The Challenge of “GOOD ENOUGH” Software

October 6th 2003

Presented by: M Usman Shakil

The Game Plan

- What is Software Quality?
- How much Software Quality?
- The Utilitarian Model
- The Double-Cycle Project Model
- Key Process Ideas (KPIs) of Good Enough Software
- Good Enough to be the Best

What is Software Quality? (The Model)

The Ideal Picture:

- **Determining & Achieving requirements**
- **Adequately staffed & enough time to do the work**
- **Quality Assurance presence in every phase of the development process, from requirements definition to final testing.**
- **Management's Commitment to quality *on the unquestioned faith that it is always worth whatever it will cost*.**

The Real Picture:

Requirements Shift and Waver
Perpetually Understaffed and Behind Schedule
Software Quality Assurance is often a Fancy word for Ad-hoc testing
First one to market wins the most market share
Management more interested in making more money than in the niceties and the necessities of the Software Engineering

What is Software Quality?

(Various Views)

Manufacturing View

Quality is Correctness, Conformance to Specifications

Problem:
Perfect Products that satisfy no one

Aesthetic View

Quality is Elegance, an ineffable experience of Goodness

Problem:
Perfectionists & Underachievers

Customer View

Quality is Fitness of use, whatever satisfies me

Problem:
Chasing will-o'-the-wisp

How much Software Quality?

(Measures)

- **Some Measurable Factors**

  Functionality, Reliability, Usability, Efficiency, Maintainability, Portability

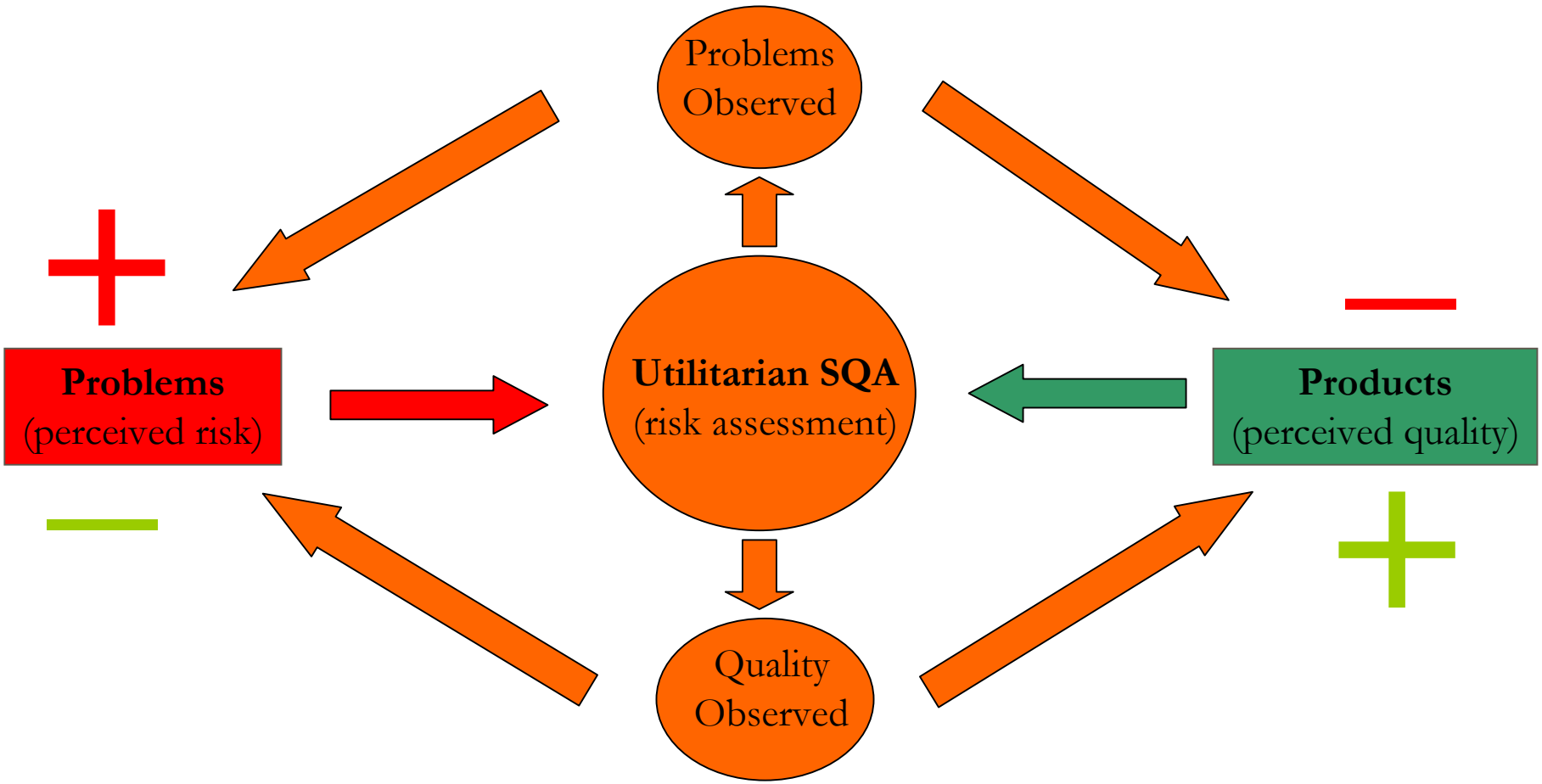
- **BUT !!**
  - **Some factors are more important or detract from others**
  - **There is no straight forward approach to measure these**
  - **No matter how each of these factors is taken into account a single BUG may negate everything else that is working right**
  - **FACT: "The Client" can never know the Quality of the Project…**
  - **They make Perceptions about Quality based on skill level, past experience and profile of use and we can not control any of these basis**

How much Software Quality?

- **SINCE**
  - Creating products of the best possible quality is very expensive
  - The Client may not even notice the difference between the best possible quality and pretty good quality
- **3 Critical Questions**
  - **How much of which quality factor be adequate?**
  - **How do we measure it adequately?**
  - **How do we control it adequately?**
- **The Solution**
  - **Instead of creating a universal metric of quality and then optimizing it, rather consider the problems directly what quality is supposed to solve**

The Utilitarian Model

Quality is the optimum set of solutions to a given set of problems



The Utilitarian Model (The Mechanics)

- **Predict, measure and control the Consequences of**
  - **Employing the Product:** Lies within the Product Quality
  - **Creating the Product:** Lies within the Process, Staff and Resources
- **Operate on a Chosen Metric**
  - For example, we might decide that a reasonable quality metric is the number of known defects in the Product

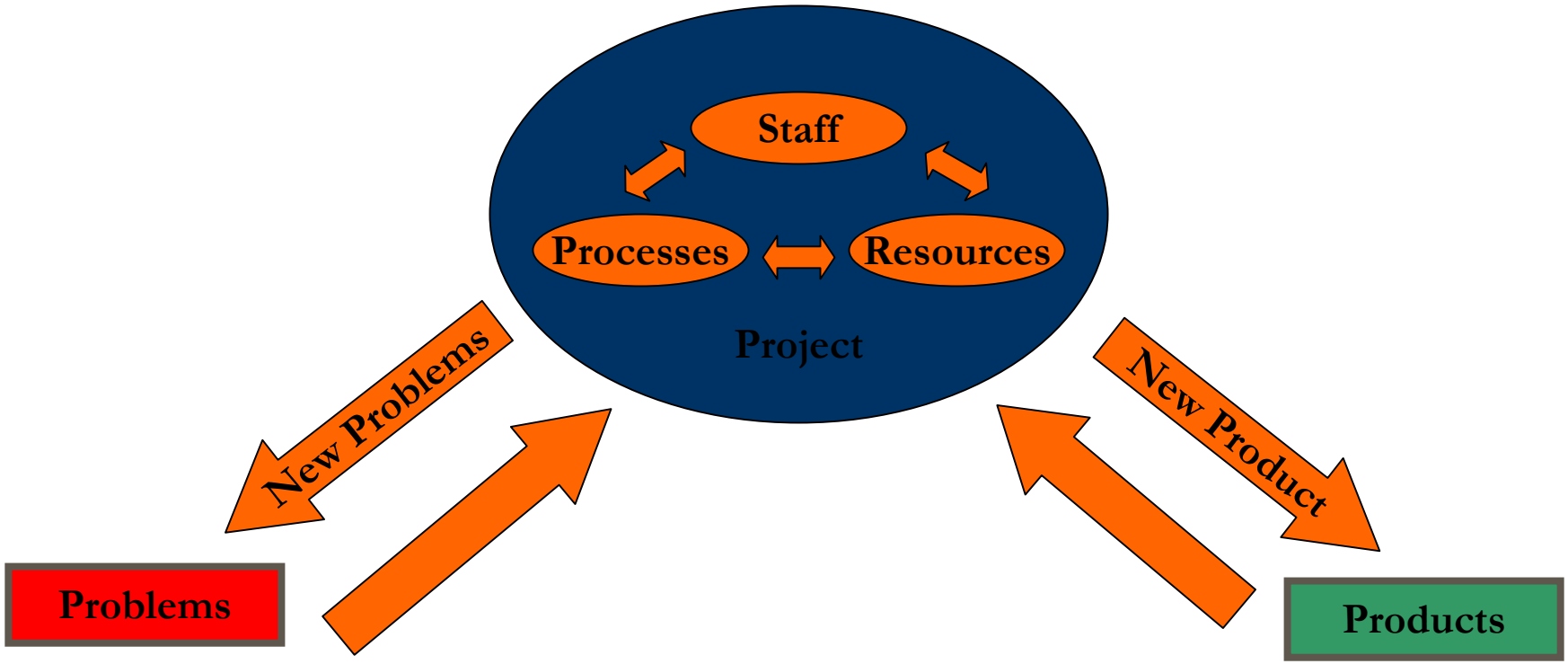
    Examine the consequences of each problem, and decide on a case-by-case basis which are important to fix. The quality metric would then either take care of itself or else become irrelevant.

It isn't the number of bugs that matters, it's the effect of

The Utilitarian Model

- **Problems**
  - We can cut too many corners while studying the problem trade-offs and matching them with appropriate processes
  - We lose sight of the full spectrum of the product, project, and customer if we are focusing on one chosen metric, however this can be prevented by working on the consequences side
- **FACT**
  - **We all use utilitarian approach**
  - **The issue is "How effectively we access and control risk".**
    We can do better by admitting what we are doing and do it directly rather than playing around with slogans like "Quality without compromises"

The Double-Cycle Project Model



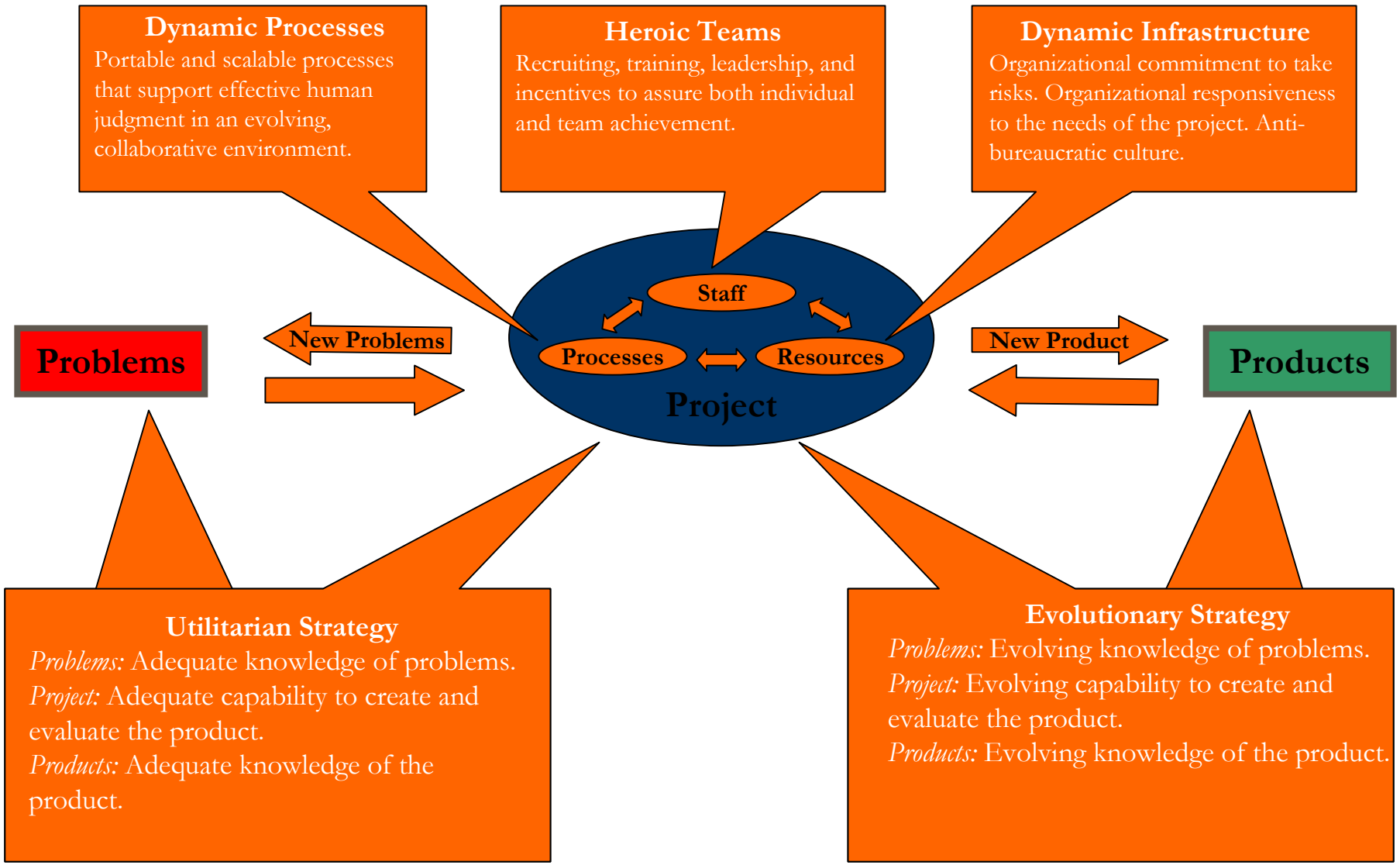
The Double-Cycle Project Model

- Problems
  - Motivators of the Projects
  - To do a project well enough is to be left with an acceptable set of problems at the end
- Products
  - Total output of the Project
  - Inverse relationship between Product quality and problems
- Project
  - The entire means by which risks are managed and products are created
  - Start with nothing but problems, and we finish with a new more livable set of problems and a new set of products and by-products.
  - The "good" of a project should be judged by the total output of all that transpires: the problems, solutions, and resulting capability.

The Double-Cycle Project Model

- **Project (contd.)**
  - **Staff**
    - Agents that solve all hard problems
    - Most critical and versatile part of the Project
  - **Resources**
    - Any thing that money can buy and support the Project staff
    - Can create problems or contribute directly to Products
  - **Processes**
    - Patterns for solving Problems, the result of problem-solving is the **Product**
    - Distinct from Staff and Resources
    - They are just concepts and manifest solutions in concert with staff and resources

5 KPI s of GOOD ENOUGH Software



5 KPI s of GOOD ENOUGH Software (contd.)

- **Utilitarian Strategy**
  - The art of qualitatively analyzing and maximizing net positive consequences in an ambiguous situation
  - There is no right or wrong project estimate, just an integrated, dynamic estimation process
  - Efficient software development necessitates risk taking: The real question is whether we take *calculated* risks or accidental risks.
  - One way to avoid accidents is through a habit of integrated, structured risk management.

5 KPI s of GOOD ENOUGH Software (contd.)

- **Evolutionary Strategy**
  - **On the Project Level:** Ongoing process education, experimentation and adjustment, rather than clinging to a notion of the "One Right Way to develop software"
  - **On the Product Level:** Planning and building the product in layers, which allows concurrent design, coding, and testing.
  - **On the Problem Level:** Keeping track of history, and learning about failure and success over time.
  - **Elements**
    - Don't even try to plan everything up front.
    - Converge on good enough in successive, self-contained stages.
    - Integrate early and often.
    - Encourage disciplined evolution of feature set and schedule over the course of the project.
    - Salvage, reuse, or purchase components where feasible.
    - Record and review your experience.

5 KPI s of GOOD ENOUGH Software (contd.)

- **Heroic Teams**
  - Ordinary skillful people working in effective collaboration
  - Programmers must exercise initiative
  - Bored People don’t work hard. If there is an exciting environment that fosters responsible heroism, good software will follow
- **Dynamic Infrastructure**
  - Upper management pays attention to projects.
  - Upper management pays attention to the market.
  - The organization identifies and resolves conflicts between projects.
  - In conflicts between projects and organizational bureaucracy, projects win.
  - Project experience is incorporated into the organizational memory

5 KPI s of GOOD ENOUGH Software (contd.)

- **Dynamic Processes**
  - **"clarify and delegate" strategy**
  - **Portability**

    How the process lends itself to being carried into meetings, shared with others, and applied to new problems.
  - **Scalability**

    How readily the process may be expanded or contracted in scope. A highly scalable process is one that can be operated by one person, manually, or by a hundred people, with tool support, without dramatic redesign.
  - **Durability**

    How well the process tolerates neglect and misuse.

GOOD ENOUGH to be the Best

- **The Guiding Principle**

  "Solve the Problem"

- **If we**
  - **develop and empower teams who**
  - **use dynamic processes within a dynamic infrastructure to**
  - **employ utilitarian and evolutionary strategies, then we will produce good enough software.**

Good enough, even, to be the best.

Reference

- James Bach
  Chief Scientist
  Software Testing Laboratories
  http://www.satisfice.com/articles/gooden2.pdf