<table>
<thead>
<tr>
<th>Area</th>
<th>Subdivision</th>
<th>Item</th>
<th>Criteria</th>
</tr>
</thead>
</table>
| Programming     | Basic coding| Code                                                                | Informative names that adhere to language conventions  
Consistent and attractive layout (spacing, indentation, etc)  
Judicious use of comments, especially for failure paths  
Reasonable function and module length  
Minimal duplication of code  
No magic numbers; constants defined in appropriate scope  
Idiomatic use of language, including avoidance of bad parts  
Appropriate and skillful use of advanced language features  
Appropriate use of known algorithms and data structures  
Appropriate use of platform and third-party libraries  
Chations for borrowed code and ideas |
|                 | Modularity  | Code                                                                | Code sensibly divided into modules and files  
Namespace, structured and coherent  
Separation of concerns (eg, presentation/content)  
Model classes encapsulate representation and functions over data  
Clean and simple module interfaces  
Functions don’t have too many arguments  
Data types immutable when appropriate  
No exposure of representations  
Abstractation barriers not violated  
Inter-module dependences controlled |
|                 | Specifications |                                                                 | Design decisions localized as much as possible  
Succinct but informative specifications for public interfaces  
Cohesive specification: avoidance of disjunctive behaviors  
Separation of results returned from side effects  
Declarative: describes what the behavior is, not how implemented  
Error and exceptional cases described, separately from normal cases  
Preconditions given, eg on session state |
|                 | Verification | Runtime assertions                                                  | Runtime assertions to check non-trivial expectations  
Representation invariants for abstract types  
Schema invariants declared, maintained (& checked if appropriate)  
Repeatability suite of tests for key methods of service interfaces |
|                 | Security     | Code                                                                | Appropriate use of security mitigations (eg, sanitization)  
Access control mechanisms implemented, as relevant  
Safe defaults used |
| Design          | Overview     | Motivation                                                          | Brief description of system to be built  
Key purposes (what problems does it solve? why should it exist?)  
Each purpose summarized in a short sentence and then explained |
|                 | Design essence | Concepts                                                            | Key concepts, each with: name, purpose, operational principle  
Concepts have short and memorable names  
Any anticipated limits: problems that concept design will avoid  
Purposes explain what concepts are for  
Operational principles are scenarios showing how purpose is fulfilled  
Concepts capture central design ideas, not routine notions  
Concepts are not just actions or user interface components |
|                 | Design essence | Data model                                                          | Abstract model of application state in diagrammatic form  
Explanations of any non-obvious elements  
Textual constraints included for constraints not expressible in diagram |
|                 | Design essence | Security concerns                                                   | Summary of key security requirements and how addressed  
How standard web attacks (such as XSS, CSRF, etc) are mitigated  
Threat model: what assumptions you’re making about attackers |
|                 | Design essence | User interface                                                       | Wireframes for application, with explanatory notes overlaid  
Transition diagram showing flow between pages, with named actions  
Error handling shown, for validation errors and anticipated failures |
| Team Work       | Plan         | Stakeholders                                                        | List of stakeholders (users, operators, others) and their roles  
Calendar of intermediate and final milestones for tasks |
|                 | Plan         | Tasks                                                               | List of tasks, expected effort, allocation to team members  
Identifiability and ease of use |
|                 | Plan         | Minimum viable product                                              | Identification of minimum viable product for next release  
Subset of concepts to be included  
Issues postponed (eg: security mitigations, user interface elements)  
Provides real value to users  
Provides opportunity for feedback  
On path to full product |
|                 | Team contract | Team contract                                                       | Expected level of refinement and effort for each team member  
Personal goals for each team member  
Frequency, length and location of team meetings  
How quality of work will be maintained  
How tasks will be assigned, and what to do if deadlines are missed  
How decisions will be made and disagreements resolved |
|                 | Meetings     | Agenda                                                              | One agenda for each meeting  
Agenda prepared in advance of meeting  
One report for each meeting, prepared in advance  
Summaries progress since previous meeting  
Identifies achieved and missed milestones  
Identifies difficulties encountered  
Identifies changes to problem analysis or design |
|                 | Reflection    | Peer review                                                         | Constructive but candid evaluations of teammate performance  
Evaluation of project from team planning perspective |
|                 | Reflection    | Lessons learned                                                     | Summary of key lessons learned |