Universität Hannover
Fakultät für Elektrotechnik und Informatik
Institut für Praktische Informatik
Fachgebiet Software Engineering

Einsatz von Checklisten in der Analytischen Qualitätssicherung

Studienarbeit

im Studiengang Mathematik mit Studienrichtung Informatik

von

Mariya Skachkova

Prüfer: Prof. Dr. Kurt Schneider
Betreuer: Dipl.-Math. Thomas Flohr

Hannover, Juli 2006

Erklärung

Hiermit versichere ich, dass ich die vorliegende Studienarbeit selbstständig und ohne fremde Hilfe verfasst habe und keine anderen als die in der Arbeit angegebenen Quellen und Hilfsmittel verwendet habe.

Hannover, 03.06.2006 Mariya Skachkova

Danksagung

Ich bedanke mich bei Herrn Dipl.-Math. Thomas Flohr für die hervorragende Betreuung meiner Studienarbeit.

Inhaitsverzeichnis

1. Einleitung

Das Ziel der Software-Technik ist die Bereitstellung von Methoden, Sprachen und Werkzeugen zur effizienten Entwicklung messbar qualitativ hochwertiger Software. Ein sehr wichtiges Werkzeug dafür sind die Checklisten. Checklisten werden verwendet, um beispielsweise Information über Schritte zu liefern, die ausgeführt werden müssen, um eine Aufgabe zu erreichen. Sie helfen Entwicklern und Programmierern sicher zu sein, dass die wichtigen Faktoren oder Qualitätsmerkmale eines Problems bedeckt sind. Die Checklisten werden auf Grundlage von Erfahrungen entwickelt und überarbeitet. Durchaus hilfreich sind sie auch im Risikomanagement. Anhand einer Checkliste mit den möglichen Risiken ist es viel einfacher Probleme frühzeitig zu erkennen, zu minimieren oder sogar vollständig zu beseitigen.

Diese Studienarbeit soll eine ausführliche Beschreibung der verschiedenen Ansätze für Checklisten in der Analytischen Qualitätssicherung geben. Dafür wird hier nur auf die Beispiele eingegangen, die man in der Literatur finden kann. Wie das Spektrum möglicher Anwendungen von Checklisten unübersichtlich groß ist, so sind auch die gefundenen Prüflisten unterschiedlich aufgebaut. In erster Linie ist das Ziel der Arbeit, die verschiedenen Formen von Checklisten aufzuzählen und deren Aufbau anhand der Beispiele zu analysieren. Dabei werde ich auch in die Gründe für die verschiedenen Typen von Checklisten eindringen. Weiterhin wird das Checklistenbasierte Lesen mit den anderen Lesetechniken verglichen, die in den Inspektionen benutzt werden.

1.1. Hintergrund

Die Qualitätssicherung ist eine der wichtigsten Aufgaben bei der Entwicklung von komplexer, umfangreicher Software. Eine ökonomisch und technisch sinnvolle Qualitätssicherung fordert die Verwendung angepasster Lösungen. Die Software Qualitätssicherung kann in organisatorischen, konstruktiven und in analytischen Maßnahmen gegliedert werden. Ziel dieser Maßnahmen ist es, Fehler möglichst frühzeitig im Software-Lebenszyklus ausfindig zu machen.

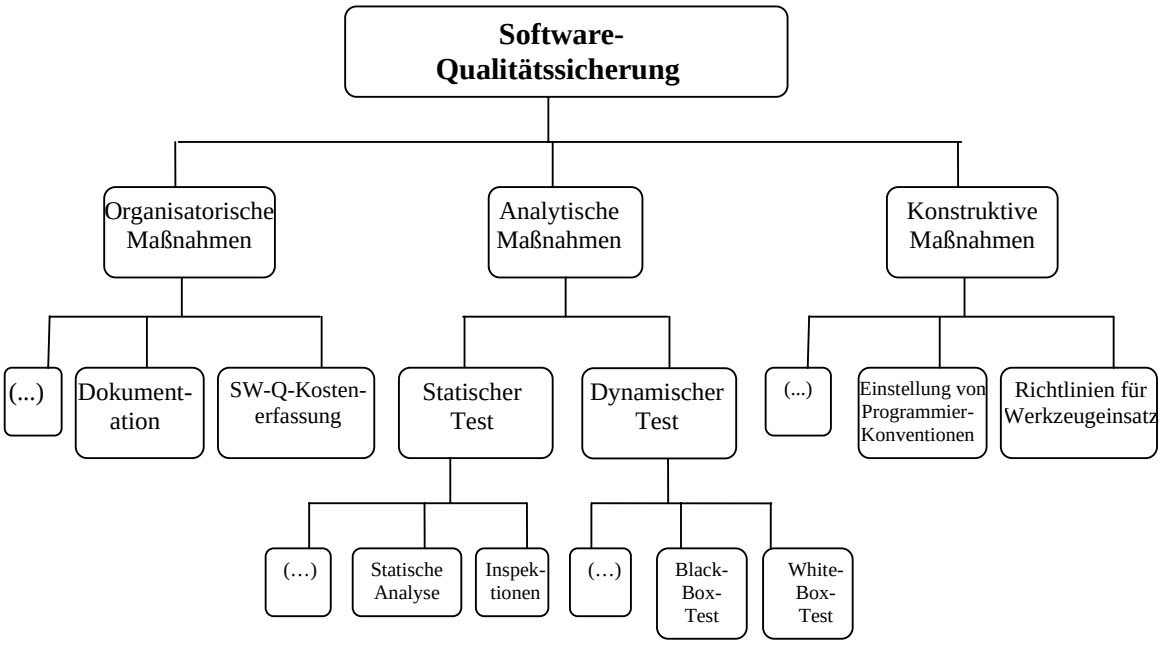


Abbildung 1: Maßnahmen in der Software-Qualitätssicherheit

Analytische Qualitätssicherung spielt in der Softwareentwicklung eine wesentliche Rolle. Es werden bis zu 50% der Entwicklungsaufwände allein für den Softwaretest aufgebracht; bei sicherheitskritischen Systemen sogar noch mehr. Trotz dieser hohen Bedeutung gibt es relativ wenige Studien, die sich empirisch mit der Anwendung von Maßnahmen der analytischen Qualitätssicherung in der Softwareengineering insbesondere im deutschsprachigen Raum befassen.

Die analytischen Maßnahmen werden u. a. durch Testen von Software realisiert. Sie prüfen, ob die gesetzten Maßnahmen stattfinden, ob sie wirken und inwieweit gesetzte Qualitätsziele erreicht sind. Analytische Qualitätssicherungsmaßnahmen setzen erst nach Fertigstellung des Dokumentes an. Die Analytischen Maßnahmen können zum einen in statische, zum anderen in dynamische Tests gegliedert werden. Zu den statischen analytischen Maßnahmen gehören die Inspektion und die Analyse.

Unter *Inspektionen* versteht man die manuellen Prüfungen und Untersuchungen von Dokumenten, Spezifikationen und Programmen, die sich in Reviews/Audits, Walkthroughts und formale Inspektionen gliedern lassen. Bei der Inspektion wird das zu prüfende Software-Produkt in systematischer manueller Weise auf Fehler, Qualitätsmerkmalle und Einhaltung konstruktiver Richtlinien untersucht. In der Literatur existieren nur wenige Anleitungen, wie

die eigentliche Fehlersuche optimiert werden kann. Die Hilfsmittel für die Inspektionen sind die Lesetechniken. Sie lassen sich nach [14] definieren als eine Reihe von Schritten oder Prozeduren, deren Zweck es ist, dem Inspektor eine Hilfestellung zu geben, um ein tiefes Verständnis der inspizierten Software zu erlangen. Mit deren Hilfe soll die Produktivität des Lesens gesteigert werden und die Fehlersuche optimiert werden.

1.2. Gliederung der Arbeit

Zunächst werden in Kapitel 2 die verschiedenen Lesetechniken für Inspektionen kurz beschrieben und dann mit dem Checklistenbasierten Lesen verglichen. Dabei werden die Vorteile und die Nachteile der Checklistenmethode gegenüber den anderen Techniken dargestellt.

Im dritten Teil der Arbeit wird der Begriff Checklist definiert. Zuerst werden die Definitionen, die ich in der Literatur finden könnte, dargestellt. Dann folgt meine eigene Definition, wie ich den Begriff verstanden habe, so dass noch hier deutlich wird, wie unterschiedlich der Begriff „Checklist“ interpretiert werden kann. Was weiterhin die unterschiedliche Aufbaustruktur erklärt. Auch die Ähnlichkeit der Checkliste mit anderen Dokumenten wie z.B. die Formulare wird in diesem Kapitel erläutert.

In Kapitel 4 werden die verschiedenen Modelle der Checklisten grafisch und textuell dargestellt. Das erfolgt anhand Beispiele, die auch im Anhang zu finden sind. Außerdem werden die Vorteile und Nachteile der vorgestellten Modelle beschrieben.

Im nächsten Kapitel werden weitere Merkmalle von Checklisten erläutert, die bei der Erstellung berücksichtigt werden müssen. Am Ende des Kapitels ist auch eine Checkliste für Checklisten, die als Hilfe zur Erstellung oder Beurteilung von Checklisten dienen kann.

Die Arbeit schließt mit Kapitel 6 mit einer Zusammenfassung der Einsatzmöglichkeiten der vorgestellten Modelle.

2. Lesetechniken und Vergleiche

In diesem Abschnitt werden kurz die verschiedenen Lesetechniken und anschließend die Studien vorgestellt, die das Checklistenbasierte Lesen mit den anderen Techniken vergleichen.

2.1. Lesetechniken

Es existieren die nachstehend aufgeführten Lesetechniken:

- Ad-hoc
- Checklistenbasiertes Lesen (CBR)
- Kontrollflussorientiertes Lesen
- Lesen durch schrittweise Abstraktion
- Szenarienbasiertes Lesen
  - Fehlerklassenbasiertes Lesen
  - Funktionalbasiertes Lesen
  - Perspektivenbasiertes Lesen (PBR)

2.1.1. Ad-hoc

Die einfachste Lesetechnik ist die Ad-hoc Vorgehensweise. In diesem Fall bekommt der Inspektor keine Hilfsmittel zum Lesen. Jeder Gutachter sucht beim Lesen nach bestem Können, nach Fehlern. Diese Technik eignet sich besonders für einfache Dokumente, wie zum Beispiel eine überschaubare Benutzerdokumentation.

2.1.2. Checklistenbasiertes Lesen

Die Checklistenbasierte Lesetechnik ist zusammen mit der Ad-hoc-Lesetechnik wohl eine der am häufigsten benutzte [8,10]. Die Checklistenbasierte Lesetechnik beruht auf einem Hilfsdokument – die Checkliste, welches eine Reihe von Fragen enthält. Diese geht der

Inspektor während seiner Vorbereitung oder der Moderator während der Inspektionssitzung durch. Dadurch wird sichergestellt, dass keine Mängelart vergessen wird oder dass keine systematischen Fehler gemacht werden.

2.1.3. Kontrollflussorientiertes Lesen

Bei Kontrollflussorientiertem Lesen erfolgt eine Überprüfung des Kontrollflusses durch den Inspektor. Der Inspektor versucht den Kontrollfluss im Dokument dadurch zu verstehen, dass Variablen, Konstanten etc. mit möglichen Werten belegt werden. Anschließend untersucht der Inspektor, ob und wie sich die Werte bei Ablauf des Kontrollflusses verändern. Das beobachtete Verhalten wird mit dem spezifizierten Verhalten verglichen und mögliche Fehlverhalten werden entdeckt. Aus dem Fehlverhalten werden dann die Fehler isoliert. Diese Technik lässt sich nur auf Dokumente anwenden, die einen Kontrollfluss enthalten.

2.1.4. Lesen durch schrittweise Abstraktion

Die Technik „Lesen durch schrittweise Abstraktion“ (engl.: Reading by Stepwise Abstraction) dient dem Finden von Fehlern in Codedokumenten. Dazu identifiziert der Inspektor elementare Codebausteine wie Anweisungen, bestimmt die Funktion der Codebausteine und fasst dann diese Codebausteine zu größeren Bausteinen zusammen, deren Funktion dann bestimmbar ist. Ziel ist es, soweit funktional zu abstrahieren, bis die gebildete Abstraktion mit der Spezifikation verglichen werden kann. Bei der Erstellung der Abstraktionen und des anschließenden Vergleichs der Abstraktion mit der Spezifikation können Fehler entdeckt werden.

2.1.5. Szenarienbasiertes Lesen

Schließlich gibt es die szenarienbasierte Vorgehensweise. Hierbei werden den Inspektoren bestimmte Szenarien vorgegeben, aus deren Kontext heraus sie das Dokument oder den Code durchlesen sollen. Die Inspektion verläuft im Team von mehreren Inspektoren, die verschiedenen Szenarien bearbeiten sollen. Damit ist gesichert, dass mehrere Gesichtspunkte des Dokumentes behandelt werden, und somit mehrere Qualitätsmerkmale überprüft werden. Die Effektivität der Szenarienbasierten Lesetechniken hängt stark von dem Inhalt und Design der zu inspizierenden Software- Artefakten ab. Es gibt zum einen fehlerklassenbasierte Szenarien, bei denen bei jedem Szenario auf eine bestimmte Fehlerklasse geachtet wird. Eine Klasse könnte zum Beispiel Rekursionsfehler sein, ein anderer Modellierungsfehler. Dann gibt es Funktionalbasierte Szenarien, bei denen jeweils eine bestimmte Funktionalität betrachtet wird. Beispiele hierfür wären Drucken oder Speichern. Schließlich gibt es perspektivenbasierte Szenarien, bei denen das Dokument jeweils aus der Sicht eines anderen Beteiligten wie Programmierer, Administrator oder Benutzer gelesen wird. Aus Benutzersicht könnte die

Verständlichkeit einer Dokumentation geprüft werden, während beim Lesen aus Administratorsicht zum Beispiel die Wartbarkeit geprüft werden kann.

2.2. Vergleiche von Checklistenbasiertem Lesen mit den anderen Lesetechniken

Softwareinspektionen sind zweifellos eine der besten Art, Softwaredokumente zu überprüfen. Leider gibt es nicht sehr viele Untersuchungen über die Verbesserung dieser Technik. Es wurden Experimente durchgeführt, um die unterschiedlichen Lesetechniken zu vergleichen. Das Ziel war, die Effektivität und die Kosten für die untersuchten Lesetechniken zu bestimmen. Mit dem Vergleich könnte man bei zukünftigen Projekten die „bessere“ Methode verwenden.

Fehlerklassenbasiertes Lesen wurde in einem Experiment von Porter, Votta und Basili mit Checklistenbasiertem Lesen und Ad-hoc Lesen verglichen [23]. Sie führten 1993 das Experiment mit Studierenden der University of Maryland durch. Der Versuch hat gezeigt, dass die Effektivität von Fehlerklassenbasiertem Lesen höher als die von Checklistenbasiertem Lesen oder Ad-hoc- Lesen ist. Er hat weiterhin keinen Unterschied der Effektivität von Checklistenbasiertem Lesen(CBR) und Ad-hoc-Lesen ergeben.

1999 wurde ein Versuch durchgeführt, der das Ziel hatte, die Wirksamkeit der Methoden Checklistenbasierten Lesens und Perspektivbasierten Lesens (PBR) für Fehlerentdeckung bei Dokumenten in UML- Modell zu vergleichen [15]. Die Ergebnisse zeigen an, dass PBR wirksamer als CBR ist, (d.h. mit PBR wurden durchschnittlich 41% mehr Fehler als mit CBR wahrgenommen). Außerdem waren die Kosten für die Fehlerentdeckung mit PBR bedeutend niedriger als mit CBR.

In einem anderen Experiment [4] wurden diese zwei Lesetechniken(CBR und PBR) noch mal in der Praxis verglichen. In der Firma, in der die Untersuchung durchgeführt wurde, hatte man bis dahin bei der Softwareinspektionen CBR als Standarttechnik benutzt und man wollte, wenn die Ergebnisse besser mit der PBR ausfallen, die Technik entsprechend wechseln. Die Resultate waren sehr ähnlich mit denen von der vorher beschriebenen Untersuchung. Sie haben wie erwartet gezeigt, dass mit PBR mehr Fehler (70), im Vergleich mit der Inspektion bei der CBR benutzt war (56), gefunden wurden. Die CBR Methode war mit viel mehr Inspektoren (18) ausgeführt, die je etwa eine Stunde in Durchschnitt für die Inspektion verbrachten. Das PBR Methode wurde mit wenigen Kritikern ausgeführt(2), die je etwa 18 Stunden für die Inspektion brauchten. Allgemein wurden weniger Ressourcen mit der PBR Methode gebraucht und mehr Fehler gefunden. Ähnliche Ergebnisse sind auch in [19] beschrieben.

Eine andere Studie [16], die im 2004 publiziert wurde, hat allerdings gezeigt, dass nur wenn das Dokument, das inspiziert wurde, eine gewisse Komplexität oder Größe übersteigt, ist die PBR Methode wirksamer und effizienter als CBR. Für wenig komplexe Dokumente hat sich CBR als wirksamer und effizienter erwiesen. Das ist das Resultat von dem Vergleich der Studien an der Universität von Kaiserslautern und an der Universität von Bari. Die Resultate der Universität von Bari zeigten nicht statistisch bedeutsame Unterschiede ($P < 0,05$) zwischen den zwei Lesetechniken, jedoch hatten die Daten dieses Mal ein Tendenz zugunsten von CBR.

Allgemein kann man sagen, dass der Erfolg in verschiedenen dokumentierten Experimenten auf die jeweils gewählten Lesetechniken zurückgeführt wird, ohne jedoch die Qualität der Hilfsdokumente zur jeweiligen Lesetechnik verglichen zu haben. Allerdings ist die Checklisten- Methode die am weitesten verbreitete und am meisten benutzte Methode bei Softwareinspektionen. Leider gibt es bisher nur wenige Informationen darüber, wie eine gute Qualität von Checklisten erzeugt werden kann. In Kapitel 4 werden unterschiedliche Modelle einer Checkliste vorgestellt, die eine Hilfe bei der Erstellung von besseren und passenden Checklisten sind.

2.3. Klassifizierung vom Checklistenbasierten Lesen

Es werden bestimmte Kriterien definiert, mit denen eine Lesetechnik klassifiziert und beurteilt werden kann[13]. Die Lesetechniken lassen sich hinsichtlich der Orientierung, der Anwendbarkeit, der Wiederholbarkeit, der Anpassbarkeit (Verbesserbarkeit) und der Überlappung klassifizieren.

Die *Orientierung* gibt an, ob mit Hilfe der Lesetechnik die Korrektheit eines Produkts oder dessen Zuverlässigkeit überprüft werden kann.

Die *Anwendbarkeit* einer Lesetechnik gibt an, wie gut der Inspektor angeleitet wird, wie ein Dokument zu lesen ist und auf welche Aspekte beim Lesen zu achten ist.

Unter *Wiederholbarkeit* wird verstanden, wenn durch Anwendung derselben Vorgehensweise dieselben Ergebnisse erzielt werden können, damit es für Dritten nachvollziehbar wird, wie das Ergebnis des Lesens zustande kam.

Anpassbarkeit einer Lesetechnik bedeutet, inwiefern die Lesetechnik an die bestehenden Verhältnisse der Software-Entwicklungsumgebung angepasst werden kann.

Unter *Überlappung* einer Lesetechnik wird verstanden, inwiefern die Anwendung einer Lesetechnik dazu führt, dass beim Lesen des Dokuments von jedem Inspektor andere Fehler gefunden werden. Durch eine geringe Überlappung steigt die Wahrscheinlichkeit, dass eine

bessere Überdeckung des Dokuments erreicht wird. Bei hoher Überlappung finden alle Inspektoren annähernd dieselben Fehler. Dies führt zum einen dazu, dass eventuell Fehler im Dokument verbleiben (geringe Überdeckung), zum anderen dazu, dass der Aufwand für mehrere Inspektoren nicht gerechtfertigt ist. Bei geringer Überlappung finden die Inspektoren unterschiedliche Fehler. Daraus resultiert zum einen eine bessere Überdeckung des Dokuments, zum anderen ist der Aufwand für mehrere Gutachter gerechtfertigt.

Das Checklistenbasierte Lesen ist korrektheitsorientiert, für die Anwendung der Technik durch den Gutachter gibt es Richtlinien. Dem Inspektor wird dabei eine Anleitung gegeben, wie er beim Lesen vorzugehen hat bzw. auf welche Aspekte er beim Lesen achten muss. Die Wiederholbarkeit der Ergebnisse ist bei dem Checklistenbasierten Lesen nicht gegeben, weil es nicht nachvollziehbar ist, wie die Antwort auf eine Frage zustande kam, die Ergebnisse hängen sehr stark vom Inspektor ab. Die Beantwortung der Fragen ist vollkommen von ihm abhängig. Sie ist bei der Checklistenbasierten Lesetechnik abhängig von der tatsächlichen Technik des Lesenden, da trotz einer einheitlichen Checkliste jeder Inspektor das Lesen des Dokumentes und die Beantwortung der Fragen unterschiedlich durchführen kann. Im Extremfall kann der Lesende das gesamte Dokument Ad-hoc lesen und anschließend die Fragen der Checkliste beantworten, ohne weitere Fehler aufzudecken. Es wird kein weiteres Arbeitsergebnis außer der Liste von gefundenen Problemen gefordert. Checklisten können an die Charakteristiken der jeweiligen Entwicklungsumgebung und bei Erfahrungsgewinn angepasst werden. Die Überlappung ist bei Checklistenbasiertem Lesen hoch.

3. Definitionen

3.1. Definitionen aus der Literatur

Es gibt sehr wenige Definitionen des Begriffs „Checkliste“ in der Literatur. Es ist bemerkenswert, dass selbst in umfangreichen Wörterbüchern und Lexika, wie z.B. im Duden oder im Brockhaus keine Angaben zu finden sind. Nur in einigen EDV-spezifischen Lexika und Büchern findet man Definitionen, wobei der Begriff Prüfliste als Synonym zur Checkliste behandelt wird.

Checkliste:

„In der Schwachstellenanalyse oft angewendet, womit mögliche Schwachstellen mit Hilfe von weitergehenden detaillierten Einzelfragen relativ rasch aufgefunden werden können.“

Lexikon der Informatik [26]

„Zusammenfassung von Auswahlkriterien als Leitfaden für Hard- und Softwareentscheidungen.“

Kleines Lexikon der Informatik [31]

„A specialized set of questions designed to help checkers find more defects, and in particular, more significant defects. Checklists concentrate on major defects. A checklist should be no more than a single page per subject area. Checklist questions interpret specified rules.”

Software Inspection [10]

„Checklisten sind Fragenkataloge, die möglichst aus geschlossenen Fragen mit nur wenigen Optionen bestehen, bzw. Ankreuz- oder Anklickbahre Felder (Checkboxen) enthalten. Sie werden bei komplexen und/oder immer wiederkehrenden Fragestellungen und Aufgaben eingesetzt.“

http://de.wikipedia.org [32]

Prüfliste:

„Vollständigkeit und richtige Reihenfolge von zu einer Aufgabe gehörigen Tätigkeiten lassen sich mit Hilfe einer Liste kontrollieren, die die einzelnen Tätigkeiten in der erforderlichen Folge enthält. Solche Prüflisten sind z.B. für das Testen von Programmen sehr nützlich. Man kann dabei dem ganzen Testplan die Form einer Prüfliste geben“

Lexikon der Datenverarbeitung [19]

„Eine Prüfliste ist eine Arbeitshilfe für die Durchführung und Dokumentation von Maßnahmen in der Qualitätssicherung und zur Einschätzung von Gefährdungspotenzialen.“

http://de.wikipedia.org [32]

Die Definitionen zeigen deutlich, dass das mögliche Anwendungsspektrum von Checklisten sehr groß ist. Weiterhin wird es deutlich, wie unterschiedlich man den Begriff verstehen kann und wie viele Arten vom Aufbau einer Checkliste gibt. Alle vorgestellten Definitionen sind jedoch nicht vollständig und keine beschreibt alle Aspekte einer Checkliste.

3.2. Eigene Definition des Begriffs „Checkliste“

Eine Checkliste ist eine Liste aus Fragen oder Anweisungen und, falls es welche gibt, der dazugehörigen Antwortmöglichkeiten. Checklisten werden vor allem als Leitfaden benutzt, um Schritte oder eventuellen Risiken in den Projekten nicht zu vergessen oder Fehler im Produkt leichter zu finden. In Abhängigkeit von ihrem Einsatz werden die Checklisten unterschiedlich aufgebaut.

3.3. Checkliste ähnliche Dokumente

Es gibt viele Dokumente, die der Checkliste ähnlich sind und die als solche auch benutzt werden können.

Sieht man die Checkliste intuitiv als eine Liste an, so kann man sie als ein *Auflistungsdokument* bezeichnen. Die einzelnen Punkte dieser Auflistung sollen vor allem helfen die Unterteile einer Aufgebe nicht zu vergessen.

Der Checkliste sehr ähnliche Dokumente sind dann *Richtlinien*, die teilweise direkt als Checkliste interpretiert oder formuliert werden können. Richtlinien ihrerseits sind nicht immer klar als solche bezeichnet. Sie liegen stattdessen als Entwürfe, Vorschläge, teilweise in Form von Formularen oder Normen vor. Auch Vorschriften können Checklistencharakter haben, da sie oft verbindliche Vorgaben in gesammelter Form aufführen, ganz ähnlich wie bei einer Checkliste.

Unter dem Begriff „Checkliste“ sind auch verschiedene Arten von Dokumenten denkbar. Beispielsweise könnte man unterscheiden, ob eine Checkliste Eintragungen im Dokument verlangt oder nicht. Bei Texteintragungen würde man die Liste eher als *Fragebogen(Fragenkatalog)* bezeichnen, während man sich bei einer Checkliste vielleicht allenfalls das „Abhaken“ im Dokument vorstellen wird. Man kann sagen, dass beide Begriffe sehr ähnlich sind und oft als Synonyme benutzt werden (z.B. im [9]). Besonders wenn man berücksichtig, dass in fast alle Checklisten wenigstens das Datum, die Inspektornamen oder das geprüfte Produkt eingetragen werden.

Dann gibt es noch die etwas „grafischer“ ähnlichen Verwandten der Checkliste, die *Tabellen* und *Formulare*. Tabellen sind zunächst nur eine besondere Form der grafischen Darstellung. Spezielle Tabellen sind Entscheidungstabellen. Diese schlagen je nach Eintreffen gewisser Bedingungen Aktionen vor. Entscheidungstabellen sind ebenfalls mit Checklisten verwandt, da das Erfassen der Bedingungen im Grunde durch eine Checkliste geschieht.

Inwiefern *Formulare* Checklisten nahe kommen, hängt sicherlich stark vom einzelnen Dokument ab. Ein Formular ist ein standardisiertes Mittel zur Erfassung von Daten, genau so wie die Checklisten geben Formulare in der Regel kurze Textfelder (zum Beispiel der Name des Entwicklers, der die Checkliste ausgeführt hat, das Datum oder der Projektname) und Einfach- (zum Abhacken) oder Mehrfachauswahlfelder vor. Auf jeden Fall ist klar, dass Formulare sehr eng mit den Checklisten verwandt sind, nur die Formulare verwendet man eher zur Erfassung der Ergebnisse einer Inspektion, der so genannter Formular Prüfprotokoll. Die Checklisten helfen den Inspektoren, zu den Resultaten zu kommen.

4. Checklisten Modelle

4.1. Allgemeine Beschreibung einer Checkliste

Eine Checkliste besteht allgemein aus Checkpunkten. Ein Checkpunkt ist eine Frage und die dazugehörige Antwortvorgabe. In den verschiedenen Modellen, die in dieser Arbeit beschrieben sind, unterscheiden sich die Checkpunkte erstens in der Form der Fragen und zweitens in dem Variablentypen, der als Antwort zurückgegeben wird.

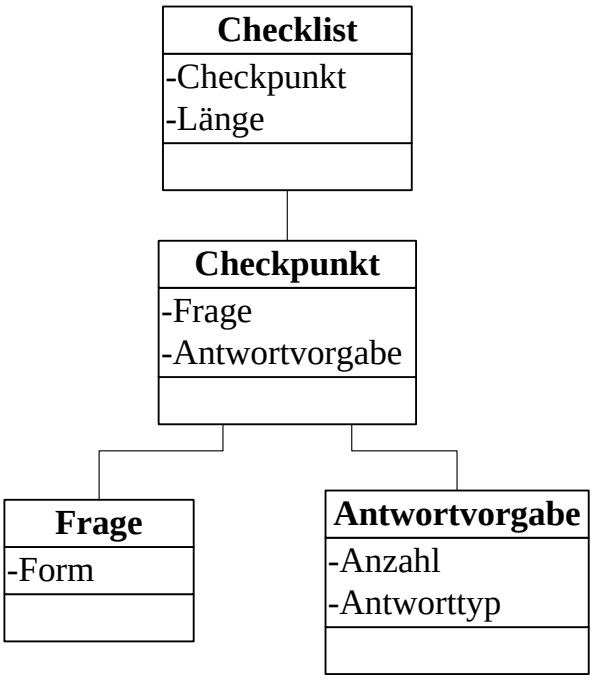


Abbildung 2: Allgemeine Beschreibung einer Checkliste mittels UML Domain-Modell

Die Anzahl der Checkpunkte und somit die Länge der Checkliste sind allgemein nicht fest vorgegeben. Es ist allerdings empfehlenswert, dass sie nicht länger als eine DIN A4 Seite ist. Das erleichtert den Prüfer, das zu prüfende Dokument und gleichzeitig alle Fragen im Auge zu behalten. Viele Checkpunkte verhindern die Konzentration auf die eigentlich wichtigen Probleme des Dokumentes und führen so häufig zu einer Entdeckung vieler unwichtiger Fehler. Anderseits können sehr kurze Checklisten mit zu wenig Checkpunkten die Prüfer nur auf sehr wenige Fehlermöglichkeiten aufmerksam machen. In Abbildung 3 werden die Beziehungen in einer Checkliste anhand des Domain-Modells gezeigt.

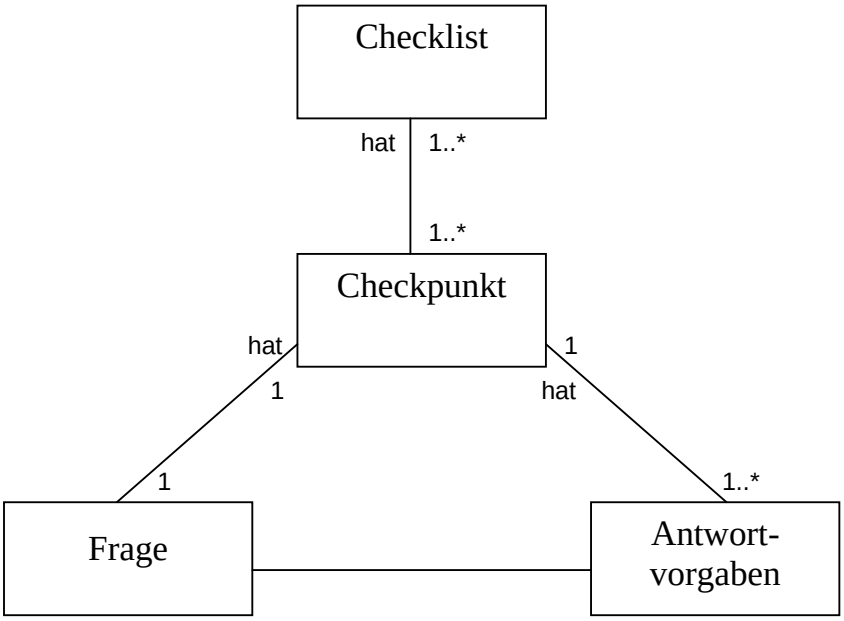


Abbildung 3: Checklist allgemein

4.1.1. Frageformen

Dabei werden die unterschiedliche Formen von Fragen beschrieben, was später bei der Erklärung verschiedener Arten von Checklisten hilfreich ist und auch als eine Grundlage für die Erstellung guter Checklisten nützlich sein kann. Die Fragen werden nach verschiedenen Kriterien unterteilt.

i.Nach der deutschen Grammatik :

i.1. Ergänzungsfragen – Zur Formulierung von Ergänzungsfragen dienen Fragewörter, z.B. *wer*, *wo* und *wann*. Sie stehen am Satzanfang. Die Antwort ist eine Ergänzung, die dem Fragewort entspricht.

- Wer, Welcher, Welche, Welches (Fragen nach dem Subjekt)
- Wo, Wohin, Woher (Fragen nach dem Ort)
- Wann (Fragen nach der Zeit)
- Wie (Fragen nach der Art und Weise)
- Weshalb, Warum, Weswegen, Wieso (Fragen nach der Ursache)

Ergänzung Frage in einer Checkliste kann die Frage „Wer ist der typischer Benutzer dieser Software?“. Diese Fragen verlangen eine Antwort in der Textform.

i.2. Entscheidungsfragen – Die Entscheidungsfrage (auch: Globalfrage, Ja/Nein Frage, Satzfrage) ist ein Typ von Fragesatz. Entscheidungsfragen sind die Fragen, auf die man nur mit „Ja“ oder mit „Nein“ antworten kann. Nur Entscheidungsfragen benötigen keine Fragewörter. Daher kann es vorkommen, dass sie einem Aussagesatz gleichen. In vielen Sprachen werden Entscheidungsfragen nur durch den Kontext oder durch Intonation als solche erkannt. Die meisten Fragen in den Checklisten sind Entscheidungsfragen.

ii. Nach Zielsetzung:
- ii.1. Wissensfragen
- ii.2. Überzeugungsfragen
- ii.3. (meist prospektive) Verhaltensfragen
- ii.4. Einstellungs-/Meinungsfragen:
- ii.5. Fragen nach Befragteneigenschaften

iii. Nach Art der Antwortvorgabe: Offen vs. Geschlossen – Um den Ausfüllenden zu motivieren kann man bei Fragebögen zwischen offenen und geschlossenen Fragen wechseln. Die Ergänzungsfragen sind offene Fragen und die Entscheidungsfragen geschlossene.

iv. Nach Formulierung: Direkt vs. Indirekt

v. Nach Funktion im Fragebogen (in der Checkliste):

v.1. Einleitungs- und Überleitungsfragen (Kontaktfragen) – Sollen kurz und einfach zu beantworten sein, um einen guten Einstieg zu ermöglichen. Eine solche Frage kann sein: „Sind die Anforderungen lesbar?“

v.2. Filterfragen – In Abhängigkeit von der gegebenen Antwort einer Filterfrage werden die weiteren Fragen entweder beantwortet oder einfach überspringt. Die Frage „Wird

die Initialisierung laut Produktbeschreibung vom Benutzer ausgeführt?“ ist eine Filterfrage. Wenn die Antwortet „ja“, dann bitte weiter mit Frage 17(zum Beispiel „Ist eine Installierungsanweisung vorhanden?“). Wenn nein, dann fahren Sie bitte mit Frage 20 fort.

v.3. Folgefragen – Folgefragen dienen dem Zweck, einzelne Aspekte aus vorhergehenden Antworten genauer zu erfassen. Zum Beispiel die Frage „Ist die Benutzerdokumentation im Verhältnis zur Produktbeschreibung wider-spruchsfrei?“ kann mit der Folgefrage „ Sind insbesondere alle Grenzwerte aus der Produktbeschreibung in der Benutzerdokumentation wiederholt?“ erweitert werden.

v.4. Motivationsfragen – In der Frage eingearbeitetes Lob kann motivierend und stimulierend wirken, z.B. „Befürworten Sie, in Ihrer Eigenschaft als versierte Fachkraft und gewiegte Kenner von EDV Systemen, eine solche Anschaffung?“

v.5. Vergangenheitsfrage – Frage, die gezielt die Erinnerung des Befragten belebt, um seine Auskunftsfähigkeit und Bereitschaft für nachfolgende Fragen zu erhöhen. Zum Beispiel die Frage „Sind die Anforderungen auch für einen Kunden verständlich beschrieben?“ motiviert den Inspektor noch einmal die Anforderungen zu lesen und so die nächste Frage „Gibt es Anforderungen, die mehr als eine Interpretation zulassen?“ leichter zu beantworten.

v.6. Kontrollfragen - Bestätigungsfragen Mittels dieser Fragenart wird ein Wissensstand kontrolliert. Man stellt zu einer Frage, für die man die Verlässlichkeit der Antwort ermitteln möchte, an einen anderen Platz im Fragebogen eine ähnliche Frage, und zwar so, dass der Befragte nach Möglichkeit nicht bemerkt, dass diese Frage so ähnlich schon einmal gestellt worden ist. Kontrollfragen dienen also der Kontrolle von Antwortmustern der Befragten. Die Frage „Ist eine sinnvolle HELP-Funktion vorhanden?“ kann an eine andere Stelle in der Checklist gestellt werden als „ Kann dem Benutzer durch die HELP-Funktion geholfen werden?“.

v.7. Innovationsfrage – Frage, die die Kreativität des Befragten anregt, um zu Neuerungen oder Verbesserungen zu gelangen. Es handelt sich um offene Fragen, die Formulierung ist meist schlicht, so z.B. „Wie könnte das Gerät verbessert werden?“. Gelegentlich werden solche Fragen auch als „Kritikfragen“ bezeichnet, wenn der der Innovation bedürftige Zustand kritisiert wird, z.B. „Wie kann dieser Fehler in Zukunft vermieden werden?“

v.8. Projektive Fragen – Fragen, die man verwendet, um vorhandene Antwortbarrieren bei Auskünften über die eigene Person oder persönliche Einstellungen zu bekämpfen. Durch geschickte Fragestellung kann erreicht werden, dass die Befragten über andere

Personen oder eine fiktive Person Auskunft geben. Aus den Antworten werden Rückschlüsse auf die befragte Person selbst gezogen. Beispielsweise könnte man in einer Checkliste, statt „Haben Sie Probleme, das Handbuch zu verstehen?“ die projektive Frage benutzen: „ Glauben Sie, dass der typische Benutzer Probleme haben wird, das Handbuch zu verstehen?“. Die Antwortbarriere in diesem Beispiel besteht im Geständnis, dass man Verständnisprobleme hat. Durch die indirekte Frage lässt sich diese Barriere umgehen, und die Beantwortung der Frage lässt bei geschickter Formulierung den Rückschluss im Sinne der ursprünglichen Frage zu.

vi. Andere Frageformen, die in Checklisten oder Fragebögen lieber nicht verwendet werden müssen:

vi.1. Fangfrage – Eine Fangfrage ist eine Frage an eine Person, die eine den Tatsachen entsprechende Antwort trickreich herbeiführt oder herbeiführen soll. Die Frage ist so gestellt, dass ein unaufmerksamer Antwortgeber sie falsch beantwortet oder sich selbst widerspricht. Dies wird erreicht, indem die Frage Antwortmöglichkeiten vorgibt oder impliziert, von denen keine zutrifft. Der Fragesteller kann eine solche Frage benutzen, um den Beantwortenden zu einer falschen Aussage zu verleiten. Fangfragen dürfen in seriösen Checklisten oder Fragebögen nicht verwendet werden.

vi.2. Rhetorische Fragen - Keine richtige Frage, da die rhetorisch fragende Person die Frage schon gleich selbst beantwortet oder die Frage so stellt, dass sie nur in ihrem Sinne beantwortet werden kann. Dies ist eine Form der Suggestivfrage und versucht den Befragten zu manipulieren.

vi.3. Suggestivfragen – Hier wird den Befragten eine Antworttendenz suggeriert, z.B. „Sind sie nicht auch der Meinung dass, ...“

4.1.2. Antwotvorgaben

Neben der Formulierung der Fragen sind natürlich die Antwortvorgaben entscheidend für die Qualität der Checkliste. Grundsätzlich unterscheidet man zwei Möglichkeiten: Offene Antwortvorgaben (manchmal auch als „offene Fragen“ bezeichnet) und geschlossene Antwortvorgaben (geschlossene Frage).

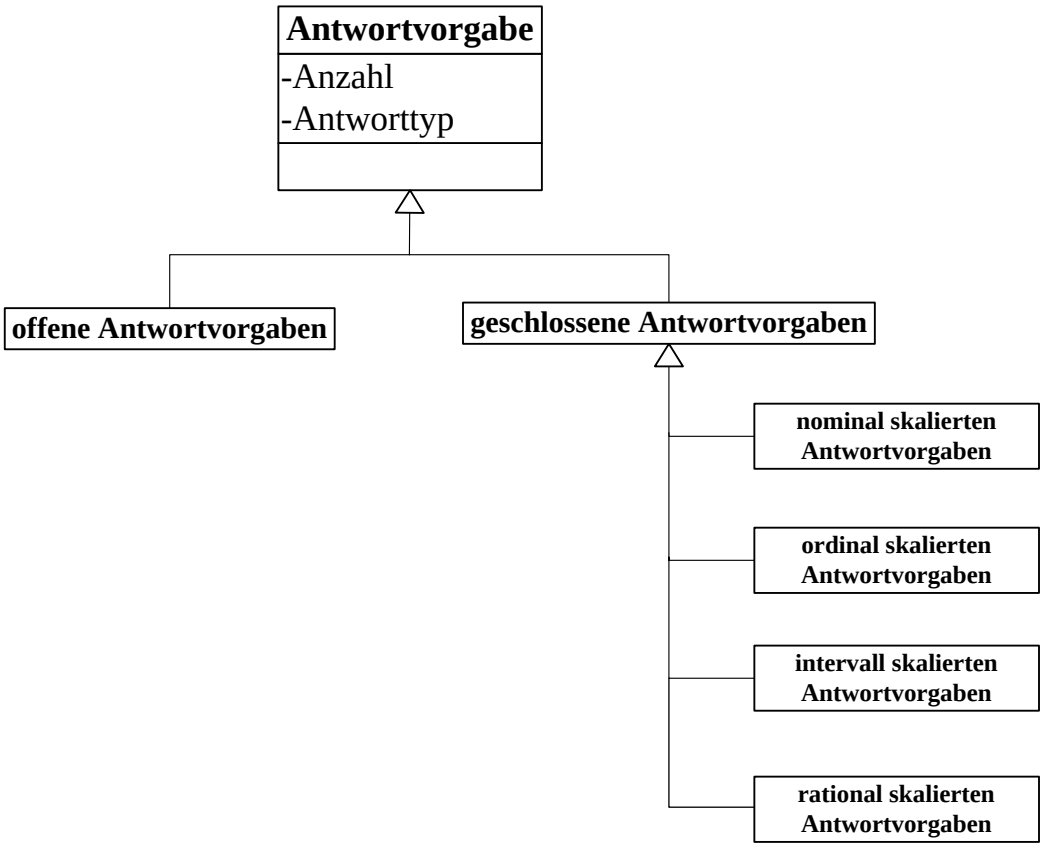


Abbildung 4 : Antwortvorgaben

i.Offene Antwortvorgaben

Sollen Meinungen mit eigenen Worten oder als Kommentar geäußert werden (z.B. wenn es um Veränderungs- oder Verbesserungsvorschläge geht), bieten sich die so genannten „offene“ Antwortvorgaben an. Hier wird zwar eine Frage gestellt, die Antwort jedoch nicht weiter vorgegeben. Sie ist nur durch den verfügbaren Platz begrenzt. Fragen, die nicht sofort eindeutig verständlich sind, sind problematisch und sollten nicht verwendet werden. Der Vorteil dieser Antwortvorgabe, die streng genommen gar keine Vorgabe ist, liegt darin, dass die Befragten mit eigenen Worten antworten und unter Umständen auf Dinge hinweisen, die vielleicht sonst entgehen würden. Nachteil dieser Methode sind der vergleichsweise hohe Auswertungsaufwand und die begrenzten statistischen Analysemöglichkeiten.

ii.Geschlossene Antwortvorgaben

Die zweite Variante für Antwortvorgaben sind die „geschlossenen“ Fragen, bei denen Antwortalternativen vorgegeben sind. Der Vorteil dieser Methode liegt in der guten statistischen Auswertbarkeit und ihrem relativ geringen Erfassungsaufwand. Nachteil ist hier jedoch die willkürliche Einschränkung der Meinungsäußerungen. Die Gefahr, den Prüfenden vor den Kopf zu stoßen oder ihn zu Antworten zu drängen, die nicht seine tatsächliche Meinung repräsentieren, zwingt zu besonderer Sorgfalt bei den Antwortvorgaben. Grundsätzlich gilt: Der Befragte muss mit den Antwortvorgaben eine sinnvolle Antwort auf die gestellte Frage geben können. Wenn bei der Entscheidung keine Abstufungen sinnvoll wären, reicht schon eine einfache Ja/Nein- Vorgabe aus.

Die geschlossenen Antwortvorgaben kann man nach der Art der Skalierung in vier Gruppen aufteilen:

ii.1. nominal skalierte Antwortvorgaben – Nominal skalierte Antwortvorgaben liegen vor, wenn die Zuordnung eines Wertes lediglich eine Benennung der Merkmalsausprägung darstellt. Sie dienen folglich nur zur Unterscheidung, und nicht zur Interpretation der Werte, beispielsweise die Zuordnung „ja“ für „richtig“ und „nein“ für „falsch“.

ii.2. ordinal skalierte Antwortvorgaben – Um eine Ordinalskala handelt es sich, wenn die Werte zusätzlich in eine natürliche Rangordnung gebracht werden können und größenmäßig sortierbar sind, ohne etwas über die Größe der Differenzen zwischen den Kategorien zu sagen. Ein Beispiel dafür ist: schlecht, mittel, gut.

ii.3. intervall skalierte Antwortvorgaben – Weiterhin gibt es noch die intervall skalierten Antwortvorgaben, die davon ausgehen, dass der Abstand zwischen den Skalenwerten sinnvoll interpretierbar ist. Der Nullpunkt und die Einheiten des Intervalls sind frei wählbar, allerdings müssen die Abstände der Intervalle gleich groß sein. Als Beispiel kann hier die Temperaturskala in Celsius angegeben werden.

ii.4. rational skalierte Antwortvorgaben – Im Gegensatz zu intervall skalierten Antwortvorgaben ist bei der Rationalskala ein natürlicher Nullpunkt fest vorgegeben und nur die Einheiten frei wählbar, z.B. beim Alter in Jahren. Ferner kann noch die Absolutskala als eine stärkere Verhältnisskala genannt werden, bei der sowohl der Nullpunkt als auch die Einheiten festgelegt werden.

4.2. Modell „Kästchen“

Das einfachste Modell einer Checkliste ist die Checkliste von Checkpunkten, die eine Entscheidungsfrage oder einfach nur eine Aussage darstellen. Als Antwortvorgabe ist nur ein Kästchen oder einfach leeren Platz vorgesehen. Die meisten Beispiele für Checklisten, die man in der Literatur finden kann, haben solchen Aufbau. Bryczynski hat 117 Checklisten analysiert, von denen nur 2 mit Antwortmöglichkeiten vorgesehen waren, d.h. alle anderen 115 waren von Typ Kästchen. [5]

In dem Fall mit einer Aussage wird der Checkpunkt einfach abgehackt, wenn die beschriebene Anweisung korrekt ausgeführt ist. Solche Checklisten können auch verwendet werden, um Information zu liefern über Schritte, die ausgeführt werden müssen, um eine gewisse Aufgabe zu erreichen, z.B. bei einer Review Sitzung. Es ist allerdings empfehlenswert bei Checklisten, die für die Prüfung und Aufdeckung von Fehlern in einem Dokument benutzt werden, Fragen anstelle von Aussagen zu benutzen. Die Fragen müssen dann so formuliert sein, dass sie mit „ja“ oder „nein“ beantworten werden können, z.B. „Sind alle Funktionen eindeutig referenzierbar?“. In diesem Fall zeigt eine Abhackung das Bejahen der Frage, also die „gute Antwort“. Sonst wird den Platz leer gelassen, was auf einen Fehler hinweist. Das ist aber ein Nachteil dieser Form der Checkliste, weil so nicht eindeutig klar wird, ob die Anweisung wirklich nicht ausreichend gut ausgeführt ist oder vielleicht der Checklistenprüfer sie sogar unabsichtlich vergessen hat.

Im Modell Kästchen sind die Fragen Entscheidungsfragen, die man nur mit „ja“ oder „nein“ beantworten kann. Es gibt nur eine geschlossene Antwortvorgabe, nämlich ein leeres Kästchen oder es wird bisschen Platz nach der Frage frei gelassen.

Checklist

Entscheidungsfrage ☐
Entscheidungsfrage ☐
.....
.....
.....
Entscheidungsfrage ☐

Abbildung 5: Modell „Kästchen“

Unten sind zwei Beispiele, die Teile von Checklisten Modell „Kästchen“ zeigen. Die vollständigen Checklisten sind im Anhang zu finden.

- General
  - Are the goals of the system defined?
  - Are the requirements clear and unambiguous?
  - Is a functional overview of the system provided?
  - Is an overview of the operational modes provided?

Beispiel 1 (aus [20])

Requisite 1. A corporate memory (historical database)

Evidence of Maturity

The organization has a process for organizing and retaining information on completed projects (a historical database).	❑
The historical database is treated as an integral part of the estimating process, and estimators have active roles in specifying and sustaining the information it contains.	❑
The database contains a useful set of completed projects.	❑

Beispiel 2 (aus [21])

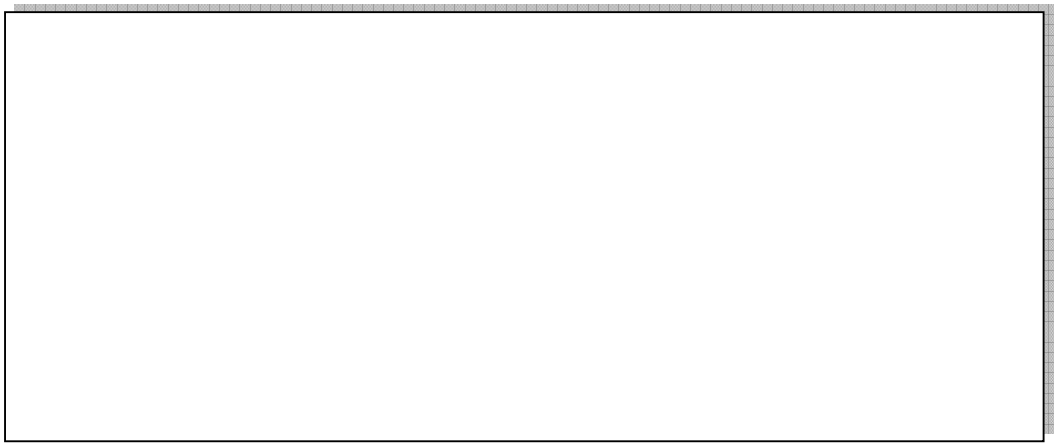
4.3. Modell „Ja/Nein“

Eine Erweiterung des Kästchenmodells stellt das Modell dar, bei dem eine Antwort von Typ Wahrheitswert verlangt wird. Der Checkpunkt besteht hier auch aus einer Entscheidungsfrage und den Antwortvorgaben „ja“ und „nein“. Die Antwortvorgaben sind nominal skaliert. Hier kann eindeutig die Antwort „nein“ gegeben werden. Somit werden die Fehler bei einer Bewertung der Checkliste direkt sichtbar. Fällt die Antwort negativ aus, dann sind auch alle Stellen im Prüfling anzugeben, die bezüglich der Fragenstellung mangelhaft sind. Bei einer guten Checkliste ist genug Platz für solche Bemerkungen vorgesehen. Als Beispiel 3 ist ein Fragment aus einer Checkliste von Model Ja/Nein gegeben.

Checklist

	JA	NEIN
Entscheidungsfrage	☐	☐
Entscheidungsfrage	☐	☐
.....		
…….		
……...		
Entscheidungsfrage	☐	☐

Abbildung 6: Modell „Ja/Nein“



Beispiel 3 (aus **[2])**

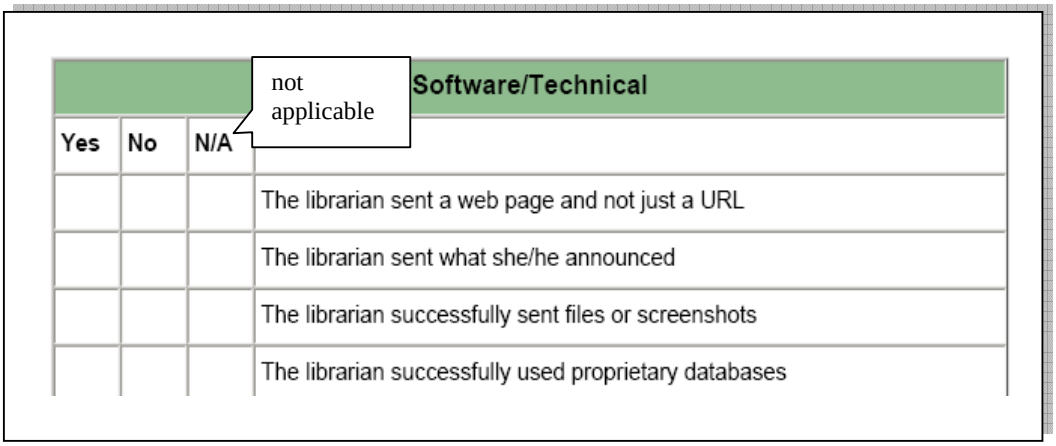
4.4. Erweitertes Modell „Ja/Nein“

Bei Modell „Ja/Nein“ können sehr leicht Verwirrungen für den Checklistenprüfer auftreten, wenn eine konkrete Frage nicht eindeutig beantwortet werden kann. Das passiert zum Beispiel bei einer Prüfung, wenn ein konkreter Checkpunkt sich weder bejahen noch verneinen lässt. (z.B. bei der Frage: „Ist die Benutzerdokumentation in allen Punkten übersichtlich strukturiert?“ in einem Projekt, das gar keine Benutzerdokumentation hat). Um die bessere Bewertung der Checkliste als vollständig ausgeführt zu werden, ist es dann hilfreich, noch eine weitere Auswahlmöglichkeit, wie zum Beispiel „nicht zutreffend“, zu haben. So vermeidet man Verwirrungen und man kann die gleiche Checkliste in verschiedene Projekte einsetzen.

In diesem Modell ist die Form der Fragen gleich derjenigen im Modell „Ja/Nein“. Hier hat man aber nicht zwei, sondern drei Antwortalternativen: „ja“, „nein“ und „nicht zutreffend“. Die letzte Antwort steht für den Fall, wenn die Frage so für das konkrete Projekt gar nicht gestellt werden kann. Beispiel 4 zeigt Teil einer Checklist von dem erweiterten Model Ja/Nein.

Checklist

	JA	NEIN	N.Z.
Entscheidungsfrage	☐	☐	☐
Entscheidungsfrage	☐		



			Software/Technical
Yes	No	N/A	
			The librarian sent a web page and not just a URL
			The librarian sent what she/he announced
			The librarian successfully sent files or screenshots
			The librarian successfully used proprietary databases

Beispiel 4 (aus [34])

4.5. Modell „Zahl“

Ein anderes Modell für Checklisten besteht dann, wenn der Checkpunkt Antwort von Typ Zahl zurückgibt. Wenn die Notwendigkeit von Schätzungen bei der Beantwortung der Fragen größer ist, was allerdings von der Formulierung und der Form der Frage abhängt, wird die Antwort durch das Ankreuzen in einer Skala dargestellt. Solche Antwortvorgaben werden Multiple-Choice-Antworten (manchmal auch Multiple-Choice-Fragen) genannt. Der Checklistenprüfer hat für die Beantwortung eine Auswahlmöglichkeit von Zahlen, meistens aus einem vorgegebenen Intervall (zum Beispiel von 1 bis 10). Er muss sich allerdings entscheiden und eine bestimmte Antwort ankreuzen. In diesem Modell muss auch gekennzeichnet werden, ob es nur ganze Zahlen oder auch rationale zugelassen sind. Weiterhin muss auch die Bedeutung der Zahlen eindeutig festgelegt werden. So müssen noch am Anfang der Checkliste zumindest für die Intervallgrenzen deutlich gemacht werden, was mit ihnen gemeint ist (z.B. bedeutet 1 sehr schlecht und 10 ausgezeichnet oder genau umgekehrt). Dann kann der Prüfer allein entscheiden, wie die anderen Möglichkeiten (2, 5, 6 und so weiter) zu interpretieren sind. In anderen Fällen sind für alle Zahlen auch die entsprechenden Erklärungen in der Checkliste gegeben.

Die Anzahl der Antwortmöglichkeiten ist durchaus willkürlich und könnte ebenso gut höher oder niedriger sein. Ebenso offen ist die Frage, ob es sich um eine ungerade Zahl von Antwortvorgaben handeln sollte bzw., ob eine gerade Anzahl von Antwortmöglichkeiten vorgegeben wird (hier müsste sich der Prüfer in jedem Fall in eine Richtung entscheiden, da es keine Mittelposition gibt).

Die Ankreuzmöglichkeiten müssen gewährleisten, dass die Prüfer die Antworten geben können, die ihre Meinung jeweils richtig widerspiegeln. Fünf, sieben oder neun Ankreuzmöglichkeiten erweisen sich in der Regel als hinreichend differenziert.

Die Mitte einer Antwort-Skala ist psychologisch ein besonders markanter Punkt. Manche Menschen möchten sich nicht gerne entscheiden und kreuzen am liebsten die Mitte an. Mitunter wird eine solche Antwortstrategie als „Antwort-Tendenz zur Mitte“ bezeichnet. Wenn zu viele von der Prüfenden von dieser Möglichkeit Gebrauch machen, kann dies zu Schwierigkeiten bei der Auswertung bzw. bei der Ergebnisinterpretation führen. Aus diesem Grund wird manchmal empfohlen, keine Mitte anzubieten, sondern eine gerade Anzahl von Antwortalternativen vorzugeben. Dies verhindert, dass Unentschlossene auf die Mitte der Skala ausweichen. Was ist aber, wenn die Frage tatsächlich weder mit schlecht noch mit gut zu beantworten ist? Ohne eine Mitte auf der Skala würden die Prüfer gezwungen, entweder in Richtung schlecht oder in Richtung gut zu beurteilen. Durch den Entscheidungszwang würde unnötigerweise Fehlervarianz erzeugt.

Empirisch ist es nicht eindeutig entschieden, ob eine gerade oder eine ungerade Anzahl von Antwortalternativen besser ist. Insgesamt scheint aber eine ungerade Anzahl von Ankreuzmöglichkeiten mehr Akzeptanz zu finden und den Umfang der Messfehler zu reduzieren.

Im Modell „Zahl“ sind die Fragen Multiple-Choice-Fragen. Hier ist die Anzahl der Antwortvorgaben frei wählbar. Die Antwort Typ ist eine Zahl, die zeigt, wie gut oder schlecht der Prüfer die konkrete Frage anschätzt.

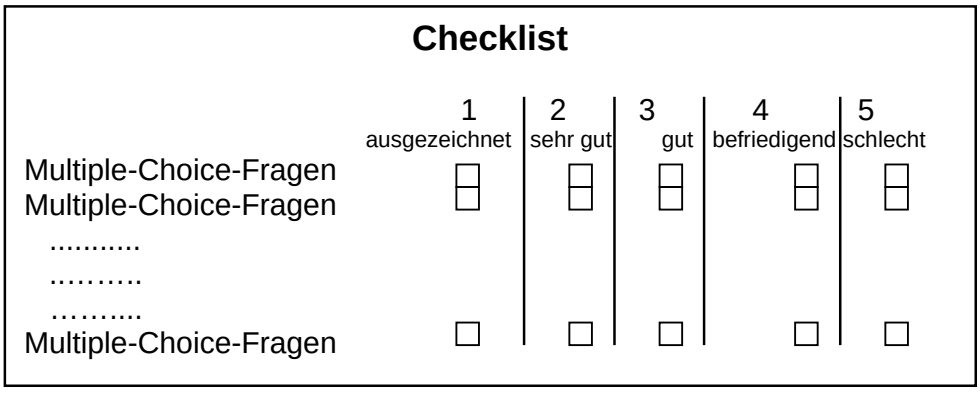
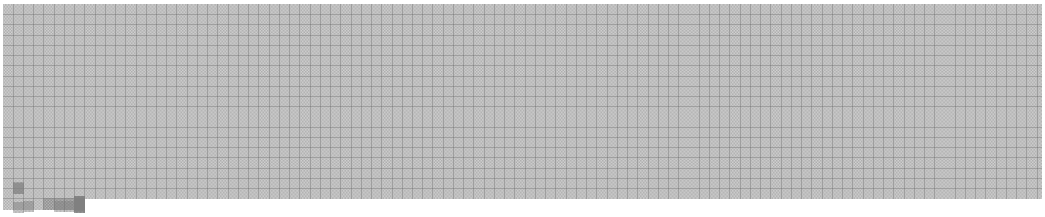


Abbildung 8: Modell „Zahl“

Beispiel 5 zeigt zwei Checkpunkte von Modell „Zahl“. In diesem Beispiel ist noch eine Antwortalternative für den Fall vorgegeben, wenn die konkrete Frage sich nicht beantworten lässt. So gewährleistet man eine Widerverwendbarkeit der Checkliste in verschiedenen Projekten.



4.7. Meta Checkliste

Eine Meta Checkliste besteht aus unterschiedlich aufgebauten Checkpunkten. Meistens sind das Checkpunkten von Modell „Text“ und Model „Ja/Nein“, wobei auch die anderen Formen benutzt werden können. Solche Checklisten würde man eher als Formulare bezeichnen. Sie werden als Prüfprotokolle z.B. bei einer Inspektion eingesetzt, um die Zusammenzufassung den gefundenen Fehler zu erleichtern. Beispiel 6 ist ein Prüfprotokoll.

Einzelprüfprotokoll

Projekt: _____

Prüfling: _____ **Gutachter:** _____
Datum: _____

Anzahl kritischer Befunde: _____ **Ich versichere: O Prüfling vollständig inspiziert**
Anzahl nicht-kritischer Befunde _____ **(ankreuzen) O Alle Befunde sind hier aufgelistet** _____

Nr.	**Befund: Beschreibung**	**Position** In Prüfling/ Referenz	**Checklistenref.**	**Kritikalität** Kritisch/wichtig/ marginal/gut

Beispiel 6

4.8. Modell von Chernak

In [6] ist ein Ansatz für die Erstellung von Checklisten vorgeschlagen. Er basiert auf einer Analyse von den in den frühen Projekten vorhandenen Fehlerdaten. Die entstehenden Prüflistenpunkte sollten zu den Projektdokumenten und der Art von Inspektionen passen. Das Ziel war eine Checkliste zu gestalten, die zum einen die häufigsten Fehler, die aus dem zu prüfenden Typ von Dokument reflektieren, ausfindig zu machen, zum anderen die Punkte zu finden, die sich später im Projekt als Fehler ergeben können und die noch in diesem Zeitpunkt wahrgenommen werden können.

Fehler sind Verstöße gegen bestimmte Regeln. Auf diese Art kann man aufgrund der statistischen Analyse von Fehlern die Regeln identifizieren, die am häufigsten gebrochen werden. Darauf basiert das Hauptkonzept des in [6] vorgeschlagenen Ansatzes.

Normalerweise werden Checklistenpunkte in der Form einer Frage gegeben: "Ist das Produktmerkmal gemäß der Regel?". Chernak sagt jedoch, dass ein Inspektionsprozess

produktiver werde und die Inspektoren mehr Fehler finden würden, wenn die Checkliste zwei Bestandteile hätte. Der erste Bestandteil ist eine "Wo man schauen kann " Kategorie und gibt einen Teil des Produktes. Er sollte den Inspektor zu einer bestimmten "Stelle" des Arbeitsprodukts führen, wo eine gegebene Regel durchgeführt ist. Der zweite Bestandteil ist eine "Wie man wahrnehmen kann" Kategorie. Er kann in der Form einer Frage gegeben werden, die dem Inspektor helfen sollte, einen Fehler wahrzunehmen.

Als Beispiel gibt Chernak im Falle von einer GUI-Spezifikation, dass der erste Bestandteil "Wo man schauen kann" sein kann: Menüoption, Editorkontrolle, Befehlknopf, usw. Im Falle von einer C++ Codeinspektion, kann er die Variable Deklaration sein. Der zweite Bestandteil "Wie man wahrnehmen kann" kann in der Form einer Frage gegeben werden. Er sollte dem Inspektor helfen, einen Fehler festzustellen. Wieder im Falle von ein GUI-Spezifikation können folgenden Fragen gestellt werden: „Ist die Menüoption erforderlich?“, „Ist die Editorkontrollbreite ausreichend?“, „Ist die Befehlsknopfwirkung klar?“ usw. Im Falle von einer C++ Codeinspektion ist eine mögliche Frage: „Ist die Variablentyp korrekt?“. Eine negative Antwort zeigt, dass einen potentiellen Fehler gefunden ist. Die Tabellen 1 und 2 zeigen, wie die Teile der fertigen Checkliste aussehen können. Als Beispiel 7 sind einige Checkpunkte aus Checkliste dieser Form gegeben.

Checkpunkt Nr.	**Wo man schauen muss**	**Wie man wahrnehmen kann**
1	Menü Optionen	Ist die Menü Option dienstbereit?
2		Ist ein an/zu Zustand nötig?
3		Ist die Menüoption erforderlich?

Tabelle 1: Beispiel für einen Teil von Checklist für GUI-Specification Produkt

Checkpunkt Nr.	**Wo man schauen muss**	**Wie man wahrnehmen kann**
1	Variablen Deklaration	Ist die Variablentyp korrekt?
2		Ist die Variablenname zulässig?
3		Ist die Variable einmalig?

Tabelle 2: Beispiel für einen Teil von Checklist für Source-Code

Diese Art von Checklisten hat eine andere Struktur als diejenigen Checklisten, die bis jetzt in dieser Arbeit beschrieben worden sind. Die Checkliste von Chernak besteht eigentlich aus zwei Checklisten. Der erste Teil ist eine Liste von potentiellen Problemstellen in dem zu prüfenden Dokument, wo eventuell Regeln gebrochen werden und Fehler auftreten können. Der zweite Teil ist eine Checkliste von Modell „Kästchen“, die dem Prüfer hilft, die Fehler zu erkennen.

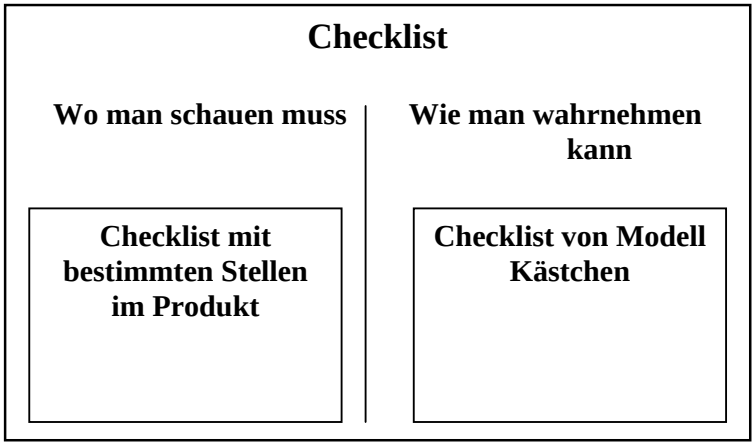


Abbildung 10: Modell von Chernak

In der Literatur findet man weitere Beispiele von Checklisten, die solchen Aufbau haben. Allerdings gibt es keine Studien, die die Vorteile von diesen Checklisten beweisen. Es ist aber offensichtlich, dass wenn die genaue Stelle im Produkt angegeben wird, wo die konkreten Fehler auftreten können, die Wahrscheinlichkeit sie zu bemerken, viel größer ist. Der Prüfer braucht insgesamt weniger Zeit für die Inspektion. Es ist nicht nötig, das ganze zu prüfende Dokument zuerst zu lesen und dann die ganze Checkliste auf einmal zu beantworten. Der Prüfer braucht sich nur auf den konkreten Teil zu konzentrieren.

Item no.	Where to look	How to detect
1	**All Design Diagrams**	Is each name unique?
2		Are all names used in the diagrams consistent?
3		Are all names used in the diagrams correct?
4	**Collaboration Diagrams**	Are the parameters and parameter types correct when compared to the “reads” and “changes” clause in the operation schemata?

Beispiel 7(aus [15])

4.9. Scoring Modell

Das Scoring Modell ist eine Erweiterung der Checkliste. Hier werden für alle Checkpunkte Punkte gegeben, die die Wichtigkeit der Frage widerspiegeln. Für die bedeutendsten

Checkpunkte werden entsprechend mehr Punkte gegeben. Die Summe aller Punkte gibt den Punktestand für die gesamte Checkliste. Anhand Formeln berechnet man die Punkte des gesamten Projekts. So kann man das Projekt mit anderen vergleichen und eventuell bei Nicht-Erreichen eines Minimums von Punkte das Projekt beenden.

Das Modell der Checkliste, die benutzt wird, ist nicht fest vorgegeben. Im [25] sind Checklisten gegeben, die von Modell „Ja/Nein“ sind. In diesem Fall werden die Punkte für den Checkpunkt gegeben, wenn die Antwort „Ja“ ist. Bei einer negativen Antwort wird nichts zu der gesamten Summe addiert. Teil einer Checkliste aus [25] ist im Beispiel 8 vorgestellt.

Nr.	FRAGE	Antwort			
		Gewicht	ja	nein	nicht relevant
1	**Allgemeiner Eindruck**				
1.1	Sauberer übersichtlicher Aufbau	2			
1.2	Gut lesbare und verständliche Kommentare	2			
1.3	Keine Tipp- bzw. Rechtschreibfehler	1			
1.4	Ausreichende Kommentierung	2			
	Summe	7			

Beispiel 8 (aus [25])

Im [7] ist das Scoring Modell mit Checkliste von Typ „Zahl“ beschrieben. In Abhängigkeit der Bedeutungen jeder einzelner Frage werden die Antwortvorgaben unterschiedlich skaliert. So benutzt man die Vorteile der Checkliste von Modell Zahl, weil die Fragen preziöser beantwortet und mehr Punkte für die wichtigsten Fragen gegeben werden.

5. Erstellung

Checklisten sind ein Grundteil von dem Inspektionsprozess. Sie helfen den Prüfern Fehler in dem entsprechenden Dokument zu finden. Jedes zu inspizierende Softwareprodukt muss allerdings anderen Qualitätsmerkmalen genügen. Daher ist der erste Schritt zur Erzeugung von Checklisten das zu inspizierende Softwareprodukt genau zu bestimmen [24].

Gilb und Graham[10] fordern, Checklisten so zu erstellen, dass sie direkt auf erstellte Regeln hinweisen, die ebenso wie die Checklisten selbst eine DIN A4-Seite nicht überschreiten sollen. Dies bedeutet aber folgerichtig, dass diese Checklisten keinen Mehrwert zu den eigentlichen Regeln darstellen, sondern allenfalls besonders häufig gebrochene Regeln hervorheben. Relativiert wird diese Forderung dadurch, dass sie empfehlen, den Inspektoren Rollen (z.B. "User", "Tester", "System", "Financial") zuzuweisen und jeder einzelnen Checkliste bis zu vier rollenspezifischen Fragen hinzuzufügen. Sie haben in ihrem Buch rollenspezifische Checklisten ermittelt, um dem Problem entgegenzuwirken: dass jeder Inspektor das gesamte Dokument mit den gleichen Fragen liest und somit theoretisch auch jeder Inspektor die gleichen Fehler entdeckt.

Es gibt allerdings keine allgemein gültige Richtlinie bezüglich der Erstellung der Checklisten. Meist beruht sie auf Erfahrungen vorangegangener Inspektionssitzungen oder auf Testergebnissen, also auf vorangegangenen Fehlerfindungsprozessen [6]. Da diese Daten in den meisten Unternehmen nicht vorhanden sind, werden Beispiele aus der Literatur herangezogen, die allerdings selten zu den unternehmensspezifischen Dokumenten passen und letztlich die Erfahrungen anderer Unternehmen darstellen. Solche Checklisten helfen daher nicht bei der Entdeckung von neuen Fehlern, die bisher nicht gefunden wurden, sondern nur beim Auffinden der typischerweise vorkommenden Fehler des jeweiligen Unternehmens. Ein weiteres Problem ist, dass Unternehmen solche Checklisten dauerhaft benutzen, so dass sie nicht flexibel an veränderte Prozesse und Fehlerquellen angepasst werden.

Zusätzlich gibt es noch generische Checklisten, die für unterschiedliche Dokumente erstellt werden. Diese sind allerdings eher zur Erstellung der Dokumente, als zur Überprüfung geeignet, da sie meist zu umfangreich sind. Viele Fragen verhindern die Konzentration auf die

eigentlich wichtigen Probleme des Dokumentes und führen so häufig zu einer Entdeckung vieler unwichtiger Fehler.

Nach Laitenberger [14] haben Prüflisten drei Grundschwächen, die bei der Erstellung neuer Checklisten berücksichtigt werden müssen:

1) Die Prüflistenfragen sind zuerst oft äußerst allgemein, z.B. "Sind die Anforderungen…?" Solche Fragen sind nicht nützlich, um den Inspektionsprozess zu unterstützen, da die Inspektoren nicht geleitet werden, wie man den angesprochenen Qualitätsfaktor überprüfen kann. In der Untersuchung von mehr als 100 Checklisten, die zu Anforderungsdokumenten typischerweise Fragen zu Konsistenz, Korrektheit und Vollständigkeit der Anforderungen behandeln, hat Bryczynski dasselbe Problem festgestellt. Auch hier waren die Checkpunkte in den untersuchten Checklisten zu allgemein [5].

2) Zweitens fehlt es an konkreter Führung, wie man die Prüfliste verwenden kann, d.h. wann man eine gewisse Frage beantworten und auf welcher Information basieren kann. Das Problem ist die mangelnde Arbeitsanleitung für die Prüfer. Ihre Aufgabe lautet meist:

    1) Lies die erste Frage der Checkliste
    2) Lies das gesamte Dokument und beantworte die Frage
    3) Lies die nächste Frage
    4) Gehe zu 2.)

    Oder alternativ:

    1) Lies das gesamte Dokument
    2) Beantworte alle Fragen der Checkliste.

    In jedem Fall wird häufig davon ausgegangen, dass die Inspektoren das gesamte Dokument lesen müssen. Dies führt meist zu Zeitproblemen, da die Dokumente häufig nicht in der für die Inspektion zur Verfügung stehenden Zeit gelesen werden können. Es wird daher empfohlen, statt oberflächlichen Lesens des gesamten Dokumentes nur wenige Seiten sehr konzentriert zu lesen, um die "tiefliegenden Fehler" aufzudecken. Allerdings gibt es keine Aussage darüber, wie die wenigen Seiten ausgewählt werden können.

3) Drittens sind die Prüflistenfragen oft nicht aktuell: Sie basieren auf den in der Vergangenheit wahrgenommenen Fehlern, wobei neue Fehlerklassen nicht in die Prüflisten einbezogen werden.

Checklisten müssen für die ausfüllende Person verständlich formuliert sein. Ziel ist es die Checkliste so zu gestalten, dass sich die betroffenen Personen sofort den Frageninhalten zuwenden können, ohne viel Energie in das Verstehen der Frage selbst verschwenden zu müssen. Die gute Checkliste ist also auf die Benutzergruppe zugeschnitten, passt sich an deren Wortschatz und Vorkenntnissen an.

Checklisten haben eine höhere Qualität, wenn sie den Inspektoren helfen Fehler zu finden. Jedoch könnte eine Inspektion Ziele außer Fehlerentdeckung haben, die bei der Erstellung auch berücksichtigen werden müssen. In [5] sind in Paar Beispile für Checkpunte gegeben, die anderen Ziele verfolgen.

1) Der Checkpunkt- "Ist irgendein Teil des Codes, ein möglicher Kandidat für Wiederverwendung?"- hat die Aufgebe mögliche Teile des Codes zu identifizieren, die wiederverwendet werden können .
2) Mit der Frage "Sind die Kommentare passend verwendet?" will man die Wartbarkeit des Codes verbessern.
3. "Ist es günstiger zu kaufen oder selber zu entwickeln?". Diese Checkpunkt wird verwendet, um sicherzustellen, dass korrekte Kostenentscheidungen gemacht worden sind.

Es müssen sehr viele Sachen bei der Erstellung einer Checkliste berücksichtigt werden. Ich habe eine Checkliste für die Erstellung von Checklisten zusammengestellt, die die wichtigsten Punkte zusammenzufassen versucht.

Checkliste für Checklisten

		Ja	Nein	NZ
	Allgemeine Fragen			
1	Ist die Checkliste nicht länger als eine DIN A4-Seite (20-25 Checkpunkte)?	☐	☐	☐
2	Passt die Checkliste genau zu dem zu inspizierenden Dokument?	☐	☐	☐
3	Ist die Checkliste aktuell?	☐	☐	☐

4	Sind die Fragen in Blöcken strukturiert?	☐	☐	☐
5	Ist genügend Raum zum Notieren von Bemerkungen gegeben?	☐	☐	☐
6	Falls das Ankreuzen der einzelnen Punkten der Checkliste vorgesehen ist: Gibt es auch die Antwortvorgabe „nicht zutreffend“?	☐	☐	☐
7	Gibt es eine Anweisung für die Ausführung der Checkliste?	☐	☐	☐
8	Falls Personendaten eingegeben werden, ist das erst am Ende der Liste eingegeben?	☐	☐	☐
	Fragen zu den einzelnen Checkpunkten			
9	Sind die Fragen eindeutig formuliert?	☐	☐	☐
10	Bei Multiple-Choice-Antwortvorgaben: Werden die wichtigsten Antwortmöglichkeiten abgedeckt?	☐	☐	☐
11	Sind, wo nötig, projektive Fragen gestellt?	☐	☐	☐
12	Weisen die Fragen direkt auf Fehler?	☐	☐	☐
13	Sind bei Antwortskalen die Skalenendpunkte (alle Skalenpunkte) bezeichnet?	☐	☐	☐
14	Gibt es Motivationsfragen?	☐	☐	☐
15	Werden bei den wichtigsten Punkte Kontrollfragen gestellt?	☐	☐	☐
16	Gibt es Innovationsfragen	☐	☐	☐

6. Einsatz

Vor allem werden Checklisten bei den Inspektionen als Lesetechnik hilfreich. Dafür können alle vorgestellten Modelle eingesetzt werden. Die Meisten Beispiele aus der Luteratur zeigen eine deutliche Tandenz zum ersten Model, also zum Modell „Kästchen“. Zudem sind die Checklisten von Modell „Ja/Nein“ in den Softwareinspektionen sehr verbreitet. Meiner Meinung nach sind die Chechecklisten, die im Modell von Chernack oder Scoring Modell erstellt sind, viel effektiver und müssen öfter eingesetzt werden. Ich könnte jeweils nur ein Beispiel für die beiden Modelle in der Literatur finden.

Auch für den Einsatz im Bereich Management, Dokumentation und Test findet man Checklisten in der Literatur. Die meisten dafür gefundenen Checklisten sind vorwiegend von Modell „Kästchen“ aber es gibt auch viele von Modell „Ja/Nein“. Der Grund für die größte Beliebtheit von Modell „Kästchen“ ist, dass Checklisten dieser Art besonders gut eingesetzt werden können, um Unterlagen auf Vollständigkeit zu prüfen. Deshalb bietet sich besonders der Bereich Dokumentation an, da dort der Aspekt Vollständigkeit eine große Rolle spielt.

Checklisten von Modell „Kästchen“ sind auch hilfreich beim Ausführen einer Aufgabe. In diesem Fall werden die benötigten Arbeitschritte als Checkpunkte in einer Checkliste zusammengefasst und abgehackt, wenn sie erledigt sind. Wenn alle Checkpunkte kontrolliert und ausgeführt sind, ist man mit der Aufgabe fertig. Solche Checklisten können sehr hilfreich im Bereich der Software- Herstellung sein. Wenn die Checkliste in kritischen Phasen beachtet wird, können bestimmte Fehler ausgeschlossen werden. Ein weiteres Einsatzgebiet von Checklisten in dieser Form ist das Risikomanagement. In[33] ist zum Beispiel eine Checkliste, bei der ein eventuelles Risiko angedeutet wird, wenn die Frage abgehackt wird. , deutet das eine eventuelle Risiko. Es ist noch eine Spalte in der Checkliste vorgesehen, wo der Projektmanager Vorschläge für mögliche Lösungen geben kann.

Software Requirements Phase	**RISK**	**ACTION**
Software Schedule: Is there an adequate software Schedule in place? Is it being followed? Are changes to schedule being tracked? Are changes to schedule made by due process, in a planned manner or are events changing the schedule with no decision of whether there is something wrong in the process or program that needs to change to make schedule? Has it been chosen to meet the needs of software development or is just a time/date when systems will need the software?		
Has all the slack/contingency time on the critical path been used up?		
Are software metrics kept and reported regularly? Monthly?		

Beispiel 9 (aus [33])

Checklisten von Modell „Text“ wurden beim Review – Sitzungen eingesetzt. Die Checkliste oder besser gesagt der Formular Prüfprotokoll dient zum Zusammenfassen der Ergebnisse eine Inspektion. Beispiele für Prüfprotokolle findet man in [18].

Im Bereich Kundenorientierung ließen sich Checklisten stärker einsetzen. Durch geeignete Checklisten könnte der Kunde stärker in den Entwicklungsprozess einbezogen werden. Ich bin der Meinung, dass hier Checklisten von Modell „Ja/Nein“ am geeigneten sind, aber wenn die Antwort präziser sein sollte, können auch Checklisten von Typ „Zahl“ eingesetzt werden.

Durch die unterschiedlichen Formen der Checklisten gibt es keine Einschränkungen in den Einsatzmöglichkeiten. Sie sind unüberschaubar groß, und werden sich in Zukunft noch weiter vergrößern.

Literaturverzeichnis

[1] Belli, Fevzi, Crigan, Radu: “Towards Automation of Checklist-Based Code-Reviews”, The Seventh International Symposium on Software Reliability Engineering (ISSRE '96), 1996.

[2] Bemer, R.W.: “A Checklist of Intelligence for Programming Systems'”, International Business Machines Corporation, New York

[3] Benninghaus, Hans: „Einführung in die sozialwissenschaftliche Datenanalyse“, München: Oldenbourg, 2005

[4] Berling, Tomas; Thelin, Thomas: “A Case Study of Reading Techniques in a Software Company”, International Symposium on Empirical Software Engineering pp. 229-238 (ISESE'04), 2004.

[5] Brykczynski, Bill: “A Survey of Software Inspection Checklists; in ACM SIGSOFT” in Software Engineering Notes, 24. Jg. (1999), 1, S.82-89.

[6] Chernak, Yuri: “A Statistical Approach to the Inspection Checklist Formal Synthesis and Improvement.” in IEEE Transactions on Software Engineering, Vol. 22, No.12, December 1996

[7] Cooper, Robert G.: “Winning at new products”, Perseus Books Group; Auflage: 3rd, August 2001

[8] Fagan, M.E.: “Design and code inspections to reduce errors in program development”, IBM Systems Journal, 15(3), 1976.

[9] Frühauf, Karol; Ludewig,Jochen; Sandmayr, Helmut : „Software-Prüfung: eine Anleitung zum Test und zur Inspektion“, 2000

[10] Gilb, Tom;Graham, Dorothy: „Software Inspection", 1993

[11] Hanbali, Imane: „Konzeption und Evaluation von Inspektions- Methoden zur Qualitätssicherung von Erfahrungen und Wissen aus dem Software Engineering", November 2003.

[12] Kan, Stephen H.: „Metrics and Models in Software Quality Engineering", 2003

[13] Laitenberger, Oliver: "Experimentelle Bewertung von Software-Lesetechniken in der industriellen Praxis" , April 1996

[14] Laitenberger, Oliver: „Cost-effective detection of software defects through perspective-based inspections", Stuttgart: Fraunhofer-IRB-Verl., 2000

[15] Laitenberger, Oliver; Atkinson,Colin; Schlich, Maud; El-Emam, Khaled: "An Experimental Comparison of Reading Techniques for Defect Detection in UML Design Documents", December 1999

[16] Lanubile, Filippo; Mallardo, Teresa; Calefato, Fabio; Christian Denger, Ciolkowski, Marcus: "Assessing the Impact of Active Guidance for Defect Detection: A Replicated Experiment", Proceedings of the Software Metrics, 10th International Symposium on (METRICS'04) - Volume 00 Pages: 269 - 279 , 2004

[17] Liggesmeyer,P; Spillner, A.: "Software-Qualitätssicherung in der Praxis - Ergebnisse einer Umfrage." , Informatik-Spektrum 17(6), 368-372 (1994)

[18] Lindermeier, Robert; Siebert, Frank: „Softwareprüfung und Qualitätssicherung: das Handbuch zur Prüfung von Softwareerzeugnissen nach DIN 66285 und ISO/IES 12119", 1995

[19] Löbel, G.; Schmid, H. und Müller, P.: „Lexikon der Datenverarbeitung", München: Verl. Moderne Industrie, 1969

[20] Miller, J., Wood, M., Roper M.: "Further Experiences with Scenarios and Checklists" in Empirical Software Engineering, 3, 37–64, 1998

[21] Park, Robert E.: "Checklists and Criteria for Evaluating the Cost and Schedule Estimating Capabilities of Software Organizations", Pittsburgh, Pa: Software Engineering Institute, Carnegie Mellon University, January 1995.

[22] Passing, Ursula: “A pilot experiment on software project size and effort estimation”, December 2002

[23] Porter Adam, Lawrence G. Votta and Victor R. Basili: “Comparing Detection Methods For Software Requirements Inspections: A Replicated Experiment”, IEEE Transactions on Software Engineering, Vol. 21, No. 6, June 1996.

[24] Schlich, Maud: „Inspektion des Systemlastenheftes“, Eine Publikation des Fraunhofer IESE, Juli 2002

[25] Schmied, Walter-Stefan; Winkler, Hermann: „Software-Qualität Ausgewählte Methoden und Werkzeuge der Softwareprüfung“, München 1983

[26] Schneider, Hans-Jochen: „Lexikon der Informatik und Datenverarbeitung“, Oldenbourg, 1991

[27] Simula, Magne Jørgensen: „A Preliminary Checklist for Software Cost Management”, Proceedings of the Third International Conference on Quality Software (QSIC’03), 2003

[28] Walker, AJ: “Quality Management applied to the Development of a National Checklist for IS0 9001 Audits for Software”, 3rd International Software Engineering Standards Symposium 1997.

[29] Würthele, Volker: “Checklisten für die Software-Bearbeitung“, Diplomarbeit 1995

[30] Zielke,Wolfgang: „Frag Dich vorwärts! : eine gute Frage ist die halbe Antwort“, München: Moderne Verl.-GmbH, 1978

[31] Zilahi-Szabó: „Kleines Lexikon der Informatik“, Oldenbourg Verlag, München 1995

Materialien aus dem Internet

[32] http://de.wikipedia.org/

[33] http://smad-ext.grc.nasa.gov/rmo/spa/SoftwareRiskChecklist.doc
Software Risk Checklist03.07.2006 Page 37 of 38 Form LeR-F0510.051

[34] http://www.qandanj.org/

[35] http://www.sozialnetz.de/

Anhang

Hier sind die vollständigen Checklisten, aus deren Teile in der Arbeit vorgestellt wurden.

Checklist for Validating Software Cost and Schedule Estimates

This checklist is designed to help managers assess the credibility of software cost and schedule estimates. It identifies seven issues to address and questions to ask when determining your willingness to accept and use a software estimate. Each question is associated with elements of evidence that, if present, support the credibility of the estimate.

Issue 1. Are the objectives of the estimate clear and correct?

Evidence of Credibility

- The objectives of the estimate are stated in writing. ❑
- The life cycle to which the estimate applies is clearly defined. ❑
- The tasks and activities included in (and excluded from) the estimate are clearly identified. ❑
- The tasks and activities included in the estimate are consistent with the objectives of the estimate. ❑

Issue 2. Has the task been appropriately sized?

Evidence of Credibility

- A structured process has been used to estimate and describe the size of the software product. ❑
- A structured process has been used to estimate and describe the extent of reuse. ❑
- The processes for estimating size and reuse are documented. ❑
- The descriptions of size and reuse identify what is included in (and excluded from) the size and reuse measures used. ❑
- The measures of reuse distinguish between code that will be modified and code that will be integrated as-is into the system. ❑
- The definitions, measures, and rules used to describe size and reuse are consistent with the requirements (and calibrations) of the models used to estimate cost and schedule. ❑

Software Engineering Institute special report, CMU/SEI-95-SR-004
This work has been sponsored by the U.S. Department of Defense

The size estimate was checked by relating it to measured sizes of other software products or components. ❑

The size estimating *process* was checked by testing its predictive capabilities against measured sizes of completed products. ❑

Issue 3. Are the estimated cost and schedule consistent with demonstrated accomplishments on other projects?

Evidence of Credibility

The organization has a structured process for relating estimates to actual costs and schedules of completed work. ❑

- The process is documented. ❑
- The process was followed. ❑

The cost and schedule models that were used have been calibrated to relevant historical data. ❑

(Models of some sort are needed to provide consistent rules for extrapolating from previous experience.)

The cost and schedule models quantify demonstrated organizational performance in ways that normalize for differences among software products and projects. ❑

(So that a simple, unnormalized, lines-of-code per staff-month extrapolation is *not* the basis for the estimate.)

The consistency achieved when fitting the cost and schedule models to historical data has been measured and reported. ❑

The values used for cost and schedule model parameters appear valid when compared to values that fit the models well to past projects. ❑

The calibration of cost and schedule models was done with the same versions of the models that were used to prepare the estimate. ❑

The methods used to account for reuse recognize that reuse is not free. ❑

(The estimate accounts for activities such as interface design, modification, integration, testing, and documentation that are associated with effective reuse.)

Extrapolations from past projects account for differences in application technology. ❑

(For example, data from projects that implemented traditional mainframe applications require adjustments if used as a basis for estimating client-server implementation. Some cost models provide capabilities for this, others do not.)

Extrapolations from past projects account for observed, long-term trends in software technology improvement. ❑

(Although some cost models attempt this internally, the best methods are usually based on extrapolating measured trends in calibrated organizational performance.)

Extrapolations from past projects account for the effects of introducing new software technology or processes. ❑

(Introducing a new technology or process can initially reduce an organization's productivity.)

Work-flow schematics have been used to evaluate how this project is similar to (and how it differs from) projects used to characterize the organization's past performance. ❑

Issue 4. Have the factors that affect the estimate been identified and explained?

Evidence of Credibility

A written summary of parameter values and their rationales accompanies the estimate. ❑

Assumptions have been identified and explained. ❑

A structured process such as a template or format has been used to ensure that key factors have not been overlooked. ❑

Uncertainties in parameter values have been identified and quantified. ❑

A risk analysis has been performed, and risks that affect cost or schedule have been identified and documented. ❑

(Elements addressed include issues such as probability of occurrence, effects on parameter values, cost impacts, schedule impacts, and interactions with other organizations.)

Issue 5. Have steps been taken to ensure the integrity of the estimating process?

Evidence of Credibility

Management reviewed and agreed to the values for all descriptive parameters *before* costs were estimated. ❑

Adjustments to parameter values to meet a desired cost or schedule have been documented. ❑

If a dictated schedule has been imposed, the estimate is accompanied by an estimate of ❑

- The normal schedule. ❑
- The additional expenditures required to meet the dictated schedule. ❑

Adjustments to parameter values to meet a desired cost or schedule are accompanied by management action that makes the values realistic. ❑

More than one cost model or estimating approach has been used, and the differences in results have been analyzed and explained. ❑

People from related but different projects or disciplines were involved in preparing the estimate. ❑

At least one member of the estimating team is an experienced estimator, trained in the cost models that were used. ❑

Estimators independent of the performing organization concur with the reasonableness of the parameter values and estimating methodology. ❑

The groups that will be doing the work accept the estimate as an achievable target. ❑

Memorandums of agreement have been completed and signed with the other organizations whose contributions affect cost or schedule. ❑

Issue 6. Is the organization's historical evidence capable of supporting a reliable estimate?

Evidence of Credibility

The estimating organization has a method for organizing and retaining information on completed projects (a historical database). ❑

The database contains a useful set of completed projects. ❑

Elements included in (and excluded from) the effort, cost, schedule, size, and reuse measures in the database are clearly identified. ❑

(See, for example, the SEI checklists for defining effort, schedule, and size measures.)

Schedule milestones (start and finish dates) are described in terms of criteria for initiation or completion, so that work accomplished between milestones is clearly bounded. ❑

Records for completed projects indicate whether or not unpaid overtime was used. ❑

Unpaid overtime, if used, has been quantified, so that recorded data provide a valid basis for estimating future effort. ❑

Cost models that were used for estimating have been used also to provide consistent frameworks for recording historical data. ❑

(This helps ensure that comparable terms and parameters are used across all projects, and that recorded data are suitable for use in the estimating models.)

The data in the historical database have been examined to identify inconsistencies, and anomalies have been corrected or explained. ❑

(This is best done with the same cost models that are used for estimating.)

The organization has a structured process for capturing effort and cost data from ongoing projects. ❑

The producing organization holds postmortems at the completion of its projects. ❑

- To ensure that recorded data are valid.
- To ensure that events that affected costs or schedules get recorded and described while they are still fresh in people's minds.

Information on completed projects includes ❑

- The life-cycle model used, together with the portion covered by the recorded cost and schedule.

  Actual (measured) size, cost, and schedule. ❑

  The actual staffing profile. ❑

  An estimate at completion, together with the values for cost model parameters that map the estimate to the actual cost and schedule. ❑
- A work breakdown structure or alternative description of the tasks included in the recorded cost. ❑

  A work-flow schematic that illustrates the software process used. ❑
- Nonlabor costs. ❑

Management costs. ❑

- A summary or list of significant deliverables (software and documentation) produced by the project. ❑
- A summary of any unusual issues that affected cost or schedule. ❑

Evolution in the organization's work-flow schematics shows steady improvement in the understanding and measurement of its software processes. ❑

Issue 7. Has the situation changed since the estimate was prepared?

Evidence of Credibility

The estimate has not been invalidated by recent events, changing requirements, or management action (or inaction). ❑

The estimate is being used as the basis for assigning resources, deploying schedules, and making commitments. ❑

The estimate is the current baseline for project tracking and oversight. ❑

Reference Session Evaluation Checklist

Software/Technical			
Yes	**No**	**N/A**	
			The librarian sent a web page and not just a URL
			The librarian sent what she/he announced
			The librarian successfully sent files or screenshots
			The librarian successfully used proprietary databases
Research/Search Skills			
Yes	**No**	**N/A**	
			The resource or item sent answered or partly answered the question
			The librarian sufficiently evaluated the item before sending it
			The librarian checked if the resource (e.g., a web site's search facility, links, etc.) was functional before sending it
			Appropriate resources were used in answering this question
			A Resolution Code was properly assigned
Communication and Model Reference Behavior			
Yes	**No**	**N/A**	
			The librarian used open-ended probing questions to elicit the customer's specific question
			The librarian maintained a steady dialog with the customer
			The librarian kept the customer informed of her/his activities

			The librarian communicated what next steps were expected of the customer
			The librarian made appropriate use of scripted messages
			The librarian informed the customer before sending an item
			The librarian explained to the customer what she/he sent
			The librarian provided instruction in the resource sent, if needed
			The librarian explained where to find the answer in the item sent
			The librarian appropriately referred the call
			The librarian appropriately took responsibility for getting back to the customer.
			The librarian asked the customer if she/he has completely answered the question
			It was clear from the customer's point of view that the librarian was ending the session

ISONORM 9241/10

Beurteilung von Software auf Grundlage der Internationalen Ergonomie-Norm ISO 9241/10

Jochen Prümper & Michael Anft

Büro für ARBEITS- *und* ORGANISATIONSPSYCHOLOGIE *GMBH*
Alt-Lichtenrade 112 • D-12309 Berlin
Tel 030 – 80 10 80 80 • Fax 030 – 80 10 80 8-20
email: info@bao.de
www.bao.de

Einleitung

Im folgenden geht es um die Beurteilung von Softwaresystemen auf Grundlage der Internationalen Norm ISO 9241/10.

Bitte beachten Sie:

→ Das Ziel dieser Beurteilung ist es, Schwachstellen bei Softwaresystemen aufzudecken und konkrete Verbesserungsvorschläge zu entwickeln.

→ Um dies zu bewerkstelligen, ist Ihr Urteil als Kenner des Softwaresystems von entscheidender Bedeutung! Grundlage Ihrer Bewertung sind Ihre individuellen Erfahrungen mit dem Software-Programm, das Sie beurteilen möchten.

→ Dabei geht es nicht um eine Beurteilung Ihrer Person, sondern um Ihre persönliche Bewertung der Software mit der Sie arbeiten.

Bitte machen Sie im folgenden Kasten zunächst einige Angaben zu der Software, auf die sich Ihre Beurteilung im folgenden beziehen wird.

Auf welches Software-Programm bezieht sich Ihre Beurteilung? (Beurteilen Sie bitte lediglich ein Software-Programm!)	
Name der Software	
Versionsnummer	
Hersteller	
gegebenenfalls Teilanwendung / Modul	
Seit wie vielen Monaten arbeiten Sie schon mit der von Ihnen beurteilten Software?	Monate
Wie viele Stunden arbeiten Sie pro Woche durchschnittlich mit der von Ihnen beurteilten Software?	Stunden
Wie gut beherrschen Sie die von Ihnen beurteilte Software? (**- - -** = sehr schlecht; +++ = sehr gut)	--- ❍ -- ❍ - ❍ -/+ ❍ + ❍ ++ ❍ +++ ❍

	Einleitung	

Noch ein Hinweis zur Beantwortung des Beurteilungsbogens:

Im folgenden Fragebogen werden die Anforderungen der Norm über Beschreibungen konkretisiert. Diese Beschreibungen weisen immer folgende Form auf:

Beispiel 1:

Die Software ...	- - -	- -	-	-/+	+	++	+++	*Die Software ...*
ist schlecht.	❍	❍	❍	❍	⊗	❍	❍	ist gut.

Im ersten Beispiel wird danach gefragt, wie gut bzw. wie schlecht die Software ist. Die Benutzerin oder der Benutzer beurteilt in diesem Fall die Software zwar als gut, sieht jedoch noch Verbesserungsmöglichkeiten.

Beispiel 2:

Die Software ...	- - -	- -	-	-/+	+	++	+++	*Die Software ...*
ist langsam.	❍	⊗	❍	❍	❍	❍	❍	ist schnell.

Im zweiten Beispiel beurteilt die Benutzerin oder der Benutzer die Software als ziemlich langsam.

Am Ende jeder Seite werden Sie gefragt, wie wichtig Sie das jeweilige Norm-Kriterium für Ihre Tätigkeit finden.

In einem weiteren Feld auf jeder Seite können Sie ein Beispiel angeben, welches die Verletzung des jeweiligen Norm-Kriteriums konkretisiert.

→ Am besten bearbeiten Sie den Beurteilungsbogen, während Sie das zu bewertende Softwaresystem vor sich am Bildschirm haben. Dadurch haben Sie die Möglichkeit, bei der Beantwortung der einzelnen Fragen die ein oder andere Sache noch einmal zu überprüfen.

→ Füllen Sie bitte den Beurteilungsbogen äußerst sorgfältig aus und lassen Sie keine der Fragen aus!

	Aufgabenangemessenheit	

Unterstützt die Software die Erledigung Ihrer Arbeitsaufgaben, ohne Sie unnötig zu belasten?

Die Software ...	- - -	- -	-	-/+	+	++	+++	*Die Software ...*
ist kompliziert zu bedienen.	❍	❍	❍	❍	❍	❍	❍	ist unkompliziert zu bedienen.
bietet nicht alle Funktionen, um die anfallenden Aufgaben effizient zu bewältigen.	❍	❍	❍	❍	❍	❍	❍	bietet alle Funktionen, um die anfallenden Aufgaben effizient zu bewältigen.
bietet schlechte Möglichkeiten, sich häufig wiederholende Bearbeitungsvorgänge zu automatisieren.	❍	❍	❍	❍	❍	❍	❍	bietet gute Möglichkeiten, sich häufig wiederholende Bearbeitungsvorgänge zu automatisieren.
erfordert überflüssige Eingaben.	❍	❍	❍	❍	❍	❍	❍	erfordert keine überflüssigen Eingaben.
ist schlecht auf die Anforderungen der Arbeit zugeschnitten.	❍	❍	❍	❍	❍	❍	❍	ist gut auf die Anforderungen der Arbeit zugeschnitten.

Die *Aufgabenangemessenheit* der Software ist für meine Tätigkeit ...

sehr unwichtig	❍	❍	❍	❍	❍	❍	❍	sehr wichtig

Bitte beschreiben Sie ein Beispiel, welches die Verletzung der oben genannten Aspekte gegebenenfalls besonders deutlich veranschaulicht:

_____

_____

_____

_____

_____

_____

_____

	Selbstbeschreibungsfähigkeit	

Gibt Ihnen die Software genügend Erläuterungen und ist sie in ausreichendem Maße verständlich?

Die Software ...	- - -	- -	-	-/+	+	++	+++	*Die Software ...*
bietet einen schlechten Überblick über ihr Funktionsangebot.	❍	❍	❍	❍	❍	❍	❍	bietet einen guten Überblick über ihr Funktionsangebot.
verwendet schlecht verständliche Begriffe, Bezeichnungen, Abkürzungen oder Symbole in Masken und Menüs.	❍	❍	❍	❍	❍	❍	❍	verwendet gut verständliche Begriffe, Bezeichnungen, Abkürzungen oder Symbole in Masken und Menüs.
liefert in unzureichendem Maße Informationen darüber, welche Eingaben zulässig oder nötig sind.	❍	❍	❍	❍	❍	❍	❍	liefert in zureichendem Maße Informationen darüber, welche Eingaben zulässig oder nötig sind.
bietet auf Verlangen keine situationsspezifischen Erklärungen, die konkret weiterhelfen.	❍	❍	❍	❍	❍	❍	❍	bietet auf Verlangen situationsspezifische Erklärungen, die konkret weiterhelfen.
bietet von sich aus keine situationsspezifischen Erklärungen, die konkret weiterhelfen.	❍	❍	❍	❍	❍	❍	❍	bietet von sich aus situationsspezifische Erklärungen, die konkret weiterhelfen.

Die *Selbstbeschreibungsfähigkeit* der Software ist für meine Tätigkeit ...

sehr unwichtig	❍	❍	❍	❍	❍	❍	❍	sehr wichtig

Bitte beschreiben Sie ein Beispiel, welches die Verletzung der oben genannten Aspekte gegebenenfalls besonders deutlich veranschaulicht:

	Steuerbarkeit	

Können Sie die Art und Weise, wie Sie mit der Software arbeiten, beeinflussen?

Die Software ...	- - -	- -	-	-/+	+	++	+++	*Die Software ...*
bietet keine Möglichkeit, die Arbeit an jedem Punkt zu unterbrechen und dort später ohne Verluste wieder weiterzumachen.	❍	❍	❍	❍	❍	❍	❍	bietet die Möglichkeit, die Arbeit an jedem Punkt zu unterbrechen und dort später ohne Verluste wieder weiterzumachen.
erzwingt eine unnötig starre Einhaltung von Bearbeitungsschritten.	❍	❍	❍	❍	❍	❍	❍	erzwingt keine unnötig starre Einhaltung von Bearbeitungsschritten.
ermöglicht keinen leichten Wechsel zwischen einzelnen Menüs oder Masken.	❍	❍	❍	❍	❍	❍	❍	ermöglicht einen leichten Wechsel zwischen einzelnen Menüs oder Masken.
ist so gestaltet, dass der/die Benutzer/in nicht beeinflussen kann, wie und welche Informationen am Bildschirm dargeboten werden.	❍	❍	❍	❍	❍	❍	❍	ist so gestaltet, dass der/die Benutzer/in beeinflussen kann, wie und welche Informationen am Bildschirm dargeboten werden.
erzwingt unnötige Unterbrechungen der Arbeit.	❍	❍	❍	❍	❍	❍	❍	erzwingt keine unnötigen Unterbrechungen der Arbeit.

Die *Steuerbarkeit* der Software ist für meine Tätigkeit ...

sehr unwichtig	❍	❍	❍	❍	❍	❍	❍	sehr wichtig

Bitte beschreiben Sie ein Beispiel, welches die Verletzung der oben genannten Aspekte gegebenenfalls besonders deutlich veranschaulicht:

	Erwartungskonformität	

Kommt die Software durch eine einheitliche und verständliche Gestaltung Ihren Erwartungen und Gewohnheiten entgegen?

Die Software ...	- - -	- -	-	-/+	+	++	+++	*Die Software ...*
erschwert die Orientierung durch eine uneinheitliche Gestaltung.	❍	❍	❍	❍	❍	❍	❍	erleichtert die Orientierung durch eine einheitliche Gestaltung.
lässt einen im Unklaren darüber, ob eine Eingabe erfolgreich war oder nicht.	❍	❍	❍	❍	❍	❍	❍	lässt einen nicht im Unklaren darüber, ob eine Eingabe erfolgreich war oder nicht.
informiert in unzureichendem Maße über das, was es gerade macht.	❍	❍	❍	❍	❍	❍	❍	informiert in ausreichendem Maße über das, was es gerade macht.
reagiert mit schwer vorhersehbaren Bearbeitungszeiten.	❍	❍	❍	❍	❍	❍	❍	reagiert mit gut vorhersehbaren Bearbeitungszeiten.
lässt sich nicht durchgehend nach einem einheitlichen Prinzip bedienen.	❍	❍	❍	❍	❍	❍	❍	lässt sich durchgehend nach einem einheitlichen Prinzip bedienen.

Die *Erwartungskonformität* der Software ist für meine Tätigkeit ...

sehr unwichtig	❍	❍	❍	❍	❍	❍	❍	sehr wichtig

Bitte beschreiben Sie ein Beispiel, welches die Verletzung der oben genannten Aspekte gegebenenfalls besonders deutlich veranschaulicht:

_____

_____

_____

_____

_____

_____

_____

	Fehlertoleranz	

Bietet Ihnen die Software die Möglichkeit, trotz fehlerhafter Eingaben das beabsichtigte Arbeitsergebnis ohne oder mit geringem Korrekturaufwand zu erreichen?

Die Software ...	- - -	- -	-	-/+	+	++	+++	*Die Software ...*
ist so gestaltet, dass kleine Fehler schwerwiegende Folgen haben können.	❍	❍	❍	❍	❍	❍	❍	ist so gestaltet, dass kleine Fehler keine schwerwiegenden Folgen haben können.
informiert zu spät über fehlerhafte Eingaben.	❍	❍	❍	❍	❍	❍	❍	informiert sofort über fehlerhafte Eingaben.
liefert schlecht verständliche Fehlermeldungen.	❍	❍	❍	❍	❍	❍	❍	liefert gut verständliche Fehlermeldungen.
erfordert bei Fehlern im großen und ganzen einen hohen Korrekturaufwand.	❍	❍	❍	❍	❍	❍	❍	erfordert bei Fehlern im großen und ganzen einen geringen Korrekturaufwand.
gibt keine konkreten Hinweise zur Fehlerbehebung.	❍	❍	❍	❍	❍	❍	❍	gibt konkrete Hinweise zur Fehlerbehebung.

Die *Fehlertoleranz* der Software ist für meine Tätigkeit ...

sehr unwichtig	❍	❍	❍	❍	❍	❍	❍	sehr wichtig

Bitte beschreiben Sie ein Beispiel, welches die Verletzung der oben genannten Aspekte gegebenenfalls besonders deutlich veranschaulicht:

_____

_____

_____

_____

_____

_____

_____

	Individualisierbarkeit	

Können Sie als Benutzer oder Benutzerin die Software ohne großen Aufwand auf Ihre individuellen Bedürfnisse und Anforderungen anpassen?

Die Software ...	- - -	- -	-	-/+	+	++	+++	*Die Software ...*
lässt sich von mir schwer erweitern, wenn für mich neue Aufgaben entstehen.	❍	❍	❍	❍	❍	❍	❍	lässt sich von mir leicht erweitern, wenn für mich neue Aufgaben entstehen.
lässt sich von mir schlecht an meine persönliche, individuelle Art der Arbeitserledigung anpassen.	❍	❍	❍	❍	❍	❍	❍	lässt sich von mir gut an meine persönliche, individuelle Art der Arbeitserledigung anpassen.
eignet sich für Anfänger und Experten nicht gleichermaßen, weil ich es nur schwer an meinen Kenntnisstand anpassen kann.	❍	❍	❍	❍	❍	❍	❍	eignet sich für Anfänger und Experten gleichermaßen, weil ich es leicht an meinen Kenntnisstand anpassen kann.
lässt sich - im Rahmen ihres Leistungsumfangs - von mir schlecht für unterschiedliche Aufgaben passend einrichten.	❍	❍	❍	❍	❍	❍	❍	lässt sich – im Rahmen ihres Leistungsumfangs - von mir gut für unterschiedliche Aufgaben passend einrichten.
ist so gestaltet, dass ich die Bildschirmdarstellung schlecht an meine individuellen Bedürfnisse anpassen kann.	❍	❍	❍	❍	❍	❍	❍	ist so gestaltet, dass ich die Bildschirmdarstellung gut an meine individuellen Bedürfnisse anpassen kann.

Die *Individualisierbarkeit* der Software ist für meine Tätigkeit ...

sehr unwichtig	❍	❍	❍	❍	❍	❍	❍	sehr wichtig

Bitte beschreiben Sie ein Beispiel, welches die Verletzung der oben genannten Aspekte gegebenenfalls besonders deutlich veranschaulicht:

	Lernförderlichkeit	

Ist die Software so gestaltet, dass Sie sich gut darin einarbeiten konnten und bietet sie auch dann Unterstützung, wenn Sie neue Funktionen lernen möchten?

Die Software ...	- - -	- -	-	-/+	+	++	+++	*Die Software ...*
erfordert viel Zeit zum Erlernen.	❍	❍	❍	❍	❍	❍	❍	erfordert wenig Zeit zum Erlernen.
ermutigt nicht dazu, auch neue Funktionen auszuprobieren.	❍	❍	❍	❍	❍	❍	❍	ermutigt dazu, auch neue Funktionen auszuprobieren.
erfordert, dass man sich viele Details merken muss.	❍	❍	❍	❍	❍	❍	❍	erfordert nicht, dass man sich viele Details merken muss.
ist so gestaltet, dass sich einmal Gelerntes schlecht einprägt.	❍	❍	❍	❍	❍	❍	❍	ist so gestaltet, dass sich einmal Gelerntes gut einprägt.
ist schlecht ohne fremde Hilfe oder Handbuch erlernbar.	❍	❍	❍	❍	❍	❍	❍	ist gut ohne fremde Hilfe oder Handbuch erlernbar.

Die *Lernförderlichkeit* der Software ist für meine Tätigkeit ...

sehr unwichtig	❍	❍	❍	❍	❍	❍	❍	sehr wichtig

Bitte beschreiben Sie ein Beispiel, welches die Verletzung der oben genannten Aspekte gegebenenfalls besonders deutlich veranschaulicht:

_____
_____
_____
_____
_____
_____
_____

	Allgemeine Angaben	

Zum Schluss bitten wir Sie, noch folgende Fragen zu beantworten:

Wie ist Ihre vertragliche wöchentliche Arbeitszeit?	Stunden
Wie viele Überstunden leisten Sie durchschnittlich pro Woche?	Stunden
Wie lange arbeiten Sie schon in diesem Unternehmen?	Jahre
Wie lange arbeiten Sie bereits am Bildschirm?	Jahre
Wie viele Stunden pro Tag arbeiten Sie durchschnittlich am Bildschirm?	Stunden pro Tag
Ihr Alter? (bitte ankreuzen)	❍ bis 20 Jahre ❍ 21 – 30 Jahre ❍ 31 – 40 Jahre ❍ 41 – 50 Jahre ❍ 51 – 60 Jahre ❍ über 60 Jahre
Ihr Geschlecht? (bitte ankreuzen)	❍ weiblich ❍ männlich
In welcher Abteilung arbeiten Sie?	
Raum für Anmerkungen	

MODEL REVIEW CHECKLIST

According to QAP-3.3, Revision 4

1. MODEL DOC. No.: MOD-	2. MODEL DOC. REVISION:	3. MODEL DOC. DATE:
4. MODEL DOC. TITLE:		
5. PI:	6. AUTHOR(S):	
7. TASK: ORD-		
8. REVIEWER NAME:	9. REVIEWER ORGANIZATION:	10. REVIEW DUE DATE:

Legend: "**X**" or checkmark (or abbreviations: **S, U, N/A**) designates:
SAT = Satisfactory; Nonmandatory comments, when used, are documented "**SAT**"
UNSAT = Unsatisfactory: a mandatory comment representing a violation of criteria, requiring Author's response
N/A = Not Applicable
Comment = Comments; Identification of problem (page no. or section/para. no., etc.); noncompliance details; optional explanations, etc.
Response = PI or Author adds response to comments.

Criteria 1-13 are for the technical reviewer(s):

No.	Criteria	SAT	UNSAT	N/A	Page/Sec/para.	Comment	Response
1.	Does the model documentation include definition of objective (intended use)? 4.1.2(h)(1)						
2.	Does the model doc. include description of conceptual model and scientific basis as well as alternatives for the selected conceptual model and rationale for not selecting alternatives? 4.1.2(h)(2)						
3.	Does the model doc. include results of literature searches and other applicable background data? 4.1.2(h)(3)						
4.	Does the model doc. include identification of inputs and their sources? 4.1.2(h)(4)						

Project Development Phase:	**RISK Yes/No /Partial**	**ACTION Accept/ Work**
System Requirements Phase		
Are system level requirements documented? To what level? Are they clear, unambiguous, verifiable ?		
Is there a project wide method for dealing with future requirements changes?		
Have software requirements been clearly delineated/allocated?		
Have these system level software requirements been reviewed, inspected with systems, hardware and the users to insure clarity and completeness?		
Has Firmware and Software been differentiated, who is in charge of what and is there good coordination if H/W is doing "F/W"?		
Are the effects on command latency and its ramifications on controllability known?		
Can the Bus bandwidth support projected data packet transfers?		
Are requirements defined for loss of power? System reaction known or planned for? UPS (Uninteruptable Power Supplies) planned for critical components?		
Is an impact analysis conducted for all changes to baseline requirements		

Software Planning Phase	**RISK**	**ACTION**
Is there clarity of desired end product? Customer & builders (system and software) agree on what is to be built and what software's role is?		
Are system level requirements on Software documented ? and are they complete/sufficient and clearly understood		
Are all interface requirements known & understood?		
Roles and responsibilities for system & software clearly defined and followed and sufficient		
Have the end user/operator requirements been represented in the conception phase such that their requirements are flowed into the software requirements?		
Has all needed equipment, including spares been laid out ? and ordered? Is there sufficient lead time to get needed equipment? Is there a contingency plan for not getting all equipment ? Is there a contingency plan for not getting all equipment when needed?		
Is the needed level of technical expertise known?		
Level of expertise available: within NASA? from contractors? Will expertise be available as the schedule demands? Is there more than one person with a particular expertise/knowledge?		
Training: Enough trained personnel? Time to train all personnel? on project ? on equipment/ software development environment, etc.? Time and resources to train additional personnel as needed ?		
Budget: Is budget sufficient for: equipment needed personnel training travel etc.		

Schedule Is Schedule reasonable considering needed personnel, training and equipment? Does system level schedule accommodate software lifecycle? Can needed equipment be made available in time?		
Has all the slack/contingency time on the critical path been used up?		
Are software metrics kept and reported regularly? Monthly?		
Are deviations to the development plan being tracked? Trended? Are the trends reported in a manner to allow timely and appropriate software and project management decisions?		
Will new development techniques be used?		
Will new or different development environment be used?		
Is this a new technology?		
Will simulators need to be designed and built? Is there time and resources allocated for this?		
Is there a schedule for development of ground and flight software? Is it reasonable, does it match reality? Is it being followed? Are changes tracked and the reasons for the changes well understood?		
Do the schedules for ground and flight software match up with what is needed for test and delivery?		
Will test software need to be designed and developed? Is there time and resources allocated for this?		
Distributed development environment: Will this be a distributed development (different groups or individuals working on parts of the project in different locations e.g. out of state)? Are there proper facilities and management structure to support distributed development?		
Inter/Intra group Management Are interfaces with other developers, suppliers, users, management, customer understood and documented? Is there a known way to resolve differences between these groups (i.e. conflict resolution/ who has ultimate authority, who is willing to make a decision)?		
Management Planning: Is management experienced at managing this size and/or type of team (Experienced Project Manager?)? Is management familiar with the technology being used (e.g. OOA/OOD and C++)? Is there a well constructed software management plan that outlines procedures, deliverables, risk, lifecycle, budget, etc. Is it reasonable, does it match reality?		