# Analytics Products Process Standards

## Standards Ratification

Ratified standards can be enforced in one of five ways:

**Aspirational:**  The team agrees the standard makes sense, but has not yet agreed to enforce it in any fashion.

**Voluntary:**  Developers apply the standard at their discretion.

**Opportunistic**:  Developers apply the standard throughout a tool as soon as they create or modify the tool.

**Selective**:  Product management and/or engineering identifies specific (presumably high-value) cases to apply the standard to, in a concerted effort.

**Comprehensive**:  Product management and engineering agree to apply the standard to all tools, in a concerted effort.

## General Process Standards

|  |  |  |
| --- | --- | --- |
| **ID** | **Description** | **Purpose** |
| Pr1 | Assign the same two developers (primary and secondary) to all development tasks for a given tool in a given PI, to the extent practical. | Minimize context switching and maximize developer expertise/ownership of tool code. |
| Pr2 | Perform all tasks for a given tool in a connected sequence (per developer) within a given PI, to the extent practical. | Minimize context switching and avoid repeated QA effort. |
| Pr3 | Perform all tasks for a given tool in dependency order within a given PI. | Avoid throwaway coding that manages out-of-order tasks. |
| Pr4 | Complete a whole task before starting another tasks. | Minimize context switching. |
| Pr5 | Save slacks for real emergencies. Treat needing to use slack to fulfill commitments as a planning or execution failure requiring retrospective review. Plan to use slack as innovation time. | Avoid overcommitting (and all it entails). Reserve 20% of our time for deep innovation. |
| Pr6 | Start hard and uncertain tasks early in PIs. | Reduce risk of failure to deliver on schedule. |
| Pr7 | Buy it when we can, and otherwise build it once. In particular, buy code-analysis tools that solve specific classes of code-quality problems. | Avoid reinventing wheels, and catch bugs as early as possible. |
| Pr8 | Automate repeatable work the first time we perform the task. In particular, automate all functional, performance, and scalability tests. | Avoid reinventing wheels, and catch bugs as early as possible. |
| Pr9 | Trace every standard to at least one task in the standard process. | Operationalize standards. |
| Pr10 | Trace every standard to exactly one review task in the standard process. | Avoid repeated application of standards. |
| Pr11 | Limit every review task to a single set of standards and concerns. | Separate standards enforcement from pair programming; avoid discretionary reviews; limit review feedback to increments that can be processed without producing delay. |
| Pr12 | Do not start work on a task until its acceptance criteria have been fully specified in writing (in a standard, story, or design document). | Avoid design uncertainty likely to entail code rework and delivery delay. |
| Pr13 | Eliminate specific classes of technical debt either wholesale (across all AP tools) systematically, or opportunistically (touch it, fix it). | Reduce technical debt systematically, and manage it according to the business impact of specific classes of technical debt. |
| Pr14 | Only create technical debt planfully (to the extent we’re aware we’re doing it). | Avoid creating work without product management’s consent. |
| Pr15 | Estimate task duration with sufficient precision to deliver to schedule all committed work on time, without using slack (see Pr5). | Avoid overcommitting. Estimate time requirements scientifically. |
| Pr16 | Publish in peer-reviewed and professional journals, or write patents, quarterly (as a team). | Build Alteryx’s intellectual property and reputation in the data-science community, and build team members’ resumes and careers. |
| Pr17 | Meetings must always have an explicit purpose. | Avoid wasting time. |
| Pr18 | All tool development must start with a kickoff process that includes the lead developer, the product owner, creative services, and Q/A. The kickoff process must specify the target user persona(s), any mathematical formalisms, minimal required functional inputs and outputs, the high-level UI design, and the high-level test plan. | Specify the tool with sufficient precision to avoid code rework. |
| Pr19 | All bug fixing must start with a review process that describes the bug’s symptoms and severity, replicates the bug, and specifies what functionality the bug fix must include. | Specify the bug-fixing work with sufficient precision to avoid code rework. |
| Pr20 | Requests from customer-facing organizations for team resources will go through product management to team management. Other team members need not monitor customer-support communication channels for e.g. escalations. Communication with customers on specific support issues will be indirect, through the support organization. Any direct communication will be anonymized. | Avoid needlesss context switching and use of engineering resource for non-engineering activities. |

## Process Outline

1. Define the business problem.
2. Review the literature.
   1. Review the peer-reviewed literature.
   2. Review the community literature.
3. Define the mathematical approach.
4. Conduct the math review:
   1. Announce the review two business days in advance.
   2. Everyone in AP and AA can attend.
   3. The primary reviewer must review within three business days.
5. Define the architecture.
6. Conduct the architecture review (AP and AA, two days plus three days).
7. Conduct Creative Services kickoff.
8. Define the user interface.
9. Conduct the UI-design review (AP, Creative Services, and QA; two days plus two days).
10. Specify tests.
11. Conduct the test-design review (AP, two plus two).
12. Code tests:
    1. unit
    2. functional
    3. regression
    4. performance
    5. scalability
    6. (QA) user.
13. Code the product.
14. Execute/perform tests:
    1. unit
    2. functional
    3. regression
    4. performance
    5. scalability
    6. (QA) user.
15. Conduct the code review (AP, two plus two).
16. Refactor and re-test.
17. Release:
    1. integration
    2. security.