How to Have a Bad Career in Research/Academia

Professor David A. Patterson

November 2001

www.cs.berkeley.edu/~pattrsn/talks/BadCareer.pdf

Outline

* **Part I: Key Advice for a Bad Career while a Grad Student**
* **Part II: Key Advice on Alternatives to a Bad Graduate Career**
* **Part III: Key Advice for a Bad Career, Post Ph.D.**
* **Part IV: Key Advice on Alternatives to a Bad Career, Post Ph.D.**
* **Topics covered in Parts III and IV**
  - **Selecting a Problem**
  - **Picking a Solution**
  - **Performing the Research**
  - **Evaluating the Results**
  - **Communicating Results**
  - **Transferring Technology**

Part I: How to Have a Bad Graduate Career

* **Concentrate on getting good grades:**
  - **postpone research involvement: might lower GPA**
* **Minimize number and flavors of courses**
  - **Why take advantage of 1 of the top departments with an emphasis on excellent grad courses?**
  - **Why take advantage of a campus with 35/36 courses in the top 10?**
  - **May affect GPA**
* **Don t trust your advisor**
  - **Advisor is only interested in his or her own career, not yours**
  - **Advisor may try to mentor you, use up time, interfering with GPA**
* **Only work the number of hours per week you are paid!**
  - **Don't let master class exploit the workers!**

Part I: How to Have a Bad Graduate Career

* Concentrate on graduating as fast as possible
  - Winner is first in class to Ph.D.
  - People only care about that you have a Ph.D. and your GPA, not on what you know
    - » Nirvana: graduating in 3.5 years with a 4.0 GPA!
  - Don't spend a summer in industry: takes longer
    - » How could industry experience help with selecting Ph.D. topic?
  - Don't work on large projects: takes longer
    - » Have to talk to others, have to learn different areas
    - » Synchronization overhead of multiple people
  - Don't do a systems Ph.D.: takes longer
* Don t go to conferences
  - It costs money and takes time; you'll have plenty of time to learn the field after graduating
* Don t waste time polishing writing or talks
  - Again, that takes time

Part II: Alternatives to a Bad Graduate Career

* **Concentrate on getting good grades?**
  - **Reality: need to maintain reasonable grades**
    - » **Only once gave a below B in CS 252**
    - » **3 prelim courses only real grades that count**
  - **What matters on graduation is letters of recommendation from 3-4 faculty/Ph.D.s who have known you for 5+ years**
* **Minimize number and flavors of courses?**
  - **Your last chance to be exposed to new ideas before have to learn them on your own (re: queueing theory and me)**
  - **Get a real outside minor from a campus with great departments in all fields; e.g., Management of Technology certificate, Copyright Law**
* **Don t trust your advisor?**
  - **Primary attraction of campus vs. research lab is getting to work with grad students**
  - **Faculty career is judged in large part by success of his or her students**
  - **try taking advice of advisor?**

Part II: Alternatives to a Bad Graduate Career

* **Concentrate on graduating as fast as possible?**
  - Your last chance to learn; most learning will be outside the classroom
  - Considered newly "minted" when finish Ph.D.
    - » Judged on year of Ph.D. vs. year of birth
    - » To a person in their 40s or 50s, 1 or 2 more years is roundoff error (27 = 29)
* **Don t go to conferences?**
  - Chance to see firsthand what the field is like, where its going
  - There are student rates, you can share a room
  - Talk to people in the field in the halls
  - If your faculty advisor won't pay, then pay it yourself; almost always offer student rates, can often share rooms
    - » Prof. Landay paid his own way to conferences while grad student
* **Don t waste time polishing writing or talks?**
  - In the marketplace of ideas, the more polish the more likely people will pay attention to your ideas
  - Practice presentation AND answering tough questions

Part II: Alternatives to a Bad Graduate Career

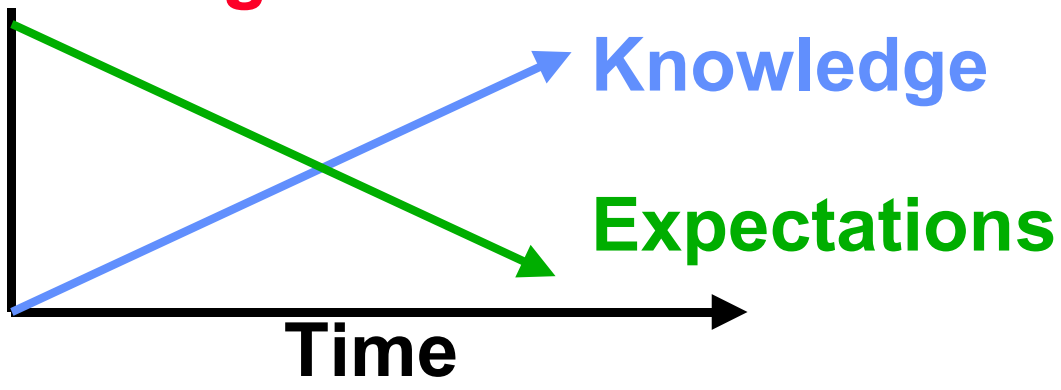
* Only work the number of hours per week you are paid?
  - Campus Faculty average is 65-70 hours/work; EECS higher
  - Students should be in that range
  - Organize each day: when most alert? nap? exercise? sleep?
  - When/how often/how long: write, read, program, email?
  - To do lists: daily, weekly, semester
* Industrial Experience?
  - 1st or 2nd summer get work experience, or 1 semester off
* Sutherland s advice (Father of Computer Graphics)
  - Be bold; Take chances on hard topics
  - see *Technology and Courage* URL on CS252, or search on Google
* Advice from a very successful recent student; Remzi Arpaci
  - Great ideas, did lots of papers, well thought of
  - I asked: Why do you think you did so well?
  - He said I gave him advice the first week he arrived
  - I asked: What did I say?
  - He said 3 observations, and still good advice today

Part II: How to be a Success in Graduate School

* 1) **Swim or Sink**
  - —Success is de termined by me (student) primarily
  - —Faculty will set up the opportunity, but its up to me leverage it
* 2) **Read/learn on your own**
  - —Related to 1), I think you told me this as you handed me a stack of about 20 papers
* 3) **Teach your advisor**
  - —I really liked this concept; go out and learn about something and then teach the professor
  - —Fast moving field, don t expect Prof to be at forefront everywhere

Summary Advice of Alternative to Bad Career in Graduate School

* **Show Initiative!**
  - don't wait for advisor (or more senior grad students) to show you what to do
* **Ask questions!**
  - lots of smart people in grad school (and even on the faculty), but don't be intimidated.
  - Either they know and you will learn, or they don't know and you will all learn by trying to determine the answer
* **When to graduate**



Outline

* Part I: Key Advice for a Bad Career while a Grad Student
* Part II: Key Advice on Alternatives to a Bad Graduate Career
* **Part III: Key Advice for a Bad Career, Post Ph.D.**
* **Part IV: Key Advice on Alternatives to a Bad Career, Post Ph.D.**
* **Topics covered in Parts III and IV**
  - **Selecting a Problem**
  - **Picking a Solution**
  - **Performing the Research**
  - **Evaluating the Results**
  - **Communicating Results**
  - **Transferring Technology**

Bad Career Move #1: Be THE leading expert

* Invent a new field!
  - Make sure its slightly different
* Be the real Lone Ranger: Don t work with others
  - No ambiguity in credit
  - Adopt the Prima Donna personality
* Research Horizons
  - Never define success
  - Avoid Payoffs of less than 20 years
  - Stick to one topic for whole career
  - Even if technology appears to leave you behind, stand by your problem

Announcing a New Operating System Field: Disability-Based Systems

* Computer Security is increasingly important
  - Insight: capability-based addressing almost right
* Idea: Create list of things that process CANNOT do!
* Key Question:
  should you store disabilities with each user
  or with the objects they can t access?
* Other topics: encrypted disabilities,
  disability-based addressing
* Start a new sequence of courses and new journal on
  Theory and Practice of Disability-Based Systems

Bad Career Move #2: Let Complexity Be Your Guide (Confuse Thine Enemies)

* **Best compliment:**
  Its so complicated, I can t understand the ideas
* **Easier to claim credit for subsequent good ideas**
  - **If no one understands, how can they contradict your claim?**
* **It s easier to be complicated**
  - **Also: to publish it must be different; N+1st incremental change**
* **If it were not unsimple then how could distinguished colleagues in departments around the world be positively appreciative of both your extraordinary intellectual grasp of the nuances of issues as well as the depth of your contribution?**

Bad Career Move #3: Never be Proven Wrong

* Avoid Implementing
* Avoid Quantitative Experiments
  - If you've got good intuition, who needs experiments?
  - Why give grist for critics' mill?
  - Takes too long to measure
* Avoid Benchmarks
* Projects whose payoff is >= 20 years
  gives you 19 safe years

Bad Career Move #4:
Use the Computer Scientific Method

Obsolete Scientific Method

* Hypothesis
* Sequence of experiments
* Change 1 parameter/exp.
* Prove/Disprove Hypothesis
* Document for others to reproduce results

Computer Scientific Method

* Hunch
* 1 experiment & change all parameters
* Discard if doesn t support hunch
* Why waste time? We know this

Bad Career Move #5:
Don t be Distracted by Others (Avoid Feedback)

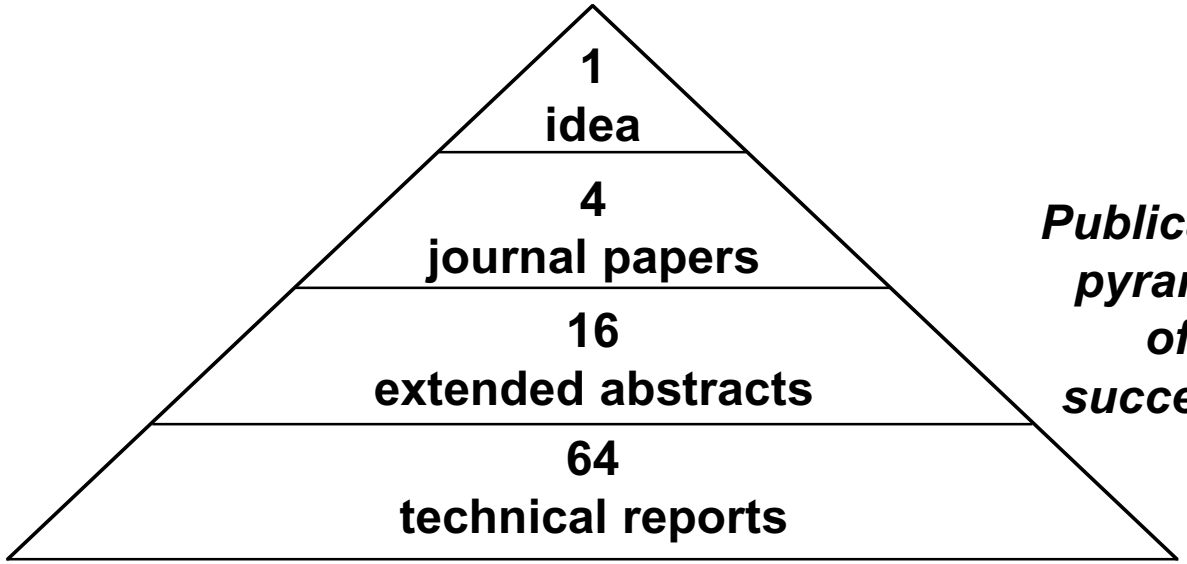
* Always dominate conversations: Silence is ignorance
  - Corollary: Loud is smart
* Don t read
* Don t be tainted by interaction with users, industry
* Reviews
  - If it's simple and obvious in retrospect => Reject
  - Quantitative results don't matter if they just show you what you already know => Reject
  - Everything else => Reject

Bad Career Move #6:
Publishing Journal Papers IS Technology Transfer

* **Target Archival Journals: the Coin of the Academic Realm**
  - **It takes 2 to 3 years from submission to publication=>timeless**
* **As the leading scientist, your job is to publish in journals; its *not* your job to make you the ideas palatable to the ordinary engineer**
* **Going to conferences and visiting companies just uses up valuable research time**
  - **Travel time, having to interact with others, serve on program committees, ...**

Bad Career Move #7:
Writing Tactics for a Bad Career

* **Papers: It s Quantity, not Quality**
  - **Personal Success = Length of Publication List**
  - **"The LPU (Least Publishable Unit) is Good for You"**



Publication pyramid of success

* **Student productivity = number of papers**
  - **Number of students: big is beautiful**
  - **Never ask students to implement: reduces papers**
* **Legally change your name to Aaaanderson**

5 Writing Commandments for a Bad Career

I. Thou shalt not define terms, nor explain anything.
II. Thou shalt replace will do with have done .
III. Thou shalt not mention drawbacks to your approach.
IV. Thou shalt not reference any papers.
V. Thou shalt publish before implementing.

7 Talk Commandments for a Bad Career

I. Thou shalt not illustrate.
II. Thou shalt not covet brevity.
III. Thou shalt not print large.
IV. Thou shalt not use color.
V. Thou shalt cover thy naked slides.
VI. Thou shalt not skip slides in a long talk.
VII. Thou shalt not practice.

Following all the commandments

* We describe the philosophy and design of the control flow machine, and present the results of detailed simulations of the performance of a single processing element. Each factor is compared with the measured performance of an advanced von Neumann computer running equivalent code. It is shown that the control flow processor compares favorably in the program.

* We present a denotational semantics for a logic program to construct a control flow for the logic program. The control flow is defined as an algebraic manipulator of idempotent substitutions and it virtually reflects the resolution deductions. We also present a bottom-up compilation of medium grain clusters from a fine grain control flow graph. We compare the basic block and the dependence sets algorithms that partition control flow graphs into clusters.

* A hierarchical macro-control-flow computation allows them to exploit the coarse grain parallelism inside a macrotask, such as a subroutine or a loop, hierarchically. We use a hierarchical definition of macrotasks, a parallelism extraction scheme among macrotasks defined inside an upper layer macrotask, and a scheduling scheme which assigns hierarchical macrotasks on hierarchical clusters.

* We apply a parallel simulation scheme to a real problem: the simulation of a control flow architecture, and we compare the performance of this simulator with that of a sequential one. Moreover, we investigate the effect of modeling the application on the performance of the simulator. Our study indicates that parallel simulation can reduce the execution time significantly if appropriate modeling is used.

* We have demonstrated that to achieve the best execution time for a control flow program, the number of nodes within the system and the type of mapping scheme used are particularly important. In addition, we observe that a large number of subsystem nodes allows more actors to be fired concurrently, but the communication overhead in passing control tokens to their destination nodes causes the overall execution time to increase substantially.

* The relationship between the mapping scheme employed and locality effect in a program are discussed. The mapping scheme employed has to exhibit a strong locality effect in order to allow efficient execution. We assess the average number of instructions in a cluster and the reduction in matching operations compared with fine grain control flow execution.

* Medium grain execution can benefit from a higher output bandwidth of a processor and finally, a simple superscalar processor with an issue rate of ten is sufficient to exploit the internal parallelism of a cluster. Although the technique does not exhaustively detect all possible errors, it detects nontrivial errors with a worst-case complexity quadratic to the system size. It can be automated and applied to systems with arbitrary loops and nondeterminism.

7 Poster Commandments for a Bad Career

I. Thou shalt not illustrate.
II. Thou shalt not covet brevity.
III. Thou shalt not print large.
IV. Thou shalt not use color.
V. Thou shalt not attract attention to thyself.
VI. Thou shalt not prepare a short oral overview.
VII. Thou shalt not prepare in advance.

Following all the commandments

How to Do a Bad Poster
David Patterson
University of California
Berkeley, CA 94720

We describe the philosophy and design of the control flow machine, and present the results of detailed simulations of the performance of a single processing element. Each factor is compared with the measured performance of an advanced von Neumann computer running equivalent code. It is shown that the control flow processor compares favorably in the program.

We present a denotational semantics for a logic program to construct a control flow for the logic program. The control flow is defined as an algebraic manipulator of idempotent substitutions and it virtually reflects the resolution deductions. We also present a bottom-up compilation of medium grain clusters from a fine grain control flow graph. We compare the basic block and the dependence sets algorithms that partition control flow graphs into clusters.

Our compiling strategy is to exploit coarse-grain parallelism at function application level: and the function application level parallelism is implemented by fork-join mechanism. The compiler translates source programs into control flow graphs based on analyzing flow of control, and then serializes instructions within graphs according to flow arcs such that function applications, which have no control dependency, are executed in parallel.

A hierarchical macro-control-flow computation allows them to exploit the coarse grain parallelism inside a macrotask, such as a subroutine or a loop, hierarchically. We use a hierarchical definition of macrotasks, a parallelism extraction scheme among macrotasks defined inside an upper layer macrotask, and a scheduling scheme which assigns hierarchical macrotasks on hierarchical clusters.

We apply a parallel simulation scheme to a real problem: the simulation of a control flow architecture, and we compare the performance of this simulator with that of a sequential one. Moreover, we investigate the effect of modeling the application on the performance of the simulator. Our study indicates that parallel simulation can reduce the execution time significantly if appropriate modeling is used.

We have demonstrated that to achieve the best execution time for a control flow program, the number of nodes within the system and the type of mapping scheme used are particularly important. In addition, we observe that a large number of subsystem nodes allows more actors to be fired concurrently, but the communication overhead in passing control tokens to their destination nodes causes the overall execution time to increase substantially.

The relationship between the mapping scheme employed and locality effect in a program are discussed. The mapping scheme employed has to exhibit a strong locality effect in order to allow efficient execution. We assess the average number of instructions in a cluster and the reduction in matching operations compared with fine grain control flow execution.

Medium grain execution can benefit from a higher output bandwidth of a processor and finally, a simple superscalar processor with an issue rate of ten is sufficient to exploit the internal parallelism of a cluster. Although the technique does not exhaustively detect all possible errors, it detects nontrivial errors with a worst-case complexity quadratic to the system size. It can be automated and applied to systems with arbitrary loops and nondeterminism.

Outline

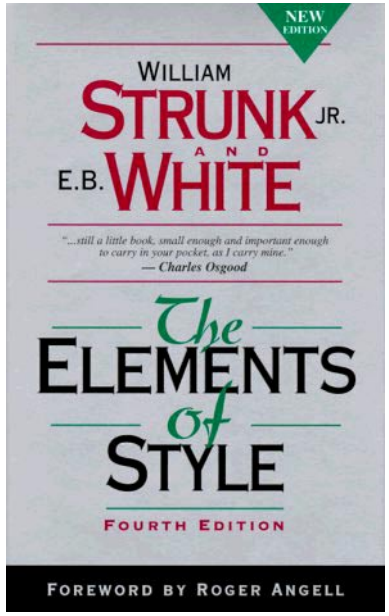
* Part I: Key Advice for a Bad Career while a Grad Student
* Part II: Key Advice on Alternatives to a Bad Graduate Career
* Part III: Key Advice for a Bad Career, Post Ph.D.
* **Part IV: Key Advice on Alternatives to a Bad Career, Post Ph.D.**
* **Topics covered in Parts III and IV**
  - **Selecting a Problem**
  - **Picking a Solution**
  - **Performing the Research**
  - **Evaluating the Results**
  - **Communicating Results**
  - **Transferring Technology**

Alternatives to Bad Papers

* **Do opposite of Bad Paper commandments**
  - **Define terms; list related work; be honest about state of prototype; honest about performance, weaknesses**
* **Find related work via Melvyl/INSPEC online search/paper retrieval**

  `www.dbs.cdlib.org`
* **Read Strunk and White, then follow these steps;**
  1. **1-page paper outline, with tentative page budget/section**
  2. **Paragraph map**
     - » **1 topic phrase/sentence per paragraph, handdrawn figures w. captions**
  3. **(Re)Write draft**
     - » **Long captions/figure can contain details ~ Scientific American**
     - » **Uses Tables to contain facts that make dreary prose**
  4. **Read aloud, spell check & grammar check (MS Word "technical writing" mode)**
  5. **Get feedback from friends and critics on draft; go to 3.**
* **Also see my advice on writing:**

`www.cs.berkeley.edu/~pattrsn/talks/writingtips.html`



Alternatives to Bad Talks

* **Do opposite of Bad Talk commandments**
  - **I. Thou shalt not illustrate.**
  - **II. Thou shalt not covet brevity.**
  - **III. Thou shalt not print large.**
  - **IV. Thou shalt not use color.**
  - **V. Thou shalt cover thy naked slides.**
  - **VI. Thou shalt not skip slides in a long talk.**
  - **VII. Thou shalt not practice.**
* **Allocate 2 minutes per slide, leave time for questions**
* **Don t over animate**
* **Do dry runs with friends/critics for feedback,**
  - **including tough audience questions**
* **Tape a practice talk (audio tape or video tape)**
  - **Don't memorize speech, have notes ready**
* **Bill Tetzlaff, IBM: Giving a first class job talk is the single most important part of an interview trip. Having someone know that you can give an excellent talk before hand greatly increases the chances of an invitation. That means great conference talks.**

Alternatives to Bad Posters (from Randy Katz)

* **Do opposite of Bad Poster commandments**
  - **Poster tries to catch the eye of person walking by**
* **Answer Five Heilmeier Questions**
  1. **What is the problem you are tackling?**
  2. **What is the current state-of-the-art?**
  3. **What is your key make-a-difference concept or technology?**
  4. **What have you already accomplished?**
  5. **What is your plan for success?**
* **9 page poster might look like**

Problem Statement	State-of-the-Art	Key Concept
Accomplish-ment # 1	Title and Visual logo	Accomplish-ment # 2
Accomplish-ment # 3	Plan for Success	Summary & Conclusion

ROC: Recovery-Oriented Computing

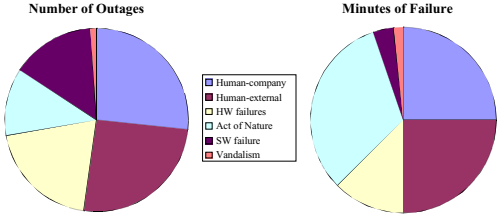
Aaron Brown and David Patterson
ROC Research Group, EECS Division, University of California at Berkeley

For more info: http://roc.cs.berkeley.edu

AME is the 21st Century Challenge

- Availability
  - systems should continue to meet quality of service goals despite hardware and software failures
- Maintainability
  - systems should require only minimal ongoing human administration, regardless of scale or complexity: Today, cost of maintenance = 10X cost of purchase
- Evolutionary Growth
  - systems should evolve gracefully in terms of performance, maintainability, and availability as they are grown/upgraded/expanded
- Performance was the 20th Century Challenge
  - 1000X Speedup suggests problems are elsewhere

People are the biggest challenge



- People > 50% outages/minutes of failure
  - Sources of Failure in the Public Switched Telephone Network, Kuhn; IEEE Computer, 30:4 (Apr 97)
  - FCC Records 1992-1994; Overload (not sufficient switching to lower costs) + 6% outages, 44% minutes

Recovery-Oriented Computing (ROC) Hypothesis

If a problem has no solution, it may not be a problem, but a fact, not to be solved, but to be coped with over time

Shimon Peres

- Failures are a fact, and recovery/repair is how we cope with them
- Improving recovery/repair improves availability
  - $\text{Availability} = \frac{\text{MTTF}}{(\text{MTTF} + \text{MTTR})}$
  - Since MTTF >> MTTR, 1/10th MTTR just as valuable as 10X MTBF
- Since major Sys Admin job is recovery after failure, ROC also helps with maintenance

ROC Principles: (1) Isolation and redundancy

- System is partitionable
  - to isolate faults
  - to enable online repair/recovery
  - to enable online HW growth/SW upgrade
  - to enable operator training/expand experience on portions of real system
  - Techniques: Geographically replicated sites, Shared nothing cluster, Separate address spaces inside CPU
- System is redundant
  - sufficient HW redundancy/data replication => part of system down but satisfactory service still available
  - enough to survive failure or more during recovery
  - Techniques: RAID; N copies of data

ROC Principles: (2) Online verification

- System enables input insertion, output check of all modules (including fault insertion)
  - to check module operation to find failures faster
  - to test correctness of recovery mechanisms
    - insert faults and known incorrect inputs
    - also enables availability benchmarks
  - to test if proposed solution fixed the problem
    - discover whether need to try another solution
  - to discover if warning systems are broken
  - to expose and remove latent errors from each system
  - to train/expand experience of operator
  - Techniques: Global invariants; Topology discovery; Program checking (SW ECC)

ROC Principles: (3) Undo Support

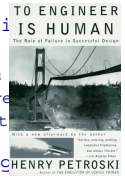
- ROC system should offer Undo
  - to recover from operator errors
    - undo is ubiquitous in productivity apps
    - should have undo for maintenance
  - to recover from inevitable SW errors
    - restore entire system state to prior version
  - to recover from operator training-mistakes
  - to replace traditional backup and restore
  - Techniques: Checkpointing; Logging; and time travel (log structured) file systems

ROC Principles: (4) Diagnosis Support

- System assists human in diagnosing problems
  - root-cause analysis to suggest possible failure points
    - track resource dependencies of all requests
    - correlate symptomatic requests with component dependency model to isolate culprit components
  - health reporting to detect failed/failing components
    - failure information, test results propagated upwards
  - unified status console to highlight improper behavior, predict failure, and suggest corrective action
  - Techniques: Stamp data blocks with modules used; Log faults, errors, failures and recovery methods

Lessons Learned from Other Fields

- 1800s: 25% railroad bridges fail
- Techniques invented since:
  - Learn from failures vs. successes
  - Redundancy to survive some failures
  - Margin of safety—3X times calculated load to cover what they don't know



- Safety now in Civil Engineering
  - Structural engineering is the science and art of designing and making, with economy and elegance, structures that can safely resist the forces to which they may be subjected
- Have we been building the computing equivalent of the 19th Century iron truss bridges?
  - What is computer equivalent of safety margin?

Recovery-Oriented Computing Conclusion

- New century needs new research agenda
  - (and its not performance)
- Embrace failure of HW, SW people and still build systems that work
- ROC: Significantly reduce Time to Recover/Repair => much greater availability + much lower maintenance costs

Legendary great bird of Arab folklore, the Roc is known to be of such huge size that it can carry off elephants and other great land beasts with its large feet. Sinbad the Sailor encountered such a bird in The Thousand and One Nights.

One Alternative Strategy to a Bad Career

* **Caveats:**
  - **From a project leader's point of view**
  - **Works for me; not the only way**
  - **Primarily from academic, computer systems perspective**
* **Goal is to have impact:**
  Change way people do Computer Science & Engineering
  - **Academics have bad benchmarks: published papers**
* **6 Steps**
  1) **Selecting a problem**
  2) **Picking a solution**
  3) **Running a project**
  4) **Finishing a project**
  5) **Quantitative Evaluation**
  6) **Transferring Technology**

1) Selecting a Problem

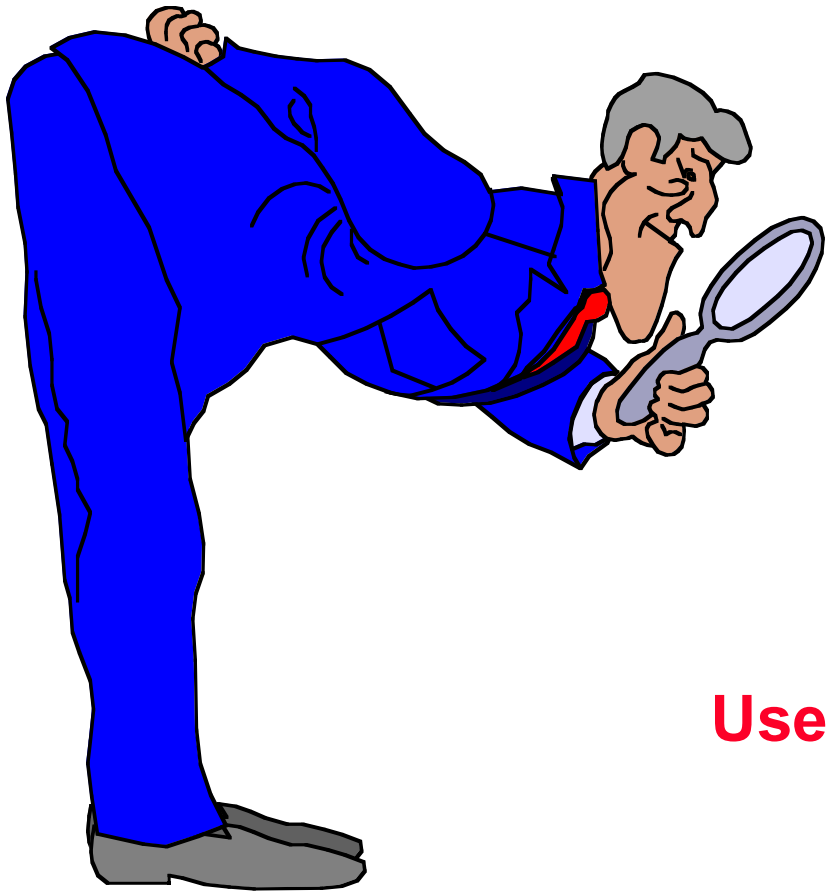
Invent a new field & stick to it?

* No! Do Real Stuff : solve problem that someone cares about
* No! Use separate, short projects
  - Always takes longer than expected
  - Matches student lifetimes
  - Long effort in fast changing field???
  - Learning: Number of projects vs. calendar time
  - If going to fail, better to know soon
* Strive for multi-disciplinary, multiple investigator projects
  - 1 expert/area is ideal (no arguments)
* Match the strengths and weaknesses of local environment
* Make sure you are excited enough to work on it for 3-5 years
  - Prototypes help

My first project

* Multiprocessor project with 3 hardware faculty (Xtree)
* 1977: Design our own instruction set, microprocessor, interconnection topology, routing, boards, systems, operating system
* Unblemished Experience:
  - none in VLSI
  - none in microprocessors
  - none in networking
  - none in operating systems
* Unblemished Resources:
  - No staff
  - No dedicated computer (used department PDP-11/70)
  - No CAD tools
  - No applications
  - No funding
* Results: 2 journal papers, 12 conference papers, 20 TRs
* Impact?

2) Picking a solution



Let Complexity Be Your Guide?

* No! Keep things simple unless a very good reason not to
  - Pick innovation points carefully, and be compatible everywhere else
  - Best results are obvious in retrospect "Anyone could have thought of that"
* Complexity cost is in longer design, construction, test, and debug
  - Fast changing field + delays => less impressive results

Use the Computer Scientific Method?

* No! Run experiments to discover real problems
* Use intuition to ask questions, not answer them

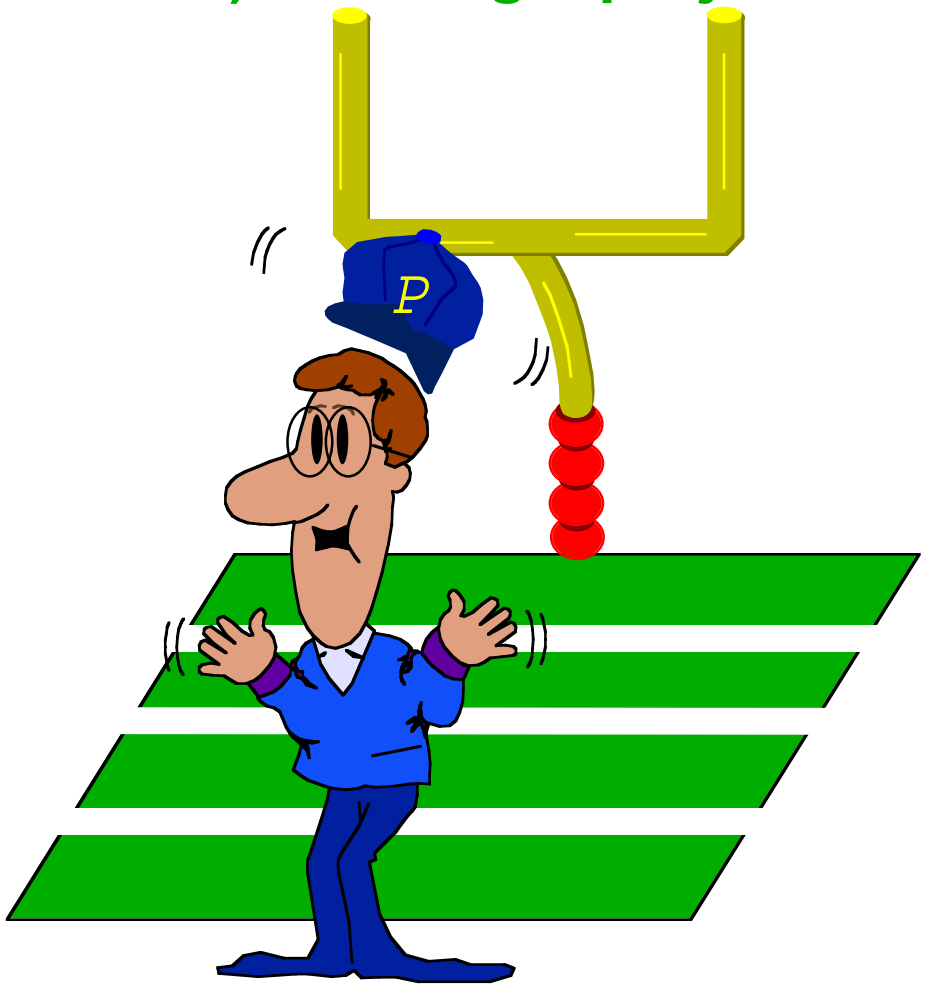
(And Pick A Good Name!)

Reduced
Instruction
Set
Computers

Redundant
Array of
Inexpensive
Disks

Recovery
Oriented
Computing

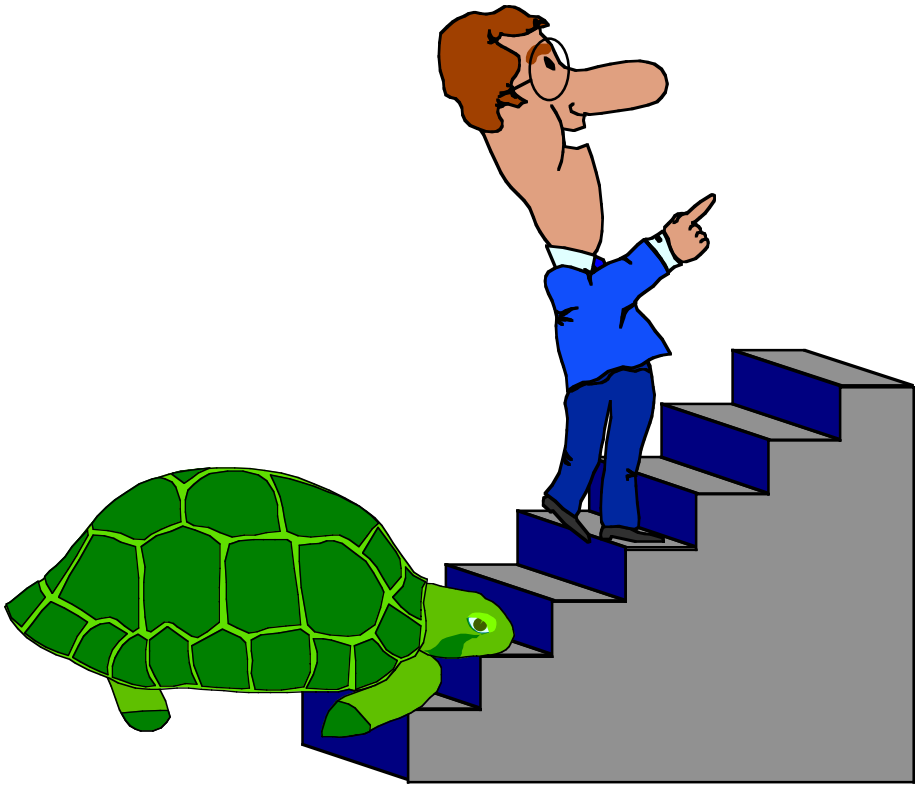
3) Running a project



Avoid Feedback?

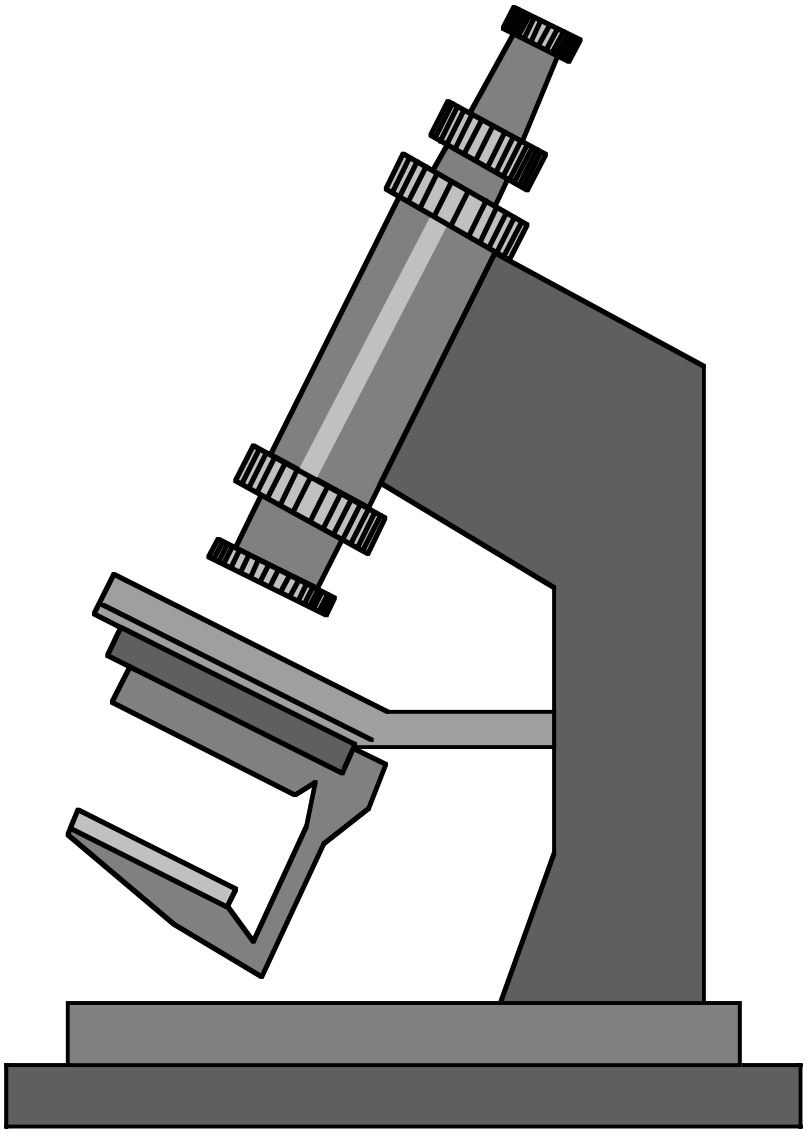
* No! Periodic Project Reviews with Outsiders
  - Twice a year: 3-day retreat
  - faculty, students, staff + guests
  - Key piece is feedback at end
  - Helps create deadlines
  - Give students chance to give many talks, interact with others industry
* Consider mid-course correction
  - Fast changing field & 3-5 year projects => assumptions changed
* Pick size and members of team carefully
  - Tough personalities are hard for everyone
  - Again, 1 faculty per area reduces chance of disagreement

4) Finishing a project



* ***People count projects you finish, not the ones you start***
* ***Successful projects*** **go through an unglamorous, hard phase**
* **Design is more fun than making it work**
  - **"No winners on a losing team; no losers on a winning team."**
  - **"You can quickly tell whether or not the authors have ever built something and made it work."**
* **Reduce the project if its late**
  - **"Adding people to a late project makes it later."**
* **Finishing a project is how people acquire taste in selecting good problems, finding simple solutions**

5) Evaluating Quantitatively



Never be Proven Wrong?

* No! If you can t be proven wrong, then you can t prove you re right
* Report in sufficient detail for others to reproduce results
  - can't convince others if they can't get same results
* For better or for worse, benchmarks shape a field
* Good ones accelerate progress
  - good target for development
* Bad benchmarks hurt progress
  - help real users v. help sales?

6) Transferring

Publishing Journal Papers IS Technology Transfer?

* No! Missionary work: Sermons first, then they read papers
  - Selecting problem is key: "Real stuff"
    - » Ideally, more interest as time passes
    - » Change minds with believable results
    - » Prima Donnas interfere with transfer
* My experience: industry is reluctant to embrace change
  - Howard Aiken, circa 1950:

    "The problem in this business isn't to keep people from stealing your ideas; its making them steal your ideas!"
  - Need 1 bold company (often not no. 1) to take chance *and* be successful
    - » RISC with Sun, RAID with (Compaq, EMC, …)
  - Then rest of industry must follow

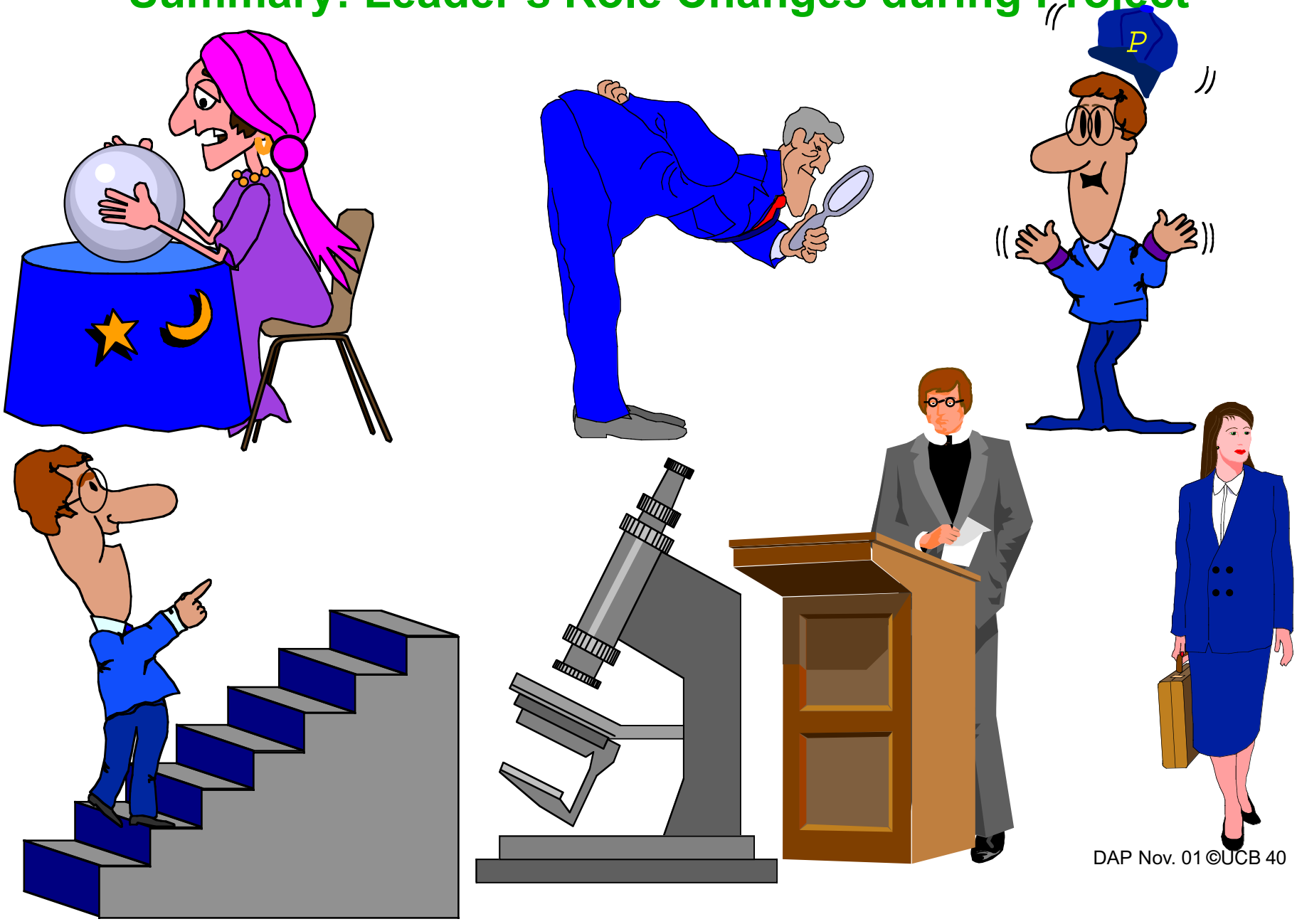
6) Transferring Technology

* **Pros**
  - **Personal satisfaction: seeing your product used by others**
  - **Personal $$$ (potentially)**
  - **Fame**
* **Cons**
  - **Learn about business plans, sales vs. marketing, financing, personnel benefits, hiring, …**
  - **Spend time doing above vs. research/development**
  - **Only 10% of startups really make it**
  - **Fame if company unsuccessful too (e.g., dot com backlash)**

Richard Hamming s Advice: You and Your Research (Latter in your Research Career)

* Doing Nobel Quality Research
  - Search Google for transcript of 1986 talk at Bell Labs
* Luck? Luck favors the prepared mind. Pasteur
* Important Problems: Great Thoughts Time Friday afternoons
* Courage: think about important unsolved problems
  - Big results usually to problems not recognized as such and people usually did not get encouragement
* Working conditions: can use creatively to lead to original solutions
  - Bell labs didn't have acres of programmers
* Drive: what distinguishes the great scientists
  - Not brains; commitment vs. dabbling; compound interest over time
* Open doors (vs. closed offices): short term vs. long term benefit
* Selling the work: not only published, but people must read it
  - as much work spent on polish and presentation as on the work itself
* Age: After 1st big success, hard to work on small problems
  - So change field at least every 10 years
* Educate your boss, Stimulation, right amount of Library work

Summary: Leader s Role Changes during Project



Acknowledgments

* Many of these ideas were borrowed from (inspired by?) Tom Anderson, David Culler, Al Davis, John Hennessy, Steve Johnson, John Ousterhout, Randy Katz, Bob Sproull, Carlo Sequin, Bill Tetzlaff and many others

Conclusion: Alternatives to a Bad Career

* **Goal is to have impact:**
  Change way people do Computer Science & Engineering
  - **Many 3 - 5 year projects gives more chances for impact**
* **Feedback is key: seek out & value critics**
* **Do Real Stuff : make sure you are solving some problem that someone cares about**
* **Taste is critical in selecting research problems, solutions, experiments, & communicating results; acquired by feedback and completing projects**
* **Faculty real legacy is people, not paper:**
  create environments that develop professionals of whom you are proud
* ***Students*** **are the coin of the academic realm**

Backup Slides to Help Answer Questions

Applying the Computer Scientific Method to OS

* Create private, highly tuned version for testing
  - take out all special checks: who cares about crashes during benchmarks?
* Never give out code of private version
  - might be embarrassing, no one expects it
* Run experiments repeatedly, discarding runs that don t confirm the generic OS hypothesis
  - Corollary