Transition to OO?

I'm going to ignore the part about "Should we?"

A summary of

- Our approach to establishing an OO development project
- Practical experience, successes and failures
- Issues we've had to consider

Outline:

- The BaBar situation
- Context of OO introduction
- People, our most important resource
- Process

Disclaimers:

- I do event reconstruction, the primary OO part of BaBar so far. Other parts of the overall computing system are evolving in different ways.
- These are my own opinions, not necessarily those of BaBar leadership.

CP Violation Studies at the SLAC BFactory

New Detector:

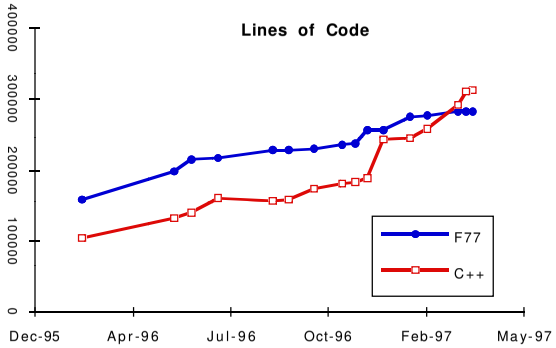
- ¥Several novel detector systems
- ¥High event and data rates
- ¥First physics data in Spring 1999

New offline software:

- ¥Generally built from scratch
- ¥Decision to use C++ in 1994
- ¥Development effort really started in Fall 1995
  - Simulation code generally older, mostly FORTRAN
  - Reconstruction code almost exclusively C++

Current status:

- ¥About 30 contributing developers
- ¥Initial working versions of most systems
  - Reconstructing data for Mock Data Challenge
- ¥About 1600 classes in 60 packages, 300KLOC



Related Talks at CHEP97

Architecture of the BaBar Reconstruction System: Practical Issues

(Tuesday 10:30 P5)

Panel Discussion: Transition to OO?

(Wednesday AM)

The BaBar distributed development / centralized release system

(Wednesday 18:00 P3)

Applying Object-Oriented Software Engineering at the BaBar Collaboration

(Thursday 14:00 P3)

An object-oriented tracking system for BaBar

(Thursday 14:30 P5)

An object-oriented extended Kalman filter tracking algorithm

(Thursday 16:00 P5)

Beta: A high level toolkit for physics analysis

(Thursday 17:00 P5)

(Two event pictures to show this stuff works)

(unfortunately, the pretty color pictures are printed directly from the display hardware, so we don't yet have them on the web)

What are you transitioning from?

Traditional methods

¥FORTRAN77

perhaps with VAX extensions

¥ Extensions to add functionality and control

ZEBRA/BOS/...

ADAMO

¥Code management tools

HISTORIAN, homegrown tools

¥Standards, practices, policies

Common lore of the HEP programming community

Locally developed, customized, documented

¥Programmer skill, commitment and ingenuity

These work!

¥ No experiment has ever failed due to software

J. Dorfan, et al

But many people think its harder than it needs to be

What changes in the transition to OO/C++?

	Better/worse?
¥FORTRAN77	**Better**
perhaps with VAX extensions	
¥ Extensions to add functionality and control	
ZEBRA/BOS/...	some native,
ADAMO	some missing
¥Code management tools	
HISTORIAN, homegrown tools	easier, but needed development
¥Standards, practices, policies	
Common lore of the HEP programming community	missing
Design idioms and normal practices	missing
Locally developed, customized, documented	missing
¥Programmer skill, commitment and ingenuity	**Much Worse**

ÒIf you expect a language to solve all your problems, you donÕt have interesting problemsÓ A. Koenig

Obstacles as of 1995

Nothing to build on

- ¥Lots to do
- ¥Too many possibilities for the first pieces

Inverted schedule

- ¥Analysis and design are high skill activities
- ¥But have to come at the beginning, before skills are developed

No clear agreement on product

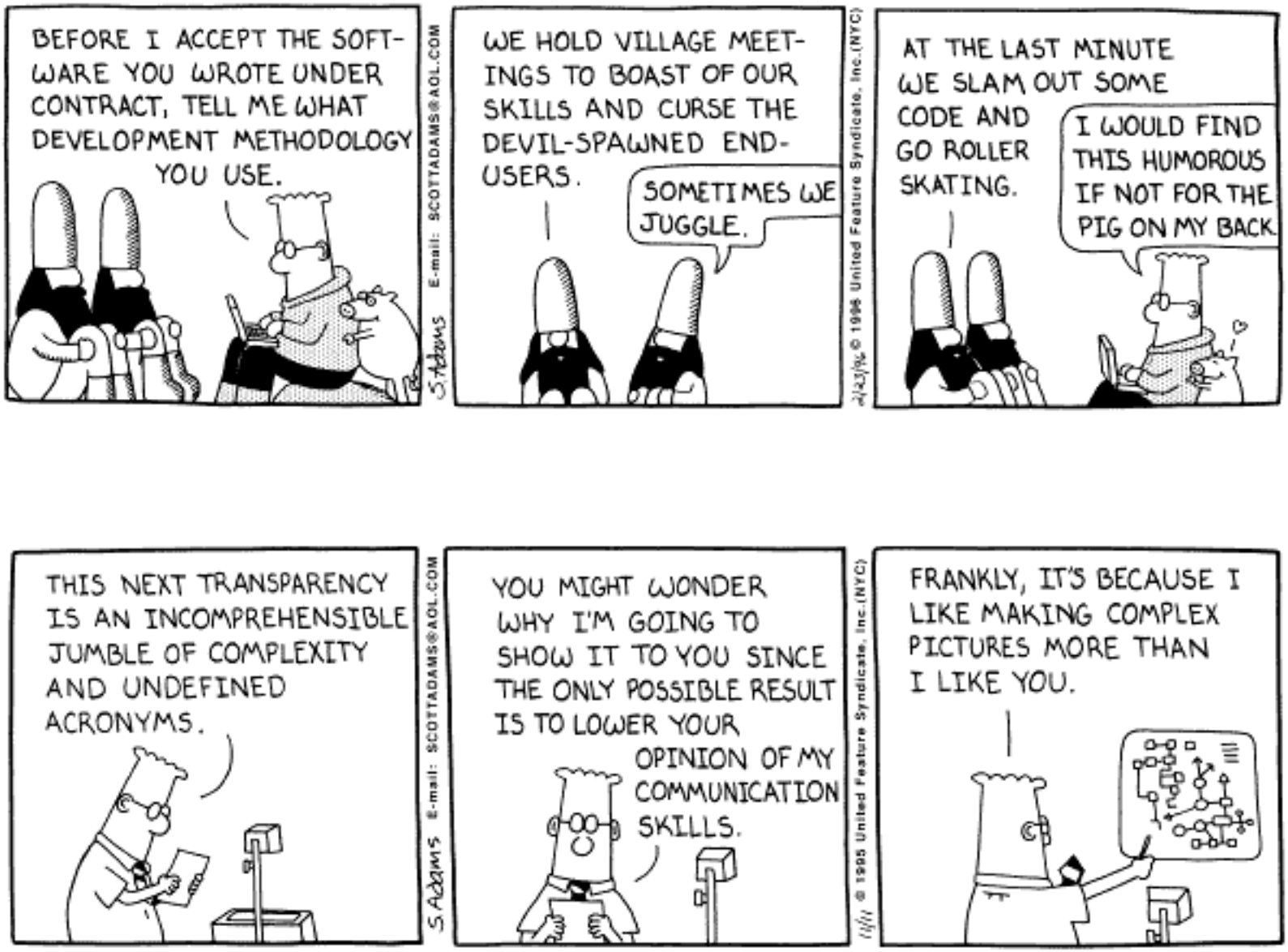
- ¥Everybody knows what a track finder does, but few agree
- ¥Real product is flexibility
  - Waiting for that smart idea in 2003

Not much expertise & effort available

- ¥Much existing expertise of doubtful applicablity
- ¥C++ advocates had limited design experience
- ¥Mismatch between enthusiasm and effectiveness

I expect these are common issues beyond BaBar

A clash of cultures over “methodology”:



Copyright © 1995 United Feature Syndicate, Inc.
Redistribution in whole or in part prohibited

Multi-prong attack

Architecture

- ¥Solving the too many first choices problem
- ¥OO design expressing traditional concepts

Iterative design & implementation process

- ¥Applying evolutionary pressure
  - Design and implementation intertwined
- ¥Strike multiple balances
  - Learn by doing, do while learning
  - Getting something in place vs. design work
- ¥In spite of imposed waterfall-model schedule

Gain control of the process by controlling the product

- ¥Code management
- ¥Quality control and assurance

Team-building

- ¥Getting the people
- ¥Formal training balancing experience and exposure

People

Total number of people who have contributed is still small

¥Authors of changes to CVS repository:

35 - 40 in C++

30 - 35 in FORTRAN

20 - 25 appear in both lists

July 96:

21 - 24 in C++

30 - 35 in FORTRAN

15 - 20 appear in both lists

¥Top 11 make 80% of C++ changes

(Finally larger than FORTRAN)

Top 6 make 80% of C++ changes

BaBar use of OOD was a mixed blessing at first

¥> 90% of the collaboration wouldn t touch it

¥But it generated significant enthusiasm in few people

Numbers are finally increasing

¥Mostly newcomers (students, post-docs)

This is going to be close!

Why is recruiting so hard? I

Two problems at entry level:

¥OO Beginners write bad code

¥And we don t have enough of them

New programming skills needed

¥Particularly hard if you feel you re a FORTRAN expert

Nobody likes to make public mistakes

¥Significant resistance in general populace to investing in software skills

First efforts are hard, not very useful, but you have to go through them

Wish we had resources to help everybody

¥Handholding

¥Improving ease of use, documentation, etc.

But we have to concentrate those on the known developers

Why is this so hard? II

Design is yet another set of skills

¥ You want me to learn two languages?

¥Have to fight the code written == job done mindset

Next problems:

¥Very hard to motivate a collaborative approach to development

It is easier to write an incorrect program than understand a correct one.

Alan Perlis, SIGPLAN Notices, September 82.

Multiple causes, mostly related to sparsity of experienced contributors

¥Second system effect

Some very baroque products being created

Look at how much I can punch out!

¥Hard transition to design, as opposed to faster coding

Few have experience with explicit design, OO methodology

Have to respect the way people work:

¥ Need to understand what everything does and when

¥ Can t think without details from coding

¥ Here it is, hope you like it a.k.a. Coder in a corner

¥ STL purists (or any other flavor)

Capability Maturity Model - levels of process quality

5) Optimizing level

4) Managed level

3) Defined process level

¥ Planned introductions to the repeatable process

2) Repeatable level

¥Disciplined process, with an understood sequence of steps

¥Basic project management, configuration management, QA in place

1) Ad hoc level

¥Organization s activity is primarily reactive

¥Programmers code and test,then assemble the system

¥No forceful management in place

Always keep the goal in mind

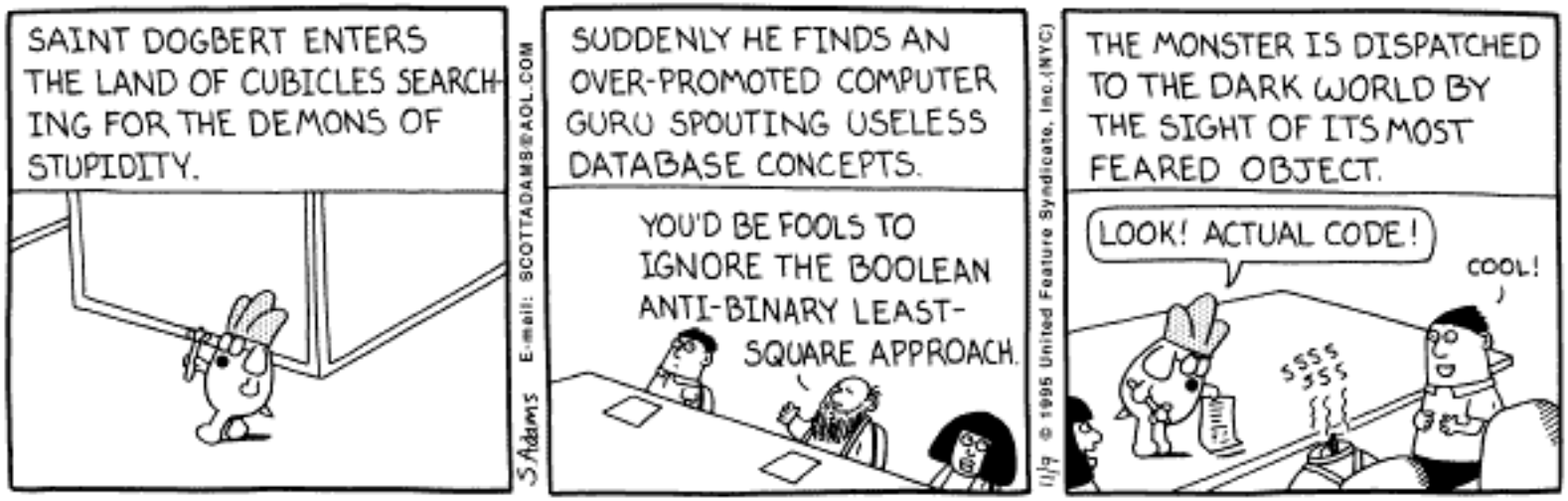
Timescales:

- ¥We re going to be using this code for a decade
- ¥We re going to be doing physics in two years
- ¥Mock Data Challenges keep each year on track

To get a system built, put effort into building it <u>now</u>

- ¥Debates are generally only resolved by experience
- ¥Waiting for things to get better is not productive

Quality of <u>actual common code</u> is final arbiter



Copyright 3 1995 United Feature Syndicate, Inc.
Redistribution in whole or in part prohibited

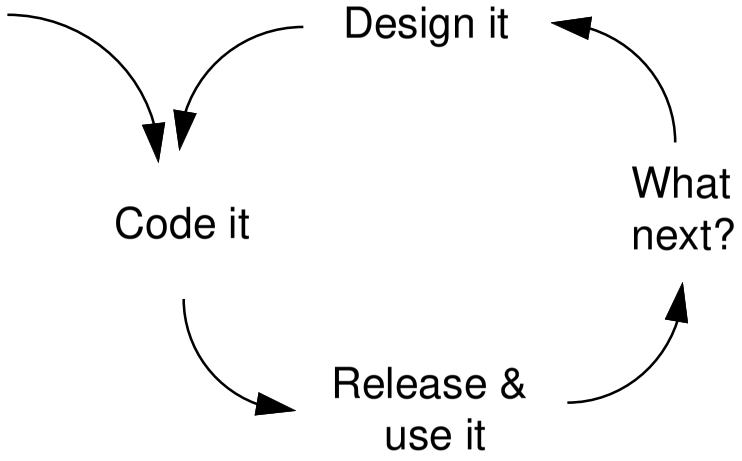
Our “design process”

We use both of those methodologies

- ¥People enter coding
- ¥Eventually, they start to draw clouds and blobs
- ¥Policy: Get them engaged, then work with them

Evolution improves the system

- ¥Relevant code is used
  - Comments are not always gentle
- ¥Release system controls the pace
  - Monthly/biweekly timescale
- ¥New designs, redesigns are ongoing
  - Driven by perceived needs



Documentation

- ¥Little formal documentation
  - A clear weak point
- ¥Informal, ongoing documentation base
  - Captured notes, slides, etc

How do you know it's working, when you don't know what it does?

Where the requirements documents?

- What do you put there besides platitudes?

Need to avoid very ugly surprises when data arrives in 1999

- Scheduled two Mock Data challenges for summer 97, 98
  - Double-blind reconstruction and analysis of selected channels
- Don t need complete code
- But won t be able to kid ourselves about missing bits

Copyright © 1995 United Feature Syndicate, Inc.
Redistribution in whole or in part prohibited

Biggest failure:

We haven't figured out how to work together

Some successful small efforts

- ¥CLHEP cooperative development
- ¥Shared code & tools with CDF, Atlas

But even these may start to diverge

Many of the effects seen in individuals appear between collaborations

- ¥We all expect the other guy to be the boson

We promised improvements in this area, and we have to live up to that

You need developers

¥People need a realistic migration path too

¥They need to feel productive

¥They need to be productive

¥Their skills need to be upgraded in place

Beware of fooling yourself about your progress

¥ Process is a means, not an end

¥What have prototypes bought you?

¥How do you know what s working?

¥Are we all building the same 30% of the solution?

¥Will you be able to finish what you intend to replace?

Nothing replaces experience

¥"Study it forever and you'll still wonder. Fly it once and you'll know." - Henry Spencer