Herschel Data Processing Lessons Learned (Moderator: B. Merín)

Overview:

The science ground segment considers that the **Herschel Data Processing** system is in the end a **successful system** and that both the software itself and the integrated development concept **should be reused** for future missions.

Key Points:

1. **Use a common data processing software framework for all instruments.** In the case of Herschel, all three instruments share approximately 75 % of the Data Processing code. This produces three times more testing, validation and improvement of that code as compared with what would be achieved with three individual developments.

2. **Involve external users in the definition of end user requirements AS EARLY AS POSSIBLE.** This is particularly important if the data processing software is going to be made available to the users. These external users could be identified among community interested in certain mission and should be funded to support the mission development by attending in-persona meetings and scrum sessions, and reading and updating user-requirement documents.

3. **Involve all stakeholders and communicate the rationale for the election of the programming language.** This is the single most important decision in data processing. Our experience shows that it is essentially impossible to choose a programming language that is guaranteed to be a good election in 10-20 years. So the recommendation is to go for a professional language that scales well, allows easy integration and automatic testing of the code. This decision should come from a wide consensus including all relevant groups and its rationale should be intensively communicated to the science ground segment to engage the developer community with it.

4. **Define different user requirements, documentation, testing needs and development timelines for internal and external software.** In the early phase of the mission, the data processing software should serve internal project experts, it should be flexible and it does not need such a high level of testing or documentation. A broader scope in requirements, testing and documentation needs should be applied to software given to the user community and prioritized during operations.

5. **Use industry or community standards for data-bases or analysis tools.** Give priority to the software development which is unique to the project, i.e. to pipelines specially suited to correct the special instrumental features for a particular mission. Re-use and/or link to existing external industry or community data-bases or analysis tools. A general tool development should only be done at levels higher than a single mission.

More expansive notes

Pre-launch activities (from 10 years before launch until launch) - preparation phase

- **Create an external user group with active scientists from the community interested in the mission.** They can be asked to help define global end-user requirements based on the mission scenario documents and to test early prototypes of the software.
- **Decide on the programming language to be used throughout the development based on a wide consensus.** This decision should be based on the inputs of consulting with all major stakeholders, including instrument, software development, archive experts and external users.
- **Communicate the rationale for the selection of the programming language and provide training on it to the developer community.**
- **Define, communicate and train the developers on a s/w development process with a set of code quality requirements (code guide-style, documentation and test coverage requirements).**
- **Let development focus on supporting the Instrument Level Tests needs.** This implies providing code to access the instrument data and make simple analysis but weighting flexibility over code quality, documentation and/or testing.

Launch, commissioning and performance verification activities – early production

- **Revise user requirements with inputs from inner and outer communities after checking real S/C data.** This set of requirements should be maintained by the data processing users group, should be communicated to all software developers at all times and should constantly evolve with the changing user-base. Such a set of low-level user requirements should be the base for the software tests and user documentation.
- **Invite external user community to the software development.** Communicate to the user community the coding guide-style and

principles so that they can develop their own applications within the project software framework. Groups with substantial observing time could be requested to contribute to software advice user groups.

During normal operations – production phase

- **Keep frequent communication with feeder CCB chairs and s/w developers to try to steer the development towards satisfying global user needs.** Controlling the actual development in each major software release is unpractical due to the large number of tickets going into them. However, weekly or bi-weekly checks on tickets at the different Configuration Control Boards might help focussing the development in the most critical operational and/or visible areas to the community.
- **Organize yearly developer meetings to promote the sharing of knowledge, coordination of development and team cohesion.** All developers in the project appreciated these meetings very much and often large prioritization changes were agreed at them.
- **Open a channel to get new user feedback into the development process.** Means for achieving this could be a partially open helpdesk system for data processing questions, a public bug-reporting system, organizing telecons with large user consortia or running on-line surveys. The input coming from this channel should be processed by internal project experts and used to update the user requirements.
- **Increase testing and documentation to focus on external users.** This essentially requires balancing resources towards more user and automatic regression testing and user documentation of the software. Eventually, software release time-scales could also be decoupled from the operational software to be able to go deeper on the testing and documentation work for some particular type of data.

During post-operations – close-out phase

- **Focus development on producing the best possible archival products.** This means that improvements on the framework should be under-prioritized at this stage over detailed data-curing tools or algorithms.
- **Expand automatic regression testing on operational and user s/w.**
- **Prepare to adapt the software to make it compatible with new missions or trends.** New software tools/concepts and missions will be coming up and the data processing software should be adapted where possible to link to them easily so that users can continue using it while transitioning to the new systems.