DESIGNING CHANGE REQUEST WORKFLOWS

By Eric Mariacher

1 ABSTRACT

"Modern" software development companies use a change management tool. There are two kinds of possible change requests in a software: bug correction and feature implementation. Most of change management tools offer the possibility to implement a change request workflow to get visibility on the current status of the change request. In this article, simple to complex bug or feature development worflows will be built, based on answers given to a short list of questions and concerns.

2 FIRST QUESTIONS

The 1st step in designing a change request workflow is to answer some questions:

- **Is the change request workflow a "bug" workflow or "feature development" workflow?**
  - Different processes are involved when handling bugs and feature requests even if the very high level view is similar.
- **Who are the stakeholders going to use this workflow?**
  - All software development stakeholders must be able to "use" this change request workflow with limited overhead.
- **What is the average and maximum level of complexity of the software projects we are going to handle through this change request workflow?**
  - It's always possible to design a perfect change request workflow, but covering 100% of possible scenarios may induce (huge) overhead. On the other hand having enough visibility on the project is vital. When the change request management tool is used for several projects, a balance has to be found between having precise visibility and minimum use overhead.

3 DESIGNING A SIMPLE BUG WORKFLOW

The 2nd step is to design the workflow (let's choose a bug workflow). This workflow meant for project health visibility must help to answer some questions and be as simple as possible.

- **What is the status of that particular bug?**
  - Though there can be a lot of statuses. They can be summarized into two main status "under control" vs "not under control".
- **Who is currently responsible for it?**
  - Typically engineering/developers vs QA (quality assurance)

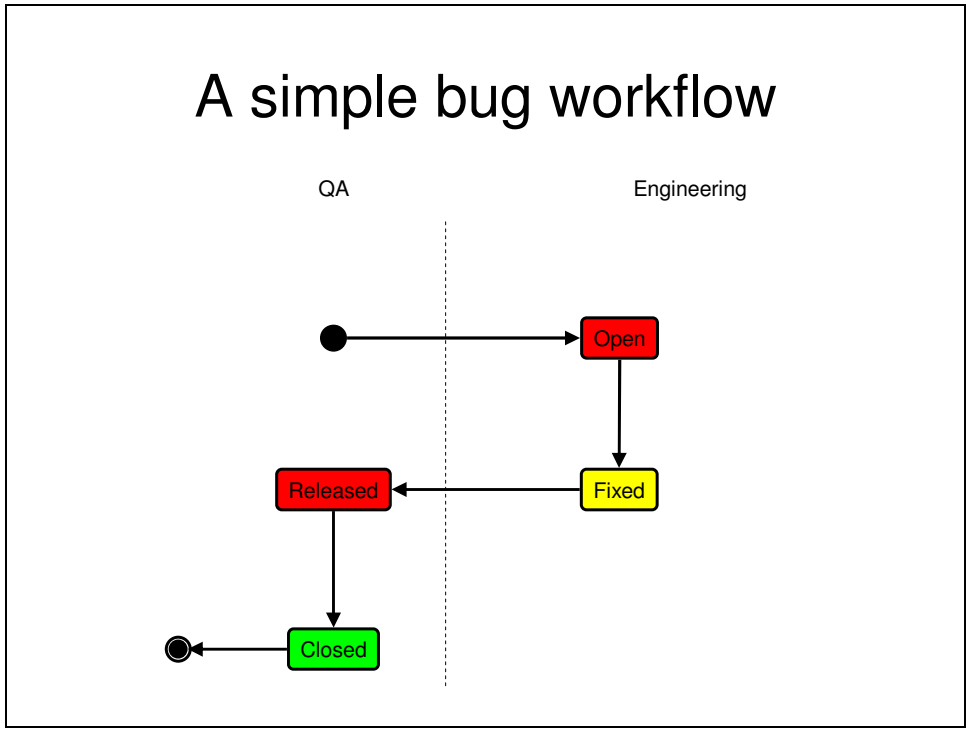
Here is a table displaying what "under control" vs "not under control" means for engineering/developers and QA

	not under control	**under control**
Engineering/developer	bug not corrected	bug corrected
QA	• bug found • bug corrected by engineering but not yet tested by QA	bug correction tested successfully

Some status names can be associated with cells from previous table:

	not under control	**under control**
Engineering/developer	***open***	***fixed***
QA	• ***open*** • ***released*** (to QA by engineering)	***Closed*** (no problem any more)

A good and simple workflow should make these 4 statuses visible:



Colors in this figure show the level of “risk” associated with bug statuses:

- **Red**: change under control
- **Yellow**: change not under control
- **Green**: change process closed

During my software development years, on a day to day basis, this bug workflow has been successfully used by engineering and QA teams. Bug current ownership/responsibility is clear:

- Bug open and fixed is owned by engineering.
- Bug released (and closed) is owned by QA.

One comment I often get when showing this workflow is why to distinguish between fixed and released status? The answer is that communication between engineering and QA is often done by delivering “software releases” . These “software releases” may not always contains all current bug fixes for various reasons. Another reason is that it is good information to know that a bug has been corrected by engineering (under control) but not yet delivered to QA. Distinguishing fixed and release statuses enables QA to only test what has been delivered to them.

4 WHEN A COMPLEX BUG WORKFLOW IS NEEDED

Sometimes a project is or becomes “hot” and knowing that a bug is open or released is not enough.

4.1 Details on open bug state

An open bug can have the following statuses:

- Open but not enough information yet to be assigned to development
  - Open status can be splitted into “evaluate (by QA)” and “open (to developer)”.
- Open and reproducible
- Open, seen once but not reproducible as of now
- Open but won’t be fixed (for various reasons such as priority, known limitations, etc…)
- Open to a 3rd party (cannot be fixed by in-house developers)
- Open and root cause identified but fix not yet identified
- Open and root cause + fix identified
- Open and root cause + fix identified but inacceptable potential side effects will occur if fixed
- …

For each of these statuses it may be relevant to create some specific statuses in the bug flow. It may be wisier to use

Change request attributes instead to discriminate between those bug states not to overload the bug workflow.

Open status to be splitted into “evaluate (by QA)” and “open (to developer)” makes sense because bug ownership belongs to 2 different groups.

4.2 Details on released bug state

A released bug can have the following statuses:

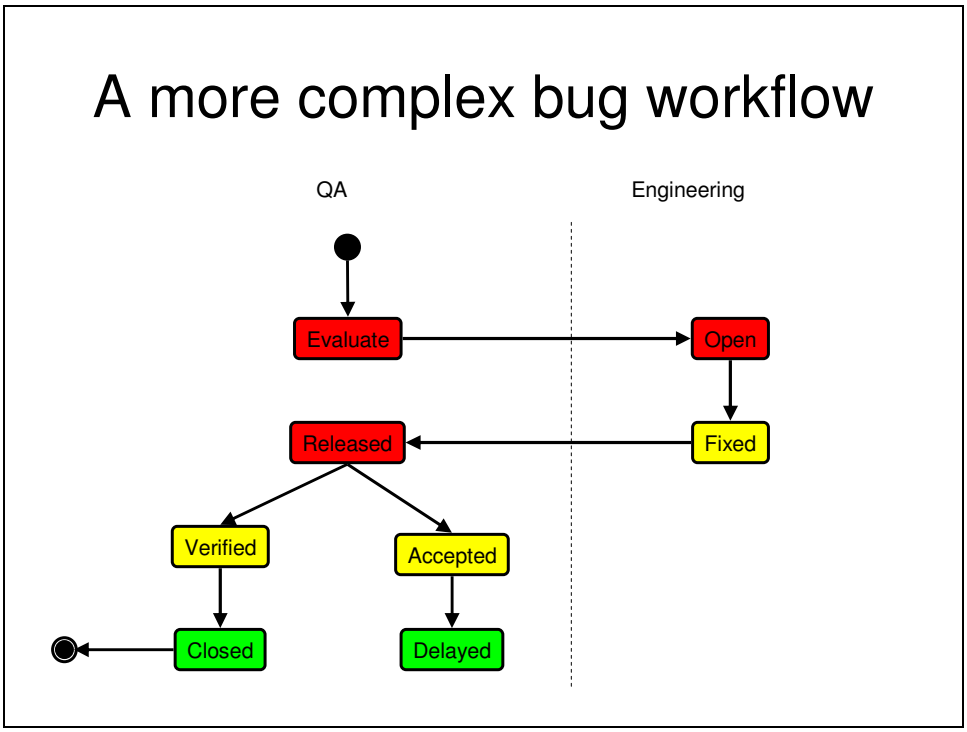
- Released but not yet tested
- Released, tested successfully on an interim release but not tested on the final “gold” release
- Released, did not pass test but accepted by QA
- …

4.3 Other possible details

A closed bug may still be open for this release but accepted by QA. Its resolution is postponed to a further release.

4.4 A more complex bug workflow

Here is what a more complex workflow could look like:



5 DESIGNING A FEATURE DEVELOPMENT WORKFLOW

The 1st step in designing a change request workflow is asking the following questions:

- **Is the change request workflow a "bug" workflow or "feature development" workflow?**
  - It's feature development workflow.
- **Who are the stakeholders going to use this workflow?**
  - Project Management/Marketing
  - Engineering/Developers
  - QA
- **What is the average and maximum level of complexity of the software projects we are going to handle through this change request workflow?**
  - Don't know yet.

The 2nd step is to design the feature development workflow. This workflow meant for project health visibility must help to answer some questions and be as simple as possible.

- **What is the status of that particular feature request?**
  - Though there can be a lot of statuses. They can be summarized into two main status "under control" vs "not under control" like bugs.
- **Who is currently responsible for it?**

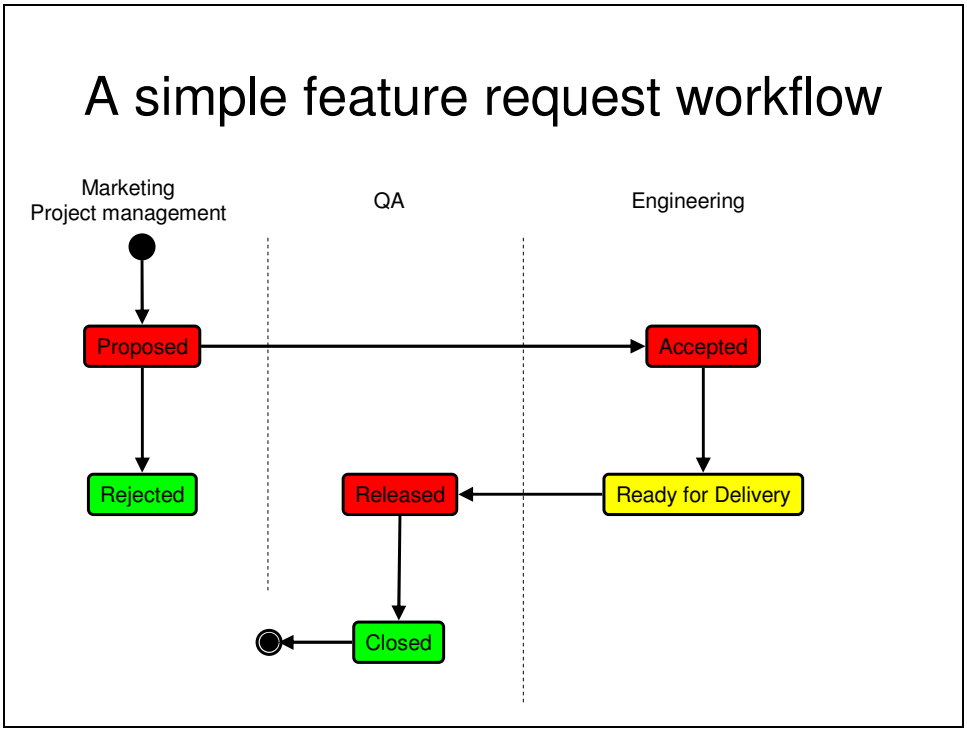
Here is a table displaying what "under control" vs "not under control" means for different teams

	not under control	under control
Marketing /Project Management	Features list not yet decided	Features list finalized
Engineering/developer	Feature not yet developed	Feature implemented
QA	Feature not yet tested	Feature tested

Some status names can be associated with cells from previous table:

	not under control	under control
Marketing /Project Management	***Proposed***	***Accepted***
Engineering/developer	***Accepted***	***Ready for Delivery***
QA	***Released***	***Closed*** (tested according to requirement)

A good and simple workflow should make these statuses visible:



6 CONCLUSION

As a general rule, a lot of stakeholders and high project visibility will trigger the need for more complex change request workflow associated with more administrative overhead.

Let's keep humble when designing a change request workflow.

Let's not forget that it must be usable by engineers and QA that will use this process on a day to day basis.

Let's not forget project managers who need to give visibility on the software project to their management.

7 ABOUT THE AUTHOR

Eric Mariacher has held various positions in embedded software development, coder, architect, project manager at IBM and now functional manager for wireless keyboards, mice and receivers at Logitech. These projects are of different sizes beginning with 8Kbytes of code running in 8bits microcontrollers coded by one programmer to projects requiring tens of developers, several megabytes of code and thousands of files on 32bits microcontrollers and PC computers.

Eric Mariacher main centers of interest are to set-up the right environment for software developers to deliver products, and also to give management the best visibility on software development process from requirement phase to field support phase. Setting up the right software development often requires reengineering software development processes. Eric Mariacher enjoys driving these software development processes reengineering measured following the CMMI best practice.

Eric Mariacher can be reached at eric.mariacher@gmail.com or eric_mariacher@logitech.com .

