# Analytics Products Condition-Management Standards

This document treats **condition management** (CM) as a set of practices designed to prevent or handle errors and other unwanted runtime conditions. When we distinguish CM from **error management** (EM), we mean the set of CM practices (broadly speaking) other than EM practices.

Our goals in condition management are to

1. prevent unwanted runtime conditions
2. as early as possible in the execution path
3. in a way that avoids or minimizes input revision
4. while honoring user-experience (UX) idioms.

## Presumed Development Target

This standard assumes development of a tool whose execution has the following steps:

1. Prepare execution environment.
2. Elicit end-user input.
3. Transform end-user input.
4. Read data.
5. Transform data.
6. Fit model.
7. Present results to end user.
8. Save model.
9. Save results.
10. Output results.

## Available Error-Management (EM) Tactics

When an error occurs, we can handle the error using any of the following tactics:

1. Catch the specific error upon input, elicit corrected input, and re-try.
2. Catch the specific error upon input, cleanse it, and continue (possibly with warning or notification).
3. Catch the specific error upon input, delete the record, and continue (same deal).
4. Catch the error with a general-purpose error-catching mechanism (e.g. a try-catch block).
5. Not handle the error.

## Available Condition-Management (CM) Tactics

We may apply any of the following condition-management tactics:

1. Force correct input.
2. Apply weak validation and re-try.
3. Apply strong validation, prompt, and re-try.
4. Warn about unexpected results or events.
5. Fail gracefully.
6. Warn gracefully.
7. Warn and prompt to alter execution.
8. Warn gracefully and alter the model.
9. Log each condition occurrence.
10. Warn and update environment.

## General CM Standards

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **Description** | **Purpose** | **Enforcement**  **Steps** | **Enforcement**  **Mode** | **Ratification**  **Date** |
| CM1 | Our tools will honor Alteryx UI idioms for communicating fatal errors, errors, warnings, and messages. | Our overall CM strategy should honor Alteryx UI idioms, so customers have a consistent experience regardless of tool type. | code review  UI testing | opportunistic | PI4 |
| CM2 | In particular, interpreter crashes will be treated as errors. | An interpreter crash is a non-catastrophic error affecting a single tool within a Alteryx dataflow (assuming the Alteryx engine itself does not crash); Alteryx treats single-tool failures as errors. (This is a case of CM1.) | UI testing | comprehensive | PI4 |
| CM3 | Model- and data-quality issues will be communicated through UI features that are more sophisticated than those in CM1. | Model and data quality do not reflect execution imperfections but degrees of modeling effectiveness. Fine-grained, contextualized UI features expressing these concepts will support our goal of extreme usability. | code review  UI testing | opportunistic | PI4 |
| CM4 | CM and EH standards requiring UI elements to force appropriate input may be enforced after e.g. an API call through any of the post-input CM tactics, as appropriate, but shall not go unhandled. | The intention here is to make API behavior as consistent as possible with UI behavior, for the sake of usability and testability. | code review  unit testing  functional testing | opportunistic | PI4 |
| CM5 | Features beyond model fitting or scoring proper that require significant execution time should be optional/selectable. | Maximize performance and let the end user choose which non-necessary modeling results they’re interested in (or can comprehend). | design review  UI testing | selective plus opportunistic | PI4 |

## Step-Specific Standards

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **Condition Management** | **Error Handling** | **Enforcement**  **Steps** | **Enforcement**  **Mode** | **Ratification**  **Date** |
| CM Step 1 | 1. Check for required environment components, and apply CM tactic 5 or 10 if a component is missing, out of date, or fails to load. | 1. EH tactics 4 and 5 throughout | code review  UI testing | opportunistic | PI4 |
| CM Step 2 | 1. Present default values. 2. Enforce security. 3. Validate input format. 4. Validate input semantics, e.g. bounds, allowed values, and column types. | 1. EH tactic 1 throughout | design review  code review  unit testing  functional testing  UI testing | comprehensive | PI4 |
| CM Step 3 | 1. Apply CM tactic 3 throughout. | 1. EH tactic 1 throughout | unit testing  functional testing  UI testing | selective plus opportunistic | PI4 |
| CM Step 4 | 1. Validate metadata data types and apply CM tactic 3. 2. Bounds-check p and apply CM tactic 6 or 7. 3. Validate data types and apply CM tactic 5 or 7. 4. If the analytical algorithm does not handle nulls, apply CM tactic 7 by offering to impute some sort of central value. 5. If n becomes excessive (in a context-specific sense), apply CM tactic 5 or 6. 6. Check that the dependent variable is not listed among the independent variables, and apply CM tactic 6, 7, or 8. | 1. EH tactics 1-2 throughout | unit testing  functional testing  UI testing | selective plus opportunistic | PI4 |
| CM Step 5 | 1. Check that values are within a transformation’s domain and apply CM tactic 2 or 3, and optionally tactic 9. 2. Otherwise, assume that BI-style transformations have occurred prior to data arrival, and apply CM tactic 4 or 5. 3. Communicate data-quality metrics as both data and visualizations to all classes of end user. | 1. EH tactics 2-3 throughout | unit testing  functional testing  UI testing | selective plus opportunistic | PI4 |
| CM Step 6 |  | 1. Abort fitting if fitting execution time exceeds a configurable threshold, and apply CM tactic 5. 2. Simply apply CM tactic 5 for all other fitting-algorithm errors. | functional testing  UI testing | comprehensive | PI5 |
| CM Step 7 | 1. Display any available variable-importance/selection results in descending order of variable importance/inclusion. 2. Only display variable-importance metrics to expert users. 3. For expert users, display standard visualizations depicting common model-fitness metrics. 4. For non-expert users, summarize model quality with heuristic visualizations. 5. For tools performing model aggregation or selection, only communicate these results to expert users. For non-expert users, simply communicate the fact that model aggregation/selection has occurred. | 1. Adhere to Alteryx UI-design standards regarding failure to render a visualization. | design review  code review  UI testing | 1-4, selective (declarative R) plus opportunistic;  5, aspirational | PI4 |
| CM Step 8 |  | 1. Apply EM tactic 1 throughout. | unit testing  functional testing  UI testing | selective plus opportunistic | PI4 |
| CM Step 9 |  | 1. Apply EM tactic 1 throughout. | unit testing  functional testing  UI testing | selective plus opportunistic | PI4 |
| CM Step 10 |  | 1. Apply EM tactic 1 throughout. | unit testing  functional testing  UI testing | selective plus opportunistic | PI4 |