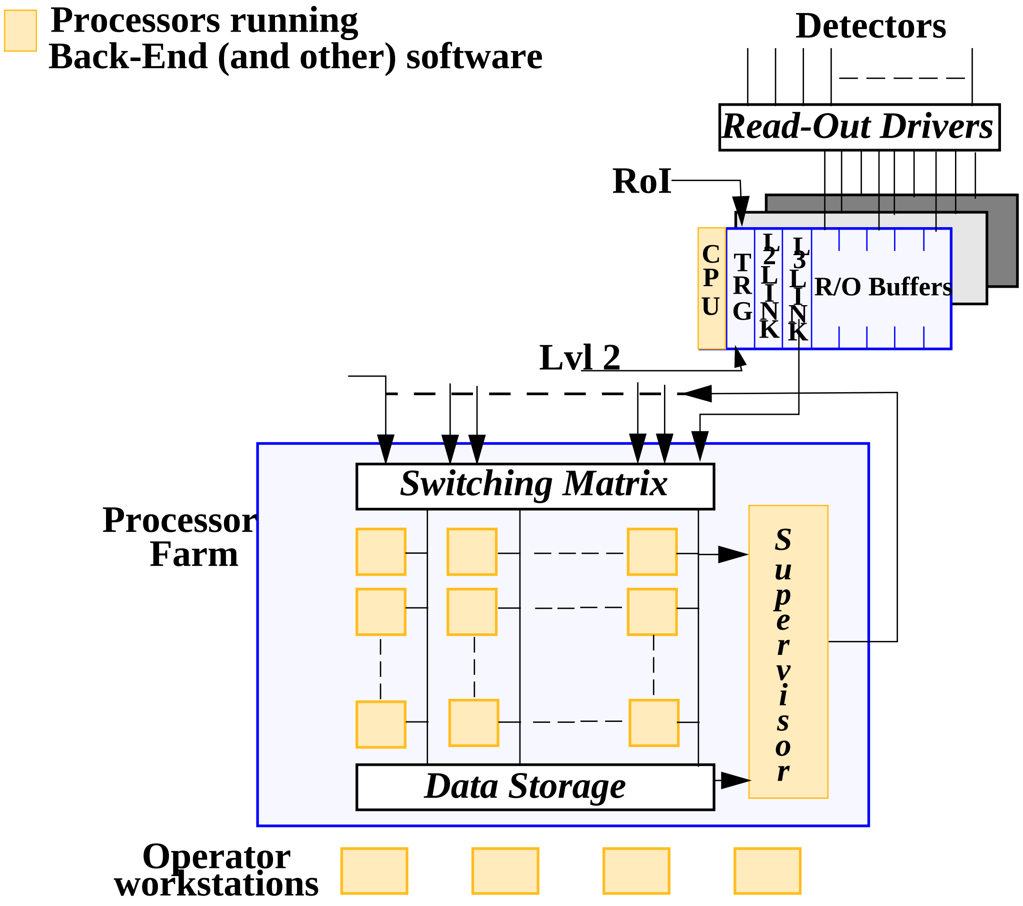
Back-end DAQ software process

Bob Jones (Robert.Jones@cern.ch)

- Introduction
  - what is the back-end DAQ?
  - what do they know about software process anyway?
  - is it the same as the Atlas Software Process?
- Process Overview
  - organisation
  - phases and deliverables
  - inspections
- Summary
  - status
  - things we might like to improve
  - some do’s and don’t s

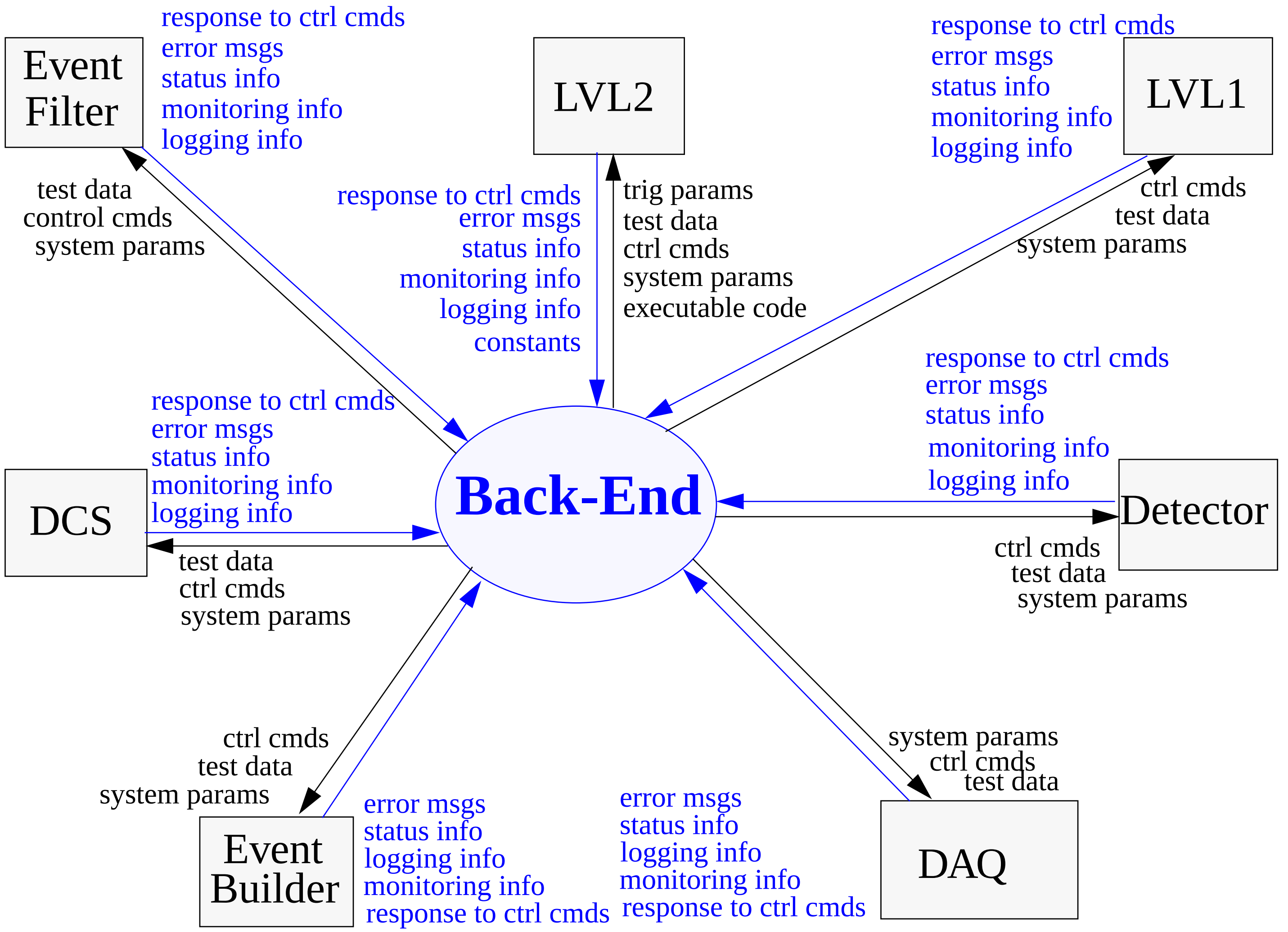
ATLAS Back-End DAQ

software for configuring, controlling and monitoring the DAQ
excludes management/processing/transportation of physics data



The back-end talks to all other online systems
It is the "glue" of the online

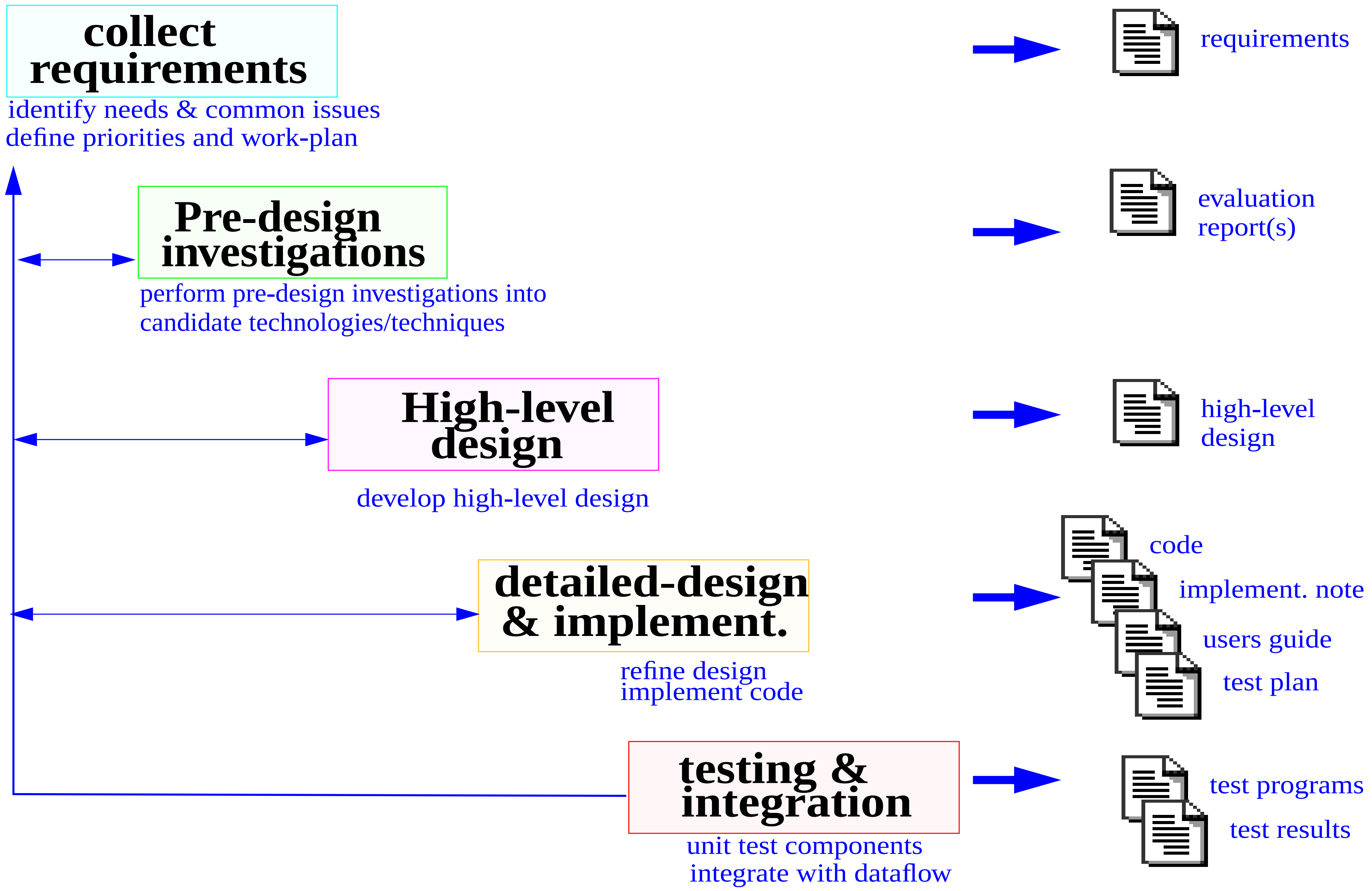
Back-End integration in ATLAS online



Is it the same as the Atlas Software Process?

- What do they know about software process anyway?
  - We are not Gurus - *just concerned developers like you*
  - Based on what we could find in the text books and could apply
  - Seen as a **best effort** approach - *not perfect but will do for now*
- Many similarities with ASP
  - domains <=> components
  - use same techniques (e.g. OMT) and tools (e.g. StP, SRT)
  - basic phases and organisation
- A few differences
  - advantage of being a smaller, more closely integrated community (8 institutes, ~15 individuals)
  - not formally described in a document - *just web pages*
  - inspections are more "human"
  - more detail on testing procedures
  - so far all Back-end software has followed the basic process
- We are providing input to the ASP
  - a sort of *ASP-Lite*

Back-end Software Process



Phases

Reviewed Deliverables

collect requirements

identify needs & common issues
define priorities and work-plan

→ requirements

Pre-design investigations

perform pre-design investigations into candidate technologies/techniques

→ evaluation report(s)

High-level design

develop high-level design

→ high-level design

detailed-design & implement.

refine design
implement code

→ code
implement. note
users guide
test plan

testing & integration

unit test components
integrate with dataflow

→ test programs
test results

Back-End components

Run control	controls DAQ configuration and data taking operations
Configuration databases	define all aspects of the DAQ configuration
Message reporting system	report/capture of information messages
Information service	general purpose information exchange
Process manager	basic job control of programs
Partition/resource manager	allows concurrent data taking activity
Status display	shows current status of data taking to the shift operator
Online bookkeeper	electronic tape log book
Test manager	bank of functionality tests for DAQ components
Diagnostics package	uses tests held in the test manager to diagnose problems
Event dump	access to sampled data for analysis and quality checking

core

TDAQ &
detector

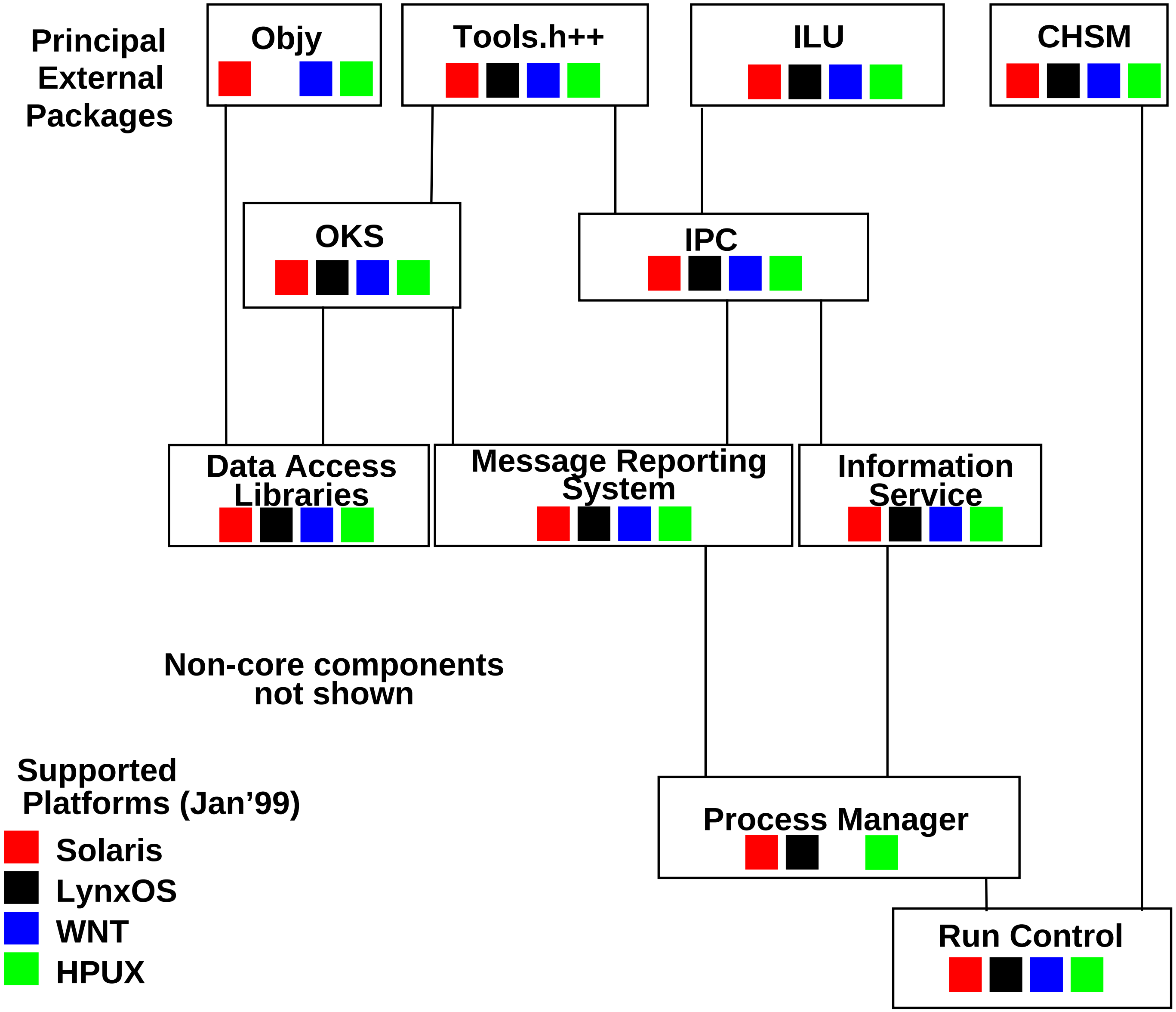
Organisation

- Work organised around components
  - small group dedicated to each component (upto 4 developers)
  - one co-ordinator for each component
  - prefer one institute per component - *clear boundaries of responsibility*
  - most developers follow a component all the way through its lifecycle
  - look for commonality between components - *don't duplicate functionality*

- Components developed according to agreed priority
  - started with core components (e.g. run control and config. databases)
  - now working on TDAQ components (e.g. Online Book-keeper)

- Component independent parts
  - Software management (i.e. use of SRT, CVS, AFS etc.)
  - Use of external software (Corba/ILU, Rogue Wave Tools.h++, CmdLine, CHSM, ACE) - *one developer responsible for each package*

Back-End components installation and dependencies



Principal External Packages

Objy 🟥 🟦 🟩

Tools.h++ 🟥 ⬛ 🟦 🟩

ILU 🟥 ⬛ 🟦 🟩

CHSM 🟥 ⬛ 🟦 🟩

OKS 🟥 ⬛ 🟦 🟩

IPC 🟥 ⬛ 🟦 🟩

Data Access Libraries 🟥 ⬛ 🟦 🟩

Message Reporting System 🟥 ⬛ 🟦 🟩

Information Service 🟥 ⬛ 🟦 🟩

Non-core components not shown

Process Manager 🟥 ⬛ 🟩

Run Control 🟥 ⬛ 🟦 🟩

Supported Platforms (Jan'99)

🟥 Solaris

⬛ LynxOS

🟦 WNT

🟩 HPUX

Inspections

- Purpose

  Improve the quality of components by assisting developers to recognise and fix faults at the earliest possible point in the development cycle

- General organisation

  Based on Tom Gilb's Software Inspection method

  Authors submit software/document to a small group of peers who review it and produce a list of comments which are given back to the authors.

  Moderator - person responsible for organising the review and collecting comments

  Producer(s) - authors of the software to be reviewed

  Reviewers - peers directly concerned by and aware of the work

  Reviewers are aided by check-lists covering issues and criteria for completeness and correctness.

  The focus of the inspection is on identifying problems, not resolving them

Inspection insights

- Prefer "real" to "virtual" meetings
  - improves team atmosphere and helps brainstorming
  - good way of training new-comers & extending knowledge of developers
  - best to start people as reviewers before they become authors
- Inspections are a lot of work
  - 3 to 4 peers work best
  - Typically need 1 kick-off, 1 logging and 1 follow-up meeting
  - Split large deliverables and assign one reviewer to each part
  - collect metrics and feedback on inspection process
- Code inspections
  - Authors must do a lot of preparation:
  - - documentation
  - - coding rules and CodeCheck tool from Spider project
  - - configuration management (SRT)
  - - testing tools (Logiscope, Insure++, Purify)

A lot of work but worth it: found faults in code and documents

Templates and guidelines used

- doc. templates developed within the project
  - generic technical note http://atddoc.cern.ch/Atlas/DaqSoft/sde/TechNote.fm
  - test plan http://atddoc.cern.ch/Atlas/DaqSoft/sde/TestPlanOutline.fm
  - test report http://atddoc.cern.ch/Atlas/DaqSoft/sde/TestReportOutline.fm
- doc. templates taken from the IT/IPT group
  - user requirements http://www.cern.ch/CERN/Divisions/ECP/IPT/DocSys/PSS05/
  - users guide http://framemaker.cern.ch/GuideTemplates/)
- check-lists and guidelines
  - brief requirements, design and general documentation check-lists
  - Spider C++ coding standards
  - Short, easy-to-read ideas for design and testing by Guru’s on the web
  - Simple "how-to" instructions for most commonly used tools (e.g. SRT)

Summary

- Back-end software

  covers 11 components (~150,000 lines C++):

  - 6 tested and integrated

  - 2 implemented

  - 2 being designed

  - one left *(any takers?)*

  now concentrating on regular incremental releases of the software

- Back-end DAQ software process

  certainly not perfect - *but perhaps the best we can do now*

  improving all the time:

  - put more order in the detailed design/implementation phase

  - improve software distribution and management tools

  - simpler doc. templates

Building Software Releases

- Each one should be better than the last

  incremental/evolutionary

  implies sufficient unit testing - *use SRT's* make check *target*

  One per month - *coincides with Back-end meetings*

  Status of last release and contents of the next are discussed in the meeting

- More platforms == more work

  implies every developer has access to every platform

  keep the list of supported platforms as small as possible

  should be rationalised across sub-system(s) / online / atlas

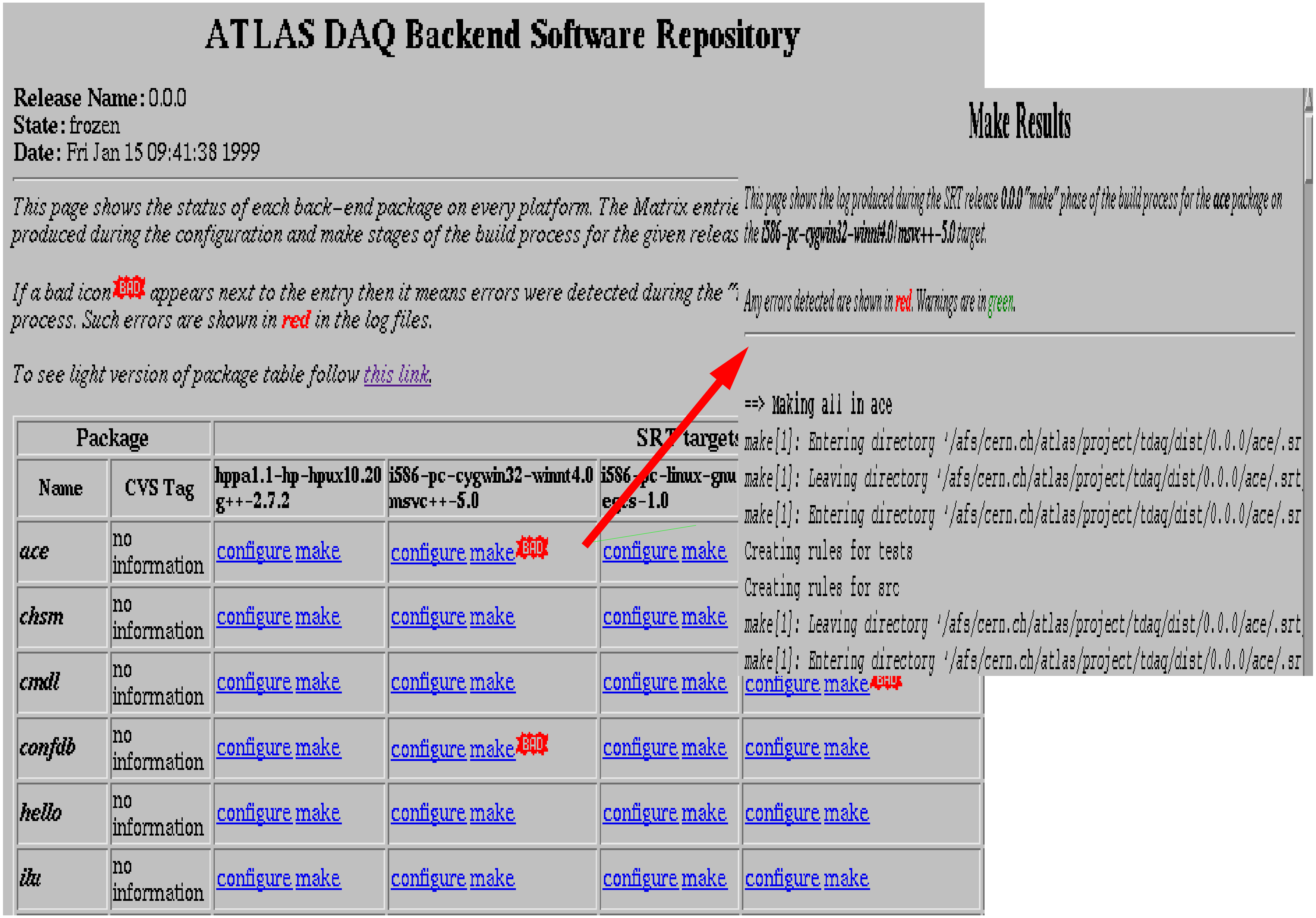
- Software librarian != developer

  He/She is not there to fix faults in the software

  Have a web page to show log of build for each component

Building a release is an important milestone but represents a lot of work for everybody

Release Information



ATLAS DAQ Backend Software Repository

Release Name: 0.0.0
State: frozen
Date: Fri Jan 15 09:41:38 1999

This page shows the status of each back-end package on every platform. The Matrix entrie produced during the configuration and make stages of the build process for the given releas

If a bad icon BAD *appears next to the entry then it means errors were detected during the " process. Such errors are shown in **red** in the log files.*

To see light version of package table follow this link.

Package		SRT targets			
Name	CVS Tag	hppa1.1-hp-hpux10.20 g++-2.7.2	i586-pc-cygwin32-winnt4.0 msvc++-5.0	i586-pc-linux-gnu egcs-1.0	
ace	no information	configure make	configure make BAD	configure make	
chsm	no information	configure make	configure make	configure make	
cmdl	no information	configure make	configure make	configure make	configure make BAD
confdb	no information	configure make	configure make BAD	configure make	configure make
hello	no information	configure make	configure make	configure make	configure make
ilu	no information	configure make	configure make	configure make	configure make

Make Results

*This page shows the log produced during the SRT release **0.0.0** "make" phase of the build process for the **ace** package on the **i586-pc-cygwin32-winnt4.0/msvc++-5.0** target.*

Any errors detected are shown in red. Warnings are in green.

```
==> Making all in ace
make[1]: Entering directory '/afs/cern.ch/atlas/project/tdaq/dist/0.0.0/ace/.sr
make[1]: Leaving directory '/afs/cern.ch/atlas/project/tdaq/dist/0.0.0/ace/.srt
make[1]: Entering directory '/afs/cern.ch/atlas/project/tdaq/dist/0.0.0/ace/.sr
Creating rules for tests
Creating rules for src
make[1]: Leaving directory '/afs/cern.ch/atlas/project/tdaq/dist/0.0.0/ace/.srt
make[1]: Entering directory '/afs/cern.ch/atlas/project/tdaq/dist/0.0.0/ace/.sr
```

Some do's and don't s

- Do's

  do start gently - *can't go from chaos to Nirvana in one step*

  do keep it simple and stick with it *(i.e don't abandon it half way through)*

  do inspect requirements, design, code, users guide

  do provide templates, checklists and examples for every deliverable

  do get a non-author to perform component testing

- Don't s

  don't burden developers unnecessarily *(e.g. paperwork)*

  don't ask developers to produce a deliverable unless you it will be used

  don't ask developers to do something which has not been tried before

  don't underestimate time and effort required for software management, integration and testing

  don't do distributed development if you can avoid it