Carnegie Mellon
Software Engineering Institute
Pittsburgh, PA 15213-3890

Teaching Disciplined Software Engineering

Watts S. Humphrey
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213-3890

Sponsored by the U.S. Department of Defense
© 2002 by Carnegie Mellon University

Why Did the Soviet Union Fail?

Think about this question for a few minutes.

Write down your thoughts.

We will then have a brief discussion.

© 2002 by Carnegie Mellon University

The Fundamental Problem

While many factors contributed, the prime reason the Soviet Union failed was autocratic management.

The Soviet economy was centrally managed.

- **All plans and budgets were set in Moscow.**
- **Prices and wages were decided centrally.**
- **All property was owned by the state.**

This caused enormous inefficiencies.

- **Uninformed decisions**
- **Slow reaction time**
- **No sense of ownership**
- **No coordination, except through headquarters**

The Soviet economy was not competitive.

© 2002 by Carnegie Mellon University

Corporate Management

Successful large corporations have learned that autocratic management is

- unresponsive
- bureaucratic
- prone to failure

Companies like GE and IBM

- decentralize decision making
- establish remote profit centers
- develop local business managers

Decentralization is required for large and complex businesses.

© 2002 by Carnegie Mellon University

Project Management

The problems of autocratic management have not yet been addressed at the project level.

Typically, most project and laboratory managers act autocratically, at least when guiding projects.

Why, if autocratic behavior is ineffective for complex work, do we continue to use it to manage projects?

© 2002 by Carnegie Mellon University

Agenda

The present situation

The growing need for software

Autocratic management

Democratic systems

Software citizenship

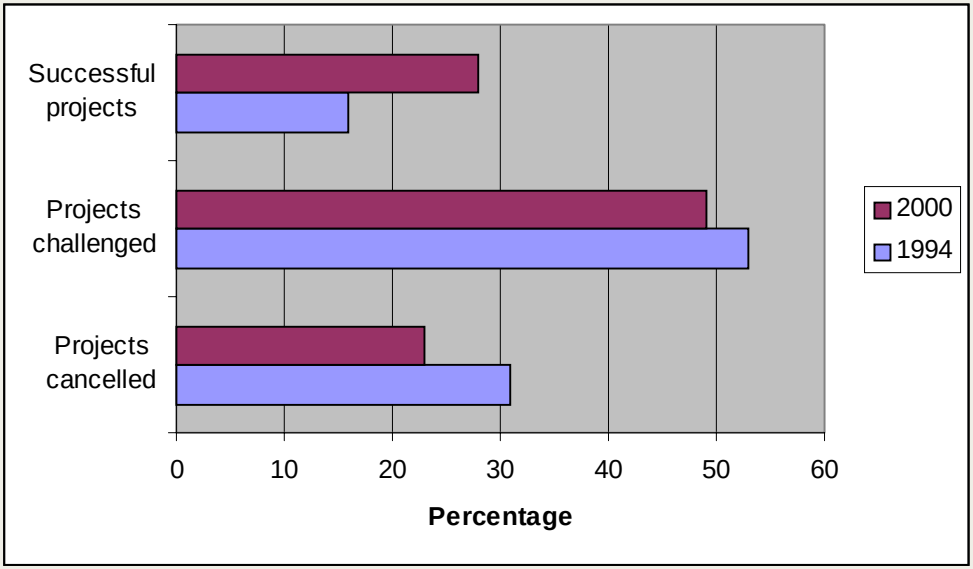
The academic challenge

Conclusions

© 2002 by Carnegie Mellon University

The Present Situation - 1

The current performance of software groups is poor*.



While there is gradual improvement, it is very gradual and 72% of projects were not considered successful.

*Standish Reports, 1994, 2000

© 2002 by Carnegie Mellon University

The Present Situation - 2

The situation appears to be improving very slowly.

It is also instructive to realize that these improvements were made because

- software customers are more demanding
- management is improving its practices with the CMM, ISO, etc.

These recent improvements have come, almost entirely, from management behavior.

© 2002 by Carnegie Mellon University

The Present Situation - 3

Technical advances have not significantly improved the business performance of software groups.

The academic community has not been involved in improving software business performance.

Should it have been?

Can present improvement trends continue without academic involvement?

This talk discusses why the answers to these questions are yes and no.

© 2002 by Carnegie Mellon University

The Growing Need for Software

Software now controls large segments of the U.S. economy.

- Factories are managed by software.
- Many products are controlled by software.
- Finance, administrative, and business operations are largely run by software.

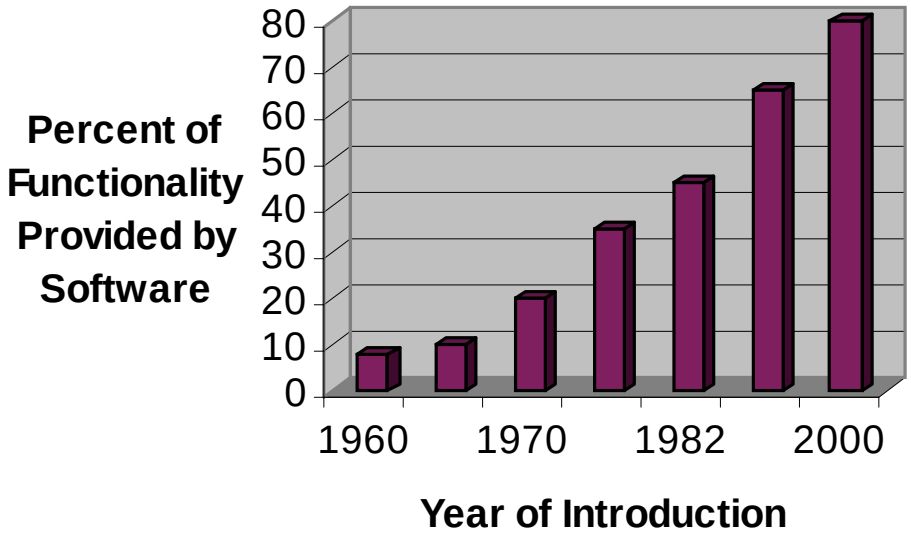
Software cost and schedule is now the gating item in many business strategies.

A good example of the pervasive nature of software is military aircraft.

© 2002 by Carnegie Mellon University

Software Is Pervasive

Software Functionality in Military Aircraft*



F-4: 1960 – 8%
F-16: 1982 – 45%
F-22: 2000 – 80%

*Ferguson, IEEE Software, August 2001

© 2002 by Carnegie Mellon University

The Software Business Problem

Managers try to run their businesses in spite of the poor performance of their software groups.

Software intensive systems are

- **becoming much larger**
- **being used in increasingly sensitive applications**
- **exposed to growing volumes of attacks**

With present practices, the current software problems will likely get worse.

© 2002 by Carnegie Mellon University

The Problem

The principal problem in the software community is that our projects are managed autocratically.

Autocratic management is incapable of handling complex and creative software work.

Autocratic management results in

- **dissatisfied engineers**
- **poor quality products**
- **delayed schedules**
- **excessive costs**

These problems cannot be solved without addressing the autocratic management culture.

© 2002 by Carnegie Mellon University

Autocratic Management - 1

Autocratic systems can be efficient when

- fighting traditional wars
- managing routine activities
- dealing with uncooperative people

Autocratic systems are not effective for

- special military forces
- creative or unpredictable work
- highly motivated people

© 2002 by Carnegie Mellon University

Autocratic Management - 2

Autocratic management is not appropriate for software work.

However, autocratic management is the normal software management style.

Software managers typically

- set software schedules
- impose processes, tools, and plans
- take technical shortcuts to meet the schedule
- do not involve the engineers in the business decisions for their projects

© 2002 by Carnegie Mellon University

Autocratic Management - 3

Engineers cannot be fully effective under these conditions.

Not surprisingly, their projects are typically

- **late**
- **over budget**
- **of poor quality**

© 2002 by Carnegie Mellon University

Autocratic Management - 4

If autocratic management is so ineffective, why do managers behave this way?

They have no choice.

If software managers asked the engineers to manage cost, schedule, and quality

- they would not know what to do
- project performance would not improve
- it would probably get worse

© 2002 by Carnegie Mellon University

Democratic Systems - 1

Democratic* systems can seem inefficient and unresponsive.

As Winston Churchill once said:

"…democracy is the worst form of Government except all those other forms that have been tried from time to time."

*Democracy: government by the people

© 2002 by Carnegie Mellon University

Democratic Systems - 2

Democracy is the only proven way to handle the problems of size and complexity.

However, democratic systems are much more difficult to establish than autocracies.

The reason is that democratic systems must be established and sustained democratically.

© 2002 by Carnegie Mellon University

Requirements for a Democracy

Why don't democracies survive in some societies?

In many countries, democracies have been replaced with autocracies.

- **Pakistan**
- **Lebanon**
- **Peru, etc.**

Why, if democracy is so effective, doesn't it survive?

© 2002 by Carnegie Mellon University

Responsible Citizenship

The ultimate power in democracies does not come from the center.

It comes from the citizens.

Power is rarely handed to the citizens; they must take it and they must fight to keep it.

As Thomas Jefferson said:

“The tree of liberty must be refreshed from time to time with the blood of patriots and tyrants.”

© 2002 by Carnegie Mellon University

Establishing a Democracy

To establish and sustain a democracy, the citizens must demand it.

This can only happen when the citizens are convinced that democratic systems are best.

In societies where this citizenship conviction is lacking, the leaders must establish the democracy.

For democracy to last, the leadership must sustain it for long enough to build a democratic culture.

For a democratic culture to survive, its principles must permeate the educational system.

© 2002 by Carnegie Mellon University

Democratic Skills

For the citizens to demand a democratic system, they must share basic convictions.

In a nation, these convictions are

- a respect for law and order
- a willingness to abide by majority decisions
- agreement to participate in juries
- a commitment to vote
- a feeling of ownership

Until the strength of these convictions can overcome historical divisions, democracy cannot survive.

This requires education.

© 2002 by Carnegie Mellon University

Democratic Project Management

The TSP is an example of how to run projects democratically.

On TSP teams, the team members

- **make their own schedules and commitments**
- **manage their own projects**
- **work together as a group**
- **are guided by a coach**
- **cooperate with their team leader**

© 2002 by Carnegie Mellon University

TSP Team Management - 1

All TSP team members hear their goals directly from management.

The entire team works together to

- **select every member's personal role in team management**
- **establish the team strategy**
- **define the team processes**
- **develop the project plan**
- **assess and evaluate project risks**

© 2002 by Carnegie Mellon University

TSP Team Management - 2

Once the team has developed its plan, the team leader leads the team in

- presenting the plan to management
- reviewing any alternate plans
- negotiating any issues with management
- agreeing on the project plan and schedule
- managing, tracking, and reporting on the work

© 2002 by Carnegie Mellon University

TSP Results

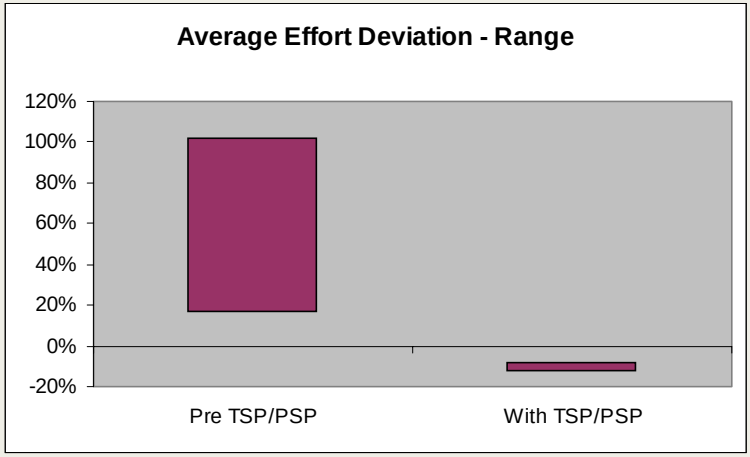
When engineers work on TSP teams, they behave like professionals.

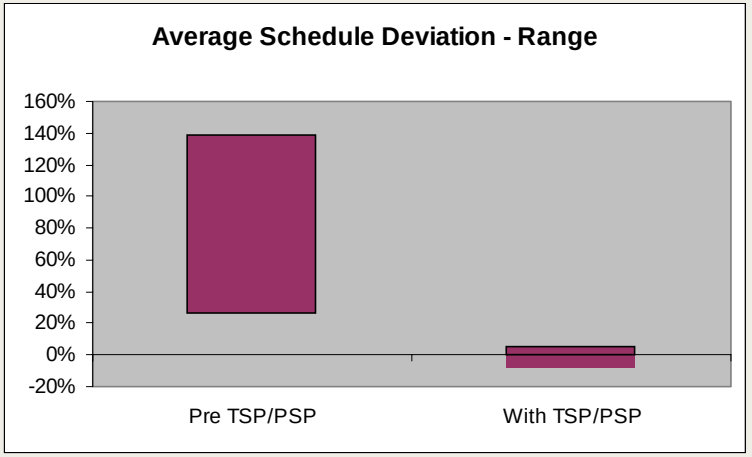
And their results are far better than before.

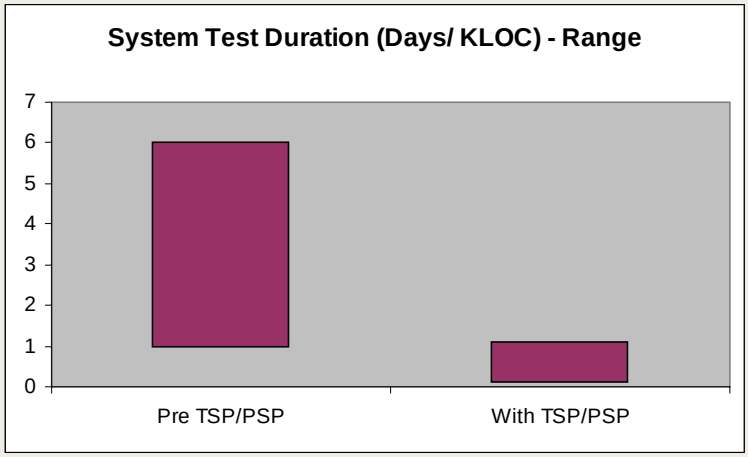
TSP teams consistently deliver quality products on schedule and for planned costs.

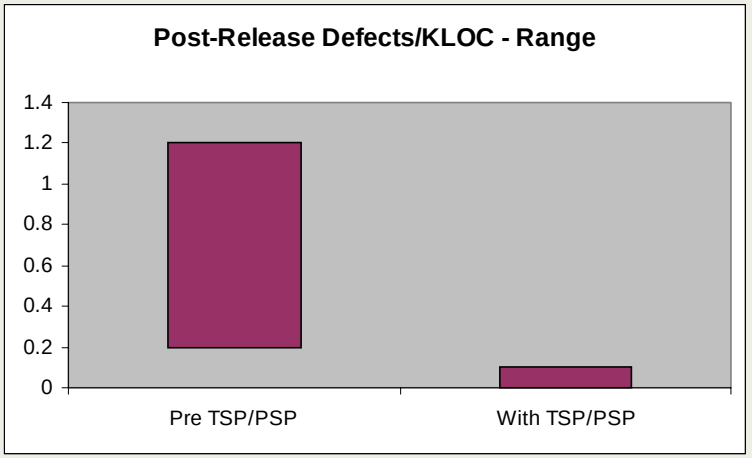
© 2002 by Carnegie Mellon University

TSP Experience – 28 teams









© 2002 by Carnegie Mellon University

Engineers Like the TSP

Engineers like to work in a democratic project environment.

Some comments from TSP team members.

- This really feels like a tight team.
- The TSP forces you to design, to think the whole thing out.
- Design time is way up but code time decreased to compensate.
- Tracking your time is an eye opener.
- Really good teamwork on this project - no duplication of effort.
- I'm more productive.
- Gives you incredible insight into project performance.
- Wonderful to have team members assigned specific roles.
- Team really came together to make the plan.
- I feel included and empowered.

© 2002 by Carnegie Mellon University

The Improvement Problem

To consistently produce software on promised costs and schedules, we need

- self directed teams
- a democratic management system

How can we get this?

We need to change the software culture.

© 2002 by Carnegie Mellon University

Changing the Software Culture

As in any society, to consistently practice democratic principles, we need an educated citizenry.

We also need a changed software culture.

It requires new values, new standards, and new forms of recognition.

© 2002 by Carnegie Mellon University

The Challenge - 1

Lasting change in the software community must start in the educational system.

We need professionals who

- **know how to make and meet commitments**
- **consistently deliver on their commitments**
- **feel responsible for the quality of their work**

We also need professionals who will not meekly accept unrealistic commitments.

© 2002 by Carnegie Mellon University

The Challenge - 2

To consistently act professionally, engineers must learn and believe several things.

First, they must know how to make realistic commitments.

Second, they must strive to meet their commitments.

Third, they must refuse to make commitments they do not believe they can meet.

The challenge is to develop these skills and convictions during an engineer's education.

© 2002 by Carnegie Mellon University

First Requirement

Engineers must act professionally.

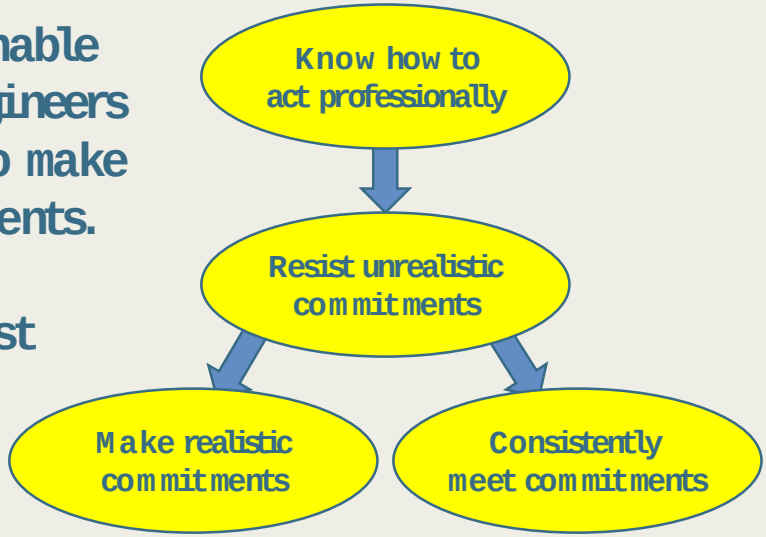
They must know how to resist unrealistic commitments.

© 2002 by Carnegie Mellon University

The Second Requirement

To resist unreasonable commitments, engineers must know how to make realistic commitments.

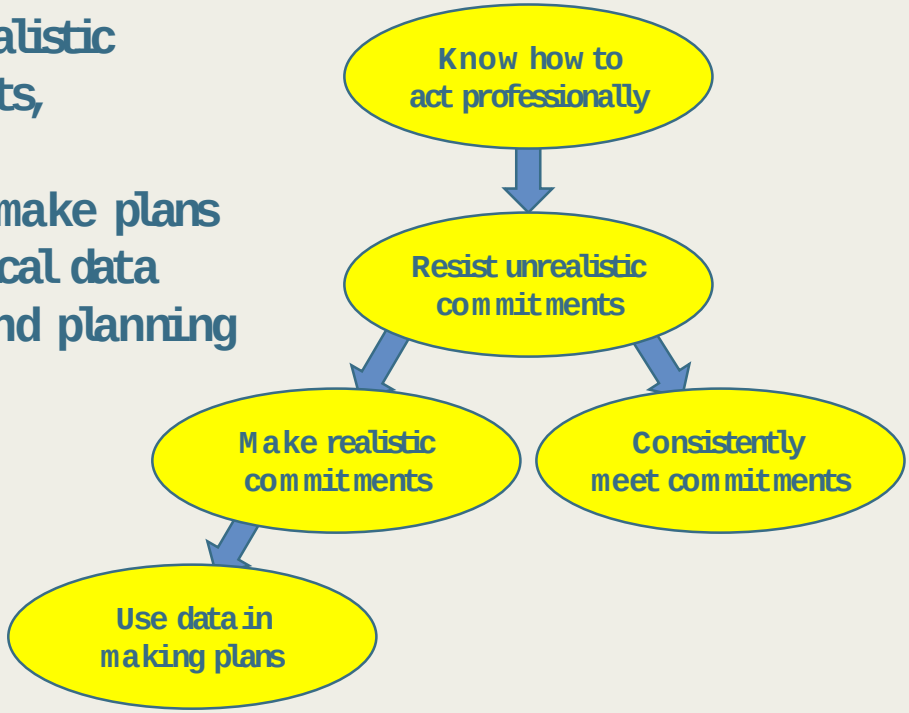
And then they must consistently meet these commitments.



© 2002 by Carnegie Mellon University

The Third Requirement

To make realistic commitments, they must
- regularly make plans
- use historical data
- follow sound planning methods

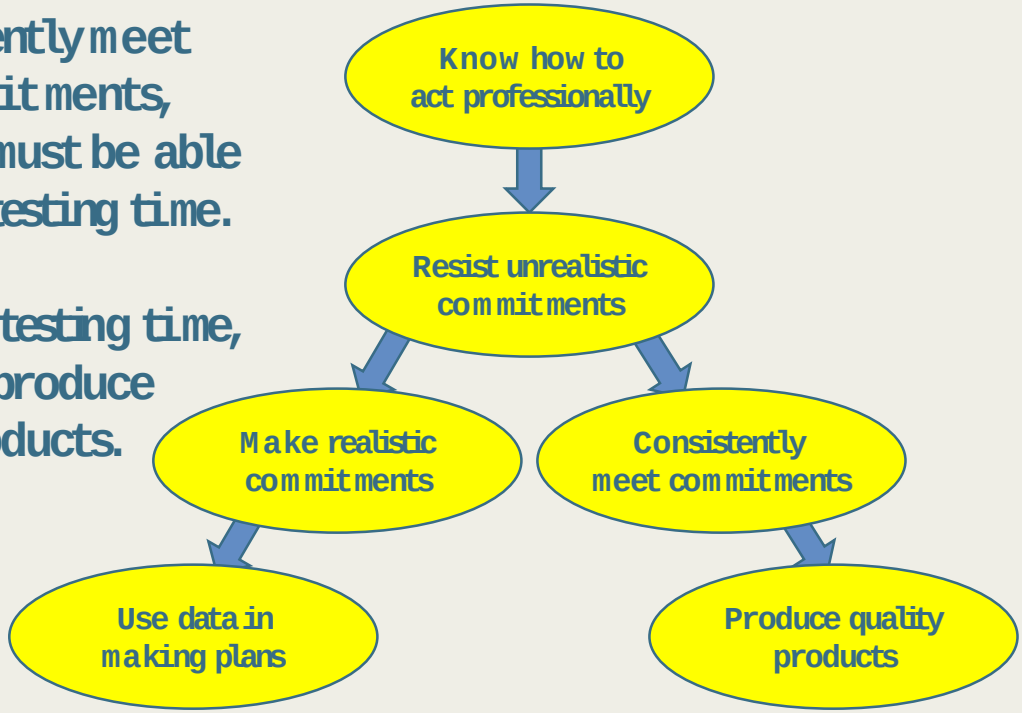


© 2002 by Carnegie Mellon University

The Fourth Requirement

To consistently meet their commitments, engineers must be able to predict testing time.

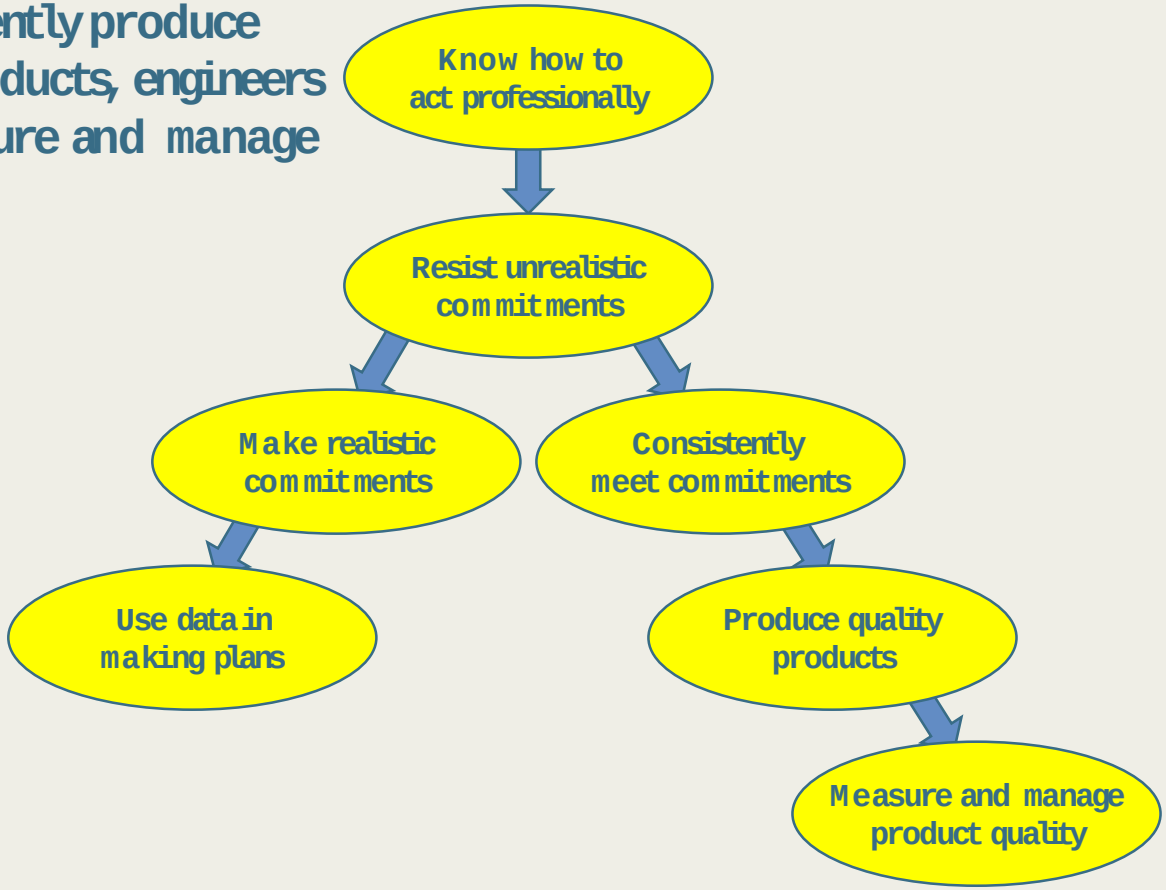
To predict testing time, they must produce quality products.



© 2002 by Carnegie Mellon University

The Fifth Requirement

To consistently produce quality products, engineers must measure and manage quality.

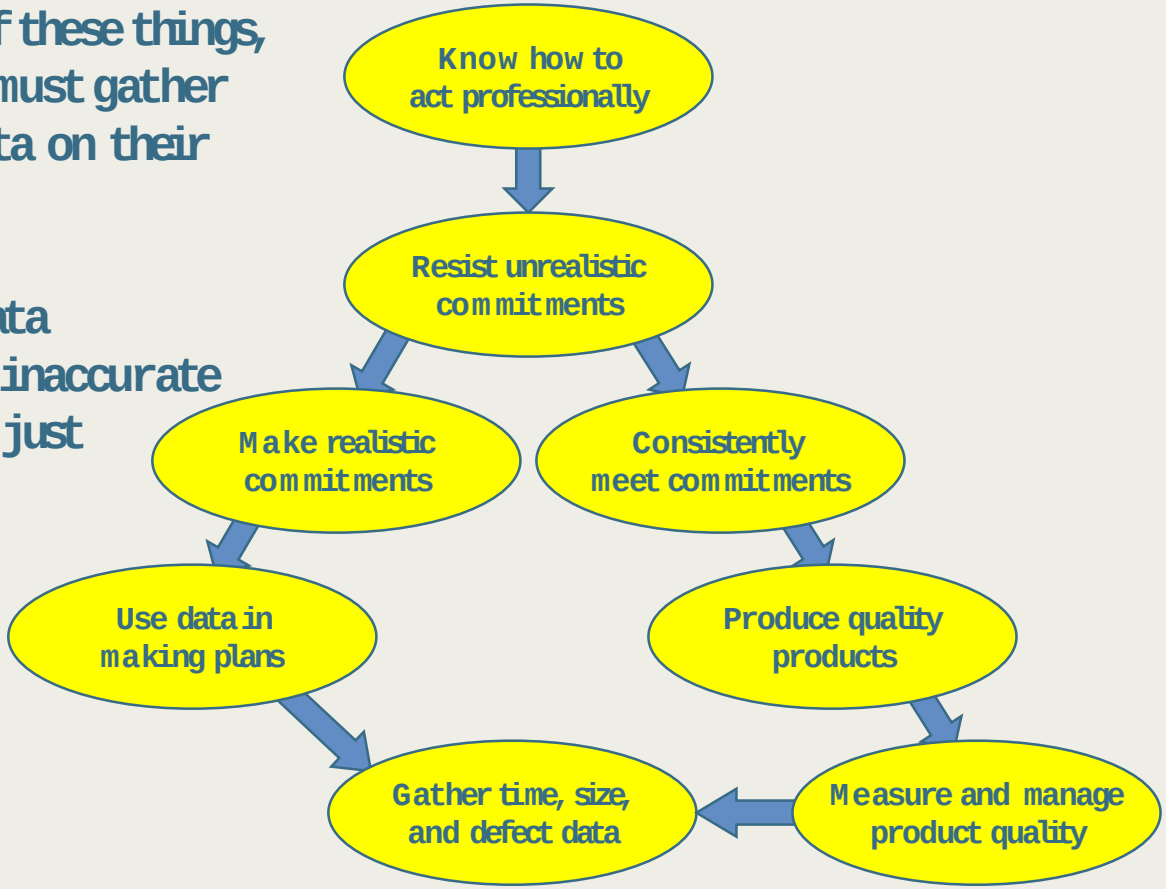


© 2002 by Carnegie Mellon University

The Sixth Requirement

To do all of these things, engineers must gather and use data on their work.

Without data
- plans are inaccurate
- quality is just talk



© 2002 by Carnegie Mellon University

The Academic Challenge

To behave professionally, engineers must know how to behave as responsible citizens.

They must feel personally responsible for doing predictable and high quality work.

This responsible attitude cannot just come from a course, it must be ingrained.

Building convictions takes time, supervised experience, and informed guidance.

To provide this, software engineering programs must offer extensive laboratory experience.

© 2002 by Carnegie Mellon University

Conclusion

Once engineers start behaving professionally
- democratic management will follow
- software performance will steadily improve

By teaching engineers to behave professionally, universities could accelerate this change.

For this to happen, the values of the software community must change.
- from unplanned coding
- to predictable quality

This will not happen until academic values change as well.

© 2002 by Carnegie Mellon University