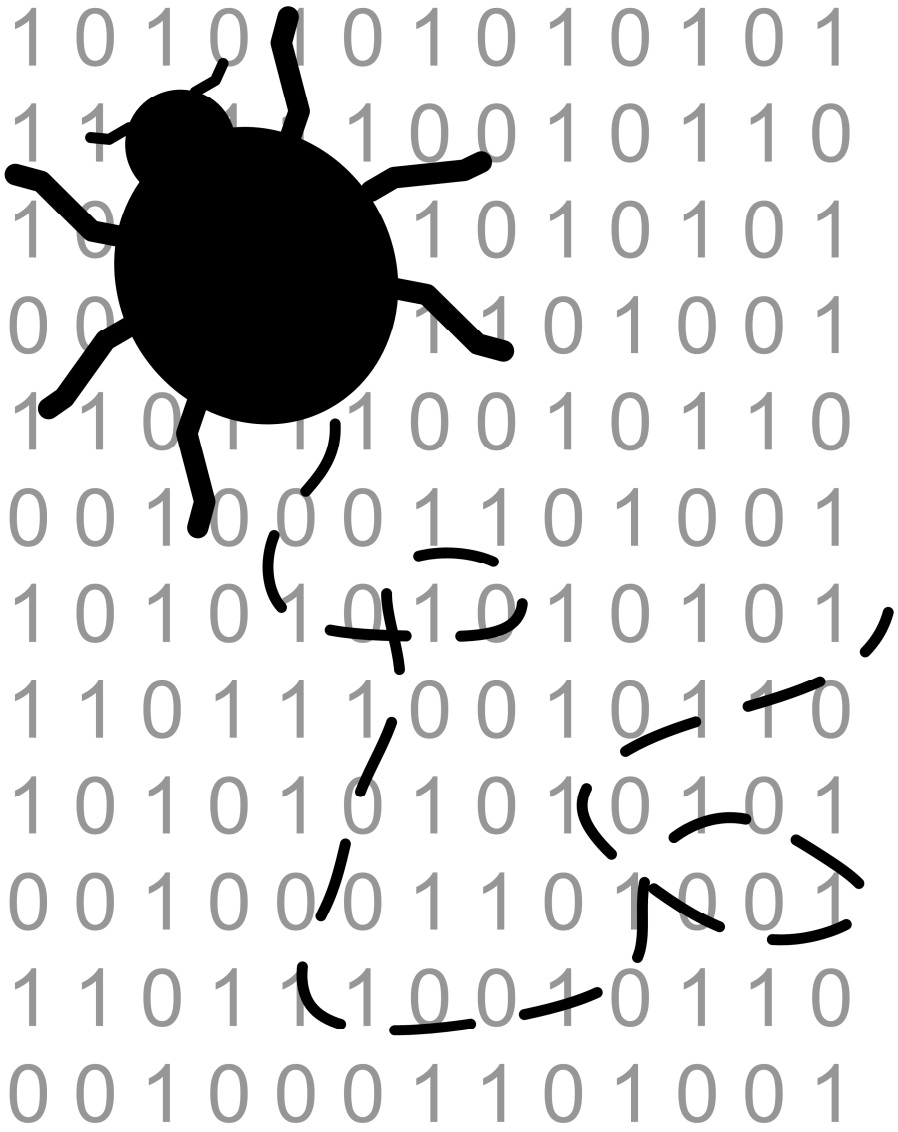
Lastenheft
„Bug Tracking System (BTS)“



Auftraggeber:

Prof. Dr. Hagen Voss

TH Georg Agricola

Inhaltsverzeichnis

1 EINFÜHRUNG IN DAS PROJEKT

Die Firma ***Casual Games Everywhere GmbH*** (**CGE** GmbH) mit Sitz in Bochum möchte ihr Angebot an online spielbaren Casual Games erweitern. Die CGE GmbH tritt zum einen als Partner von Internet-Diensteanbietern auf und stellt diesen sein Sortiment an Casual Games gegen Gebühr als ein Marketing-Instrument zur Verfügung und zum anderen vertreibt CGE seine Online-Spiele über verschiedene Spiele-Portale auch in eigener Regie.

Casual Games erlauben es Internet-Diensteanbietern einerseits, die Verweildauer von Webseiten-Besuchern zu erhöhen und somit potentiell mehr Werbeeinnahmen zu generieren und andererseits eine stärkere Kundenbindung zu erreichen.

Im Rahmen verschiedener Entwicklungsprojekte für Casual Games hat sich gezeigt, dass aufgrund der vielen gemeldeten Bugs (Fehler) ein datenbankgestütztes Bug Tracking System erforderlich ist.

Ein Bug Tracking System (BTS) ist ein Werkzeug in der Softwareentwicklung, das dazu dient, Produkt- / Programmfehler zu erfassen und zu dokumentieren.

Ein kurzes Beispiel soll an dieser Stelle den Workflow beim Auftauchen eines Bugs veranschaulichen. Ein Anwender (Spieler, Gamer) melde der Entwicklungsfirma des Spiels einen Bug, der während des Spielens aufgetaucht sei (s. **Abb. 1**).

Abb. 1: Bug im Spiel **The Witcher**: Schwebendes Pferd.

Ein Mitglied des Software-Testing-Teams postet den gemeldeten Bug firmenintern weiter. Eine kurze Analyse der vom Spieler eingereichten Screenshots führt dazu, dass der gemeldete Bug als gültiger (echter) Bug mit dem **Status new** angelegt wird.

Aufgrund einer weitergehenden Untersuchung wird der Schweregrad des Bugs mit **Critical** (kritisch) und dessen Priorität mit **Urgent** (dringend) deklariert, d. h. dass er bereits im nächsten Build behoben sein muss.

Darüber hinaus wird der Bug mit dem entsprechenden Produkt (hier: The Witcher) assoziiert und mit passenden Stichworten verschlagwortet (sog. Tagging).

Anschließend wird die Behebung des Bugs einem Mitglied des Entwicklerteams zugewiesen, wodurch der Bug den neuen Status **assigned** erhält. Die während des Fehlerbehebungsprozesses geführte Kommunikation zwischen Entwicklern und Testern wird fortlaufend dokumentiert.

Nachdem der zuständige Entwickler eine Lösung zur Behebung des Bugs erarbeitet hat, stuft er den Bug aus seiner Sicht als **completed / fixed** ein.

Daraufhin prüft ein Mitglied des Testteams die erarbeitete Lösung und kontrolliert, ob der Bug tatsächlich vollständig behoben werden konnte. Sollte dies der Fall, erhält der Bug den Status **closed / verified**, andernfalls wird er dem zuständigen Entwickler zu weiteren Bearbeitung wieder zugewiesen, wodurch der Bug auf den Status ***reassigned*** gesetzt wird.

Die Eingrenzung bzw. Lokalisierung eines Fehlers geschieht meistens durch eine Folge von Fragen und Antworten zwischen Anwendern, Entwicklern und Testern. **Deren wechselseitige Kommunikation** zu **dokumentieren**, ist eine **der zentralen Aufgaben** eines **Bug Tracking Systems**.

Nach einer längeren vorausgegangenen Entscheidungsfindungsphase hat sich die Geschäftsführung von CGE dafür entschieden, das Bug Tracking System in eigener Regie zu realisieren.

2 AUFGABENSTELLUNG & ANFORDERUNGEN

Ziel des vorgeschlagenen Projekles **Bug Tracking System** (**BTS**) ist, ein datenbank-gestütztes Bug Tracking System zu realisieren, das über ein Shell-Client bedient werden kann.

Ein Bug Tracking System (BTS) ist ein Werkzeug in der Softwareentwicklung, das dazu dient, Produkt- / Programmfehler zu erfassen und zu dokumentieren.

Die Eingrenzung bzw. Lokalisierung eines Fehlers geschieht meistens durch eine Folge von Fragen und Antworten zwischen Anwendern und Entwicklern. Diese wechselseitige Kommunikation zwischen Anwendern und Entwicklern zu dokumentieren, ist eine der zentralen Aufgaben eines Bug Tracking Systems.

Nachfolgend sind einige typische Fragestellungen aufgelistet, die mit Hilfe eines Bug Tracking Systems beantwortet werden sollen:

- Welche Probleme sind aufgetreten?
- Welcher Art sind die Probleme?
- Wie wurde das Problem dokumentiert?
- Wer hat das Problem gemeldet?
- Welcher Entwickler ist für das Problem zuständig?
- Wer hat das Problem als solches verifiziert?
- Welches Produkt ist von diesem Problem betroffen?
- Welche Produktversionen sind davon betroffen?
- Was wurde bereits zur Problembehebung unternommen?
- Wieviel Aufwand wird die Lösung des Problems voraussichtlich erfordern?
- Wie hoch war der tatsächliche Aufwand?
- Anzeigen der vollständigen Bug-Historie für einen Bug

Aufgrund der geforderten vielfältigen Analysemöglichkeiten werden Bug Tracking Systeme in der Regel als datenbankgestützte Systeme ausgelegt.

Typische Bug Tracking Systeme unterstützen das Konzept des Lebenzyklus eines Bugs (bug life cycle), der über den Status eines Bugs verfolgt werden kann.

Notwendige Voraussetzung dafür ist natürlich, dass der Status eines Bugs innerhalb eines BTS gemäß eines vorgegebenen Schemas von berechtigten Personen modifiziert werden kann.

Aufgrund einer ersten fimeninternen Analyse zum Datenmodell des BTS ergaben sich die folgenden Anforderungen hinsichtlich der Basisentitäten (Relationen / Tabellen):

Nr	Klasse	Attribute (vorläufig)
1	Bug	bug_id {PK} , date_reported , summary , description, resolution , priority , hours
2	BugHistory	bug_history_id {PK}, all bug attributes , change_by, date_changed, change_action
3	BugPriority	priority_id {PK} , priority_name
4	BugStatus	status_id {PK} , status_name
5	BugSeverity	severity_id {PK} , severity_name, description
6	Tag	tag_id {PK} , tag
7	Comment	comment_id {PK} , author , comment_date , comment
8	Account	account_id {PK} , account_name , first_name , last_name, email, password_hash , portrait_image , hourly_rate , account_type , account_role
9	Product	product_id {PK} , product_name, description
10	Screenshot	image_id {FK} , screenshot_image , caption
11	AccountType	type_id {PK} , type_name
12	AccountRole	role_id {PK}, role_name

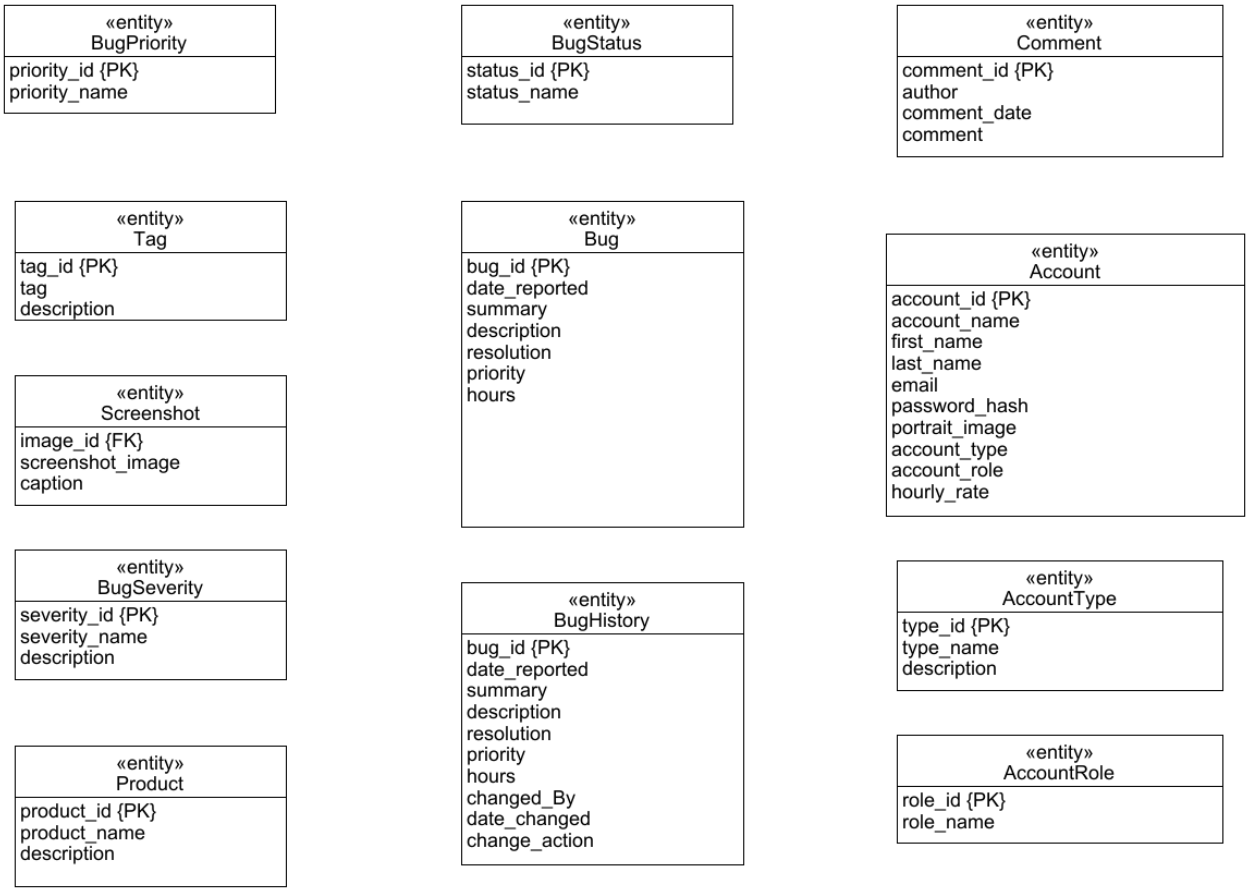


Abb. 2: Vorläufiges fachliches Datenmodell (noch ohne Assoziationen) zum Bug Tracking System

Darüber hinaus existieren zwischen den Klassen folgende Assoziationen

- Jedem Bug wird genau ein BugStatus zugewiesen. Zu jedem BugStatus kann es mehrere Bugs geben.
- Jedem Bug wird genau eine BugPriority zugewiesen. Zu jeder BugPriority kann es mehrere Bugs geben.
- Jedem Bug wird genau eine BugSeverity zugewiesen. Zu jeder BugSeverity kann es mehrere Bugs geben.
- Jeder Bug kann mit einer beliebigen Anzahl von Tags versehen werden und umgekehrt, d. h. jedem Tag kann sich auf eine beliebige Anzahl von Bugs beziehen.
- Jeder Bug kann mit beliebig vielen Comments verknüpft werden, wohingegen jeder Comment zu genau einem Bug gehören muss.
- Ein Comment kann eine beliebige Anzahl von anderen Comments referenzieren. Zwischen Comments existiert daher eine reflexive (0..n, 0…n) – Assoziation.
- Jeder Bug kann sich auf eine beliebige Anzahl von Products beziehen und umgekehrt, d. h. jedes Product kann eine beliebige Anzahl von Bugs enthalten.
- Zu Dokumentationszwecken können einem Bug mehrere Screenshots zugewiesen werden. Jeder Screenshot hingegen muss zu genau einem Bug gehören.
- Ein Product kann von mehreren Accounts (Benutzern) betreut werden. Dies gilt auch umgekehrt, d. h. ein Account kann mehrere Products verwalten.
- Ein Account (Benutzer) gehört zu genau einem AccountType. Ein AccountType kann beliebig viele Accounts umfassen.
- Für jeden Bug wird eine Bug-Historie (BugHistory) geführt, in der alle Änderungen eines Bugs protokolliert werden. Daher kann es für jeden Bug beliebig viele Einträge in der BugHistory geben, wobei umgekehrt jeder Eintrag in der BugHistory genau einem Bug zugeordnet sein muss.

- Jeder Eintrag in der BugHistory muss zu genau einem Benutzer (Account) gehören, wobei umgekehrt jeder Benutzer beliebig viele Einträge in der BugHistory vornehmen kann.
- Zwischen Bugs und Accounts existieren drei unterschiedliche Assoziationen, nämlich **reported_by**, **assigned_to** und **verified_by**. Diese geben an,
    - wer einen Bug gemeldet hat,
    - wem ein Bug zu Bearbeitung zugewiesen wurde,
    - wer einen Bug überprüft / verifiziert hat.
- Hierbei gelten die folgenden Anforderungen an die drei Beziehungstypen:
    - **Reported_by**: Jeder Benutzer (Accounts) kann mehrere Bugs melden, wobei ein Bug mit genau einem berichtenden Benutzer (Account) verknüpft ist.
    - **Assigned_to**: Jedem Benutzer (Accounts) können mehrere Bugs zu weiteren Bearbeitung zugewiesen sein. Ein Bug muss aber von genau einem Benutzer (Accounts) bearbeitet werden.
    - **Verified_by**: Jeder Benutzer (Accounts) kann mehrere Bugs verifizieren, also als behoben klassifiziert werden, wobei ein bestimmter Bug nur von genau einem Benutzer (Account) verifiziert worden sein muss.

Die zu den Bugs abgegebenen Kommentare (= Fragen und / oder Antworten) sollen in ihrer kompletten Historie verfolgt werden können. Dies bedeutet, dass in der Regel ganze Kommentar-Threads nachzuverfolgen sind (Beispiel s. a. **Abb. 3**).

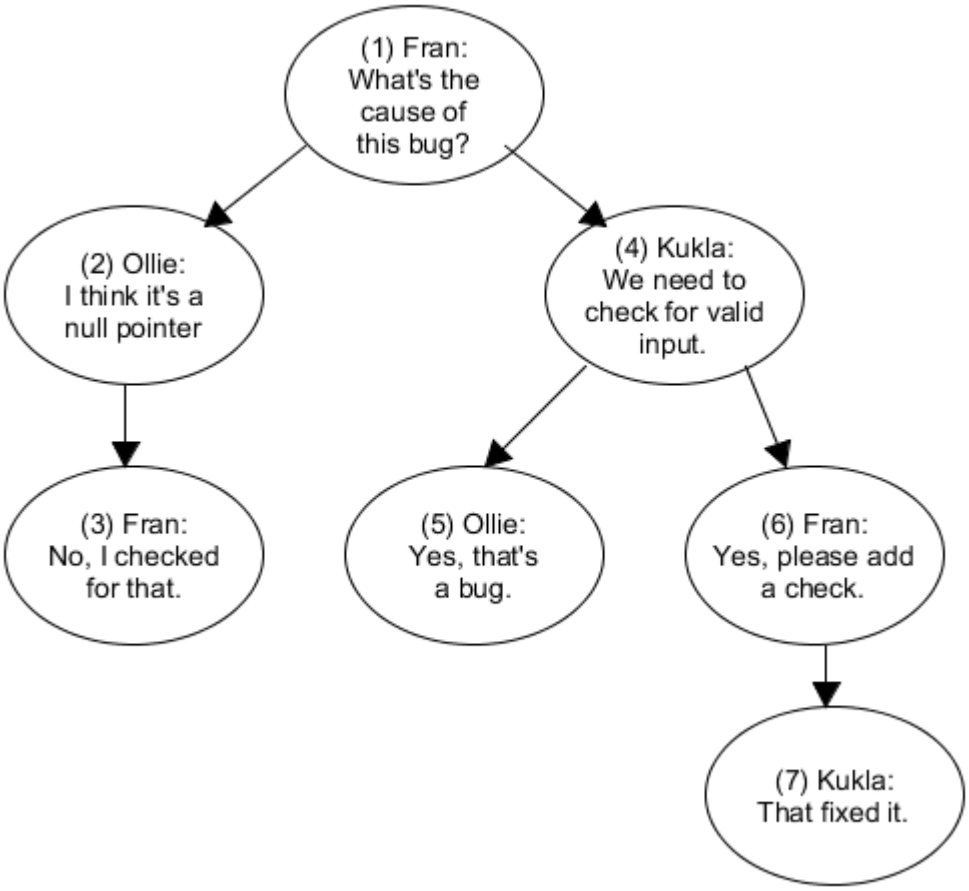


Abb. 3: Beispiel für threaded Kommentare.

Zu den einzelnen Produkten (Products) kann es durchaus mehrere Ansprechpartner (Accounts) geben und vice versa. Dies bedeutet, dass ein Ansprechpartner (= Account) für verschiedene Produkte zuständig sein kann, und zwar in verschiedenen Rollen.

2.1 Der Lebenszyklus eines Bugs (Bug Life Cycle)

Innerhalb des Software Testing nutzt man das Paradigma des Bug Life Cycles, d. h. man unterstellt, dass ein Bug einen eigenen Lebenslauf besitzt.

Daher werden die innerhalb eines Bug Tracking Systems zu bearbeitenden Bugs durch verschiedene Zustände charakterisierbar.

Das folgende Zustandsübergangsdiagramm (State Chart) zeigt die verschiedenen Zustände eines Bugs innerhalb des Bug Tracking Systems und deren Verknüpfungen bzw. Übergänge (Transitionen) untereinander.

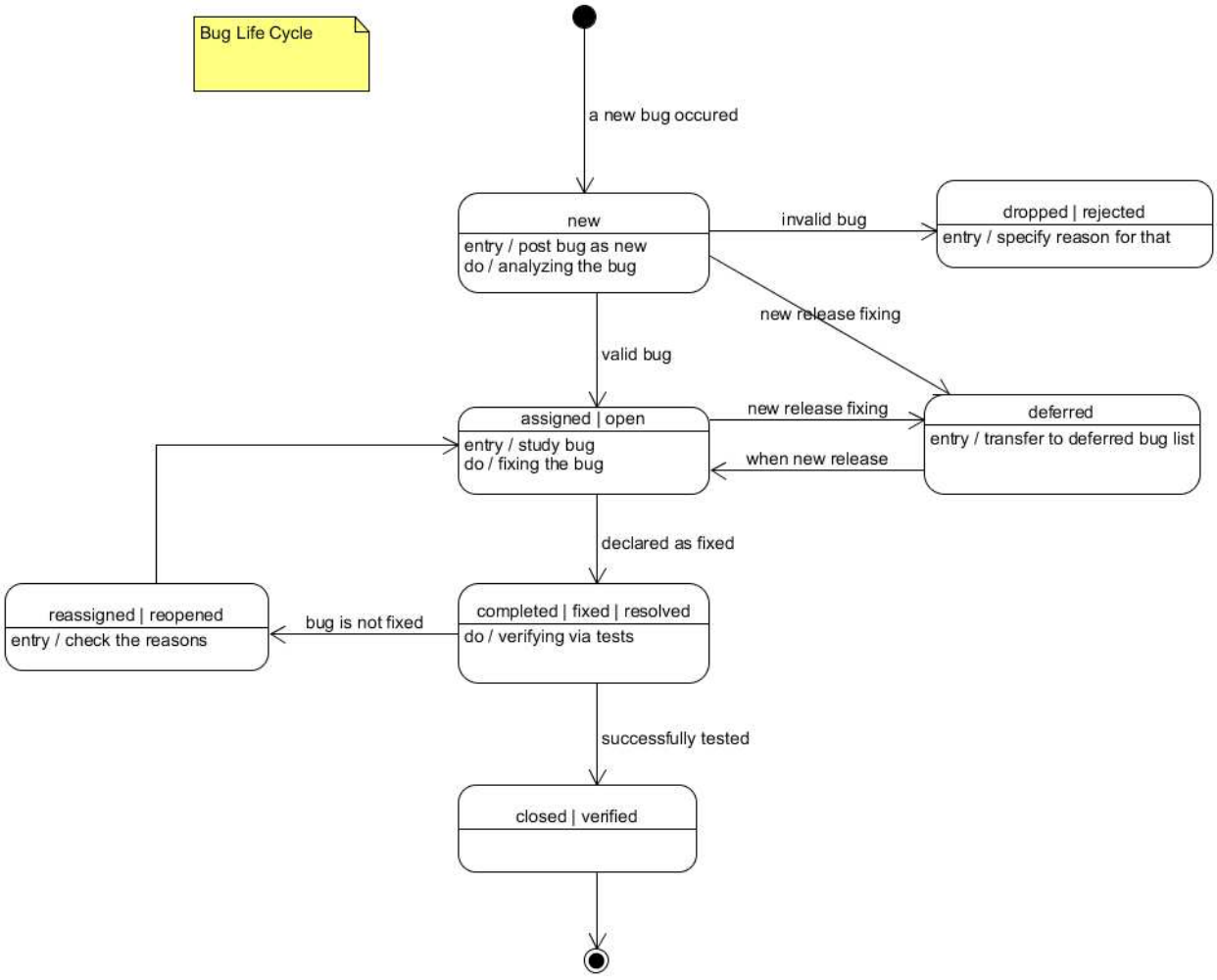


Abb. 4: Zustandsdiagramm für den Bug Life Cycle.

Es wird dabei von den folgenden Zuständen ausgegangen:

- **new**: Ein neuer Bug wurde gefunden. Der Bug wird zunächst analysiert.
- **dropped / rejected**: Der neue Bug wurde als ungültig deklariert bzw. als Bug zurückgewiesen.
- **assigned / open**: Ein gültiger Bug wurde einem Entwickler zur Behebung zugewiesen.
- **deferred**: Die Behebung eines gültigen Bugs wurde auf das nächste Release aufgeschoben.
- **completed / fixed / resolved**: Der für den Bug zuständige Entwickler hat den Bug als behoben deklariert. Das Testteam prüft, ob der Bug tatsächlich beseitigt wurde.
- **reassigned / reopened**: Das Testteam hat festgestellt, dass der Bug entweder gar nicht oder nur teilweise behoben wurde. Der Bug wird wieder an den zuständigen Entwickler verwiesen.
- **closed / verified**: Das Testteam hat verifiziert, dass die Behebung des Bugs erfolgreich gewesen ist. Der Bug gilt damit als behoben und der Fall kann abgeschlossen werden.

Die möglichen Zustandübergänge (Transitionen) werden durch die Pfeile zwischen den Zuständen symbolisiert (s. a .**Abb. 4**).

Ein Zustandsübergang benötigt in der Regel ein auslösendes Ereignis (Trigger). Üblicherweise werden die auslösenden Trigger am Transitionspfeil notiert.

Beispiel (s. a. .**Abb. 4**):

> Damit ein neu auftauchender Bug vom Zustand **new** in den Zustand **assigned / open** übergehen kann, muss er zuvor als gültiger Bug spezifiziert worden sein, also zuvor das Ereignis **valid bug** eingetreten sein.

2.2 Detailinformationen zur Bug-Klassifikation

2.2.1 Status eines Bugs (Bug Status):

Der aktuelle Status eines Bugs (Bug Status) richtig sich nach dem Lebenszyklus eines Bugs (bug life cycle) und ist daher efiniert durch die folgenden Levels:

- **new**: Ein neuer Bug wurde gefunden. Der Bug wird zunächst analysiert.
- **dropped / rejected**: Der neue Bug wurde als ungültig deklariert bzw. als Bug zurückgewiesen.
- **assigned / open**: Ein gültiger Bug wurde einem Entwickler zur Behebung zugewiesen.
- **deferred**: Die Behebung eines gültigen Bugs wurde auf das nächste Release aufgeschoben.
- **completed / fixed / resolved**: Der für den Bug zuständige Entwickler hat den Bug als behoben deklariert. Das Testteam prüft, ob der Bug tatsächlich beseitigt wurde.
- **reassigned / reopened**: Das Testteam hat festgestellt, dass der Bug entweder gar nicht oder nur teilweise behoben wurde. Der Bug wird wieder an den zuständigen Entwickler verwiesen.
- **closed / verified**: Das Testteam hat verifiziert, dass die Behebung des Bugs erfolgreich gewesen ist. Der Bug gilt damit als behoben und der Fall kann abgeschlossen werden.

2.2.2 Priorität eines Bugs (Bug Priority):

Die Priorität eines Bugs (Bug Priority) ist definiert durch die folgenden Levels:

- **Urgent**: muss im nächsten Build bereinigt worden sein.
- **High**: muss in einem der folgenden Builds gelöst worden sein, spätestens aber bis zum Release.
- **Medium**: kann nach dem Release oder im nächsten Release gelöst werden.
- **Low**: kann, muss aber nicht bereinigt werden.

2.2.3 Schweregrad eines Bugs (Bug Severity):

Der Schweregrad eines Bugs (Bug Severity) ist definiert durch die folgenden Levels:

- **Critical**: Der Bug beeinträchtigt kritische Funktionen oder Daten des Systems. Es gibt kein Workaround. Beispiel: Nicht erfolgreiche Installation, Vollständiges Fehlen eines Features.
- **Major**: Der Bug beeinträchtigt wichtige Funktionen oder Daten. Es gibt ein Workaround, der aber zu undurchsichtig ist, um eine echte Lösung darstellen zu können. Beispiel:
- **Minor**: Der Bug beeinträchtigt nur untergeordnete Funktionalität oder nicht-kritische Daten. Es gibt einen einfach umzusetzenden Workaraound.
- **Trivial**: Der Bug beeinträchtigt keine Funktionalität oder Daten. Ein Workaround ist nicht erforderlich. Die Produktivität oder Effizienz des Systems wird nicht reduziert. Beispiele: Unschönes Layout, Schreibfehler / Grammatikalische Fehler.

2.2.4 Tagging von Bugs (Tags):

Für das Verschlagworten (Tagging) von Bugs gelte die auf der nächsten Seite aufgeführten Liste von Schlagworten (Tags).

Liste für die Bug Tags

Tag	Description
alert	For critical bugs requiring immediate attention. Triggers IRC notification.
ci	A Bug affecting the Continuous Integration system
config-agent	A bug affecting os-collect-config, os-refresh-config, os-apply-config.
containers	A bug affecting container based deployments.
documentation	A bug that is specific to documentation issues.
i18n	A bug related to internationalization issues.
low-hanging-fruit	A good starter bug for newcomers.
networking	A bug that is specific to networking issues.
promotion-blocker	Bug that is blocking promotion job(s).
quickstart	A bug affecting quickstart.
selinux	A bug related to SELinux.
pyclient	A bug affecting python-client.
ui	A bug affecting the user interface.
upgrade	A bug affecting upgrades.
validations	A bug affecting the Validations.
apport-bug	A bug reported using 'Report a Problem' in an applications Help menu contains lots of details.
apport-crash	A crash reported by apport - Ubuntu's automated problem reporter.
package-conflict	A bug reported by apport when a package operation failed due to a conflict with a file provided by another package.
desktop-file	The bug requests the addition/fix of a .desktop file.
fix-to-verify	A bug that is Fix Released and should be verified when performing iso testing of daily builds or milestones.
ftbfs	Bugs describing build failures of packages.
hw-specific	A bug that requires a specific piece of hardware to duplicate.
iso-testing	A bug found when performing iso testing.
likely-dup	The bug is likely a duplicate of another bug ((maybe an upstream bug too) but you can't find it.
manpage	This bug is about a package's manpage being incorrect.
metabug	This bug has a high probability of duplicate reports being filed.
desktop-icons	Bugs related to the desktop, especially the alignment, display and grid of icons.
needs-reassignment	A bug that was reported about the wrong package but the package maintainer isn't sure which package it belongs to.
packaging	The bug is likely to be a packaging mistake.
screencast	This bug report includes a screencast of the bug in action.

string-fix	This bug is a string fix (not code) and is great for new contributors. For spelling and grammatical errors.
unmetdeps	Bugs that indicate packages not being installable due to missing dependencies.
upgrade-software-version	Bugs that request new software versions - please help reviewing them carefully.
work-intensive	Triaging requires intensive work to validate/reproduce.
dist-upgrade	A bug that was encountered when upgrading between releases of specified product.
testcase	A bug which contains a test case - steps to recreate the bug.

2.3 Rollenbasierte Benutzerrechte (Account Role):

Für das BTS ist sicherzustellen, dass nur berechtigte Anwender / Nutzer Änderungen vornehmen dürfen.

Hierfür bietet sich ein rollenbasiertes (AccountRole) Zugriffsrechtekonzept an, dass die Zugriffsrechte an allen Datenbank-Objekten regelt.

Die Rolle eines Nutzer (AccountRole) ist definiert durch die folgenden Einträge:

- **BTS_Admin**: fungiert als Adminstrator des BTS und besitzt alle Rechte.
- **BTS_Simple_User**: darf nur lesend auf alle BTS-Objekte zugreifen
- **BTS_Advanced_User**: darf in der Regel nur lesend auf BTS-Objekte zugreifen. Ein schreibender Zugriff ist nur auf die folgenden BTS-Objekte erlaubt:
    - **Bug, Screenshots, Comment.**

2.4 Typen von Benutzern (Account Type):

Für das BTS sind bestimmte Arten von Benutzern (Account Type) einzurichten.

Es wird vorläufig von den folgenden Benutzertypen ausgegangen:

Product Manager	Software Engineer	Software Tester
Artist / Illustrator	Programmer	Community Manager
Game Designer	Administrator	°Individual (private person)

2.5 Anwendungsfälle des Bug Tracking Systems

Eine vorläufige Anwendungsfallanalyse ergab die folgenden funktionalen Anforderungen:

- Mitarbeiter / Anwender registrieren
    - bei Mitarbeitern ist der Stundensatz anzugeben
    - bei Anwendern ist dies obsolet
    - Mitarbeiter-Foto / Bild hochladen
- Bug-bezogene Funktionen:
    - Bug melden (durch Mitarbeiter / Anwender)
    - Bug an Mitarbeiter zuweisen
    - Bug durch Mitarbeiter verifizieren
    - Bug-Status festsetzen / ändern
    - Bug-Priorität festsetzen / ändern
    - Bug-Schweregradt festsetzen / ändern
    - Bug mit Produkten assoziieren
    - Bug-Tags vergeben
    - Screenshots hochladen und in Datenbank speichern
    - Kommentar-Threads verfolgen / einsehen
    - Kommentare eingeben
    - Bug – Historie anzeigen (vollständig oder teilweise)
- Produkt-bezogene Funktionen:
    - Produkt CRUD
    - Ansprechpartner für Produkte eintragen
- CRUD für die Tabellen : BugStatus, BugPriority, BugSeverity , Products, Accounts, Screenshots, Tags,

CRUD : Create, Read, Update, Delete.

- Zu implementierende anwendungsbezogene Abfragen:
    - Welche Probleme sind aufgetreten?
    - Welcher Art sind die Probleme?
    - Wie wurde das Problem dokumentiert?
    - Wer hat das Problem gemeldet?
    - Welcher Entwickler ist für das Problem zuständig?
    - Wer hat das Problem als solches verifiziert?
    - Welches Produkt ist von diesem Problem betroffen?
    - Welches Produkt hat welche Probleme?
    - Was wurde bereits zur Problembehebung unternommen?
    - Wie hoch war der tatsächliche Aufwand zur Problembehebung?
    - Welche Mitarbeiter waren an der Problembehebung beteiligt?

2.6 Weitere Anforderungen

Das **BTS** soll als datenbankgestützte Anwendung nach dem Client-Server-Prinzip konzipiert werden, wobei das Ziel darin besteht, die **Kernfunktionalität des BTS** möglichst **ohne** HTML-basierte Web-Clients und serverseitiges PHP zu implementieren.

Dies bedeutet, dass Anwender über Shell-basierte Clients Anfragen an die serverseitig gehostete BTS-Anwendung stellen, deren Bearbeitung hauptsächlich durch auf dem Server ablaufende Stored Routines bewerkstelligt werden.

Hierfür sind die **Anwendungsfälle des BTS** unter Zuhilfenahme von **PL/SQL**, der prozeduralen Programmiersprache von MySQL, in Gestalt von **Triggern, Stored Procedures oder Stored Functions** ggf. in Verbindung mit **Transaktionen** zu realisieren. Da Trigger / Stored Procedures Datenbank-Objekte sind, die direkt im RDBMS (MySQL) gespeichert sind, können diese über den Shell-basierten Client MySQL aufgerufen werden, um so einzelne Anwendungsfälle zu testen.

Da es sich um eine datenbank-gestützte Anwendung handeln soll, ist die referentielle Integrität sowie die Datenintegrität mit Mitteln des zu verwendenden RDBMS (MySQL) sicherzustellen. Hierfür muss in MySQL die transaktionssichere Storage Engine **InnoDB** für Tabellen verwendet werden. Gegebenenfalls sind dafür auch Stored Procedures, User Defined Functions und verschiedene Trigger-Prozeduren einzusetzen.

Zur Beantwortung der verschiedenen an das BTS gerichteten Fragestellungen sind auf die einzelnen Fragen zugeschnittene SQL-Views vorzusehen, die für diese als Datenbasis dienen.

CRUD-Operationen (Create, Read, Update, Delete) für die Basistabellen sollen über speziell dafür ausgelegte Stored Procedures vorgenommen werden.

Screenshots sollen vom Anwender hochgeladen werden können und anschließend in den Datenbanktabelle Screenshots abgelegt werden. Hierfür kann die MySQL-Funktion **LOAD_FILE(....)** eingesetzt werden.

Für das BTS ist ein adäquates Datenbank-Sicherheitskonzept, dass für den Benutzer rollenbasierte Zugriffsrechte auf die Datenbank-Objekte festlegt.

2.6.1 Anfordungen an die Dateninhalte

Die folgenden Basistabellen müssen **mindestens 10 Datensätze** enthalten:

- Accounts
- Bugs,
- Screenshots,
- Products,

Die nachfolgenden Tabellen sollten **mindestens 20 Datensätze** enthalten, die inhaltlich stimmig sind.

- Comments,

Für die nachfolgenden Basistabellen sind die vorgeschlagenen Wertelisten aus dem Lastenheft zu verwenden:

- Tags
- BugPriority
- BugStatus
- BugSeverity

2.7 Umgebung

2.7.1 Software

- Betriebssystem PC / Notebook
  - Windows 8 / 10
  - Linux
- XAMPP Bundle aus Webserver, RDBMS und Skriptsprache PHP
  - Apache 2.x, MariaDB 10.4.x, PHP 5.7.x
- Datenbank-Client:
  - HeidiSQL 11.x,
- Tool zum Zeichnen von UML-Diagrammen für die Datenmodellierung:
  - UMLet 14.x
  - UMLetino (Web-Version von UMLet)
- Datenbankentwurfs-Tool:
  - DB Designer 4.x
  - MySQL Workbench 6.3 Community Edition (CE)

2.7.2 Hardware

- PC, Notebook
- ...

2.7.3 Produktschnittstellen

- Zum Bediener
- zum DBMS