Software Project Case Study:
The LIGO Data Analysis System

Project Science Workshop II
Williamsburg, VA
November 13th – 16th, 2002

Kent Blackburn
LIGO Laboratory
California Institute of Technology

Overview

- *Setting the Stage*
- *Concept to Planning*
- *Development of Requirements*
- *Architectural and Detailed Designs*
- *Software Construction*
  - *Risk Management*
  - *Quality Assurance and Testing*
  - *Metrics*
  - *Deployment*
  - *Change Control*
- *Operations and Maintenance*
- *Summary & Lessons Learned*

Definitions

1. *LIGO - Laser Interferometer Gravitational-Wave Observatory*
2. *VIRGO – French/Italian Interferometer Gravitational-Wave Project*
3. *IFO - Gravitational-wave interferometer*
4. *LDAS - LIGO Data Analysis System*
5. *LSC - LIGO Scientific Collaboration*
6. *GriPhyN – Grid Physics Network Project*
7. *iVDGL – international Virtual Data Grid Laboratory Project*
8. *LAL - LIGO/LSC Algorithm Library*
9. *DSO - Dynamically loaded Shared Object (resolved at runtime)*
10. *CDS - Computer & Data System*
11. *CVS - Concurrent Version System, software version control*
12. *OO/OOP - Object Oriented / Object Oriented Paradigm*
13. *ODBC - Open DataBase Connectivity Standard*
14. *Frame - Unit of IFO data, typically stored on disk or tape*
15. *ILWD - Internal Light Weight Data Format for use in LDAS*
16. *XML - eXtensible Mark-up Language (extensive wrt HTML)*
17. *RDS - Reduced Data Set (Frame)*
18. *Beowulf - Inexpensive cluster of computers (PCs)*
19. *MPI - Message Passing Interface, a parallel computing library*
20. *RAID - Redundant Array of Independent Disks*
21. *JBOD - Just a Bunch Of Disks*

Software Project Case Study:

The LIGO Data Analysis System

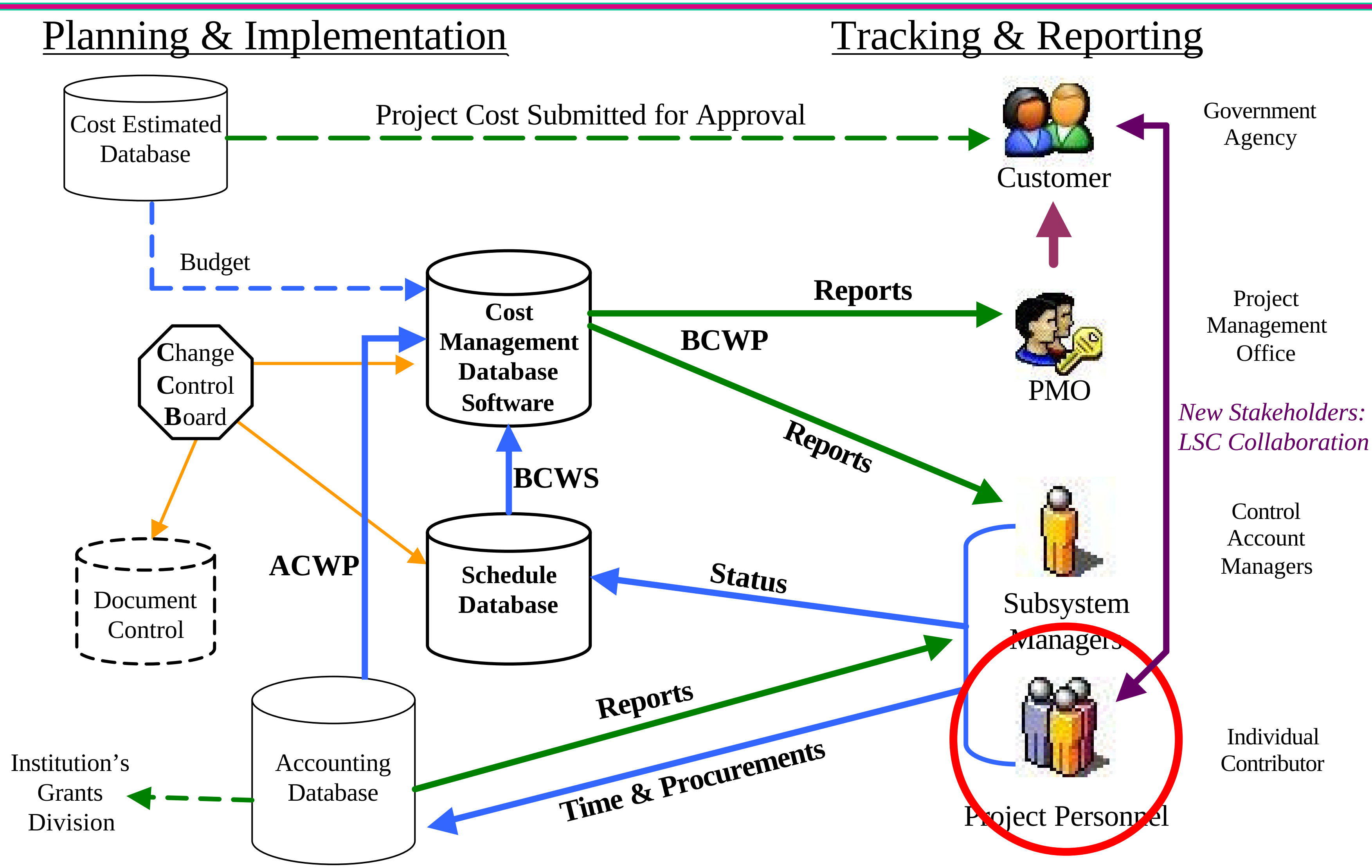
Setting the Stage

Motivation Behind This Talk

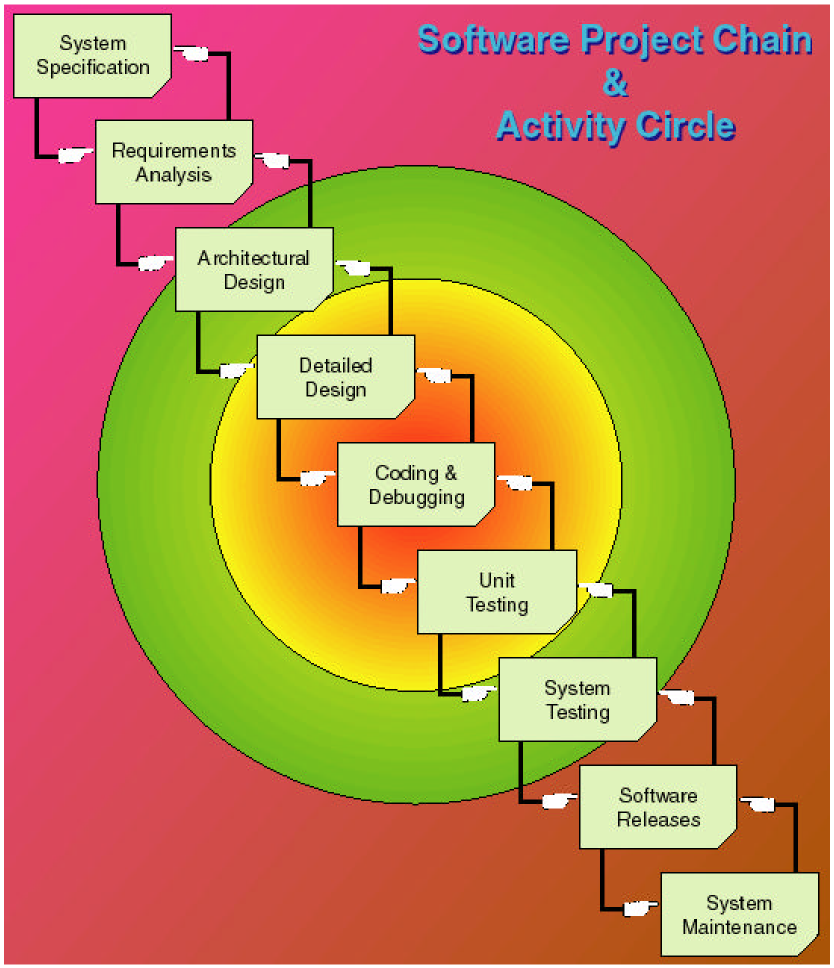
- *Software project management has unique problems.*
- *Software now vital to achieving the goals of large scientific projects.*
- *Reasons for my giving this talk at this time.*
  - *LDAS is 90% complete*
    - *Next 6 to 9 months dominated by porting, reliability and performance tuning.*
    - *Successfully used in all Engineering Runs and first Science Run.*
      - *Strong user community participation!*
    - *Glimpse at the statistics (estimates):*
      - *A million lines of code built in-house*
      - *A million lines of commercial code*
      - *A million lines of borrowed (open source) code*
      - *22 man-years to reach 90% in 4.5 calendar years.*

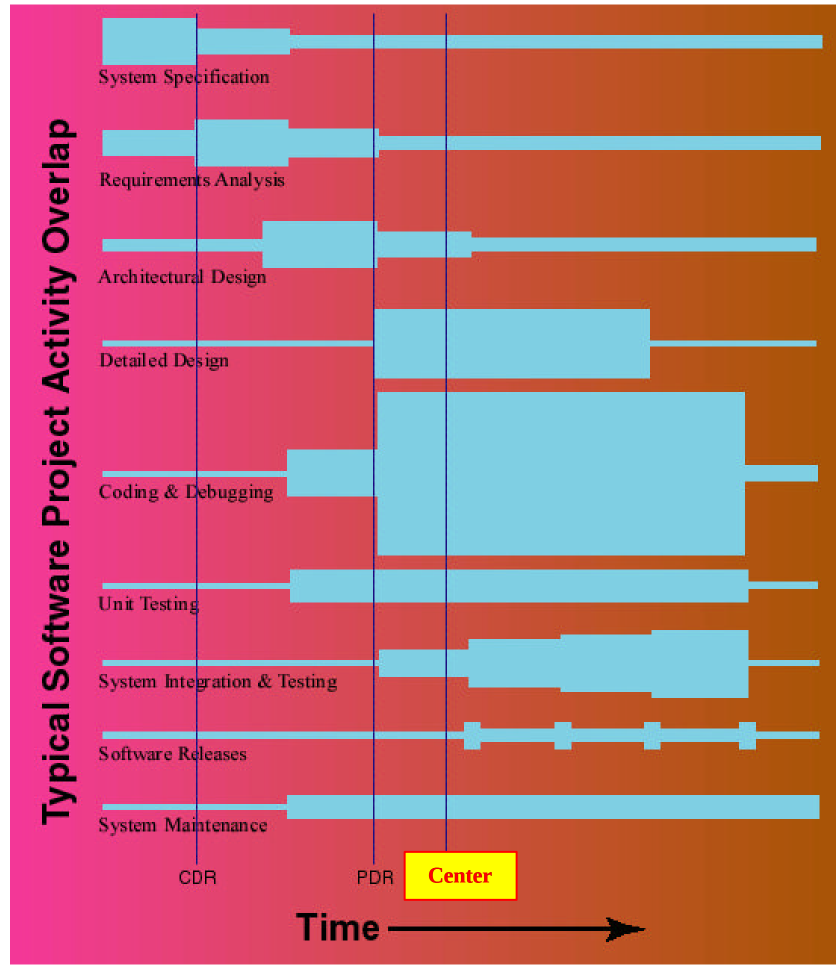
Where Does LIGO Software Project Fits into Yesterday's PMCS

Richard Fischer
Project Management Services, Inc.



Software Development Cycle





Upstream flow does happens!

Work profile as a function of time!

The Software Culture

- *Must now include software developers to list of cultures.*
- *Differ from scientists in motivation and commitment.*
- *Most often hired as contractors or consultants.*
  - *80% of LDAS software developers are contractors.*
  - *Management of LDAS software project retained by Laboratory.*
  - *Has unfortunate side effect of isolating software developers*
    - *Lack strong ownership to deliverables and schedules.*
- *Results in interesting team dynamics with challenges for managers.*
- *Software market supports huge diversity in technical skills and formal education of available software developers.*
- *Expertise tends to be very trendy, e.g., C++ was hot, now its Java.*
- *Importance to today's civilization compared to that of the builders of the aqueducts and ditches in the times of the Roman Empire…*
  - *From a 1994 NASA Goddard Science Colloquium given by a sociologist.*
  - *If this surprises you, imagine your life without plumbing…or the benefits of software that runs the computers around us.*

Software Project Challenge

- *Software costs are going up, and hardware costs are going down (NASA/EOS estimated $100 per line of code).*
- *Software development time is getting longer, and maintenance costs are getting higher, while at the same time hardware development time is getting shorter and less costly – No Moore's Law for people.*
- *Software errors are becoming more frequent as hardware errors become almost nonexistent.*
- *Changing hardware technology is making software obsolete long before delivery!*
- *Only 25% of all software projects result in working systems*
  - *Likely to be lower in the aftermath of "dot.com - dot.bomb" era.*

Software Project Costs

Table 1: Software Project Costs by Development Phase

Software Project Phase	Percent of Project
Requirements	*3*
Design	*8*
Programming	*7*
Testing	*15*
Maintenance	*67*

Table 2: Cost of Correcting Software Errors

Software Development Phase:	Requirements Analysis	Design	Code and Unit Test	Integration Test	Validation and Documentation	Operational Maintenance
Development Funds	*5%*	*25%*	*10%*	*50%*	*10%*	
Errors Introduced	*55%*	*30%*		*10%*		*5%*
Errors Found	*18%*	*10%*		*50%*		*22%*
Relative Cost to Correct	*1x*	*1-1.5x*		*1-5x*		*10-100x*

Motivation for C++

- **Source**: Hughes Department of Defense Composite Software Error History.

Hybrid TCL/C++ Architecture

- *Advantages of object oriented programming languages extract a major price - complexity.*
  - *This leads to additional training requirements.*
  - *Introduces avenues for subtle misuses of the language.*
  - *Advantages come late in the software project: ease of extensibility, maintainability.*
- *C++ constructed as extension to C with similar syntax but very complex semantics.*
- *C++ evolved greatly as it matured to the ISO/ANSI Standard (no compilers there yet).*
- *Increasingly common for software projects to adopt a hybrid solution:*
  - *Scripting language can be 50 times better than compiled languages in lines of code per task.*
  - *Combine scripting languages with C++ - This is architecture used for LIGO Data Analysis.*
  - *Offsets risks and cost of pure C++ project.*

Table 3: Language Complexity of some of the more common computer languages

Language	Pascal	Modula-2	Modula-3	C	C++ v1	C++ v3	Ada	ANSI C++
Keywords	35	40	53	29	42	48	63	62
Statements	9	10	22	13	14	14	17	15
Operators	16	19	25	44	47	52	21	54
Ref. Man. Pages	28	25	50	40	66	155	241	650

Source: AT&T Lucent Technologies

Software Project Case Study:

The LIGO Data Analysis System

Concept to Planning

Historical Setting

- *LIGO construction proposal did not outline plan for analysis of data collected from interferometers.*
- *In June 1996, the NSF convened a panel chaired by Dr. McDaniel to review long term uses of LIGO once operations underway and to make recommendations…*
  - *One outcome from recommendations was to develop an environment to scientifically exploit LIGO data.*
- *In spring of 1997 LIGO authored the "White Paper Outlining the Data Analysis System (DAS) for LIGO I.*
  - *Authored by experts both inside and outside the LIGO Project.*
- *The LIGO Scientific Collaboration (LSC) was formed later that same year.*

Task Definition

- *Funding for data analysis system secured under the LIGO Project construction and later its operations budgets.*
- *Data Analysis System to be developed within LIGO Laboratory.*
  - *Requirements determined inside of LIGO.*
    - *LIGO planned and managed data analysis software project.*
    - *80% of software developers on-sight contractors.*
  - *This initiated the data analysis system in a "linear project" environment.*
    - *Much of this would change with the advent of the LIGO Scientific Collaboration and Grid Computing – leading to "complex project" reality!*
- *Followed closely project control methods used by LIGO sub-system.*
  - *Internal technical reviews:*
    - *conceptual design review, preliminary design review, ~~final design review.~~ mock data challenges*
    - *Brought in external reviewers to support of technical reviews.*
    - *Final design review replaced by "mock data challenges"!*

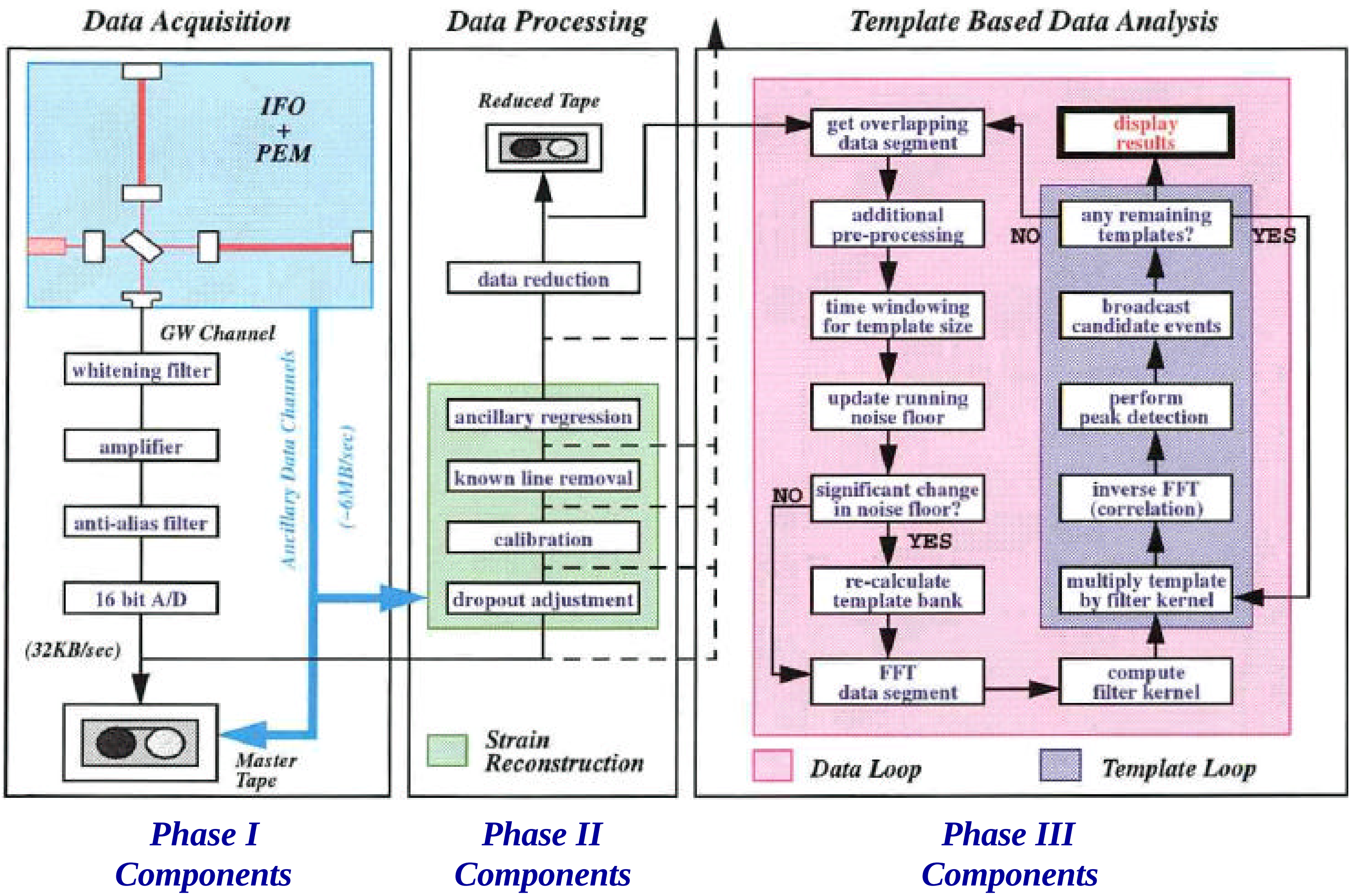
The LDAS Concept

- *The LIGO Science Requirements Document (SRD) calls out 24x7 operations with 90% duty cycle on each IFO.*
  - *No longer a collection of short experimental runs using a collection of software tools to analyze relatively small "chunks" of data.*
  - *All data needed to be analyzed in the time it takes to collect it – no spills!*
  - *Established the need for a **data-pipeline** system concept.*
- *LDAS needed to support the wide variety of astrophysical searches and detector characterization outlined in the data analysis white paper:*
  - *Binary Inspiral, Supernovae Bursts, Period Pulsars and Stochastic Background using gravitational strain signals and veto channels.*
  - *Detector characterization involving thousands of ancillary channels.*
  - *Because of the large volume of LIGO data, LDAS would also need to support data reduction pipelines.*

Define Boundaries: Interfaces and Scope

- *CDS would collect the data from the interferometer and write it to disk in the newly adopted Frame format.*
  - *Frame format developed in collaboration with VIRGO.*
    - *Added multi-organizational/international "complexity" to format.*
- *LDAS would (only) read raw Frame data to carry out the scientific investigations:*
  - *Avoids possible feedback paths into the instrument.*
    - *LDAS scope excluded"quick-look" real-time functionality that require interfaces with instrument – addressed by CDS.*
  - *LDAS responsible for writing all Frame data to tape and archive.*
- *Plug-&-Play interface for all search code modules.*
  - *Dynamic runtime loading of search code libraries.*
- *User Interfaces would be provided so that the LIGO (and the LIGO Scientific Collaboration) could "operate" the analysis system.*

Data-Pipeline Concept



Original Data-Pipeline Illustration - Circa 1996.

Estimate Computational Needs

- *Different scientific topics require different analysis methods.*
- *Computational cost dominated by two particular searches:*
  - *All-Sky Periodic Pulsar Search*
    - *Uses large (long time interval) Fourier Transforms.*
    - *The longer the time interval, the deeper (more complete) the search.*
    - *Total compute performance on the planet be inadequate to "complete" the search with LIGO I data – better is the enemy of the good!*
    - *Settle for what we can get on available computers – no longer dominates scale.*
  - *Binary Inspiral Search*
    - *Uses optimal filtering techniques to match each data segment to O(100,000) template waveforms.*
    - *Possible to keep up with LIGO I data with about 100 GFLOPS!*
- *No single computer provides this level of performance.*
  - *Distributed computing required.*

Estimate Data Input/Output Needs

- *LIGO has three distinct interferometers generating data.*
- *Each IFO collects about 5000 signals…today!*
  - *During the conceptual design of LDAS this was estimated at roughly 500 signals…10x growth!*
- *Each IFO collects about 3 MB of data per second.*
- *90% duty cycle from 3 interferometers generates 100 - 200 TB of data to archive per year.*
  - *Current tape storage will require hundreds to thousands of tapes.*
    - *Automated access to all those tapes requires tape robotics and large tape storage silos.*
  - *Hard disks are rapidly becoming competitive with tape storage!*

Conceptual Plan

- *Develop a distributed data analysis system around the concept of an analysis data-pipeline.*
  - *Must be capable of concurrently handling multiple analysis in support of different scientific topics while keeping up with data generation by LIGO's interferometers.*
  - *Data analysis systems will be located at the LIGO observatories, Caltech and MIT.*
    - *On-Site systems allow for tracking interferometer performance and conducting near real time looks on the universe.*
    - *Off-Site systems allow for more thorough post analysis after optimal instrumental characterization (calibration) has been achieved.*
      - *Caltech's data analysis system interfaces with large robotic tape storage unit.*
      - *Several LSC institutions configured local LDAS systems;*
        - » *Unforeseen support and change control issues result from our successes!*

Use Prototypes to Support Cost Estimation and Reduce Risks

- ***A good conceptual design must be strongly coupled to the technologies currently available … use prototyping to reduce risks associated with candidate technologies!***
- *Necessary technologies for the data analysis concept.*
  - *Distributed computing technologies*
    - *Sockets, Remote Process Control (RPC), Parallel Computing Libraries (MPI).*
  - *Use of Shared Objects (SO) in Steering/Scripting Languages.*
- *Distributed computing prototyped in pre-curser to genericAPI*
  - *Integrated C/C++ shared objects to communicated objects over sockets from within the TCL/TK steering/scripting language.*
- *Parallel computing prototyped using public domain MPI library from Argonne National Laboratory (MPICH).*
  - *Demonstrated basic parallel algorithms over existing Sun workstation.*

Formal Review of Concept

- *The Conceptual Design Review for LDAS was held in December of 1997.*
  - *Primarily reflected requirements for scientific scope and goals of LIGO data analysis.*
- *Review committee made up of senior LIGO Project staff.*
- *Conceptual Design and Requirements Documents captured in LIGO Document Control Center.*
- *Looking back, requirements for scientific goals were well mostly preserved over time … however, many implementation plans did not survive into detailed design.*

Software Project Case Study:
The LIGO Data Analysis System

Development of Requirements

Generic Design Requirements

- *Software Design:*
  - *Design Requirements*
    - *Efficiency*
    - *Portability*
    - *Modularity*
    - *Extensibility*
    - *Flexibility*
    - *Maintainability*
  - *Design Components*
    - *Computer Languages*
    - *ISO/ANSI, ODBC Standards & Data Formats*
    - *Relational Database*
    - *Modules*
    - *Libraries*
    - *User Interfaces including The World Wide Web*
- *Hardware Design:*
  - *Design Requirements*
    - *Computational Performance*
    - *Network Bandwidth*
    - *Spinning Disk Storage*
    - *Tape Archive Storage*
    - *Connectivity*
    - *Security*
  - *Design Components*
    - *Servers / Beowulf Cluster*
    - *Fast Ethernet / Gig-E*
    - *RAID / JBOD Disks*
    - *HPSS / SAM-QFS*
    - *Wide Area Networks*
    - *Private Local Area Networks, Gateways and Firewalls*

Software Standardization

- *The most significant risk reduction action during the conceptual design is to formalize standards for the project.*
  - *Loose software standards sink software projects!*
- *Identify target set of hardware, operating system, computer languages and other software technologies to meet requirements.*
  - *Highly recommended that software style issues be formalized*
    - *E.g., C++ language supports procedural and object oriented coding styles – LDAS adopted strict object oriented style where possible!*
- *Where possible, take advantage of existing standards.*
  - *POSIX, ISO/ANSI languages, etc.*

Documentation

- *Formally document and publish all requirements, specifications, reviews, test results and user guides.*
  - *Conceptual Design Requirements.*
  - *Preliminary Design Requirements.*
  - *~~Final Design Requirements~~ … "Mock Data Challenge Reports".*
- *Use the Web, it's a marvelous publishing media.*
  - *LDAS website - http://www.ldas-sw.ligo.caltech.edu*
- *The LDAS plan called out writing a "baseline" requirements and "baseline"specification for each software module.*
  - *These were tremendously useful for focusing the implementation effort of the software developers!*

Software Project Case Study:

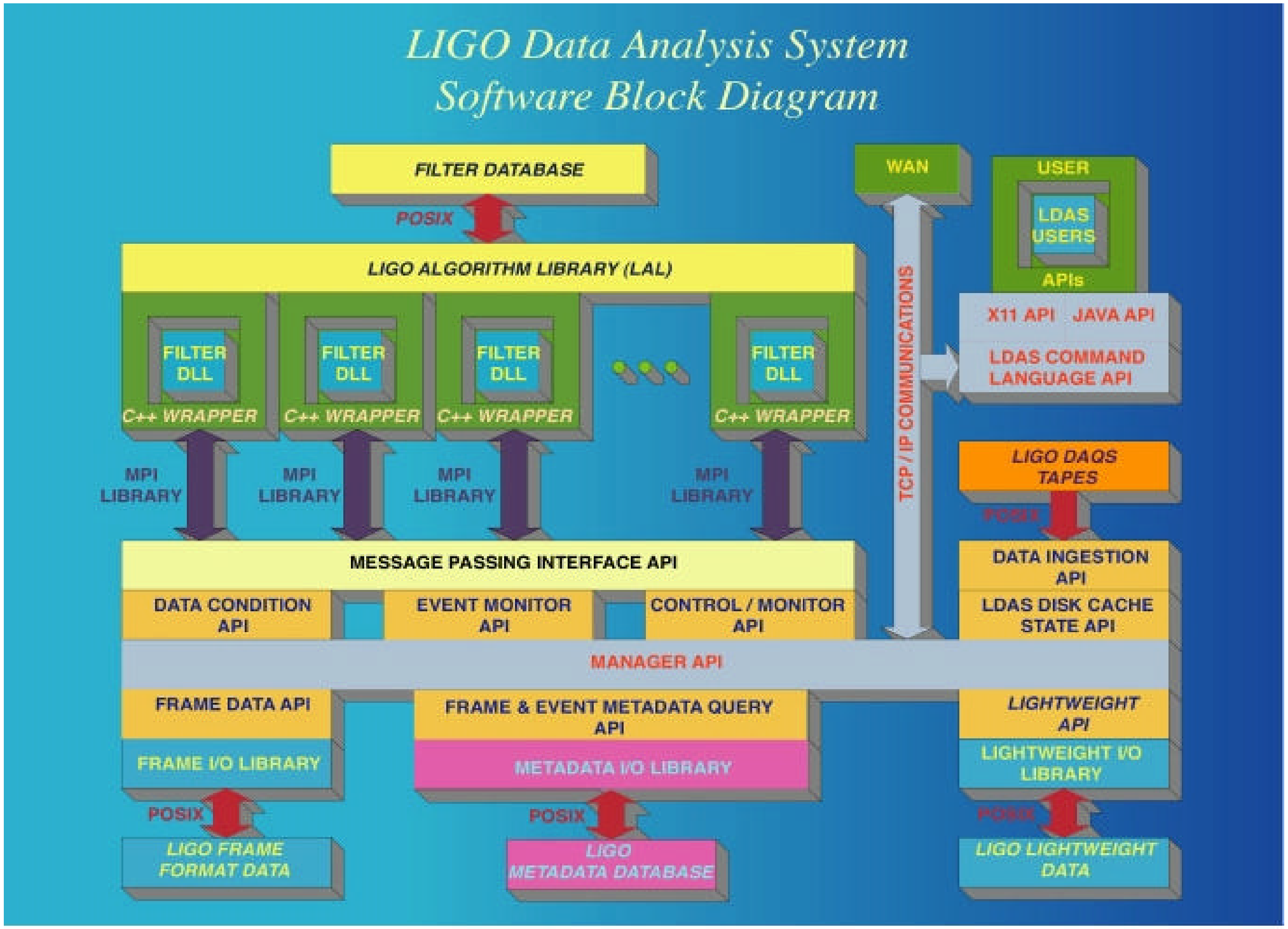
The LIGO Data Analysis System

Architectural and Detailed Designs

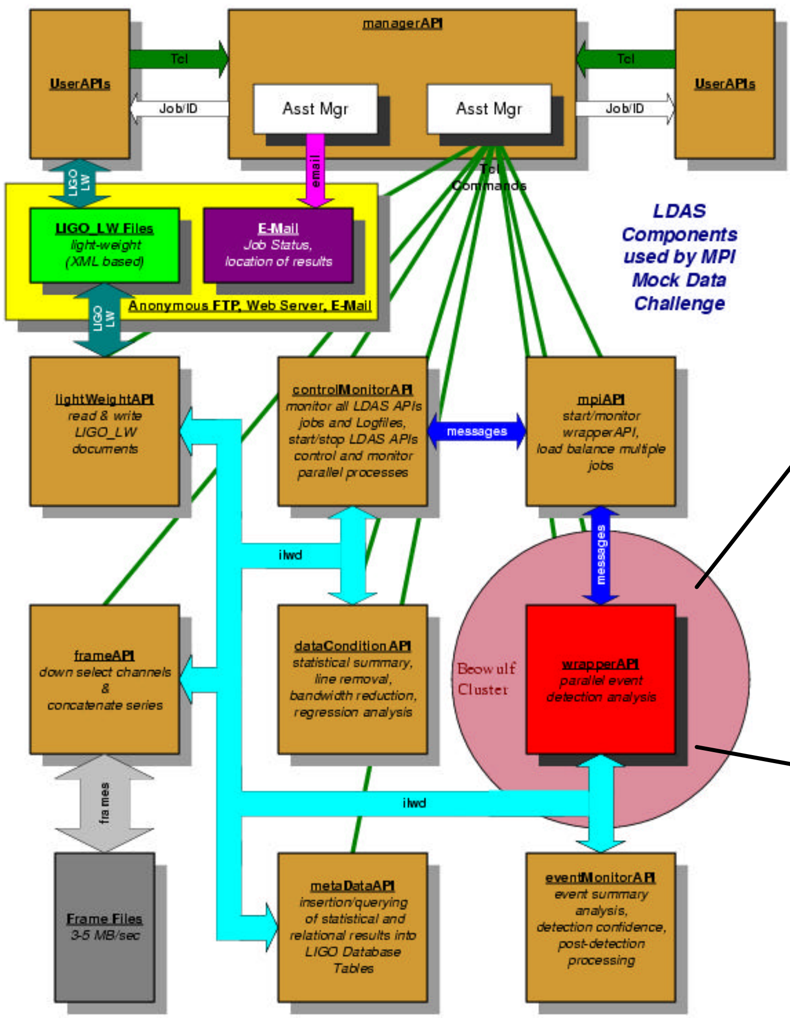
Divide and Conquer

- *Divide software project into individual software modules based on functional requirements.*
- *Identify any common functionality found in all software modules and break it out into libraries.*
- *Estimate the complexity of each software module based on the requirements – this will assist in setting the relative risks and costs of each software module.*
- *If any software module stands out in estimate, break it up further into sub-components that are roughly equal in complexity to all other software modules.*
- *Determine prerequisites for each component.*
- *Integrate each component with its prerequisite dependencies into baseline schedule.*

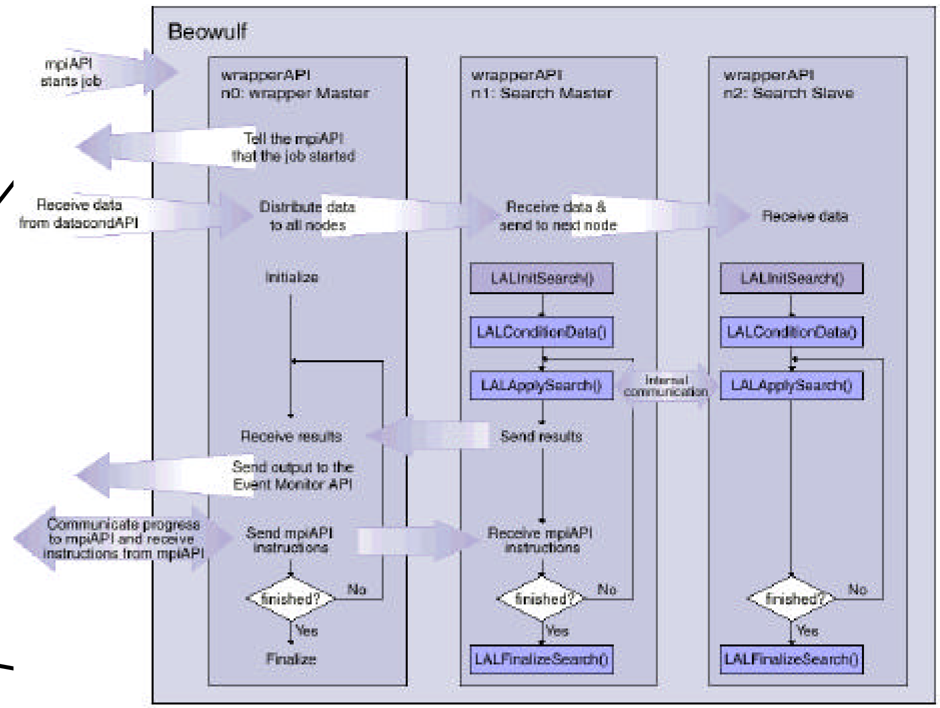
Identify Unit Modules



Isolated Pipeline Code from Science Code!



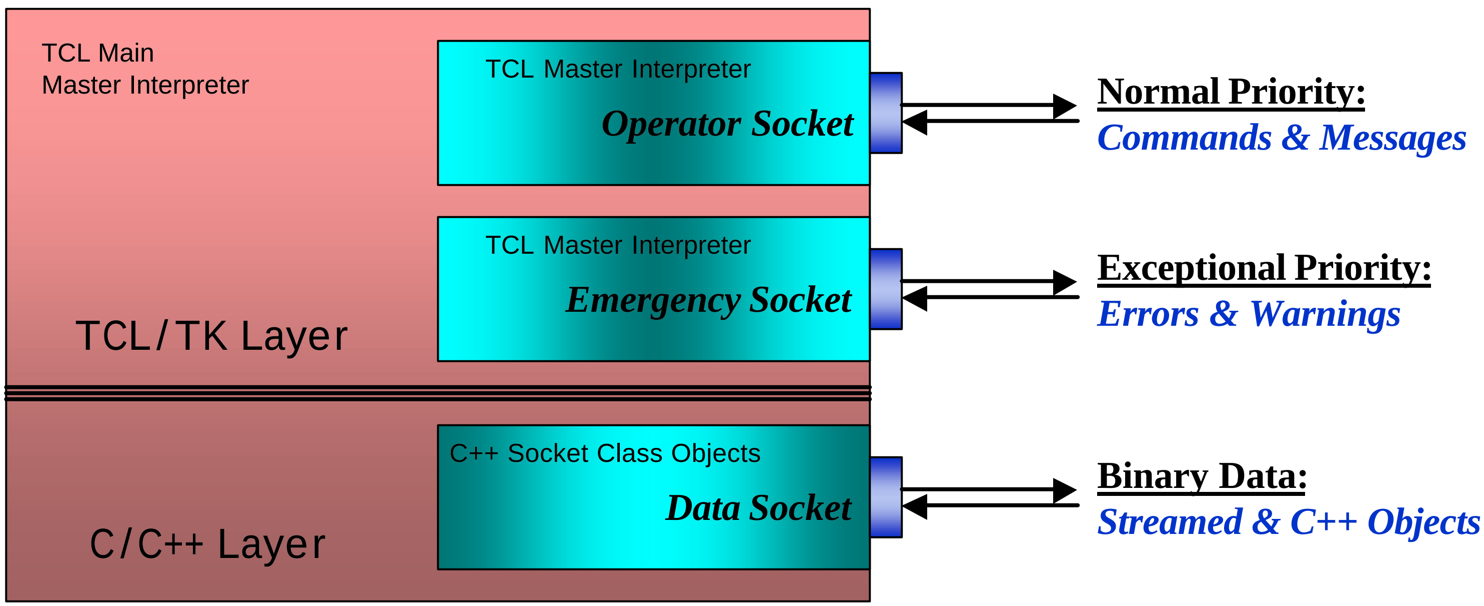
Search Code Interface:



Search codes dynamically loaded at runtime!

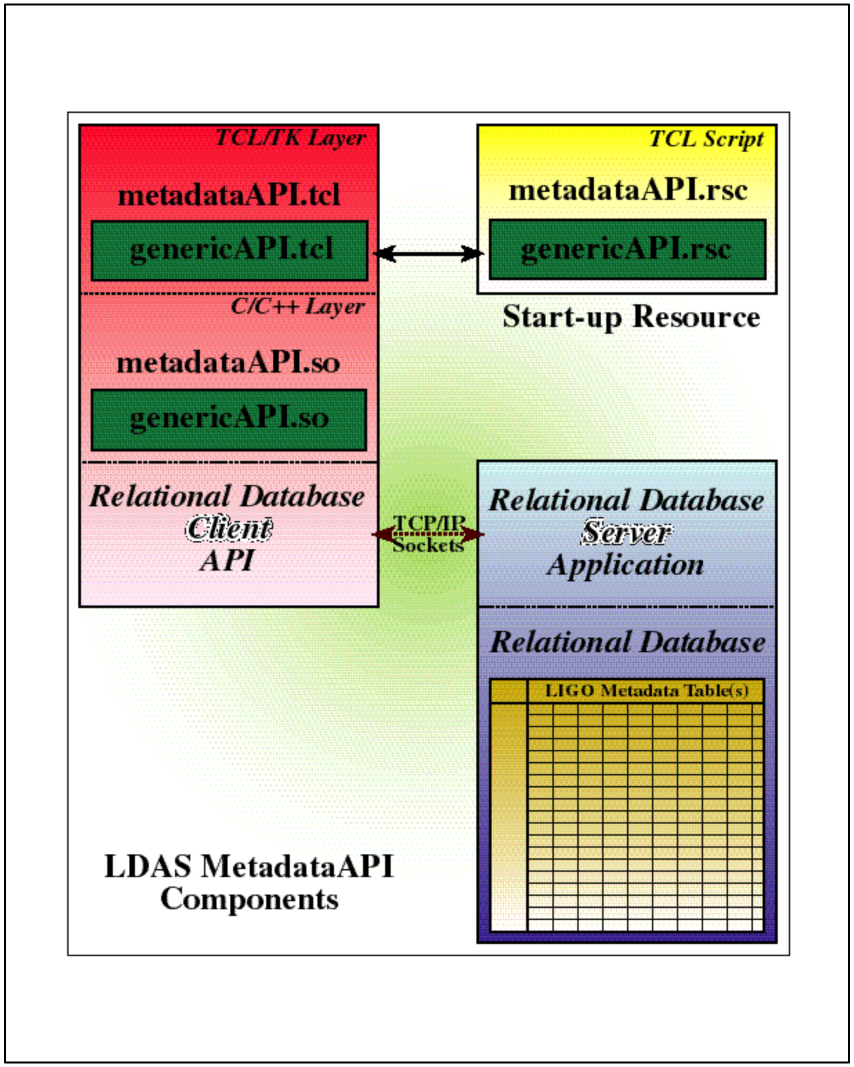
“Build vs. Buy vs. Borrow” for Distributed Interfaces

- ***Used a commercial C++ OOP socket library from ObjectSpace.***
  - *Negotiate a 95% discount and got the source code and free runtime license!*
    - *Risks of using commercial product without source code too great…*
  - *Company dropped this product line one year later!*
    - *Able to continue using product thanks to having all the source code!*



"Build vs. Buy vs. Borrow" for Database

- *Held a two day database workshop in October of 1998 at Caltech.*
  - *Invited Database experts from several disciplines.*
  - *Presented LDAS requirements.*
  - *Debated pros and cons of …*
    - *Relational Databases.*
    - *Object Oriented Databases.*
    - *Commercial Databases.*
    - *Public Domain Databases.*
  - *After workshop, LDAS settled on commercial relational database.*
  - *Were able to negotiate <u>free</u> license for LIGO from IBM for DB2.*
  - *<u>Built</u> the client software using ODBC compliant standards.*



Include Security in Design

- *LDAS system designed to run on a private network.*
  - *One gateway to the internet allowing remote access with strictly control services from trusted hosts.*
  - *Communicate with data acquisition sub-system through a read-only file system service to accessing the data.*
- *Only user accounts on the system are those needed to operate the LDAS software system and database.*
- *Users make encrypted requests for data analysis on a strictly controlled port using unique identifiers for each analyst (a kind of user name and password without Unix privileges).*
- *Includes ability to integrate emerging technologies such as GRID computing security tools in future.*

Formal Review of Preliminary Design

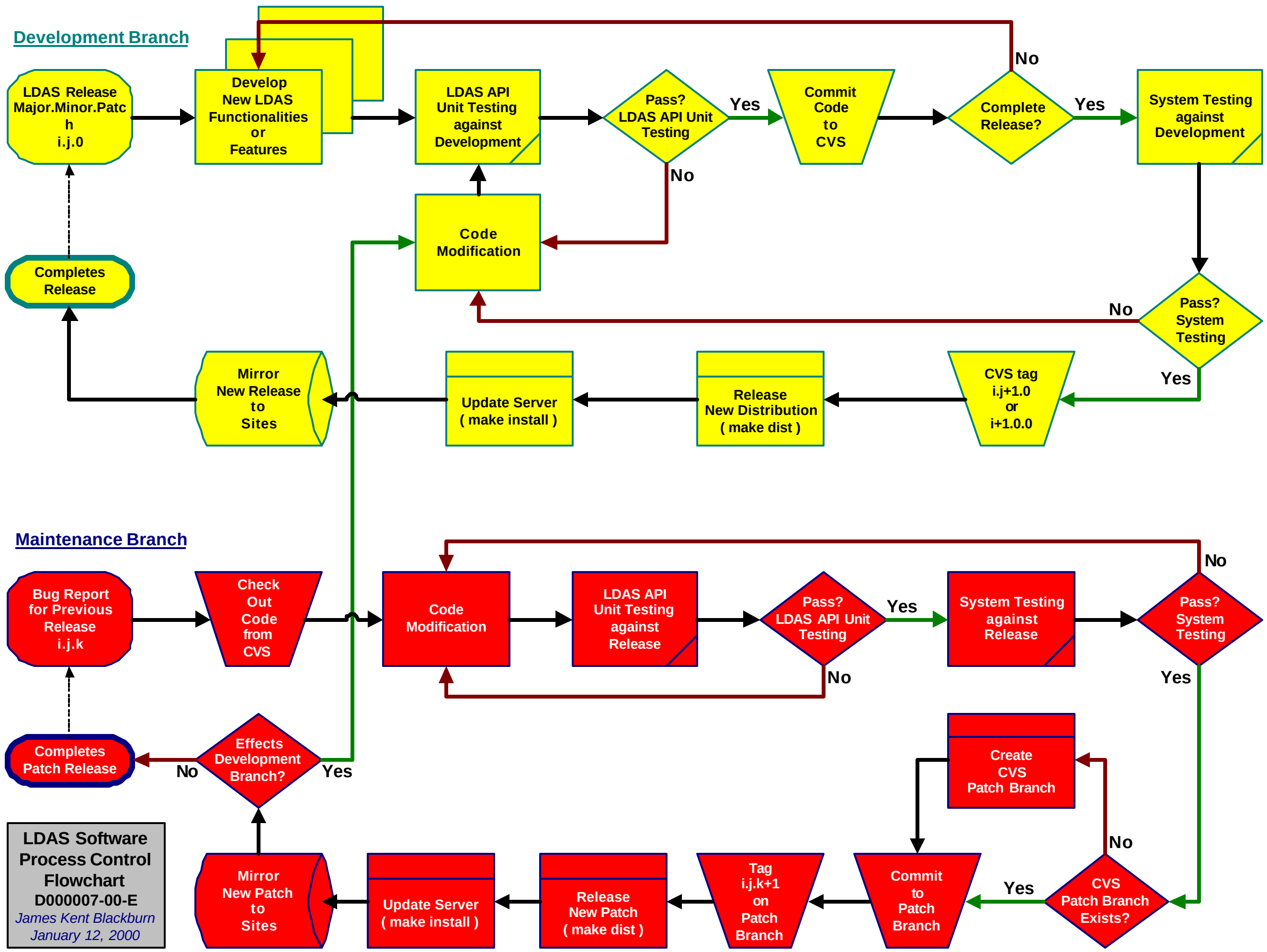
- *The Preliminary Design Review for LDAS was held in March of 1999, 15 months after Conceptual Design Review.*
- *Review committee consisted of both internal and external members (LIGO Project & LSC).*
- *Preliminary Design Documents submitted to LIGO Document Control.*
- *Review revealed a new and growing community of interested users of LDAS – the LSC.*
  - *Increased core community of stakeholders by factor of four!*

Software Project Case Study:

The LIGO Data Analysis System

Software Construction

Define Flow for Software Development and Testing



Bulk of software developers hired had no experience in formal software development.

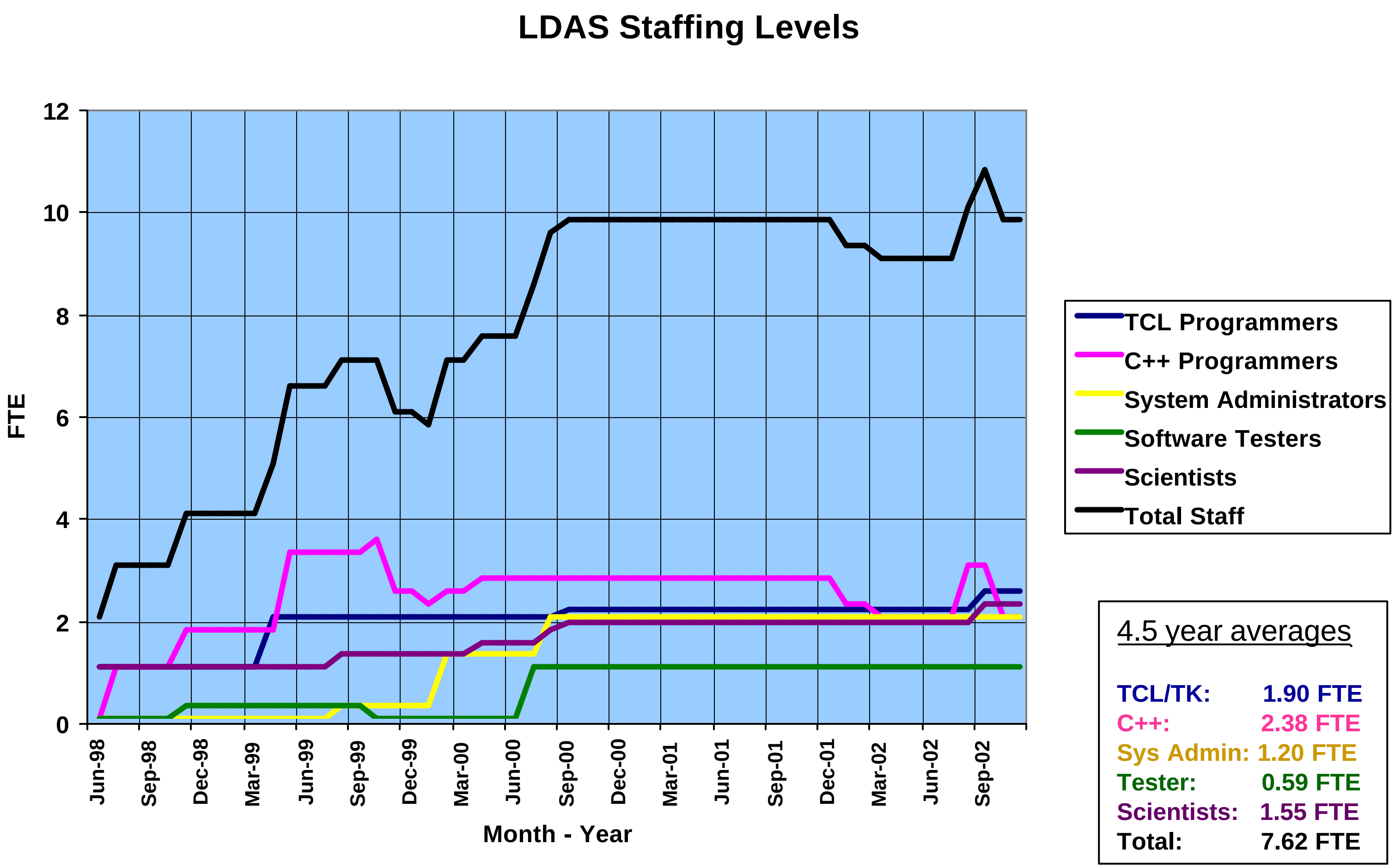
Staffing & Schedule Estimation

- *Estimated based on hiring of 2-3 TCL/TK developers & 3-4 C++ developers.*
- *Used personal <u>experience</u> and <u>complexity</u> of each software module:*
- *Multiplied time I estimated I would need to do coding by 3 …*
  - *Estimated 3 months of coding for each of the 8 C++ libraries.*
    - *~ 2 man-year of development schedule*
  - *Estimated 6 months with two developer for each of 12 "API" module.*
    - *Teamed up one C++ and one TCL programmer per API.*
    - *~ 12 man-years*
  - *Estimated one month of user interface development per "API" module.*
    - *TCL/TK development*
    - *~ 1 man year*
  - *Estimated 2 releases per year*
    - *3 weeks per release involving all 6 developers and 1 tester*
    - *~ 1 man-year*
  - *Background system testing would require a software tester for 2 years*
- *Estimated totals: 18 man-years & 3 to 4 years to complete.*
- *Eventually adopted Microsoft Project to "capture" schedule as snapshots.*

Staffing for Construction

- *All Design and Prototyping done by staff scientist.*
- *When the time came to "bend metal" we hired a single TCL Script developer and a single C++ developer.*
  - *TCL Script developer was very experienced with PERL scripting language but had little TCL/TK experience.*
    - *Sent developer to TCL/TK school – very successful experience!*
  - *C++ developer fresh out of Caltech with BS in Physics.*
    - *Had previous mentoring exposure with this developer during a Summer Undergraduate Research Fellowship involving significant C++ code development for LIGO noise models.*
- *Programming staff did their own system administration for first year.*
  - *Not recommended!*
- *Complete staffing needs required an additional 2 years!*
  - *Dot.com era significantly impacted ability to find and keep people.*

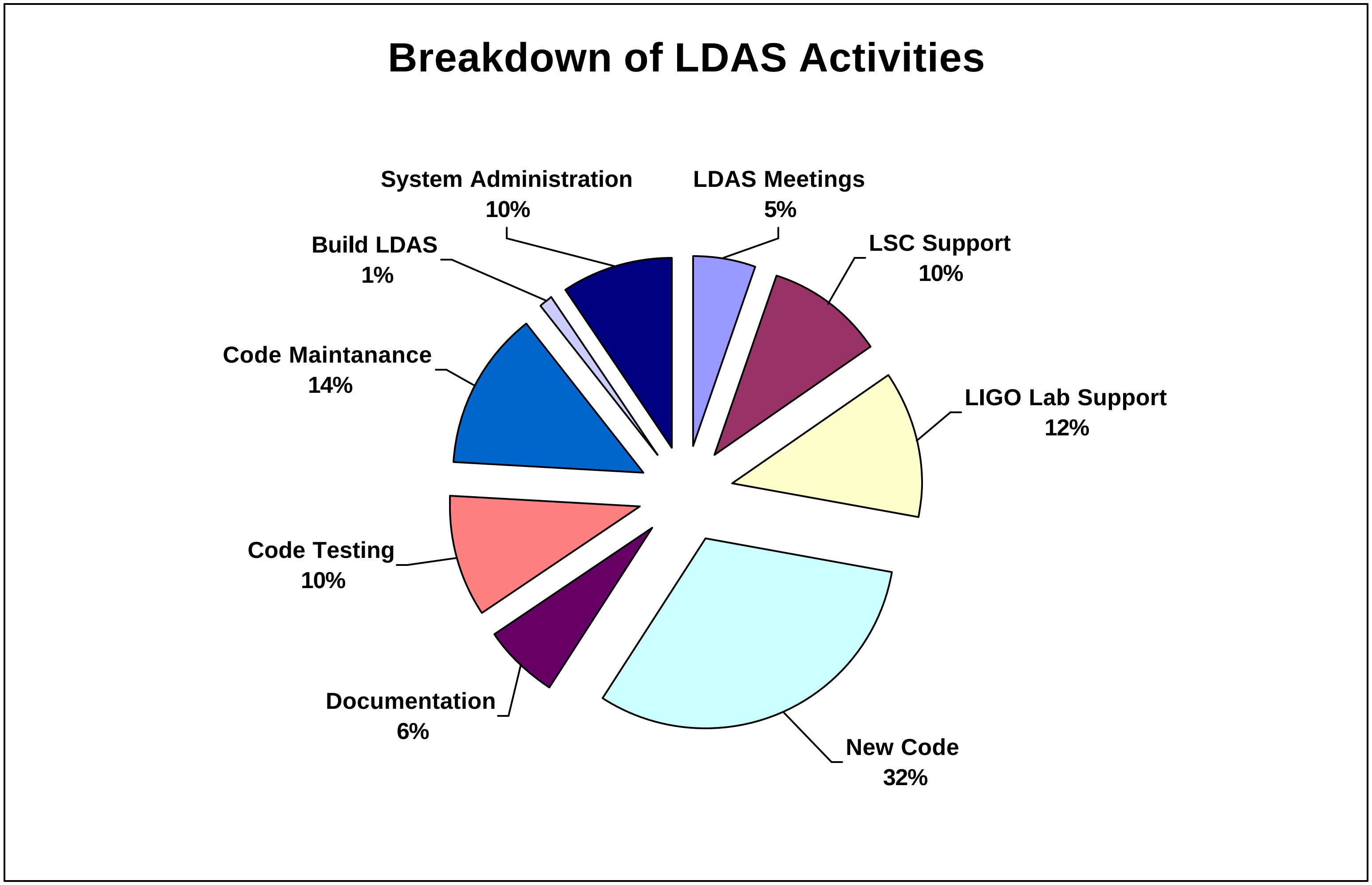
Staffing Timeline



4.5 year averages

TCL/TK: 1.90 FTE
C++: 2.38 FTE
Sys Admin: 1.20 FTE
Tester: 0.59 FTE
Scientists: 1.55 FTE
Total: 7.62 FTE

What Developers Did



Statistics accumulated weekly from Summer 2000 through Summer 2001.

Interaction Between Developers and User Community During Construction

- *Established milestones (alpha releases, beta releases, final releases) and advertise them to both developers and users.*
  - *Include description of emerging functionality with each milestone.*
- *Included a "change history" with released versions.*
- *Included a list of "open problem" with releases.*
- *Provided web based user documentation and examples.*
- *Established a regular software developers meeting – twice weekly works well for us.*
- *Established a regular software users group teleconference – twice weekly works well for us.*

Risk Management through Tools

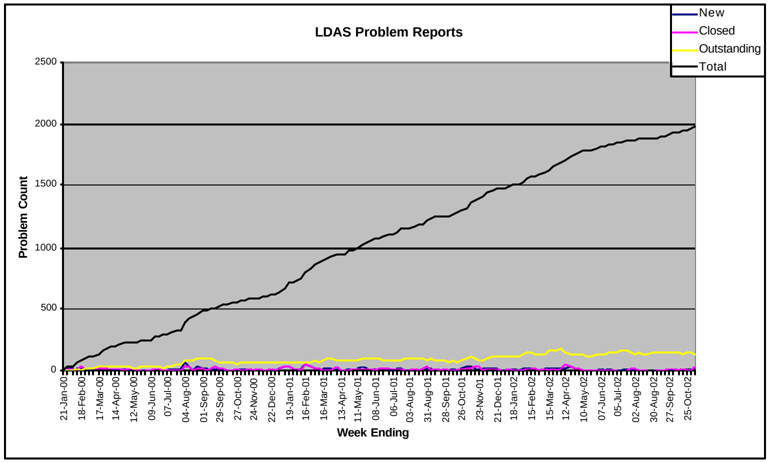
- ***Software risk management usually handled in one of two ways:***
  1) *Assign/hire a risk management officer: Part Chicken Little, part Eeyore - "the sky is falling", … "it'll never work".*
     - *Responsible for creating risk management plan and carrying it out.*
     - *Manage a "top 10" lists of risks impacting project.*
       – *Schedule slip, requirements creep, developer gold-plating, low quality software, unachievable schedule, unstable development environment, high turnover, customer-developer friction, work environment, …*
  2) *Integrate an automated "defect tracking tool" into project culture.*
     - *Relies on risks coupling to defects (not the complete picture but works).*
     - *Allows automated assignment of risks, publishing and prioritizing of lists, tracking of steps taken to reduce risk and by whom, …*
     - *Distributes work of risk management across project team.*
     - *Eliminates alienation among staff.*
- ***LDAS adopted the defect tracking tool method while informally addressing duties of a risk officer with scientific staff.***

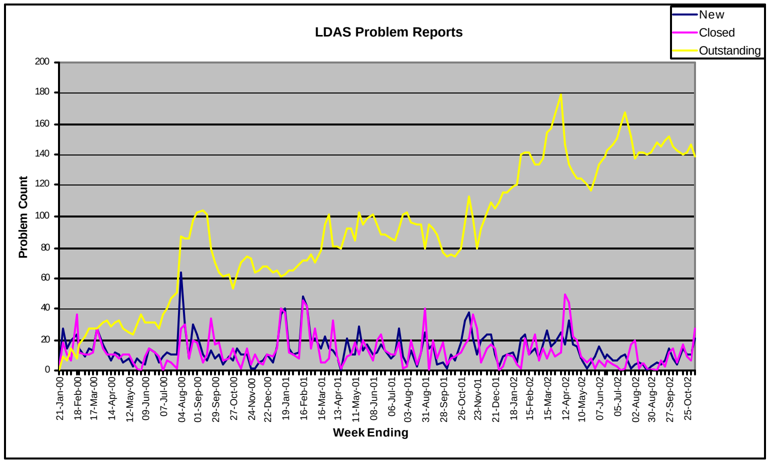
Risk Management through Metrics

- *Predictive power of software metrics only as good as the input data … "garbage in – garbage out"!*
- *Popularity of "5-10 lines of code per day" hides progress towards unit and system level functionality and nonlinear complexities of project.*
  - *Excludes functionality per line associated with computer language.*
  - *LDAS adopted a kilobytes per unit time metric using CVS.*
    - *Trends in code per unit time found to be more useful metric for risk management.*
- *Software defect tracking extremely useful for making sure bugs do not get forgotten or neglected.*
  - *Requires cultural commitment from users and developers to follow formal procedures as opposed to simply picking up the phone or sending email.*
  - *LDAS adopted GNU's web based GNATS package.*
    - *Status and summaries available to any web browser - easy to use.*

Problem (Defect) Tracking

category	open	analyzed	suspended	feedback	closed	Total
build process	3	0	0	0	97	**100**
controlMonitorAPI	0	0	0	0	79	**79**
data archive	0	0	0	0	1	**1**
dataConditionAPI	20	2	5	2	299	**328**
dataIngestion	1	0	0	0	0	**1**
dbaccess	0	0	0	0	4	**4**
diskCacheAPI	4	0	0	1	18	**23**
distribution	1	0	0	0	6	**7**
documentation	18	1	0	3	130	**152**
eventMonitorAPI	1	0	0	0	27	**28**
frameAPI	23	0	0	2	179	**204**
frameCPP	6	0	0	1	19	**26**
general library	1	0	0	0	1	**2**
genericAPI	3	1	0	0	115	**119**
GUILD	0	0	0	0	5	**5**
ilwd	3	1	0	1	31	**36**
ilwdfcs	1	0	0	1	2	**4**
integration tests	1	0	0	0	5	**6**
lightWeightAPI	1	0	0	0	56	**57**
managerAPI	15	1	1	0	130	**147**
MDC ready	0	0	0	0	3	**3**
MDC run	0	0	0	0	0	**0**
metaDataAPi	2	2	0	4	88	**96**
mime	0	0	0	0	0	**0**
mpiAPI	4	0	0	1	57	**62**
remoteAPI	0	0	0	0	0	**0**
SWIG	0	0	0	0	1	**1**
sys admin	0	2	0	0	219	**221**
userAPI	1	0	0	0	8	**9**
wrapperAPI	1	0	0	0	20	**21**
Totals:	**110**	**10**	**6**	**16**	**1600**	**1742**



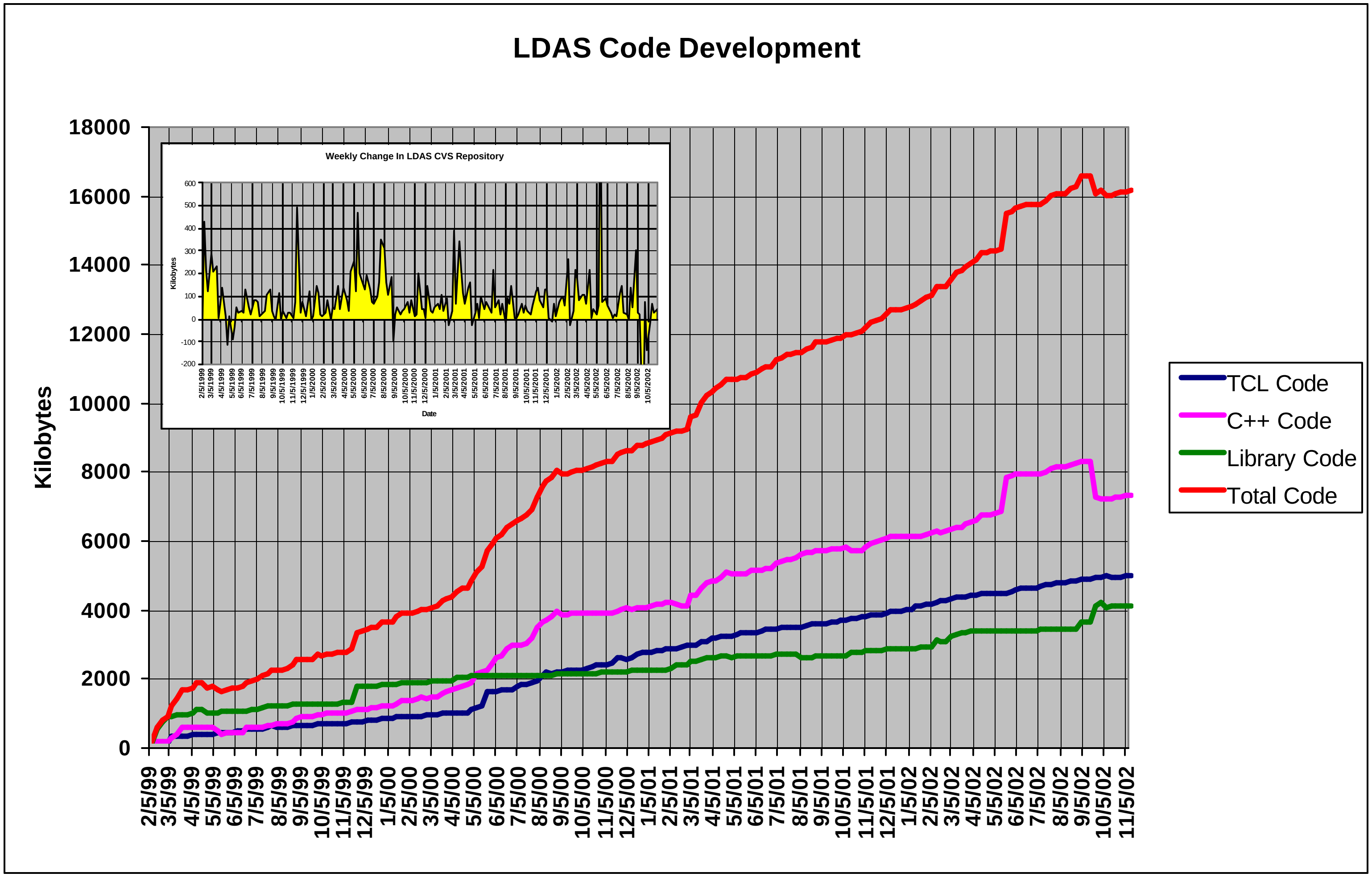


92% *However, trends reveal an additional "metric" of progress!*

Source Code Configuration Control

- *Source code is the deliverable for a Software Project.*
- *Preservation of source code is central to longevity, efficiency and risk management.*
  - *Source code is stored on media which is known to failure.*
- *Source code evolves on an hourly time scale!*
  - *As soon as more than one developer has access to the same source code the possibility for conflicting evolution exists.*
- *Source Code Configuration attempts to address these issues.*
  - *LDAS adopted the public domain package* ***CVS – Concurrent Version Control****.*
    - *Centralizes the "vault" used to store the source code.*
    - *Protects against conflicting evolution by multiple developers.*
    - *Stores every "frame" in the source code development "movie".*
  - *LDAS system administration does nightly backups of centralized "vault".*
  - *CVS support multiple development "branches" – NOT adopted by LDAS.*

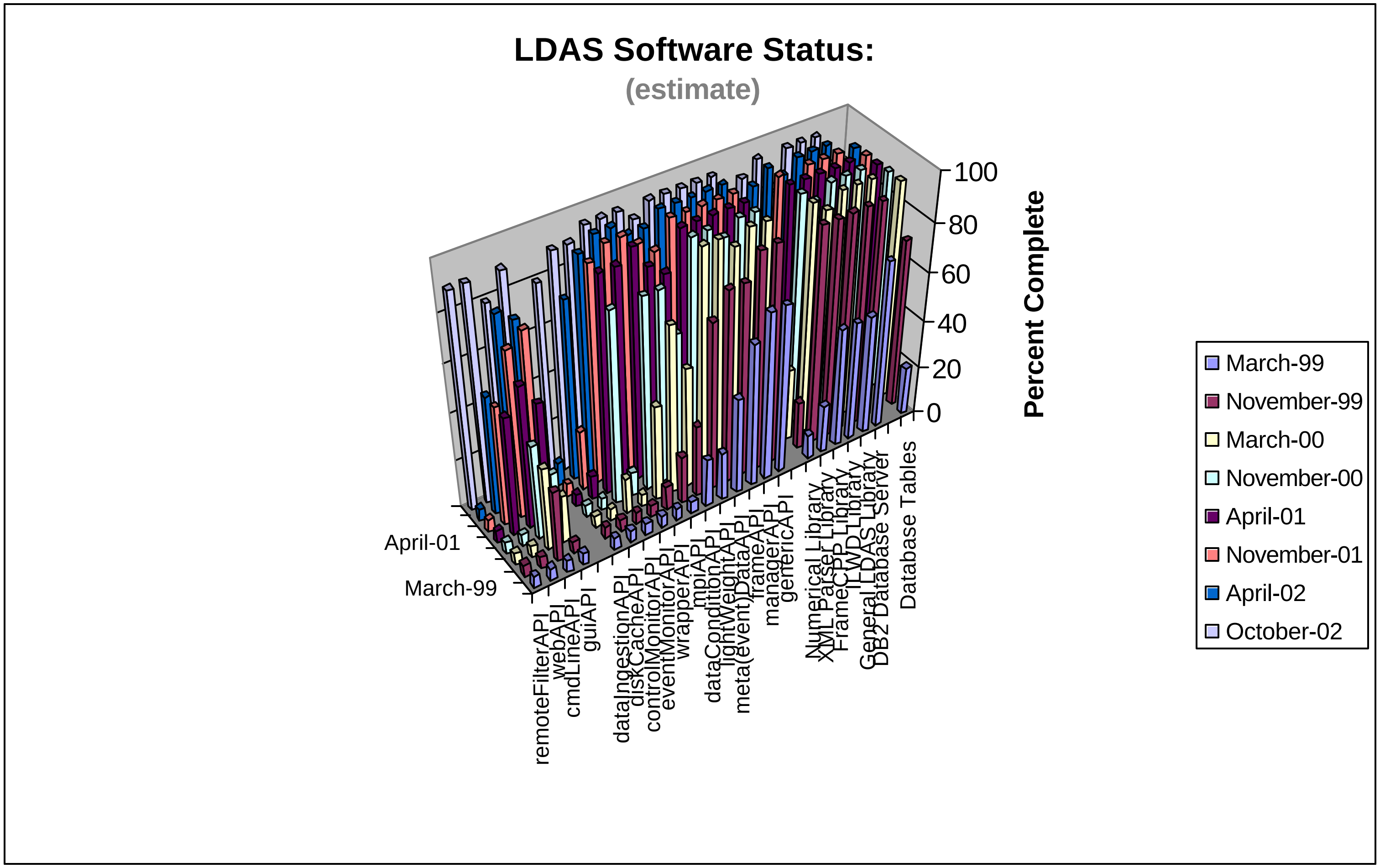
Tracking Code Development with CVS



Here's the kilobytes of source code metric (not lines of code).

Tracking Progress

(6 month snapshots)



Each component tracked using Excel.

LIGO

Unit Testing

- *Primarily applies to C++ software development.*
- *Each "quantum" of functionality has standalone test program developed in parallel to test unit functionality in a controlled way.*
  - *Essential to include exception handing as part of controlled test.*
  - *Less than 25% of functional "quanta" have been unit test enabled.*
    - *Estimated using commercial package (Insure++).*
    - *Better than average according to literature (5-10% typical)!*
    - *Couples to defect metrics.*
  - *Software developers view it as a tax (much like documentation).*
- *All unit tests are added to a "make-check" target in the Makefiles used to build software, guaranteeing systematic evaluation.*
- *Unit tests are very effective at stabilizing code, reducing risks, and absolutely essential for migration to new platforms.*

System Testing

- *System testing measures software interface and integration defects in the operational software's environment.*
- *A dedicated LDAS staff (software tester) used for system testing.*
- *Tester conducts system testing throughout development and deployment phases of software project.*
- *All testing scripts are captured in CVS with project's source code.*
  - *Estimate that better than 75% of interfaces are being tested.*
    - *Having a dedicated tester largely responsible for relative completeness!*
- *Some set of system tests capable of running indefinitely are needed.*
  - *Generates metrics for memory management issues (memory leaks).*
  - *Generates metrics for multi-threaded processing issues.*
- *Many performance metrics are integrated directly into the operations of a running LDAS system.*

Software Quality Assurance

- *Quality Assurance procedures must be applied routinely.*
  - *Nightly*
    - *Exercise a set of "looping stress tests" which continuously issue data-pipeline requests to the development and test systems for each astrophysical search code (approximately 200 – 500 jobs per hour).*
    - *Concurrently issue a set of metadata queries on the database (200 – 500 jobs per hour).*
    - *Concurrently issue request to produce a reduced data set frame from full size raw frames (currently once every 100 seconds).*
    - *Establishes a one in few thousand reliability metric.*
  - *Weekly*
    - *Repeat complete release suite of system tests at least once.*
      - *Brackets time frame when software bugs were introduced.*
  - *Once per LDAS software release*
    - *Complete full suite of systems tests every two days as minor bugs are identified and fixed in preparation for release.*
    - *Establishes a one in a few tens of thousands reliability metric.*
  - *Have plans to introduce Monthly QA procedures.*

Nightly Builds of LDAS

- *Use a custom script in a Unix "cron" job to automate nightly builds:*
  - *Checks out current code from CVS.*
  - *Configures build instructions (to optimize or not, etc).*
  - *Compiles and links code.*
  - *Runs all integrated unit tests and assign pass/fail.*
  - *Email results to responsible staff.*
- *Carried out each night for all supported operating systems.*
  - *Redhat Linux on Intel & Sun Solaris on UltraSparc for us.*
- *Complete nightly build procedure takes over 6 hours!*
  - *Hence the importance of automating during the night.*
  - *Gives results each morning, reduces turnaround time during work hours.*
  - *Reduces cost and risk for free!*

Mock Data Challenges
Insightful and Risk Reducing

- *LIGO Management decide to forgo a formal "Final Design Review" in favor of a series of formal "Mock Data Challenges" of the LDAS and LAL software as it became sufficiently mature.*
- *Progressively more complex Mock Data Challenges were integrated into the LDAS and LSC's LAL (LIGO/LSC Algorithm Library) development schedules.*
  - *MDC-1: August 2000, LDAS's data conditioning (**ldas-0_0_10**)*
  - *MDC-2: December 2000, LDAS's parallel computing (MPI) (**ldas-0_0_12**)*
  - *MDC-3: January 2001, LDAS's database (**ldas-0_0_12 - ldas-0_0_15**)*
  - *MDC-4: May 2001, LDAS using inspiral search codes (**ldas-0_0_17**)*
  - *MDC-5: September 2001, LDAS using burst/stochastic search codes (**ldas-0_0_20**)*
  - *MDC-6: November 2001, LDAS using periodic pulsar search codes (**ldas-0_0_22**)*
- *These were a rather distractive to our development schedule, but at the same time, greatly improved everyone's software!*
- *LDAS was only at its "alpha" development during each of these MDCs.*
  - *Early enough that significant changes to design could be worked accommodated.*
  - *Early enough that it impacted ability to keep momentum toward detailed design.*

Outcome of MDCs

- *Identified need for top-level oversight to assist in coordinating the distinctly different software development needs of the LIGO Laboratory (LDAS, DMT, CDS) and LIGO Scientific Collaboration (LAL & search codes).*
  - *Formation of a "LSC Software Coordination Committee".*
  - *Formation of a "LSC Software Change Control Board".*
  - *Organized weekly LIGO/LSC Software Users Group meetings.*
  - *Formalized a "LSC Computer Committee" to address long term LIGO Data Analysis Issues.*
- *This greatly eased the tension associated with the rapid expansion of users and developers added during the Mock Data Challenges; Removed risks associated with previous chaotic conditions.*

Release Preparation

- *Establish with software team a deadline for current development tasks – at least 4 weeks in advance.*
  - *Should reflect schedule & be shared with user community (LSC, GRID).*
  - *Must track progress towards this deadline in weekly software meetings.*
- *When development tasks complete, "freeze" write privileges to CVS repository for all but the "librarian".*
- *Begin cycle of full unit and system testing on a tight schedule (~1 to 2 days).*
- *Identify critical bugs that must be supported for the release.*
  - *All bug fixes must be fully tested on the development system and have a "code read".*
  - *Document all bugs not fixed for release in the Problem Tracking System.*
- *Once critical bugs are resolved, push pre-release software system onto other non-developmental systems for portability testing.*
- *Final software step is to tag the CVS repository with the official release tag.*
  - *Rebuild the official tagged code base and distribute to all systems and customers.*

Distribution Control

- *Large number of LDAS Systems in existence around the world!*
  - *4 development standalone computers running LDAS at Caltech.*
  - *1 full scale LDAS Integration Development System.*
    - *Each of the nightly builds stored for one week.*
  - *1 full scale LDAS Test System.*
  - *4 full scale LDAS Operations Systems within LIGO Laboratory.*
  - *4 full scale LDAS Systems operating outside of LIGO Laboratory.*
- *Single image-server at Caltech distributes full distribution of LDAS and all infrastructure software through automated mirroring technology to all Operations Facilities.*
  - *Provides rapid recovery in the event of a disaster.*
  - *Guarantees that the exact same version of software present everywhere.*
  - *Also provide source code distributions for all others interested once registered.*
    - *Useful for developers not located at Caltech.*

Software Change Control

- *Software change is driven by many factors*
  - *Evolution of hardware.*
  - *Evolution of operating systems.*
  - *Evolution of compilers.*
  - *Evolution of third party software packages you've become dependent on through the "buy verses build verses borrow" decision process.*
  - *Emerging technologies (e.g., Grid computing).*
  - *Evolving (or overlooked) usage model for software.*
    - *Motivated by user community once they have hands on experience.*
- *Non-linear combination of any or all of the above typical.*
- *In long lifetime software packages these will challenge the stability of your code in a matter of months!*

Change Control Procedure

- *Suggestions which many lead to a change requests are vetted in "LIGO/LSC Software Users Group".*
- *Mature requests for changes submitted to the "LIGO/LSC Software Committee".*
  - *Scientific merit and cost of change evaluated.*
  - *Expert opinions brought in from the "LIGO/LSC Software Change Control Board"*
  - *Approved changes are passed on as new requirements to development team, often with newly identified resources to facilitate implementation if necessary.*
- *Procedure very inclusive of collaboration – works well.*

Software Project Case Study:

The LIGO Data Analysis System

Operations and Maintenance

Operations and Development During Engineering Runs

- ***Engineering Runs used "alpha" and "beta" versions of LDAS***
  - *Only a subset of software modules were mature enough to operate.*
  - *The insights gained into the "usage model" was tremendously insightful!*
    - *E1: (ldas-0_0_10) supported ingestion of triggers from external software.*
    - *E2: (ldas-0_0_12) ditto and archiving of frame data to tape.*
    - *E3: (ldas-0_0_15) ditto and some simple signal processing.*
    - *E4: (ldas-0_0_16) ditto*
    - *E5: (ldas-0_0_19) ditto*
    - *E6: (ldas-0_0_22) ditto and one astrophysical (burst) search data-pipeline.*
    - *E7: (ldas-0_0_23) ditto and all astrophysical searches in data-pipeline.*
    - *E8: (ldas-0_2_0) first use of a beta version of LDAS*
  - *LDAS Operations during E7 collected and analyzed data for 17 days.*
    - *1 in 40 user requests failed due to software bugs.*
    - *Prior testing suggested this would be closer to 3 in 1000.*
    - *As a result, LDAS was instrumented with enhanced run-time diagnostics to trace origins of these differences.*

Operations and Development During First Science Runs

- ***First Science Run used a "beta" versions of LDAS***
  - *All software modules are mature enough to operate – just not complete.*
  - *Insights gained into the "usage" was tremendously valuable!*
    - ***S1**: (**ldas-0_4_0**) supported all types of astrophysical searches.*
      - *Less than 1 in 10,000 jobs failed from bugs in LDAS – consistent with testing.*
  - *The S2 Science Run will also operate with a "beta" version of LDAS.*
    - *Must be able to operate eight weeks, continuous, and around the clock.*
- *The first completed version of LDAS (1.0.0) is expected in time for the S3 Science Run.*
  - *Will operate more efficiently and with much larger Beowulf Clusters.*
  - *Will support even longer operations.*
  - *Remaining development will address reliability and performance.*

Software Maintenance

- *"A successful software project is never more than ~90% complete" – Kent Blackburn.*
  - *Compilers and third party packages that are integral to your software project are constantly evolving and for better or worse you are coupled to this evolution.*
  - *Successful project will last through many generations of new hardware and new operating systems – Moore's Law!*
  - *Bugs and features typically surface for years – can be much more costly to fix compared to the early phases of project.*

Software Project Case Study:

The LIGO Data Analysis System

Summary &
Lessons Learned

Summary

- *LDAS Software development nearly completed – around 90%!*
- *Software successfully deployed and operated during LIGO Engineering Runs and First Science Run.*
  - *Early indications are that it will succeed (beat the less than 1 in 4 odds).*
- *Nearly 22 man-years of software construction have gone into LDAS.*
  - *Expect to complete original design goals in 6 to 9 months (2 - 3 more man-years).*
  - *Original planning called out 18 man-years to complete.*
  - *About a 20% cost and schedule overrun in unusual cultural complexity.*
- *Development expected to continue beyond original design requirements in support for GRID Computing (GriPhyN & iVDGL) and other LSC motivated requirements creep.*

Top Ten Lessons Learned

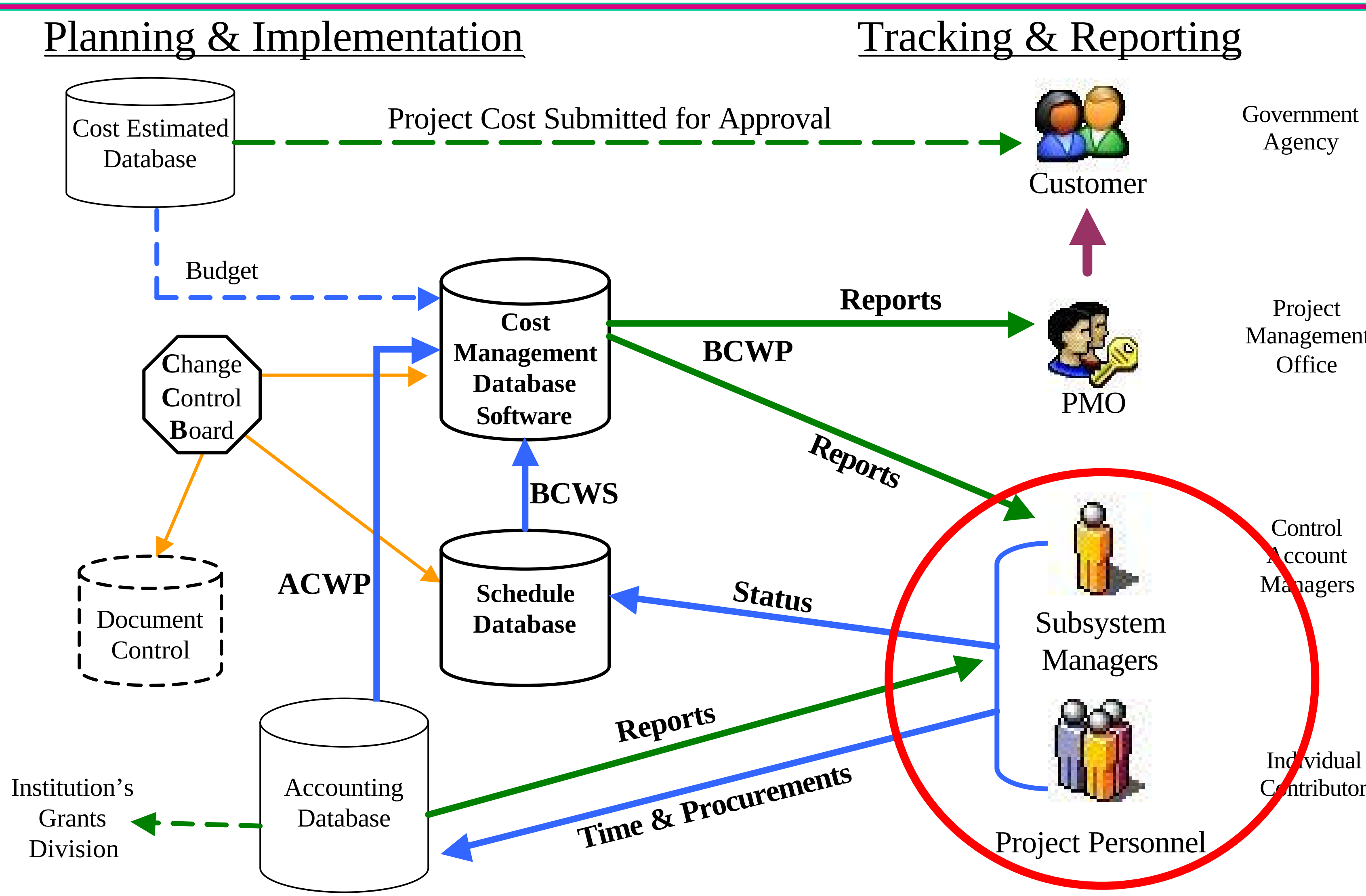
- *Start software projects as soon as possible.*
  - *Costs and time to delivery now exceed comparable hardware projects.*
- *Hiring software developers is time consuming process.*
  - *Start interview process early – very competitive market place.*
  - *Consider training for those with "right fit" but lacking expertise.*
- *Know your user community from the start.*
- *Keep software manager close to the day-to-day business of development – active risk management.*
- *Bring a systems integration tester on board as soon software integration becomes important to risk management.*

Top Ten Lessons Learned

- *Use a defect tracking tool to support risk management but not to define risk management.*
- *Use the world wide web to disseminate any and all information, documentation, system health and status, etc.*
- *Integrate self diagnostics into the nominal operations of the system.*
  - *Include a control and monitor function in design.*
- *Make sure adequate system administration and hardware support are available to software developers.*
- *Be ready for success and cost of staying current!*

Richard Fischer
Project Management Services, Inc.

Consider Using A Bigger Software Circle - Fold Software Projects into PMCS!



The End