

# Building Complex Embedded Software

## Lessons Learned from the OTI Experience with Embedded Smalltalk and Java

Brian Barry  
Bedarra Research Labs  
March 14, 2008



# Background



- **Object Technology International, Inc. (OTI)**
  - A university spin-off formed to commercialize research on embedded virtual machines and development tools
  - Acquired by IBM 1996
  - Better known for VisualAge, Eclipse, IBM j9 Java engine in Websphere
- **BUT OTI had another business deploying VM based embedded systems (1985-2000)**
- **Deployed systems:**
  - Canadian NAVY CANEWS-2 ESM System (Passive Radar Receiver)
  - Instrumentation: Tektronix TDS 400/500/600 Oscilloscopes, HP Network Advisor
  - Process Control: Allen Bradley Network Manager
  - Integrated Manufacturing: TI WORKS (MMST, Control Works)
  - Telephony: Embedded PBX, cellular phones
  - Automotive: IBM Network Vehicle (Smalltalk), IBM Automotive Platform (Java), Motorola Mobile GT, Vehicle Monitoring System

# Tools, Processes and Practices Are Lacking



## ■ Twenty years ago...

- Typical platform: 10 MHz 16 bit microcontroller
- Typical program: 10K lines of ASM
- Typical tools: compiler, editor, debugger
- Typical process and practices: ????

## ■ Ten years ago...

- Typical platform: 66 MHz 32 bit microprocessor 2MB RAM/ROM
- Typical program: 100K -200K lines of C plus some ASM
- Typical tools: SEE ABOVE
- Typical process and practices: SEE ABOVE

## ■ Today...

- Typical platform: networks of 32/64 bit microprocessors
- Typical program: 1M-5M lines of C++/Java
- Typical tools: compiler, SEE ABOVE
- Typical process and practices: SEE ABOVE

➔ Tools and processes have not kept up with the problems and technology

# Deployed Applications



100101110101010  
1000100010010010101000101  
111010101001001100100001  
01000111100100111  
01110000111101010000111  
10101010101011001010100010101  
101111010101001001100101101100  
000101010001010100010111  
10101010010011001000010100  
01111001010101001000010101011110  
00010101000101010001011110  
001010101001000010101011110  
11001010101001000010101011110  
00010110001010100010111  
101011001001100100001010001111  
0010101001000010101011110  
100101110101010  
1000100010010010101000101  
111010101001001100100001  
01000111100100111  
01110000111101010000111  
10101010101011001010100010101  
101111010101001001100101101100  
000101010001010100010111  
10101010010011001000010100  
011110010101010010000101010111  
100001010001010100010111  
1010101001001100100001010001111  
10101010100100010101011110  
100101110101010  
000100010010010101000101  
1110101001001100100001  
01000111100100111  
0111000011110101000111  
101010101010110010100010101  
101111010101001001100101101100  
000101010010100010111  
01010101001101000010100  
011110010101001000010101011110  
000101010001010100010111  
001010101001000010101011110

# CANEWS-2 ESM System



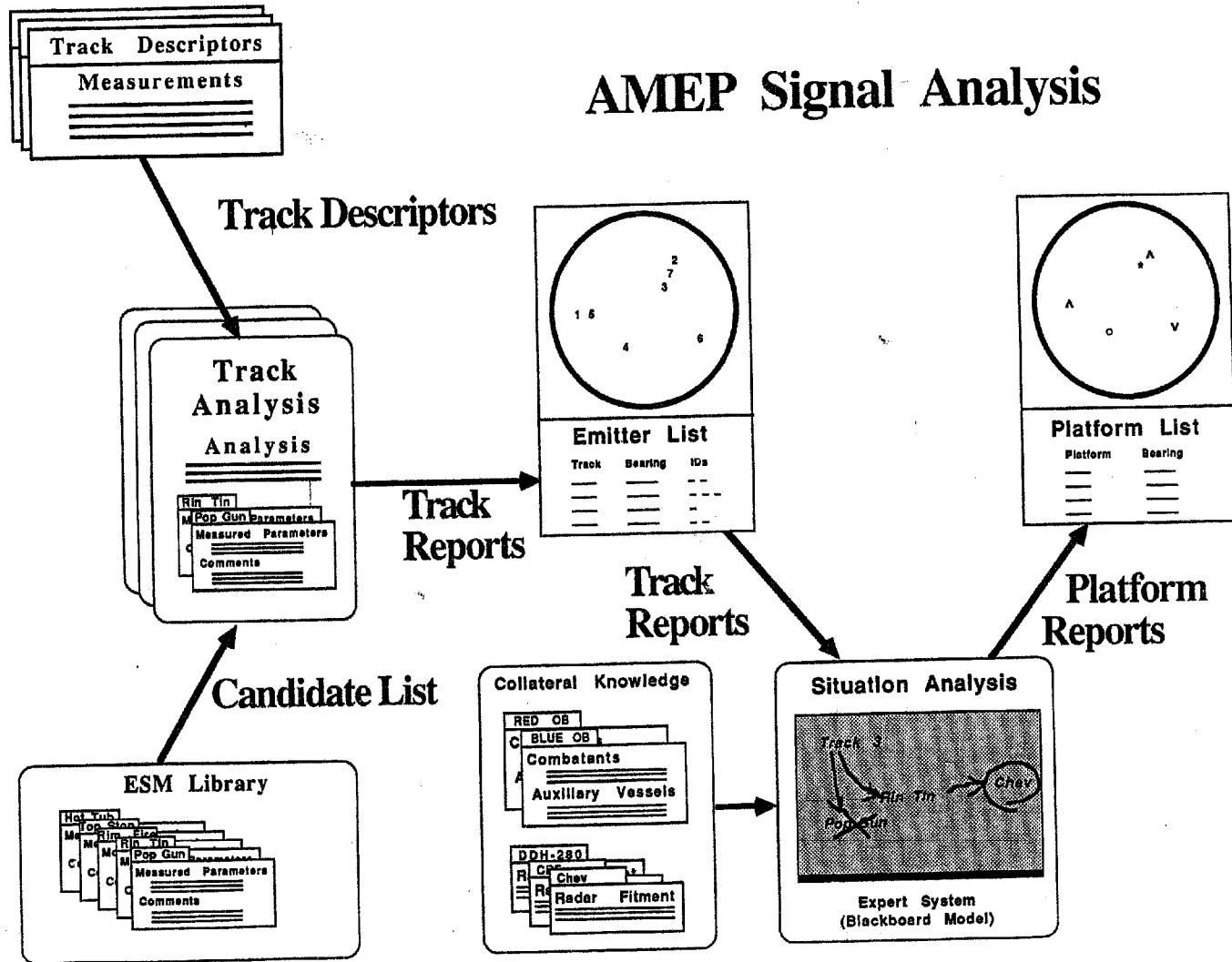
- Canadian Navy Prototype
- Actra Multi-processor Smalltalk
  - custom VLSI pre-processor
  - 4 - 6 MC68030 SBCs
  - stand-alone VME bus configuration
- 130K Lines of Code
  - 90% Smalltalk, 10% C
  - C modules prototyped first in Smalltalk
- Development Effort
  - 10 - 15 developers; 20 months (20 person-years)

**“Prototyping a Real-Time Embedded System in Smalltalk”, Brian Barry, OOPSLA’89 Proceedings**

**“Using Objects to Design and Build Radar ESM Systems”, Brian M. Barry, D.A. Thomas, J.R. Altoft and M. Wilson, OOPSLA’87 Proceedings**



# AMEP Signal Analysis



# Tektronix TDS 400/500/600 Oscilloscopes



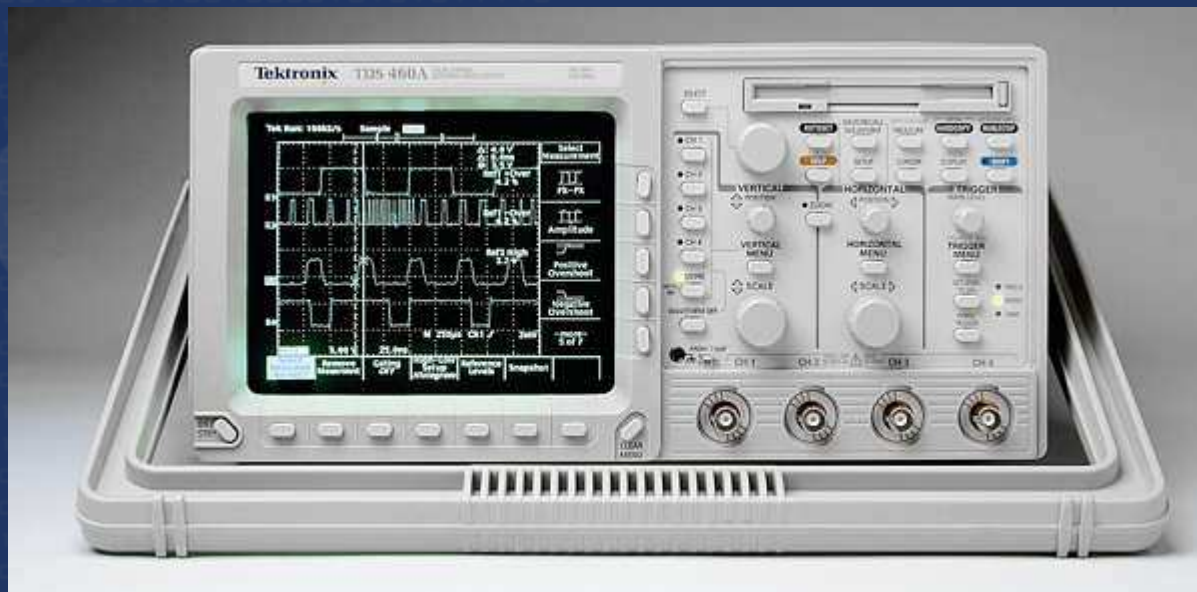
- 500MHz bandwidth, fast sampling
- User friendly UI, waveforms + icons + soft keys
- Multiprocessor architecture, MC68020 and DSP
  - Smalltalk + C (legacy) + assembler (DSP)
  - approximately 450 classes
  - nearly all code in ROM
  - extremely small DRAM requirements
- Smalltalk not visible to end-user
- Significant reuse and quality improvements over C

**“Development of Reusable Test Equipment Software Using Smalltalk and C”, Alan Dotts and Don Birkley, Addendum to OOPSLA ‘92 Proceedings**

# Tektronix TDS 400



**Tektronix**



# Allen Bradley Network Manager



- Process control applications (LAN/1, LAN/PC)
- Tools and utilities provided for:
  - Network configuration
  - Fault management
  - Troubleshooting
  - Performance management
  - Security
  - Remote access
- Strong emphasis on graphical presentation
- Smalltalk + C

# HP Network Advisor



- Ethernet, Token-Ring troubleshooting tool
- Rule based expert system
- Custom designed network interfaces
- RISC based data capture and analysis hardware
- Rich user interface
  - Browsers
  - Virtual control panels
  - Charts and graphs

# TI Works Integrated Manufacturing Facility



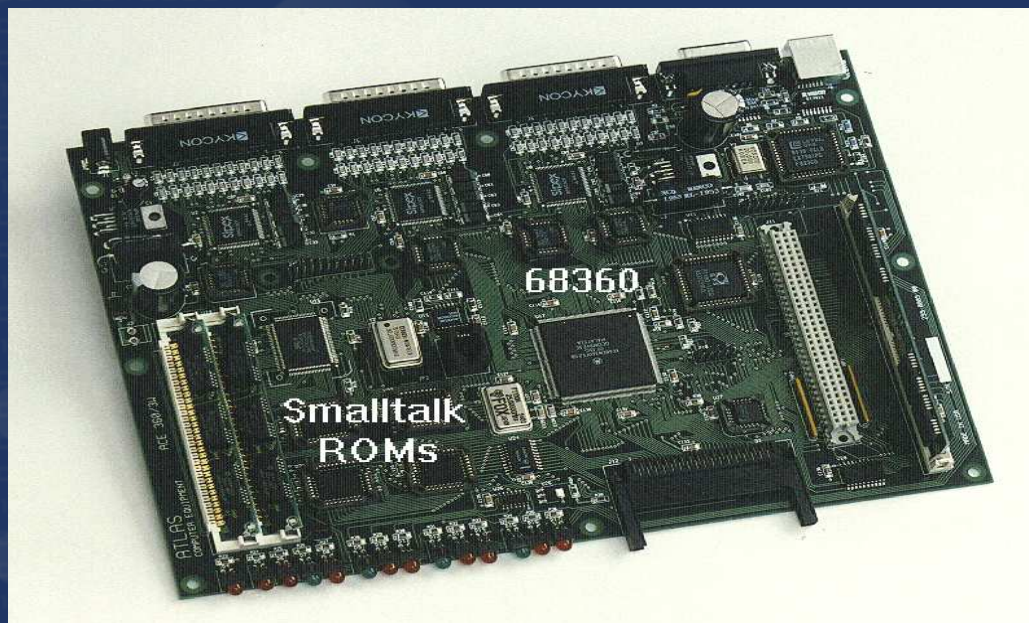
- Integrated wafer fabrication facility
- Goal to reduce fabrication costs and cycle times by 10
- High level factory management tasks- planning, scheduling, production tracking
- Low level tasks - process control and diagnostics, performance monitoring, intra-machine scheduling, quality control
- Smalltalk initially used to construct a simulation of the system architecture and the user interface
- Smalltalk then selected as the implementation language

**“The MMST Computer Integrated Manufacturing System: An Overview”, McGehee, J., Hebley, J., and Phauth, M., Texas Instruments Technical Journal, September/October 1992.**



# Embedded PBX

- Low cost PBX/Voice Mail System
- Used the Motorola 68360 and a TI DSP
- 2 MB ROM and 1 MB RAM
- 500 Classes
- Hardware and software developed in less than one year

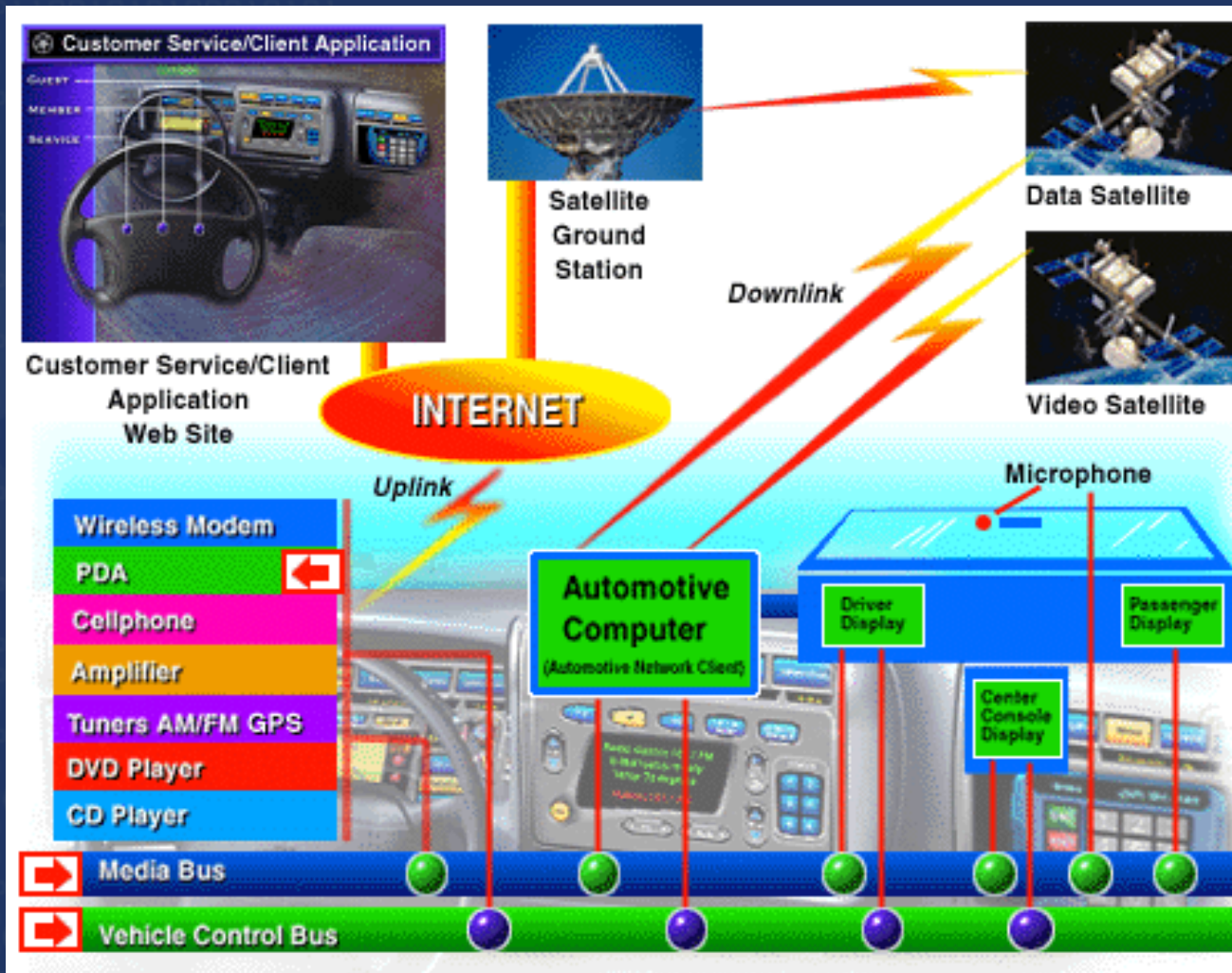


# Network Vehicle

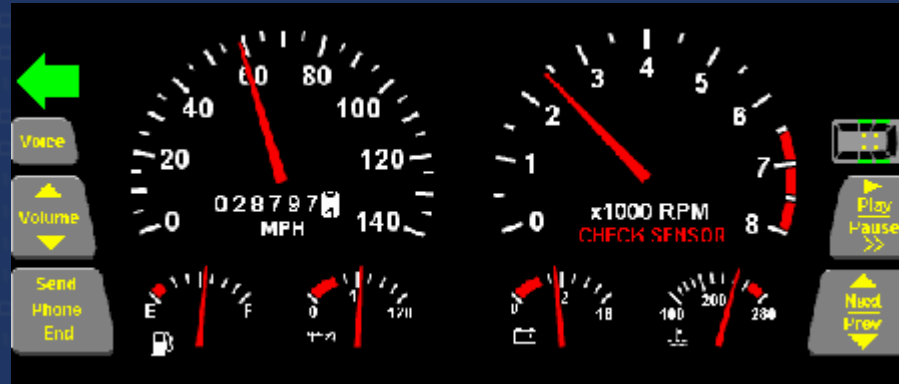


- Ad Tech project to demonstrate the car of the future
- Featured at COMDEX 97 and CEBIT 98
- Smalltalk for control bus device interfacing and for off vehicle wireless communication
  - Socket level interface to speech engine for command/control
  - Socket level communication with Java VM for MMI
- Java for web functions (client and server) and GUIs
- Diagnostics, device control, entertainment, PDA integration, navigation, remote commands
- Pentium processors, Class2 bus (engine and cockpit), MML fibre bus (audio), custom serial bus for displays

# Network Vehicle



# Passenger and Dashboard Displays



# IBM Automotive Platform



- Java reference hardware and software platform for Automotive development
- First version on MIPS 411, 4M/4M RAM/ROM
- 1/4 VGA color display, touch screen, general purpose I/O, attached phone, GPS, Codec (Speech), Serial
- Second and third version Intel Pentium, 4M/4M RAM/ROM (8M/8M development), in-dash form factor
- Targeted to run general telematic type of apps with voice control. sample apps included
  - hands free phone, email, navigation, web/internet access, automotive control bus access, roadside assistance
- 1 year plus effort including JVM porting, application development, server integration



## Mobile GT “Red Box”

- Motorola reference hardware and software platform for Automotive development
- 823e PPC, 16M/8M RAM/ROM, 3x3” processor board and I/O board, small box packaging with LCD
- 1/4 VGA color display, touch screen, attached phone, Codec, Serial
- Applications provided
  - Designed hands free phone, email, navigation, browser, HVAC control
- For MMI migration to speech
- 3 month Java effort based on pre-existing reference platform



## Vehicle Monitoring System

- Field test system for vehicle status monitoring and telemetry back to Internet server.
- Supports vehicle monitoring through WWW browser.
- 823e PPC, 8M/8M RAM/ROM
- 3x3" processor board and I/O board
- Cost, power and reliability are major design points
- No display or MMI. Monitoring via vehicle control bus. Telemetry via GSM wireless module.
- 3 month Java effort based on pre-existing reference platform

# The Big Picture Lessons Learned



- Customers have more to do than just build software
- They are usually building on an existing product – green fields are rare
- They have an existing but weak development process – almost always based on small software projects
- They are mostly domain experts - few real software experts
  - Require support and expect help in building the many “demos” required to overcome management obstacles
- There is never enough time/money/people
  - Result - cycles are long and most projects fail
- Even though we were a technology provider, in fact skills/process/practices transfer was usually the key to success
  - You must have the skills in order to transfer them!

# Technology and Process Lessons



- Scaling down is a lot harder than scaling up
- Version everything; configuration manage everything; the more fine grained the better
- Find a component model that works for you and learn to love it
  - But beware the pitfalls of component based development
- If you expect cross-product integration or product line development, build a common platform (kernel, common services, component libraries)
- Hardware/software co-design is normal – better get used to it
- Use actor based process structuring to organize and manage services
  - Services are good but lots of anonymous services are confusing
- Make it cheap to make mistakes, because it will happen often
  - First get it right then make it fast!
- Adopt and follow Agile Development methods
- To achieve high productivity combine the robustness of cross-development with the incremental interactive feel of in-target development
- “If you can’t measure it you can’t manage and control it”
  - Even truer for software processes than for manufacturing ones

# Scaling Up is Easier than Scaling Down



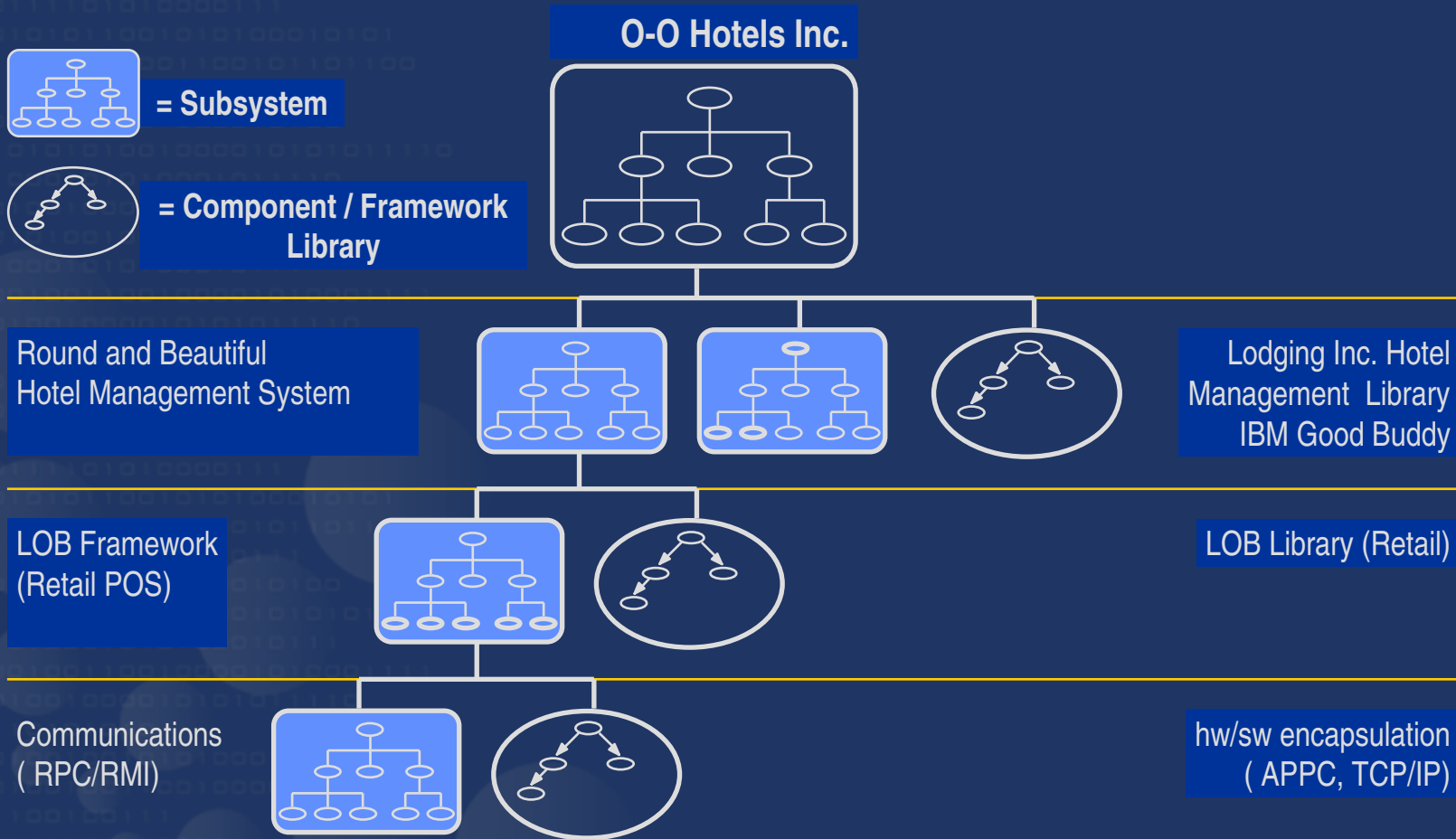
- Workstation technology usually does NOT scale down well
  - J2ME/SE, Windows CE
- Size is the critical performance issue (not speed)
  - key drivers for parts are cost, pin count and power
  - footprint dominates designs for low margin devices
  - workstation Java > 20MB or more
  - stripping tools are useful but not enough
  - class libraries not designed to be small or modular
- Multithreading
  - not adequately specified
  - APIs suggest priority based preemptive scheduler (NOT!)
- Dynamic loading of class files is the wrong model

# Or You Could Design VMs for Embedded...



- Key requirements for embedded systems
  - be defensive and make minimal assumptions
  - isolate VM from platform
- Small and modular class libraries
- Abstractions for objects and structs in memory
- Incremental GC but under programmer control
- Interrupt handling (eg async message queue)
- Mixed mode VM with Self-style dynamic compilation
- Mixed language implementation (Java + C)
- AOT compilers - trade performance for flexibility

# Configuration Management



*The Software Bill of Materials*

# Component Based Development

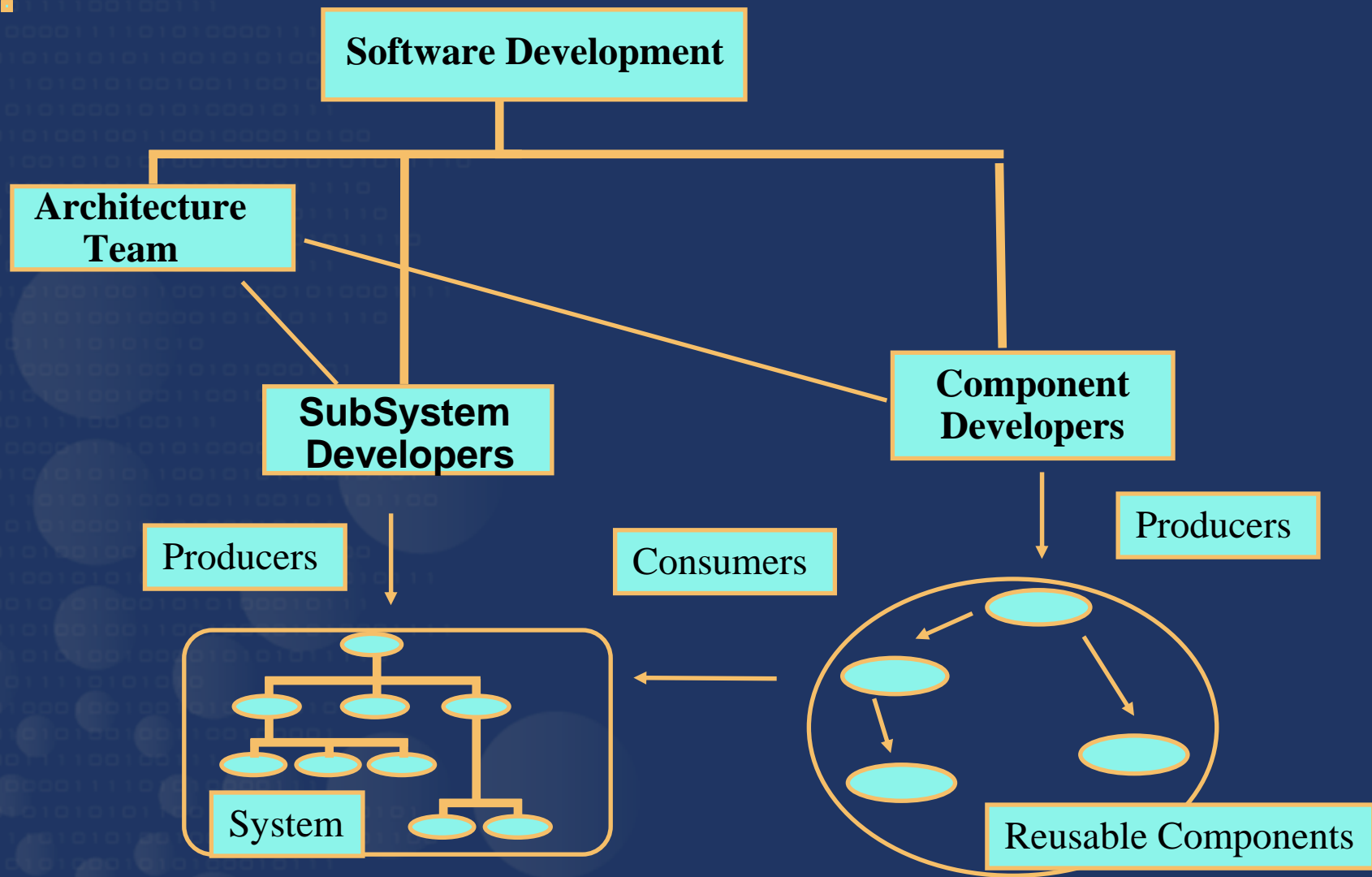


*“Sounds great, and I bet it’s really easy to do! After all, it’s just plug and play...”*

## What the Component Salesman didn’t tell you...

1. Component-based development requires **two cooperating life cycles** and a **major cultural change** for developers
2. Components are small and there are lots of them!
  - You can't reuse what you can't find!
  - Configuration Management is essential
3. Component Libraries need constant re-investment
4. The development process must understand components and manage their life cycles too

# Organizing for Component Reuse



# Process Structuring with Actors



- Assume that system consists entirely of communicating services
- Begin with a simulation or animation of the whole system and then build it out, i.e. build an executable model
- As the system grows it takes on characteristics of a solution
- **Process Structuring with Actors**
  - Actors execute concurrently
  - Message passing can be synchronous or asynchronous (but both is hard!)
  - Actors are “large scale” services (Hewitt’s were fine grained)
  - Actors have personified roles
  - Generic Actors: Servers, Workers, Notifiers, Couriers, Administrators
  - Application Specific Actors: PulseCourier, RadarTrack, TrackManager
- **The taxonomy of known Actors, some generic, many more application specific, creates a vocabulary that populates the programming model and defines its semantics**

# Designing Actor Based Systems I



- **Hewitt's design steps:**
  - Decide what the actors are
  - Determine their message protocols
  - Define their behavior
- **We added structure by:**
  - Providing a taxonomy for Actors
  - Introducing OO inheritance
  - Defining Working Groups of collaborating Actors
  - Only a few Actors in each Working Group
- **Actors know about**
  - Superior
  - Subordinates
  - Colleagues (share a common supervisor)

# Designing Actor Based Systems II

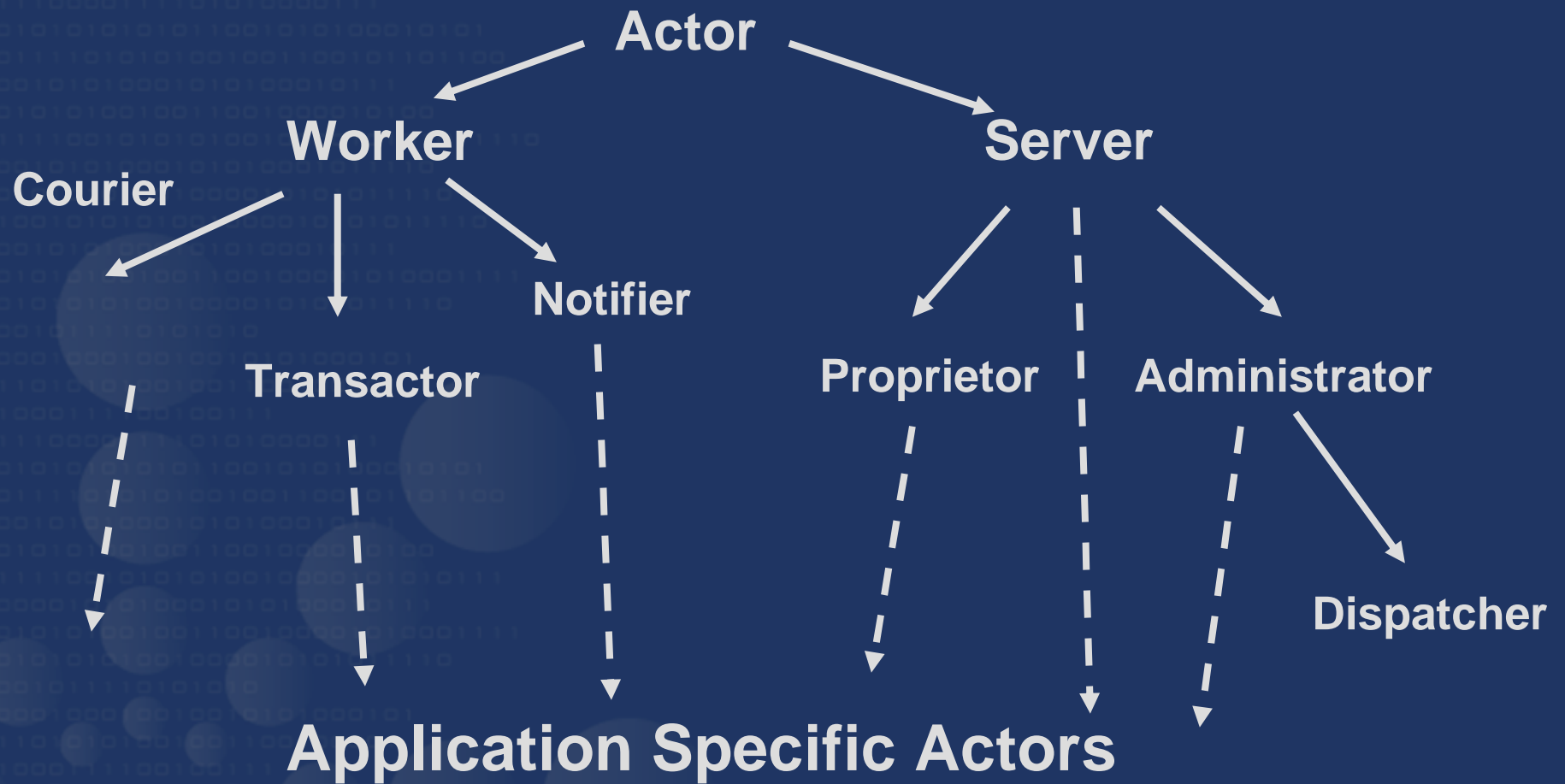


- **Modeling and Mapping Concurrency**
  - Semantics for message passing are similar for actors and objects
    - Decisions made about concurrent behavior are easily changed
  - Uniform semantics for local and remote Actors
    - Processor/Actor assignment becomes a runtime optimization
  - When in doubt, assume a component is an actor
    - It will usually become obvious when the assumption is wrong
- **Well defined life cycle policies**
  - Initialization (e.g. order of object creation)
  - Finalization (e.g. handling child Actors)
  - Standard activation sequence
  - Configurable serialization
  - Policies can be refined in subclasses
- **Delegation used to:**
  - Share responsibility
  - Task subordinates
  - Refer to Supervisor

# Actor Taxonomy



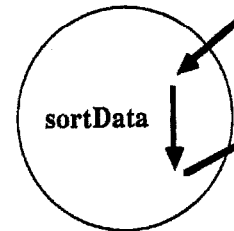
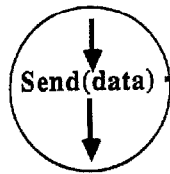
- **Simple inheritance is an easy model to understand and use**
  - Mixins, multiple inheritance, etc. are more powerful but add complexity [always choose the simplest powerful solution]
- **Generic Actors**
  - Worker: send to servers for work, perform computation
  - Notifier: event handling Worker
  - Courier: messenger Worker, used for delegation and communication
  - Transactor: adds ACID properties to computation
  - Server: provides services
  - Proprietor: manages resources, mitigates access
  - Administrator: manages worker pool
  - Dispatcher: provides asynchronous communication
- **Standard system services**
  - ClockServer, DirectoryServer, LogServer, etc.



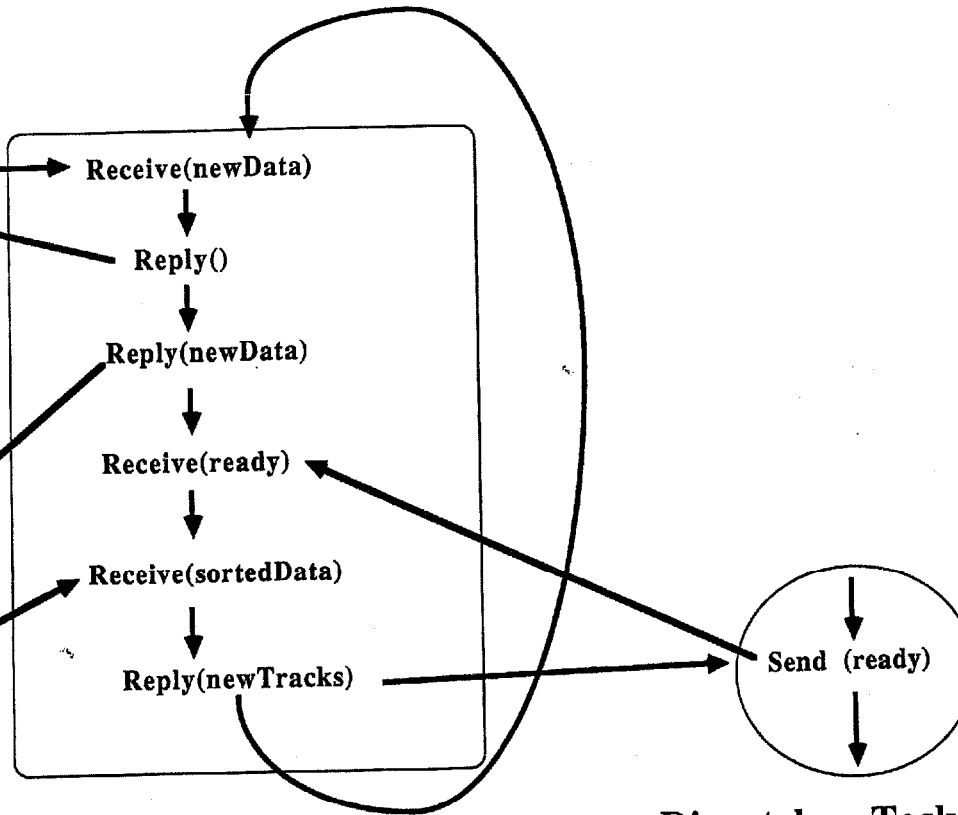
# Example from CANEWS-2 Lab System



**Data Courier**

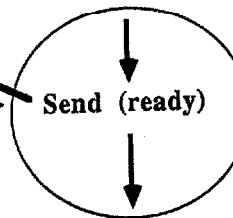


**Sorter Task**



**Acquisition Task**

**Dispatcher Task**



# Pulse Courier Service



## PulseCourier methods

### initial Task

```
super initialTask.  
busInterface :=  
    self find: #BusInterface.  
busInterface isNil ifTrue: [  
    self error "No BusInterface found by Pulse Courier"].  
busInterface assignBufferFor: self parent.
```

### activity

```
"Get data and deliver it to Acquisition"  
| pulses |
```

```
true whileTrue: [  
    pulses := busInterface pulsesFor: self parent.  
    self parent newData: pulses]
```

# Acquisition Service



Acquisition methods

initial Task

```
super initialTask.  
courier := self spawn: PulseCourier.  
dispatcher := self spawn Dispatcher.  
sorter := self spawn: Sorter
```

trackerDispatcherReady

```
trackerDispatcherReady := true.  
newTracks is Empty  
    whileTrue: [self receive]. "sender is blocked"  
^newTracks
```

newData: trackData

```
self buffersFull  
    whileTrue: [ self receive]. "sender is blocked"  
dataBuffers add: trackData
```

# Peopeware: Lean and Agile Development Process



- Draws from OTI, Lean Manufacturing and other experiences
- A process and a set of best practices
- Four phases or stages:
  - Envisioning: The Art of Tangible Requirements
  - Definition: Convert requirements to architecture, features, components, schedules, plans
  - Development: Build the software (“classical agile practices”)
  - Release Engineering: Coordinate and ship the product (“the End Game”)
- Typical Breakdown of effort
  - Envisioning: 10-15%
  - Definition: 15-20%
  - Development: 40-55%
  - Release Engineering: 10-15%
- Stages are concurrent not serialized
  - They’re all active to some degree all the time

# How Agile Development is Different



- **Accepts there is a “software physics”**

- Validate risks so they can be understood and managed
  - You can't build what you don't understand or can't test
  - You can't do new design/feature in a release time box
  - You can't build components and dependent applications in the same time box
  - The only hard decision is what you are NOT going to do
  - Done means finished AND acceptance tested!

- **Directing and Managing => Leadership and Coaching**

- Work With versus *Work For* - Coaching versus Directing
  - Increased self discipline for teams and individuals who own deliverables, quality and schedule
  - Increased individual ownership with associated responsibility and accountability
  - Leadership identifies and manages risk

- **My Way => The Best Same Wrong Way**

- Common vocabulary, practices applied sensibly and metrics aligned with practices
  - Make sure everyone knows the same way before fixing it - Improve process each release of the company i.e. triage process/practice/tools defects like other defects

- **Independent Technical and Coaching Ladders**

- Coaches valued for people skills; Technical leaders valued for technical skills
  - Peer evaluation is an important promotion metric for both
  - Mandatory constructive annual reviews (PMPs)

# Java/Smalltalk Cross Development: Motivation

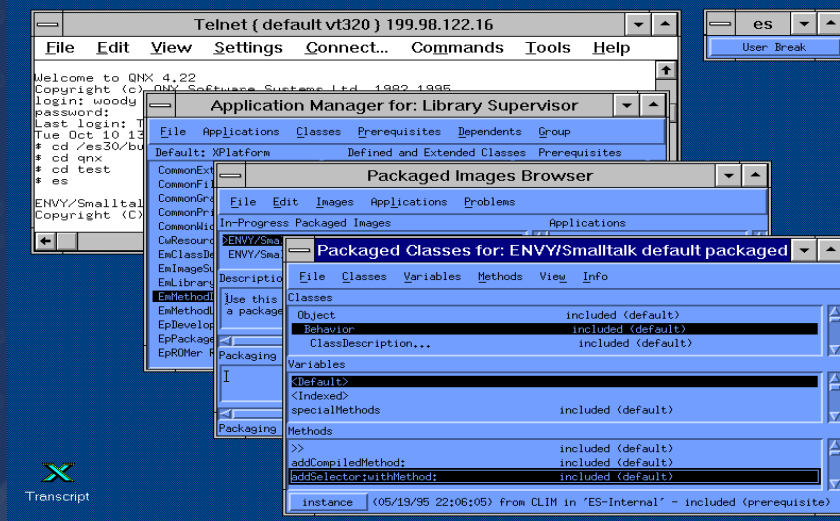


- **Two development alternatives:**
  - In-target tools provides ease of use but
    - requires GUI port and fast comms interface
    - not appropriate for small devices
  - C based cross development tools more flexible but
    - Debugging is difficult
    - Packaging/ROMing is hard
    - Sensitive to host/target mismatches
    - Cross-loading tedious and time consuming
- **Neither provides the productivity of workstation Java or Smalltalk**
  - In practice most users start developing on workstations and port late to target
  - Solution must combine best aspects of both alternatives PLUS ease of use

# Cross-Development IDE from 10,000 Feet



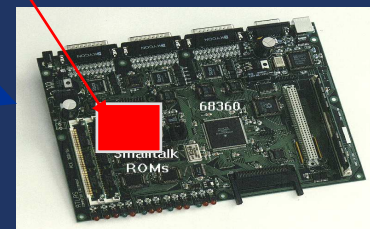
## Development Environment (Workstation)



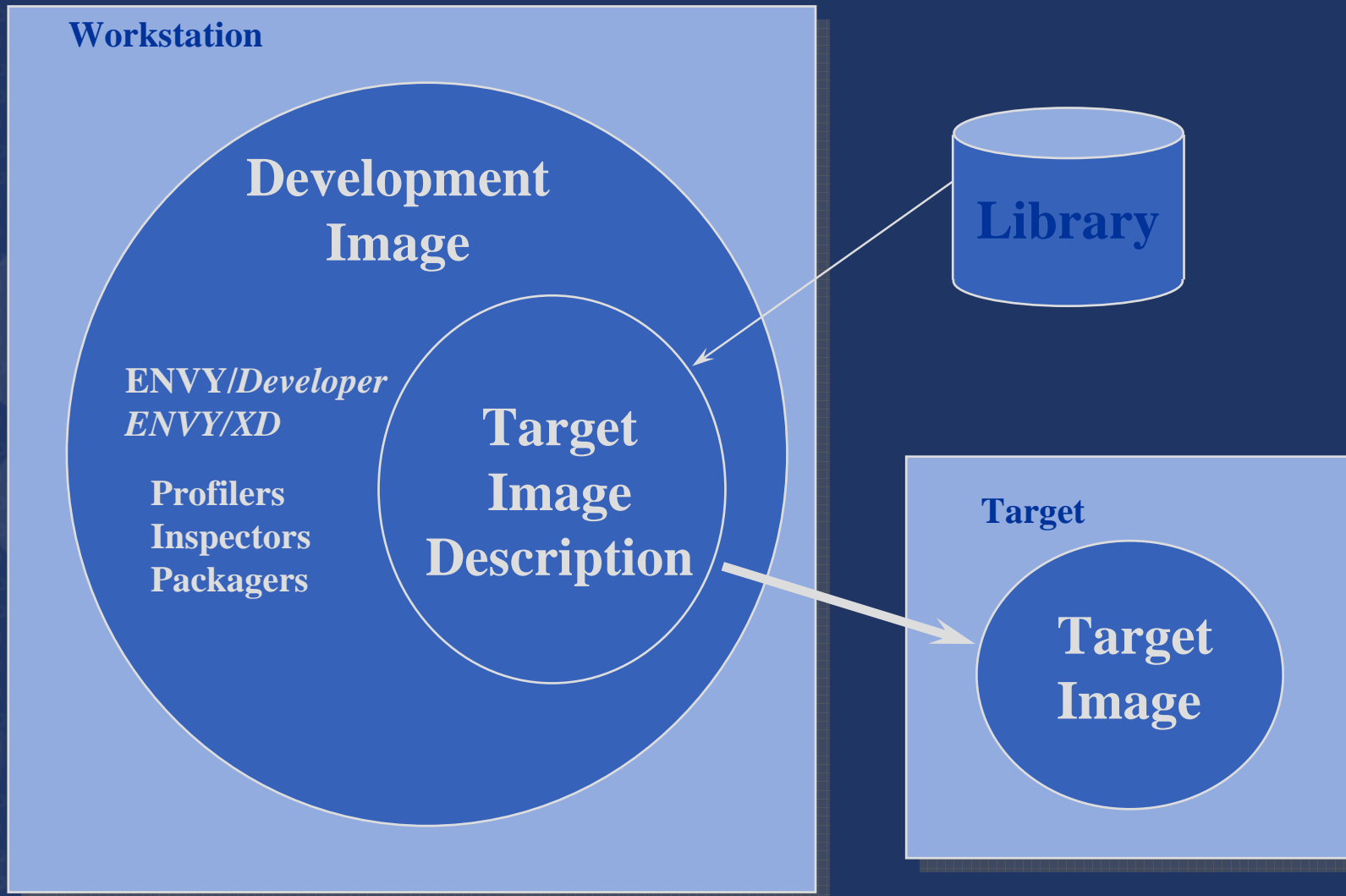
Development  
Runtime  
Support

Target  
System

Hot-replace  
Control  
Debug



# Inside View



# Measuring the Process Fundamentals



- What do we want to measure?
  - Efficiency: are resources being optimally deployed?
  - Progress: is the project on track for time and budget?
  - Productivity: how much code per unit of labor?
  - Rhythm: “heartbeat”, how active is the project day to day?
  - Quality: how good is the software being produced?
  - Testing: what is the current state and trend of testing?
- “Heisenberg Uncertainty Principle of Software Physics”
- Once management starts collecting and publishing a particular metric it will affect the value, and change team behavior
  - It will also change the implicit values of other (unpublished) metrics because effort will be redeployed
  - Example: “Clean Room” experiments at IBM Federal Systems

# Final Thoughts



- Too often R&D agendas are driven by yesterday's problems instead of tomorrow's challenges
  - The result is technological Maginot Lines
- A bolder approach is more likely to succeed
- Adopt a simple yet powerful hypothesis and allow it to drive the research:
  - the Smalltalk project at Xerox PARC
  - Doug Engelbart's Augment
  - Amoeba
- If your bet is even close to right there will be huge payoffs
  - And if you're wrong you will at least have fun

# Current Approaches Paper Over the Cracks



- **Attempt to solve the problem with a thin veneer of usability**
  - Wizards, templates, visual tools, generators
  - Superficially hides details like SOAP, Java, WSDL, XML, ....
  - A lot of effort mapping between various models and levels
- **The strategy works until users are forced to open the box**
  - In practice users always need to understand, debug, and modify generated code
- **In the end, users must know MORE, not less, with this approach!!**
  - Need to understand both the AF stack and all the helper tools
- **The complexity is accidental but the problem is essential**
  - The flaws are inherent in the paradigm
  - Can't be fixed by fiddling at the edges

# Solutions for a mobile world?



- **We need a ubiquitous platform**
  - Designed for portability across a wide range of devices
  - A few powerful concepts and operations, not many weak ones
  - Dramatically simplifies application writing
- **Keep assumptions simple and small in number**
  - original VA/Smalltalk VM assumed less than 12 basic platform functions (clock, memory, interrupts)
  - single source stream targeted over 30 platforms
- **Most functions supplied by dynamically (un)installable modules**
  - Don't assume you can reboot!
  - E.g. install file i/o support on demand when storage device is detected/accessed
- **Admit full range of optimization technologies: static, dynamic, hybrid/combined**
- **Provide support for richer features like dynamic dispatch, lexical closures, multi-methods, proxies etc. through a pay by use model**
  - Don't pay for what you don't need or want
  - Everyday programs will need to be aware of the platform context
  - Suggests reflection will become a "normal" part of programming
- **Built-in configuration and dependency management**

# References



“Prototyping a Real-Time Embedded System in Smalltalk”, Brian Barry, OOPSLA‘89 Proceedings

“Using Objects to Design and Build Radar ESM Systems”, Brian M. Barry, D.A. Thomas, J.R. Altoft and M. Wilson, OOPSLA‘87 Proceedings

“Using Active Objects for Structuring Service Oriented Architectures”, Dave Thomas and Brian Barry, Journal of Object Technology 2(7):7-14, July-August 2004

“New Complexities in the Embedded World – The OTI Approach”, Kim Clohessy, Brian M. Barry and Peter Tanner, Lecture Notes in Computer Science 1357 Springer 1998 Pp.472-481

“Development of Reusable Test Equipment Software Using Smalltalk and C”, Alan Dotts and Don Birkley, Addendum to OOPSLA‘92 Proceedings

“The MMST Computer Integrated Manufacturing System: An Overview”, McGehee, J., Hebley, J., and Phauth, M., Texas Instruments Technical Journal, September/October 1992.