Software Experiences Relevant to the SKA

Duncan Hall

CALIM09

2009 April 2

Outline

A recap: 2008 November 11
Estimating the cost of software
Can large software really be that hard?
Why is large software so hard?
We can't "wish" the hardness away
But wait – there's more
So what should we do?
The role of architecture
Summary

A recap: 2008 November 11

Three industry case studies and one survey:

- **1: Speed to market**

  Searching for the keys under the lamp post

- **2: The costs of COTS**

  The devil is in the detail

- **3: Many cooks in the kitchen**

  Communication is the key to addressing inherent conflicts

- **4: Learnings – from CALIM08**

http://www.skatelescope.org/pages/WP2_Meeting_10Nov08/Day2-3_Hall.pdf

CALIM08: Software Development Survey

- Was there a formal software process?
- Were architectural definition documents used?
- How did the change control board function?
- What was the review process?
- What were the team dynamics?
- How best to communicate across the team?
- Time and cost against estimated budget?
- What could have been done better?
- Suggestions for SKA software development?

Summary of responses in backup slides

Sound familiar?

- Over-commitment
  - Frequent difficulty in making commitments that staff can meet with an orderly engineering process
- Often resulting in a series of crises
  - During crises projects typically abandon planned procedures and revert to coding and testing
- In spite of ad hoc or chaotic processes, can develop products that work
  - However typically cost and time budgets are not met
- Success depends on individual competencies and/or “death march” heroics
  - Can’t be repeated unless the same individuals work on the next project
- Capability is a characteristic of individuals, not the organisation

Mark Paulk et alia: The Capability Maturity Model for Software; in “Software Engineering” M. Dorfman and R. H. Thayer, 1997

What are the SEI CMMI “Maturity Levels”?

Level	Process Characteristics
1	Process is informal and ad hoc
2	Project management and project oversight practices are institutionalised
3	Organisational processes, including technical and project management, are clearly defined and repeatable
4	Processes are stabilised and aligned to goals, and product and process are quantitatively controlled
5	Process improvement is consistently and rigorously practised at organisation and project levels

Estimating the cost of software: a matrix formulation

An ill-conditioned non-linear problem:

$$Effort\ (€) = \prod_{\substack{\forall P \ \forall S \\ P,S>1}} \left\{ \left[\text{Problem space} \right] \otimes \left[\text{Solution space} \right] \right\}$$

where

$$\left[\text{Problem space} \right] = \underbrace{\begin{bmatrix} Size\ (+) \\ Complexity\ (+) \\ Time\ (-) \\ Interfaces\ (+) \\ Reliability\ (+) \\ \vdots \end{bmatrix}}_{\text{Requirements}} \times \left[e^{\underbrace{\frac{d}{dt}\left[\text{Problem space} \right]}_{\text{Change requests}}} \right]$$

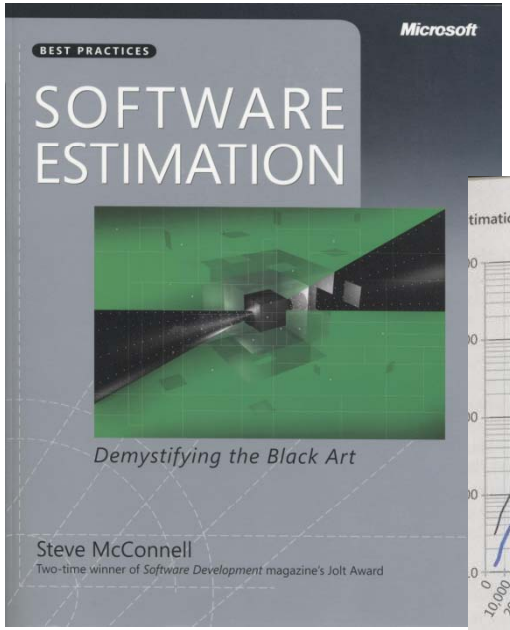
$$\left[\text{Solution space} \right] = \begin{bmatrix} People\ skills\ (-) \\ Colocation\ (-) \\ Processes\ (-?) \\ Prior\ experience\ (-) \\ Reuse\ opportunities\ (-) \\ Toolsets\ (-?) \\ \vdots \end{bmatrix}$$

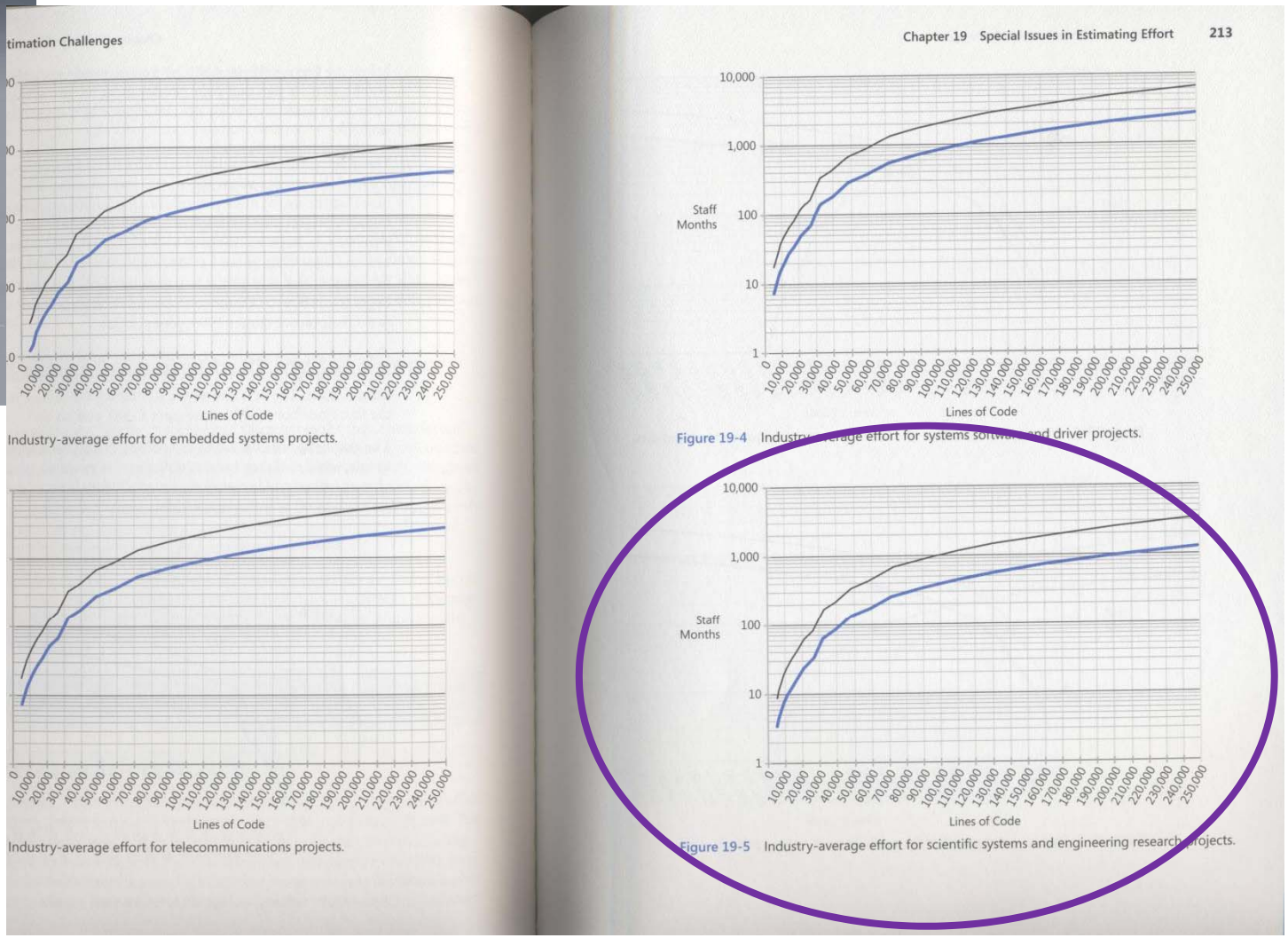
Can large software really be that hard?

Yes!

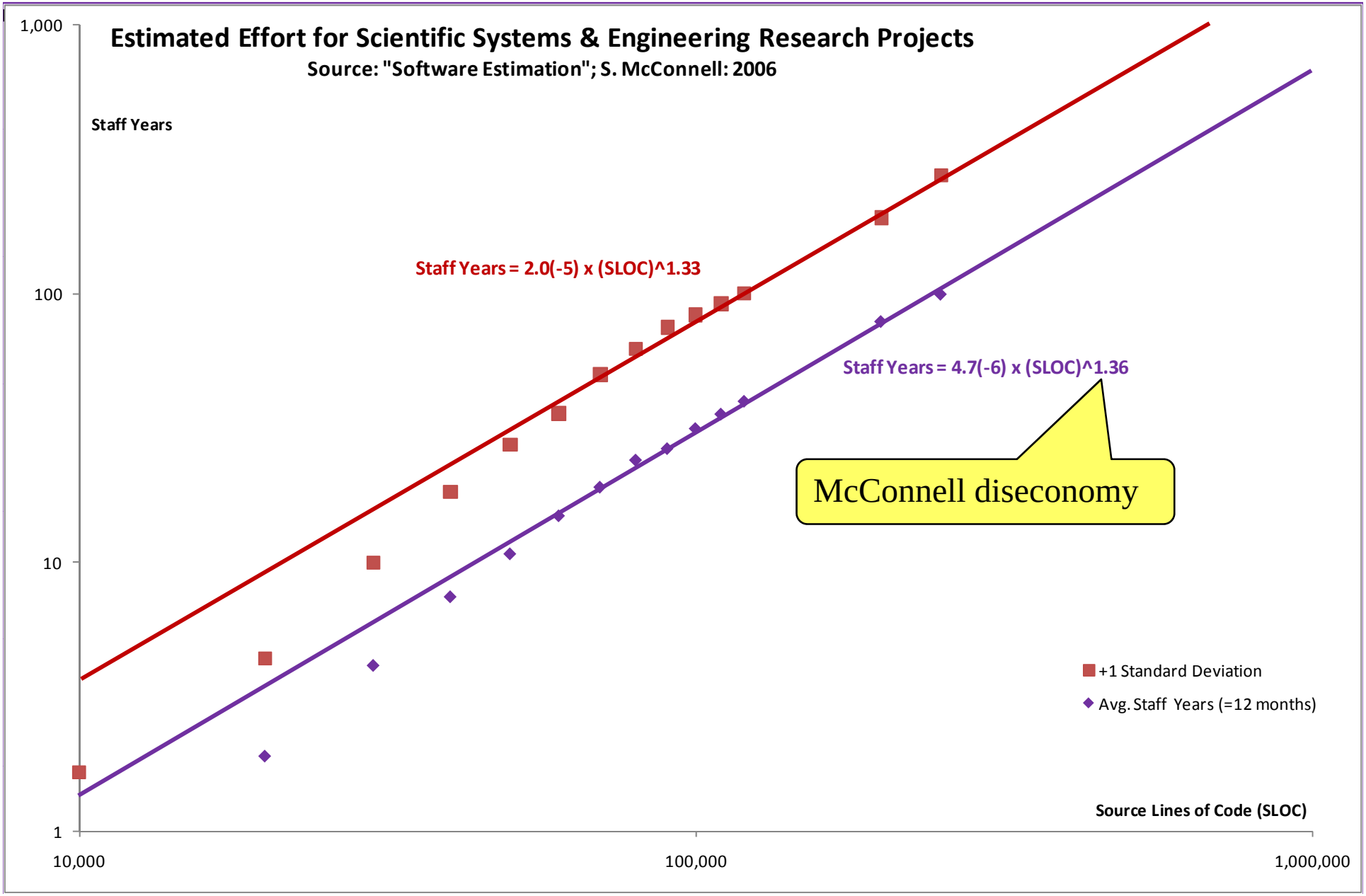
- First-order model for estimating effort
- Diseconomies of scale
- Confirmation from the literature

First-order model for estimating effort

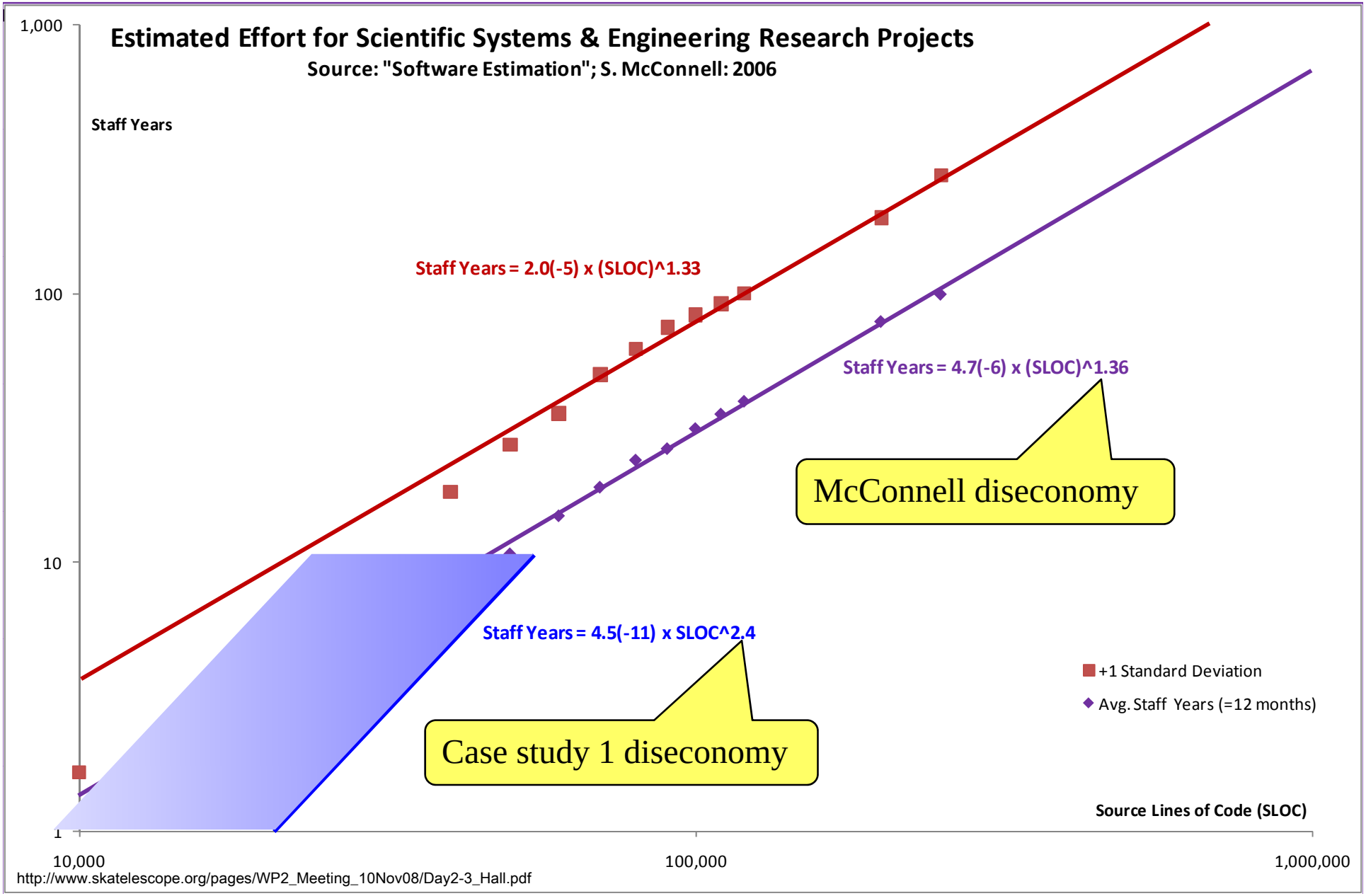




McConnell's data on log-log axes



2008Nov11 Case study c.f. McConnell data



http://www.skatelescope.org/pages/WP2_Meeting_10Nov08/Day2-3_Hall.pdf

How big are the “legacy” codes?

Table I. Sample community codes for radio astronomy imaging.

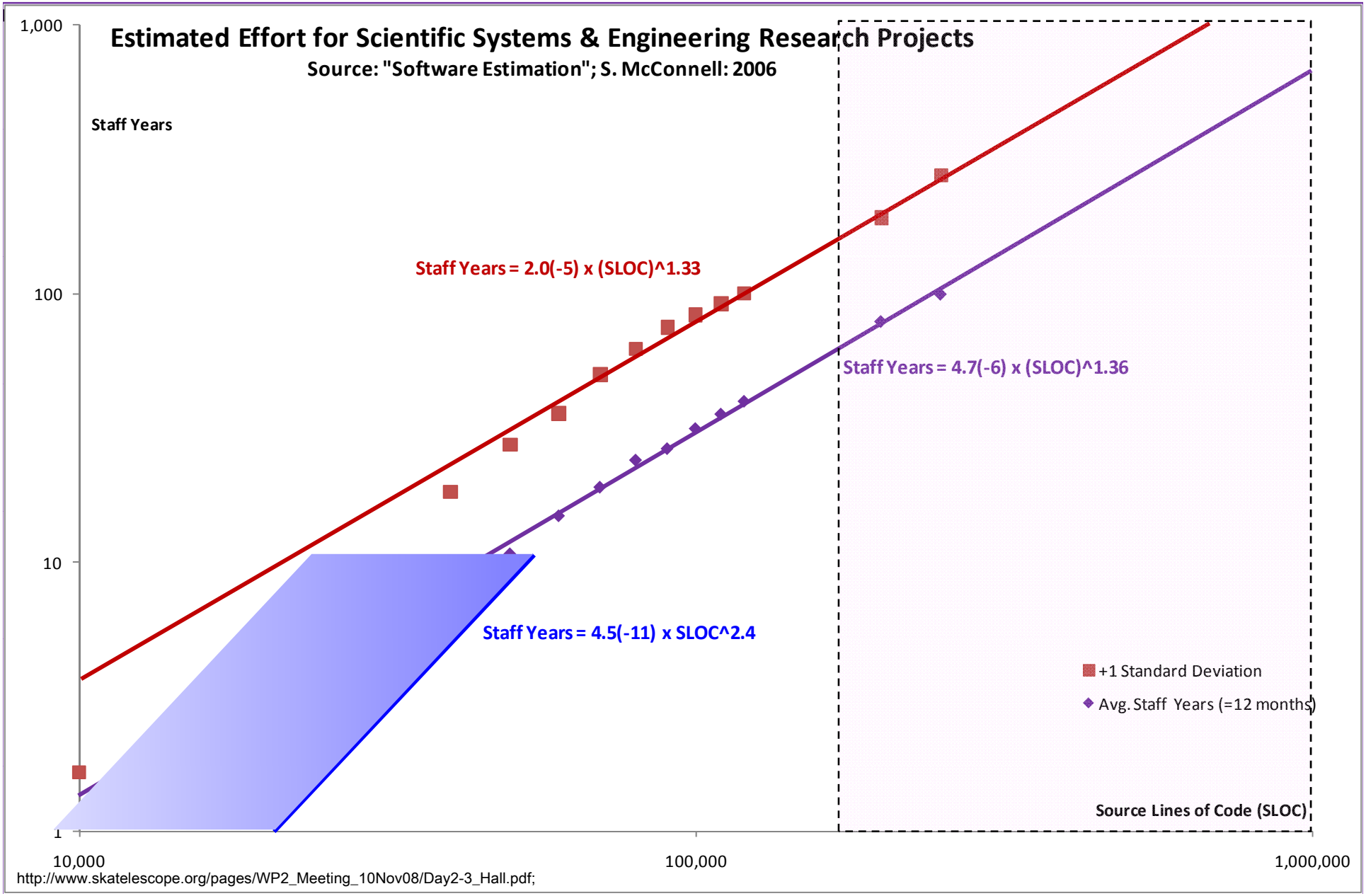
Package name	Development languages (ordered by prevalence)	Size (MSLOC[a])
Astronomical Image Processing System (AIPS)[b]	Fortran 77, C	0.6
Multi-channel Image Reconstruction, Image Analysis, and Display (MIRIAD)[c]	Fortran 77, C	0.2
Astronomical Information Processing System (AIPS++)[d]	C++, Glish [14], Fortran 77	1.0

[a]MSLOC $= 10^6$ SLOC, as measured by SLOCCount (written by David A. Wheeler).
[b]Modified version of base release 15OCT97.
[c]Release v4.
[d]Modified version of code base v1.8 #667.

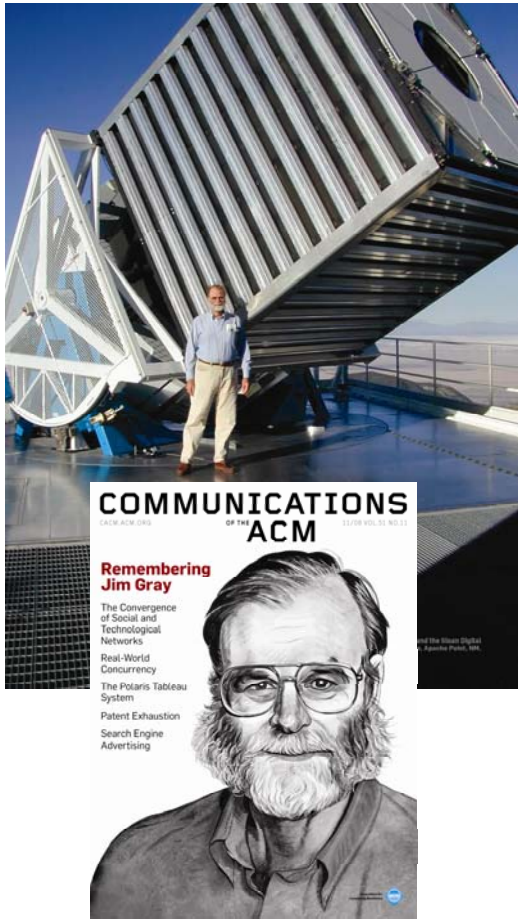
MSLOC:	
Debian 4.0	283
Mac OS X 10.4	86
Vista	50
Linux kernel 2.6.29	11
OpenSolaris	10

Kemball, R. M. Crutcher, R. Hasan: “A component-based framework for radio-astronomical imaging software systems” Software Practice and Experience: 2008; 38: pp 493–507
http://en.wikipedia.org/wiki/Source_lines_of_code

Legacy: ~20 to ~700+ staff years effort?



SDSS: Gray and Szalay



Where the Rubber Meets the Sky: Bridging the Gap between Databases and Science
MSR-TR-2004-110: 2004 October

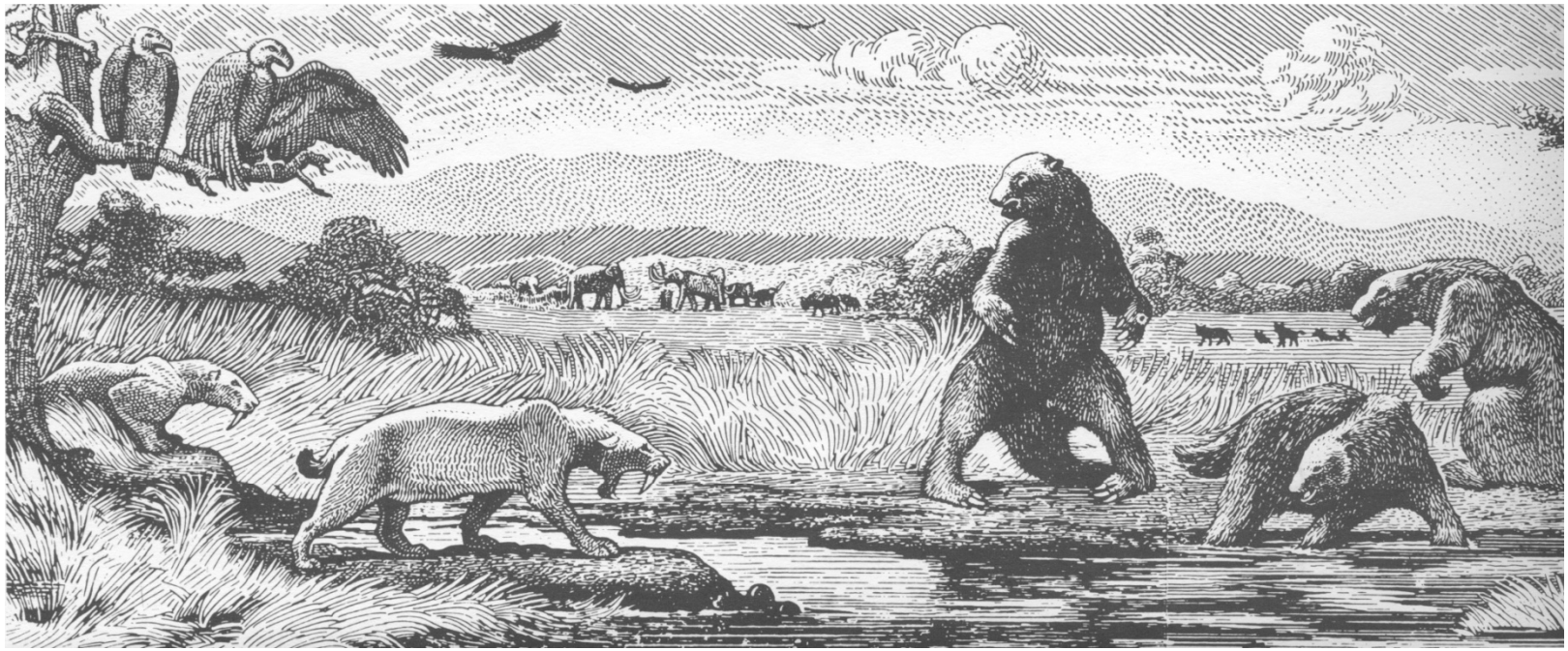
- One problem the large science experiments face is that software is an out-of-control expense
- They budget 25% or so for software and end up paying a lot more
- The extra software costs are often hidden in other parts of the project – the instrument control system software may be hidden in the instrument budget

Why is large software so hard?

Three answers:

- The Tar Pit
- Correlation metaphor
- Human frailty

Brooks' "Tar Pit" metaphor:

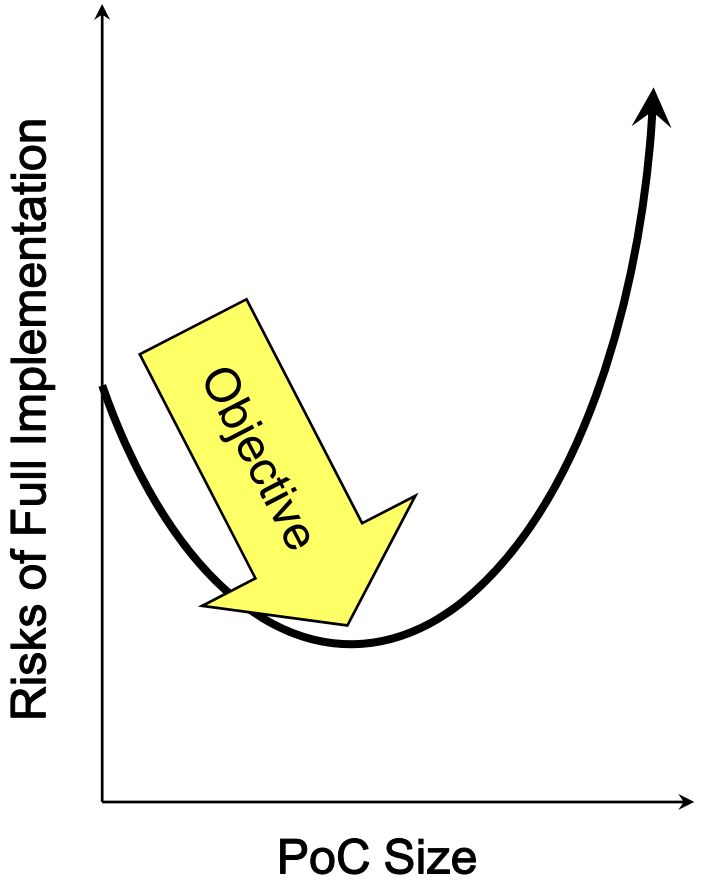


- "Large-system programming has over the past decade been such a tar pit, and many great and powerful beasts have thrashed violently in it
- Most have emerged with running systems – few have met goals, schedules, and budgets
- Large and small, massive or wiry, team after team has become entangled in the tar ...
- Everyone seems to have been surprised by the stickiness of the problem, and it is hard to discern the nature of it
- But we must try to understand it if we are to solve it"

Fred Brooks; "The Mythical Man-Month – Essays on Software Engineering Anniversary Edition": 1995

Agile is good for small “Proofs of Concept”

- **Focus on areas of uncertainty:**
  We want to learn something from the PoC
- **Clearly define what we want to learn about:**
  State scope, objectives, issues, assumptions, risks and deliverables
- **Set the PoC size to minimise risks of total project**



Sizing a POC is a balance between “learning” and “doing” to minimise the risks of full implementation

Why not use “Agile” all the time?

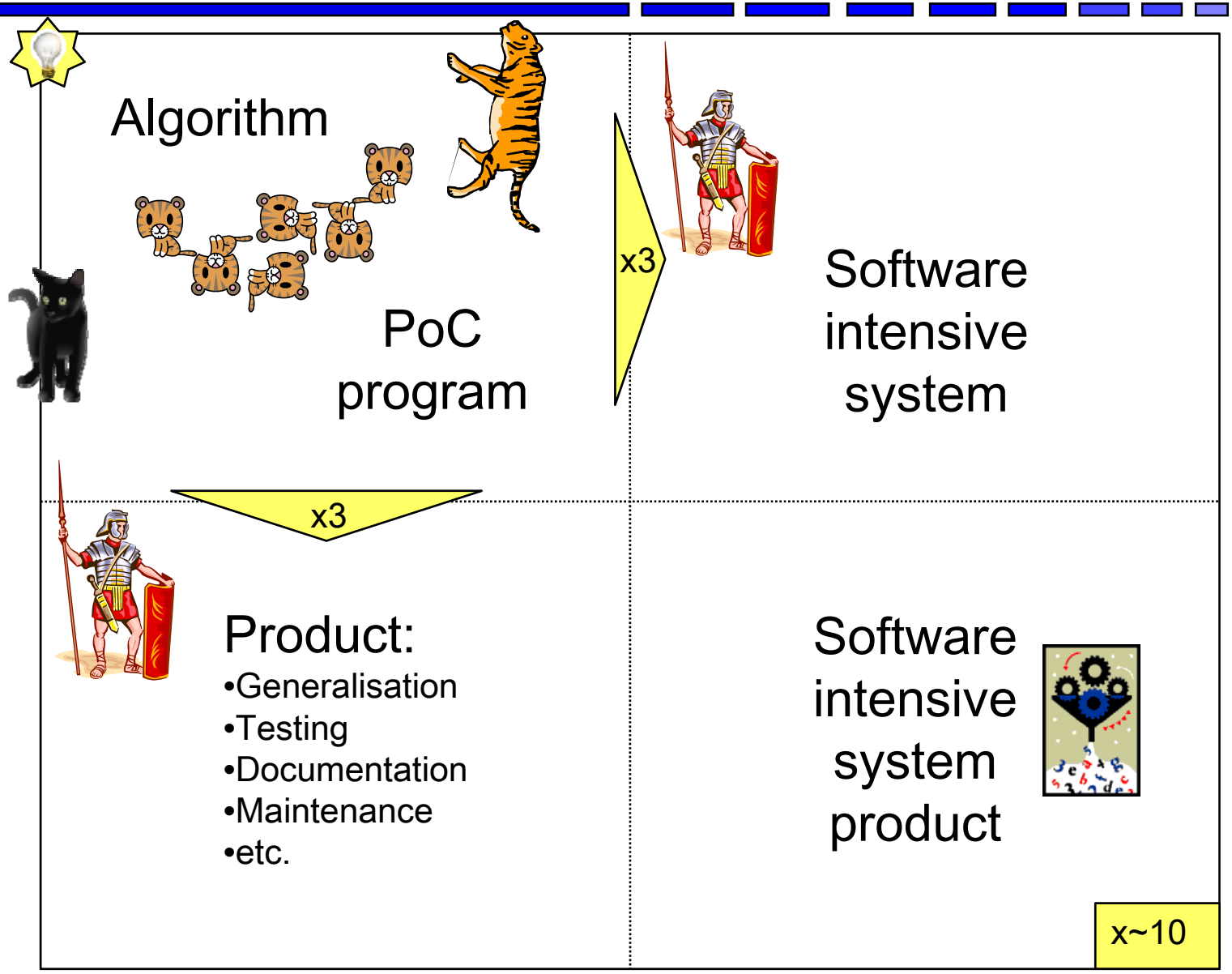
- “One occasionally reads newspaper accounts of how two programmers in a remodelled garage have built an important program that surpasses the best efforts of large teams

- And every programmer is prepared to believe such tales, for he knows that he could build any program much faster that the 1,000 statements/year reported for industrial teams

- Why then have not all industrial programming teams been replaced by dedicated garage duos?”

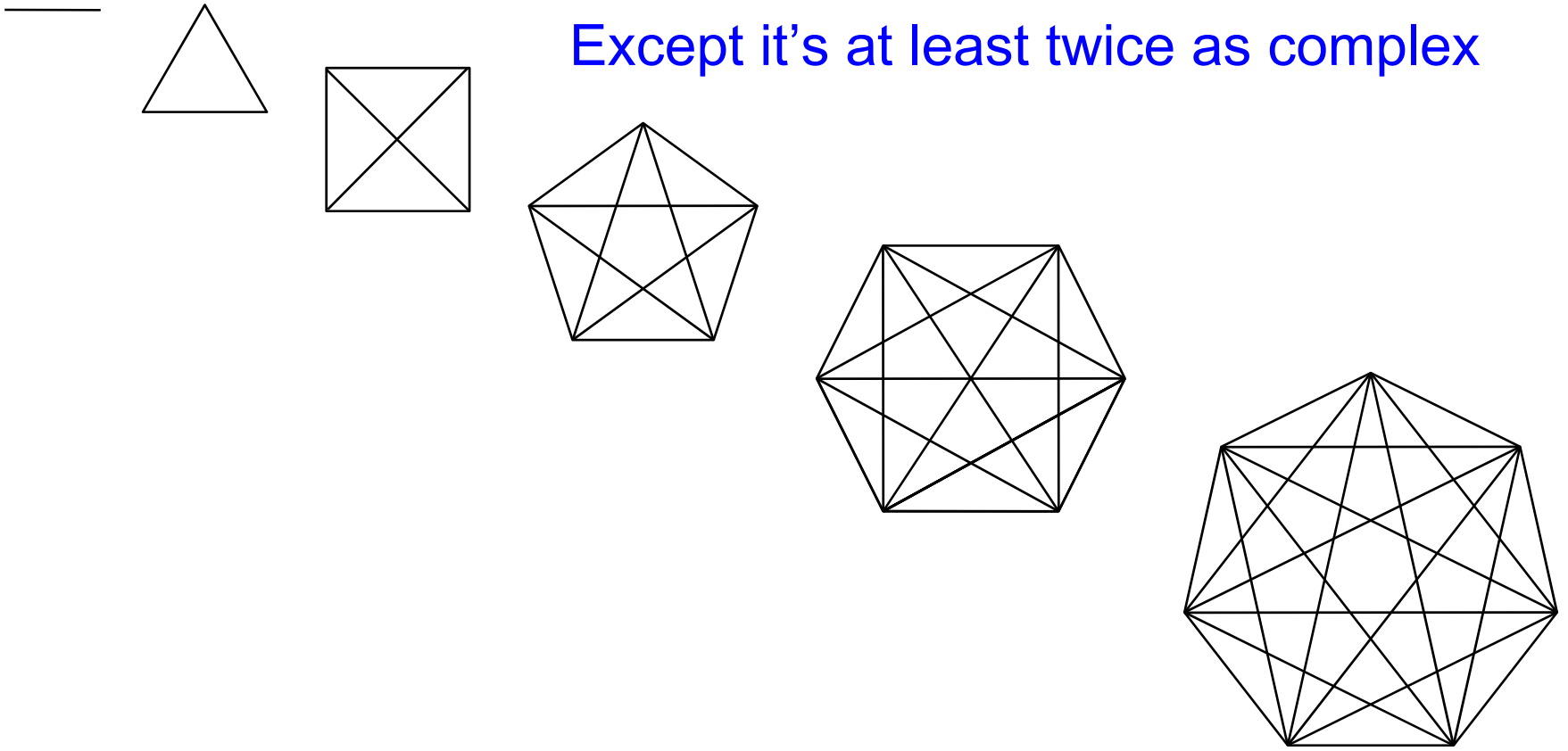
Fred Brooks; “The Mythical Man-Month – Essays on Software Engineering Anniversary Edition”: 1995

A software intensive system product is much more than the initial algorithm:



Fred Brooks; “The Mythical Man-Month – Essays on Software Engineering Anniversary Edition”: 1995

Collaboration is just like correlation:



Except it's at least twice as complex

- [Without modularisation] Must establish, coordinate and regularly use and maintain ~n^2 links
- So worst-case diseconomy of scale likely to have slope >2 on log-log effort-size charts

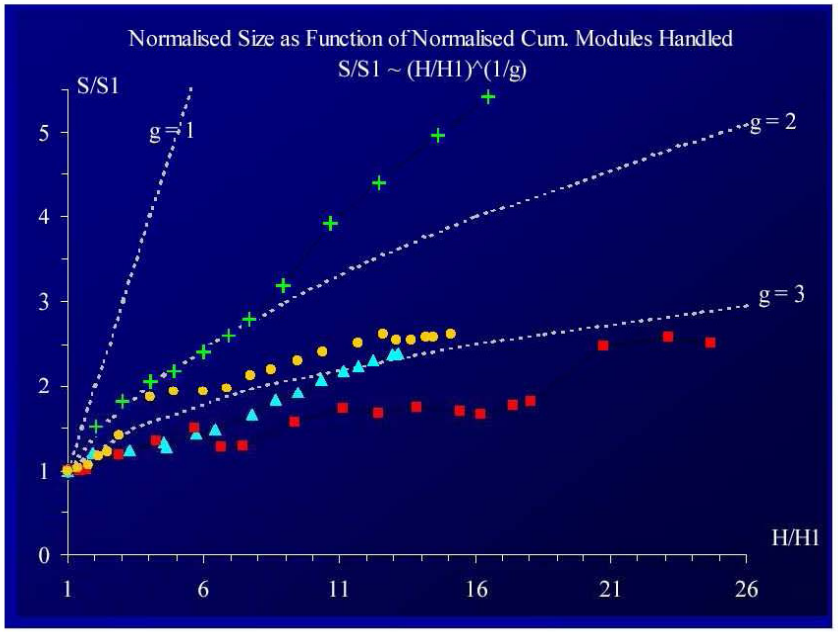
IBM's Bruce Elmegreen concurs:

SIZE

real time C (1980's +20 yrs)

operating system S3 (1970's +15yrs)

cobol 1980's +14yrs)



EFFORT

$EFFORT \sim SIZE^3$

$COMPLEXITY = EFFORT/SIZE \sim SIZE^2$

Bigger systems accommodate bigger and better software packages, but the effort to make them increases much faster than the system size.

→ relative expense of software should increase

Lehman and Ramil 2001

Human frailty:

- We must work together to complete large projects in reasonable time, and have other people try to catch our mistakes
- Once we start working together, we face other problems
- The natural language we use to communicate is wonderfully expressive, but frequently ambiguous
- Our human memory is good, but not quite deep and precise enough to remember a project's myriad details
- We are unable to track what everyone is doing in a large group, and so risk duplicating or clobbering the work of others
- Large systems can often be realised in multiple ways, hence engineers must converge on a single architecture and design

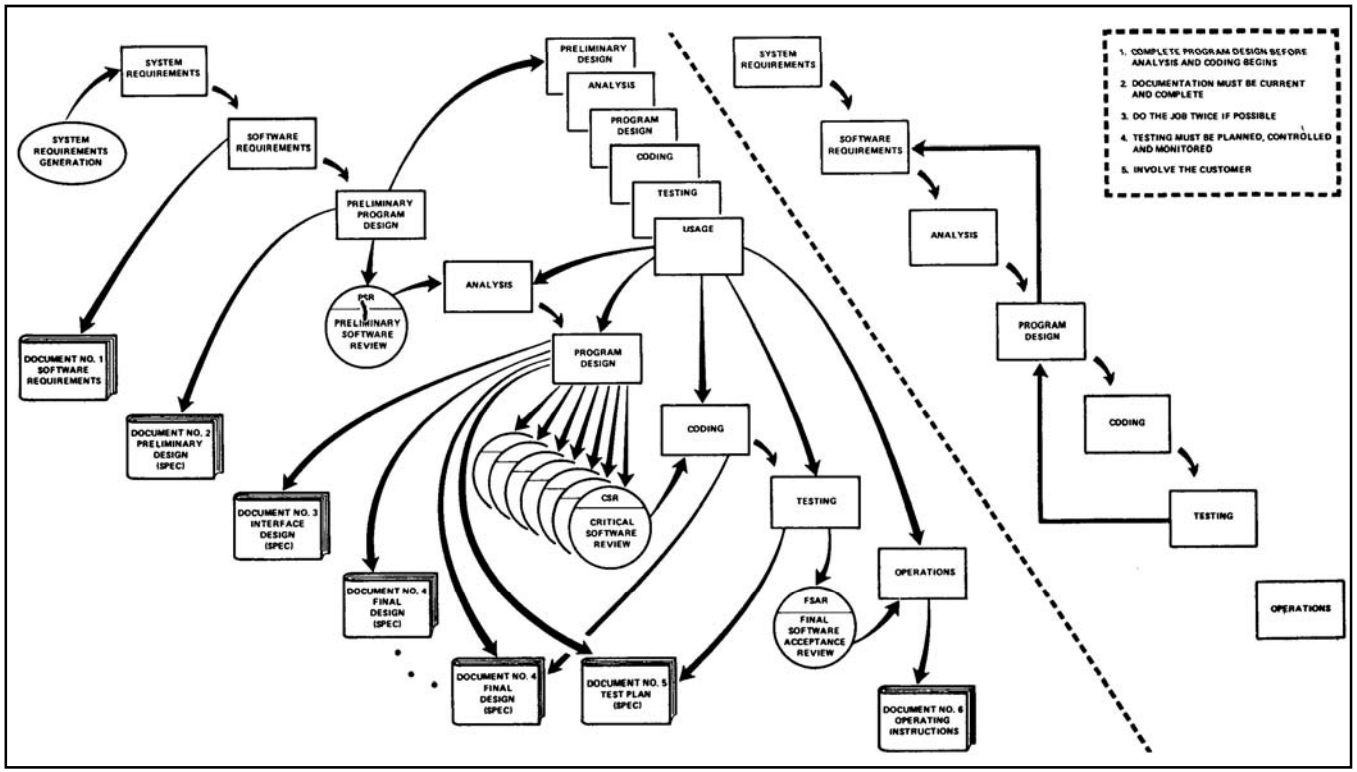
We can’t “wish” the hardness away

And why not? Three answers:

- It's been around a long time
- There are no silver bullets
- Bad things happen if we rely solely on wishes and prayers
  - But of course, any assistance may be of help

Myth busting – number 1:

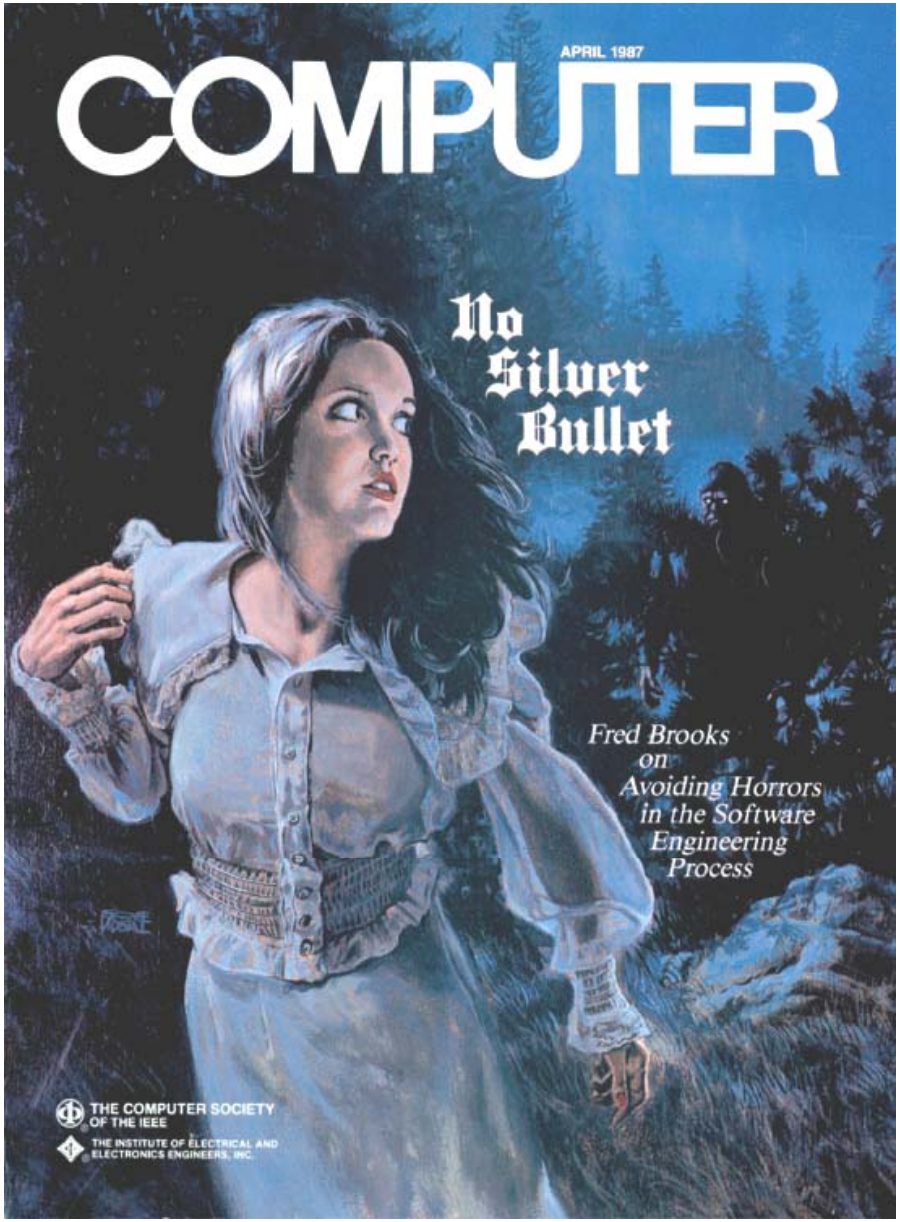
- **The myth**
  - The old guys used “waterfall” - also expressed as “traditional software engineering”
  - We are a lot better now
- **The reality?**
  - They were giants in the old days: Parnas, Jackson, Brooks ...
  - As documented in 1970 by Royce, aspects of the mis-named “waterfall” are very similar to today’s “agile”



IEEE WESCON, Aug 1970, pp 1-9

Myth busting – number 2:

- The myth
  - Modern approach ‘X’ will slay the Werewolf of intractable software development
- The reality?
  - There is no silver bullet: Brooks’ “essential” hardness is ever-present
  - Various brass and lead bullets do reasonable jobs to address “accidental” hardness – but each has its own risks and required overhead

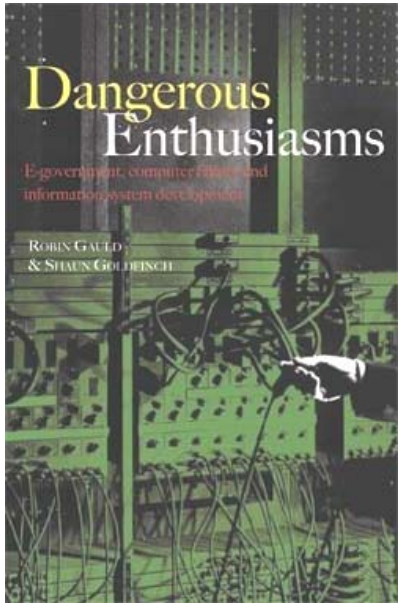


Myth busting – number 3:

- The myth
  - Just get good coders – and let them have at it
- The reality?
  - Yes, it is “not impossible” that locking coders in a room with a gap under the door will eventually result in on-time in-budget delivery that meets all expectations
  - However:
    - The attendant risks are high
    - Software has become central to large science projects
    - Software projects in the public domain can be subject to embarrassing scrutiny

Public domain – four pathological enthusiasms:

1. Idolisation – technological infatuation
2. Technophilia – the “myth of the technological fix”
3. Lomanism – enthusiasm induced by overzealous sales tactics, as epitomised by Willie Loman in Arthur Miller’s Death of a Salesman
4. Faddism – the tendency to link software development to the latest fad, such as “XP” or “XML” or management theory X, or Y, or Z



But wait – there's more:
Another technology wave is gaining momentum

Increasing CPU clock speed can reduce “efficiency”

Power Cost of Frequency

- Power ∝ Voltage^2 x Frequency (V^{2}F)
- Frequency ∝ Voltage
- Power ∝ Frequency^3

	Cores	V	Freq	Perf	Power	PE (Bops/watt)
Superscalar	1	1	1	1	1	1
“New” Superscalar	1X	1.5X	1.5X	1.5X	3.3X	0.45X

Jack J. Dongarra: “An Overview of High Performance Computing and Challenges for the Future” SIAM 2008 Annual Meeting, http://live.blueskybroadcast.com/bsb/client/CL_DEFAULT.asp?Client=975312&PCAT=900&CAT=901

Efficiency is driving the move to multicore

Power Cost of Frequency

- Power ∝ Voltage^2 x Frequency (V^{2}F)
- Frequency ∝ Voltage
- Power ∝ Frequency^3

	Cores	V	Freq	Perf	Power	PE (Bops/watt)
Superscalar	1	1	1	1	1	1
"New" Superscalar	1X	1.5X	1.5X	1.5X	3.3X	0.45X
Multicore	2X	0.75X	0.75X	1.5X	0.8X	1.88X

(Bigger # is better)

50% more performance with 20% less power

Preferable to use multiple slower devices, than one superfast device

Jack J. Dongarra: "An Overview of High Performance Computing and Challenges for the Future" SIAM 2008 Annual Meeting, http://live.blueskybroadcast.com/bsb/client/CL_DEFAULT.asp?Client=975312&PCAT=900&CAT=901

Multicore represents a disruptive challenge

Major Changes to Software

- Must rethink the design of our software
  - Another disruptive technology
    - Similar to what happened with cluster computing and message passing
  - Rethink and rewrite the applications, algorithms, and software
- Numerical libraries for example will change
  - For example, both LAPACK and ScaLAPACK will undergo major changes to accommodate this

Jack J. Dongarra: “An Overview of High Performance Computing and Challenges for the Future” SIAM 2008 Annual Meeting, http://live.blueskybroadcast.com/bsb/client/CL_DEFAULT.asp?Client=975312&PCAT=900&CAT=901

Dongarra: focus must now be on software

Conclusions

- For the last decade or more, the research investment strategy has been overwhelmingly biased in favor of hardware.
- This strategy needs to be rebalanced - barriers to progress are increasingly on the software side.
- Moreover, the return on investment is more favorable to software.
  - Hardware has a half-life measured in years, while software has a half-life measured in decades.
- High Performance Ecosystem out of balance
  - Hardware, OS, Compilers, Software, Algorithms, Applications
    - No Moore's Law for software, algorithms and applications

Jack J. Dongarra: "An Overview of High Performance Computing and Challenges for the Future" SIAM 2008 Annual Meeting, http://live.blueskybroadcast.com/bsb/client/CL_DEFAULT.asp?Client=975312&PCAT=900&CAT=901

So what should we do?

Some possible answers:

- Don’t step into that tar pit! (?)
- Rely on heroes who can walk over tar pits?
- Create superstructures over the tar pit?
- Drag in other willing – or unwilling – participants?
- Learn from survivors of tar pit turmoil?

Jeff Kantor: LSST Data Management Project Manager

- Define – in terms of requirements, scope, plans, resources etc. – “Data Challenges” as prototype projects for the 25 contributing institutions to address
- These project provided calibration data for JK-developed effort and cost estimation using combination Use Case-Function Point-COCOMO II
- A single commercial software development method – ICONIX – with shared toolsets and repository of WIP
- SysML; Sparx EA tool for all requirements traceability
- COTS used for monitoring and control and user interface
- Software and computing (excluding m&c):
  - $110 million = 25% of LSST construction cost
  - Estimated to account for 40% of operating cost

Scot Kleinman: SDSS Operations Development coordinator[1]

- Lessons Learned" – just that: next time round this is what we'd do from the start
- Initially operations software was being developed by folk who thought it "had to be done like this" – where "this" was a personal view
- Initially often:
  - No formalism for requirements documentation
  - No traceability
  - Lack of uniform version control
  - No user testing, no model checking
  - Little interaction with end users - most software was delivered before users could check it
- Science requirements eventually led to insistence on version control

1: Lead author of "Lessons Learned from Sloan Digital Sky Survey Operations": http://arxiv.org/abs/0810.2811

Roger Jones: Chair ATLAS International Computing Board

- Regular e.g. weekly teleconferences coupled with face to face meetings thrice annually; used email archiving and commercial tools e.g. Rational Rose
- Governance focus: quarterly formal reports / reviews and bi-monthly formal reports; well defined expected deliverables and metrics
- Decomposed problem space into subcomponents that could be wholly “owned” and delivered by specific teams
- Stated the need to now treat software as a key “detector” in nuclear-physics-speak, i.e. an integral component of the experiment
- Costs of developing the software difficult to track
- Computing hardware (boxes and people) costs amounted to somewhere between 2/3 and 3/4 of total spend on other hardware – an unexpected and surprisingly high proportion
- Reasonably formal framework of UML modelling
- Software development over an 11+ year period

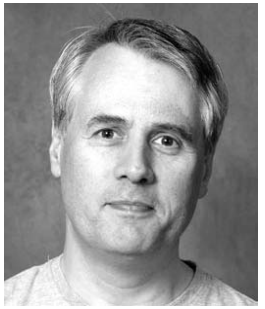
Wolfgang von Rueden: CERN Head of IT, SKA IEAC

- Experiment applications developed by end users – but must be certified to run on CERN infrastructure
- Extensive certification requirements required to be met prior to promotion from design → pre-production → production
- Multi-platform and open source ETICS “Einfrastructure for Testing, Integration and Configuration of Software” – typically 3 to 4 years for certification of an application
- ETICS provides software professionals with an “out-of-the-box” build and test system, powered with a build and test product repository
- Assuring quality is the principal aim of the ETICS system
- Developers should take into account Functionality, Reliability, Usability, Efficiency, Maintainability and Portability
- This is based on and highlighted in the ISO/IEC 9126 standard

http://etics.web.cern.ch/etics/quality.htm

Judith Segal (UK Open University)
Chris Morris (UK Science Technology Funding Council)[2]

- Industry developers typically have good ideas of what packages – e.g. human-resources or accounting – should do, and they can readily understand the systems' requirements
- In contrast, scientific software's purpose has often been to improve domain understanding – e.g. by running simulations or testing algorithms – so requirements have "emerged" as understanding of the domain progressed through the process of developing the software
- Despite the medium to largish size of some legacy scientific codes, they have typically been developed using methods like "code and fix"
- A lack of trust in "reverse engineering"



2: Co-editors of IEEE Software special issues on "developing scientific software" 2008 Jul-Aug and 2009 Jan-Feb

From the Aug 2008 IEEE Software editorial:

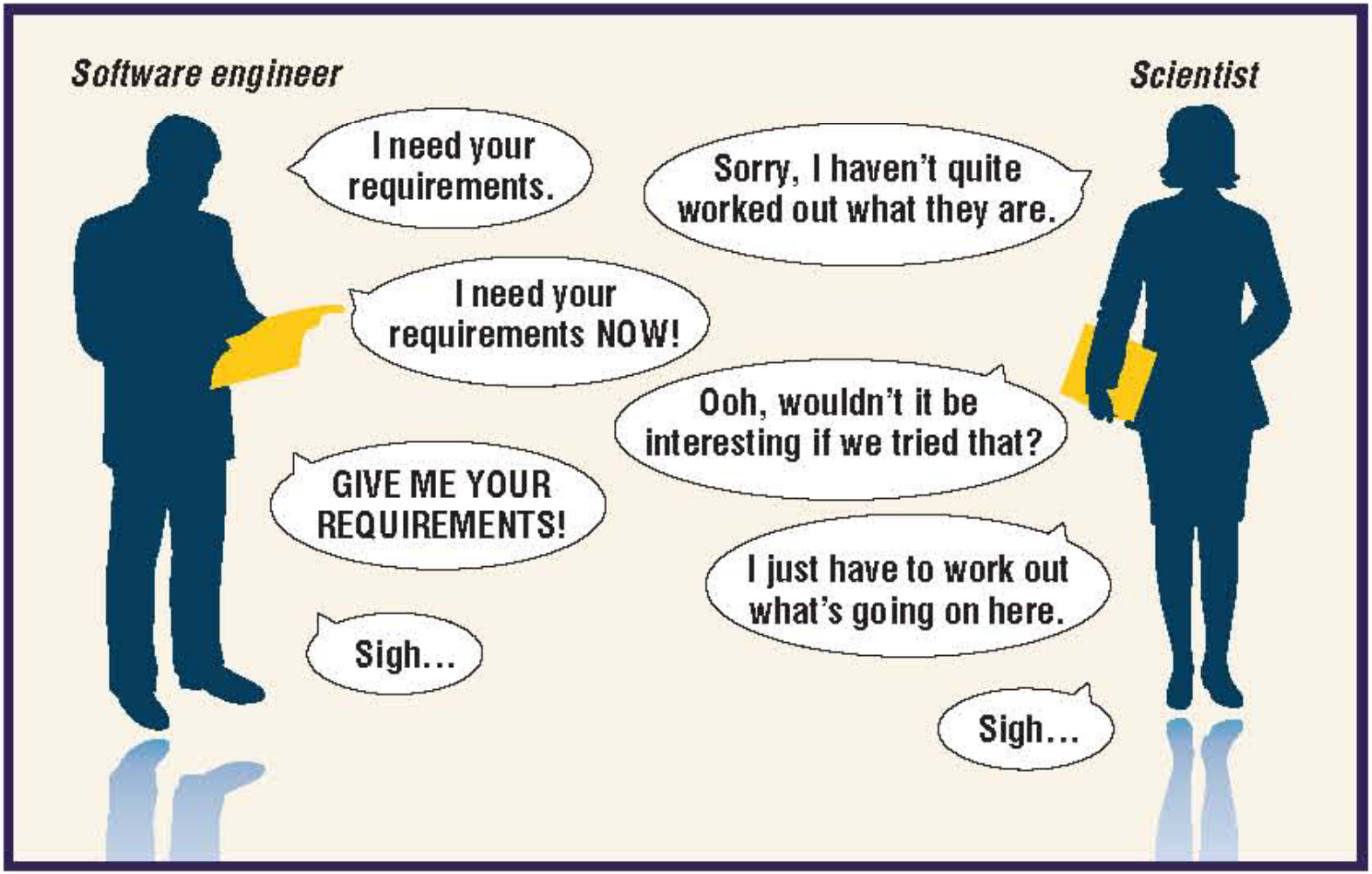


Figure 2. A clash between software engineers and scientists. The former expect requirements to be specified up front; the latter expect them mostly to emerge.

The role of architecture

The Winchester Mystery House:

- 38 years of construction: 147 builders, $5.5 million (in $ of 1884 – 1922)
- 160 rooms, 47 fireplaces, 40 bedrooms, 6 kitchens, 3 lifts, 2 basements, 950 doors
- 65 doors to blank walls, 13 staircases abandoned, 24 skylights in floors
- 0 architects, 0 architectural plans

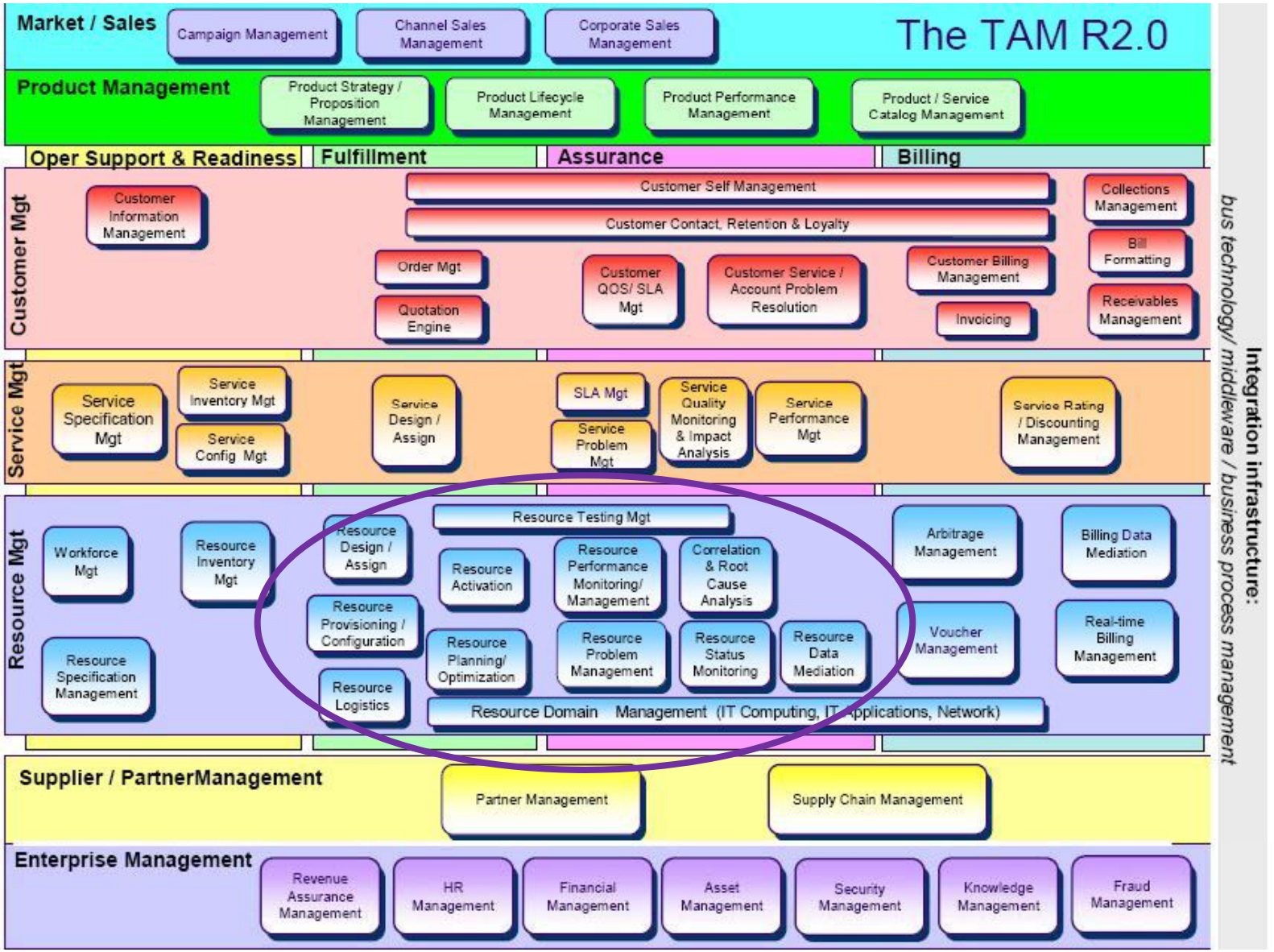
http://www.winchestermysteryhouse.com/facts.html

Dealing with complexity:

- Abstraction – multiple viewpoints
- Decomposition – aka modularisation
- Loose coupling; high cohesion
- Standards
- Integration

Reuse

Application architecture example: TAM



http://www.tmforum.org/ApplicationsFramework/2322/home.html

TAM Products and Services Directory

Enterprise Management

- Asset Management
- Financial Management
- Fraud Management
- HR Management
- Knowledge Management
- Revenue Assurance Management
- Security Management

Service Management

- Service Configuration Management
- Service Design/Assign
- Service Inventory Management
- Service Performance Management
- Service Problem Management
- Service Quality Monitoring & Impact Analysis
- Service Rating/Discounting Management
- Service Specification Mgt
- SLA Mgt

Customer Management

- Bill Formatting
- Collections Management
- Customer Billing Management
- Customer Contact, Retention & Loyalty
- Customer Information Management
- Customer QoS/SLA Management
- Customer Self Management
- Customer Service/Account Problem Resolution
- Invoicing
- Order Management
- Quotation Engine
- Receivables Management

Resource Management

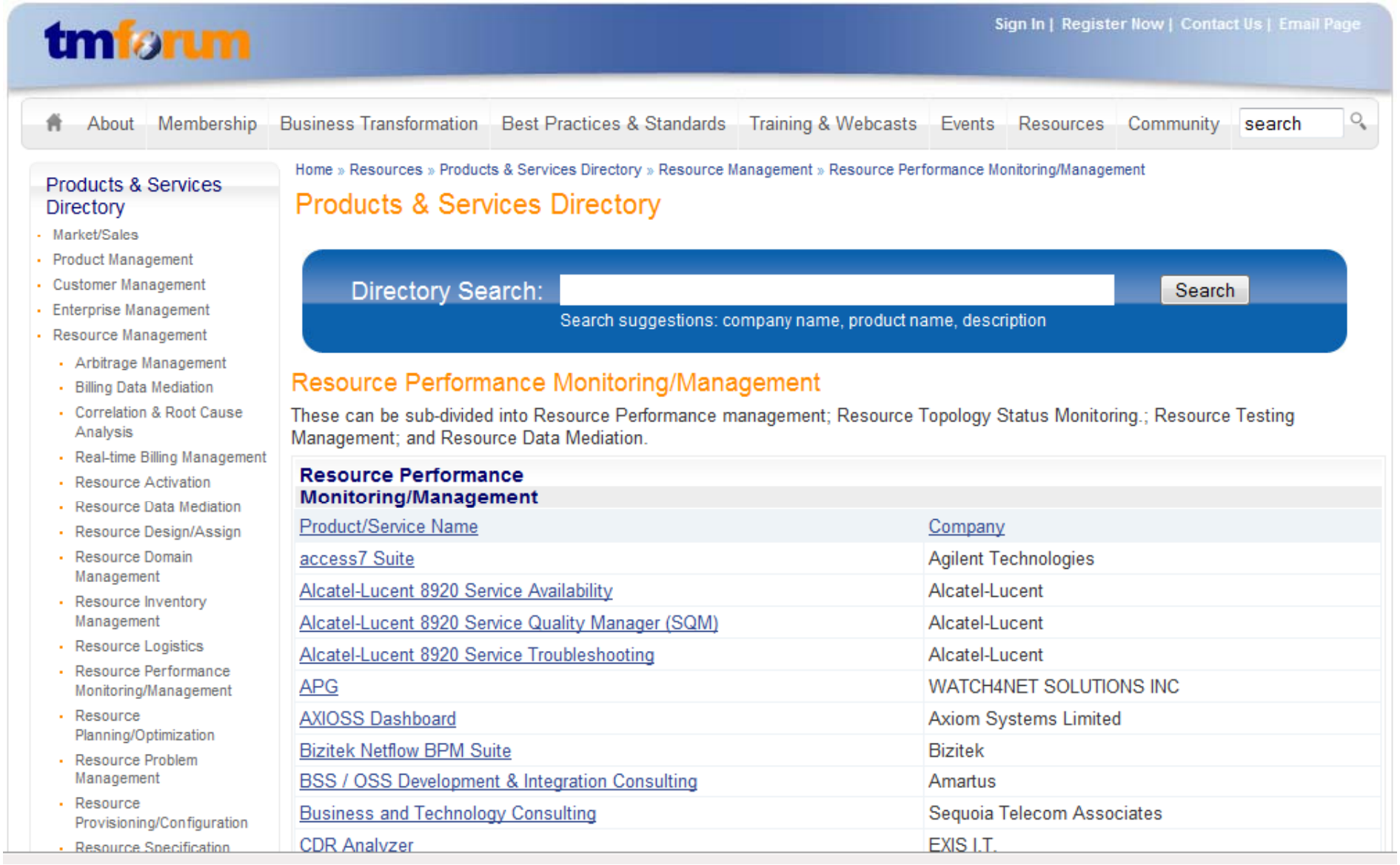
- Arbitrage Management
- Billing Data Mediation
- Correlation & Root Cause Analysis
- Real-time Billing Management
- Resource Activation
- Resource Data Mediation
- Resource Design/Assign
- Resource Domain Management
- Resource Inventory Management
- Resource Logistics
- Resource Performance Monitoring/Management
- Resource Planning/Optimization
- Resource Problem Management
- Resource Provisioning/Configuration
- Resource Specification Management
- Resource Status Monitoring
- Resource Testing Management
- Voucher Management
- Workforce Management

Supplier/Partner Manager

Services Directory

http://www.tmforum.org/ProductsServices/2528/home.html?catID=2528

Resource Performance Monitoring/Management



tmforum

Sign In | Register Now | Contact Us | Email Page

About | Membership | Business Transformation | Best Practices & Standards | Training & Webcasts | Events | Resources | Community | search

Home » Resources » Products & Services Directory » Resource Management » Resource Performance Monitoring/Management

Products & Services Directory

- Market/Sales
- Product Management
- Customer Management
- Enterprise Management
- Resource Management
  - Arbitrage Management
  - Billing Data Mediation
  - Correlation & Root Cause Analysis
  - Real-time Billing Management
  - Resource Activation
  - Resource Data Mediation
  - Resource Design/Assign
  - Resource Domain Management
  - Resource Inventory Management
  - Resource Logistics
  - Resource Performance Monitoring/Management
  - Resource Planning/Optimization
  - Resource Problem Management
  - Resource Provisioning/Configuration
  - Resource Specification

Products & Services Directory

Directory Search: Search

Search suggestions: company name, product name, description

Resource Performance Monitoring/Management

These can be sub-divided into Resource Performance management; Resource Topology Status Monitoring.; Resource Testing Management; and Resource Data Mediation.

Resource Performance Monitoring/Management

Product/Service Name	Company
access7 Suite	Agilent Technologies
Alcatel-Lucent 8920 Service Availability	Alcatel-Lucent
Alcatel-Lucent 8920 Service Quality Manager (SQM)	Alcatel-Lucent
Alcatel-Lucent 8920 Service Troubleshooting	Alcatel-Lucent
APG	WATCH4NET SOLUTIONS INC
AXIOSS Dashboard	Axiom Systems Limited
Bizitek Netflow BPM Suite	Bizitek
BSS / OSS Development & Integration Consulting	Amartus
Business and Technology Consulting	Sequoia Telecom Associates
CDR Analyzer	EXIS I.T.

http://www.tmforum.org/ProductsServices/ResourcePerformance/2564/Home.html

Summary 1/2

- Size matters
  - Large software systems are much harder to create than smaller ones
- “Horses for courses”:
  - Agile is good for exploration
- Developing large systems requires collaboration
- Collaboration requires coordination
- Coordination requires reference frameworks
  - Architectures
  - Common language and definitions
- Science projects are now adopting “industrial” approaches to develop large software systems

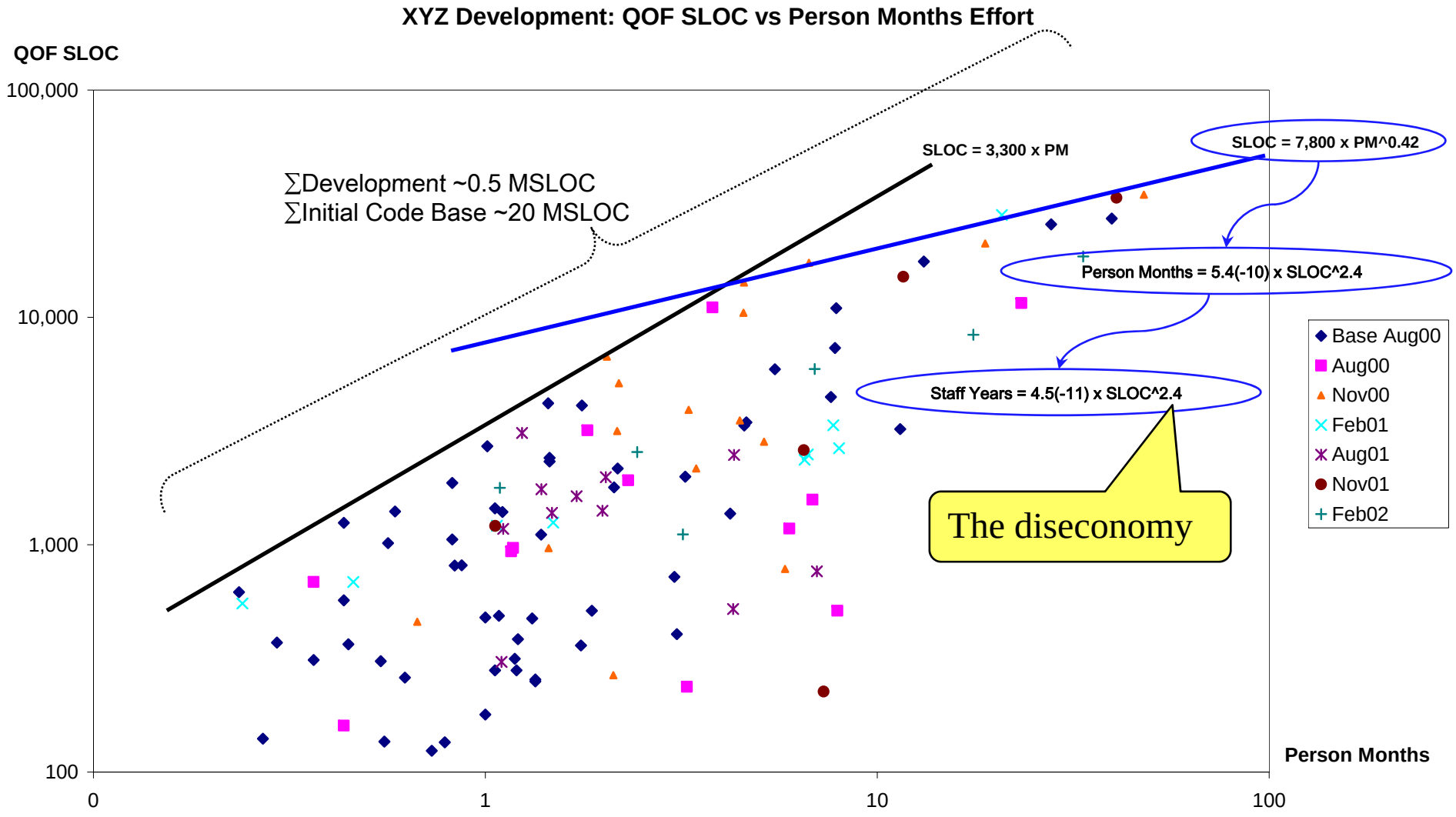
Summary 2/2

http://en.wikipedia.org/wiki/File:Pogo_-_Earth_Day_1971_poster.jpg

Backup slides

http://www.gbuwizards.com/files/powerpoint-cartoons.pdf

Diseconomies of scale: Case study 1



Software Development Survey [Gerry Harp]

- Was there a formal software process?
- Were architectural definition documents used?
- How did the change control board function?
- What was the review process?
- What were the team dynamics?
- How best to communicate across the team?
- Time and cost against estimated budget?
- What could have been done better?
- Suggestions for SKA software development?

Was there a formal software process?

- The process was not followed uniformly by all developers and was a continual source of tension regarding the effort that went into things such as code review, unit testing, documentation etc.
- It was the usual difficult trade-off where an under-resourced project is under intense pressure from the community to develop new things but is intolerant of defects
- In my opinion, more weight should be given to formal process in large s/w projects and the necessary time required to carry it out built into the culture ...

Were architectural definition documents used?

- We did not require such documents uniformly
- They were useful when they existed (minority)

How did the change control board function?

- The "change control board" met rarely and consisted of managers who did not understand the technical issues
- Much work was held up waiting for CCB action
- The smart engineers ignored the CCB and got away with making design changes without submitting them for approval

What was the review process?

- There were no formal reviews
- Individuals reviewed each others code, but major problems arose because some developers were not open to constructive criticism

What was the review process?

- No formal process
- As the lead programmer I reviewed all the code, but there were no agreed upon standards
- Several programmers were very protective of their code and would not let anyone else make changes (or suggestions)

What was the review process?

- Through use and bug reports
- Designated developers reviewed code by inspection
- Unit tests were required
- The review load became too large and eventually was dropped

What were the team dynamics?

- My personal opinion is that you need a minimum of 4-6 people in one place with a commitment to actually attend at least a couple of days a week or a week per month
- Software developers can be very productive in isolation and instant messaging works very well but places with about the minimum or more in one place are more productive per head
- [Project] is spread too thinly for political reasons, as all the participating institutes wanted a few bodies
- As the alternative was probably not to get the bodies at all, it probably works as well as possible
- But 20 people in ~4x5 instead of ~10x2 people (roughly) would be better!

How best to communicate across the team?

- Meetings - but not too much travelling - try and optimise/alternate for everyone, have onex3 days instead of 3x1 day etc...!
- Workshops - if a project is as thinly spread as [project], then as an astronomer/designer, I found it invaluable to be working in the same room as software developers
- Mostly just head-down but being able to say:
  - 'is this possible‘
- Or for them to say:
  - 'what do you mean by this‘
- And consult as required

How best to communicate across the team?

- Frequent phone contact
- Face-to-face, on-site contact was required for system integration
- email usually initiated a phone call
- We also had detailed interface documents

- ‘Tiger' team short-term (< 3 month) visits

Time and cost against estimated budget?

- It's now on track after being re-baselined

- Way behind compared to original 1992 promises
- [Euro / $] 30% over compared to my 1995 estimates
- The goal posts have moved sufficiently that it's not easy to tell for more recent baselines

What could have been done better?

- Better documentation!

- Programmer documentation and programmer support
- Often I find myself spending long lengths of time trying to figure out how to (re)use an existing code (class) and/or make extensions to an existing code

- Common coding standards
- More unit tests

What could have been done better?

- More standardisation - more common software libraries etc.

- Get more users and testers to collaborate with to test early and often
- A better costing of human-hour for different sub-projects and hence more human resources to do the job

What could have been done better?

- Since standards were not established, there were a variety of coding styles making it difficult for some people to read code written by others

- There was also a perception of program ownership which caused problems in resolving bugs

What could have been done better?

- We did not succeed in generating an interface (be it command-line or graphical) that users liked
- User interfaces are extremely difficult and extremely time consuming to develop

- User interface is very poor
- I allocated too few resources to developing a simple to use robust interface

Suggestions for SKA software development?

- The concept that hardware is the major investment and value is no longer valid and much less so with newer instruments
- The investment in software is significant (if not dominant) when we consider software that gets developed throughout the life time of a telescope
- Lack of software investment sometimes prevent making full use of the hardware for years so one has to be careful of not underestimating it

Suggestions for SKA software development?

- Consult/work with professional software developers
  - Rather than using predominantly software focused astronomers
- Use a package-based architecture
- Ensure the interface is clear, intuitive and makes it easy to manage the catalog

Suggestions for SKA software development?

- Identify specific tasks that users want to perform on their data
- Use model science/use cases as guidance
- Try out on 'real' users as frequently as possible
- [Ensure] science input/oversight

- Avoid developing too many "features"
- Start small and with an architecture
- At least have small iterations and automated tests

- Start early

Suggestions for SKA software development?

- Produce layered documentation for both software engineers and average astronomer end-users
- Use … standards wherever these exist and are applicable – or other standards developed in the radio community
- Develop new standards cooperatively if necessary
- Minimise the unfamiliarity in both terminology and 'look and feel'

Suggestions for SKA software development?

- Have a feasible process for your culture
- Do not accept personnel contributions smaller than 50%
- Keep it simple
- Focus on architecture
- The penalty that comes with distributed software projects is quite high
  - Don't underestimate it

Suggestions for SKA software development?

- [Version control] goes without saying
- Systems Requirement Document must detail all scientific and operational functionality
- Define programming and documentation standards that are accepted and followed by all developers
- Conduct regular code reviews
- Define in detail the software interfaces between subsystems and build software simulators for all subsystems
- Define a test plan and testing procedures – unit and system tests
- Develop prototypes, then re-factor until you reach the final product

Suggestions for SKA software development?

- Develop a software engineering process and use it:
  - To do this you *MUST* provide a culture in which it is expected that things such as design documents, code review and testing are just as important as writing new code
  - You must enforce this as much as is practicable
  - You can help enforce this through your formal project review processes
- Ensure developers are face to face as much as is possible when it really matters:
  - This is most important during early concept and design, and prototype implementation
- Make sure the user community is heavily involved the whole way through; not just at the beginning and not just at the end:
  - The users must take responsibility as well as the developers for the health and progress of the project
  - Project scientists (and support teams) are vital to:
    - Help develop science requirements
    - Convert them into software requirements
    - Ensure that modules are being developed as stated
    - Ensure that the functionality is as agreed to – or to negotiate changes to

Suggestions for SKA software development?

- [Build a] good estimate of the magnitude of the problem
- [Employ] good managers
- [Develop an] optimal release schedule:
  - Not too short so that it prevent long term development
  - Nor too long so as to get feedback from users to make corrections

Suggestions for SKA software development?

- Keep it simple:
  - Start by implementing only the absolutely essential functions
  - Omit all "nice to have" features
  - Ignore demands for anything pretty or fancy
  - Write down a description of this minimum system at the beginning
- Use only proven and well established tools, operating systems, and code libraries
- Resist … demands to incorporate the latest "hot" techniques and languages
- Hire … the best available people
- Maximize their ability to communicate – [if possible] keep them all at one site
- Add more only when it would clearly increase – not decrease – productivity
- If schedules slip, de-scope or defer functionality rather than increasing staff

Suggestions for SKA software development?

- Architect a platform to cater for changing requirements and configuration over lifetime of use
- Define and closely manage software development cycles that follow a formal process and are time and output constrained
  - i.e. take an iterative approach
- Allocate significant development and test effort to move 'prototyped‘ functionality into a production-ready state

Suggestions for SKA software development?

- Make a realistic estimate of the manpower requirements
- Document the code and the design formally and carefully:
  - This … will be crucial in SKA, where software is likely to be the dominant and a large component
  - Hence it will not be done by a small tight group, probably not even co-located and will not have people who wrote the code even available to the project for long
- Ensure that the software is as easily usable by the end user as it is for the people who will use to develop new algorithms and techniques
- Also ensure that the latter kind of researchers are explicitly supported

Suggestions for SKA software development?

- Single lead engineer/key decision maker
  - I think the greatest problem we had on the [project] was that there was not a lead engineer/decision maker who ensured that the overall system was being developed as it should be, and that it behaved as required and expected
  - Nobody was really in charge
- Unit testing
- Code reviews would probably be useful

Suggestions for SKA software development?

- Capabilities should be engineered end-to-end and only released for general use once debugged all the way through
- New capabilities should be brought online piece by piece
- Software should support only approved modes of observation
- This avoids limits the spread of complexity from the telescope into the reduction software

Greg Wilson: “Scientists would do well to pick up some tools widely used in the software industry”

- “Software Carpentry” – topics include:
  - Version control
  - Automating repetitive tasks
  - Systematic testing
  - Coding style
  - Reading code
  - Basic data crunching
  - Basic Web programming
  - A quick survey of how to manage development in a small geographically distributed team
- “None of this is rocket science – it’s just the programming equivalent of knowing how to titrate a solution or calibrate an oscilloscope”
- For example – software system integration:
  - “Good programmers don't write programs: they assemble them
  - Combine tools and libraries that others have written
  - Thereby creating something that others can then recombine
  - This lecture explores various ways to combine things
  - Helps a lot to design with combination in mind”

http://swc.scipy.org/
http://www.americanscientist.org/issues/pub/wheres-the-real-bottleneck-in-scientific-computing