Software Project Management

School on transferable skills: managing, entrepreneurial and IPR

June 4, 2025

Facultat de Física, Universitat de Barcelona

Francesc Vilardell Sallés

My professional background

- PhD in Astrophysics from the University of Barcelona some time ago

- For the last 20 years I have been professionally:
  - Working with telescopes and other ground-based facilities
  - Developing software
  - Participating in some ESA missions: mainly Ariel, but also Gaia and Plato
  - Doing some teaching

- Currently working as senior engineer for space traffic management at GMV

Index

1. Introduction
2. Management in space software projects
3. Management in open-source software projects
4. Summary

Index

1. Introduction
2. Management in space software projects
3. Management in open-source software projects
4. Summary

Definitions and scope

- Software project management:
  - **Software.** Programs and other operating information that instruct the execution of a computer
  - **Project.** Planned set of interrelated tasks to be executed over a fixed period of within certain cost and other limitations
  - **Management**. Achieving goals in a way that makes the best use of all resources

- Management is used **everywhere**:
  - To develop the scripts for your PhD thesis
  - To organize a party or a dinner
  - To enable your presence here: transportation, schedule, cost,…

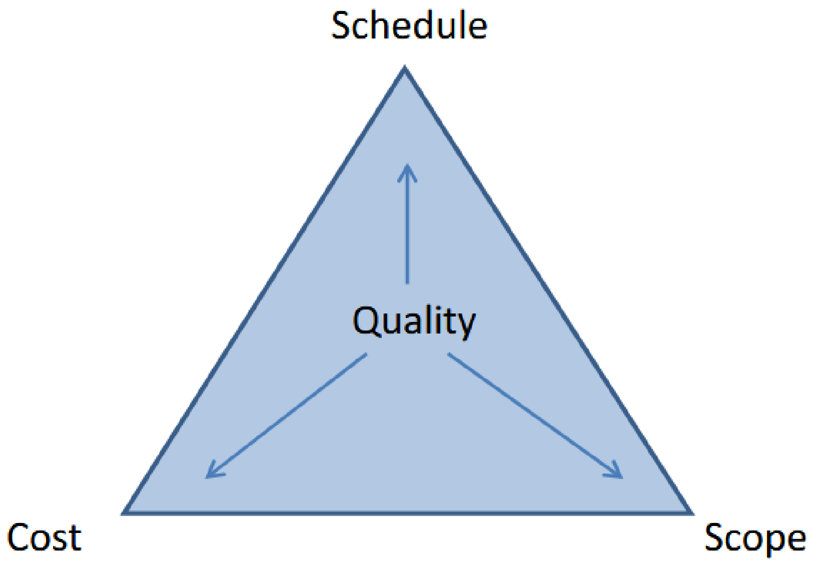
- Software project management refers to **working with people**

- **Scope.** Software project management **for science and space applications**

Purpose

Software project management is needed for several aspects:

- **Efficiency.** Ensure that projects are completed on time, within budget and with optimal resource utilization
- **Quality assurance.** Ensures high quality results by following standards and best practices



Purpose

Software project management is needed for several aspects:

- **Risk management.** Identify and mitigate the risks, ensuring smooth progress throughout the project lifecycle
- **Communication and collaboration.** Enhance teamwork and stakeholder engagement with clear objectives, roles and responsibilities
- **Resource optimization.** Maximizes efficiency by carefully allocating resources such as time, money, and manpower.
- **Decision making.** Provides a framework for informed decision-making, aligning with project goals and objectives
- **Adaptability.** Enables teams to adapt to changes and challenges effectively, ensuring project success in dynamic environments
- **Strategic alignment.** Helps aligning project activities with organizational objectives

Software project management allows **achieving amazing technical goals** that would be impossible otherwise

Key roles

Depend on the project management system. The **bare minimum** are:

- **Project Manager / Office.** The person or team responsible for planning, executing, monitoring, and closing the project. They ensure that the project meets its goals and objectives.

- **Team members.** Individuals who perform the tasks required to complete the project. This includes developers, testers, designers, and other specialists.

- **Stakeholders.** Anyone with an interest in the project. This can include clients, end-users, sponsors, and other team/institution members.

Team management in software projects

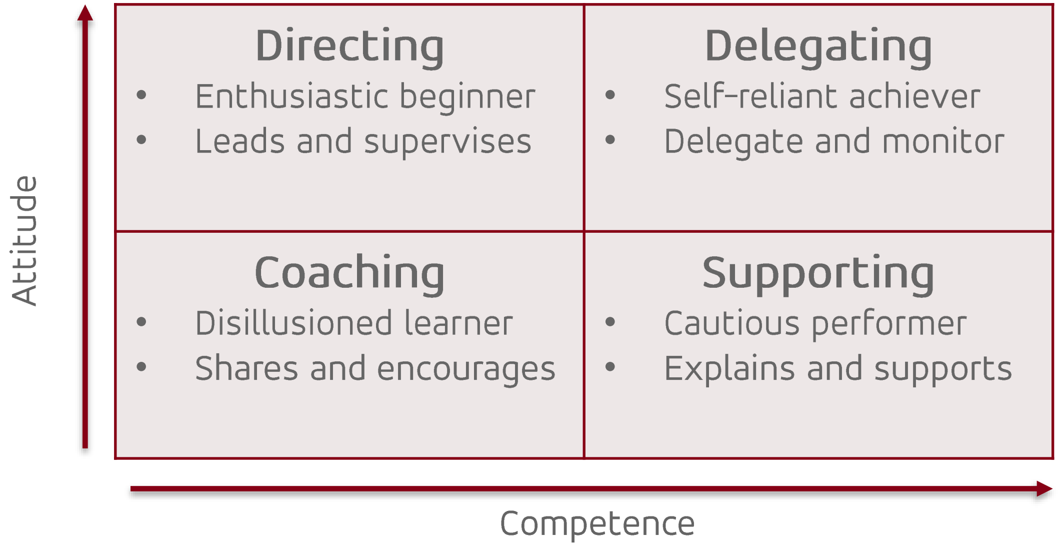
- The project team, including the project manager, are **the most valuable assets,** especially in software projects

- Responsibilities and project goals should be clearly defined → share the proposal with the entire team

- Project meetings should:
  - Have clear agendas and scope
  - Foster participation
  - Have clear actions and deadlines → an email or text document can be used to deliver minutes

Team management in software projects

- The project manager or team manager should:
  - Know the team and place each member to the proper position in the project
  - Adapt the leadership to the different capacities of the team (Situational Leadership, Hersey & Blanchard)
  - Multiply team members' capacities (Multipliers, Liz Wiseman)



Steps in Software Projects

- **Planning.** Define the project execution strategy:
  - Define the tasks to be done (scope)
  - Define schedule and costs (including effort)
  - Identify risks
  - In scientific and commercial projects, this should be done **when writing the project proposal**

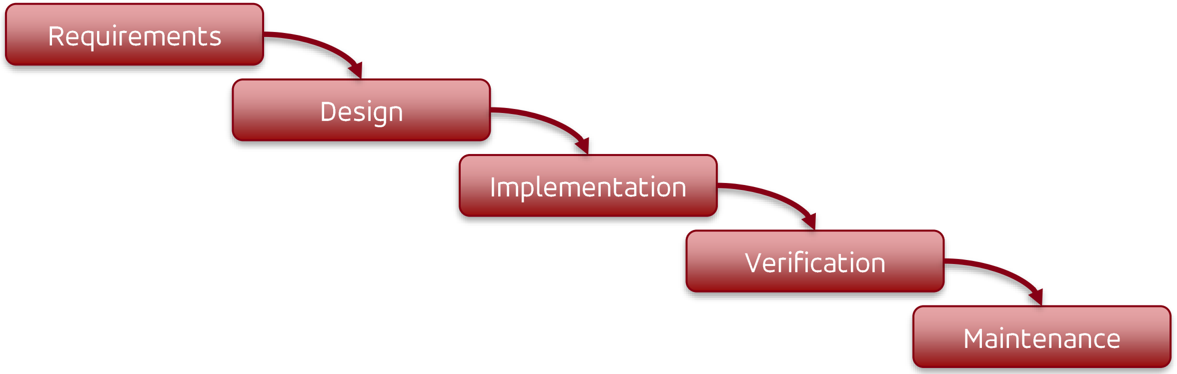
- **Execution.** Deliver products according to the defined plan, schedule and costs

- **Monitoring and control.** Follow any deviations from planned activities, including:
  - Scope, schedule and costs
  - Risks

- **Closure.** Defines the end of the activities. Ideally, it should include:
  - Lessons learned
  - Recommendations for future projects

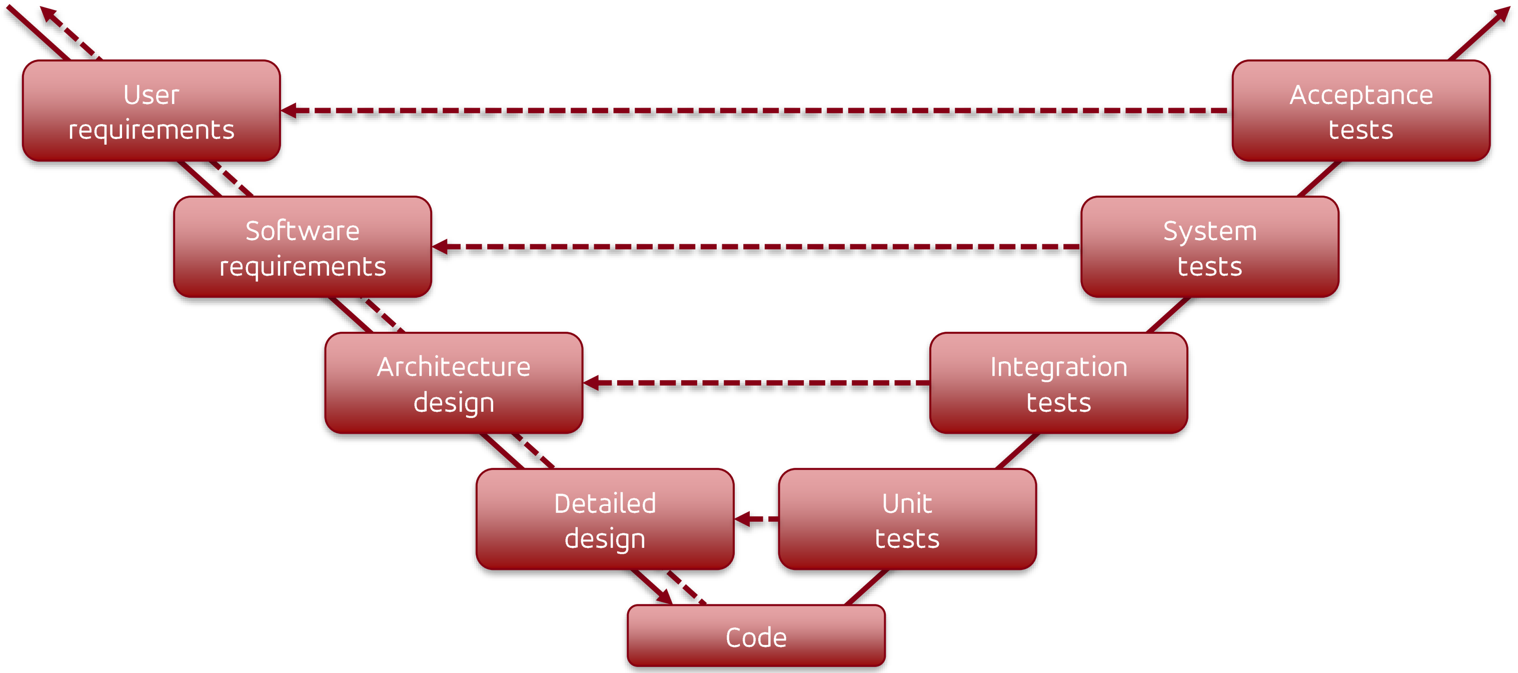
Software development models

- Waterfall.
  - Most software development models are a derivative of this model
  - Classical approach, consisting in several **sequential** phases
  - Useful when requirements and work to be done are clear



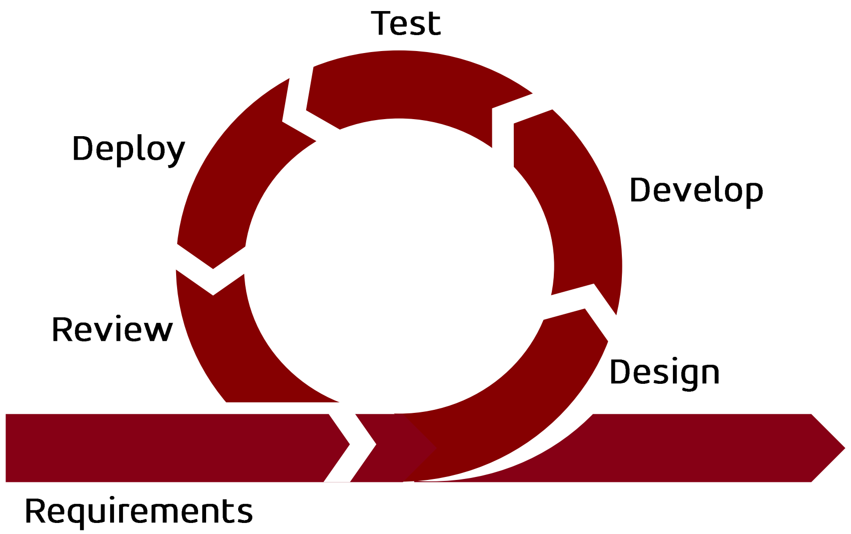
Software development models

- V-model.
  - Development phase associated to a testing phase
  - Focused on quality



Software development models

- Agile.
  - Development divided into small incremental parts
  - Adaptability to changing customer requirements
  - Focused on stakeholders' satisfaction
  - Requires stakeholder commitment



Case studies

- Space projects software. ECSS standard
  - European Collaboration for Space Standardization
  - Officially adopted by ESA in June 1994
  - Standard used for all ESA space-related projects and its **stakeholders**
  - Currently over 140 active standards
  - Covering management, engineering, product assurance, and space sustainability disciplines

- Open-source software.
  - Richard Stallman launched the GNU Project in 1983
  - Linus Torvals released the Linux kernel in 1991
  - Google released Android as open source in 2008, based on Linux
  - Nowadays most electronic devices include some form of open-source software

Tailoring

Each project requires its own project management system

Aspect	Space Software Projects	Open-source Projects
Governance	Centralized (agency or contractor)	Decentralized (community or individual-led)
Standards	Strict compliance (safety, reliability)	Informal or community-defined
Timeline	Long-term, milestone-driven	Continuous, release-driven
Risk Tolerance	Extremely low	Moderate to high
Documentation	Extensive and mandatory	Varies widely
Testing	Formal V&V processes	Community testing, CI/CD
Deployment	One-time or infrequent	Continuous delivery

Index

1. Introduction
2. Management in space software projects
3. Management in open-source software projects
4. Summary

Software for space missions

Use cases:

- **Simulation and modeling.** Before launch, software is used to:
  - Simulate orbital mechanics, thermal environments, and system interactions
  - Model failure scenarios and test redundancy strategies
  - Validate mission plans and software behavior under extreme conditions

- **Ground systems and mission control.** Software supports:
  - Mission planning and scheduling
  - Real-time monitoring of spacecraft health and status
  - Command sequencing and anomaly detection

Software for space missions

Use cases:

- **Scientific data collections and analysis.** In all scientific missions, software:
  - Operates instruments and sensors (e.g., spectrometers, cameras, telescopes)
  - Handles data acquisition, compression, and transmission
  - Enables real-time or post-processing of data for scientific insights (e.g., Gaia DPAC)
- **Mission critical operations.** Software is the brain of space missions
  - Spacecraft navigation and attitude control (e.g., maintaining orientation in space)
  - Autonomous decision-making when communication delays make real-time control impossible (e.g., Mars rovers)
  - Telemetry and command systems for sending and receiving data between Earth and spacecraft

Software for space missions

Challenges:

- **Extreme Reliability Requirements**
  - **Failure is not an option.** A single software bug can lead to mission failure, loss of spacecraft, or even endanger lives (e.g., The Ariane 5 rocket failure in 1996 was caused by a software error in converting a 64-bit float to a 16-bit integer)
  - **No patching after launch.** Once deployed (especially in deep space), software must work flawlessly without the need for updates

- **Long Development and Mission Lifecycles**
  - Projects often span decades from conception to decommissioning.
  - Software must remain maintainable and operable over long periods, even as hardware and personnel change. (e.g., Voyager spacecraft launched in 1977 still receive software updates written in legacy languages)

Software for space missions

Challenges:

- **Harsh and Unpredictable Environments**
  - Software must handle radiation-induced bit flips, extreme temperatures, and communication delays.
  - Systems must be fault-tolerant and capable of autonomous recovery (e.g., Mars rovers use watchdog timers and redundant systems to recover from faults without human intervention)

- **Long Development and Mission Lifecycles**
  - Projects often span decades from conception to decommissioning.
  - Software must remain maintainable and operable over long periods, even as hardware and personnel change. (e.g., Voyager spacecraft launched in 1977 still receive software updates written in legacy languages)

- **Scientific Complexity**
  - Software often encodes complex scientific models, simulations, or data processing pipelines
  - Developers must collaborate closely with domain experts (e.g., astrophysicists, geologists)

Software for space missions

Challenges:

- **Rigorous Standards and Certification**
  - Must comply with strict standards (e.g., NASA-STD-8739.8, ECSS-E-ST-40C)
  - Requires extensive documentation, testing, and verification at every stage
- **Interdisciplinary Collaboration**
  - Teams include scientists, engineers, software developers, and mission planners
  - Requires strong communication and integration across disciplines
- **Limited Resources**
  - Spacecraft have limited memory, processing power, and energy
  - Software must be highly optimized and efficient

ECSS standards for space software projects

Management

- ECSS-M-ST-10C Rev.1. Project planning and implementation
- ECSS-M-ST-10-01C. Organization and conduct of reviews
- ECSS-M-ST-40C Rev.1. Configuration and information management
- ECSS-M-ST-60C. Cost and schedule management
- ECSS-M-ST-80C. Risk management

Engineering

- ECSS-E-ST-40C Rev.1. Software

Product assurance

- ECSS-Q-ST-80C Rev.2. Software Product Assurance

ECSS standards for space software project management

- Fundamental principle is the customer–supplier relationship → In many cases, colleagues working for the same goal

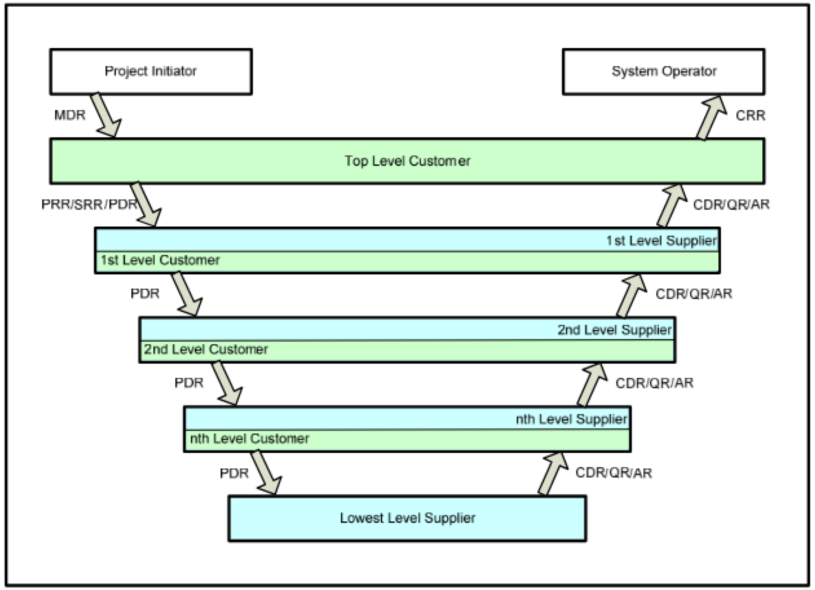
- The concept of the customer–supplier relationship is applied recursively to subcontractors

- Responsibilities:
  - Correct procedures are followed
  - Software is delivered on schedule and to budget
  - Final products are of the expected quality (correct, safe, efficient and easy to maintain or modify)

- Development model?
  - Waterfall?
  - V-model?
  - Agile?

ECSS space software project management activities

- Organization and handling of **reviews** (ECSS-M-ST-10-01C)
- Development model
  - Usually waterfall or V-model is used (for projects with several subcontractors)
  - **ECSS-E-HB-40-10A.** Agile software development handbook



ECSS space software project management activities

- Organization and handling of **reviews** (ECSS-M-ST-10-01C)
- Development model
  - Usually waterfall or V-model is used (for projects with several subcontractors)
  - **ECSS-E-HB-40-10A.** Agile software development handbook

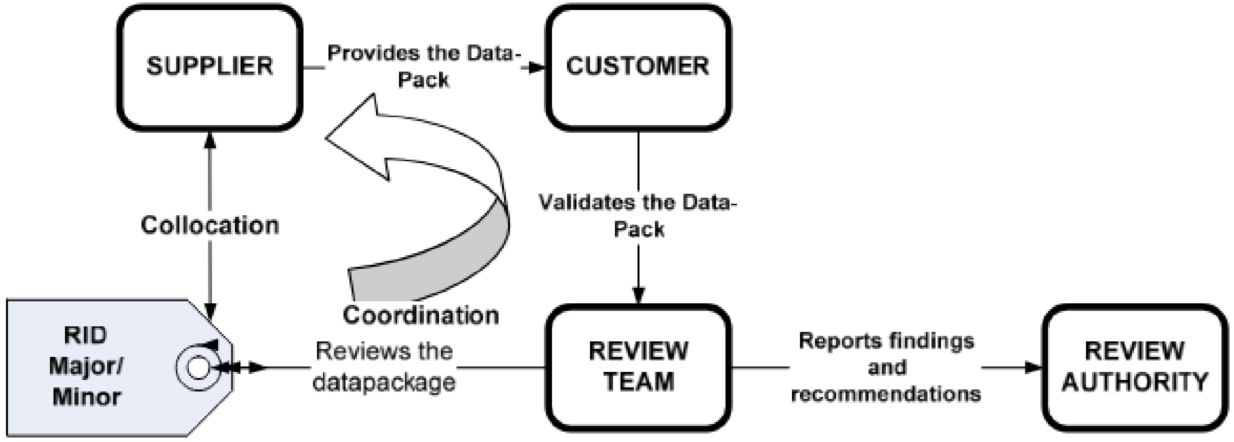
Project Management Plan ECSS-M-ST-10C	Software Development Plan ECSS-E-ST-40C
• Project organization and roles	• Life cycle description
• Project breakdown structures	• Standards and techniques*
• Configuration management	• Development and testing environment*
• Cost and schedule management	• Documentation plan
• Risks management	• Tailoring
• Product assurance management	• Supplier management

*Open-source software

ECSS review process

- Deliverables are provided from the supplier to the customer

- During a review, customer and review team can provide
  - Review Item Discrepancies (RIDs). Mostly for documentation
  - Software Change Requests (SCRs) can also be raised to propose a program change versus requirements
  - Software Problem Reports (SPRs)



ECSS software lifecycle

Reviews are the main interaction points between the customer and the supplier:

- System Requirements Review (SRR).
- Preliminary Design Review (PDR).
- Critical Design Review (CDR)
- Qualification Review (QR)
- Acceptance Review (AR)
- Operational Readiness Review (ORR)

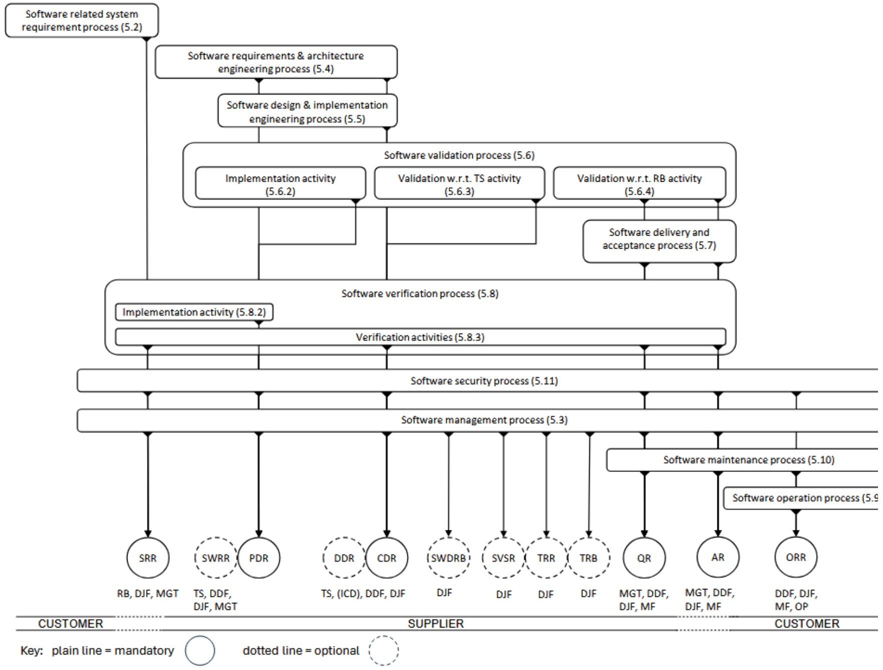
ECSS project lifecycle

The ECSS software lifecycle is usually aligned with the overall mission lifecycle (Phases 0–F):

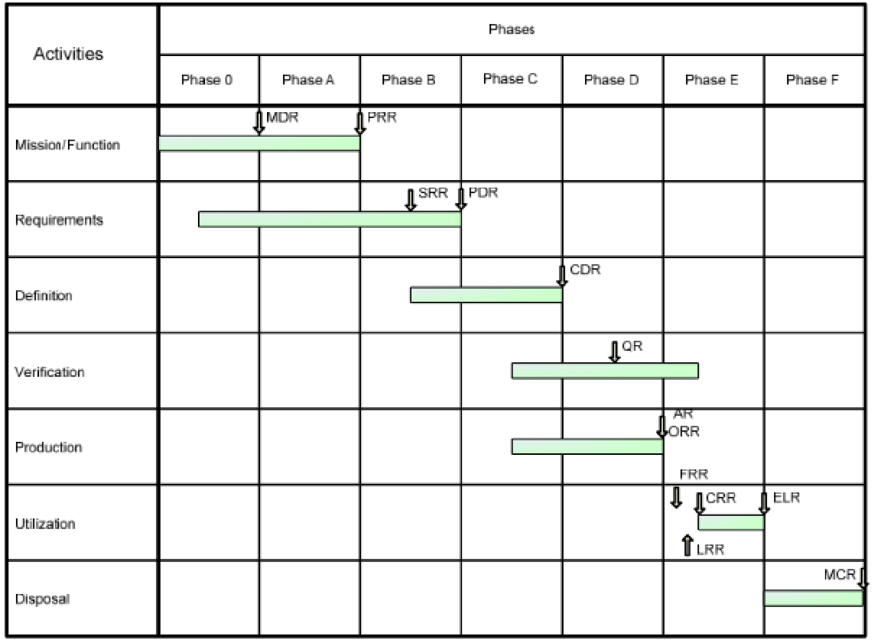
Phase	Purpose	Software Activities
0	Mission analysis	High-level software feasibility
A	Feasibility	Preliminary software requirements
B	Preliminary definition	Software architecture and planning
C	Detailed definition	Detailed design and interface specifications
D	Qualification and production	Coding, integration, and verification
E	Utilization	Deployment, operations, and maintenance
F	Disposal	Decommissioning and data archiving

ECSS project lifecycle

The ECSS software lifecycle is usually aligned with the overall mission lifecycle (Phases 0–F):



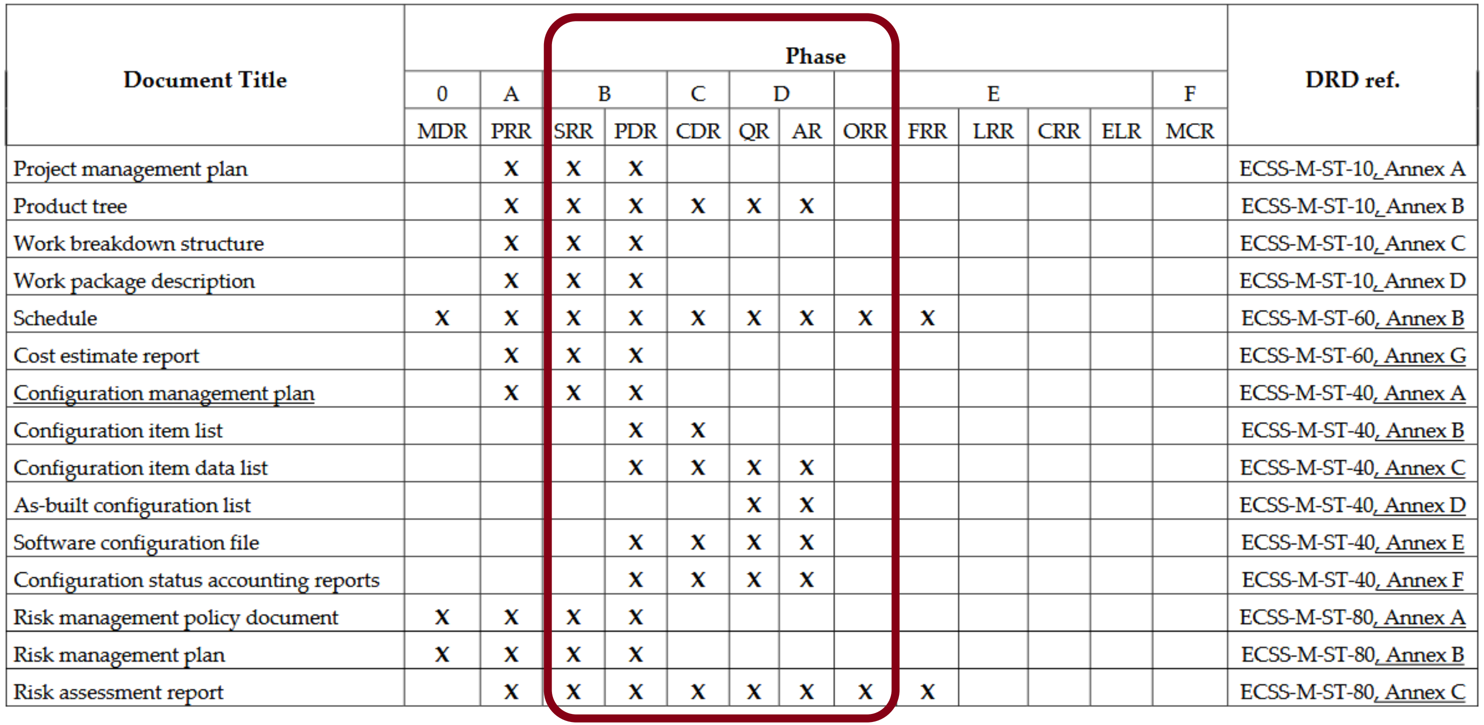
Software related system requirement process (5.2)
Software requirements & architecture engineering process (5.4)
Software design & implementation engineering process (5.5)
Software validation process (5.6)
Implementation activity (5.6.2)
Validation w.r.t. TS activity (5.6.3)
Validation w.r.t. RB activity (5.6.4)
Software delivery and acceptance process (5.7)
Software verification process (5.8)
Implementation activity (5.8.2)
Verification activities (5.8.3)
Software security process (5.11)
Software management process (5.3)
Software maintenance process (5.10)
Software operation process (5.9
SRR
SWRR
PDR
DDR
CDR
SWDRB
SVSR
TRR
TRB
QR
AR
ORR
RB, DJF, MGT
TS, DDF, DJF, MGT
TS, (ICD), DDF, DJF
DJF
DJF
DJF
DJF
MGT, DDF, DJF, MF
MGT, DDF, DJF, MF
DDF, DJF, MF, OP
CUSTOMER
SUPPLIER
CUSTOMER
Key: plain line = mandatory
dotted line = optional



ECSS project lifecycle

The ECSS software lifecycle is usually aligned with the overall mission lifecycle (Phases 0–F):

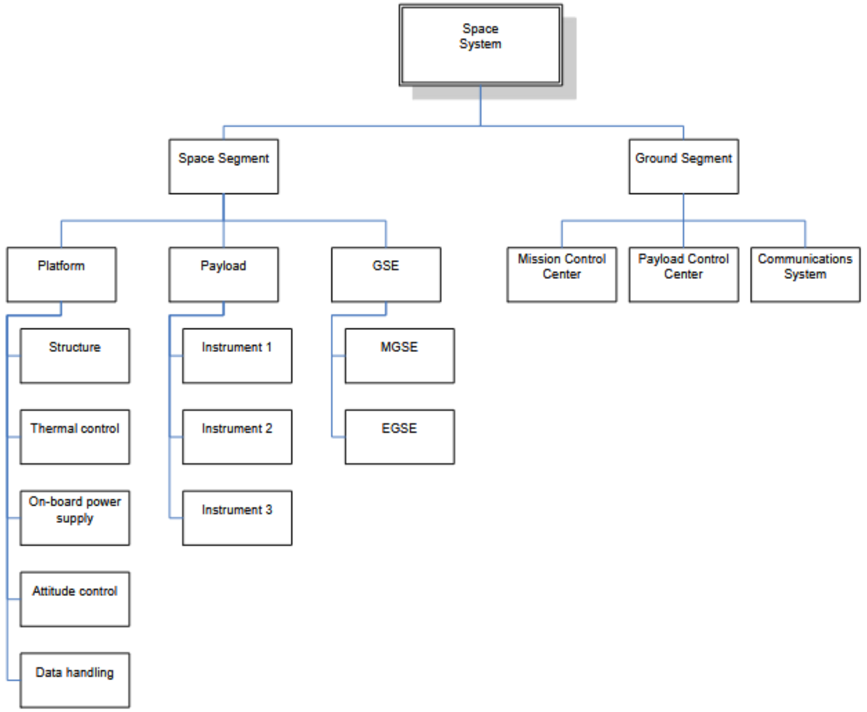
Table F-1 :Management Documents Delivery per Review



Document Title	Phase													DRD ref.
	0	A	B		C	D		E					F	
	MDR	PRR	SRR	PDR	CDR	QR	AR	ORR	FRR	LRR	CRR	ELR	MCR	
Project management plan		X	X	X										ECSS-M-ST-10, Annex A
Product tree		X	X	X	X	X	X							ECSS-M-ST-10, Annex B
Work breakdown structure		X	X	X										ECSS-M-ST-10, Annex C
Work package description		X	X	X										ECSS-M-ST-10, Annex D
Schedule	X	X	X	X	X	X	X	X	X					ECSS-M-ST-60, Annex B
Cost estimate report		X	X	X										ECSS-M-ST-60, Annex G
Configuration management plan		X	X	X										ECSS-M-ST-40, Annex A
Configuration item list				X	X									ECSS-M-ST-40, Annex B
Configuration item data list				X	X	X	X							ECSS-M-ST-40, Annex C
As-built configuration list						X	X							ECSS-M-ST-40, Annex D
Software configuration file				X	X	X	X							ECSS-M-ST-40, Annex E
Configuration status accounting reports				X	X	X	X							ECSS-M-ST-40, Annex F
Risk management policy document	X	X	X	X										ECSS-M-ST-80, Annex A
Risk management plan	X	X	X	X										ECSS-M-ST-80, Annex B
Risk assessment report		X	X	X	X	X	X	X	X					ECSS-M-ST-80, Annex C

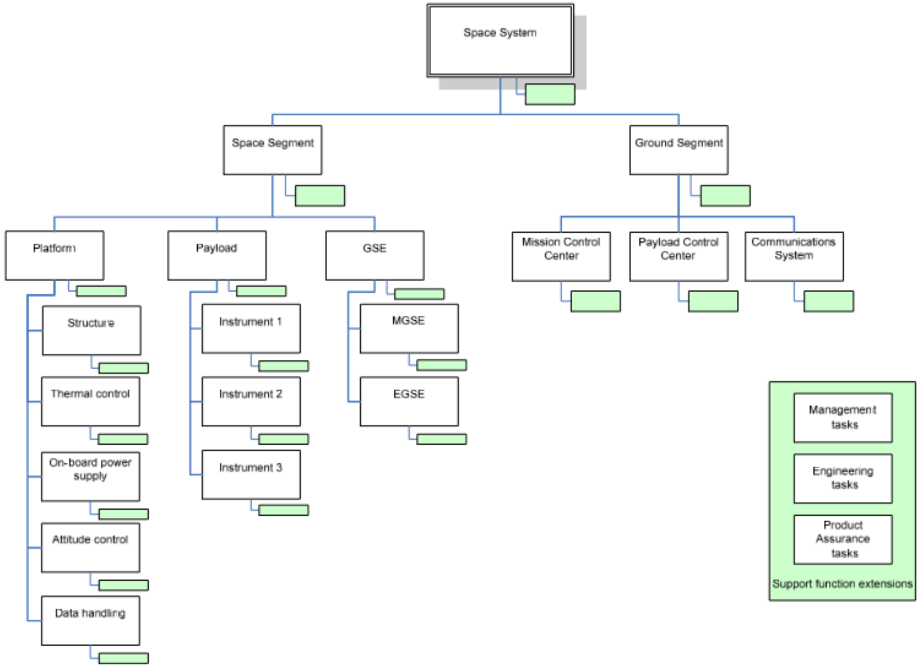
Product tree

- Breakdown of the project into successive levels of system components
- Generally used in space missions or large software products



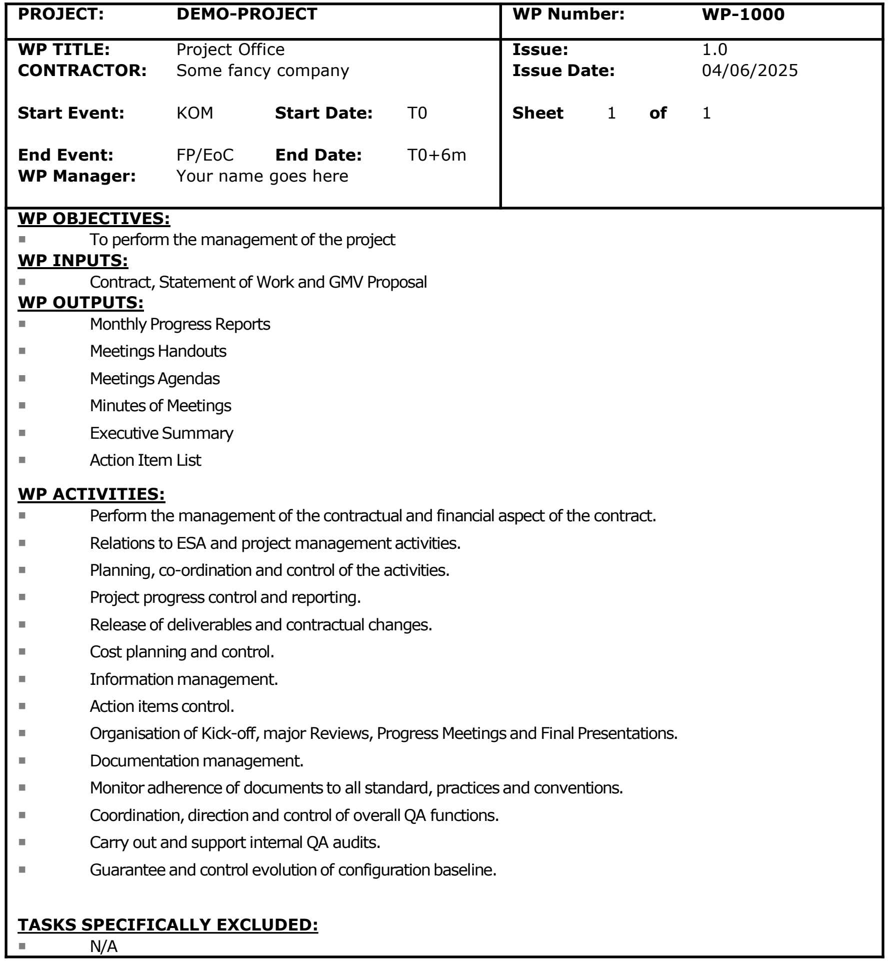
Work Breakdown Structure (WBS)

- Divides the project into manageable work packages
- Organized by the nature of the work
- Used in most projects, **including science projects**



Work package (WP) description

- Lowest level that can be measured and managed for planning, monitoring and control
- Represent the total work scope
- The work of each supplier is explicitly identified
- Minimum content specification:
  - Requires a WP manager name and organization
  - Start event identification and date
  - End event identification and date
  - Description of the objectives of the WP
  - Description of the tasks
  - List of the inputs necessary to achieve the tasks
  - Interfaces or links with other tasks or WP's
  - List of the expected outputs and deliverables



PROJECT:	**DEMO-PROJECT**			**WP Number:**			**WP-1000**	
WP TITLE:	Project Office			**Issue:**			1.0	
CONTRACTOR:	Some fancy company			**Issue Date:**			04/06/2025	
Start Event:	KOM	**Start Date:**	T0	**Sheet**	1	**of**	1	
End Event:	FP/EoC	**End Date:**	T0+6m					
WP Manager:	Your name goes here							

WP OBJECTIVES:
- To perform the management of the project

WP INPUTS:
- Contract, Statement of Work and GMV Proposal

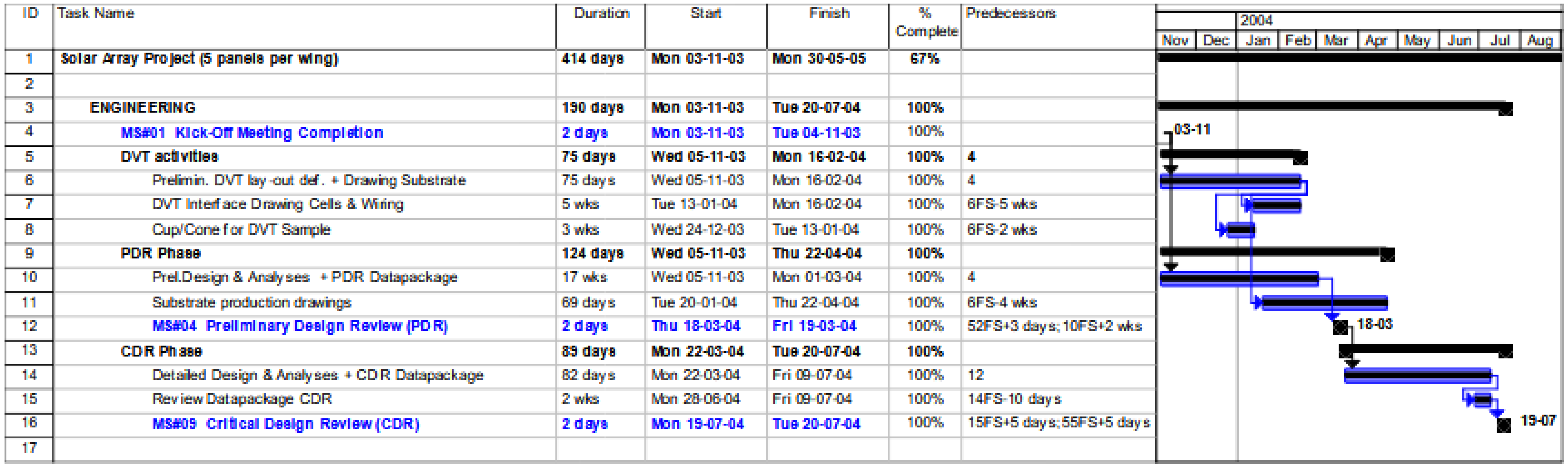
WP OUTPUTS:
- Monthly Progress Reports
- Meetings Handouts
- Meetings Agendas
- Minutes of Meetings
- Executive Summary
- Action Item List

WP ACTIVITIES:
- Perform the management of the contractual and financial aspect of the contract.
- Relations to ESA and project management activities.
- Planning, co-ordination and control of the activities.
- Project progress control and reporting.
- Release of deliverables and contractual changes.
- Cost planning and control.
- Information management.
- Action items control.
- Organisation of Kick-off, major Reviews, Progress Meetings and Final Presentations.
- Documentation management.
- Monitor adherence of documents to all standard, practices and conventions.
- Coordination, direction and control of overall QA functions.
- Carry out and support internal QA audits.
- Guarantee and control evolution of configuration baseline.

TASKS SPECIFICALLY EXCLUDED:
- N/A

ECSS schedule management

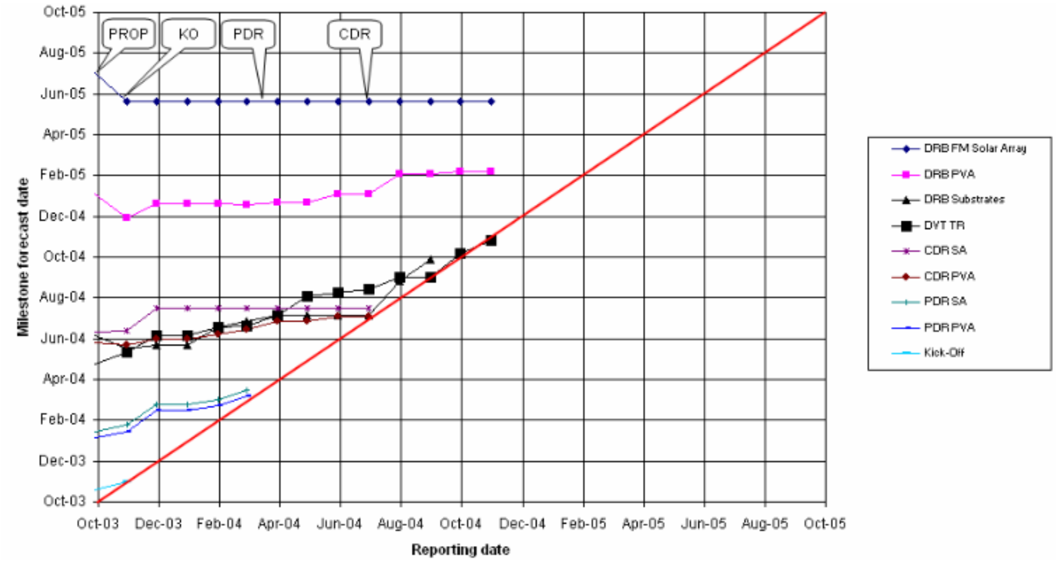
- Schedule relies on Work Package (WP) estimated duration and dependencies
- Milestones and reviews should be defined accordingly
- Resource allocation is defined for each WP
- Project schedule presentation is usually a Gantt chart

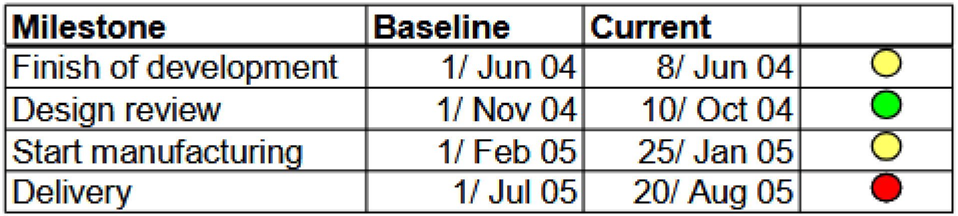


ID	Task Name	Duration	Start	Finish	% Complete	Predecessors
1	**Solar Array Project (5 panels per wing)**	**414 days**	**Mon 03-11-03**	**Mon 30-05-05**	**67%**	
2						
3	**ENGINEERING**	**190 days**	**Mon 03-11-03**	**Tue 20-07-04**	**100%**	
4	**MS#01 Kick-Off Meeting Completion**	**2 days**	**Mon 03-11-03**	**Tue 04-11-03**	100%	
5	**DVT activities**	**75 days**	**Wed 05-11-03**	**Mon 16-02-04**	**100%**	**4**
6	Prelimin. DVT lay-out def. + Drawing Substrate	75 days	Wed 05-11-03	Mon 16-02-04	100%	4
7	DVT Interface Drawing Cells & Wiring	5 wks	Tue 13-01-04	Mon 16-02-04	100%	6FS-5 wks
8	Cup/Cone for DVT Sample	3 wks	Wed 24-12-03	Tue 13-01-04	100%	6FS-2 wks
9	**PDR Phase**	**124 days**	**Wed 05-11-03**	**Thu 22-04-04**	**100%**	
10	Prel.Design & Analyses + PDR Datapackage	17 wks	Wed 05-11-03	Mon 01-03-04	100%	4
11	Substrate production drawings	69 days	Tue 20-01-04	Thu 22-04-04	100%	6FS-4 wks
12	**MS#04 Preliminary Design Review (PDR)**	**2 days**	**Thu 18-03-04**	**Fri 19-03-04**	100%	52FS+3 days;10FS+2 wks
13	**CDR Phase**	**89 days**	**Mon 22-03-04**	**Tue 20-07-04**	**100%**	
14	Detailed Design & Analyses + CDR Datapackage	82 days	Mon 22-03-04	Fri 09-07-04	100%	12
15	Review Datapackage CDR	2 wks	Mon 28-06-04	Fri 09-07-04	100%	14FS-10 days
16	**MS#09 Critical Design Review (CDR)**	**2 days**	**Mon 19-07-04**	**Tue 20-07-04**	100%	15FS+5 days;55FS+5 days
17						

ECSS schedule monitoring

- Gantt chart
- Milestone Trend/Slip Chart
  - Slopes indicate delay
- Milestone list
  - Delays indicated by colors





Milestone	Baseline	Current	
Finish of development	1/ Jun 04	8/ Jun 04	🟡
Design review	1/ Nov 04	10/ Oct 04	🟢
Start manufacturing	1/ Feb 05	25/ Jan 05	🟡
Delivery	1/ Jul 05	20/ Aug 05	🔴

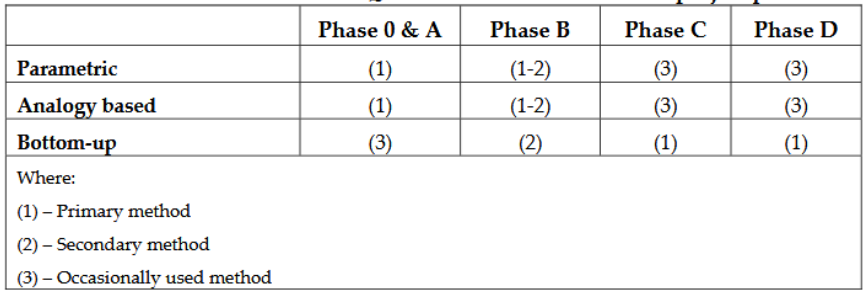
ECSS cost management

- Having a strong cost estimation is essential for ECSS projects to succeed

- Several estimation methods. Most of them rely on solid Work Breakdown Structure (WBS)
  - **Bottom-up.** Estimate the effort of each task and add up the cost of all tasks.
  - **Analogy.** Top-down approach. Use a similar project and realize minor adjustments
  - **Parametric.** Use empirical relations to estimate costs.

- The best estimation method strongly linked to project phase

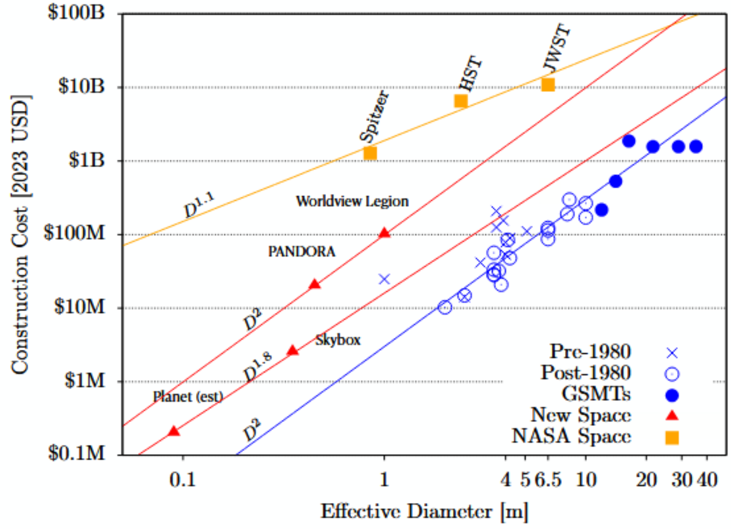
Table Q-1: Cost estimate method vs. project phase



	Phase 0 & A	Phase B	Phase C	Phase D
Parametric	(1)	(1-2)	(3)	(3)
Analogy based	(1)	(1-2)	(3)	(3)
Bottom-up	(3)	(2)	(1)	(1)

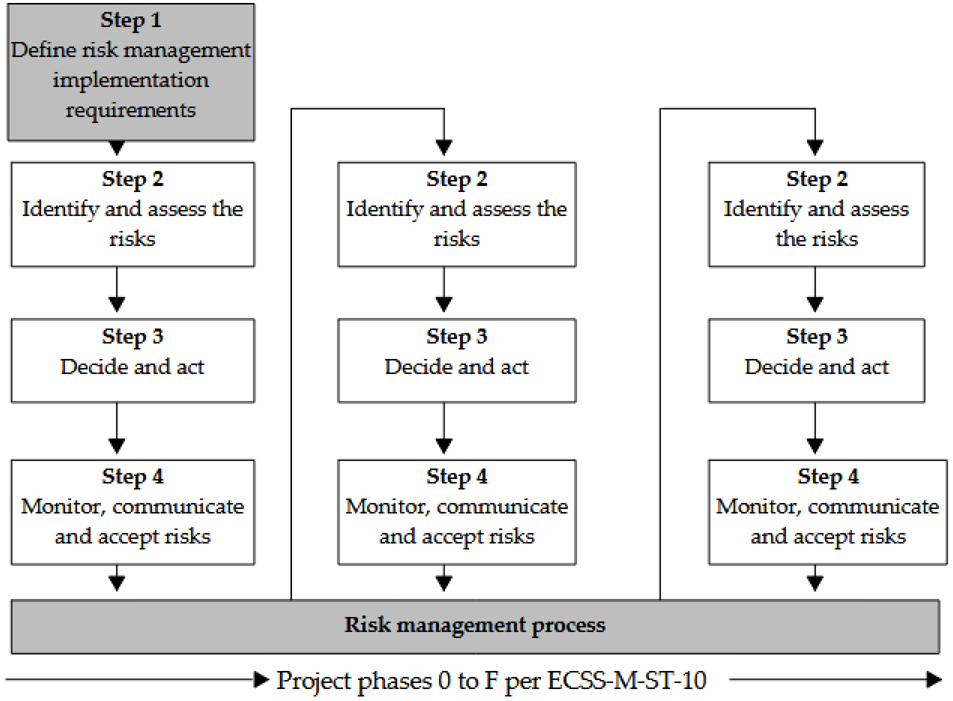
Where:
(1) – Primary method
(2) – Secondary method
(3) – Occasionally used method

Douglas+ (2023)



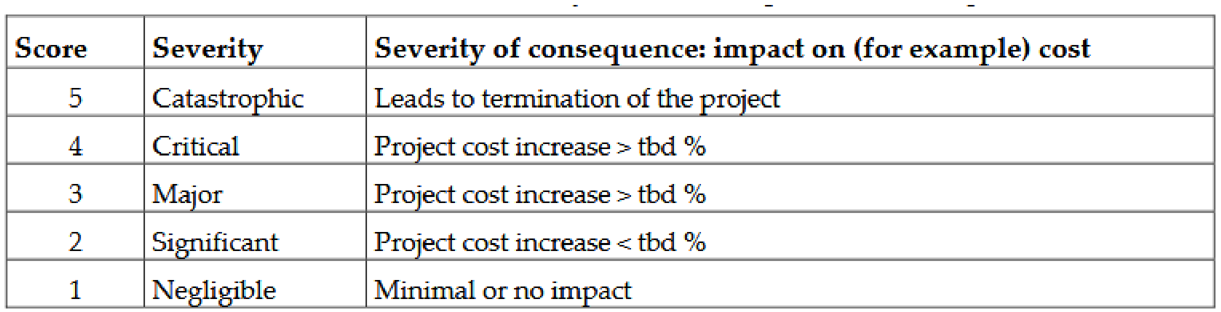
ECSS risk management

- Mandatory for all ECSS projects
- Included in the cost estimation
- Implemented as a team effort
- Iterated through the entire project life cycle
- Communicated to the client

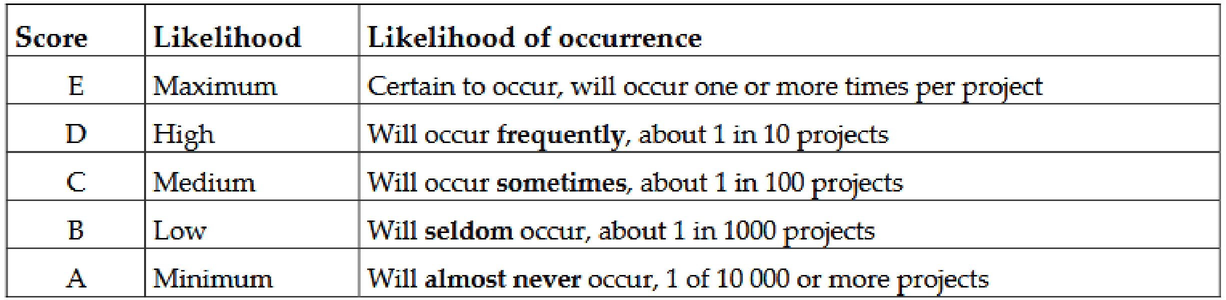

Step 1
Define risk management implementation requirements
Step 2
Identify and assess the risks
Step 3
Decide and act
Step 4
Monitor, communicate and accept risks
Step 2
Identify and assess the risks
Step 3
Decide and act
Step 4
Monitor, communicate and accept risks
Step 2
Identify and assess the risks
Step 3
Decide and act
Step 4
Monitor, communicate and accept risks
Risk management process
Project phases 0 to F per ECSS-M-ST-10

ECSS risk policy

- Two main axes:
  - Severity
  - Likelihood
- Risk index → mitigation action



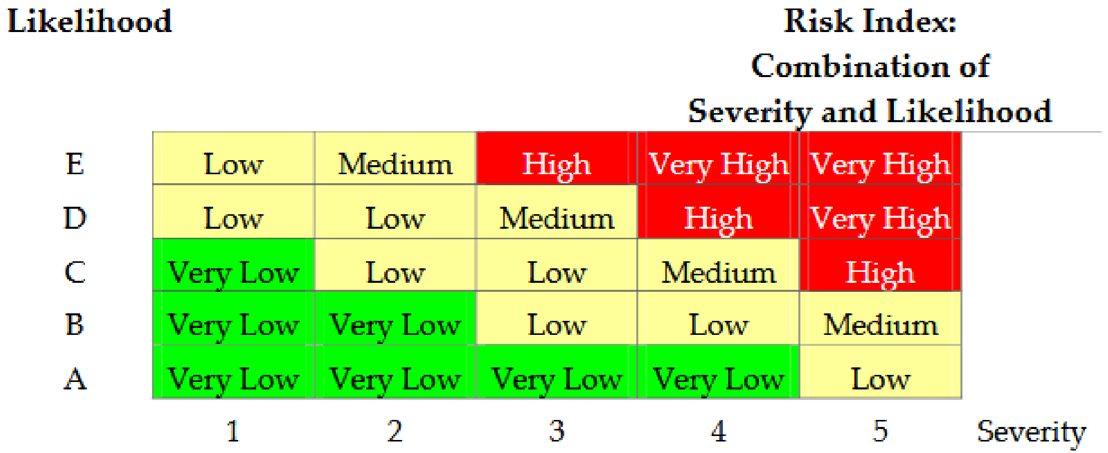
Score	Severity	Severity of consequence: impact on (for example) cost
5	Catastrophic	Leads to termination of the project
4	Critical	Project cost increase > tbd %
3	Major	Project cost increase > tbd %
2	Significant	Project cost increase < tbd %
1	Negligible	Minimal or no impact



Score	Likelihood	Likelihood of occurrence
E	Maximum	Certain to occur, will occur one or more times per project
D	High	Will occur **frequently**, about 1 in 10 projects
C	Medium	Will occur **sometimes**, about 1 in 100 projects
B	Low	Will **seldom** occur, about 1 in 1000 projects
A	Minimum	Will **almost never** occur, 1 of 10 000 or more projects

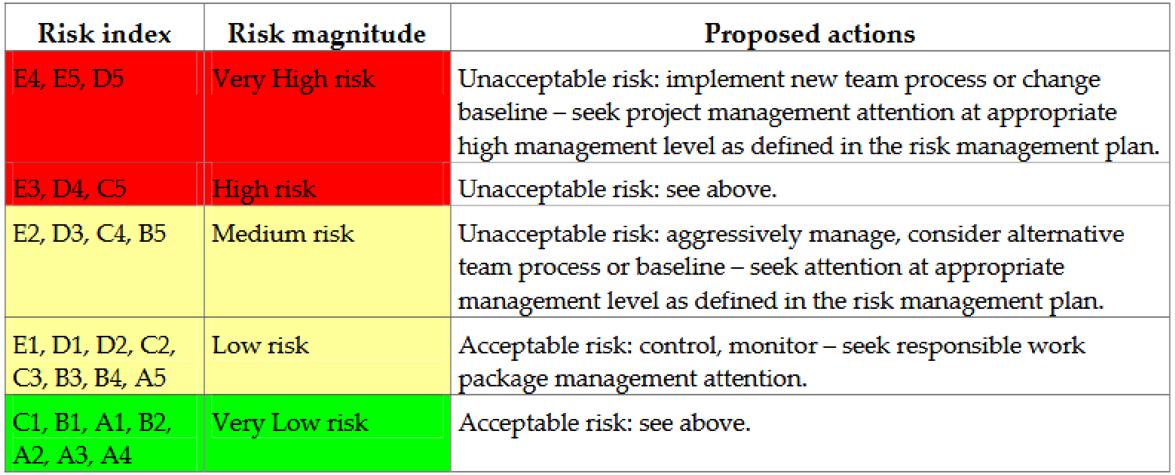
ECSS risk policy

- Two main axes:
  - Severity
  - Likelihood
- Risk index → mitigation action



Risk Index: Combination of Severity and Likelihood

Likelihood					
E	Low	Medium	High	Very High	Very High
D	Low	Low	Medium	High	Very High
C	Very Low	Low	Low	Medium	High
B	Very Low	Very Low	Low	Low	Medium
A	Very Low	Very Low	Very Low	Very Low	Low
	1	2	3	4	5 Severity



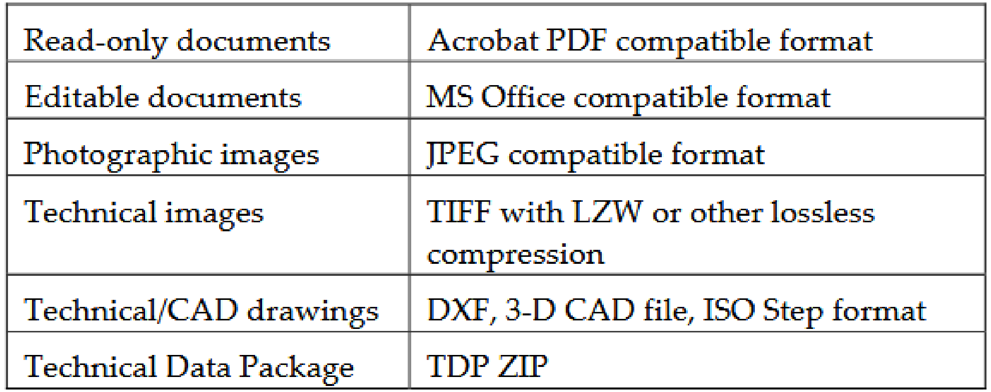
Risk index	Risk magnitude	Proposed actions
E4, E5, D5	Very High risk	Unacceptable risk: implement new team process or change baseline – seek project management attention at appropriate high management level as defined in the risk management plan.
E3, D4, C5	High risk	Unacceptable risk: see above.
E2, D3, C4, B5	Medium risk	Unacceptable risk: aggressively manage, consider alternative team process or baseline – seek attention at appropriate management level as defined in the risk management plan.
E1, D1, D2, C2, C3, B3, B4, A5	Low risk	Acceptable risk: control, monitor – seek responsible work package management attention.
C1, B1, A1, B2, A2, A3, A4	Very Low risk	Acceptable risk: see above.

ECSS configuration management

- Consistent record of product’s characteristics

- In the case of software projects, applies to:
  - Documentation
  - Development tools (e.g., compilers, linkers)
  - Validation specifications
  - Test procedures
  - Maintenance procedures
  - Deviations
  - Engineering changes

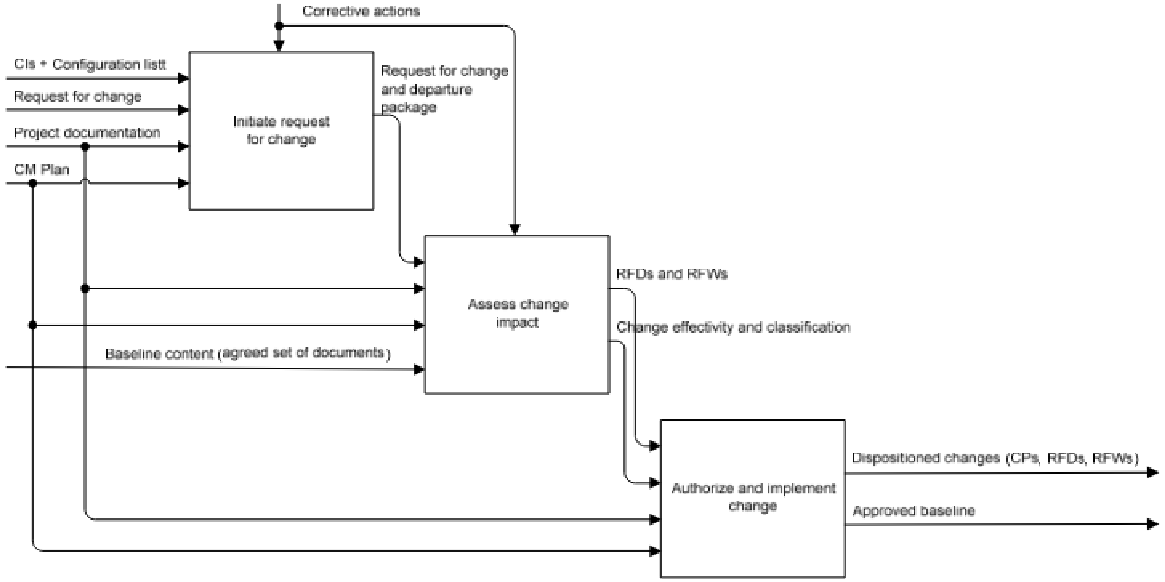
- Default documentation delivery formats



Read-only documents	Acrobat PDF compatible format
Editable documents	MS Office compatible format
Photographic images	JPEG compatible format
Technical images	TIFF with LZW or other lossless compression
Technical/CAD drawings	DXF, 3-D CAD file, ISO Step format
Technical Data Package	TDP ZIP

ECSS configuration control

- RIDs, SCRs and SPRs can only be updated **after authorization**
- Authorization is also needed for:
  - Request for Deviation (RfD). Departure from a customer's requirement.
  - Request for Waiver (RfW). Product that does not conform to its approved configuration baseline



Corrective actions
CIs + Configuration listt
Request for change
Project documentation
CM Plan
Initiate request for change
Request for change and departure package
Assess change impact
Baseline content (agreed set of documents)
RFDs and RFWs
Change effectivity and classification
Authorize and implement change
Dispositioned changes (CPs, RFDs, RFWs)
Approved baseline

Index

1. Introduction
2. Management in space software projects
3. Management in open-source software projects
4. Summary

What is open-source software?

- **Definition.** Software with source code that anyone can inspect, modify, and enhance.

- Key Characteristics:
  - Publicly accessible source code
  - Permission to use, modify, and distribute
  - Often developed collaboratively

- Nowadays, open-source software strongly relies on public open-source repositories: Github, Gitlab, ...

- **Repositories** can have hundreds of millions of contributors and a similar amount of projects

- **Projects** can have hundreds or thousands of contributors

Open-source software benefits

- Transparency
  - Anyone can inspect the source code
  - Helps identify bugs, security flaws, or unethical behavior
- Community Collaboration
  - Contributions from a global pool of developers
  - Diverse perspectives lead to innovation and rapid problem-solving
- Cost-Effectiveness
  - Most open-source software is free to use
  - Reduces licensing costs for individuals and organizations
- Flexibility and Customization
  - Users can modify the software to suit their specific needs
  - Encourages experimentation and innovation

Open-source software benefits

- Security and Reliability
  - “Many eyes” principle: more people reviewing code can lead to more secure and stable software.
  - Popular projects often have robust testing and peer review.

- Learning and Skill Development
  - Great resource for developers to learn from real-world codebases.
  - Encourages best practices and community standards.

- Vendor Independence
  - No lock-in to a single vendor or proprietary ecosystem.
  - Users have control over updates and integrations.

Open-source software challenges

- Documentation and Support
  - Documentation may be incomplete or outdated
  - Support is often community-based and not guaranteed
- Governance and Decision-Making
  - Disagreements can lead to project splits or stagnation
  - Lack of clear leadership can hinder progress
- Onboarding New Contributors
  - Complex codebases and unclear contribution guidelines can deter newcomers.
  - Requires active community management

Open-source software challenges

- Sustainability
  - Many projects rely on unpaid volunteers
  - Maintainers may face burnout or lack of resources
- Fragmentation
  - Forking can lead to multiple versions of a project
  - Lack of standardization can confuse users and contributors
- Security Risks
  - While open code can be reviewed, it can also be exploited
  - Some projects lack regular security audits or updates

Version control in software projects

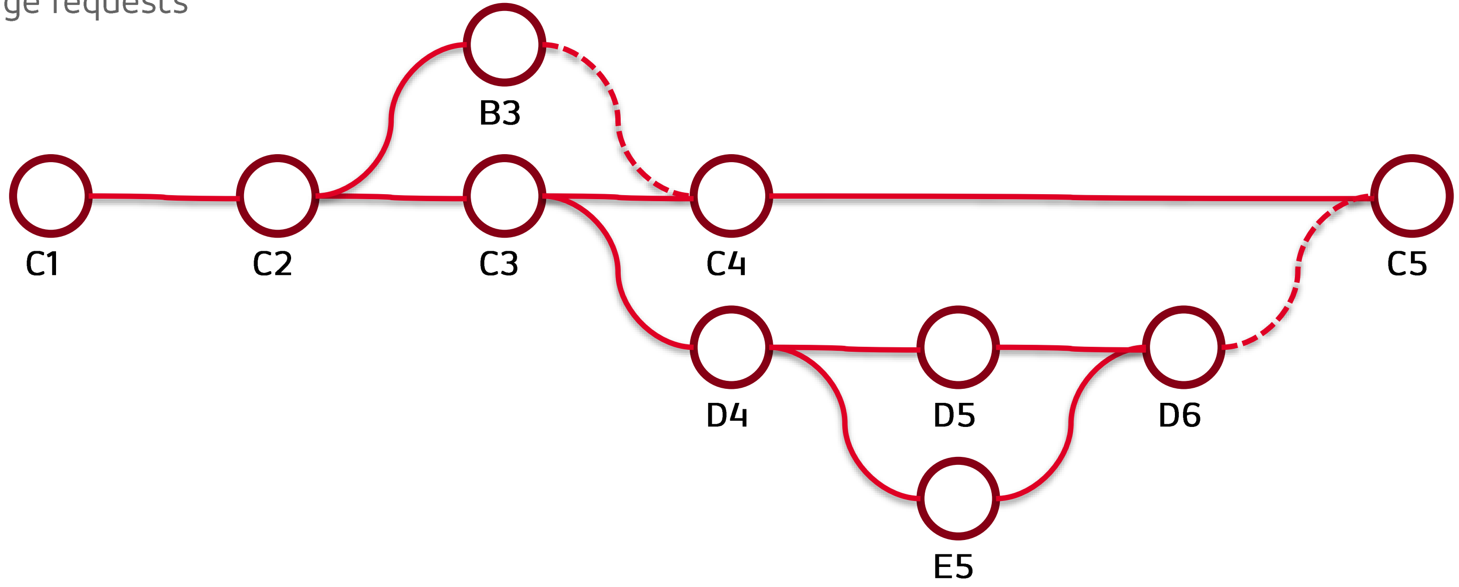
- What is version control?
  - System to track all changes in written text
  - Can be used to manage multiple versions of any text
  - Source code is basically written text → version control can be used

- Adoption:
  - Almost **all** open-source software projects rely on version control
  - Also used in most space software projects

- Tools:
  - BitKeeper
  - Mercurial
  - Subversion
  - Git

Version control in software projects

- Version control at work:
  - Branching
  - Issue resolution
  - Merge requests



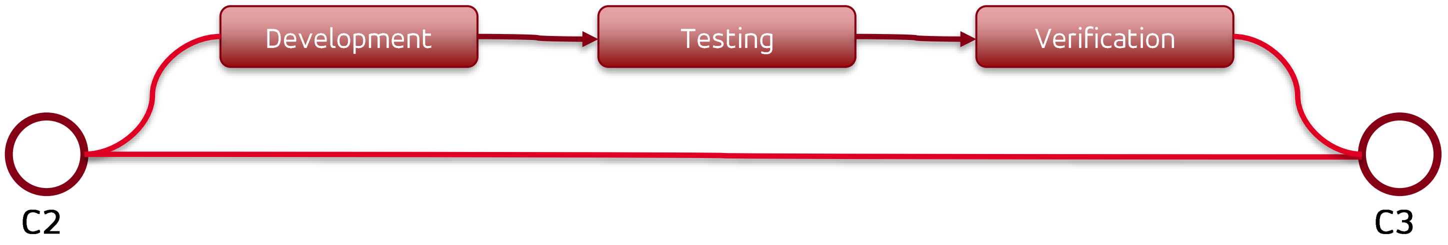
Version control in software projects

- Huge advantages:
  - **Collaboration.** Developers can work independently and merge changes afterwards
  - **Change tracking.** Every modification in code is automatically tracked
  - **Error recovery.** Any mistake can be resolved by rolling back to previous version
  - **Efficiency.** Configuration control automatically managed
  - **Experimentation.** New ideas can be tested without disturbing main development
  - **Code quality.** Allows continuous improvements in code
  - **Documentation.** Documentation can also benefit from version control
  - **Backup and recovery.** All the history of development can be restored at any point

- Minor disadvantages:
  - **Learning curve.** Learn how a version control tool works

CI/CD in software projects

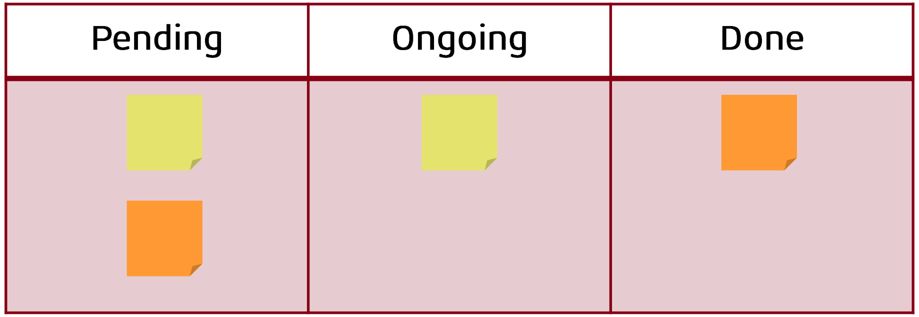
- Continuous Integration / Continuous Delivery
- Relies on version control
- Concept. Each new development is tested before merging
- Workflow:
  - Developers write code as well as tests
  - Tests are **automatically** run after each new commit or before / after merging
  - A report on the test results is **automatically** generated (e.g., code coverage)
- <u>Agile development becomes natural</u>



Agile software project management

- Scrum.
  - Work organized in fixed-duration sprints: 2-4 weeks
  - Each sprint aims at solving specific issues from the backlog **together**
  - Daily stand-up meetings to keep the team aligned
  - Development team is usually small (less than 10 members)
  - Two specific figures:
    - Product owner. Interacts with stakeholders and gives priority to the backlog issues
    - Scrum master. Helps development team to meet and solve barriers

- Kanban
  - Issues are **pulled** from pending backlog
  - Only one issue can be pulled at a time
  - Time to complete issues needs to be monitored



Open-source software project management

- Project governance:
  - **Project founder or single leader.** Sometimes called Benevolent Dictator for Life
  - **Foundation.** A non-profit foundation oversees the project and provides legal and financial support
  - **Meritocratic team.** Influence earned through contributions
  - **Consensus.** Discussions in public forums

- Key roles:
  - **Maintainers.** Review and merge pull requests, manage releases, guide project direction
  - **Contributors.** Submit code, report bugs, write docs, participate in discussions
  - **Reviewers.** Provide feedback on pull request, ensure code quality
  - **Community Managers.** Foster engagement, moderate discussions, onboard new contributors
  - **Users.** Use the software, report bugs, request features

Open-source software project management

- Management is mainly focused in:
  - Define project roadmap and milestones
  - Issue prioritization (code **and** documentation)
  - Approving pull requests
  - Resolve conflicts
  - Ensuring project sustainability

- Roadmap and milestones:
  - **Roadmap.** Strategic plan that outlines the goals and direction of the project over time
  - **Milestone.** Group issues and pull requests to be accomplished before a new release
  - Many projects name milestones with **semantic versioning** (e.g., v1.2.3)
  - Kanban-style boards are sometimes used to define milestone progress

- Community building is essential for the success of an open-source project

Typical documentation

- Documentation is also **a must** for open-source projects
- Documentation is usually part of the distributed code and presented in plain text
- Four common documents:
  - **License.** Legal terms of the software
  - **README.** Describes the project and provides the references to additional documents
  - **Code of conduct.** Describes how to interact with other contributors.
  - **Contributing guidelines.** Describes how to contribute to the project

Open-source licenses

- Licenses define **legal terms** under which software can be **used, modified, distributed or re-licensed**
- An open-source license protects contributors and users
- **All** open-source software projects **should** have a license.
- Licenses can only be changed with new software versions.
- Some licenses allow for **commercial and proprietary** software
- More on Intellectual Property Rights (IPR) tomorrow

Python

Release	Derived from	Year	Owner
0.9.0 thru 1.2	n/a	1991-1995	CWI
1.3 thru 1.5.2	1.2	1995-1999	CNRI
1.6	1.5.2	2000	CNRI
2.0	1.6	2000	BeOpen.com
1.6.1	1.6	2001	CNRI
2.1	2.0+1.6.1	2001	PSF
2.0.1	2.0+1.6.1	2001	PSF
2.1.1	2.1+2.0.1	2001	PSF
2.1.2	2.1.1	2002	PSF
2.1.3	2.1.2	2002	PSF
2.2 and above	2.1.1	2001-now	PSF

License templates

BSD 2-Clause License

Copyright (c) [year], [fullname]

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

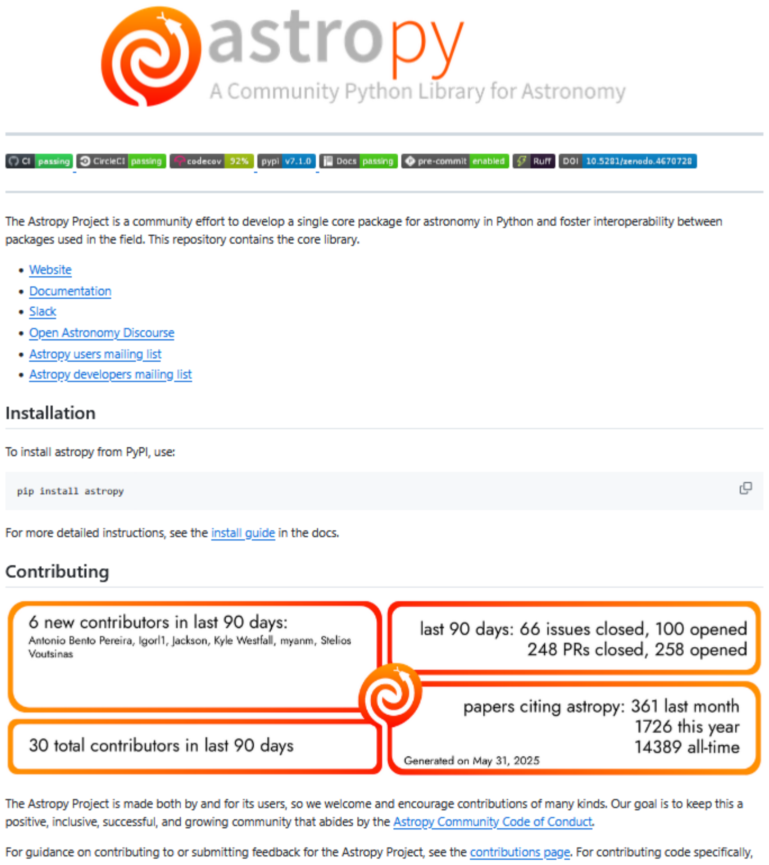
1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Project README

- Entry point for most open-source projects
- Typical information:
  - What the project does
  - Why the project is useful
  - Reference to the license
  - How users can get started (e.g., installation instructions)
  - How users can get help (e.g., documentation)
  - Who maintains and contributes
  - How to contribute
  - How to cite the software



The Astropy Project is a community effort to develop a single core package for astronomy in Python and foster interoperability between packages used in the field. This repository contains the core library.

- Website
- Documentation
- Slack
- Open Astronomy Discourse
- Astropy users mailing list
- Astropy developers mailing list

Installation

To install astropy from PyPI, use:

```
pip install astropy
```

For more detailed instructions, see the install guide in the docs.

Contributing

6 new contributors in last 90 days:
Antonio Bento Pereira, Igorl1, Jackson, Kyle Westfall, myanm, Stelios Voutsinas

last 90 days: 66 issues closed, 100 opened
248 PRs closed, 258 opened

papers citing astropy: 361 last month
1726 this year
14389 all-time

30 total contributors in last 90 days

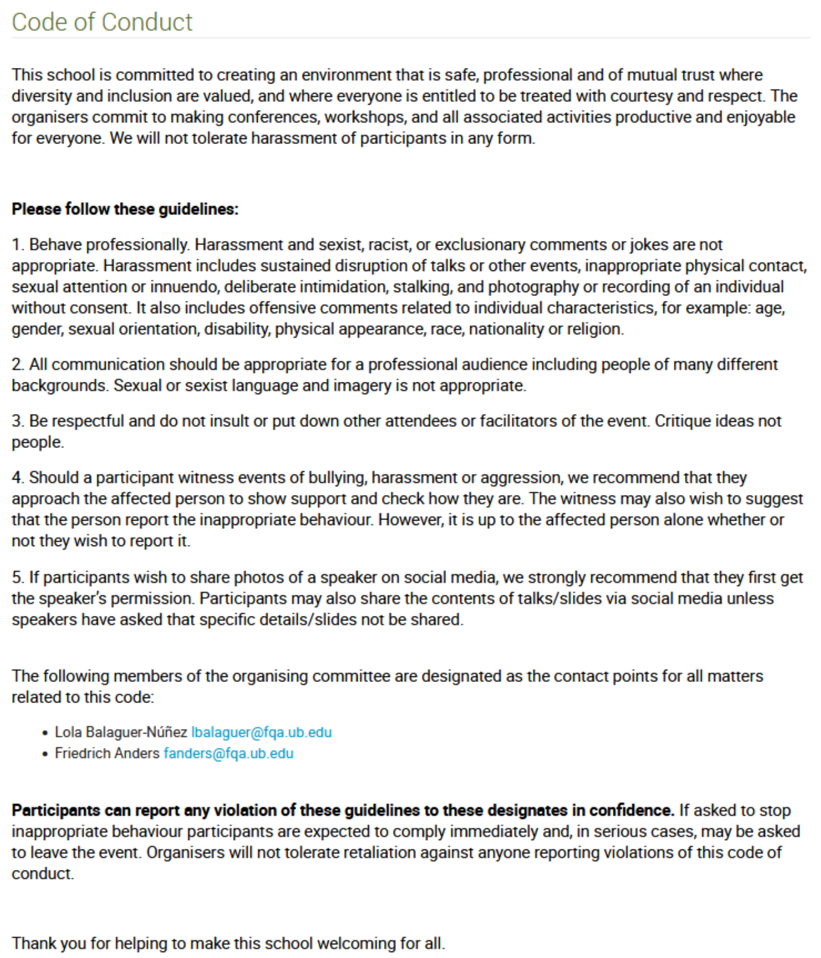
Generated on May 31, 2025

The Astropy Project is made both by and for its users, so we welcome and encourage contributions of many kinds. Our goal is to keep this a positive, inclusive, successful, and growing community that abides by the Astropy Community Code of Conduct.

For guidance on contributing to or submitting feedback for the Astropy Project, see the contributions page. For contributing code specifically,

Code of conduct

- Mainly aimed at enforcing basic common-sense rules
- Sometimes useful to:
  - Avoid abuse of power/authority
  - Avoid discrimination against minorities
  - Promote the use of professional language/behavior
- Typical in other areas too



Code of Conduct

This school is committed to creating an environment that is safe, professional and of mutual trust where diversity and inclusion are valued, and where everyone is entitled to be treated with courtesy and respect. The organisers commit to making conferences, workshops, and all associated activities productive and enjoyable for everyone. We will not tolerate harassment of participants in any form.

Please follow these guidelines:

1. Behave professionally. Harassment and sexist, racist, or exclusionary comments or jokes are not appropriate. Harassment includes sustained disruption of talks or other events, inappropriate physical contact, sexual attention or innuendo, deliberate intimidation, stalking, and photography or recording of an individual without consent. It also includes offensive comments related to individual characteristics, for example: age, gender, sexual orientation, disability, physical appearance, race, nationality or religion.

2. All communication should be appropriate for a professional audience including people of many different backgrounds. Sexual or sexist language and imagery is not appropriate.

3. Be respectful and do not insult or put down other attendees or facilitators of the event. Critique ideas not people.

4. Should a participant witness events of bullying, harassment or aggression, we recommend that they approach the affected person to show support and check how they are. The witness may also wish to suggest that the person report the inappropriate behaviour. However, it is up to the affected person alone whether or not they wish to report it.

5. If participants wish to share photos of a speaker on social media, we strongly recommend that they first get the speaker's permission. Participants may also share the contents of talks/slides via social media unless speakers have asked that specific details/slides not be shared.

The following members of the organising committee are designated as the contact points for all matters related to this code:

- Lola Balaguer-Núñez lbalaguer@fqa.ub.edu
- Friedrich Anders fanders@fqa.ub.edu

Participants can report any violation of these guidelines to these designates in confidence. If asked to stop inappropriate behaviour participants are expected to comply immediately and, in serious cases, may be asked to leave the event. Organisers will not tolerate retaliation against anyone reporting violations of this code of conduct.

Thank you for helping to make this school welcoming for all.

Contributing guidelines

- How to report issues. What information should be included (usually based on a template)
- How to submit pull requests. To ease the reviewing process:
  - How to fork the repository
  - How commit messages should look like.
  - How to describe the pull request (usually based on a template)
- Code style and formatting rules. To ensure a consistent code formatting (e.g., naming variables, files, functions,...)
- Testing guidelines. To validate contributions:
  - How to run tests locally
  - How to write new tests
  - Required test coverage
- Documentation guidelines

Index

1. Introduction
2. Management in space software projects
3. Management in open-source software projects
4. Summary

Take away messages

- **People matters.** Software project management is devoted to work with people
- **Tailoring.** Choosing the proper management system is essential
- **ECSS.** A standard exists to manage ESA software projects
- **Open source.** Open-source software projects also require management

Thank you for your attention

Francesc Vilardell Sallés
francesc.vilardell.salles@gmv.com