Project Requirements 4.1 and 4.2
Designing an ideal Workflow tool

For building an extensive, useful workflow tool, there are several factors that we take into consideration: usability and adaptation, onboarding ease, workflow development capabilities, and overall tool aesthetics. Instead of starting from scratch, we've done an analysis of popular workflow development tools, including Taverna, Kepler, and Vistrails with respect to the qualities listed and will use these findings to build requirements for our tool. A few key findings are as follows:

1. Taverna is fairly easy to use, provides the necessary development capabilities with the ability to define beanshell scripts, and has a clean UI. However, getting started with Taverna is not as straight-forward as users have to install several dependencies for its configuration. Additionally, Taverna lacks customization in workflow visibility as the components are automatically placed in a linear fashion.

2. Although it is fairly easy to setup Kepler, it lacks core user experience qualities and requires the additional understanding of "Director" components making it not as straight-forward to build workflows. It is also not as simply to define customized scripts for components.

3. Vistrails has a customizable, drag and drop UI which is convenient for the user. It also allows for definition of python scripts to extend workflow capabilities. Vistrails is fairly easy to setup and use. In addition, it uses color signals to show the progression of workflow execution.

4. All of the tools require native downloads rather than being SaaS.

Given these observations we are deciding to design a workflow tool which uses Vistrails as its base model and adds upon it by pulling out some of the other positive capabilities from the remaining workflows and other additions.

The usability requirements for our workflow tool are as follows:

1. Customizable workflow views through drag and drop capabilities.
2. The ability for users to visualize the execution of each component in the workflow, in real-time, when running a workflow.
3. The ability to drag and drop workflow components onto the canvas to build the workflow.

Workflow requirements from a developer perspective:

1. A developer should be able to define custom scripts with the option to choose from a diverse set of programming languages, including python source and Java Beanshell.
    a. We can apply technology like CGI(Common Gateway Interface) to implement this requirement. We use environment variables to pass inputs, and use stdout to get the output of the subroutine. So that our workflow tool can support any language. The user scripts need to read environment variables as it's input and print the result to stdout/error message to stderr.
2. The ability to onboard with ease through a SaaS model where developers are able to access the capabilities of the workflow tool online, without downloading packagings locally.
    a. In order to achieve this, we would build our workflow tool locally and deploy its components on an AWS EC2 instance, making it accessible via the web.
3. A Rest module that allows users to visualize their API calls and the ability to funnel the response body into another API call or a custom script as described in requirement 1.
    a. To implement this functionality, we would build an API interface module which would be draggable onto the main canvas. Once this module is put on the main canvas, a dialogue box is opened allowing the user to include the API endpoint and expected response body. The necessary parameters should be appended to the url. When the workflow is then ran, the tool will reach this api module, send an http request to the given endpoint, then retrieve the response body to propagate through next connected component.
4. In the case of complex workflows and development teams, the tool should offer a real-time collaboration feature allowing users to build a comprehensive workflow composed of sub-flows.
    a. To implement this feature, we would deploy the workflow tool onto a cloud platform such as AWS, so that each team member can access the tool online at the same time. We would also deploy a cloud based database to store the workflow information and provenance. To ensure the consistency of data, all writes would be atomic and reads would be locked while someone is writing. Reads will read from the most recent write according to the timestamp assigned to each transaction. If there are a lot of users,

we can also scale the workflow application by increasing the number of deployed instances and expand our database through sharding.

5. Allow users to visual workflow output in forms other than just the console. For example, provide options to visualize output in a graph or table.
    a. In order to do this, our workflow tool would require the inclusion of a standard output module to complete the workflow. We will create this standard output module which should be draggable to the main canvas. Once this output has been dragged to the main canvas, a dialogue will open prompting the user to choose their desired output visualization (console, graph, or table). If console is selected, the workflow will simply execute and print to the console. If graph is selected, the output's value must be in json form with an array of nodes and an array of corresponding links. The structure of a node would be Node: { name: x, id: y} and the structure of a link would be Link: {source: y, target: y2}. In the backend of our application, the tool will take this input and surface a graph to the user. If table is selected, the standard output data should similarly be in a structured form as follows: {rowtitles: [header1, header2], rows: [{val1: val, val2: val}].