

Building Accurate Schedules from Software Requirements

Software Quality Consulting Inc.

Steven R. Rakitin
President

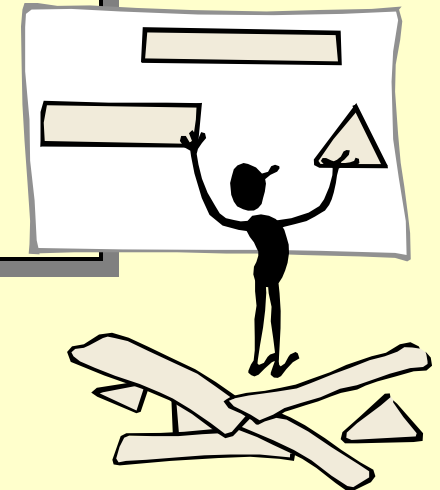
- Consulting
- Training
- Auditing

Phone: 508.529.4282
Fax: 508.529.7799

www.swqual.com
info@swqual.com

Agenda

- **About Requirements**
 - **Ambiguity**
 - **Tools to Reduce Ambiguity**
- **Estimating and Scheduling Best Practices**
 - **Why Most Estimates and Schedules Are Wrong**
 - **Yellow Sticky Method**
- **Summary**

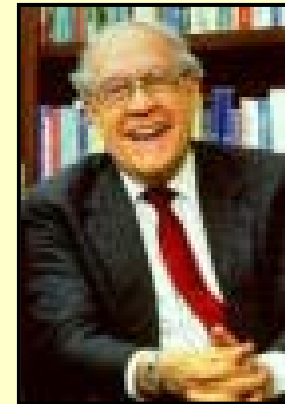


About Requirements

- **Why are requirements so important?**
- **Where do requirements come from?**
- **Why are requirements hard to write?**

Why are requirements so important?

“The hardest single part of building a software system is deciding precisely what to build. No other part of the conceptual work is as difficult as establishing the detailed technical requirements, including all the interfaces to people, to machines, and to other software systems. No other part of the work so cripples the resulting system if done wrong. No other part of the system is more difficult to rectify later.”



Brooks, F., "No Silver Bullet: Essence and Accidents of Software Engineering", *IEEE Computer*, Vol. 20, No. 4, April 1987, p. 10-19.

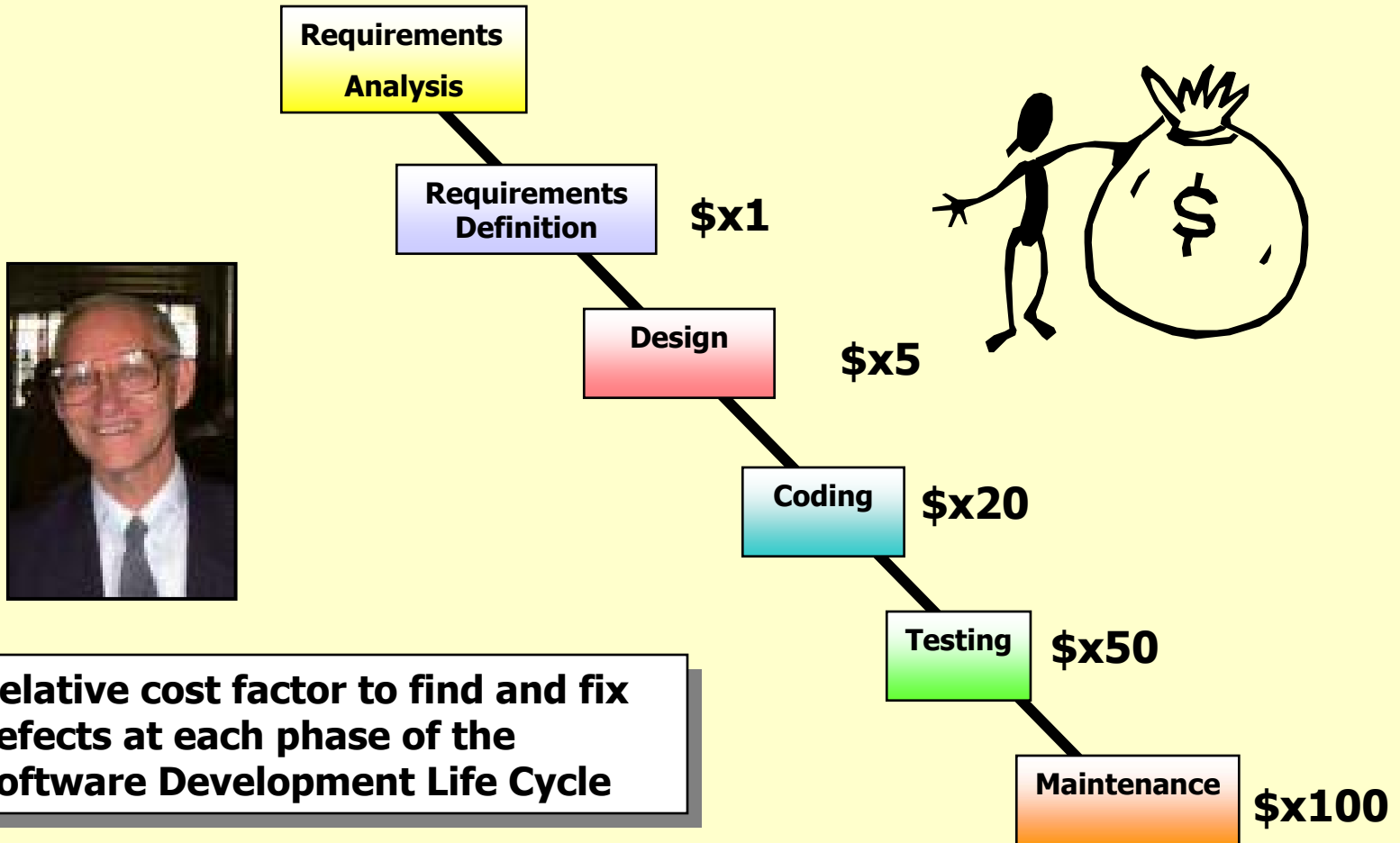
Why are requirements so important?



“A significant and important aspect of requirements errors is that if they are not prevented or removed, they tend to flow downstream into design, code, and user manuals. Historically, errors which originate in requirements tend to be the most expensive and troublesome to eliminate later.”

Jones, C., Software Quality: Analysis and Guidelines for Success, International Thomson Computer Press, 1997.

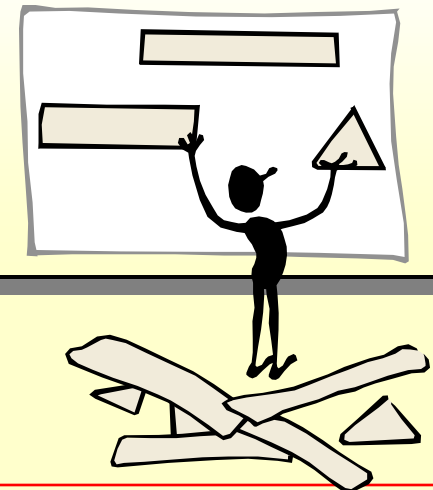
Why are requirements so important?



Boehm, B., *Software Engineering Economics*, Prentice-Hall, 1981

Why are requirements so important?

- Many projects are **delivered late with fewer features than promised**
- Often, inability to deliver what was promised, on time is directly related to **poorly written requirements and uncontrolled changes to requirements**
- Accurate estimates and schedules can **only** be developed from **complete, concise, unambiguous requirements**



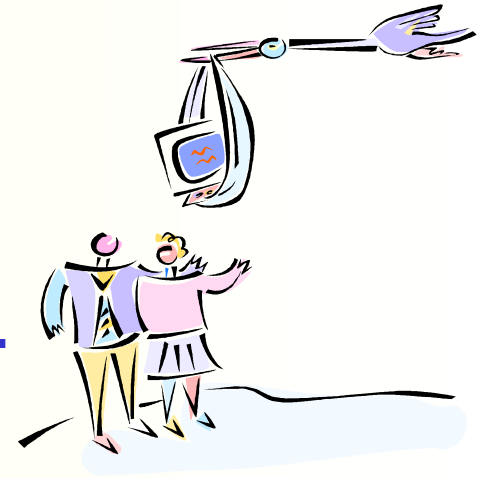
About Requirements

- Why are requirements so important?
- **Where do requirements come from?**
- Why are requirements hard to write?

Where do requirements come from?

- **Some sources:**

- **Product Marketing**
- **Marketing surveys and user feedback**
- **Focus Groups and discussions with users**
- **Problem reports from current products**
- **Descriptions of competitor's products**
- **Descriptions of existing products**
- **Observing users at work**
- **Enhancement requests**
- **Staff**
- **Requirements Elicitation Techniques...**

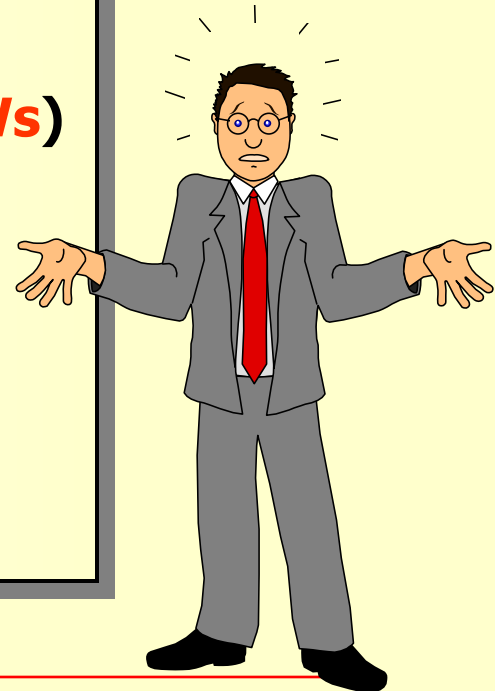


About Requirements

- Why are requirements so important?
- Where do requirements come from?
- **Why are requirements hard to write?**

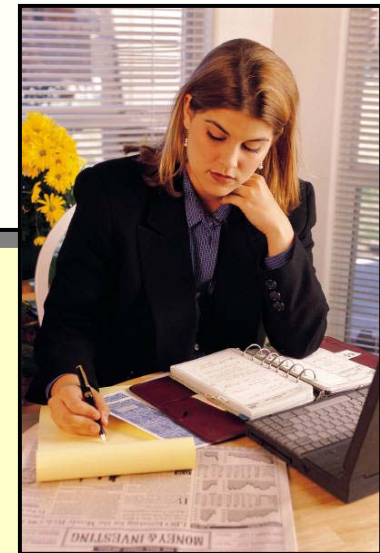
Why are requirements hard to write?

- Requirements written in **English** are inherently **vague** and **imprecise**...
- We **rarely train** people in how to write good requirements
- We often have trouble separating requirements (**WHATs**) from design (**HOWs**)
- Impact of poorly written or non-existent requirements not understood...
- **Misconception** - spending time writing requirements delays product release...



Why are requirements hard to write?

- In your organization, what percent of people have **excellent writing skills?**
- Of those, how many understand the **intricacies of writing software requirements?**
- Of those, how many are in a position where **writing requirements is part of their job?**



Try this exercise...

List at least 5 **nouns** and 5 **verbs** that are specific to each group



Users



Developers



Stakeholders



QA

Ambiguity

- **Some Common Problems**
- **Tools to Reduce Ambiguity**

Some Common Problems

- **Errors of Omission...**
- **Errors of Commission...**
- **Errors of Clarity...**
- **Errors of Understanding...**



Some Common Problems

- **Errors of Omission**

- **Important information left out or not stated:**

What was stated	What was intended
"..the application shall provide cost estimates."	"The application shall predict size for: specifications, source code, and user manuals; software staffing for technical employees and management; software schedules from requirements through delivery, and software cost estimates for all software development activities."

- **"We **assumed** you understood we always do it that way..."**

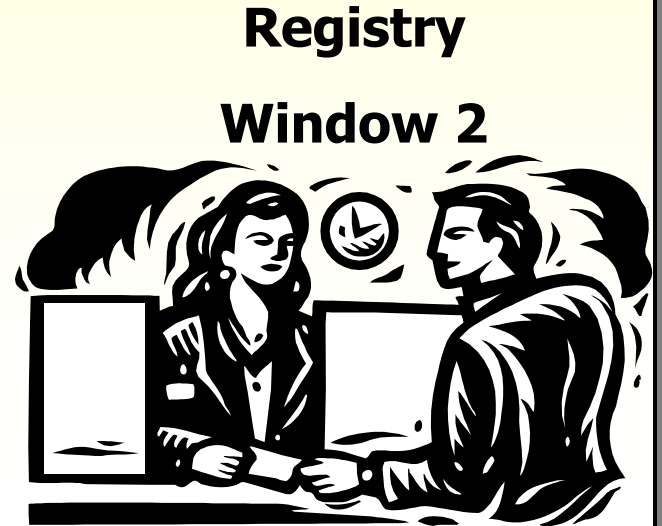
Some Common Problems

- **Errors of Commission**

- **Information is wrong or contradictory or both:**

In the requirements spec for a motor vehicle registration system, a requirement stated that **proof of valid insurance must be provided before issuing a vehicle registration.**

In another section of the same spec, a requirement stated that **vehicle registration information was required before issuing a proof of insurance certificate.**



Some Common Problems

- **Errors of Clarity**

- **Requirements stated in ways that lead to confusion:**

- **What was written:**

“Commanded pre-task calibration updates offset coefficients only, for the corresponding calibration table.”

- **What was intended:**

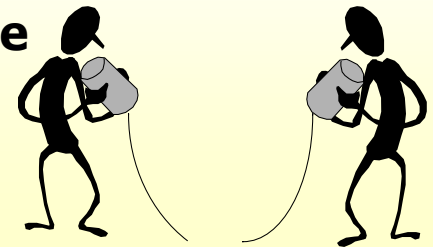
“Commanded pre-task calibration updates only the offset coefficients for the corresponding calibration table.”



Some Common Problems

- **Errors of Understanding**

- “We need a social process that [helps] get the words off of the paper and into our heads.”
- Fundamental problem in writing requirements is **ambiguity**.
- **Ambiguity** leads to **errors of understanding**...
- People **internalize requirements, apply their own definitions of words** and create their own vision...
- We need **processes and tools** to help ensure that **everyone’s vision is the same**.



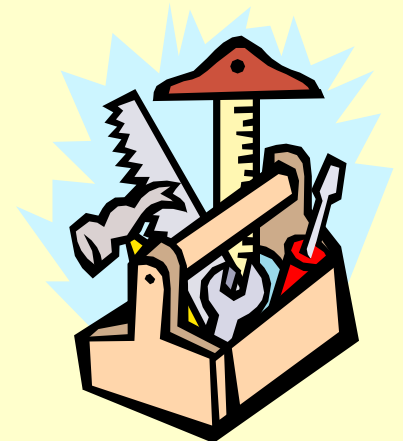
Gause, D. and Weingberg, G., *Are Your Lights On? How to Figure Out What the Problem Really Is*, Dorset House, 1990.

Ambiguity

- **Some Common Problems**
- **Tools to Reduce Ambiguity**

Tools to Reduce Ambiguity

- Use effective **requirements elicitation** techniques...
- Write requirements with an **audience** in mind...
- Use **simple techniques** to reduce ambiguity
- Review requirements from **different perspectives**...



Tools to Reduce Ambiguity

Requirements Elicitation:

- First step in bridging gap between **problem domain** and **possible solutions**
- Draws on **available sources of information** that describe problem domain
- Involving **customers** significantly increases likelihood of **success**

Examples:

- **Brainstorming**
- **Storyboards**
- **Use Case Diagrams**
- **Rapid Prototyping**



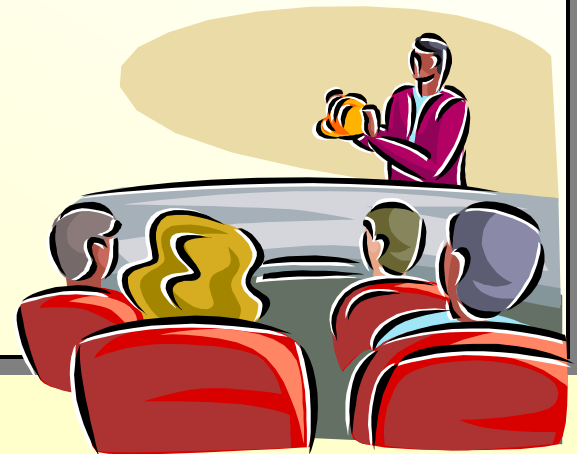
Tools to Reduce Ambiguity

Write requirements with an audience in mind

- **Who will be reading the requirements?**

- **Project Manager**
- **Developers**
- **QA staff**
- **Technical Writers**
- **Product Marketing**
- **Training**
- **Support**
- **Others?**

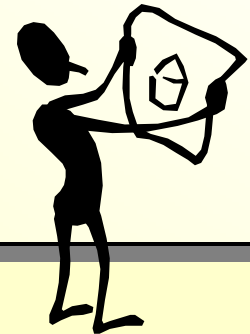
Ask your audience to identify the kinds of information they need to do their job



Tools to Reduce Ambiguity

Some simple techniques to reduce ambiguity:

- Use **pictures** to illustrate **work flow** and **functionality...**
- Create a **glossary** and **use terms consistently**
- Every requirement clearly **identified** and **numbered**
 - **“The software shall...”**
- Use short sentences avg. < **20 words per sentence**
- **Avoid using ambiguous words:**
 - **could, would, should, might, may, user-friendly, etc...**
- Use a **good document outline...**
 - **IEEE STD 830...**

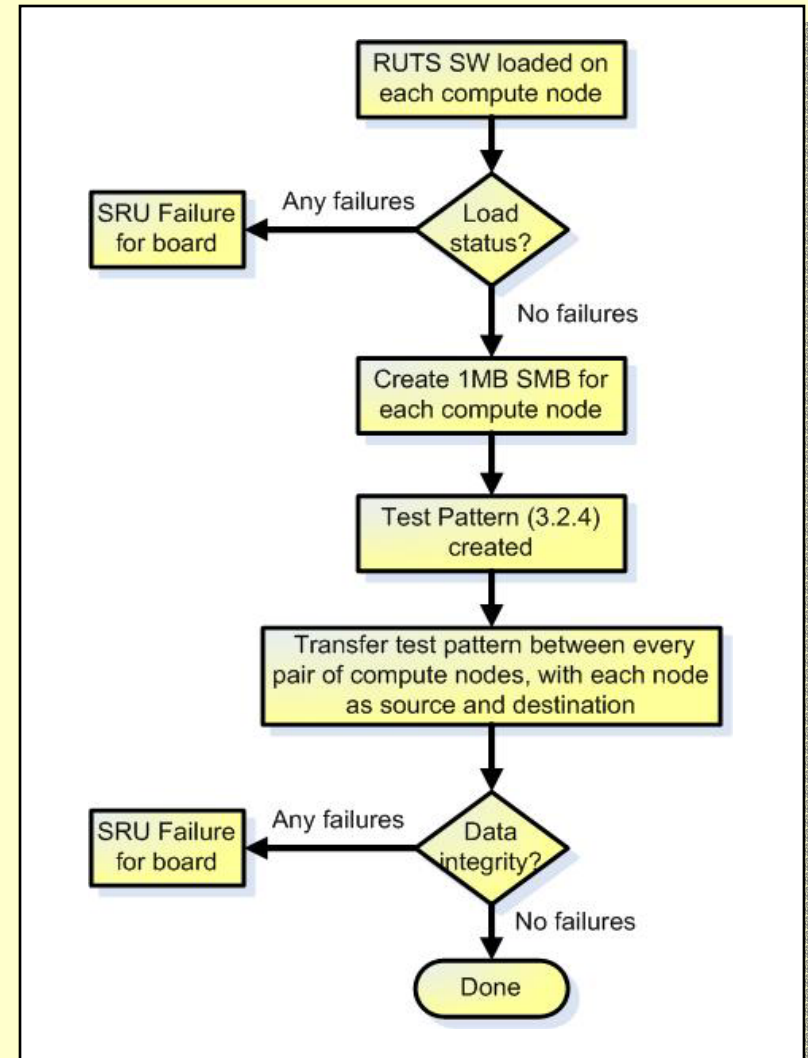


Use pictures to convey complexity...

- **MCH6 Operational Testing**

RUTS test software shall be loaded onto each compute node. The load status shall be checked and a failure for any node shall result in a declared SRU failure for the board. The RUTS software shall create a 1MB SMB (Shared Memory Buffer) on each compute node. A test pattern shall be created (3.2.4). For each MCH6 board, this pattern shall be transferred between compute nodes exhaustively (i.e., transfers shall be performed between every pair of compute nodes, with each node as source and destination.) Data integrity shall be checked at the end of each transfer. Failure of any data transfer shall result in a declared SRU failure for the MCH6 board on which the transfer occurred.

- 117 words, avg. 16.7 words per sentence



Tools to Reduce Ambiguity

Review requirements from different perspectives:

- **Development Perspective**

- Is every requirement **implementable**?
- Does every requirement have **one interpretation**?
- Is all information we need included?

- **SQA Perspective**

- Is every requirement **testable**?
- Does every requirement have **one interpretation**?
- Is all information we need included?

- **Doc Perspective**

- Is terminology used **consistently**?
- Does every requirement have **one interpretation**?
- Is all information we need included?



Tools to Reduce Ambiguity

- **Be on the lookout for...**

- **persuasive connectors** such as: certainly, therefore, clearly, it follows that... **ask for clarification.**
- **vague verbs** such as: handled, rejected, processed, skipped, managed, eliminated, etc. - **can be interpreted in many ways**
- When **lists** are given, but not completed, **be sure all items are understood.** Watch for lists with: **"etc."** , **"such as..."**



- For statements that **imply certainty** such as: always, every, all, none, never - **ask for proof**

Pressman, R., Software Engineering: A Practitioner's Approach, McGraw-Hill, 4th ed., 1997

Building Realistic Schedules

- **Why Are Most Estimates and Schedules Wrong?**
- **Estimating and Scheduling Best Practices**

Why are estimates and schedules wrong?

- **We play ridiculous negotiating games...**
 - **Doubling and Add Some**
 - **Reverse Doubling**
 - **Spanish Inquisition**
 - **"Guess the date I'm thinking..."**
- **We don't teach people how to do it!**
- **We allow requirements creep without assessing impact**
- **We don't hold people accountable**



Yourdon, E., Death March: The Complete Software Developer's Guide to Surviving 'Mission Impossible' Projects, Upper Saddle River, NJ: Prentice-Hall PTR, 1997

Typical “scheduled-backwards” Project

- **Customers promised more than can be delivered**
- **Project starts with a **predetermined end date****
- **Estimates based on **time available** rather than **time required****
- **Task **interdependencies** not identified**
- **Unexpected things that **ALWAYS** happen not anticipated...**
 - requirements **WILL** change
 - key members of the project team **WILL** leave
 - key assumption about product **WILL** prove wrong
 - previously unknown or ignored dependencies **WILL** arise
 - key resources **WILL** be pulled off to fight most recent “fire”

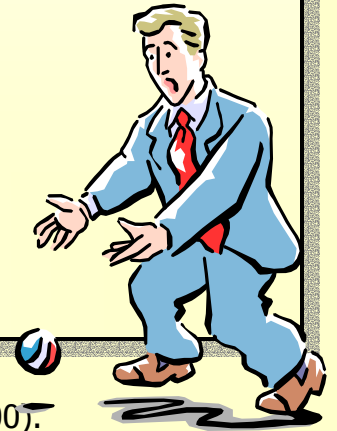
Typical “scheduled-backwards” Project

- **Eventually, schedule slip can't be ignored:**
 - Project Manager cuts **features** and **cranks up coding**
 - **Process is abandoned**
 - **Design Reviews and Inspections eliminated**
 - **Testing time is cut...**
- **Result: Everyone Loses!**
 - **Organization loses**
 - **Customers lose**
 - **Company loses**



Typical “scheduled-backwards” Project

- **Is schedule pressure a good thing?**
 - “When we are behind schedule and under pressure, we stop breathing.”
 - Undue schedule pressure can lead to some bad outcomes:
 - **Doing things faster than we’re able to**
 - **Skipping essential steps because there isn’t time to do them**
 - **Avoiding organizing because there isn’t time**
 - Working under higher than normal pressure for long periods of time leads to higher than average mistakes



“Don’t Forget to Breathe,” by Dwayne Phillips (*Cutter E-Mail Advisor*, 7 June 2000).

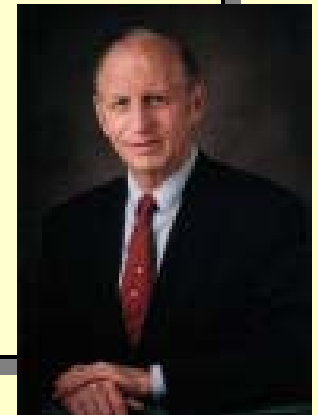
My Observations

- **Projects that are late are often “scheduled backwards”**
- **Negative impact on morale, quality, and productivity**
- **People not trained in estimating and scheduling**
- **Management frequently over-commits**
- **Lack of accountability**
- **Interdependencies ignored**
- **“People issues” ignored**
- **Personal “Quality Standards” often higher than company or customer’s**



Observations from Software Experts

- **To become predictable, we must learn to **accelerate the work, not just the schedule****
 - **Understand problem and create detailed plans**
 - **Engineers create plans, not managers**
 - **Management ensures staff is trained in planning**
 - **Senior managers review plans for completeness**
 - **Engineers measure and increase task time**
 - **Use plans to drive work**
 - **There's no substitute for quality**
 - **There's no substitute for a highly motivated team**



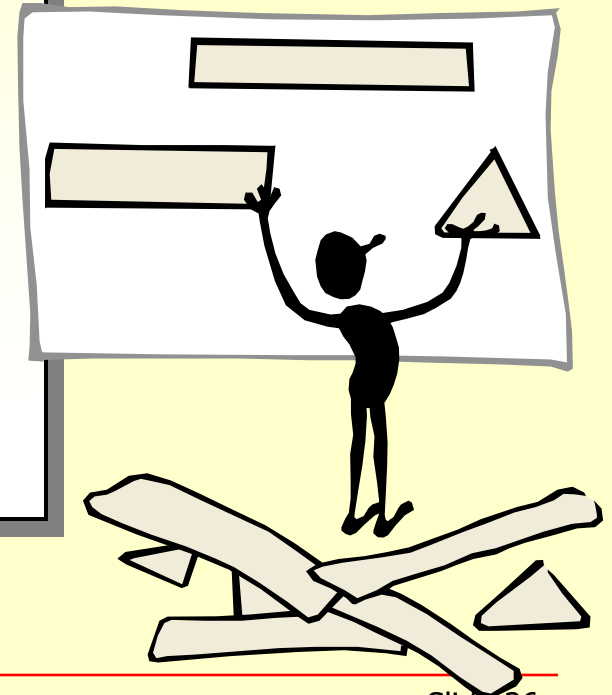
Humphrey, W., *Winning with Software: An Executive Strategy*, Addison-Wesley, 2002

Building Realistic Schedules

- **Why Are Most Estimates and Schedules Wrong?**
- **Estimating and Scheduling Best Practices**

Estimating and Scheduling Best Practices

- **COCOMO II**
- **Pro-Chain Methodology**
- **PSP Probe and TSP Planning**
- **Function Points / Feature Points**
- **Software Project Survival**
- **Wideband Delphi**
- **Cards on a Wall**
- **Yellow Sticky Method**



Yellow Sticky Method

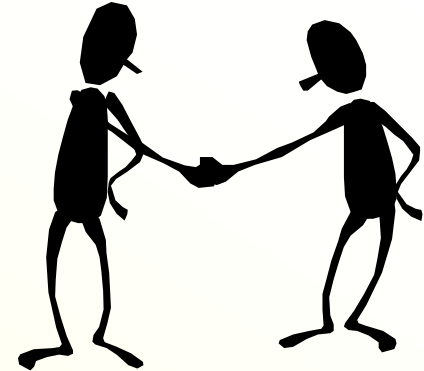
- Start with **Software Requirements Specification (SRS)**
- Someone from **Marketing** groups requirements:
 - **Must Haves** - not worth introducing without these features
 - **Wants** - features could be in a future release if necessary
 - If everything is **Must-Have** then no-tie rank
- Commit to deliver **ONLY Must Haves NOT Wants**



Commitment Management

Commitment Management

- **Foster a culture** based on making and meeting commitments...
- **Accountability!**
- **Measure and reward...**
- **Get buy-in from the organization BEFORE** making commitments to Customers
- **Deliver a clear, consistent message** to Customers
- **Set the bar low enough** so you can consistently beat it
- **Control customer "touch points"**



Yellow Sticky Method

- **Project Team:**
 - Software Development
 - SQA
 - Documentation / Training
- **Project Team** reviews **SRS** and identifies **specific tasks** each person will perform - for **Must Haves** and **Wants**
- **Each person** estimates how long it would take **them** to complete their **tasks** if **they could work on them uninterrupted**
- Use **Wideband Delphi** for new tasks



Yellow Sticky Method

- **Rules:**
 - Task duration should be from 1-5 days max
 - Tasks over **5 days** decomposed to sub-tasks
 - Apply **80% rule** when building schedule
 - Include vacation, holidays, trade shows, etc.
- **For each Task:**
 - Name of person responsible
 - Task description
 - Estimated duration (days)
 - Dependencies on other tasks

Name
Task
Duration
Dependencies

Yellow Sticky Method

- **Building the Schedule:**

- Chart paper is placed on a wall – week numbers marked
- Project team works together...
- Place tasks on chart where task ends - apply **80% Rule**
- Constructive criticism by peers...
- Iterative approach...

Name

Task

Duration

Dependencies

Name

Task

Duration

Dependencies

Name

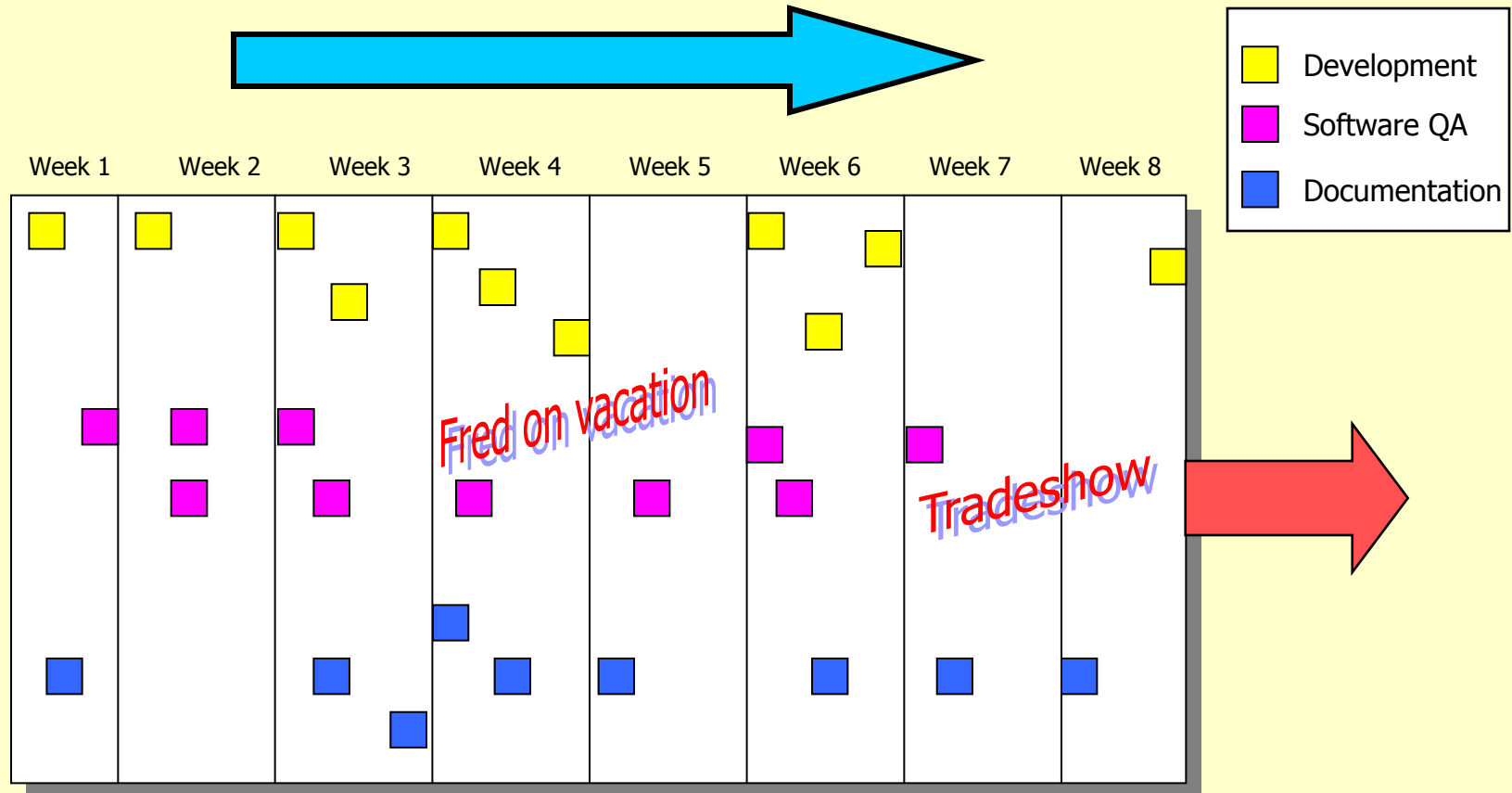
Task

Duration

Dependencies

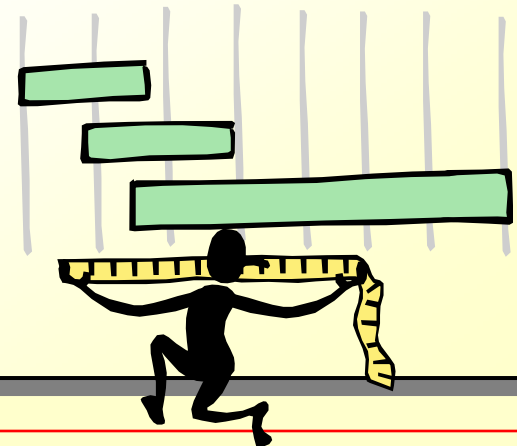
Yellow Sticky Method

Building the schedule GOING FORWARDS:



Yellow Sticky Method

- Result is an **ACCURATE, REALISTIC** schedule everyone has **BOUGHT INTO**
- Negotiate **realistic** schedule that meets business needs
- Use **Project Management tool** to track progress
- **MANAGE** Project to the Schedule
- **WHEN** unexpected things happen:
 - try to catch up
 - drop off **Wants** but not **Must Haves**



Summary

- **Starting projects **without requirements** significantly **increases probability of failure****
- **Writing requirements is hard because **English** is imprecise and **inherently vague** and because people aren't **trained****
- **Ambiguity** is a critical problem with requirements
- **Scheduling forwards using **Yellow Sticky Method** results in accurate, realistic schedules that can **actually be met****
- **Worst case, you'll deliver exactly what was promised. Best case, you'll deliver more**
- **People work harder to meet schedules they **set themselves****

Additional Workshops...

Software Quality Consulting offers workshops on:

- **Writing Software Requirements**
- **Building Realistic Project Schedules from Software Requirements**
- **Software Verification & Validation**
- **Accurate Estimating and Scheduling Using the Yellow Sticky Method**
- **Predictable Software Development**
- **Peer Reviews and Inspections**
- **Improving the Effectiveness of Testing**
- **Risk Management for Embedded Software Development**

- **For more information, please visit**
 - http://www.swqual.com/training/on_site.html
- **Subscribe to my e-newsletter...**
 - www.swqual.com/newsletter/Subscribe.htm



Thank you....

...for taking time to attend this talk.

If you have any questions, please feel free to call or e-mail...

Software Quality Consulting Inc.

Steven R. Rakitin
President

- Consulting
- Training
- Auditing

Phone: 508.529.4282
Fax: 508.529.7799

www.swqual.com
info@swqual.com