A transaction is a business activity that is usually initiated through a request and includes a change of state of a data or an application. This may not be confused with the low-level transaction model of a database. Some examples of transactions in banking are changing of address, closing an account, opening an account etc. Similar examples can be found in different sectors like marketing, sales etc.

A transaction is initiated/created by a user/employee with a maker role. It can include necessary fields/properties that provide all information pertaining to the transaction. Some common properties can be

1. Type of transaction
2. Date of transaction
3. State of a a transaction
4. Maker user ID

Every transaction goes through a pipeline of steps before it is completed. The pipeline should include an approval process where the transaction is reviewed by a group of users with reviewer role. The number of reviewers required depend upon the type of transaction. For example, closing a bank account might need two reviewers while change of address needs single reviewer. Some additional properties to support the review process include

1. Reviewer 1 user ID
2. Reviewer 1 Vote
3. Reviewer 1 Rejection Reason
4. Reviewer 2 user ID
5. Reviewer 2 Vote
6. Reviewer 2 Rejection Reason

Each of the user or employee who is qualified to be a maker or reviewer is configured with a queue or an inbox. The maker’s inbox contains transactions that are created by him/her along with the status. The maker creates the transaction, adds necessary fields and submits for processing. If the transaction type qualifies for maker-checker process, it will automatically be moved to the inboxes of the corresponding reviewers. The reviewer’s inbox contains transactions that are pending for his/her review. If the transaction is approved, it will be added to the processing queue for further steps of completing the transaction. If the reviewer needs more information for checking, it can be moved to the maker’s inbox to reiterate the process. If the reviewer rejects the transaction, he/she can add the reason and move to the rejection queue for further steps of terminating the transaction. A simple flow engine for a transaction going through review process is depicted below



This model allows for automatic collaboration between makers and reviewers for processing a transaction. It is also possible to automate the selection of reviewers based on the type of transaction and roles of the users using pre-configured mappings of who can review what type of transactions. Throughout this process, the properties of transaction are updated to facilitate the status checks by any users.

The review process is basically a process workflow that can be created using the maker’s and reviewer’s tasks as steps in the workflow. There are several tools available in the market that can be used to setup this workflow. Enterprise tools like Microsoft Flow, Sharepoint workflow, Salesforce workflow or open source engines likes WexFlow, Activiti or it can be easily implemented using simple if and loops of any programming language.

**Class design**

There are three classes as shown below



A user can have list of roles like maker, checker. Depending on the role of the user, he/she is authorized to perform the corresponding action. A user who has maker role will be able to create a transaction and initiate the approval workflow. The transaction will be stamped with the maker ID accordingly. The workflow will select a user who has checker role and inserts the transaction in the user’s inbox. The workflow must ensure that the maker of the transaction cannot check the same when selecting the reviewer. If the transaction is approved, the approval workflow can move the transaction to the next workflow in the pipeline. If the reviewer requests for more information, the workflow will identify the maker and move the transaction to the maker’s inbox. If the transaction is rejected, the workflow is terminated by stamping necessary details in the transaction. At any point, the status of the transaction can be determined by querying the transaction object.

2. Sonar Cube

The objective is to use a tool/software that ensures the sanity of the code that is checked in the source control with respect to the following aspects

1. Logical checks
2. Coding styles and conventions pertaining to the team
3. Security vulnerabilities
4. Regression checks
5. Test coverages

To automate these checks as part of source control commits, the best way is to have a solid build system for the projects that includes all the above checks as part of building the projects. The corresponding violations can be flagged either as warnings or errors based on the business justifications. When a code change is submitted for review, builds are automatically run, and the corresponding violations will be added as comments to the pull request or the code review. The code change will be prevented from merging until the build is completed successfully without any warnings/errors. Several build systems like TFS, Jenkins facilitate configuring complex pre and post build tasks that can be used to run tools that check for the sanity of the code. The following are some of the tools/approaches that can be used for checking the code

**Logical checks**

The automatic technique of scanning each line of code or compiled binaries for any gaps without executing the code is referred to as static code analysis. Some tools like Coverity, SonarCube etc. analyze the source code while tools like FxCop does the analysis on the managed code assemblies. In either case, these tools scans for common errors like null reference/pointer, memory leak/corruption, buffer overflows, error handling, incorrect usage of APIs or obsoleted APIs etc. The tools can be easily integrated with the IDEs like Visual studio, eclipse so that they can be caught at the early stage of developments even before the code reviews.

**Coding styles/practices**

Coding standards are guidelines and conventions followed in a team to improve the overall quality and readability of the source code. These standards are different for different teams and the same static code analysis tools can be used to ensure the standards. The build system can include code styling linters like StyleCop to catch the styling issues. As mentioned above these can be hooked in the IDE’s to highlight the issues while writing the code.

**Security checks**

Common vulnerabilities exploits in an application are Cross site scripting, hard-coded credentials/keys , SQL injection etc. Several in-market vulnerability scanners are available as static code analysis tools that can be run as part of the build system. Tools like credential scanners can be configured to run before pushing commits to prevent the exposure of the credentials even before submitting the code for review.

**Regression checks and Test coverage**

As part of post build task, tests can be automatically run to catch for potential regressions from the code change. These tests can be grouped as unit or integration to identify the impact of the regression. In addition to regression checks, code coverage tools can be run to mandate the desirable coverage of tests for any change in code submitted for review.