Handling Problems of Aging Software and Unsupportable Hardware – How we did it.

By Eric Shulman
Space & Missile Systems Center
Los Angeles AFB, Calif

Alan D. Matt
Aerospace Corp

Wilkie Haw
Aerospace Corp

10 March 2003

The Problem – A Maintenance Headache

- The Jovial Programming Language was widely used by the Air Force in the 1960s and 1970s.
- A number of these systems are still in operation
- Jovial was developed in 1959-60
- Jovial is similar to ALGOL (Also ancient)
- Inertial Measurement Software (IMS) for the Atlas Centaur and Titan IV *are* written in the Jovial Programming Language.
- CPU used for IMS is 1750A with PACE Chip Set (circa 1980, very ancient)
- This Jovial Based IMS Code is still in use today

Rocket

Development

Cartoon

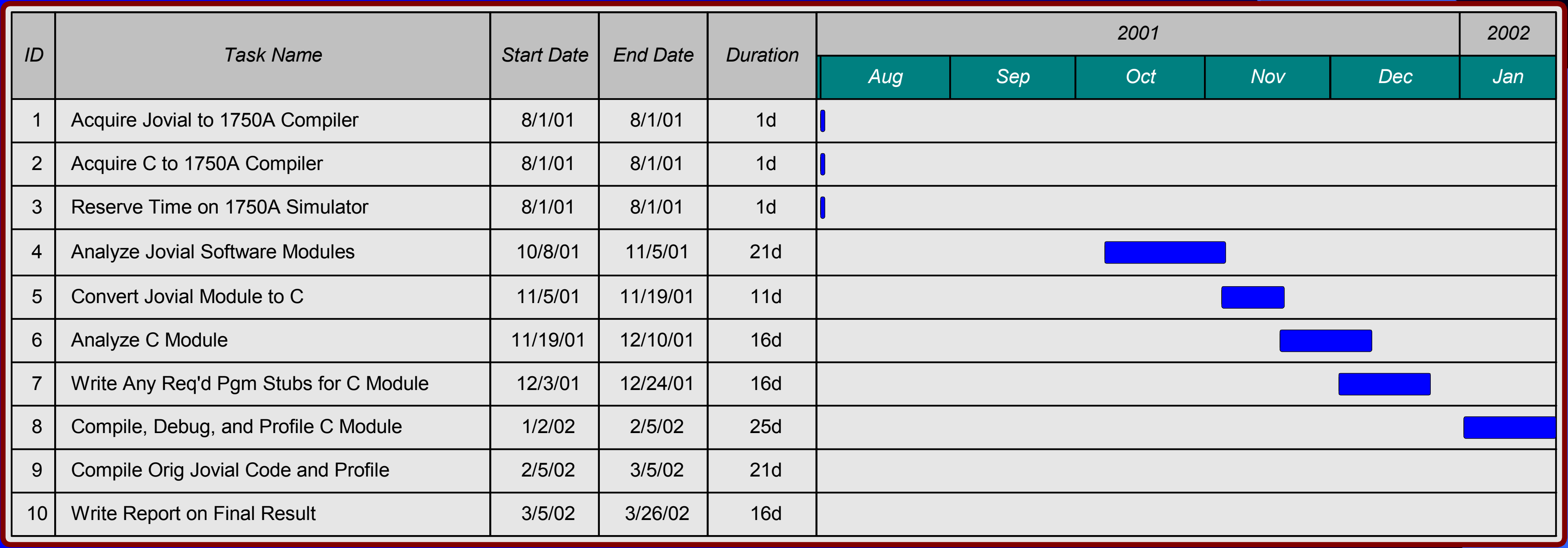
Project Goals

Our goal was to convert a tightly coupled Jovial J73 software module (with bit manipulation and COMPOOLs) into C code using the Embedded Information Systems Re-Engineering (EISR) Conversion Tool. This should raise the difficulty level to duplicate a more typical case scenario.

The Schedule

- The project was initially projected to have a schedule between 3 to 6 months involving 2 full-time people (Probably closer to 6 months) – 1 MTE.
- The project was task driven and schedule slips were planned to occur as operational programs forced us to time-share the Atlas Centaur simulator.
- The project actually involved 3 part-time personnel working from 3 December 2001 through completion on 27 June 2002 (We only used .6 MTE).
- 1 MTE = 1810 Hours (STE used in presentation, it is fully burdened).

Initial Gantt Chart



ID	Task Name	Start Date	End Date	Duration
1	Acquire Jovial to 1750A Compiler	8/1/01	8/1/01	1d
2	Acquire C to 1750A Compiler	8/1/01	8/1/01	1d
3	Reserve Time on 1750A Simulator	8/1/01	8/1/01	1d
4	Analyze Jovial Software Modules	10/8/01	11/5/01	21d
5	Convert Jovial Module to C	11/5/01	11/19/01	11d
6	Analyze C Module	11/19/01	12/10/01	16d
7	Write Any Req'd Pgm Stubs for C Module	12/3/01	12/24/01	16d
8	Compile, Debug, and Profile C Module	1/2/02	2/5/02	25d
9	Compile Orig Jovial Code and Profile	2/5/02	3/5/02	21d
10	Write Report on Final Result	3/5/02	3/26/02	16d

The Project

- This project was to evaluate the ability of the Embedded Information Systems Re-Engineering (EISR) Conversion Tool Version 1.0 (owned by the USAF) to convert a portion of an AF system still using the Jovial Programming Language to a newer programming language (in this case C). Cost, schedule, and risk were to be addressed.
- We spent 3 weeks locating the best conversion candidate
- This was a proof of concept project only, the re-engineered software was not planned to be deployed.
- The project evaluated potential Hardware rehosting options.
- Please note that the project was a success despite the number of problems encountered.

The Project Cont'd

- Estimated cost to completely rewrite: About 3 persons for 1 year or 3 person years (1 Person Year = 1810 hrs) or a total of 5430 hours.

- The actual conversion of the Jovial to C Code took 1082 hours or .6 person years and was comprised of the following tasks:
  - Time Spent in preparation to Final Compile
    - Analyzing C code, Converting the code, performing prelim compiles to identify the more serious problems
  - Time spent in final C Compile
  - Time spent is Assembling code
  - Time spent in Linking
  - Time spent in Open Loop Testing
  - Time spent in Closed Loop Testing

Project Scope

- Chosen Target - The Inertial Measurement Software part of the Atlas Centaur Launch Vehicle
- Using Embedded Information Systems Re-Engineering (EISR) Conversion Tool Version 1.0 (owned by the USAF)
- Use existing 1750A based hardware platform for first part of effort
- Re-engineer the existing Jovial J73/Assembly code to C/Assembly
- Six Degrees of Freedom (6DOF) simulator will be used instead of launch vehicle (Delta from actual vehicle < .006%).

So What Do We Do First

C-17

Landing

On Carrier

Before You Can Even Start We Needed To:

- Document exactly what you are trying to do
- Create GANT Chart(s) that detail all of the tasks and proposed completion dates.
- Review available funding for both equipment and people versus time and tasking
- Determine where the project will be located? Are there any security issues? There were.
- Determine people power loading chart
  - What type of people are required
  - Are the required people available
  - Do the people have appropriate clearances
  - What tasks would be assigned to which people
  - **Do the people get along (Could be problems if they don't)**
  - Available STE (Burdened Staff Hours) since using FFRDC

The Development Team You Don’t Want This

Picture of

Catfight

Do Hardware or Software First

- You do not want to re-engineer both the hardware and the software at the same time – too many unknowns
- We did the software first because it was the main focus of our effort and was the correct first step in re-engineering the system.
- If we had re-engineered the hardware first, unless we used hardware emulation, we would have had to re-engineer the software at the same time – a problem.

Re-Engineering the Software Step 1 – Research

- Need to evaluate:
  - Available Compilers for target
    - Experience, references, pick best value
  - Evaluate EISR Tool for known issues/limitations
  - Evaluate Software Engineering Environment to use
    - Identify compiler bugs/limitations (if possible)
    - Write representative test code to try to exercise the compiler/linker
  - Testing equip/environment (including availability)
  - Get team together find out each member strengths and weaknesses. Get interaction started. Review the tasking breakdowns, etc.

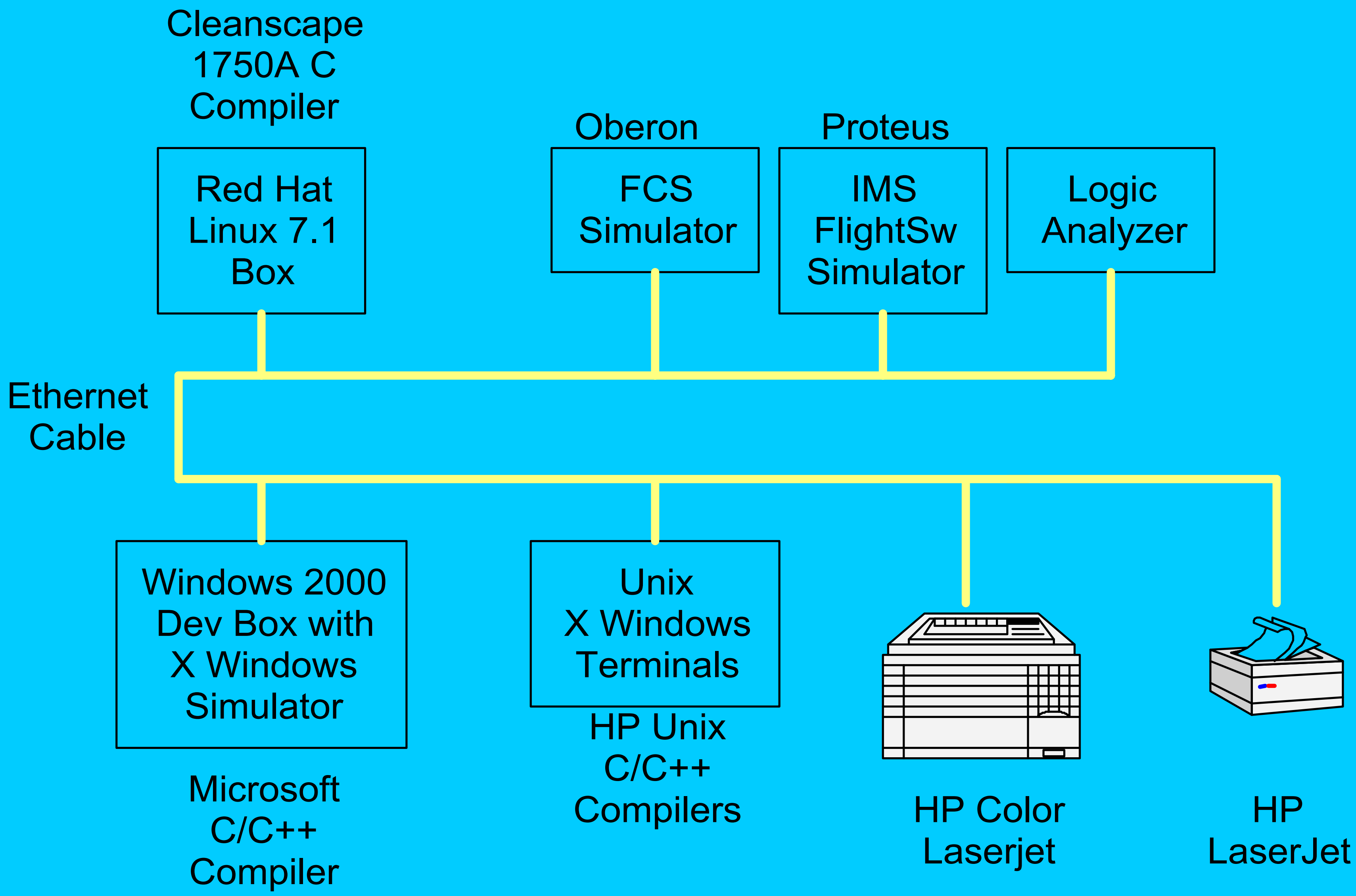
Re-Engineering the Software
Step 2 – Re-Engineering, Preliminary

- Purchase Selected C Cross Compiler for 1750A Processor
- Introduce team members
  - Have team meeting and evaluate team interaction (Do they get along)
  - Finalize the tasking assignments and tentative due dates
- Acquire development computers, install operating system (Linux 7.1 & Windows 2000), network the machines, install compilers
- Install and Learn to use the EISR Tool (Get training) – 12 hrs
- Evaluate/review the 54 Jovial modules (5724 SLOCs)
- Evaluate/review the 28 Assembly Modules (1147 SLOCs)
- Evaluate/review the 13 COMPOOL Modules (1907 SLOCs)

Purchased/Acquired Equipment

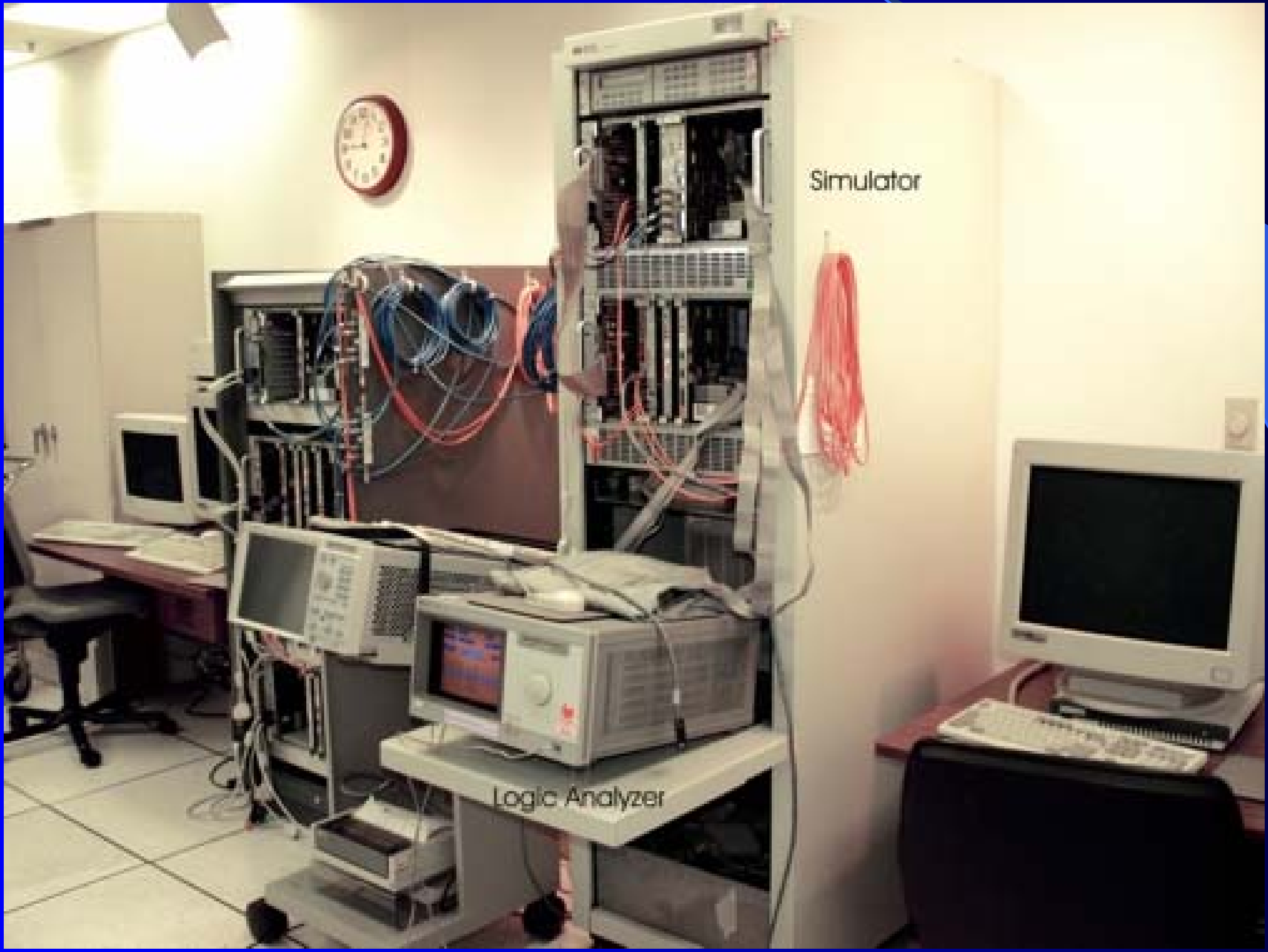
- Cleanscape 1750 Developers Kits (Runs on Linux 7.1) – PN 1750-KIT-LRH (Cost $16,966 inc support)
- Cleanscape LINT-PLUS Static Source Code Analyzer for C
- Red Hat Linux 7.1 (HP Vectra VLi8 800 MHz P3)
- Microsoft Visual C++ 6 & 7 (Ver 6 has a scoping problem in it that 7 corrects) – Gateway P3 1Ghz & Micron CT 800 MHz respectively

Jovial to C Support Equipment

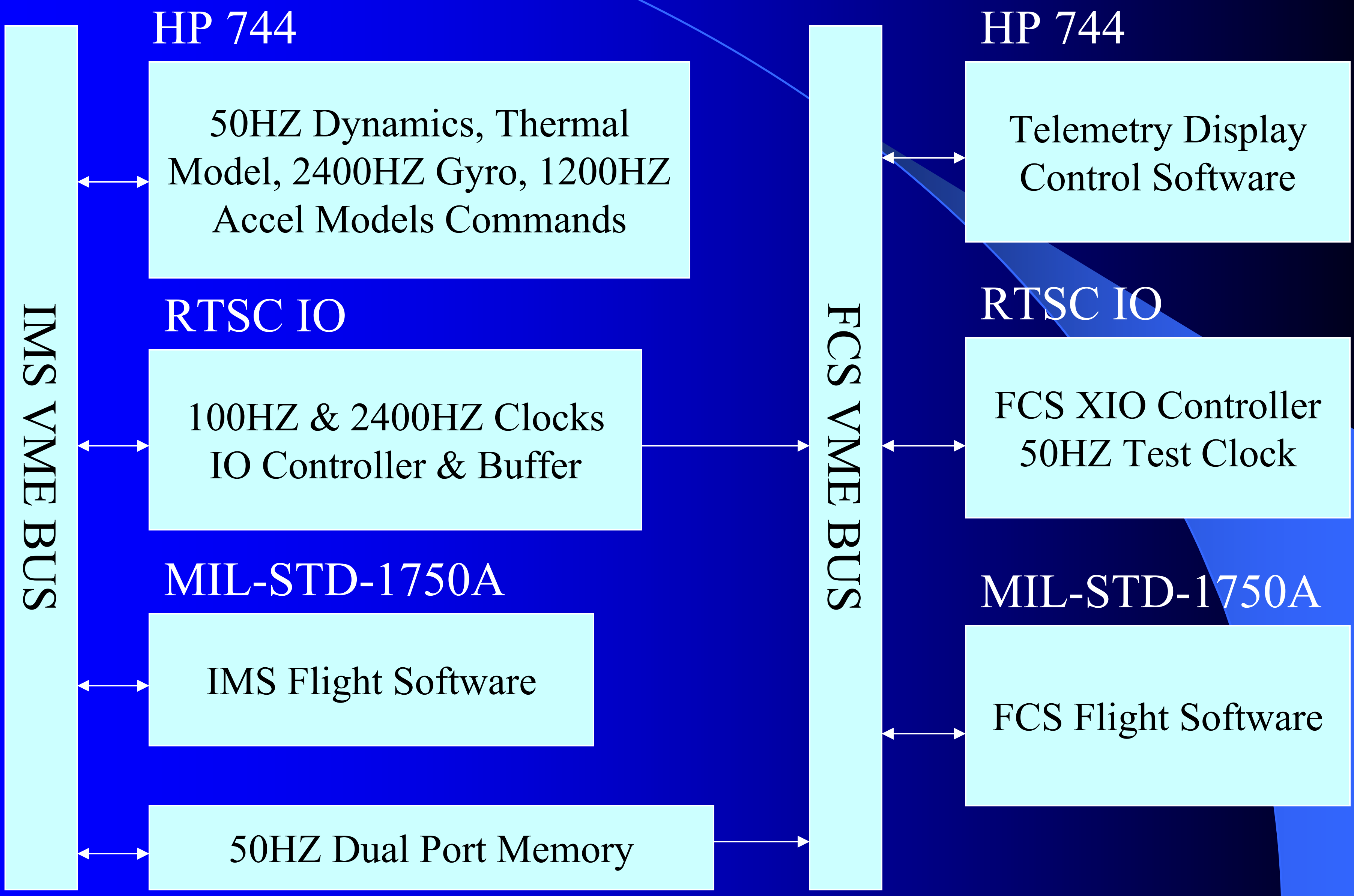


Atlas Centaur Simulator

Six Degrees of Freedom (6DOF) Simulator



Atlas/Centaur Real Time Simulation


IMS VME BUS
HP 744
50HZ Dynamics, Thermal Model, 2400HZ Gyro, 1200HZ Accel Models Commands
RTSC IO
100HZ & 2400HZ Clocks IO Controller & Buffer
MIL-STD-1750A
IMS Flight Software
50HZ Dual Port Memory
FCS VME BUS
HP 744
Telemetry Display Control Software
RTSC IO
FCS XIO Controller 50HZ Test Clock
MIL-STD-1750A
FCS Flight Software

Initial Things That Went Wrong Early

- Project was slated to start with FY02 money
- Budget held up in Congress, delaying start of program
- Had trouble getting initial STE from SMC Mgmt to start program
  - Project originally allocate $275k for FY02
  - 7 Nov 01 – Received emergency $10K to start project
  - 14 Nov 01 – SMC Mgmt only allowed .1STE (181 hrs)
  - 28 Nov 01 – Received additional $90k of funding
  - 13 Dec 01 – Received final $175k for FY02 funding
  - Dec 01 – SMC Mgmt only allowed .6 STE (60% of req'd STE or 1086 hrs)
  - Project originally budgeted for 1 STE (1810 hrs)
  - Note: Was able to complete project within .7 STE or 1267 hrs (Total effort including final paperwork)

The Bottom Line is: Be Careful

- Review/Identify all potential program delays
  - If DoD, be careful on the budget approval cycle
  - Are the team members currently available
  - Take vacations into account
  - Research potential equip availability issues
  - Address acquisition delay issues
  - Identify all potential single point failures
- If using FFRDC's or outside contractors
  - Take into account any people ceilings/restrictions (available MTE/STE hours and head counts
  - Review any clearance issues (e.g., Only Govt personnel can work on, etc)

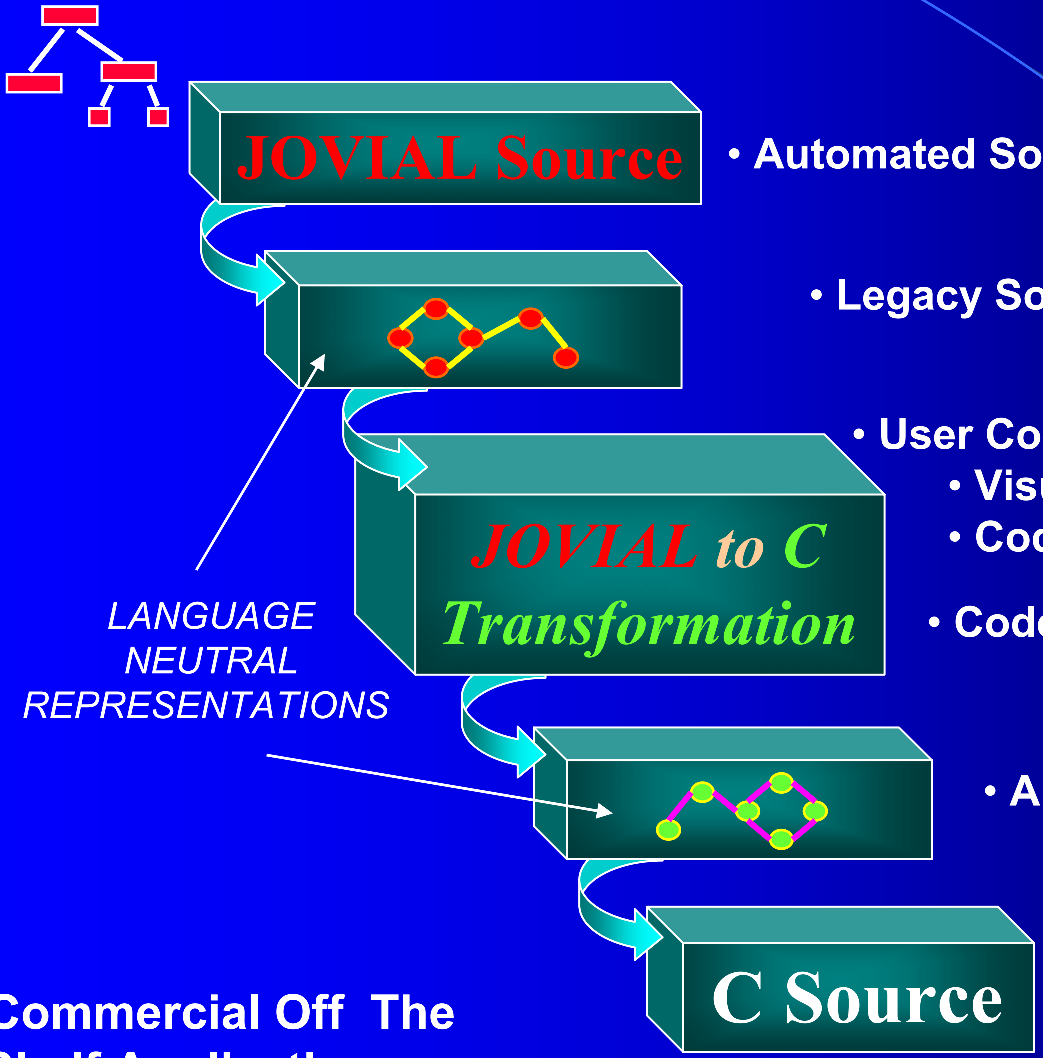
Initial Things That Went Right

- We acquired the computers, operating systems, compilers on schedule
- Compilers up and running on schedule (Despite a disk crash on the Linux machine)
- The team members with the required skills were identified early (Everyone even got along)
- Available lab space located
- Bottom Line: A safety factor had been added in to allow for budget approval delays, equip acqusition delays, computer crashes, and simulator sharing.

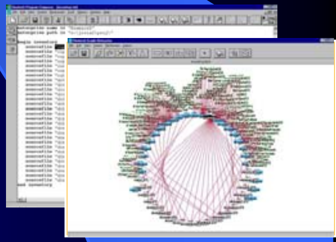
EISR Tool Summary

- It is a two pass tool
  - It identifies the symbols used in the Jovial .JOV and .CPL files, looks at their usage, and produces the .C and .H files.
  - Provides graphic representation of functions and formal arguments
- Runs on Windows NT and 2000
- Allowed us control and provided insight into our conversion effort.
  - We were warned about overlay problems, macro conversion problems, and unconstrained array problems, to name a few.
- The tool provided a graphical representation of our functions and formal parameters.

Re-engineering Approach

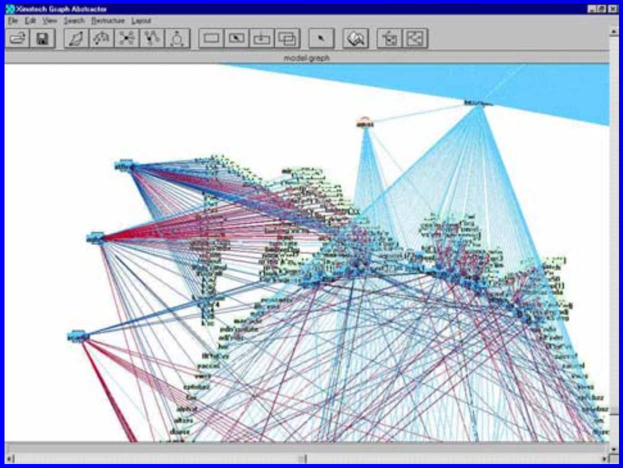


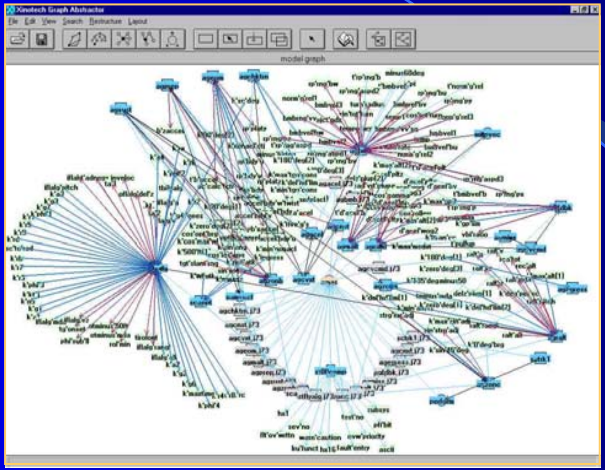
- Automated Source Parsing and Capture
- Legacy Source Code Analysis
- User Configured Environment
  - Visualization View
  - Code Editor View
- Code Segment Selection

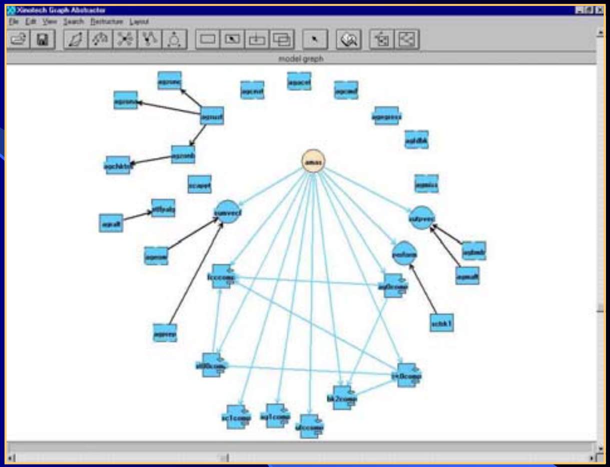


- Automated Conversion
- Automated Source Code Generation
- Commercial Off The Shelf Application

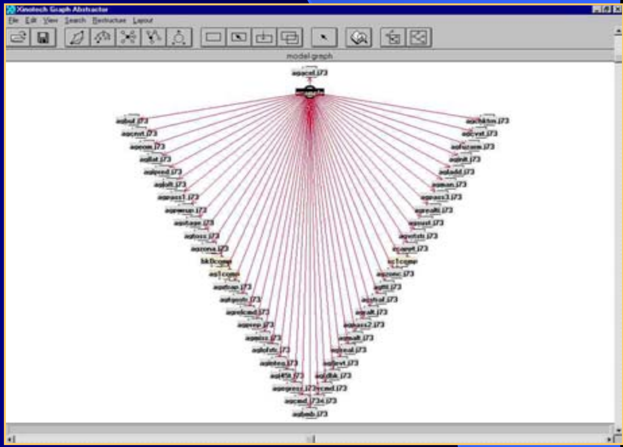
Screenshots - Code Analysis


Xinotech Graph Abstractor
File Edit View Search Restructure Layout
model graph


Xinotech Graph Abstractor
File Edit View Search Restructure Layout
model graph

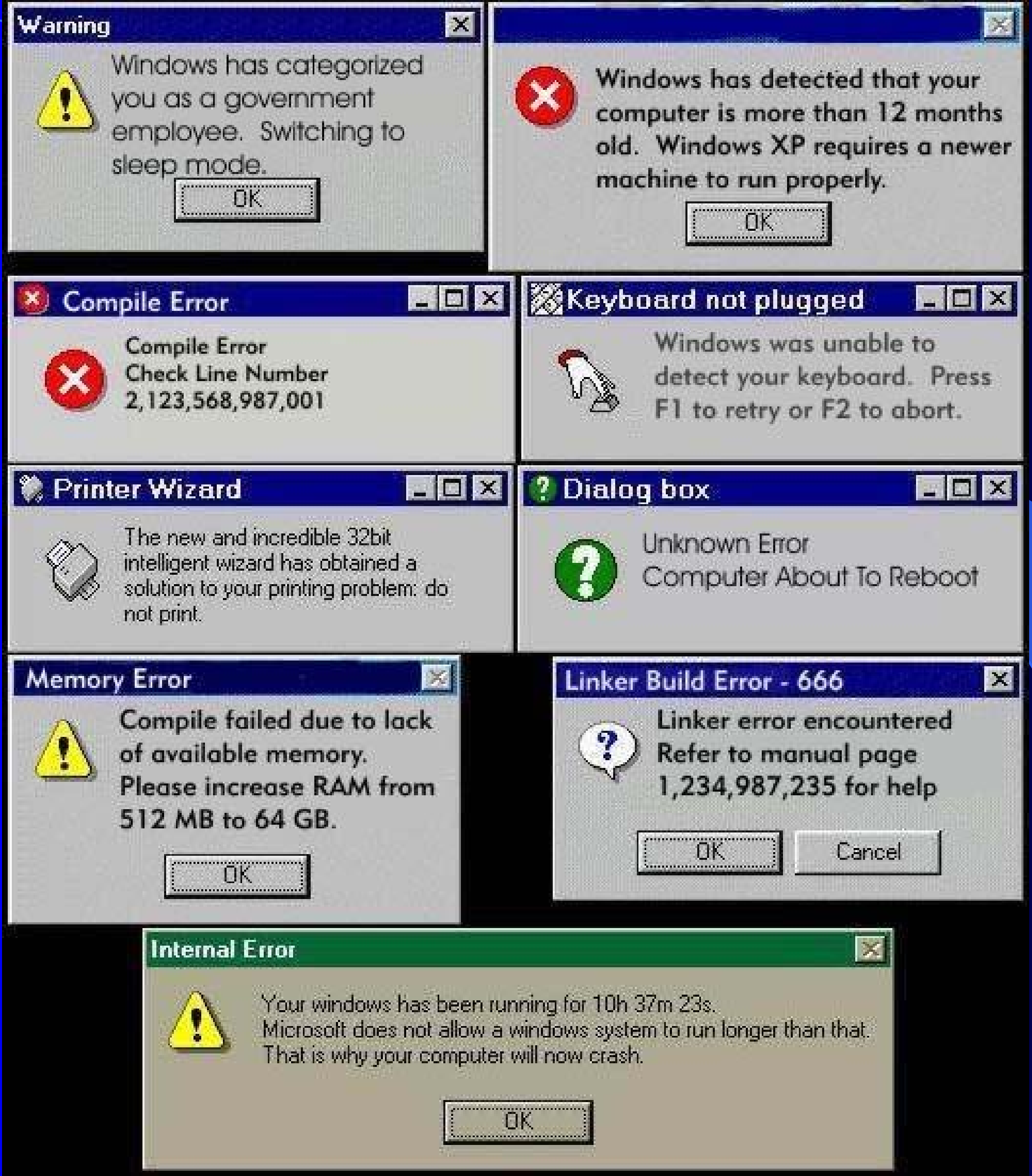

Xinotech Graph Abstractor
File Edit View Search Restructure Layout
model graph

- Windows NT Based
- Graphical and Text Based Views
  - **Code**
  - **Segment coupling**
  - **Variable / Data dependency**
  - **Control flow**
- Options for View Customization
  - **Select code artifacts of interest**
  - **Transform selective segments**


Xinotech Graph Abstractor
File Edit View Search Restructure Layout
model graph

Re-Engineering the Software Step 3 – Convert Code/Initial Compile

- Convert the 54 Jovial Modules to C
  - Took 67 hrs and 20 minutes (Did 5 passes)
  - Done by 1 Nov 01 (Prelim schedule had 19 Nov 01)
- Did initial compile with Microsoft C++ 6 for all modules to get major bugs out (Great debugger).
  - The Cleanscape compiler did not have clear error messages/debugging information
- Did final compile with Cleanscape cross compiler



Warning

Windows has categorized you as a government employee. Switching to sleep mode.

OK

Windows has detected that your computer is more than 12 months old. Windows XP requires a newer machine to run properly.

OK

Compile Error

Compile Error
Check Line Number
2,123,568,987,001

Keyboard not plugged

Windows was unable to detect your keyboard. Press F1 to retry or F2 to abort.

Printer Wizard

The new and incredible 32bit intelligent wizard has obtained a solution to your printing problem: do not print.

Dialog box

Unknown Error
Computer About To Reboot

Memory Error

Compile failed due to lack of available memory. Please increase RAM from 512 MB to 64 GB.

OK

Linker Build Error - 666

Linker error encountered
Refer to manual page
1,234,987,235 for help

OK Cancel

Internal Error

Your windows has been running for 10h 37m 23s.
Microsoft does not allow a windows system to run longer than that.
That is why your computer will now crash.

OK

At Times We Wanted To

Garfield

Destroying

Computer

Cartoon

EISR Tool Issues

- Ver 1.0 yielded error messages with no indication where the error had occurred (Ver 1.1 corrected this error, but we were done with this phase by then)
  - We used a trial and error method to locate the errors
- Jovial arrays start with a 1, so 1st element of array name would be (name(1)), C arrays start with 0 (name[0]). The EISR tool ver. 1.0 & 1.1 handled this inconsistently.
- Jovial array notation is name(12) whereas C notations is name[12]. The EISR tool 1.0 & 1.1 handled this inconsistently.
- C requires declarations to be at the beginning of the module. The EISR tool, Ver 1.0 & 1.1, kept putting the declarations in the middle of the modules.

EISR Tool Issues Cont'd

- The EISR tool incorrectly converted the Jovial absolute operator (abs) which can operate on both type float and integer numbers to an Integer absolute function in C (abs). A number of the C absolute operations should have called the C floating point function fabs. – 16 hrs to work around

- The EISR tool failed to include any guarding in the produced .H files, which it had the wrong extension on (.CPL).
  - Guarding though preprocessor directives, prevents things like multiple variable declarations, etc.
  - Example:

```
#ifndef _MODULE1_H
#define _MODULE1_H
// Other stuff
#endif
```

Cleanscape Compiler/Assembler Issues

- Cleanscape was quick to patch compiler bugs (usually within 72 hrs)
- Compiler documentation shows room for improvement
- Debugging was at command prompt level with erroneous line number indications
- Encountered linker problems due to poor documentation
- We were able to workaround or resolve all compiler/linker issues with outstanding support from Cleanscape

Sample of Compiler Errors/Problems

- "Too Many Commons" Prob – 2 Day Delay (Fixed)
  - Compiler set incorrect max size less than the 64K word single block limit
- C compiler does not like constants in the R value (Used define statements to work around)
- Compiler had a large number of non-std reserved words (We changed identifier names)
- Compiler had a problem with negative value floating point numbers (Found to be position dependent in the code, very unusual problem)

Sample of Language Conversion Issues

- Encountered Stack Problems – Orig Jovial/Assem compiler used Reg 13 for the stack. New C/Assem compiler used Reg 15.

- Encountered Overlay Issues – Jovial directly supports overlays (where two identifiers can refer to the same address) . C requires the use of the union construct (necessitates a more extensive rewrite of the code).
  – We worked around the problem by moving the variables and arrays that had to be overlayed from the C and Header files to the Assembly modules

Sample of Language Conversion Issues Cont'd

- Had to author C replacement functions for native Jovial functions such as: bitset, bitreset, bittest, etc – took 19 hrs
- Had to author C shift functions to replace integrated Jovial Ones – 19 hrs
- We had to highly optimize our code
  - 1750A only directly addresses 64K word, though it can handle up to 16 blocks of 64K words each
  - Jovial Based IMS Ver 9 Code was 61,443 Words (max 65,536) – 13,498 Words code & 47,945 Words Data
  - C Based IMS Ver 22 Code was 61,407 Words – 13,354 Words Code & 48,053 Words Data

Build Code Summary

	Code	Data	Total
Jovial	13,4989	47,945	61,443
C	13,354	48,053	61,407
Delta	C Smaller by 1.06% (144 Words)	C Larger .225% (108 Words)	C Smaller .05% (36 Words)

Linker Map

IREL (Runtime Initialization Code)	0000 - 0020
NREL - Normal User code	0040 - 344A*
KREL - Constants (Excluding in-line constants)	344C- 34D9
SREL - Global and Static Variables	34DB - F003*

Addresses from Operational System Memory Map

Cmd File That Builds Executable Image

```
;RELOAD50 @cims.cmd
;For Finalbuild1.3   22Jan02
cims  -F
0020   -Z    ; ZREL (Gen purpose seg) – Mapped to either RAM or ROM
intrvect.rb  ; The Interrupts start at 20 (Addresses are in hex)

1500  -I    ; IREL (Run-time initialization code) – Mapped to ROM
3500  -K    ; KREL (constants excluding in-line ones) – Mapped to ROM
4500          -S    ; SREL (Global & Static Var) – Usually mapped to
RAM
9900  -N    ; NREL (Normal user code) – Mapped to ROM
start50.rb                ; startup object file (req'd)
ctype.rb  data50.rb       ; runtime-library support files
io50.rb
                          ; Program object files go here
mainmod.rb
initvect.rb
variables.rb
aero.rb                   ; Contains the INPUT1 and OUTPUT1 routines
A000 -N
dualportmem.rb            ; Hardware interface
```

With Severe Difficulty Assigning Absolute Physical Addresses Sometimes We Felt Like This

Charlie Brown

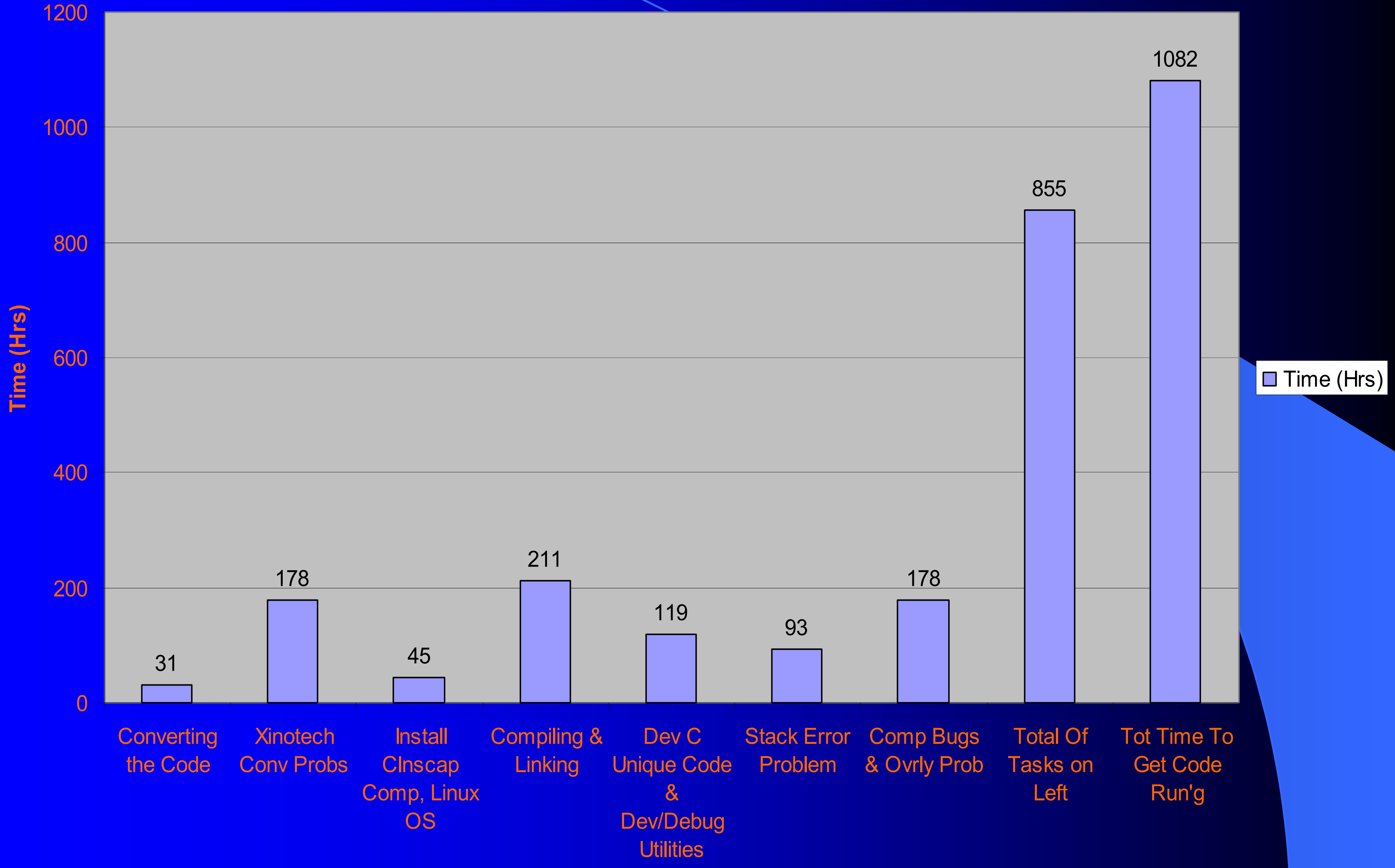
Tie To Tree

Cartoon

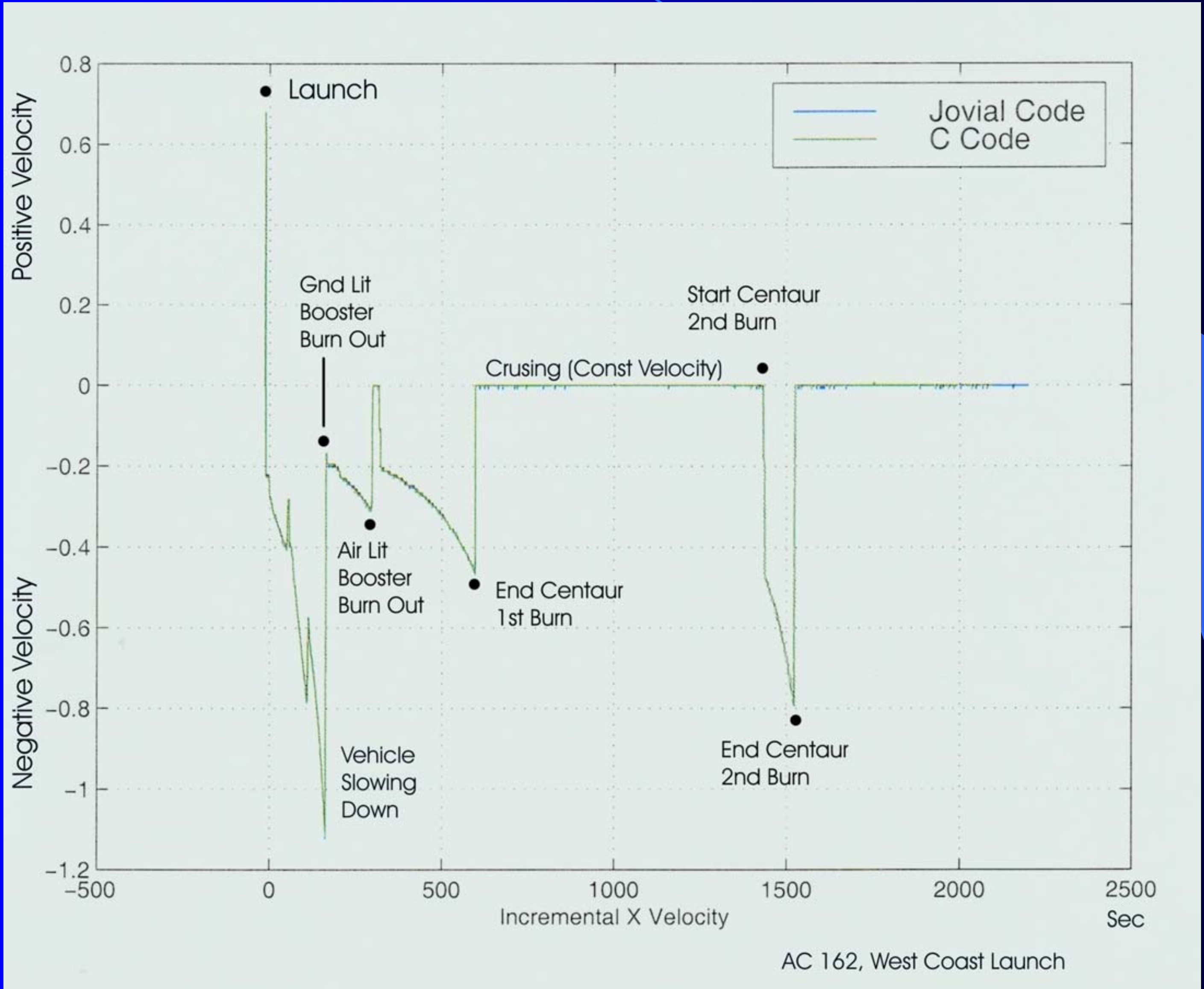
Re-Engineering the Software Step 4 – Integration Testing

- Atlas/Centaur Real Time Simulator is being used in lieu of the actual launch vehicle.
- Since we had a fully functional 1750A IMS and Flight Ctrl Sw operating in Jovial/Assembly, we decided to debug the system at the CSC (Computer Software Component) level rather than at the unit level.

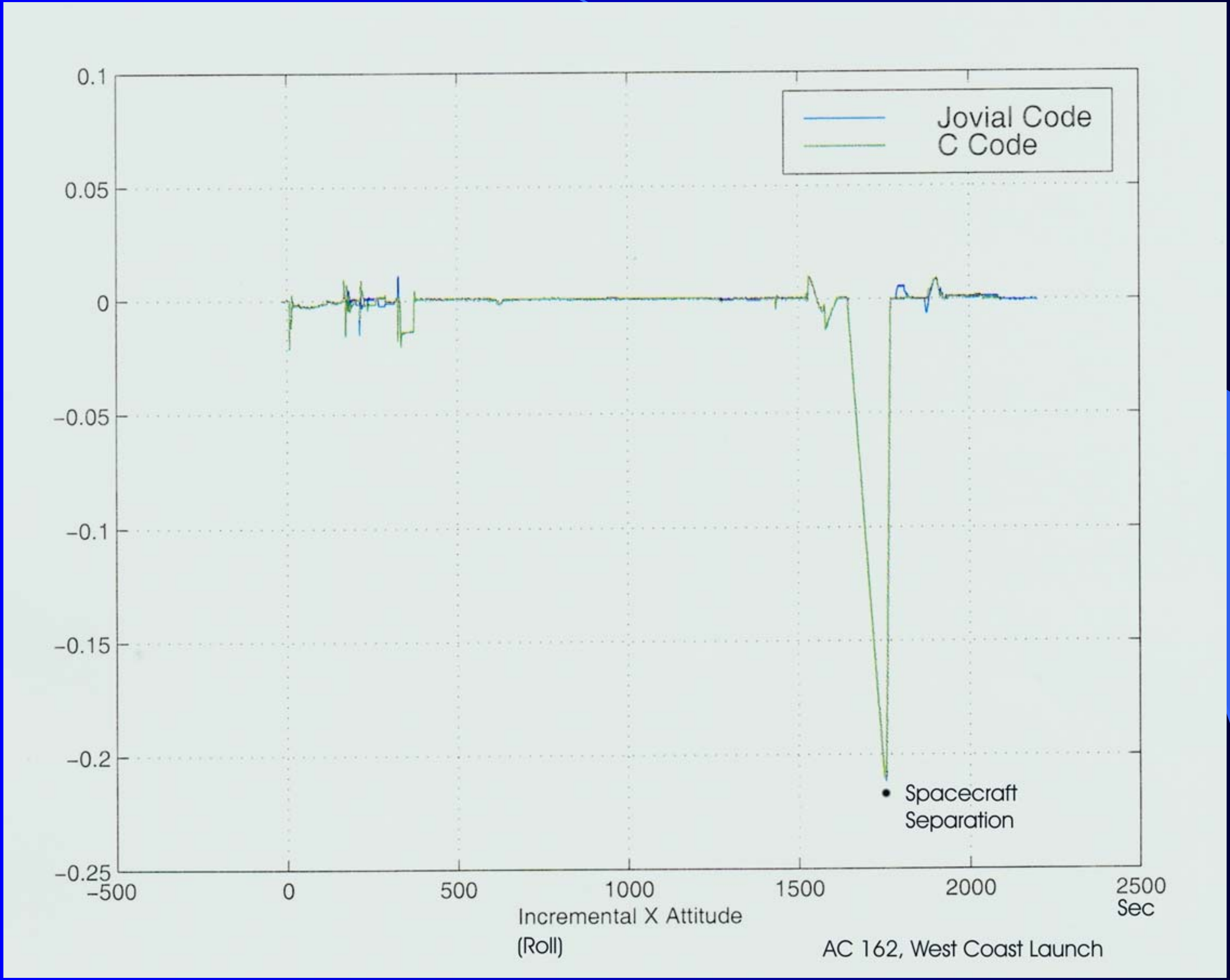
Problem Types Versus Time



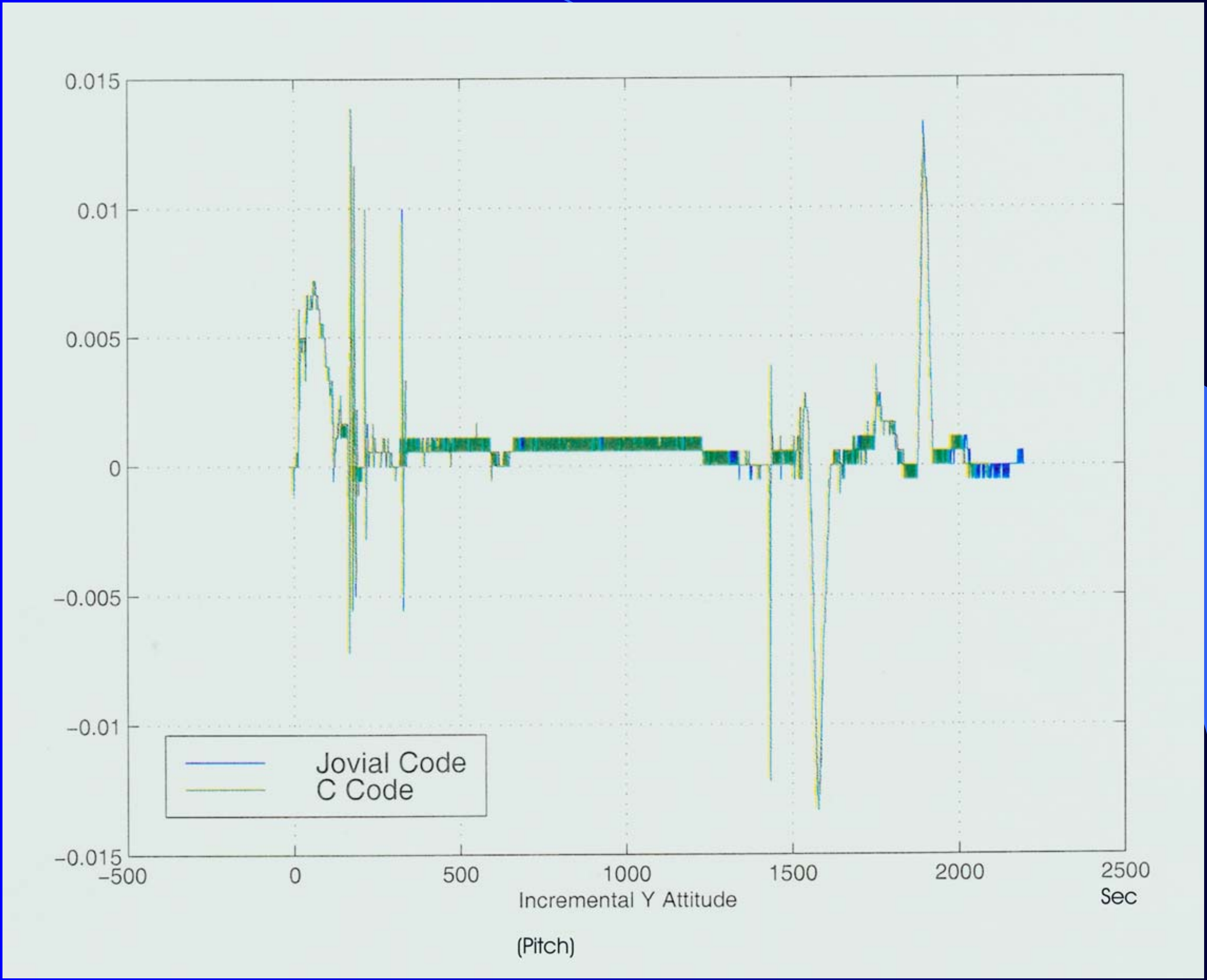
Sample Performance Plot



Sample Performance Plot 2

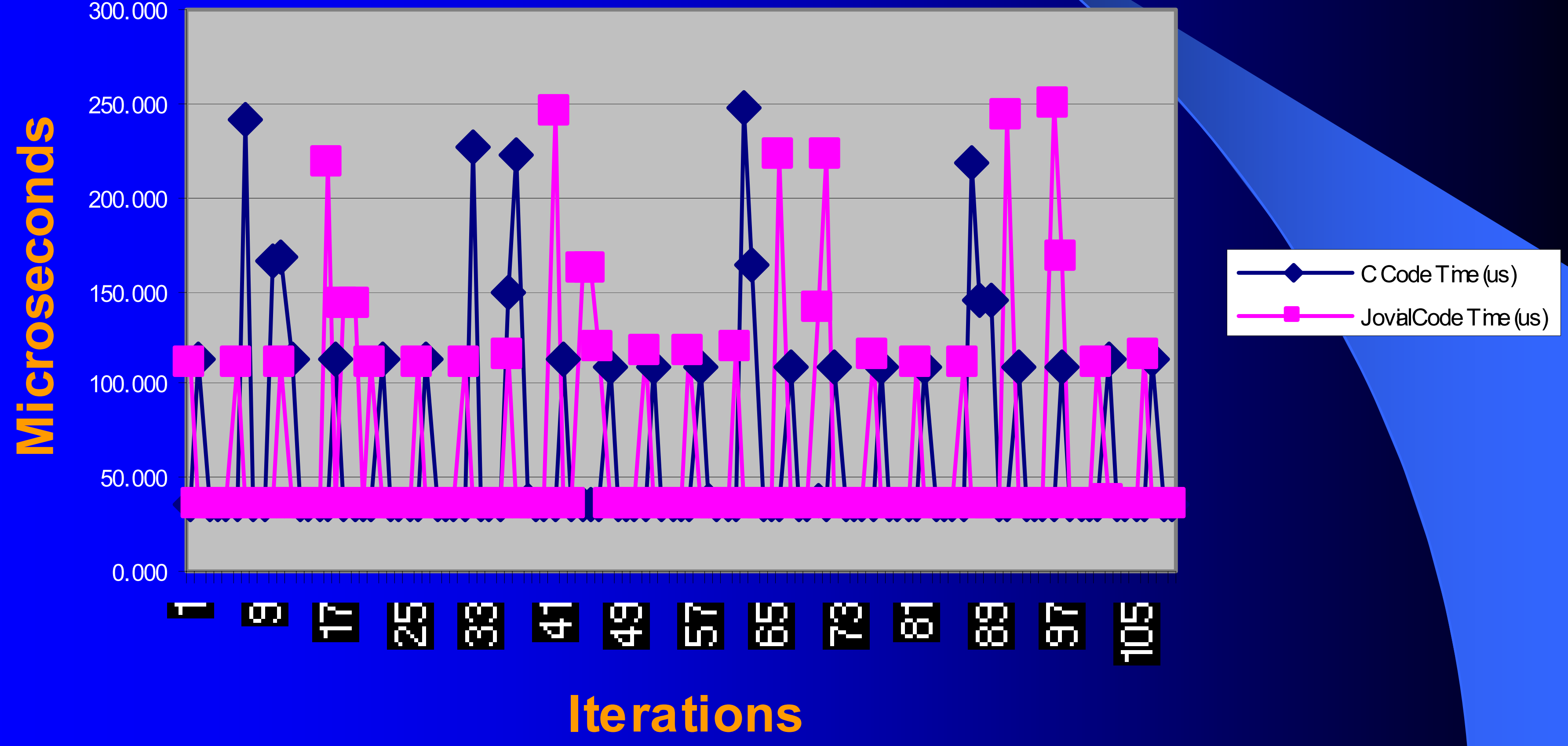


Sample Performance Plot 3



Sample Performance Plot 4

100 Hz Hw Interrupt Execution Time



Step 5 - Replacing Flight Critical Processors

- Choices

Power PC 750	IBM
Power PC 603	Motorola
RAD 6000	Harris
RAD 3000	Harris
RH 32	Honeywell

- Tool Selection
  - Compilers
  - RTOSE
- Documentation
- (Porting Issues)
- Training
- Timing and Memory
  - 16 bits word vs. 32 bits word
- ASM Mapping (Ints, IO, Timers, Load, Store)

Replacing Flight Critical Processors – Our Choice

- A balance between availability, cost and reliability = Power PC
  - MVME2300 Power PC 603 - $5K
  - 750s are around $10K
- The tougher part, though interesting, will be mapping the Assembly instructions.
- Green Hills MULTI Development Environment and optimizing compilers available for Motorola's New PowerPC 740/750 – C++
  - We are currently using this compiler on a different program. It works well, limited learning curve, so we use it.
  - $4,000 for single seat ANSI STD C++ hosted on Linux 7.1

Problems Encountered No One Thought About

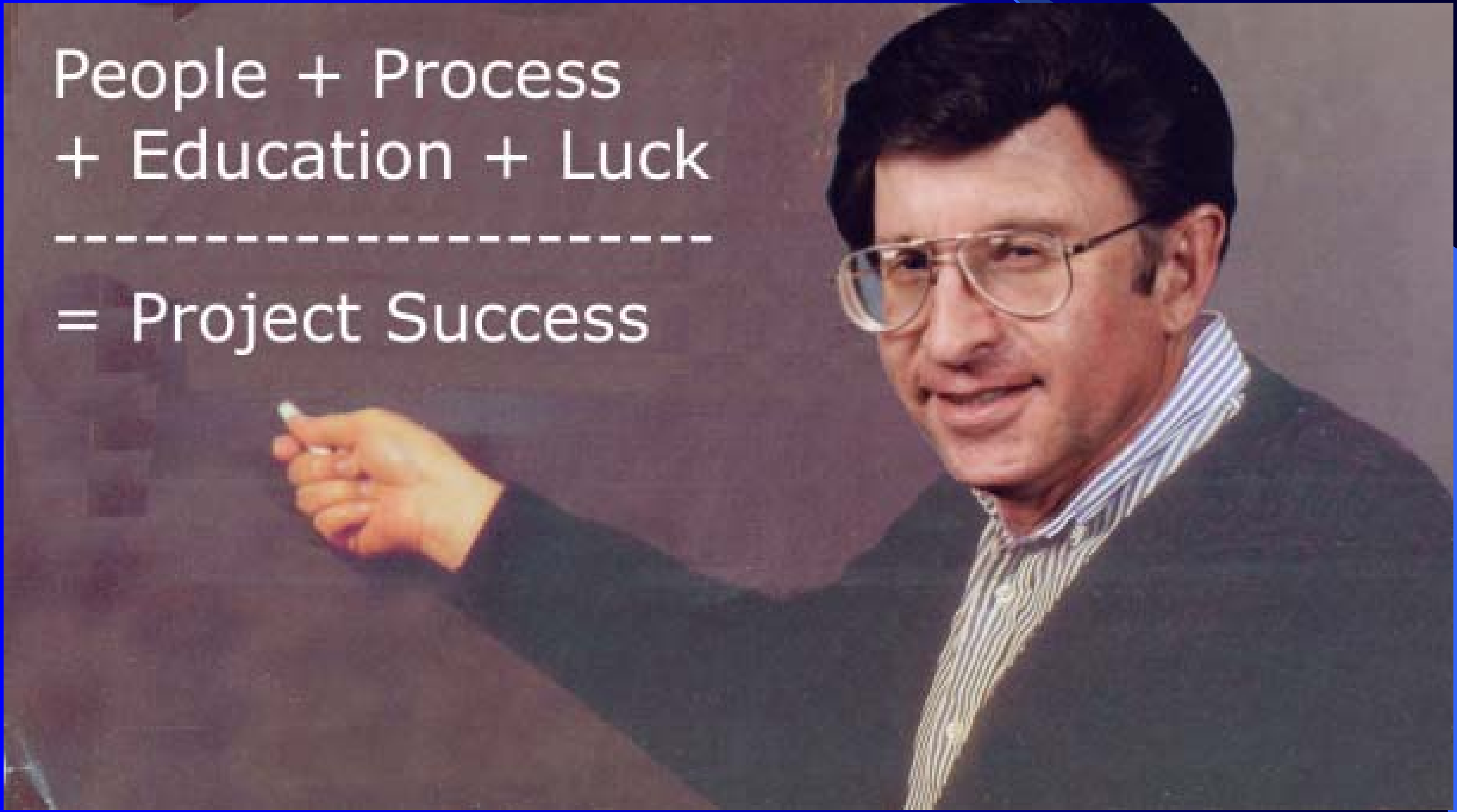
- We required permission to publish the array offset tests we ran using Microsoft's Visual C++ 6 and 7.
- Microsoft declined to respond to the fax and certified letter that we sent to corporate headquarters.
- Had to go to Ed Foster at Infoworld (i.e., Gripline) to get to the correct person at Microsoft to get the permission to publish.

Summary

- Despite the problems we encountered with the EISR Conversion Tool and the Cleanscape Compiler, the project was a success.
- Without the re-engineering tool it would have takes 5430 hrs using 3 people.
- Using the tool, we accomplished the equivalent work in 1082 hrs using 3 people.
- That is a 59.78 percent manpower reduction on a first use (Had a learning curve that does not have to be repeated.)

Summary Cont'd

- Though a simulator was used in lieu of the actual rocket. Delta error between the simulator and the actual rocket is less than .006% (measured)
- Performance Plots Available Upon Request


People + Process
+ Education + Luck

= Project Success

When the code conversion effort goes horribly wrong

Delta II

Explosion

AVI File

Contact Information

Eric Shulman
SMC/AXEC
Space and Missile Systems Ctr
Los Angeles AFB
2420 Vela Way Ste 1467
El Segundo, CA 90245-4611

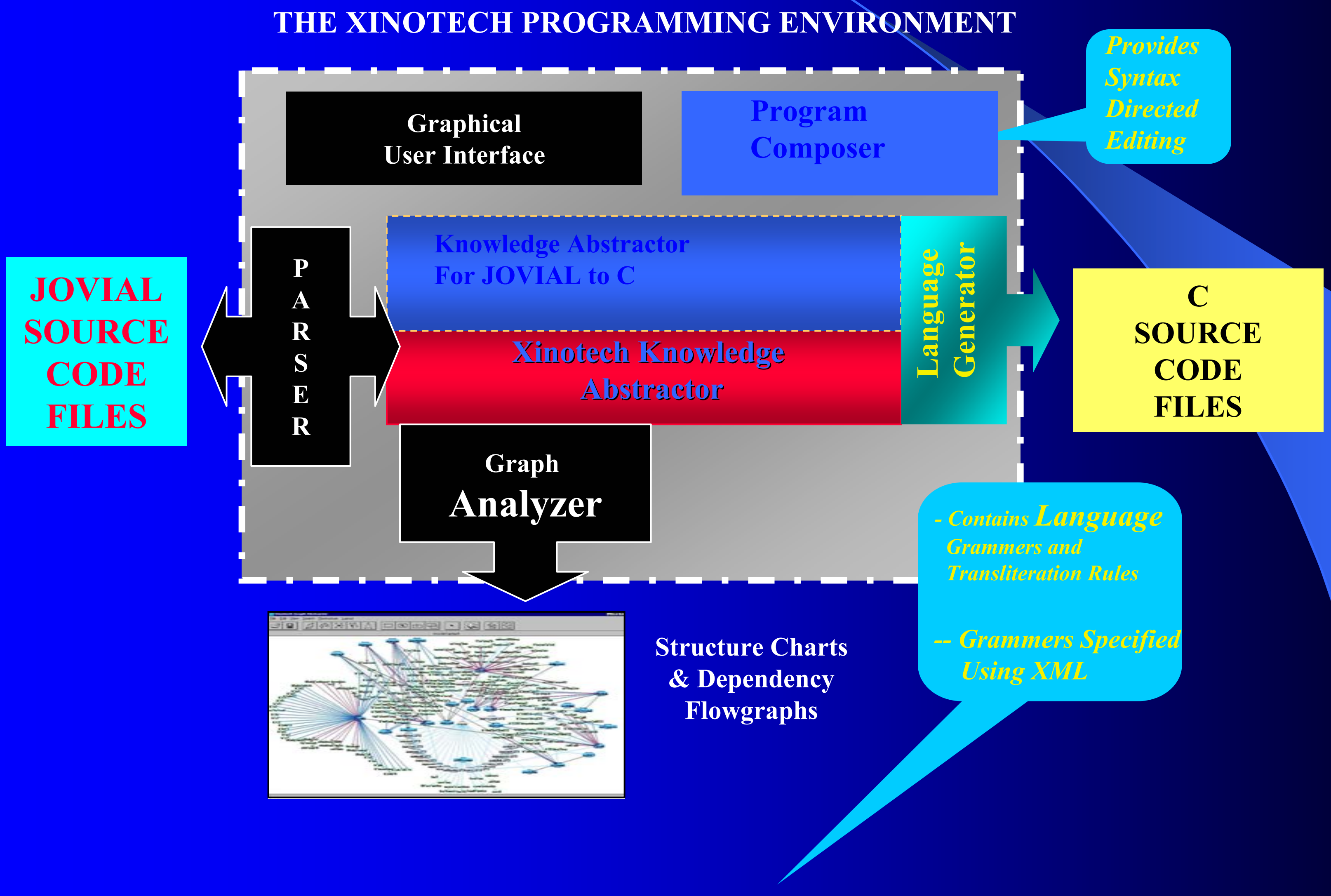
Voice: (310) 363-2436
Fax: (310) 363-2532
DSN: 833-2436
E-mail(W): eric.shulman@losangeles.af.mil
E-mail(H): ericshul@ieee.org

Backup Slides

Conversion Between C and Jovial Types

ANSI C type	Jovial Type	ANSI C type	Jovial Type
unsigned bitfield	B n	signed long	S 31
signed bitfield	N/A	void *	P
char	C	char *	P C
unsigned char	N/A	C-type *	P Jovial-type
signed char	C 7	struct {. . .}[N]	TABLE
unsigned short	U 15	struct {. . . [N]; [N]}	PARALLEL TABLE
signed short	S 15	const	N/A
unsigned int	U 31	Volatile	ABNORMAL
signed int	S 31	N/A	CONSTANT (that is, can be placed in ROM)
unsigned long	U 31		

EISR Language Translation Technology


THE XINOTECH PROGRAMMING ENVIRONMENT
Graphical User Interface
Program Composer
Provides Syntax Directed Editing
JOVIAL SOURCE CODE FILES
PARSER
Knowledge Abstractor For JOVIAL to C
Xinotech Knowledge Abstractor
Language Generator
C SOURCE CODE FILES
Graph Analyzer
- Contains Language Grammers and Transliteration Rules
-- Grammers Specified Using XML
Structure Charts & Dependency Flowgraphs

AC-162 11 Oct 2001

Actual NRO Payload not depicted

T-00:02.4 Engine Start

Atlas booster and sustainer engines are ignited and undergo checkout prior to liftoff.

T+00:00Launch

The Atlas 2AS rocket, designated AC-162, lifts off and begins a vertical rise away from launch pad 36B at Cape Canaveral Air Force Station, Florida.

T+00:08Roll Program

During vertical ascent, Atlas begins a seven-second roll maneuver to align itself with proper flight azimuth. Following the roll, the Centaur inertial guidance system controls pitch and yaw programs.

AC-162 11 Oct 2001

T+00:58.8Air-lit SRB Ignition

The remaining two solid rocket boosters strapped to the Atlas are ignited once onboard computer software determines the two ground-start SRBs have burned out, about two seconds earlier.

T+01:12.1Jettison Ground-Lit SRBs

The two spent solid rocket boosters that were ignited on the ground are jettisoned to fall into the Atlantic Ocean.

T+01:56.3Jettison Air-Lit SRBs

Computer software will determine the air-start solid rocket boosters have burned all their propellant and should be jettisoned from the Atlas vehicle. The two SRBs will fall into the Atlantic Ocean.

AC-162 11 Oct 2001

T+02:43.8Booster Engine Cutoff

BECO occurs when axial acceleration of 5.0 g is obtained on the rocket. Sustainer engine provides the continued boost toward orbit for the Atlas rocket.

T+02:46.9Jettison Booster Package

The bottom engine structure with the two booster engine nozzles is separated from the Atlas vehicle.

T+03:24.1Jettison Payload Fairing

The 14-foot diameter aluminum payload fairing that protected the NRO payload during launch is separated once heating levels drop to predetermined limits.

AC-162 11 Oct 2001

T+04:58.5Sustainer Engine Cutoff

SECO is commanded once minimum residual propellant is sensed inside the Atlas booster stage.

T+05:00.6Atlas/Centaur Separation

The Atlas booster stage separates from the Centaur upper stage. Over the next few seconds, the Centaur engine liquid hydrogen and liquid oxygen systems are readied for ignition.

T+05:17.1Centaur Engine Start 1

MES 1, the longer of the two Centaur firings begins to inject the upper stage and NRO spacecraft into a parking orbit with a perigee of 95 nautical miles and apogee of 490.1 nautical miles inclined 28.2 degrees.

AC-162 11 Oct 2001

T+09:55.9Centaur Engine Cutoff 1

MECO 1 occurs the Centaur engines are shutdown, arriving in a planned parking orbit. The vehicle begins a coast period over the mid-Atlantic before arriving at the required location in space for the second burn.

T+23:54.2Centaur Engine Start 2

MES 2 occurs over the Atlantic Ocean between the African Ivory Coast and Ascension Island before the rocket passes over the equator. The burn lasts until all the Centaur fuel is used, placing the NRO payload into a geosynchronous transfer orbit.

T+25:24.3Centaur Engine Cutoff 2

At the point of MECO 2, the Centaur/NRO vehicle should be in the required transfer orbit with a perigee of 147.9 nautical miles, apogee of 20,246.4 nautical miles, inclined 26.5 deg. Moments later, the stage begins aligning to the satellite separation attitude.

AC-162 11 Oct 2001

T+27:24.3Start Spinup

The Centaur's reaction control system thrusters initiate the required spinup of the NRO satellite to 5 rpm, or 30 degrees per second.

T+29:11.3Spacecraft Separation

The classified payload for the U.S. National Reconnaissance Office is released into orbit from the Centaur upper stage to complete the AC-162 launch.

Image and data source: International Launch Services and Lockheed Martin Astronautics.