There is no magic

Bob Martin
bobmartin08008@mac.com

The Source of My Biases

- Managed building ~100M lines of code
  - System software, e.g., UNIX, C++, Tuxedo
  - Network management
  - Service control points, e.g., 800 and credit card calling
  - Embedded systems, e.g., ATM switches
- Lead 3,500 programmer organization to 3-4x competitive productivity, 10x competitive quality, and 95% on time/content
- Perspectives on software development
  - "Oversaw" ~15,000 Bell Labs programmers as Bell Labs CTO
  - 8 years on National Research Council Computer Science and Telecommunications Board, e.g., reviewed failed government projects
  - Learned about software development in different cultures through US and foreign tours
  - Chaired first IEEE Software Industrial Advisory Board
- Love technology – "one a year"
- Not expert in contemporary software technology/tools

I think about this when asked about Python

Software Systems Frequently Fail

	Program	System
Program	1	3
Product	3	9

Relative development effort

Boehm survey

Average overrun: 89.9% on cost, 121% on schedule, with 61% of content

Because:

- Poor Management

  Well-proven techniques are often not used - ignorance, arrogance, or naivety

- Novelty

  You don’t know what you don’t know

- Second System Effect

What Year Was This Conference?
NATO Software Engineering Conference - 1968

Design

- Guidelines
  - External function, e.g., user language
  - Internal function, e.g., parsers
- Techniques
  - Deductive or inductive
  - Modularity and interfaces
  - Complexity control
- Proscriptions
  - Completeness, modularity, efficiency
  - Self-monitoring and performance improvement
  - Incremental systems
  - Balance
    - Security, control, … vs. cost
    - Limited goals to attain excellent performance
- Design problems
  - Data structures on system design
  - Fixed resources
  - Cooperating processes
- Documentation

Production

- Organization for producing software
  - Number and quality of people
  - Structure of large groups
  - Control and measurement
  - Internal communication
- Production techniques
- Monitoring

Service

- Distribution
  - Media
- Acceptance criteria
- Documentation
- Adaptation
- Maintenance
  - Error detection & reporting
  - Response and distribution
- Documentation
- Performance
- Feedback into design

Project MAC

- MIT's Project MAC
  - Project on **M**athematics **a**nd **C**omputation, later backronymed to **M**ultiple **A**ccess **C**omputer, **M**achine **A**ided **C**ognitions, or **M**an and **C**omputer
  - Darpa funded 1963
  - Created Multics with processes, pages, segmentation, hierarchical file system, and lots and lots more!
  - Trained Ken Thompson and Dennis Richie who built Unix carefully selecting a few of Multics ideas (Unix is play on Multics name)
- Columbia's Project MAC

Multics Overview Paper - 1965

“Experience has shown that privacy and security are sensitive issues in a multi-user system where terminals are anonymously remote”

F. J. Corbató MIT and V. A. Vyssotsky Bell Labs

Productive Teams

- Skilled, ***disciplined*** programmers
    - Productivity of best programmers are ~10+ times the average

- Skilled, ***disciplined*** teams

Who Are You

Experienced **programmers**

How many programming languages have you used to produce a working program?

1	0%
2	3%
3	16%
4	25%
5-10	41%
>10	16%

What is the largest program you have written, measured in lines of code?

10 – 100	0%
100 – 1,000	28%
1,000 – 5,000	50%
5,000 – 10,000	6%
> 10,000	16%

Have you ever worked with others before this class to produce a program?

Yes	94%
No	6%

Who Are You

Experienced, *high ego* programmers

How good a programmer are you relative to others in this class?

Top 10%	10%
Top 11-25%	28%
Top 26-50%	34%
Top 51-75%	21%
Bottom 50-25%	3%
Bottom 24-11%	0%
Bottom 10%	0%

97% are in the top half
(A record for this survey!)

Columbia has helped me learn to program by:

Teaching me algorithms and data structures	100%
Teaching me programming style	10%
Reviewing the details of great programs	0%
Reading/correcting at least:	
1 program greater than 100 lines of code	90%
5 programs greater than 100 lines of code	25%
10 programs greater than 100 lines of code	10%

Not learning how to program (Question not included in this year's survey)

Is The Person Next To You A Good Programmer?

- Why is so little time spent teaching college students how to program?
  - Compare it to the importance/effort placed on teaching writing
- We read great books to write prose – read great code
  - Lions' *Commentary on Unix*
  - *The Practice of Programming* – Kernighan & Pike
- Write code the way you write prose
  - Have reviews of your code to find and learn from your errors
  - Use a style guide

Who Are You

Experienced, high ego programmers, *who write lousy code*

What is the error rate per 1,000 lines of code in your completed programs?

<.5	3%	**Best in class in industry**
.5 – 1	6%	
1 – 5	22%	
>5	22%	
Don't have a clue	47%	

What techniques do you use to produce high-quality code? (Check all that apply)

Write a requirements specification before writing code	50%
Write a design specification before writing code	66%
Have colleagues review your design for ease of use	47%
Develop a performance model	17%
Build a prototype	50%
Follow a programming style guide	50%
Have a colleague inspect your code	37%
Do root cause analysis on bugs	33%
Gather metrics on code quality	17%
Use automated testing	40%

} **Don't learn from errors or others**

This is one of the highest use of technique in five years of survey. Why? (Both Edwards' classes were high)

A Guess

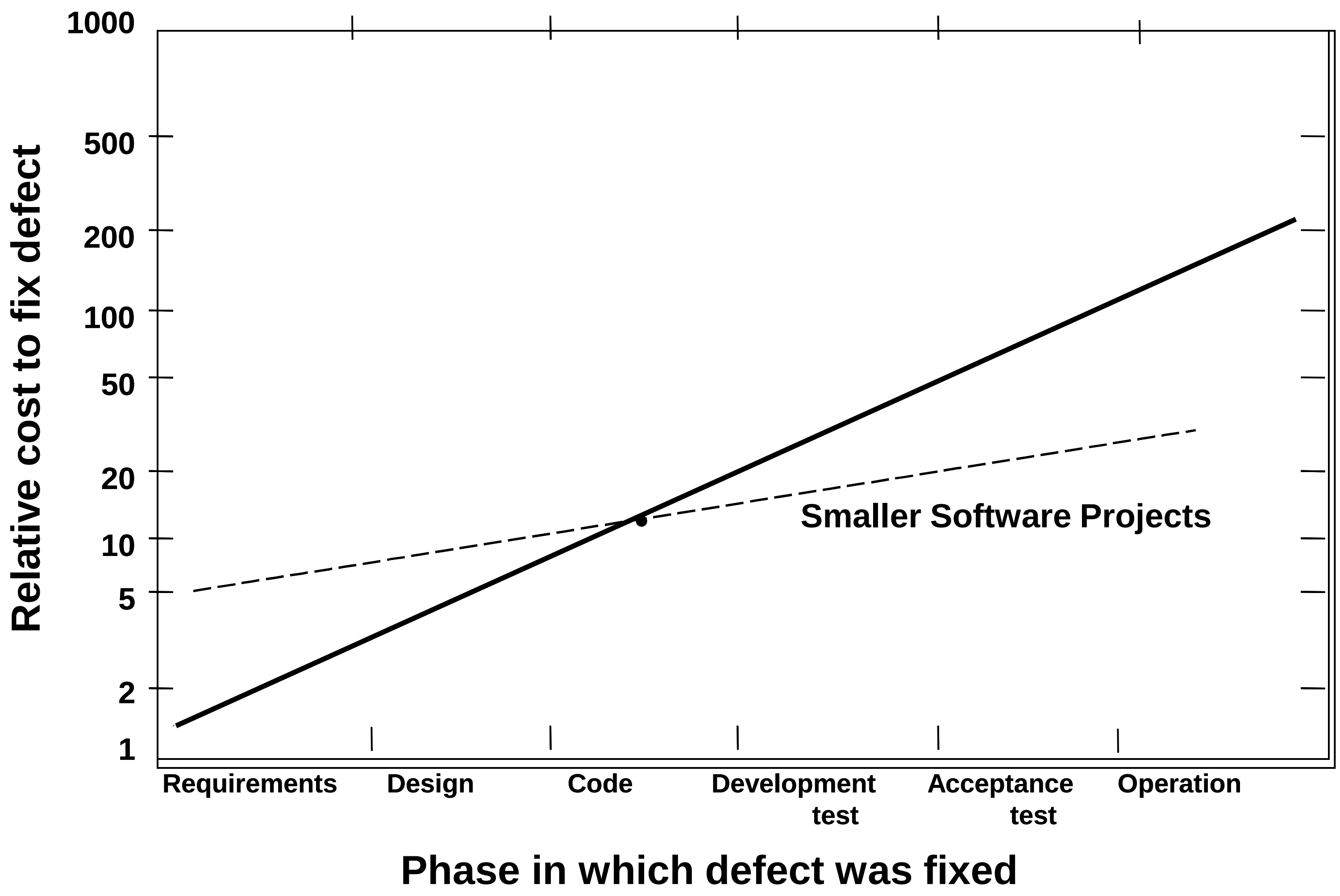
Process-oriented geeks took the survey
They will be rich

Code gunslingers did not
They will be happy

Watts Humphrey’s Software Engineering Model

- Personal Software Process
  - Process discipline & measurement
    - Fault injection/removal, personal process, programming style guide
  - Estimating & planning
    - Estimate program size and do testing
  - Quality management & design
    - Design & code review

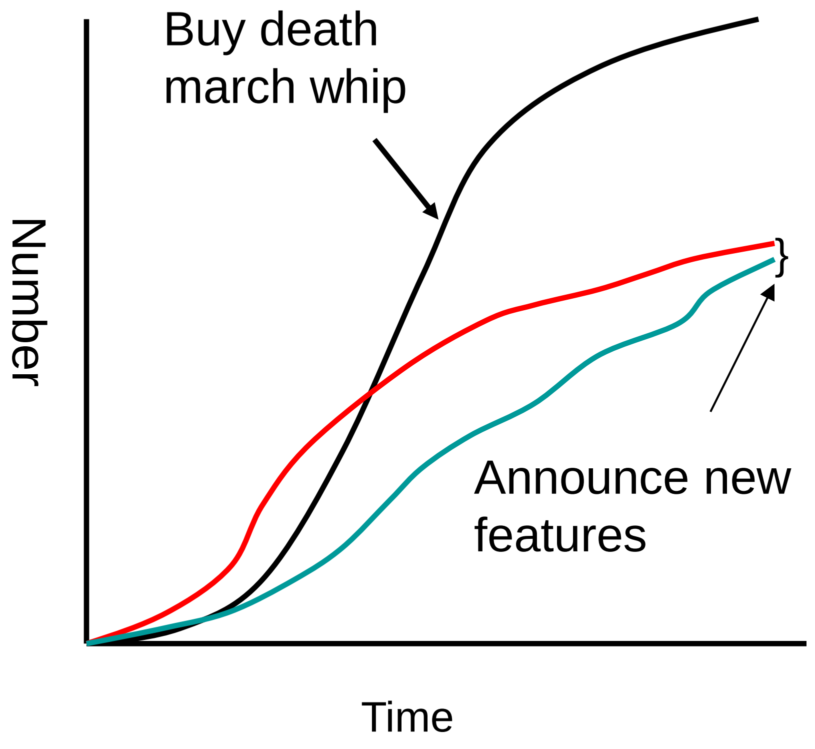
Find and Fix Defects Early



* Barry Boehm - A View of 20th and 21st Century Software Engineering

Testing

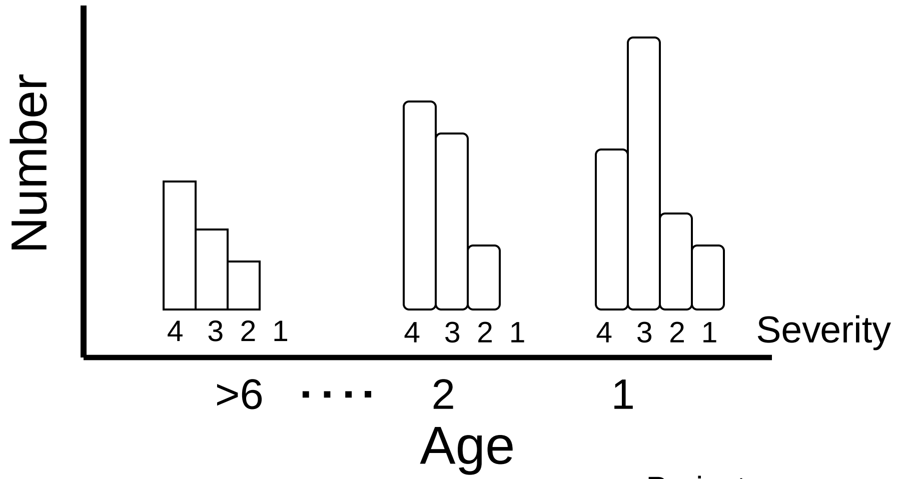
- Full lifecycle activity
  - Multi-level - unit, subsystem, system
  - Often 50% of time & effort
  - “50% of Google code is for testing”
- What to test
  - Requirements & design
  - Coverage
  - Usability
  - Performance
- One-day *automated* build/test
  - Domain specific tools
  - Field-driven test libraries
- Measure with S curves

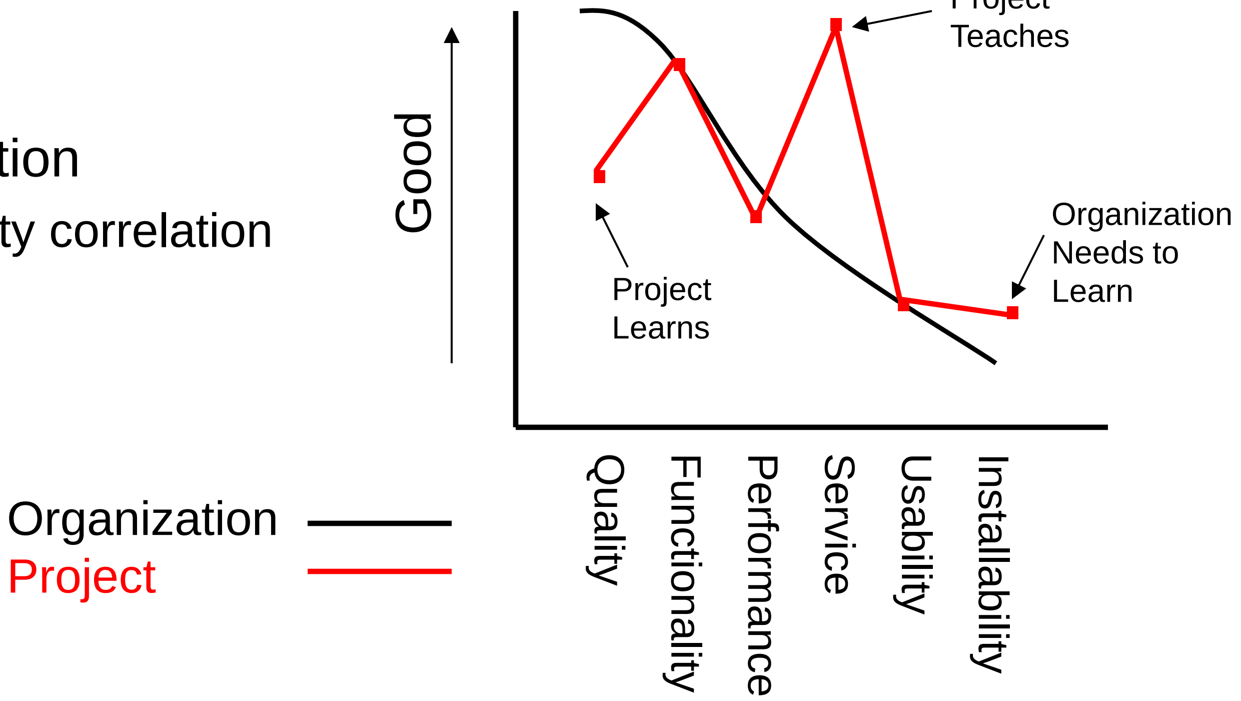


Test cases run	—
Bugs found	—
Bugs closed	—

Metric Driven Improvement

- Quality
  - Measures
    - Fault density
    - Service responsiveness
  - Improvement
    - Root cause analysis
    - Team & individual
- Productivity
- Customer satisfaction
  - Huge fault density correlation





What Is Important to You?

Rate the importance of each of the following in assuring a successful, quality project – High (H), Medium (M), Low (L):

Project plan	H
Language specification	M-H
Architecture document	M-H
Design document	M-H
Test plan	M-H
Automated testing system	M-H
Project meetings on status & plans	M-H
Project meetings to review designs, code, …	H
Clear roles, e.g., tester, architect, …	M
Great people	H++

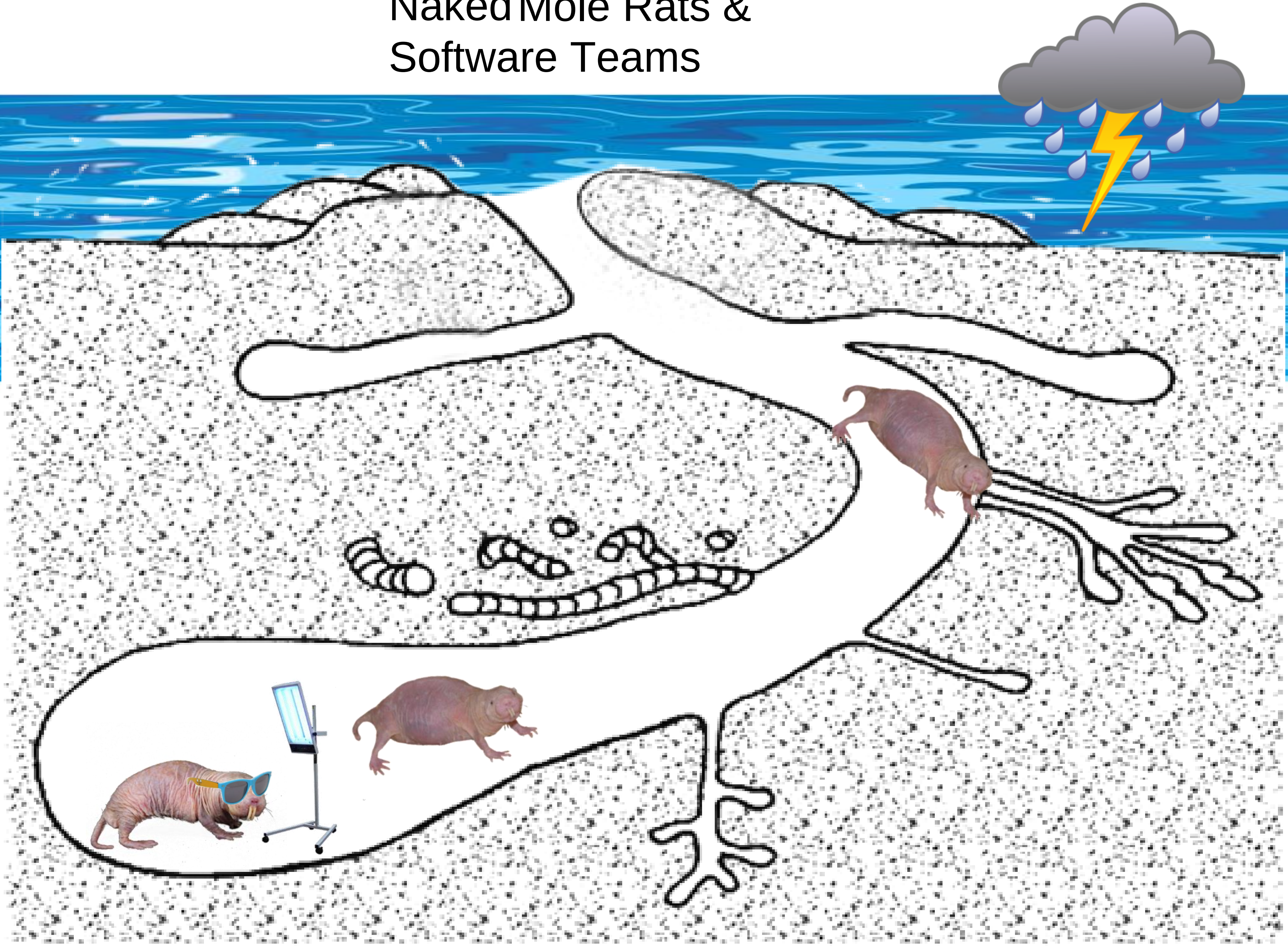
Again, unusually high value on technique/process relative to other years.

Rank order the value/effectiveness of each role in assuring a successful, on time, quality project – High (H), Medium (M), Low (L):

Architect	3.1/1.4
Project manager	3.1/1.6
Programmer	3.1/1.4
Tester	2.7/1.6
Documenter	2.1/1.6

Hmmm, you want to soar with the eagles but are stuck with high-ego turkeys

Naked Mole Rats & Software Teams



Building Complex Things

- Rank order the importance of:
  - Architect
  - Project manager
  - Craftspeople

Architecture & Design

- Control & Reduce Complexity
  - Chunk independence
    - Reduce n^2 effects
    - Structure for changeability
    - Achieve Steve Job's "Taste"
  - Inject huge-payoff CS technology
    - Tiny languages, finite state machines, appropriate algorithms & data structures, formal methods, etc.
- Check it
  - Prototype
  - Audit with smart friends
- Watch out for crumbling with time
  - Throw old away, don't fix forever*

* *"An Architecture History of the Unix System"* - Feldman Usenix '84

"A Model of Large Systems Development" – Belady & Lehman IBM Systems Journal '76

Performance Estimation

How much water flows through the Mississippi each year?

“Programming pearls: the back of the envelope”
Communications of the ACM, Vol 27, Issue 3, 3/84

Effort Estimation

- Individual productivity varies tremendously
- Approach & algorithm matter
  - ✓ How many lines of code to determine text word frequency count, most frequent first
    - "1" - Six Unix tools, piped together
    - 100 - C, Pascal
    - ~600 - ACM paper on commenting code
    - 1,000 - Cobol, PL/1
    - 5,000 - Unisys SW VP
    - 10,000 - IBM SW VP
    - ??? with very detailed requirements
- An established team's productivity is quite constant
  - ✓ Function of system size, complexity & age
  - ✓ Improves with quality

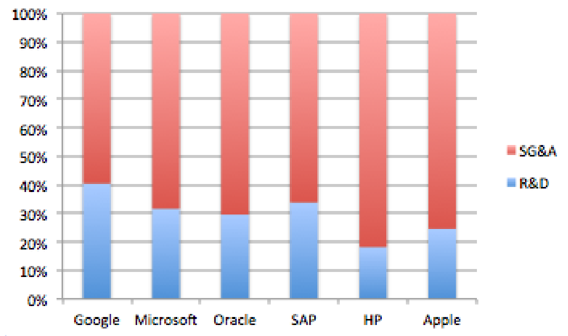
Getting Ready For Your Profession

What profession do you plan to pursue?
Analyst/Product manager
New Product Programmer
Maintenance Programmer
Tester
Teacher
Business person
Don’t know

Will You Be A New Product Programmer?

Very unlikely even if you work for a software company!

- 1/5 – 2/5 do R&D



- 1/4 - 1/3 are “programmers,” while 1/2 are testers

- 1/10 – 1/5 work on new products

Thus, less than 2%!

- Learn to test
- Learn to maintain

Productive Teams

- Skilled, ***disciplined*** programmers
- Skilled, ***disciplined*** teams

Why Teams?

- Takes teams to do complex things
- Team decisions are better than individual decisions
  - ➢ Individuals can become enamored with their own errors – Francis Crick
  - ➢ Diverse backgrounds and skills

Desert Survival Game*

Rank Order Survival Value of:

Flashlight
Jackknife
Sectional map of the area
Plastic raincoat
Magnetic compass
Compress kit with gauze
Cosmetic mirror
.45 caliber pistol (loaded)
Parachute
Bottle of 1000 salt tablets
1 quart of water per person
Edible Animals of the Desert book
2 pairs of sunglasses
2 quarts of 180 proof vodka
1 top coat per person

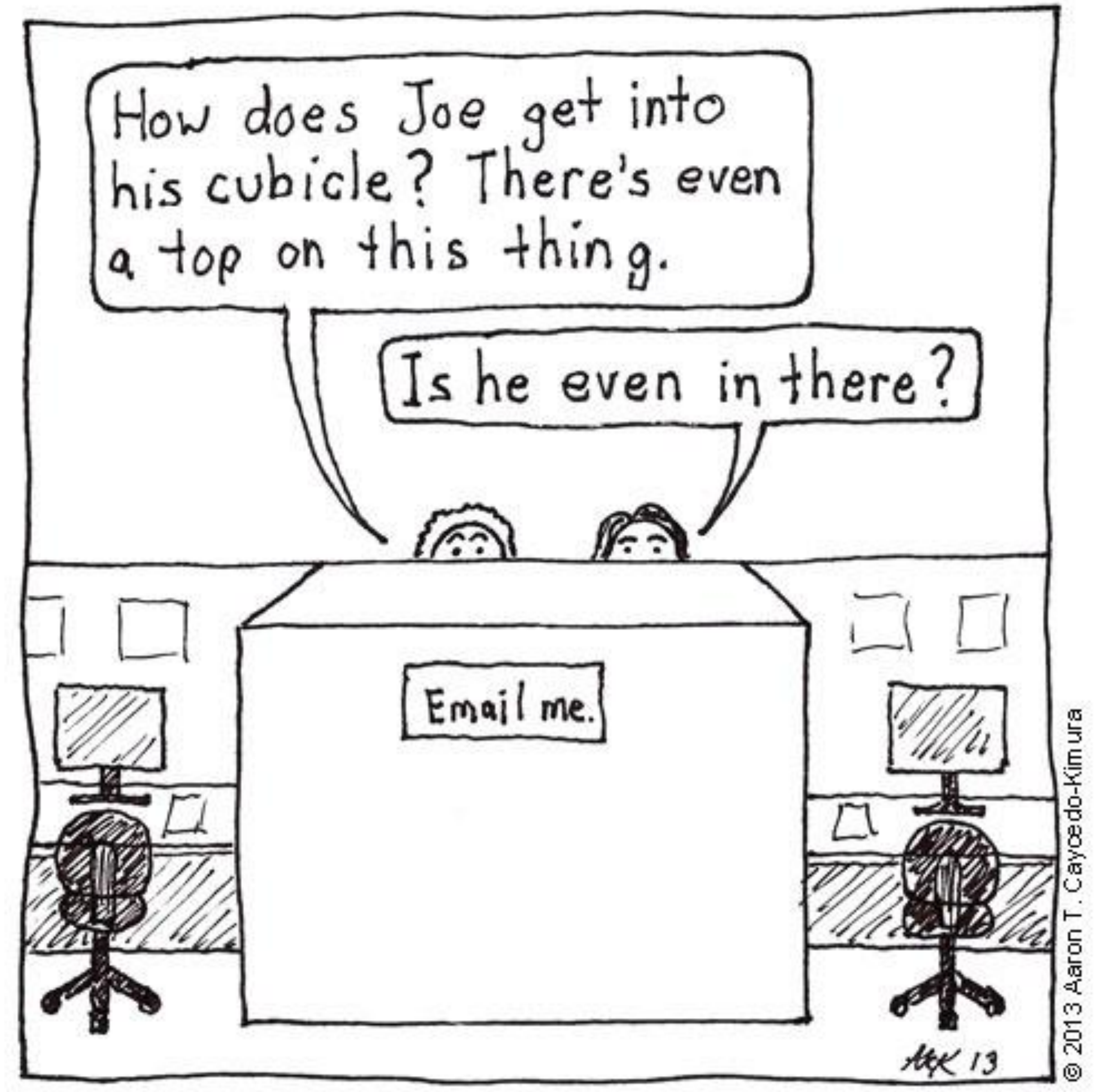
First as an individual, then as a team

Decision Quality Rank Order

All Female Teams
All Male Teams
Teams with Males & Females
Individuals

**Desert Survival Situation™, Human Synergistics International*

But, Programming In Teams Isn't Fun for Everyone


How does Joe get into his cubicle? There's even a top on this thing.
Is he even in there?
Email me.
AKK 13
© 2013 Aaron T. Cayœdo-Kimura

Watts Humphrey's Software Engineering Model

- Team Software Process
  - Launch
    - Assign roles (and respect them), estimate effort, assess risks, produce plan
  - Execution
    - Track actual effort, schedule & defects; meet weekly
  - Post Mortem
    - Assess & improve

Team Behavior

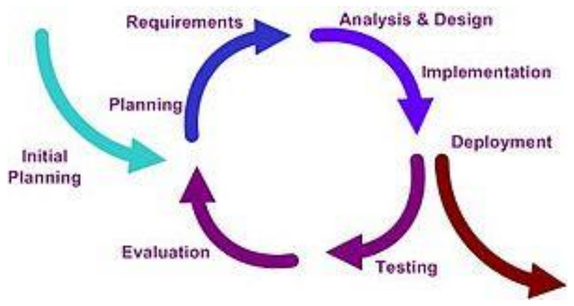
- Steve Job’s rocks in a rolling tin can with grit metaphor*
  - The forceful grinding together produces beautiful polished rocks
  - Balance contention and diplomacy
  - Suppress revenge behavior – (Humans like revenge, even at their own peril)
- Where you were raised influences your behavior**
  - Rain forest tribes - polytheistic or non-interventionist monotheistic, less likely woman are inferior
  - Desert tribes - interventionist monotheistic, warrior classes

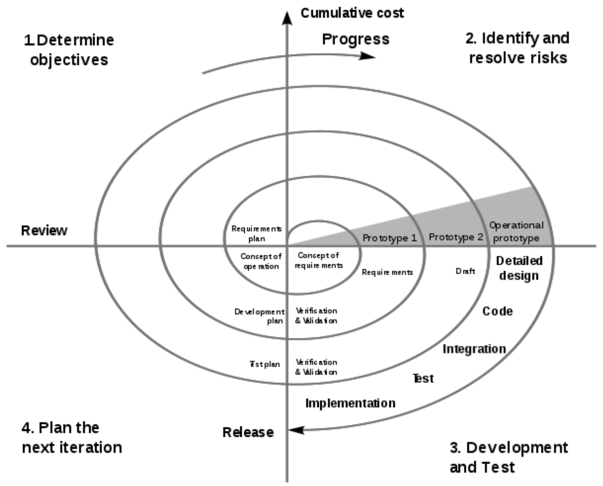
* The film: “*Steve Jobs – The Lost Interview*,” an extraordinary exposition on entrepreneurship and product innovation
**** S**tanford's Robert Textor's “Cross Cultural Summary” - tables on 400 cultures and 500 cultural traits

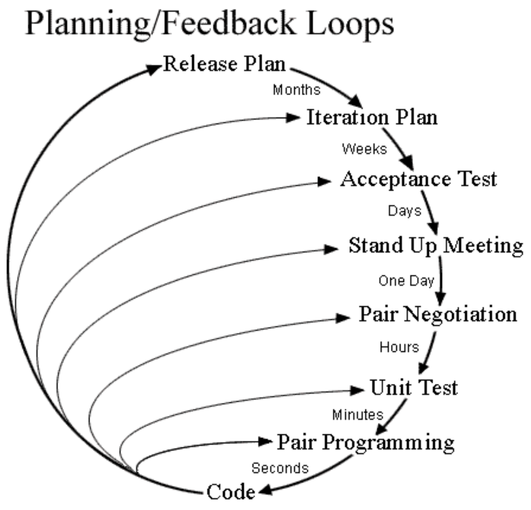
Development Process – few ideas, many names

The few ideas

- Phases, i.e., requirements, design, code, ***lots*** of test
- Little steps to refine and learn
- Inspections*/reviews to find defects







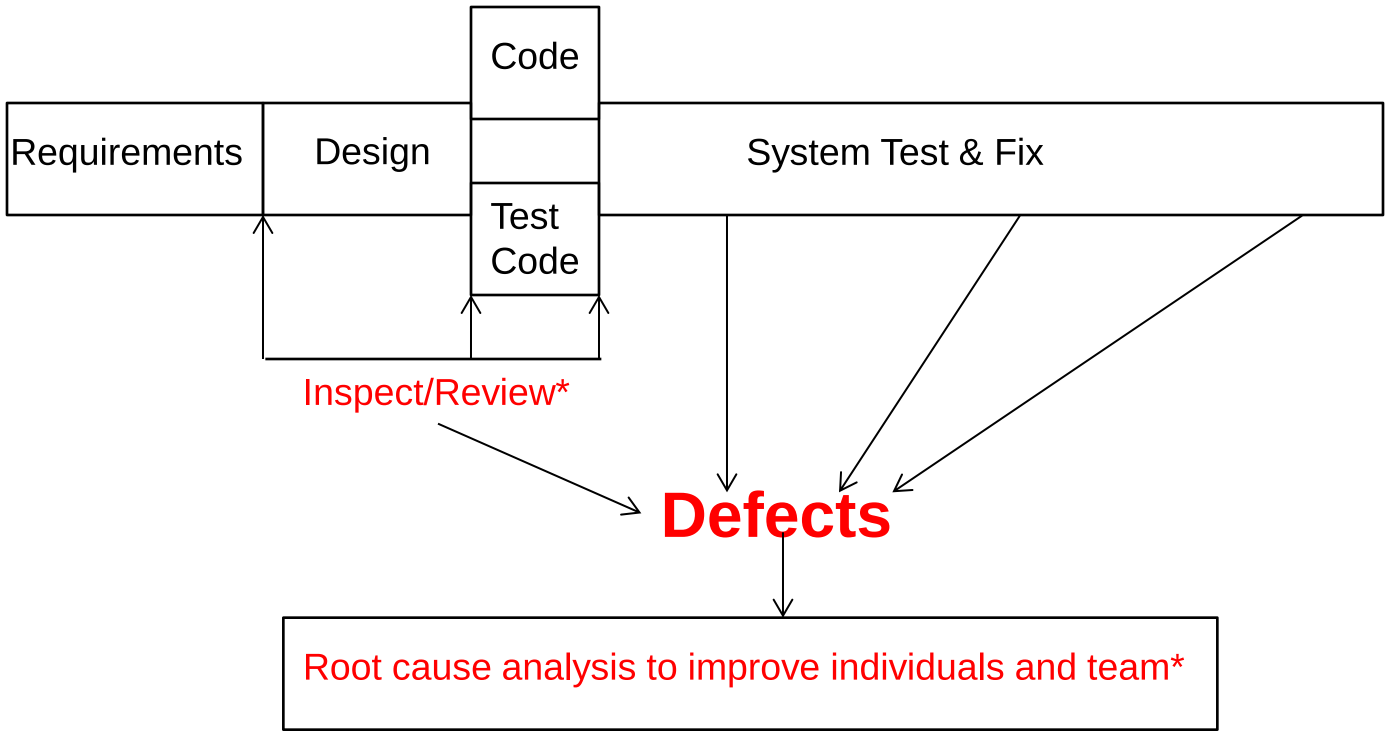
Incremental
1970's

Spiral
1988

Agile/Extreme
2001

*M.E., Fagan (1976). *"Design and Code inspections to reduce errors in program development"*. *IBM Systems Journal* 15 (3): pp. 182–211.

A Mini-Phase

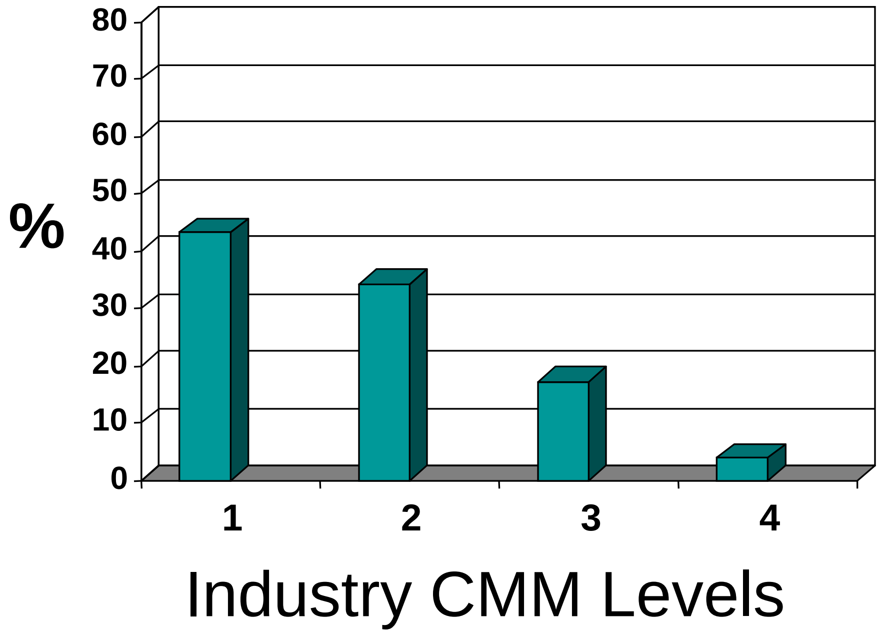


***Best In Class**

Process Discipline Works

Software Engineering Institute's Capability Maturity Model (CMM) Levels:

1. Initial
   - Cowboys are us
2. Managed
   - Configuration Management
   - Project Planning
3. Defined
   - Organizational Process Focus
   - Risk Management
4. Quantitatively Managed
   - Quantitative Project Management
5. Optimizing
   - Casual Analysis and Resolution



Improving process maturity results in annual improvements of:

1. 35% in productivity
2. 22% in defect detection
3. 19% reduced time to market
4. 39% fewer defect rates

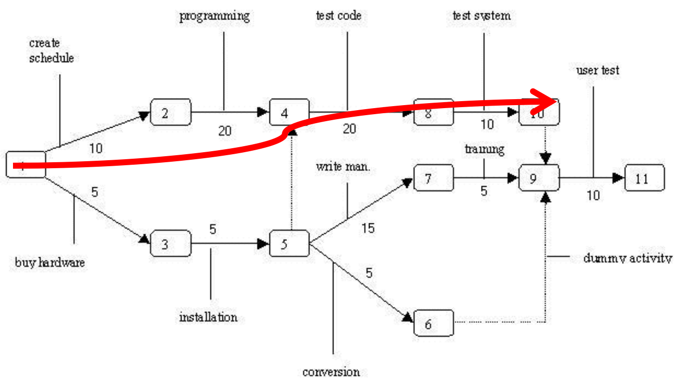
But How Much Discipline?

- Content not process makes a great product
- Discipline alone makes building lousy things predictable
- It is a function of intended use

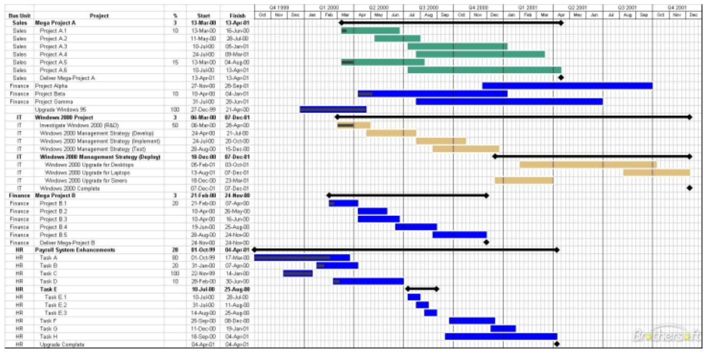
Use	**Discipline**
Test an algorithm	Very little
Typical commercial product (or your class project)	Medium
Life or significant financial exposure	Lots

The Project Plan

- The plan
  - Delicate balance of top/down and bottom/up
  - PERT to plan, GANTT to manage
  - Railroad train feature loading
- Critical path control
  - Don’t multitask
  - Buffer schedule
  - Base schedule on 50% estimates
- Call the shot
  - Best to wait until specification/design done
  - Domain expertise crucial
  - Sizing models help (+/- 30%)
  - Starvation not gluttony



PERT Chart



Gantt Chart

“Whilst it is true a large programming project might require a large programming team, a large programming team will always build a large project” - J. K. Buckle *“Managing Software Projects”*

Controlling The Schedule

- The weekly/biweekly project meeting
- Milestones
  - Early and few big milestones are crucial
  - Build team morale, tempo & customer confidence

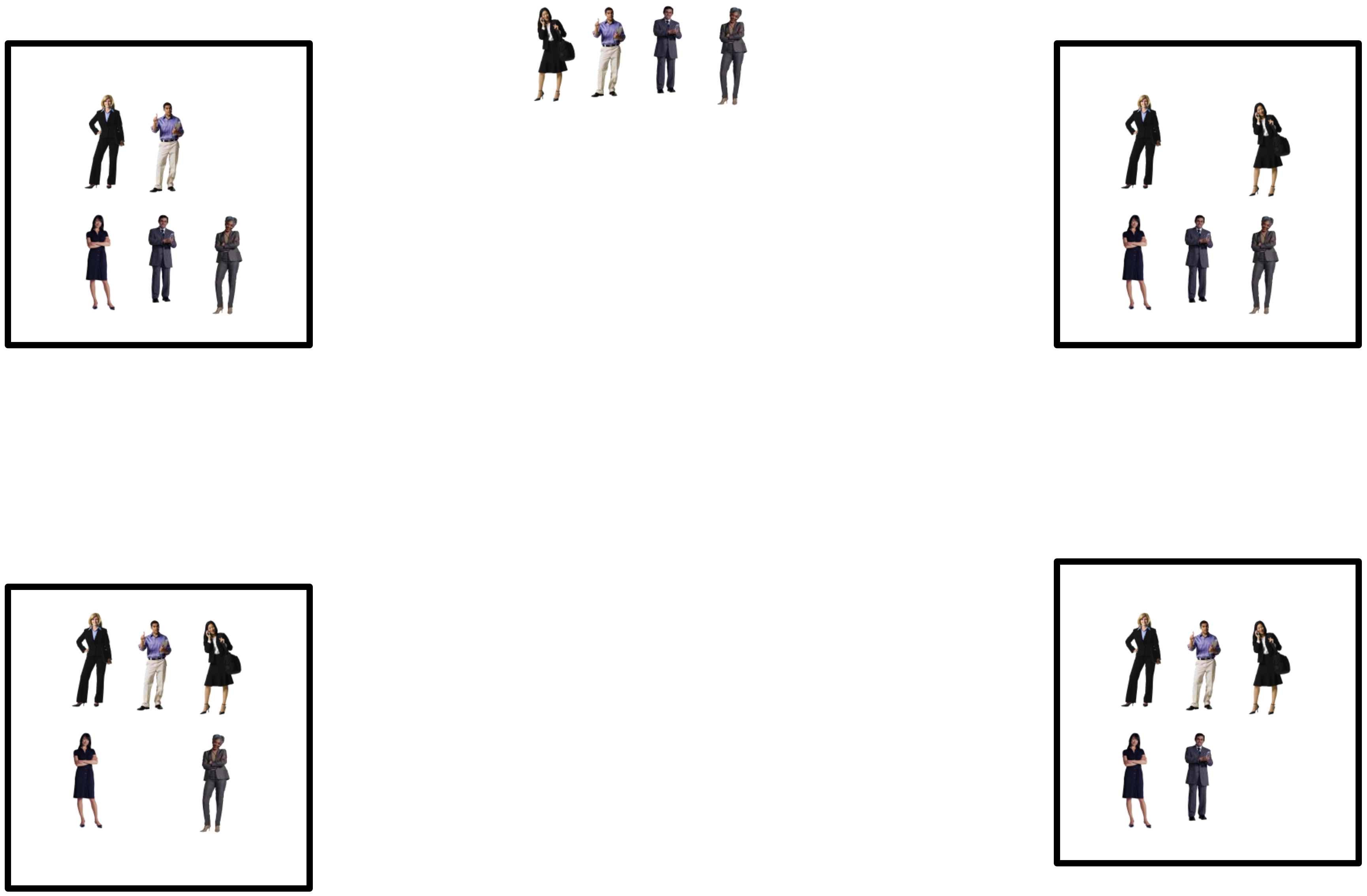
  “.. the disaster is due to termites, not tornadoes; and the schedule has slipped imperceptibly but inexorably. Indeed, major calamities are easier to handle; one responds with major force…” - Brooks
- Slips
  - If you must, do it once
  - Iterative development provides an out

“A customer will always forgive you a slipped schedule or missed feature, they will never forgive you for bad quality or bad performance” – Buckle (Perhaps, the major Obamacare Website error!)

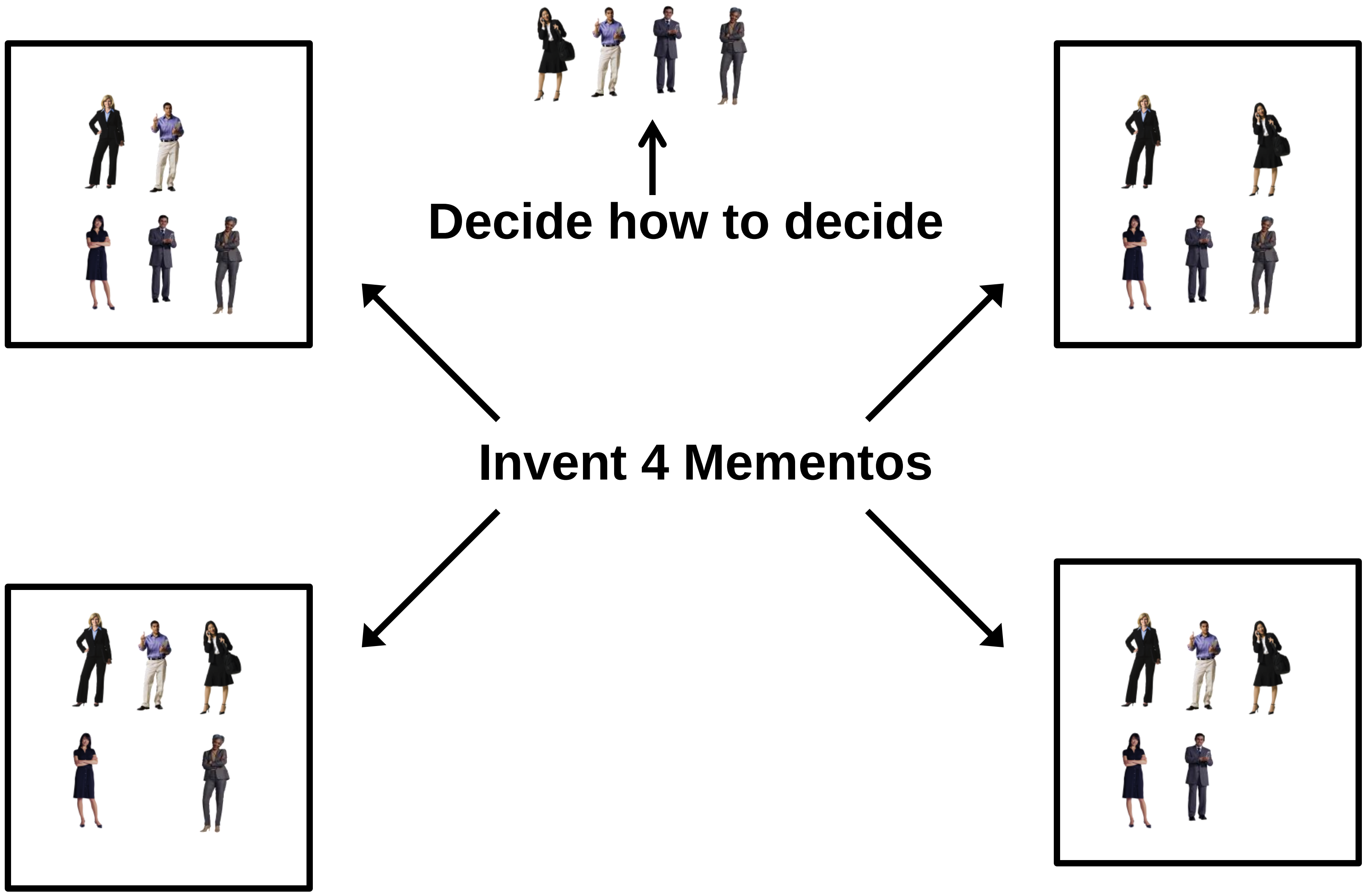
The Mementos Game – Start With A Group

Put 4 Teams In Rooms

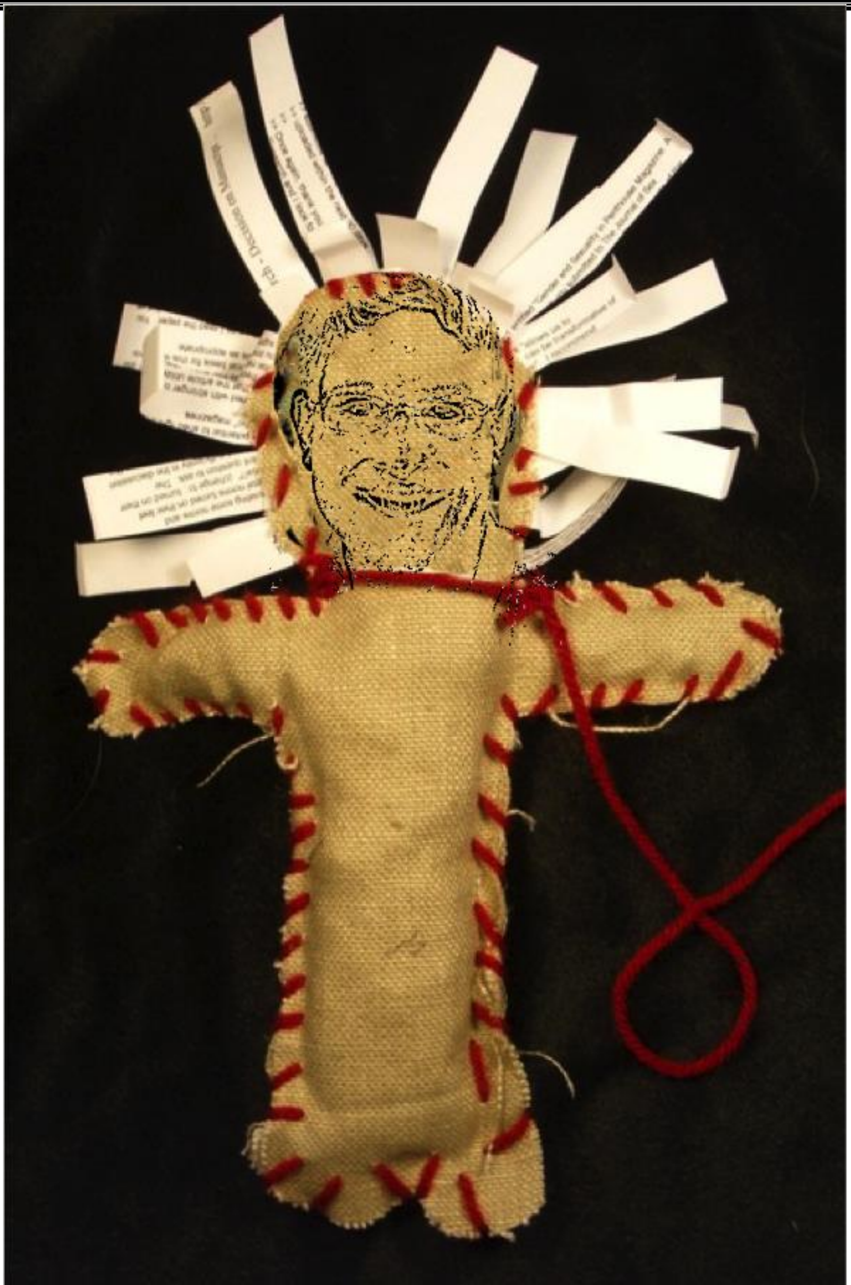
Remove 4 Judges



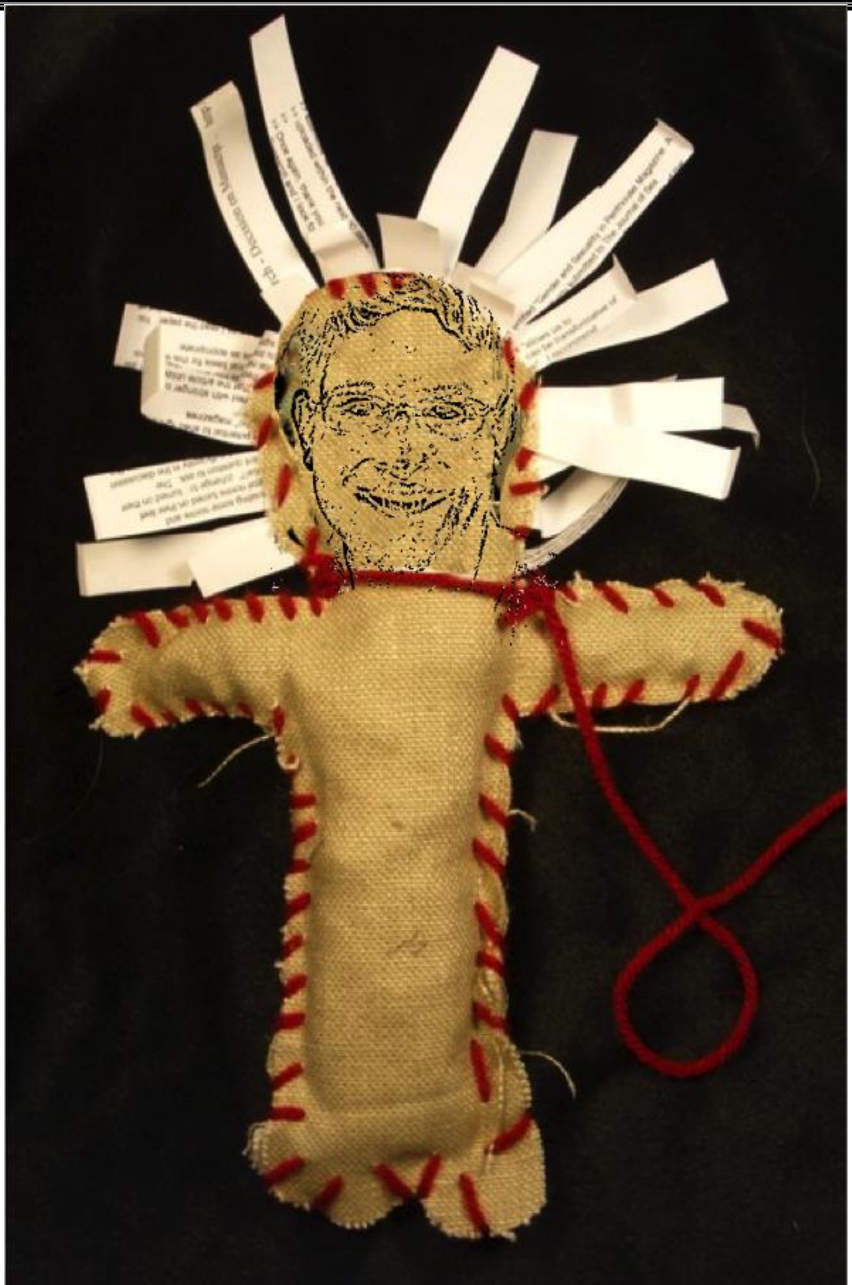
Next 30 Minutes



The Stephen VooDoo Doll, Model C - $3.95



The Stephen VooDoo Doll, Model C++ - $4.95



Another Survey

How many of you want:

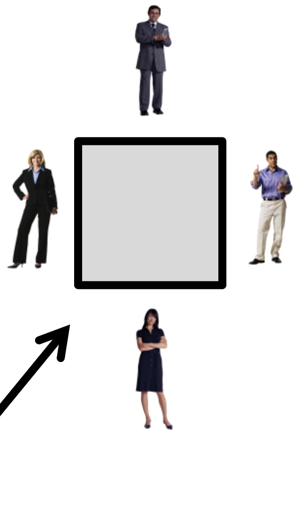
- Neither voodoo doll - no Stephen reminder!
- Model C version - $3.95 ?
- Model C++ version - $4.95?
  - The pins just might work!

Next 10 Minutes

Each team reviews all mementos and picks best

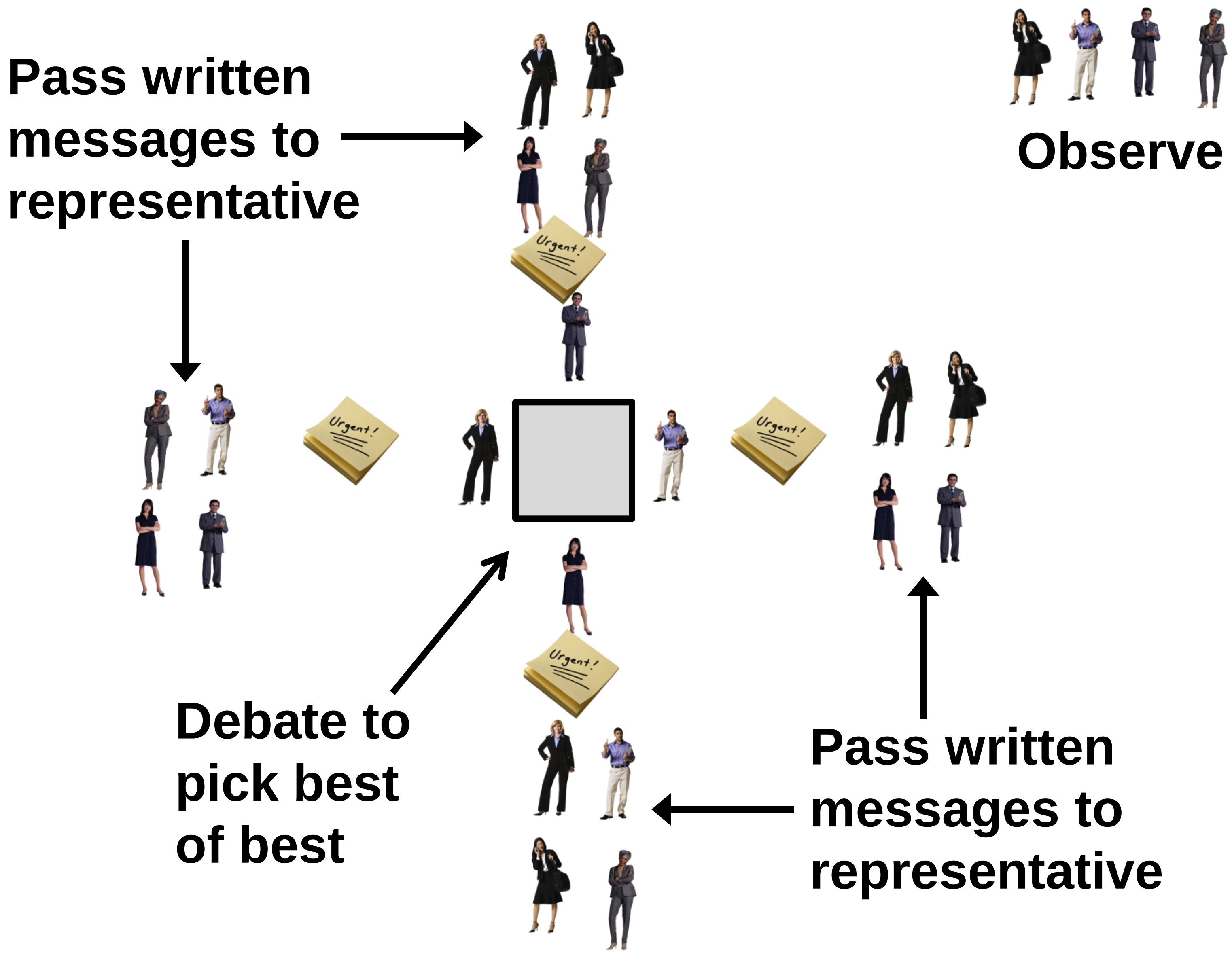
Pick The Best of Best

Observe

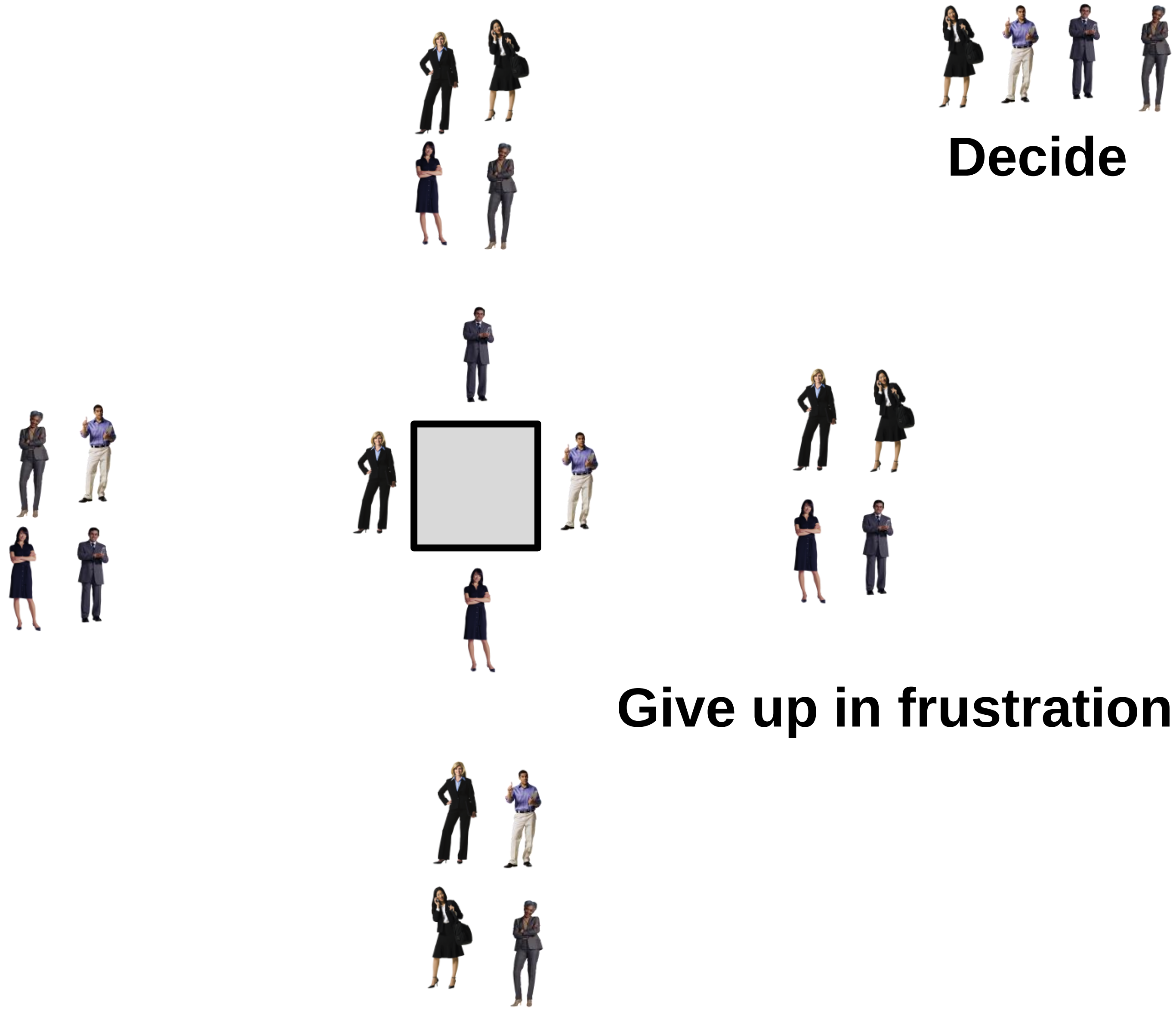


Debate to pick best of best

Next 30 Minutes


Pass written messages to representative
Observe
Urgent!
Urgent!
Urgent!
Urgent!
Debate to pick best of best
Pass written messages to representative

Next 5 Minutes



Mementos Game - Lessons

- People support what they invent
- There is cohesion within a team and competition with/derision of other teams regardless how capriciously the teams were formed.*
- Even on meaningless things, decision making is hard when ego is involved
- Decisions by a third person are often easy – team roles!!!
- Most decisions are really 50/50 and can/should be made quickly

* Scientists from Bertrand Russell through E. O. Wilson observe that the behavioral genetics of all social groups, e.g., ants and people, lead to these team/group behaviors.

Keep It Moving

“An officer in charge on an Indian agency made a requisition in the autumn for a stove costing seven dollars, certifying at the same time that it was needed to keep the infirmary warm during the winter, because the old stove wore out.”

Then, after glacial dignity

“The stove is here. So is spring” *

Project manager style

- Most decisions are 50/50
- Avoid acting like “A glacier with dignity”

* *“An Autobiography”* - Theodore Roosevelt

Team Dynamics Lessons

- Desert Survival – teams work, but ego interferes
- Memento – egos and pride interfere
- Inclusion is great – people have unique skills
- Contention can be great, but watch revenge
- Decisions – assign roles and make decisions, don’t ego debate endlessly

Leveraging Team-Member Talents

- Have team pizza dinners to bond and discover unique skills
- Use quality circles to leverage multi-cultural participation

Who *Will* You *Be?*

Experienced, high ego sleep-deprived programmers, who write lousy code

How do you plan to complete your assigned work on time and with high quality? (Check all that apply)

Estimate the size of job based on experience and cut back the project scope to meet schedule	75%
Have a detailed plan with intermediate milestones to guide my work efforts	69%
Get my teammates to do some of my work	28%
Work insanely	69%
Plead with Edwards for extension	16%
Threaten Edwards for extension	13%
Attempt to bribe Edwards for extension	16%
Contract an allegedly incurable illness for an extension	9%
Have a very old, imaginary relative die (for the third time) for an extension	16%

Final Suggestions

- Try Proven Techniques
  - Middle weight development process, iterative/spiral/Agile
  - Defect root cause analysis
  - Inspections/reviews
    - Architecture & usability (with another team)
    - Code
  - Automated testing
- Build A High-Performance Team
  - Have pizza dinners
  - Debate as an inclusive team
  - Assign roles and decision authorities
- Read
  - Lion’s “*Commentary on Unix”*
  - Fred Brooks’ *“The Mythical Man Month”*
- Watch *“Steve Jobs – The Lost Interview”*
- If all this fails, send Edwards for another Taiwanese McDonalds’ modeling engagement

Tenrecs – Each Unique With Special Talents

Backup

Dev Bootcamp

- 19 weeks - 1000 hrs in 9-week immersion, recommend sleep the other 500
- 12:1 student teacher ratio
- $500 room in hacker houses
  - Provides computers, yoga, stretching, meditation and mindfulness training, career week
- Results - $76K average salary, 75% employed
- Skills
  - Ruby
  - Agile Development
  - Rails Application Framework
  - Ajax, jQuery
  - Test Driven Development
  - Intra & inter personal issues that hinder teamwork
  - HTML & CSS
  - Git & Source Control
  - Pair-programming, code reviews
  - Interview and presentation skills

Learning

- Disasters
  - Mine
  - Others
- Really good companies
  - IBM - Discipline & inspections
  - Sun - Tempo
  - Japan - Quality circles & reuse
  - Microsoft - Reuse
  - Apple - User centered design & tool kits, *"Design of Everyday Things"* - Don Norman
- Really smart professionals
  - Al Aho – Swamp drainer
  - Fred Brooks – *"Mythical Man Month," "No Silver Bullet"*
  - Barry Boehm – Estimation and cost of errors
  - Michael Cusumano – Software factories
  - Edsger Dijkstra – Importance of time in systems
  - Watts Humprehy - Encapsulated team and personal discipline - *"TSP, PSP"*
  - Martyn Thomas - Formal methods
  - Ken Thompson - The programming craft

What makes a program(er) good?

- Has correct functionality, performance and security
- Has a low defect rate
- Was implemented with reasonable productivity
- Can be maintained and evolved by others
  - Its style has good “taste”
- Uses appropriate algorithms and libraries

Facts of Life

- Failure is the norm
  - But it needn’t be
- Error correction costs increase exponentially during the lifecycle
- Programmers are a special breed*
  - They like machines not people
  - They have high need for approval/recognition
  - They must respect the leader
  - Beware, complicators and prima donnas lurk in the weeds
- Software engineering is complexity management
  - Race with technology/tools
  - Complexity is still ahead
- Good software engineering is old hat
  - But is not always used - why????

  “Two things are infinite: the universe and human stupidity; and I'm not sure about the the universe” - Einstein

* “*Psychology of Computer Programmer* ” - Gerald Weinberg

Reaching Consensus

- Important
  - Takes teams to do complex things
  - Team decisions are better than individual decisions
  - Must be effective for speed
- Hard
  - Ego, individual and team
  - Conflicting goals
  - Lack of decision criteria/expertise
- Complex - Business School Pet Topic
  - Leadership, e.g., the cult of the personality
  - Sources of power, e.g., charisma, common enemy
  - Organizational structure
  - Decision models
  - Game theory

A Leadership Decision Model*

Autocratic **Consultative** **Group**

Speed $\longleftrightarrow$ **Buy-in**

$\longleftarrow$ **Conflicting goals**

Leader $\longleftarrow$ **Expertise** $\longrightarrow$ **Group**

Low $\longleftarrow$ **Decision importance** $\longrightarrow$ **High**

*** Vic Vroom, Yale**

More Madagascar Diversity