# Unifying Bug Tracking with Design Data Management

Roger March,

Chief Technology Officer, IC Manage Inc.

Design verification continues to take up a majority of the IC design effort in a project. The effort is further complicated by the fact that design and verifications teams can be composed of many groups dispersed across local and global organizations. It is critical that members of a project be aware of a bug's existence and status. By unifying bug tracking within design management with revision control, organizations can provide bug traceability to the entire design team throughout the design process. This approach will reduce the overall verification effort and help guarantee that defects do not make their way into the final chip, -and prevent them from appearing in derivative designs.

## Bug Tracking Challenges

Today's bug tracking work flow might look like the following:

1. An issue is originated by a design or verification engineer; customer, -application engineer, usually via a web browser.
2. The bug tracking system logs the issue and notifies a defined dispatcher.
3. The dispatcher evaluates the issue and assigns it to an appropriate handler.
4. The issue is -investigated, fixed and closed.
5. The originator is notified that the issue has been resolved.

Some important criteria in a bug tracking flow are the ability to:

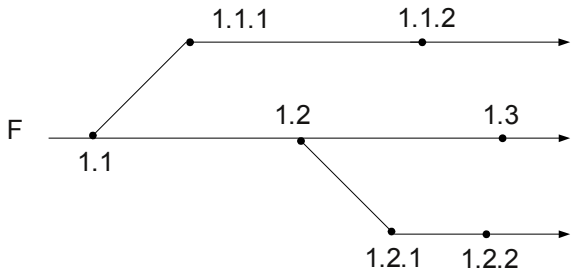
- Ensure design team members have real-time alters regarding bugs that may impact them, with status information.
- Reliably reproduce the original bug -occurrence within the design and all subsequent states of the design as the bug is fixed and verified. A system lacking this feature may make it difficult or impossible for the handler to verify the issue. A verification engineer will have a similar problem confirming the bug has actually been resolved as the design is modified to correct the issue.
- Trace the bug states through the design. Once a fix has been accepted, the verification team must determine whether the changes also apply to other instantiations of the object being fixed or derivative designs of the same object. Performing this task requires both the means of identifying the other instantiations and derivatives and the ability to propagate the changes where appropriate.

These challenges can be met by tightly integrating the IC design management system with the bug tracker. Without this binding, the time and effort to reproduce bugs and verify that they are fixed will be high, and organizations risk leaving avoidable problems not only in the original design, but in related products.

## File-based Design Data Management

It is important to understand the type of the underlying design management system. It affects its integration with the bug tracking system and the best procedures to apply. There are two primary types of IC design data management systems today: File-based and Change-based.

File-based IC design data management systems, often based on CVS and RCS, are simple version archivers which manipulate the version trees of individual files:

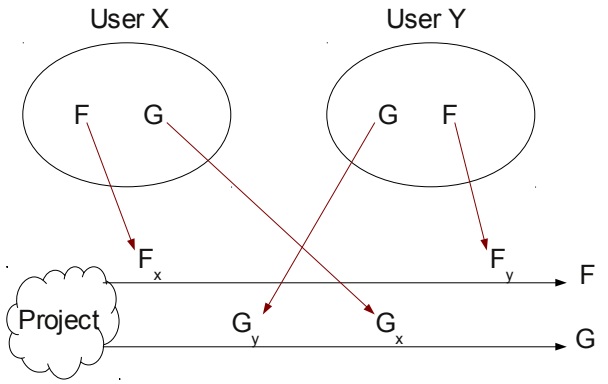



A **project** is typically a directory containing these version-ed files. Changes can be applied to the most recent version of a file on either the mainline or a branch. A project can be moved to a particular state by either specifying a time or enumerating the version of a file.

An -significant limitation with this approach with regard to revision control is that it only deals with individual file changes, while meaningful changes to a project often consists of a set of files. When such multi-file changes

are committed, the files are each checked in serially. If more than one user submits changes at the same time, the change sets can be interleaved, making it impossible for the time line view of the project to reproduce the exact design state of either user.

For example, in the following scenario two users, **X** and **Y**, wish to commit their change sets consisting of files **F** and **G** back to the **Project**. Each user's change set represents a consistent project state, both commit at the same time:




Since everything goes in file by file, $\mathbf{F_x}$ makes it first followed by $\mathbf{G_y}$. Now the trouble happens, $\mathbf{G_x}$ conflicts with the committed $\mathbf{G_y}$ and cannot proceed. Similarly $\mathbf{F_y}$ conflicts with $\mathbf{F_x}$ and must be resolved. The head of the project tree is in a broken state and will remain so until the files can be resolved; at no point does the project contain either user **X** or **Y**'s state.

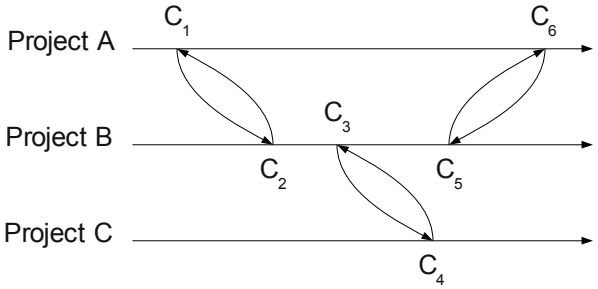
File-based design data management systems introduced the concept of **tagging** to assist with these state problems. Tagging is a means of changing a project to an enumerated file version state, usually implemented as additional meta-data within the file's version tree. While this creates an illusion of project state progression, tags generally carry no relationship information vis-a-vis other tags; thus tag dependencies must be managed outside the configuration management system.

## Change-Based Design Data Management Systems

**Change-based** design data management systems differ from file-based systems as follows:

1. **Atomic changes**. Files are committed atomically in change sets. Either the entire set is committed or none of changes is committed. This eliminates the problem of interleaved submissions and provides a consistent view of state changes along a project development time-line. The change ID that is issued by the commit transaction is a reference identifier that can be used to explicitly tract the design state at any point in the future. The time line view of a project is identical to its change view and implements the ability to reliably reproduce the original bu state.
2. **Branch integration history**. Change-based IC design management systems incorporate higher order state management. A branch integration history mechanism provides the ability to record deltas that have been merged from one file to another. These deltas define a dependency relationship between two files. They explicitly record differences between the files and, by inference, differences in the projects containing them. The integration history also provides an audit trail through successive generations of branches. This makes it possible to compute the files involved between any ancestor and descendant project as well as the specific changes on each file, making it possible to trace the bug states through the design.

The diagram below shows three projects with a file branched between them:




Let us say a problem is found in **Project B** and identified to have originated at $\mathbf{C_5}$. The system can then be used to find that **Project A** may also be at risk. It can also be used to show that a dependency exists on **Project C** beginning at $\mathbf{C_3}$. In this case it can be confirmed that **Project C** is not at risk.

Atomic changes and branch integration history make it easy to define and manage project states. Users can quickly move to a specific design state to confirm the

existence of a bug. They can then query the design data management system to determine projects susceptible to the defect. Finally, the mechanics of branch integration history will assist the user in applying fixes to the dependent projects by making sure that all relevant deltas are applied.

## Bug Management Using Traditional File-based Design Data Management Systems

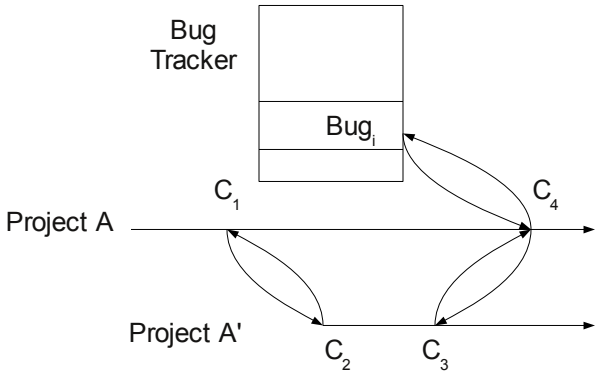
Traditional design data management systems lack binding between the state of the files and the bug tracking system. Thus bug tracking practices for this type of system involves ongoing manual interventions, such as:

1. Manually annotating the defect tracker indicating when and where the changes were committed.
2. Constructing a tag to identify the resolution state point within the configuration system. Although creating such a tag can be a significant task, annotating with an identifying tag can provide unidirectional binding from the specific defect to project state, reproducing the resolution state for verification
3. Manually moving the fix from the resolution branch to the current design state, to compensate for the lack of integration history.
4. Consistently repeating this process for each dependent design that the defect change impacts.

## Integrating Bug Tracking with Change-based Design Data Management

Tightly integrating a change-based design data management system with revision control and a bug tracking system facilitates a bidirectional binding between the defect and any of its states from open to closed. This makes it a trivial task to move a project to its resolution state for verification. Similarly, the reason for the change, fixing a bug, is explicitly identified in the configuration management system. When used in conjunction with integration history, even changes in dependent, or shared IP projects can be identified as being the result of a specific bug.

In the following example a defect has been discovered in **Project A** occurring at $\mathbf{C_1}$ called $\mathbf{Bug_i}$:




The project is branched to the defect state a **Project A'** at $\mathbf{C_2}$. A fix is created and committed at $\mathbf{C_3}$. **Project A'** can now be sent to the verification team to verify the fix. When this is complete, the fix is then integrated back into the **Project A** branch at $\mathbf{C_4}$. Finally $\mathbf{Bug_i}$ is marked as being closed by $\mathbf{C_4}$.

Any user querying the bug tracker can easily see how $\mathbf{Bug_i}$ was resolved. A user querying the Configuration Management System can also see that $\mathbf{C_4}$ was in response to a $\mathbf{Bug_i}$ and the entire chain of action used to resolve it.

## Bug Management Using Unified Bug Tracking and Design Data Management Systems

This section will look at best practices work-flow which incorporates a bug tracking system that is tightly integrated with a change-based design management and version control system.

1. The bug is submitted to the bug tracker, ideally defining the defect state.
2. The dispatcher validates and clarifies the bug, and creates an environment in which the defect can be observed. The dispatcher's task is complete when a defect binding is created to the design management change ID along with instructions on how to demonstrate the defect. This is an unambiguous description of the defect
   - If the originator defined the design state when the bug occurred the dispatcher simply tells the design management system to create a workspace at that state and then runs a test to validate it.

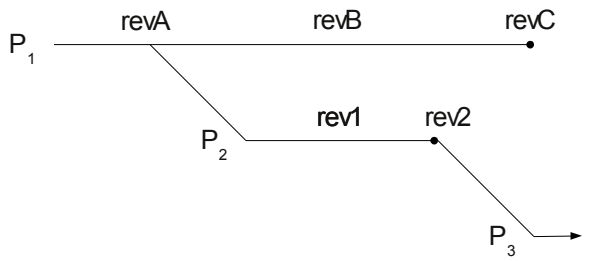
- If the originator only vaguely defines the bug state, the dispatcher may need to move a workspace through several states to locate defect point.

3. The dispatcher sends the bound defect to the handler.
4. The handler creates a resolution branch from the bug state using the dispatcher's information in the bug. A branch is needed because resolving the defect is going to require design changes.
5. The handler fixes the bug, and binds it to the resolution state ID (the change state of the design system corresponding to the fix). The defect is passed back to the verification engineer (or originator) for confirmation.
6. The verification engineer uses the defect binding to create the workspace (a user's projection of specific change state) with the resolution state ID. The engineer will confirm that the defect was fixed, and run any applicable regressions against in the workspace to make sure that the fix did not introduce any new problems. If there are problems, the verification engineer will annotate them in the defect and passed it back to the handler. The verification team closes the defect after all tests have passed.
7. Throughout the bug management process design team members need to have real-time alerts regarding bugs that may impact them. The capability of always having the most up to date information about a defect's status will allow the team to deal with it effectively.

## Bug Containment and IP Reuse

The clearing of bugs in a design is just the tip of the iceberg in the process of defect containment. A bug may appear in a design which is a component shared throughout a larger tree of derivative designs. Resolving bugs starts the process of propagating the fix to all designs which it may affect.

The following is an example of a derivation tree of a design component:




Three projects are related by derivation (**$P_1$**, **$P_2$**, **$P_3$**). Assume that a defect was discovered and resolved in **rev1** of **$P_2$**. The first task in defect containment is to determine which designs are at risk. The design data management system is queried to discover that out of all the projects under management, only **$P_1$**, **$P_2$**, **$P_3$** need to be considered. A further evaluation of the dependencies reveals that only **rev2** of **$P_2$** and the active development state of **$P_3$** require further action. For each of these a derived defect is created and linked to the originating defect, for further investigation.

If the initial bug resolution is believed to be sufficient to fix the problem in the derivative designs, the design management system should be capable of propagating the fix to the derivative design points for verification and ultimately for clearance by binding the design resolution state to the bug.

## Conclusion

Companies that utilize a unified design data management and bug tracking methodology will reduce the effort required to resolve and contain defects over their design set. By binding the bug state to the design state, bug tracking and management become a complete entity. By linking these two systems together, the value of each system to the design team is increased.

The ability for teams to explicitly define and reproduce the bug and resolution states will improve communication about the design, as well as providing design team members with real-time notices regarding bugs that may impact them. This is especially true for designs which span a large organization or geographic space.

***Roger March*** is Chief Technology Officer for IC Manage, Inc., where he created their current design data management solution and continues to improve its performance and scalability for global multi-site use. He has a BSEE from San Jose State University.