you test the product, the better
  - This is important
  - Sometimes we don’t talk about this enough
- At Tektronix we have over a dozen, unique testing types that are preformed by various groups

Testing

- UI Check Testing
- UI Stress Testing
- PI Stress Testing
- Functional/System Testing
- Accessory Testing
- Testing against standards and/or past products (compatibility testing)
- Performance Testing
- Benchmark Testing
- Evaluation Checklist Testing
- Unit Testing
- Integration Testing
- Smoke Testing
- Exploratory Testing
- Beta Testing
- Expert User Testing / Field Testing

Evaluation Checklist Testing

- Most formal testing type done by SWE
- Each SWE captures a list of tests in a document
  - It's a bulletized list (checklist style) versus a detailed test procedure
  - Generally 2-5 pages in length
  - Each SWE does their own area; they have internal knowledge of the code and this should be reflected in the checklists
    - Good SWEs start the list during design and coding
  - Stored in configuration management system and reusable

Evaluation Checklist Testing

- Capturing checklists takes one day
  - Generally happens at end of each iteration or spiral or about 33% of the way into the testing phase if waterfall model
  - Whole team does this at the same time
    - Important everyone stops coding or defect fixing and puts on their testing hat
- Second day is spent executing these manual tests
  - Checklists are modified as necessary
  - Defects are opened; usually see a nice spike in defect curve during these

Evaluation Checklist Testing

- Evaluation Checklists are used 2-3 more times during the project
- During next evaluation cycle:
  - You only allocate one day
  - Rotate the checklists between team members
    - So everyone is testing another area
  - Checklists are updated
    - Can add, subtract or modify tests
- Last cycle is done just prior to release to manufacturing

Evaluation Checklist Testing

- Several advantages:
  - Get SWEs to play testers
  - Formal method (structured; test cases are reusable)
  - Gives us an assessment of where the product is
  - Cross training of SWEs
  - Keeps ownership of quality with the SWEs
- Disadvantages:
  - Checklists can become long after a while
  - Can only do it 3..4 times per project, as it’s manual testing

Software Quality Team

- If you have a defect-free mentality, do you need an SQA team?
  - No! So the SWE teams goal is to put the SQE team out of business
  - Yes! Until the SWE team can prove themselves, better have someone representing the customer quality side
    - Our SWE teams haven't put our SQE teams out of work yet…

Software Quality Team

- Having a dedicated Software Quality Engineering team has been critical to our success at Tektronix
- Groups which haven't had a dedicated team in the past are now forming them
- We typically have a 1:8 ratio
  - Doesn't mean you should use this number
- SQEs find 25% of the defects on average
  - Good cost/benefit (12% of the team finds 25% of the defects)

Software Quality Team

- Important characteristics:
  - Equal to SWEs
  - Software Quality Leader on “core team”
  - Separate identity, but also part of SWE team
  - Separate manager can be good, but not required
  - Plays critical role in software process improvement initiatives
  - Heavy focus on automated testing (so they are SWEs that write code to test the product instead of making the product work)
  - Start working on the project at the beginning

Other Beliefs

- More unit testing
  - By the SWE team
  - Using static and dynamic code analysis tools
- Daily builds and smoke tests
  - Rarely have failed builds
  - Constantly checking quality level
- Peer Reviews
  - Find 60% of all defects
  - Have a process and follow it

Summary

- Stop the insanity and start getting better each day
- Don’t accept the typical excuses
- Embrace a Defect-Free Mentality
  - It’s an attitude
- Remember my challenge to you – take away something from this presentation and act on it

“Never be afraid to try something new. Remember, amateurs built the ark. Professionals built the Titanic.”

Questions

&

Answers

Unit Tests

- Unit Testing has many definitions
  - Testing a separately testable element, a logically separate part of a computer program, a module, a function, a component, a class
- Does not matter what SDLC you use
- One of the best methods to remove defects from your code
  - Study by Caper Jones found it can remove 10-50% of the defects
  - Study by Thayer and Lipow showed unit testing with path coverage and parameter passing tests can remove 73% of the defects

Unit Tests

- Static tasks:
  - Requirements traceability (map requirements to units and combine with risk management to help focus unit testing activity)
  - Code review
  - Structural analysis (static code and complexity analysis tools)
- Dynamic tasks:
  - White box testing
  - Coverage analysis (path, branch, statement)
  - Execution analysis (memory allocation, performance)

Unit Tests

- There are a number of tools available for static analysis, white box testing and coverage analysis
  - Many are free
  - They are getting very good at finding valid defects
- Excuses:
  - It takes a lot of time: yes it does, but it is actually more efficient to find it now versus in system testing

Unit Tests

- Applied Software Measurement by Capers Jones (1991) showed that the average cost per defect was best during unit testing
  - Unit Testing - 3.25 hours per defect
  - Integration Testing – 6.25 hours per defect
  - System Testing – 11.5 hours per defect
- Why do we ignore the data? Insanity!

Daily Builds / Smoke Tests

- Typical project has multiple pieces
  - UI + Embedded
- Typical project has multiple sites
  - US + India
- Typical project has many SWEs
  - 20-25 not uncommon

Daily Builds / Smoke Tests

- Excellent luck with doing automated daily builds using simple shell scripts
- Generally happens “between shifts”
  - Night in US, morning in India
- Install package is done early in the project
- Most upgrade to a new build every other day
- Keep the last dozen builds or so
- Bad builds are rare (< 10% of the time)

Daily Builds / Smoke Tests

- Combined the daily build process with some simple tests to verify the most important product features work as expected
  - We call these smoke tests
- Pick some automated unit tests, system tests and toss in a little stress
  - Starts as a few minutes of tests and grows to a few hours as project goes on
- Entire SWE team receives email on status of build
  - How the build did
  - How the tests did
- If build or tests fail, a defect is opened and they are usually worked on immediately

Daily Builds / Smoke Tests

- Daily Builds and Smoke Tests encourage a defect-free mentality
  - If you have a bad build, some SWE is in the spotlight in front of his/her peers
  - Don’t allow bad code / critical defects to go unnoticed for long
  - Encourages team to get a solid core of the system running early in the project and then build on this every day
    - Getting better every day!

Peer Reviews

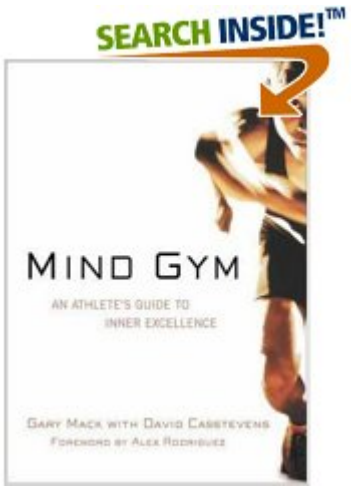
- The Instant Replay of the sports world!
- Walkthroughs, formal technical reviews, code reading, inspections
- Myers, Boehm and Yourdon all found reviews can find between 30-70% of the defects
- Gilb and Graham found 60-90%
- A study at NASA's Software Engineering Laboratory found that code reading detected about twice as many defects per hour of effort as testing (Card 1987)

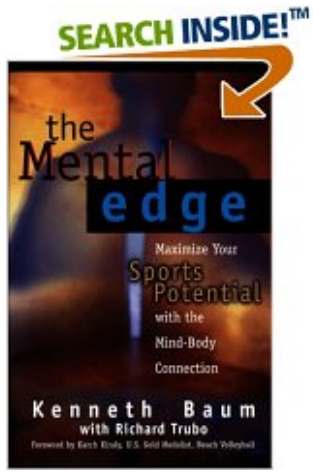
Peer Reviews

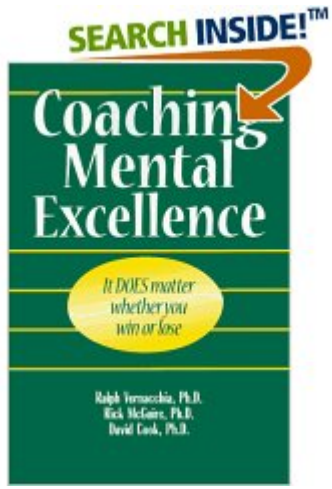
- Took us two years to figure out what really worked
- Currently have a process that includes informal and formal technical reviews
  - 90% of them are informal
- Heavy review of requirements, architecture/design, user interfaces, project plans and test plans
  - Very few code reviews
- Have standardized review meeting notices, review commentary forms and review meeting minutes

Mental Sports Bibliography

- “Mind Gym” by Gary Mack (2001), McGraw-Hill.
- “The Mental Edge” by Kenneth Baum (1999), Perigee Books.
- “Coaching Mental Excellence” by Vernacchia, McGuire and Cook (1996), Warde Publishers, Inc.
- http://www.mindtools.com

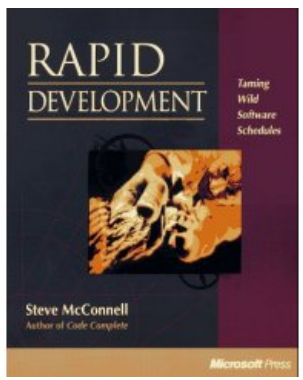


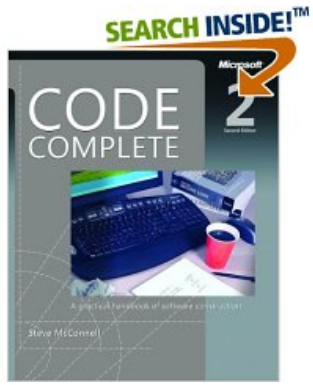




General Bibliography

- Niels Malotaux, http://www.malotaux.nl/nrm/English/
- "A Software Technology Evaluation Program“, David N. Card, *Information And Software Technology*, v. 29, no. 6, July/August 1987, pp. 291-300
- “Nine Steps to Delivering Defect-Free Software” by Terence M Colligan (1998), http://www.tenberry.com/errfree/steps.htm
- “Rapid Development”, Steve McConnell, Microsoft Press, 1996
- “Code Complete”, Steve McConnell, Microsoft Press, 2004 (2nd Edition)





General Bibliography

- Tom Gilb, Evo Slides, http://www.gilb.com/Download/EVO-Slides.pdf
- "Software Defect Reduction Top-10 List", Barry Boehm and Victor Basili, http://www.cebase.org/www/AboutCebase/News/top-10-defects.html
- Standish Group 1994 CHAOS Report, http://www.standishgroup.com/sample_research/chaos_1994_1.php
- 2001 CHOAS Update data, http://www.softwaremag.com/archive/2001feb/CollaborativeMgt.html

Unit Testing Bibliography

- “Unit Testing: Approaches, Tools, and Traps”, Tom Rooker, Software Quality Professional, vol 8, no 3, June 2006
- “Why Bother to Unit Test” http://www.ipl.com/pdf/p0828.pdf
- http://www.mobilein.com/WhitePaperonUnitTesting.pdf

Metric Bibliography

- Karl Wiegers, “A Software Metrics Primer”, http://www.processimpact.com/articles/metrics_primer.html
- V. Basili, G. Caldiera, and H.D. Rombach, “Goal Question Metric Approach”, Encyclopedia of Software Engineering, pp. 528-532, John Wiley & Sons, Inc., 1994

  http://www.cs.umd.edu/~basili/publications/technical/T86.pdf
- “Applied Software Measurement”, Capers Jones (1991) McGraw-Hill