How A Commercial Software Product Is Architected

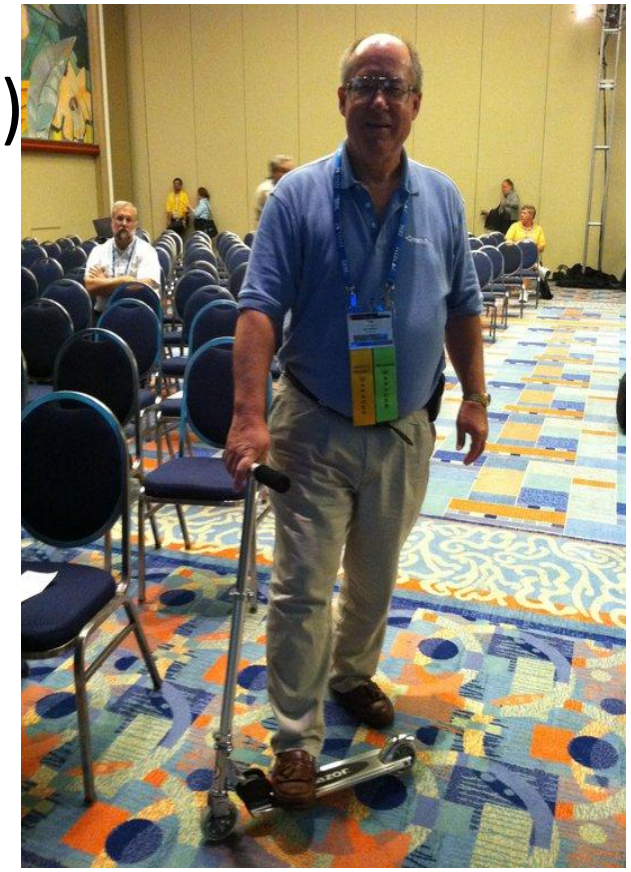
Session 11180

August 8, 2012

tom_harper_cw@bmc.com

Background

- Wrote first code in 1960 for IBM 1620 (fifty-two years ago)
- Coded in machine language (was no assembler yet)
- Have written products that have sold over one billion $
- Helped start two software companies (BMC and Neon Enterprise Software)
- Owner of eleven software patents



Conceptualizing The New Product

- Categories of new products
  - Brand new; never been done before
    - First spreadsheet
    - First DBMS
  - Redo of an existing product
    - Faster SORT
    - New IEBGENER

Market Changes

- In last thirty years, many development-type personnel now work for ISVs
- People in installations are primarily installers and diagnosticians
- To them, a new product is just work and risk
- Hard to even find Beta test sites any more

Brand New Products

- Best choice
- No competition
  - In software, being first is everything
  - Second is irrelevant
- Half of all new products fail
  - Be ready to accept this
  - Part of the business
- Hard work

Redo of Existing Products

- Somewhat less risk
- Primary competitor is inertia and bundling
- Performance is only one criteria
  - Price (lower is not always perceived as better)
  - Stability
  - Risk
  - Conversion effort
  - Change control

Resources Required

- Hardware and software environment for development, testing, QA, and distribution
- Strong financial backing
  - Most customers will not want to purchase from you unless you have a strong balance sheet
  - Most developers seriously underestimate the time it takes to develop a product
- Legal and support infrastructure
- Software licenses
- Documentation
- Backup and recovery; offsite archive

Resources Required

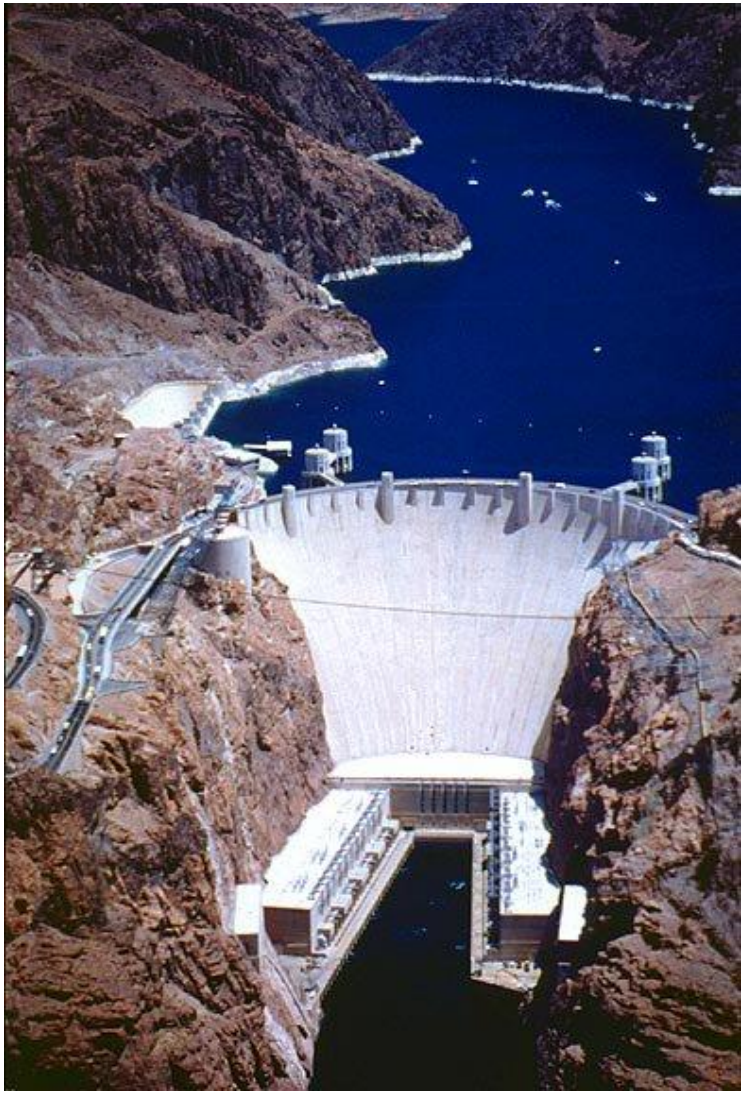
- The key idea code that the product is based on is seldom that large
- Bulk of the code is mundane
  - Reports
  - Messages
  - License code checking
  - Failure information
  - Diagnostic tools

Legal Protections

- Open source?
- Trade secret?
- Patents?

Choosing A Language

- HL Assembler
  - No restrictions
  - No license issues
  - Advanced MACRO facility
- Structured Programming
  - Reduced APAR rate
  - Long-preferred methodology

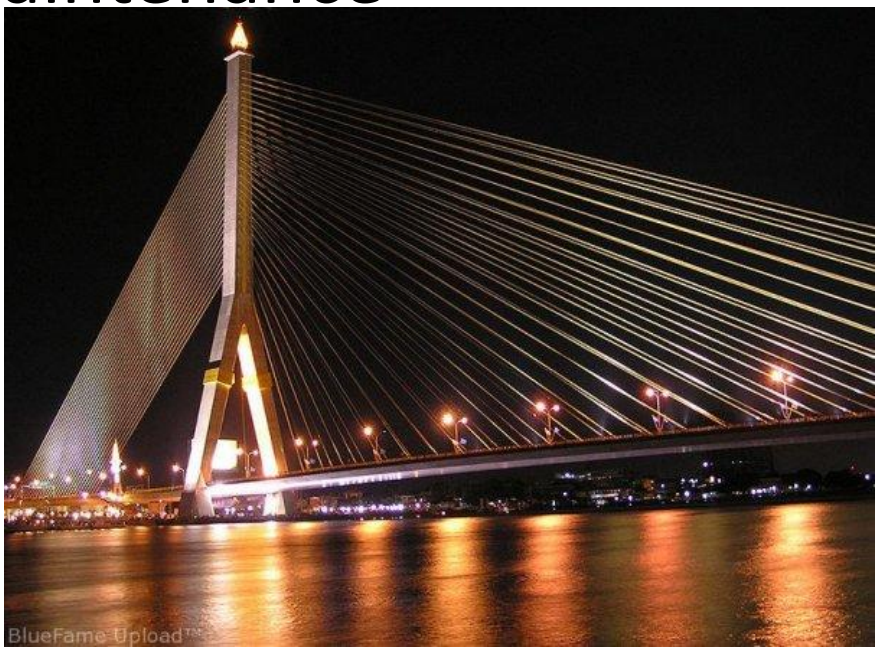


Language Choice

- Very controversial
- SAS has announced it will drop support of C
- Assembler always works in every z environment
- High-Level Assembler and Structured Programming Macros weaken other high-level language arguments

Technical Design Fundamentals

- Library Management System
  - Allows for BUILDing and PROMOTEing
  - Allows for multiple versions and releases
- How to deliver maintenance
  - Cumulative
  - Individual
  - SMP/E
  - IEBCOPY



Creating Building Blocks

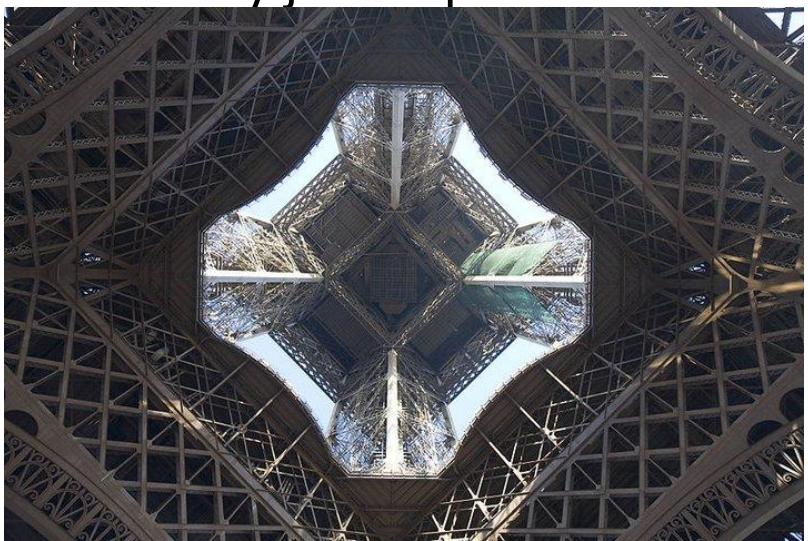
- Encapsulate functions
  - Perhaps require an INIT call which returns a token
  - Token is address of a control block which is used for subsequent operations
  - Termination call when finished
- Consider stacking PCs for some functions
  - Can be slow
  - Can be faster if you want to tolerate callers in any AMODE, ARMODE, programming state, etc.

Creating Building Blocks

- Functions should issue their own messages and issue return codes
- Any resources acquired should be released in case of an abend
- Macros should be used to invoke
- Optional tracing should be provided

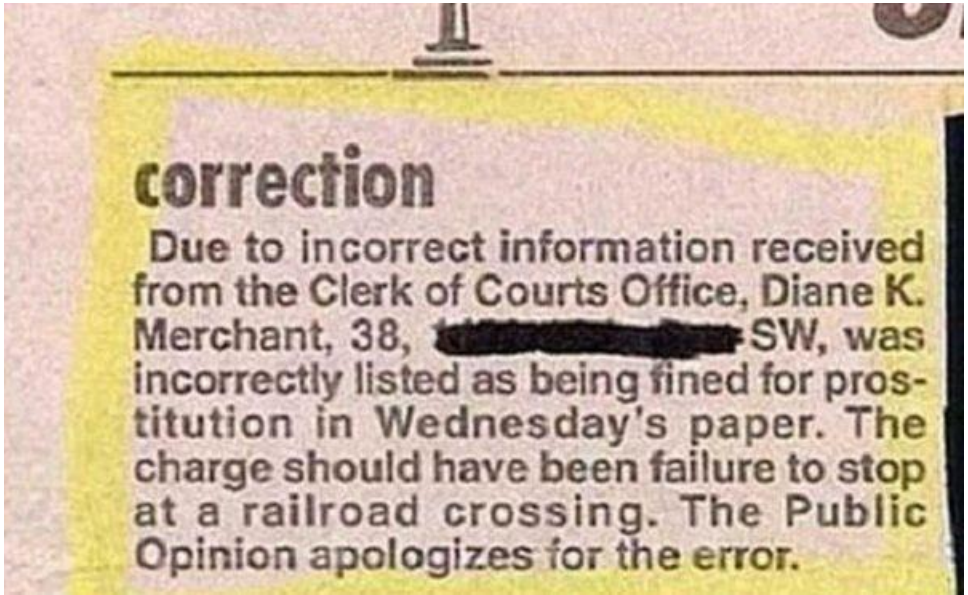
Managing Storage

- Save areas and local working storage
  - One STORAGE OBTAIN | RELEASE per unit of work
  - Should step down and up to new | prior save area
  - R13 should point to local working storage
  - Provides macros to accomplish this
- Storage service (a primary building block)
  - Should provide cushions to allow for recovery to run
  - Storage owned by job step task



Managing Storage

- Don’t forget cell pools and data spaces
- Exploit subpools and keys
- Avoid key zero storage whenever possible
- Be aware of high and low private, and of backing storage
- Never do STORAGE OBTAIN|RELEASE in critical path code

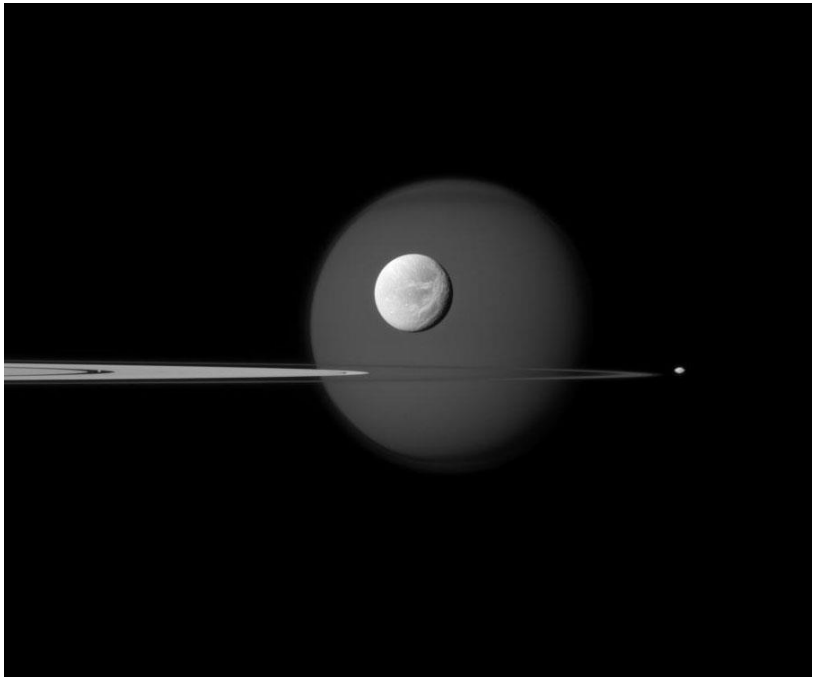


correction

Due to incorrect information received from the Clerk of Courts Office, Diane K. Merchant, 38, [redacted] SW, was incorrectly listed as being fined for prostitution in Wednesday's paper. The charge should have been failure to stop at a railroad crossing. The Public Opinion apologizes for the error.

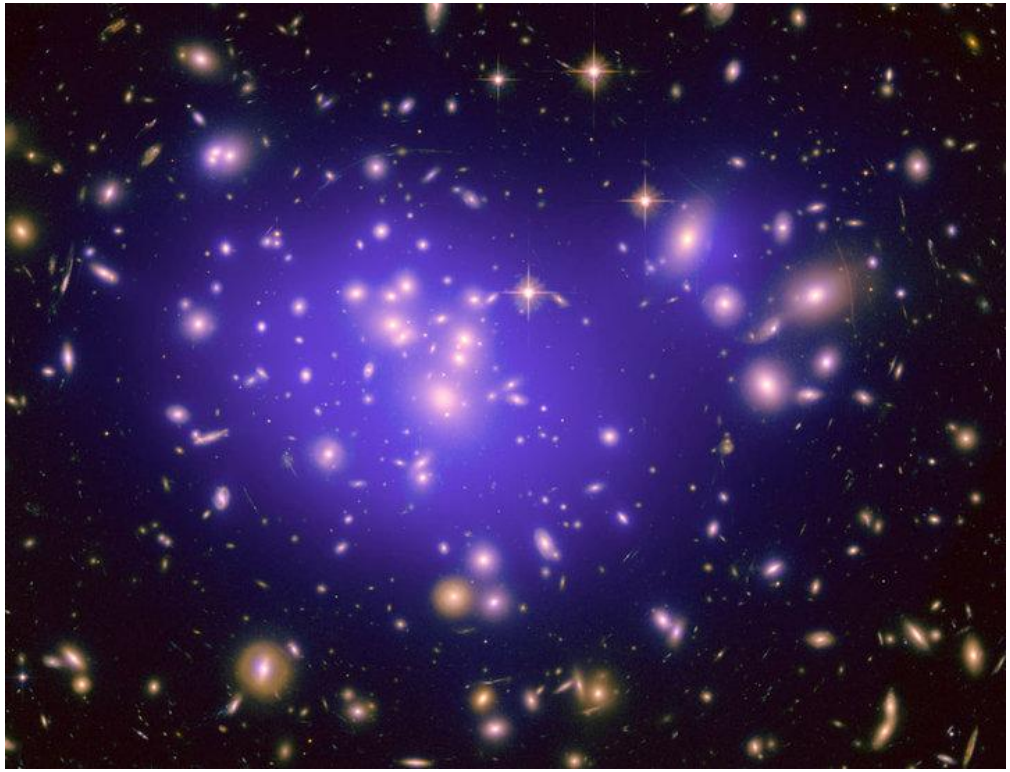
Diagnostics

- Should be built-in day one
- CTRACE
  - Whenever external facility is invoked
- First Failure Data Capture
- LOGREC
- System log
- SDUMP
- IPCS Formatting



Organizing Your Code

- Understand what is done once and what is done in critical path



Coding Guidelines

- Write RENT, “baseless” code
- Have standard eye-catcher block in module which contains assembly date and time, fixes applied, z/OS level assembled on, who assembled, etc.
- Have standard eye-catcher in front of control blocks

Documenting Code

- Always place documentation in code
- Anywhere else gets separated and lost
- Include a log of changes
- For MACROs, follow guidelines similar to IBM newer MACROs.

Managing Code

- Consider a product for managing your source code, MACROs, LIST data sets, etc.
- SCLM is a good choice, and it is kind of free
  - Requires PDSEs
  - Performance is not a strong point
  - Adequate function

Messages and Print Lines

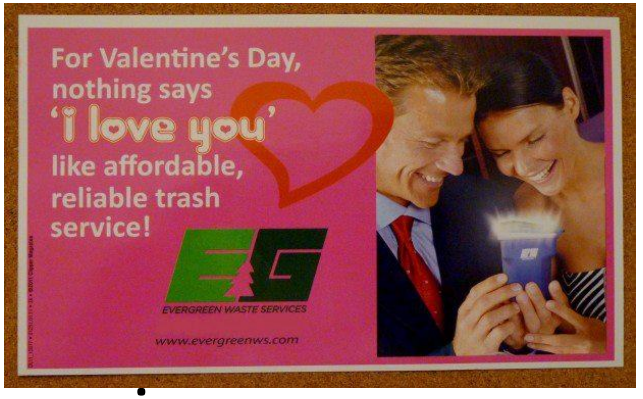
- Externalize in a PDS
- Allow for symbolic substitution
- Allow of choice of date formats
- Allow for various substitution methods:
  - Hex
  - Numeric
  - Character
  - Date / Time
  - Scaled numbers

About Performance

- Understand what performance metrics are important to your customers
  - Elapsed time
  - Minimization of CPU time
- Consider multiple TCBs/SRBs
- Consider generating code
- Instructions are free; cache lines are expensive

I/O

- IBM's access methods are adequate in most cases
- For extremely performance-sensitive applications, consider a STARTIO driver
- Not difficult, but documentation is hard to come by
- More flexible
- Needs constant attention
- Needs another entire day of sessions



Authorized Programming

- Many functions require your code to be authorized, so often unavoidable
- Supervisor state in itself is not dangerous
  - Key zero is dangerous, and should be avoided
  - Ignorance is also dangerous; no known cure for stupid
  - Protect your code from other idiots: Use PROTECT MACRO on your own code.

Recovery Routines

- Important to gather information about failure
- Consider writing information to multiple places in case customer loses it:
  - Console LOG
  - LOGREC
  - SDUMP
- Release storage cushions

Recovery Routines

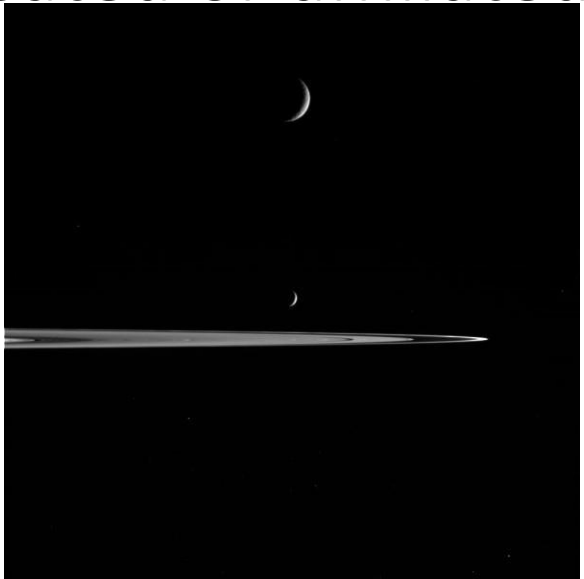
- Display save area chains, linkage stacks
- Display module IDs, assembly dates and times, and fixes installed
- List options in effect
- List z/OS level and any other software levels
- List PSW, AMODE, ARMODE, condition code
- Determine service issued, if any
- List of BEAR register

Recovery Routines

- Build SYMREC Symptom String
- Build meaningful dump title for SDUMP
- Take SDUMP
  - Retry SDUMP is address space is busy
  - Print out SDUMP data set name
  - Analyze SDUMP return code

Recovery Routines

- Release resources
  - Latches in SRB mode
- Consider building in your debugging tool, such as XDC, based on presence of a DD statement in your JCL (not compensated or affiliated in any way with Colesoft)

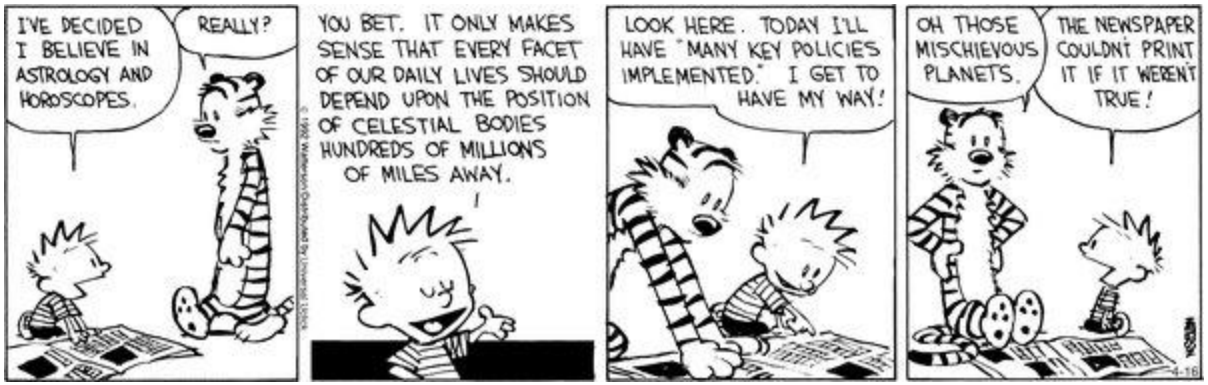


Design

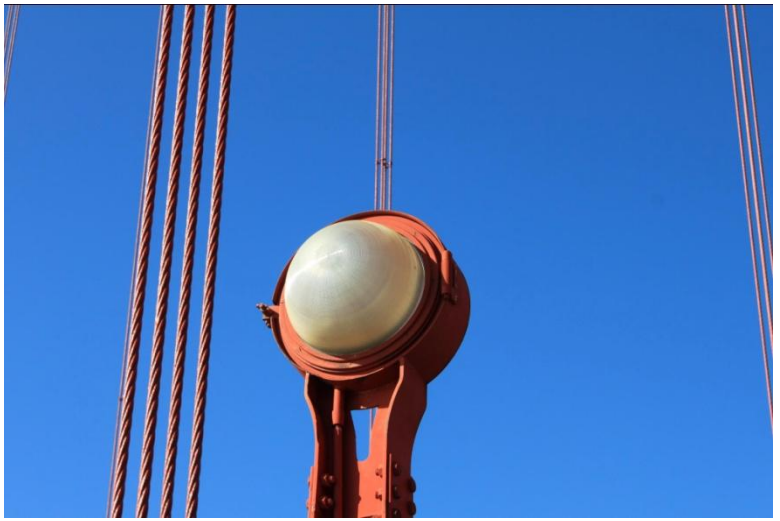
- Design is all-critical
  - Implications in development time
  - Implications in support
  - Implications in success
- No matter what
  - Do not develop under time pressure
  - Do it right the first time
  - Having deadlines does not make code born any sooner

Design

- Consult with other experts
- Try not to copy what has been done before
  - Different criteria in effect years ago
  - Don’t propagate poor design choices
- Do try and utilize existing services if they are stable
  - z/OS System Logger (great track record)
  - VSAM RLS (poor track record)



Design



- Control blocks are key
  - Primary control block
    - Always passed as parameter
    - Always pointed to by same register
    - Embed in initial module and change key
    - Allows VCON access
  - Make easy to access with DO loops
  - Think carefully!
  - Allow reserved space

Design

- Register assignments
  - R14,R15,R0,R1 – Linkage registers
  - R13 – Save area and local work storage
  - R12 – Base register for literals
  - R11 – Primary control block
  - Remaining registers – other control blocks
- Avoid FP registers
- Be conscious of all 64-bits

Design

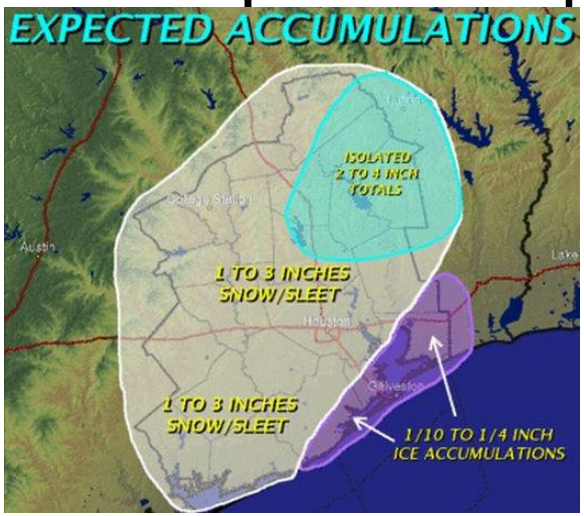
- Separate initialization | termination
- Only optimize critical path code
- Design is more critical to performance than compilers trying to optimize a poor design
- Remember, instructions are free; cache misses are expensive

Design

- Example of processing an IMS data base
- Prior designs used LRU buffer pool
- Basically, block-faulted through data base
- New design inverted IMS insertion algorithm
- Performance increased by a factor of six

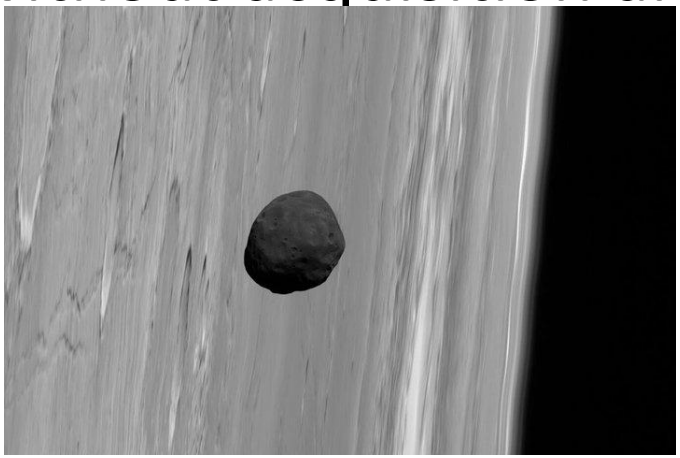
Design

- Be aware of DASD and TAPE architecture
- Unit of transfer is a track
- Tape blocks are compressed and written as 256K byte blocks
- Implement parallelism with multiple TCBs | SRBs
- Be aware of serialization



Design

- Serialization – Avoid!
  - Better to have a dedicated control block per thread
  - Roll up stats at end of process
  - Especially avoid in critical path code
  - Cell pools use CS or CDS
- Services which provide tokens which provide work space without acquisition are fastest



Prototyping

- Before huge effort is invested, prototype
- Time to diagnose performance issues

Team Staffing

- Select team members carefully
- Don’t select all “chiefs”
- Seldom are more than six people effective
- Remember “The Mythical Man-Month”
- Review all critical issues with at least one other person

Post Mortem

- Was product successful?
  - Why or why not?
  - Customer evaluation and feedback
  - Never-ending process
  - Education
  - New functions to be exploited