**Bug Management in Gaming Studio**

# **BACKGROUND OF THE COMPANY**

ABC is a gaming studio which produces games from platforms such as mobile, Nintendo to Xbox and PS5. The studio works in small factions of teams where each team has a manager, teams of developers and team of testers. In each development cycle, the development team works on gaming features and the testing team tests the features before they are rolled out to gaming users. Of the genres that ABC company works is Online gaming platform where multiple users play.

While playing online games a user may encounter many problems. In technical terms these problems are called bugs. A bug can be anything as simple as “a graphical glitch like a tree in the wall”, to “Missing valuable gaming currencies or in-game items”. Because of the fast-changing technologies and complexities, online games lead to a plethora of bugs.

There can be different types of bugs such as

* User Interface
* Gameplay – Features in the games
* Live chat support

These kinds of bugs are reported very frequently by users and testers. The motto of the process

Once a bug is reported, bugs are assigned by managers, fixed by developers, and tested by testers. In this process, a bug goes through many handovers and dependencies. A tester cannot test the fix until the developer submits the code. A developer cannot submit until his code is reviewed by the manager. A system engineer cannot upload a package till the developer has a **Go** from the testers. Through the process, few handovers can be reduced by doing them in parallel manner. A manager and developer is sitting idle while the tester does the testing. This time can be utilized to get the code reviewed by managers. Developer submitting and troubleshooting code packages on a game server takes away his valuable hours that could have been utilized better to fix more bugs. Thus, hiring a system engineer who will be in charge of roll-outs, roll-backs of packages can improve the overall process efficiency.

# **ISSUES AND SOLUTIONS**

* + 1. **Lack of documentation*:***

**Existing step:** All the documentations are done only in cases when an employee resigns or the project is being handed over to another team.

**Improvement:** In the step when a developer gets the code reviewed by the Tech manager and tester, he should document the code right then and there. Saving the hassle of doing imperfectly later.

* + 1. **Unavailability of an employee*:***

**Existing step:** Unless it’s urgent, the rest of the dependencies would wait for the employee to return and finish the task.

**Improvement:** If possible, work should be allowed online (work from home). Another employee should be assigned to collaborate with the absent employee to work on the task.

* + 1. **Expedite the bug fixes review hand over:**

**Existing step:** Development of bug fix, and the corresponding code review is done sequentially

**Improvement:** Testing and Review can be done parallelly to save hand over time*.*

* + 1. **One change/ fix impacting other features of games**

During each testing, testers should do a grand thorough test, just to check if the new update has affected any existing features.

* + 1. **Policies and Rules**

‘*If it ain’t broke, don’t fix it!*’.

The above philosophy employed by the game studio refrained many developers from making changes in existing code. The game code was taken over from a small studio thus it had many patchy fixes and patchy systems. But improving the code and making it robust for future generations was difficult.

# **CURRENT PROCESS (AS IS)**

The current process of the game studio to fix a bug steps are as follows

1. Once a bug is identified and reported by a gamer, it is gathered and reported into a project management tool by a Tester employee.
2. Manager of the game team will assign a developer according to the competencies, severity (high/medium/low) and the experience of the developer.
3. The developer now writes code to fix the bug. He then performs a mini-test and forwards the code to the testing team to do thorough testing.
4. Testing team performs all the variations to check if the bug is fixed.
5. Developer submits code on the server.
6. A developer creates a package that can be integrated into the live game.
7. Package is uploaded onto the live game.
8. Bug is fixed.

# **SOLUTION PROCESS (TO BE)**

Adding to the steps mentioned in the AS-IS process, below are the additional solution steps that are included in the process.

1. Instead of developers creating packages, the company can hire a system engineer to be permanently in charge of creating and managing packages. That way if there is any bug related to the package, it can be handled by the system engineer thus reducing the overhead of a developer.
2. Testing and Code review process can go hand in hand (in parallel) to reduce the handover duration.
3. Testers can be divided into two categories. One type of testers will test the bug fixes provided by the developers and the other will always check if the fix is affecting the rest of the game logic. That way in case a fix is affecting any other feature of the game or worse, if it’s crashing the game, the company can sieve out such fixes from getting uploaded into the game.
4. As developer submits the code on server with the confirmation from tester that the bug has been fixed, she should document the learnings and new features that were introduced through the bug fix.
5. The above philosophy employed by the game studio refrained many developers from making changes in existing code. The game code was taken over from a small studio thus it had many patchy fixes and patchy systems. But improving the code and making it robust for future generations was difficult.

-The below file represents the swim lane diagram for the to be process



# **BUSINESS RULES AND ERD**

# **BUSINESS RULES**

* Bug is reported by either a game user or the in-house tester.
* Manager does the assessment of the bug, checks it’s severity and type. After which he decided the ideal match of a developer that would be able to fix the bug.
* Developer writes a code to fix the bug and gets it reviewed by the manager.
* Once the tester gives a **Go** that code has fixed the issue, developer uploads the code to the code server and documents his learnings into documentation.
* Now the status of the bug has changed from “Assigned” to “Resolved”.
* System engineer then picks up the code file and creates a package to be uploaded onto the game server.
* Bug is resolved.

# **ENHANCED ENTITY RELATIONSHIP (ERD)**

The below points are covered in this section

# **Entities**

# **Relationships**

# **Cardinalities**

# **ERD Diagram**

1. ABC company receives many defects in games reported by users across the world from different types of devices such as PS5, Nintendo, PC or Mobile. The company is keeping all the records of the bug to avoid rework on the duplicate or similar type bugs. They are maintaining unique game id for the bug, bug details, bug status, bug reporting date, and bug resolution date.

2. The manager takes the responsibility to assign the bugs in the pool to appropriate developers based on its severity (High, medium, low), bug type (UI, game, chat, online) and bug platform.

3. A developer can be assigned to one or more bugs at a time by the game manager. When a bug is assigned to a developer it will be in “Assigned State”, for duplicate bugs the status would be “duplicate”, for the defects which are not bugs would be in “removed” status. At last, once the bug is fixed by the developer – the status would be changed to “Ready To Test” and the developer would also upload the bug fix code in code file on a server.

4. A developer is also responsible for maintaining all the updates on the bug in a document such that multiple developers can update comments on multiple documents of the bugs and vice versa.

5. As soon as the status of the bug is changed to “Ready To Test” - a Tester picks up that bug and checks if it is working fine in the testing environment. If the bug is fixed, the status would then be changed to “Done” else it would be changed to “New”. A same Tester can test one or more bugs at a time.

6. The code server has the details of the fixes on a bug that needs to upload to the production environment. Each code in the server is accessed by one or more System Engineers, which further transfers the code into one or more packages. A package is then uploaded to a game server.

-The below files have the ERD diagram for the process



# **ENHANCED ENTITY RELATIONSHIP DIAGRAM (EERD)**

# **SUPERTYPE**

**Employees** –

Generic Employee entity that will be supertype to multiple types of employees in the organization.

# **SUBTYPES**

An employee can be of different departments and designations. For the sake of this project, the Supertype divides into **Manager, Developer, Tester, and System Engineer.** With each of them having association with the game they’re working on.

# **DATA DICTIONARY**

The below file contains the data dictionary for the ‘Bug management in gaming management’ process



The Below Table contains the data dictionary for the ‘Bug management in gaming management’ process

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Sno.** | **Table Name** | **Table Description** | **Attributes of the table** | **Attribute description** | **Data Type** | **Data Type Size** | **Constraints** |
| 1 | Bug\_Details | This table contains details of the bugs  reported for a particular game | Bug\_ID | A unique identifier to keep a track of bug details | int | 5 | Primary Key |
| Game\_ID | The game identifier for which the bug was reported | int | 8 | Foreign Key |
| Developer\_ID | The developer employee who is responsible for resolving the bug | int | 10 | Foreign Key |
| Tester\_ID | The tester employee who is responsible for  testing the application once the bug is resolved | int | 10 | Foreign Key |
| Bug\_Details | Details of bug reported for a particular game | varchar | 100 | Not Null |
| Bug\_Type | The kind of bug that is reported | varchar | 10 | Check - {'UI','Game','Chat','Online'} |
| Bug\_Platform | The platform where the bug was found | varchar | 10 | Check - {'PS5','PC','Nintendo','Mobile'} |
| Bug\_Severity | The severity of the bug reported | varchar | 10 | Check - {'High','Medium','Low'} |
| Bug\_Report\_Date | Captures the date when the bug was reported | date | date | Not Null |
| Bug\_Resolve\_Date | Updates the date once the bug is fixed | date | date | Not Null |
| Bug\_Status | The status of the bug fix | varchar | 15 | Check - {'New','Approved','Done','Removed','Duplicate','Ready To Test"} |
|  |  |  |  |  |  |  |  |
| 2 | Game\_Details | This table contains details of different kinds  of games within the company | Game\_ID | A unique identifier for different kinds of games available | int | 8 | Primary Key |
| Game\_Name | The name of the game | varchar | 30 | Not Null |
|  |  |  |  |  |  |  |  |
| 3 | Emp\_Details | This table contains details of the employees  in the company | Emp\_ID | A unique identifier of employee working in the company  (Can be developer, tester, manager etc | int | 10 | Primary Key |
| Emp\_Name | The name of the employee working for the company | varchar | 30 | Not Null |
| Emp\_DOJ | The date when the employee joined the company | date | date | Not Null |
|  |  |  |  |  |  |  |  |
| 4 | Manager\_Emp\_Details | This table contains details of Manager who manages  bugs for a particular game | Manager\_Game\_ID | This is a composite key indicating employee(manager) and game pair  - key & value pair | int | 10 | Primary Key |
| Game\_ID | The game identifier for which the bug was reported | int | 8 | Foreign Key |
| Emp\_ID | The manager employee ID who is assigned to work on the bug fix for the game | int | 10 | Foreign Key |
|  |  |  |  |  |  |  |  |
| 5 | Tester\_Emp\_Details | This table contains details of Tester who would be  testing bugs for a particular game | Tester\_ID | This is a composite key indicating employee(tester) and game pair  - key & value pair | int | 10 | Primary Key |
| Game\_ID | The game identifier for which the tester will test post the bug fix | int | 8 | Foreign Key |
| Emp\_ID | The tester employee ID who is assigned to work on the bug fix for the game | int | 10 | Foreign Key |
|  |  |  |  |  |  |  |  |
| 6 | Developer\_Emp\_Details | This table contains details of Developer who would be  resolving bugs for a particular game | Developer\_ID | This is a composite key indicating employee(developer)  and game pair - key & value pair | int | 10 | Primary Key |
| Game\_ID | The game identifier for which the developer will work on the bug fix | int | 8 | Foreign Key |
| Emp\_ID | The tester employee ID who is assigned to work on the bug fix for the game | int | 10 | Foreign Key |
|  |  |  |  |  |  |  |  |
| 7 | Sys\_Engineer\_Details | This table contains details of System Engineer who creates  game packages and uploads the package into the server  once the bug is fixed by the developer | Sys\_Eng\_ID | This is a composite key indicating employee (system Engineer)  and code\_server pair - key & value pair | int | 10 | Primary Key |
| Code\_Server\_ID | This is the server Id where the fix code can be used by System  engineer to create a game package | int | 8 | Foreign Key |
| Emp\_ID | The system Engineer employee ID who is assigned to work  on creating packages post bug fix | int | 10 | Foreign Key |
|  |  |  |  |  |  |  |  |
| 8 | Developer\_Logs | Post bug bix, the developer maintans logs  to keep a track of the fixes made | Developer\_Log\_ID | This is a unique log Id for the bug fixed by developer | int | 10 | Primary Key |
| Doc\_ID | The doc ID where the developer has documented the learnings from the bug bix | int | 8 | Foreign Key |
| Developer\_ID | The developer ID who works on the log creation | int | 10 | Foreign Key |
|  |  |  |  |  |  |  |  |
| 9 | Bug\_Fix\_Document | The developer who fixes the bugs of a game, documents the cause of bugs, fixes made and learnings | Doc\_ID | The Doc ID where the developer has documented the learnings from the bug bix | int | 8 | Primary Key |
| Doc\_Name | A document name for the file created | varchar | 30 | Not Null |
| Doc\_Date | The date the document was created | date | date | Not Null |
| Bug\_ID | Foreign Key - A unique identifier to keep a track of bug details | int | 5 | Foreign Key |
|  |  |  |  |  |  |  |  |
| 10 | Code\_Server\_Details | This contains server details of the code where a  developer uploads the code fixes made for the bugs reported | Code\_Server\_ID | This is the server Id where the fix code is uploaded | int | 2 | Primary Key |
| Code\_File\_Name | This represents the code server name | varchar | 30 | Not Null |
| Developer\_ID | The developer employee ID who uploads the code in the code server | int | 5 | Foreign Key |
| Game\_ID | Game ID for each code that is getting uploaded in a server |  |  |  |
|  |  |  |  |  |  |  |  |
| 11 | Game\_Package\_Details | This contains game package details that is created  by the system engineer once the fix code is available | Package\_ID | The unique package ID to keep a track of the game packages created | int | 10 | Primary Key |
| Bug\_ID | The bug ID for which the package is to be created for the code fix done | int | 5 | Foreign Key |
| Sys\_Engineer\_ID | The sys\_engineer employee ID who works on the package creation | int | 10 | Foreign Key |
| Package\_Name | This is the name provided for the package created by the system engineer | varchar | 30 | Not Null |
|  |  |  |  |  |  |  |  |
| 12 | Game\_Server\_Details | This contains game server details where the game packages is uploaded by the system engineer for the game package created | Game\_Server\_ID | The unique Game server ID to upload the package created | int | 2 | Primary Key |
| Game\_Server\_Name | This is the name provided for the server created by the system engineer | varchar | 30 | Not Null |
| Bug\_ID | The bug ID for which the package is to be uploaded in the server | int | 5 | Foreign Key |

# **REPORT REQUIREMENT FOR BUSINESS**

1. **Query Report-1**

**Business Case:**

For any bug that was reported, once the bug is fixed, how the fix was made should be documented by the employee who worked on it. The document should include details of the bug resolution date, whether the document has been created or not, if not created then how many days has it been since the bug was fixed.

**Benefits:**

By having this report, the employee who worked on the bug fix can be notified to create a bug fix documentation to have records on how it was fixed, how long did it take to fix the bug, what was the severity of the bug, how many users were impacted. By doing so, if a similar bug is reported in the future, then this document can be referred for a quick resolution.

**Report Query Question:**

Give a report that shows the bug list for which the bug fix document was not created post the bug was fixed. Report should display the bug resolved to date, right next to it display how many days has it been since the resolved date and the current date. Also, display the employee who worked on it (emp id, concat first name, last name with space in between), what is the bug Id, severity of the bug. Display the output only if there is no documentation available. Order the output by ascending order by BugID.

**SQL Query:**

Select B.Bug\_Id, B.Bug\_Severity, DM.Emp\_ID,

(E.Emp\_First\_Name || ' ' || E.Emp\_Last\_Name) AS "Employee\_Name",

BM.BUG\_RESOLVED\_DATE,

abs(trunc(sysdate) - BM.BUG\_RESOLVED\_DATE) AS "No of Days since Bug resolved"

from **RUP21001.**bug\_details B

join **RUP21001.**bug\_management BM

on B.Bug\_ID=BM.Bug\_ID

join **RUP21001.**bug\_fix\_document BD

on B.Bug\_ID = BD.Bug\_ID

join **RUP21001.**Document\_Management DM

on BD.Bug\_Doc\_ID = DM.Bug\_Doc\_ID

join **RUP21001.**Emp\_Details E

on DM.Emp\_ID=E.Emp\_ID

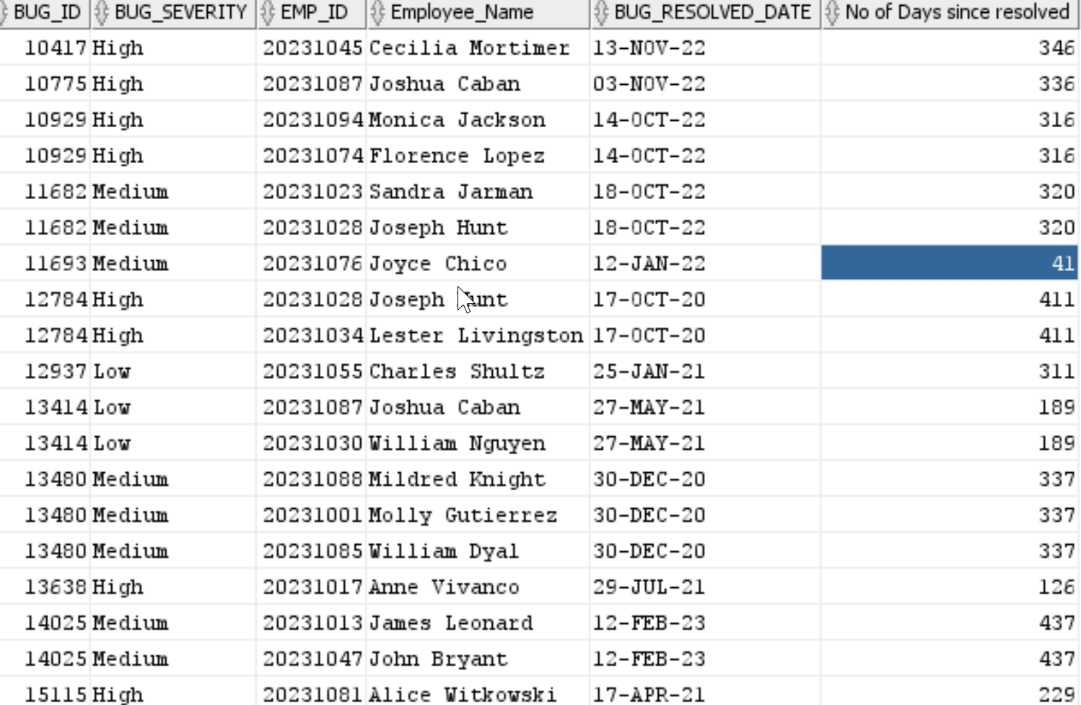
where DM.Doc\_Created\_Date is not Null

and B.Bug\_Status='Resolved'

order by B.Bug\_ID

;

**Output :**





1. **Query Report-2**

**Business Case:**

There are 9 distinct types of bugs based upon where are they generated in the game. For example, Save Glitch, In-Game Purchase, Crashing, Graphics, and so on. A manager would like to know which type of bug is being created the greatest number of times and what’s the avg fix time (resolve date - report date) for the same.

**Benefits:**

By knowing what type of bugs generate the greatest number of bugs and consume most work hours, a manager can reassign the bug severity to these types and allocate resources accordingly. Usually, the development team is divided according to the types of bugs they work upon. So, the relocation can include assigning a senior programmer to the type of the most severe bug.

**Report Query Question:**

Give a report that will give bug fix duration (resolved - reported) and the count of the bugs for each bug type.

**SQL Query:**

Select BD.bug\_type, trunc(Avg(BM.bug\_resolved\_date - BM.bug\_report\_date)) as Avg\_Bug\_Fix\_Duration, count(\*) as No\_Of\_bugs

from **RUP21001.**bug\_management BM

join **RUP21001.**bug\_details BD

on BM.bug\_id = BD.bug\_id

group by bd.bug\_type

order by Avg\_Bug\_Fix\_Duration, No\_Of\_bugs desc

;

**Output:**

Graphical user interface, application

Description automatically generated



1. **Query Report-3**

**Business Case:**The Project Manager would like to know for each game how many packages have been created for the bug reported.

**Benefits:**This will help to understand which game is more active depending on how many packages have been created for the game, the higher the count that means requires more system engineer employees are required for that game.

**Report Query Question:**Create a report that shows a count of packages created for each game, order the report by a count of packages in descending order.

**SQL Query:**

select gd.game\_name, count(gpd.package\_id) Count\_of\_Packages from RUP21001.game\_package\_details GPD

join RUP21001.Bug\_Details BD

on GPD.package\_id = bd.package\_id

join RUP21001.Game\_Details GD

on bd.game\_id = gd.game\_id

group by gd.game\_name

order by count\_of\_packages desc

;

**Output:**



**Table

Description automatically generated**

1. **Query-Report-4**

**Business Case:** For the annual employee evaluation, the Project Manager is trying to find ways in which he could measure the employee performance. He observes that there is a difference between the number of bugs resolved by each employee and that could help to provide the potential annual bonus too.

**Benefits:** Employees can be evaluated without any bias and the annual bug resolution performances can be shared with each individual employee. The bonus based on the bugs will help motivate the employees to improve their productivity and resolution rate

**Report Query Question:** Create a report showing the developer and the number of bugs they have resolved. The project manager has decided to award the extra bonus for employees that will be added to their net salary as follows –

1. If bug resolved count is between 1 to 10 – provide a bonus of $150 per bug resolved
2. If bug resolved count is between 11 to 20 – provide a bonus of $250 per bug resolved
3. If bug resolved count is between 21 to 30 – provide a bonus of $350 per bug resolved
4. If bug resolved count is greater than 31 – provide a bonus of $500 per bug resolved

Display the First Name, Last Name, Salary, Count of Bugs, and Bonus

**SQL Query:**

**SELECT**

**EMP\_ID, EMP\_FIRST\_NAME AS FIRST\_NAME, EMP\_LAST\_NAME AS LAST\_NAME, TO\_CHAR(EMP\_SALARY, '$999,999.00') AS ANNUAL\_SALARY, BUG\_COUNT,**

**(CASE**

**WHEN BUG\_COUNT <10 AND BUG\_COUNT > 0 THEN TO\_CHAR(BUG\_COUNT\*150, '$999,999.00')**

**WHEN BUG\_COUNT <21 AND BUG\_COUNT > 10 THEN TO\_CHAR(BUG\_COUNT\*150, '$999,999.00')**

**WHEN BUG\_COUNT <31 AND BUG\_COUNT > 20 THEN TO\_CHAR(BUG\_COUNT\*150, '$999,999.00')**

**ELSE TO\_CHAR(BUG\_COUNT\*150, '$999,999.00') END) AS "EMPLOYEE BONUS"**

**FROM (**

**SELECT**

**D.EMP\_FIRST\_NAME, D.EMP\_LAST\_NAME, D.EMP\_SALARY, B.EMP\_ID, COUNT(BUG\_ID) BUG\_COUNT**

**FROM RUP21001.BUG\_MANAGEMENT B**

**LEFT JOIN RUP21001.EMP\_DETAILS D**

**ON B.EMP\_ID = D.EMP\_ID**

**GROUP BY D.EMP\_FIRST\_NAME, D.EMP\_LAST\_NAME, D.EMP\_SALARY, B.EMP\_ID**

**)**

**ORDER BY BUG\_COUNT DESC;**

**OUTPUT:**

****

**Table

Description automatically generated**

1. **Query Report-5**

**Business Case:**

For any bug that was reported for a game, the company needs to keep a track of the progress being made in the bug fix, there are several kinds of status being maintained for the bug pix progress like resolved, new, committed, etc. So, the Department Head or top-level manager requests the status of progress for each game.

**Benefits:**

By having this report, the company can maintain records of the status of the work, how long does it take to resolve the bugs reported, what escalations can be made to stakeholders that fixing a bug for a particular game would take X number of days.

**Report Query Question:**

Give a report that shows the bug status for all the bug statuses available, and if a bug doesn’t have a status type display it as ‘Null’ for each game. Provide the count of each bug status for each of the games the company handles.

**SQL Query:**

select gd.game\_name, bug\_status,

(

case when bug\_status = 'Commited' Then count(bug\_status)

when bug\_status = 'Resolved' Then count(bug\_status)

when bug\_status = 'Removed' Then count(bug\_status)

when bug\_status = 'New' Then count(bug\_status)

when bug\_status = 'Approved' Then count(bug\_status)

else 0

end

) as count

from bug\_details BD join game\_details GD

on bd.game\_id = gd.game\_id

group by bug\_status, gd.game\_name

order by gd.game\_name

;

**OUTPUT:**

**Table

Description automatically generated**

