ACS Workshop: Summary of course conclusions 2009-2014

Prepared by Matias Mora, based on ACS Course conclusions from the 5 latest Workshops:

- *2009, Valparaíso – 23 participants / 3 instructors*
- *2010, Antofagasta – 20 participants / 6 instructors*
- *2012, Asiago – 7 participants / 3+2 instructors*
- *2013, Socorro – 4 participants / 2+1 instructors*
- *2014, Sao Paulo – 9 participants / 2 instructors*

1. Things that have worked well on earlier experiences

The integration of the different components should be done gradually throughout the course, as opposed to concentrating it in the final hours of the last day. Even if development teams complete their component at different paces, partial integration with simulated components can be started early on. Having accurate IDL simulations of all components is critical for this to work. [2010+2012]

There are frequent confusions among the course participants regarding the functional requirements of their specific component. Basic unit tests could be provided at the beginning of the course, together with the IDL definitions, to make them comply to the intended requirements [2009]. Implementing basic test cases before the actual component functionality, and running it first against the IDL simulator has been identified to help a lot in this aspect; to facilitate the first step, provide a test CDB with all components deployed in simulation upfront [2010].

Mix participants from different organizations and geographic locations when defining the development teams. Working with people you do not know beforehand produces stronger team identities and better interaction between teams. [2009]

Advanced topics like the Alarm System have a steep learning curve and should only be charged on participants who have already some experience with basic component development. [2009]

Using continuous integration improves and simplifies the job of the ITS team. A Jenkins build and test server was set up to be triggered by changes in the code repository (through the corresponding plugin) and provide instant feedback to the development teams. Bugs could be rapidly corrected and teams were forced to keep their code up to date in the repository. Due to the automated pass/fail criteria developers impose on themselves a stricter discipline to maintain a stable repository. [2010+2012]

In large workshops (e.g. X participants and X instructors) it has been very useful to hold daily instructors coordination meetings. A complete analysis of the daily activities is done every evening, identifying problems and positive recommendations for the next day, or even major changes to the planned activities to allow better results. Instructors give daily morning feedback on the overall status to the teams (in the shape of a short summary and/or daily conclusions on the project twiki), based on the outcome of the meeting and their worklogs. [2010]

Effective presentations are better than many or long presentations. Technical and visual coherence between presentations is to be emphasized during the course preparation. Some examples during the presentations that are close to what has to be implemented during the practical session are helpful. [2010]

Use Eclipse or some other IDE for all groups and provide upfront configured plugins. This will largely prevent common programming language problems [2010+2012]. The Eclipse ACS plugin provided by the ACS community has been successfully used to configure Java development environments [2013].

Pay attention to providing good support documentation, i.e. ACS environment cheat-sheet, API documentation, VM configuration instructions. [2010]

For courses with few participants, projects with more components than teams can still be used by providing a full implementation of the missing component upfront. It has also been positive to count with a completely implemented component to use as an example. [2012]

Using a centralized git repository (instead of CVS or SVN) as version control system allowes to closer follow the concept “commit locally very often, but pushing upstream only working code”. Having distributed local repositories is also a more accurate representation of the geographic distribution intrinsec in many of the projects that use ACS. [2012]

A fresh VM installation using VirtualBox/VmWare and the supported CentOS version with standard update repositories is the recommended development platform. Details such as user accounts and IDE of choice setup should be prepared in advance. This has resulted in a very clean and up to date environment. In particular, VirtualBox has better cross-platform support for Linux, Windows and OSX hosts. [2012]

For any practical exercise, a complete working implementation was tested beforehand by the instructors. [2013]

Step-by-step code examples were provided in the code repository for xUnit, BACI properties and IDL simulators. In addition, existing “Hello World” examples were used to introduce basic component implementation guidelines. [2013]

The Lego telescope model is visually attractive for the development and integration process. [2010+2014]

Four days of practical course has been generally a good amount of time. Three days seems to be too short to cover all relevant topics. [2014]

2. Things that should be improved in the future

Improve/reimplement ACS examples. Less generic examples, and more on specific isolated use cases. Components accessing other components. Access to CharacteristicComponents and BACI Properties. Examples and documentation must be kept up to date and complete [2010+2012+2013+2014]. ACS examples in C++ do not contain good exception handling examples, causing confusion in development teams. New exception handling examples should be provided for each language. The C++ “Hello World” example should use the same IDL as the Java and Python versions [2013].

When interacting with hardware it is fundamental to test very well the final intended deployment. If running on a VM, the capabilities and options to interact with the host OS devices must be understood. In general it is recommended not to use VMs for hosts that need to control hardware devices. If providing simulators for hardware devices, try to be accurate in the behavior, e.g. operation delays. [2010]

Some IDLs in the Observatory project have methods that are never used by other components (e.g. Database component). This is a problem because they are never tested in practice, and because they require time to implemented, which is not essential for the final project goal. [2012]

Component unit testing is in this case language independent, as they execute the component and not the plain implementation. In principle, every group can decide what xUnit implementation to use. However, the main goal of the exercise should be the component implementation. Therefore, it is advisable to use the xUnit option that allows for quickest test cycles [2012]. In past experiences this has usually been PyUnit.

The TAT testing framework adds an unnecessary complexity to the process of running a unit test. A lot of time overhead was consumed by the environment start/stop routines and reference file setup, which would sometimes differ in the development machines and the build/test server. A simplified test environment should be provided, using command line scripts and the xUnit plugin for Jenkins. [2013]

3. Things that could be tried out in the future

Prepare complete IDL-level unit tests for all components beforehand and use them as test reference by the ITS team. The group would still have to implement their own basic tests, but functional requirements would be much clearer. [2010]

Introduce “interactive” talks, including trivial examples to be implemented and tested during the talk by the development teams. This would eventually improve the understanding of theoretical concepts. [2010]

Incorporate GUIs to see things that are happening with hardware devices or their simulation (e.g. CCD camera display, telescope simulation GUI). [2010]

Use some sort of online status report tool to show the progress of each team on a screen. Here they could report milestones (e.g. “Telescope is now able to get reference of Instrument”) [2010]. Alternatively, use an bugtracking (e.g. JIRA, trac, github) and/or agile tool (e.g. kanban board) to track the progress of milestones visually. This could also replace the worklogs, by writing progress and problems to the specific task.

Consider code generation to generate component implementation skeletons. However, this must be fully implemented for all supported languages first. [2010]

The course could be modularized. The theoretical depth and amount of hands-on exercise could be adjusted on-demand. Create hand-out documents from the current talks with all required theoretical packground for reference. This would allow to reduce even more the time spent on talks, and maximize the hands-on time. [2012]

4. Language-specific comments

C++ is always the most complicated and hard to understand programming language (even for experienced C++ programmers, some ACS and CORBA specific aspects are confusing); this has to be considered when assigning component requirements [2010]. A successful approach for using CharacteristicComponents was to provide a normal ACSComponent IDL at the beginning of the course, leaving the extension to a CharacteristicComponent to the development team; this allowed a successful gradual development without the complexity of handling BACI properties from day 1. [2013]

Python components are usually the first ones to be finished. As a scripting language it is fundamentally simpler, and hides maybe too much of the technical details from the developer. [2012]