**CHECKLISTS**

**A.1 Introduction**

Checklists are an integral part of successful Inspections. They provide the inspectors a focus that is derived from the organization’s history of problems to avoid. They serve to educate when the inspectors are ﬁrst using them. They should evolve as the organization learns about common defects that can be added to the checklists. They can change as true Defect Prevention is put into practice. Checklists serve to remind Producers and inspectors of some things the mind overlooks or forgets.

Some organizations such as the SEI have suggested a checklist of checklists [SEI89] to assure that each checklist is addressing common intents for:

* Completeness
* Correctness
* Style
* Rules of construction
* Multiple views or perspectives reﬂected in the product
* Metrics to be gathered
* Technology

This may be more than is necessary, but if it helps, fine. My view is that checklists need to be complete but as short as possible. I prefer one-page checklists, but in some situations two pages may be needed. Checklists that get too large do not get used and are not well thought out. This was a reason why the checklists in the VTAM study were not as successful as expected and one reason why there was regression in each of the next three VTAM releases.

The programmers just put them aside. We thought we had done a good job of creating checklists that addressed all possible and meaningful defects, but we missed the point that checklists must be useful for them to be used.

They do not need to address every possible defect that can be made; this is what language manuals do. Checklists should be selective and focus on the key defect possibilities. But in safety critical software the checklists may be larger for specific legal and insurance reasons.

Most software does not have the same constraints as safety critical software. The choice is yours, but keep the programmers involved in feedback on the usefulness of the checklists. Use the data to determine which items on the list could be dropped, what may be missing, and what should be added.

Checklists can be categorized into two types: Work Products and Quality Attributes. Each type can be classified into sub-types, for example:

**Work Products**

* Requirements specifications
* High-level design
* Low-level design
* Code
* Plans

**Quality Attributes**

* Usability
* lnstallability
* Performance
* Maintainability

Checklists vary throughout the public literature. It is not my intent to try to deﬁne a universal checklist. I am not sure one can be defined, and if it were, it would not be accepted by most users. Therefore, this appendix supplies checklists as samples only. The reader can use them as is, modify them, or build their own. I expect the checklists will be modified and improved.

Over time, an organization will eventually have its own unique checklists.

**A.2 Use of Checklists**

Checklists serve best when they are product and domain speciﬁc. Checklists provide guidance for the types of defects that a software organization considers most prevalent and critical. The checklists vary by work product to be inspected. The checklists should be *tailored* based on proyect-specific knowledge by adding or deleting checklist items. Checklists are not a replacement for style guides, which are used to create the work products. However, Producers could use Inspection checklists when developing the work product.

The Moderator is responsible for distributing or ensuring that the inspectors have the current and appropriate checklists for Preparation and for ensuring the use of the checklists during the Inspection Meeting. An initial set of checklists should have been provided to each inspector during their Inspections training. The latest version should be made available on the web or master document library.

Inspectors are responsible for using the checklists during Preparation and the Inspection Meeting.

SQA can be responsible to ensure checklists are being used during Inspections and should review the appropriateness of the checklist contents when they are created and modiﬁed.

Checklists should be baselined in the Process Asset Library as part of the Organization Standard Software Processes and defined as a project-level process, when the checklist is tailored speciﬁcally for the project.

How a checklist is used is an individual choice. Some inspectors read it before the Preparation to give them a focus for the Inspection. Some use it at the end to check that they have addressed all relevant checklist items. Some do both. As an inspector gains more practice with Inspections the dependency on the checklist will decrease, since much of what is on the checklist will have found its way into the inspector‘s memory and practice. Nonetheless, it is helpful to review the checklist from time to time.

It is suggested that the Moderator ask if the checklist was used during Preparation and then review the checklist items at the end of the Inspection Meeting to ensure it has been sufficiently addressed in the Inspection Meeting. This is most helpful when Inspections are first put into practice. It should not be unexpected that the Inspection team finds they need to go back to some part of the work product when reviewing the checklist items at the end of the Inspection.

This case is optional, and is useful during start up. One of the purposes the checklist serves is coverage mapping.

Checklists are used:

1. During Preparation by inspectors to ensure that the material has been sufficiently examined for the Inspection Meeting.
2. At the conclusion of the Inspection Meeting by the Moderator to ensure the work product has been sufficiently inspected.

Data from Inspections should be reviewed periodically to update the checklists and make them more appropriate to the work product being inspected. This is a responsibility of the SEPG or Inspection Coordinator. A checklist that does not change cannot be as useful for improving effectiveness.

Checklists are typically intended to help an inspector find a defect, but other goals besides defect detection have been included. This is a choice, but then the Inspection process may be applied for other purposes than finding defects in the product or process.

Gilb provides a list of the characteristics of checklists, some of which I include below:

[GIL93]

* “Checklists must ultimately be derived from the rules of the process which itself is being checked by Inspection;
* Checklists should be kept updated to reﬂect experience of frequent defects;
* A set of questions for one document type should never exceed one single page (about 25 items);
* The checklist does not need to contain every possible question;
* A checklist should concentrate on questions which will turn up major defects;
* Checlists are part of the on-the-job training of a checker.”

Correspondingly there are items that checklists should avoid including: [BRY99]

* Checks that can be done with automated tools; e.g., Lint
* Outdated checklist items
* Items better suited as entry/exit criterion; e.g., “Is the compilation listing free of fault messages?”
* Items that are too general; e.g., is the code maintainable?

**A.3 Checklist Examples**

The following work product checklists are provided as examples only. The specific checklists to be used by any organization can be obtained initially from these lists and modiﬁed over time based on data from practiced Inspections.

These checklists are compiled from many sources, almost too numerous to quote. Most are available one way or the other in the public domain in books, articles, conference proceedings, web postings, etc. Many sources the reader may refer to are well documented in Brykczynski's survey of 117 checklists. [BRY99] Many of the references included in the bibliography of this book contain checklists of one type or another. These will all be noted with a <CKL> at the end of the reference.

These checklists should not be viewed as necessarily complete, but as examples of items found on checklists.

Checklists should not address code style. Style guides provide value for consistent form and format of presentation, and the Producer might use one to create a work product. Correctness is the focus of checklists. They must be used when required, but if they provide options, then the Producer has choice. Every newspaper and magazine has a style guide, which is the best? Which is correct, if that is a question that even applies? We assume style guides are correct. Well, they at least can lead to consistency and in that they provide their primary service.

Remember that style guides typically lead to minor defect identification unless stated otherwise.

Separate and speciﬁc checklists are used for each of the different types of Inspections:

* Requirements Speciﬁcation (RS)
* Architecture (AR)
* High Level Design (HLD)
* Low Level Design (LLD)
* Code (CD)
* Unit Test Plan (UTP)
* Unit Test Case (UTC)
* Function Integration Test Plan (FTP)
* Function Integration Test Case (FTC)
* System Integration Test Plan (STP)
* System Integration Test Case (STC)
* User Documentation (DOC)
* Problem Reports (PR)
* Change Requests (CR)

Example:

REQUIREMENTS SPECIFICATION CHECKLIST

1. The format and structure of the requirements document maps to the organization’s standard for requirements documents.
2. Any deviations to format and structure have proper approval and justification.
3. The requirements statements are not inconsistent.
4. The requirements are traceable to customer input or problem statements to be addressed by the project.
5. The requirements map to the needs analysis.
6. The requirements are stated clearly and cannot be misinterpreted.
7. All terms used, which can have more than one meaning, are qualified so that the desired meaning is readily apparent; e.g., review, area.
8. The customer proﬁles are clearly defined.
9. The hardware environment is completely defined, including engineering change levels and any constraints.
10. The pre-requisite and co-requisite software and firmware is identified, including release levels and any constraints.
11. The installation and migration requirements are completely defined.
12. The performance criteria are quantified such that they are testable.
13. All of the requirements can be verified during at least one test activity.
14. Error recovery and backup requirements are completely defined.
15. The requirements are expressed such that they can subsequently be modiﬁed or changed.
16. All materials required for the requirements Inspection have been received and are in the proper physical format.
17. The requirements document is complete; it deﬁnes the known customer needs.
18. The human interface meets project standards.
19. The infrastructure has been addressed, e.g., backup, recovery, checkpoints.
20. Reliability, serviceability, maintainability, and performance objectives have been identified.
21. Security considerations have been identified, if applicable.
22. The requirements are an adequate base for design.
23. Requirements are complete, correct, consistent, and unambiguous.
24. Quality attributes have been defined clearly.
25. Requirements are stated at the requirements level; i.e., they are not design solutions.

Some of the responses to these items can be gray; e.g., item 6 (“cannot be misinterpreted”). If an inspector says yes it can be, then the team needs to discuss to understand if this is generally true or limited to the one inspector. The team can come to a conclusion that this is a valid misunderstanding or is isolated to one inspector. This is a gray area where a defect may be in the eyes of some but not all.

**Fuente:**

Appendix A: Checklists

Radice, R.. (2002). High quality low cost software inspections. Massachusetts, EUA: Paradoxicon Publishing. Pp. 421-426

**Otras Fuentes:**

[A survey of software inspection checklists](https://doi.org/10.1145/308769.308798)

## [Improving Quality Through Software Inspections](http://www.processimpact.com/articles/inspects.html)

[Requirements Inspection](https://www.google.com.mx/url?sa=t&rct=j&q=&esrc=s&source=web&cd=13&ved=0ahUKEwjS0Jq7mMPZAhXGtlMKHaz7Ctg4ChAWCD0wAg&url=http%3A%2F%2Fwww.uccs.edu%2FDocuments%2Ftboult%2FSRS-walkthrough.doc&usg=AOvVaw0ApNqy9QZaUK6dOlLrMXMV)

Entity Relationship Checklists

|  |
| --- |
| The diagram is easy to read and understand. |
| Entity definitions are clear, unambiguous and meaningful. |
| All instances (rows) of the entity have the same characteristics and are subject to the same rules. |
| Each entity has a unique identifier identified with an IDENTIFICATION attribute. |
| Each entity represents a set of similar things (does not represent a class of things that are mixed together - Look for "or" in the definition). |
| There are no redundant relationships. |
| There are no many-to-many relationships. |
| Data constraints for bi-conditional relationships (optional to optional) are documented in the comments section of the child entity. |
| Data constraints for one-to-many relationships optional on the one side are documented in the comments section of the child entity. |
| Attributes are identified for each entity (except for associatives). |
| Attribute definitions are clear, unambiguous and meaningful. |
| There are no attributes that are really foreign key relationships. |
| All attributes are functionally dependent on the key, the whole key and nothing but the key. |

Use Cases Checklists

|  |
| --- |
| The use-case name is meaningful and un-ambiguous |
| The brief description clearly describes the primary goal of the use case |
| Associated actors and information exchanged are clearly defined |
| Pre-conditions have been specified |
| The Basic Flow and Alternate Flows are complete, correct and consistent |
| Post-conditions have been specified |
| Applicable non-functional requirements have been captured |

Design and UML Class Diagrams Checklists

|  |
| --- |
| Is the class diagram complete? Does the business object contain all the information, so that sender and receiver can carry out their activities? |
| Did you check that the business object contains only data elements that can be interpreted by outsiders? |
| Did you check whether it is possible to use standard messages? |
| Are the relationships labeled in a meaningful manner? Are the directions of the arrows correct? |
| Is the class diagram correct? Intensive reading of the class diagram together with knowledge carriers and running through each scenario will bring most mistakes to light. |