Project Ideas

Potential Jenkins Projects

Your team has several options to choose a project from:

1. Ideas obtained from research groups at UIUC who use Jenkins
2. Develop plugins, more likely contributing to an existing plugin (you do not have to submit pull requests) than Starting a new plugin
3. Feature requests on the Jenkins issue tracking page (search by New Feature on the panel to the left of that page)
4. Ideas submitted by current CS 427 students
5. Other ideas that you may have

Ideas obtained from research groups at UIUC who use Jenkins

Please click on any of the links below to take you directly to the project descriptions.

<table>
<thead>
<tr>
<th>Project 1</th>
<th>Project 6</th>
<th>Project 11</th>
<th>Project 16</th>
<th>Project 20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project 2</td>
<td>Project 7</td>
<td>Project 12</td>
<td>Project 17</td>
<td>Project 21</td>
</tr>
<tr>
<td>Project 3</td>
<td>Project 8</td>
<td>Project 13</td>
<td>Project 18</td>
<td>Project 22</td>
</tr>
<tr>
<td>Project 4</td>
<td>Project 9</td>
<td>Project 14</td>
<td>Project 19</td>
<td>Project 23</td>
</tr>
<tr>
<td>Project 5</td>
<td>Project 10</td>
<td>Project 15</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1. **Project 1**: Making Jenkins automatically find all versions of its dependent software. During some MP's, some students ran into problems due to the fact that certain software that Jenkins depends on (JDK, ANT, etc.) had multiple versions on the host computer and Jenkins did not know which one(s) to choose. The idea in this project is to extend Jenkins to automatically find all versions of software on its host machine, and present these to the user to select from.

2. **Project 2**: Extend Jenkins to create build reports incrementally. The default behaviour in Jenkins is to produce build reports only at the end of a build. The goal here is to produce the reports as information is available, so that if a user stops a build midway, the reports will already contain the results so far.

3. **Project 3**: Make Jenkins to package files in a way that preserves the file permissions after a build. Currently, when Jenkins packages directories as part of building a job, it does not preserve the read-write-execute bits on those files. This project aims to solve this problem.

4. **Project 4**: A Jenkins plugin to monitor and limit the amount of server resources that Jenkins uses. Since Jenkins may not be the only critical software running on a server, it is desirable to monitor the amount of computational resources that Jenkins consumes. A Jenkins plugin that solves this problem should also let users configure an upper bound for the amount of resources that Jenkins runs on.

5. **Project 5**: When projects have dependencies among themselves, Jenkins should allow for a tree view for seeing the status of all the projects in the hierarchy. A software system may be composed from several projects that depend on themselves in some way. Each of these will also usually have their own tests, which Jenkins may be required to run in some order. This project is aimed at providing a tree-like or graph-like view of these projects. It should be possible to use this view while Jenkins is running (e.g., to know which project's tests are currently being run) and when Jenkins is not running (e.g., to view the build state of the entire system when there are failures).

6. **Project 6**: When Jenkins is reporting test results for multiple projects, merge the reports together. The current configuration is to put test results in different folders. The testers then have to go into each folder to view the reports on a project-by-project basis. The aim of this project is to remedy this by merging the reports from multiple projects and presenting this in a more human readable form.

7. **Project 7**: Make Jenkins report whether there are newly passing tests or newly failing tests based on historical information. On every run, Jenkins will only report on all the tests that pass and/or those that fail. The goal of this project is to use information from previous runs so that Jenkins can also report on which tests were passing before, but are now failing (or vice-versa).

8. **Project 8**: Report tests results in a different format, especially for projects that are not written in Java/JUnit. By default, Jenkins reports test results in a format similar to that used in JUnit. The goal here is to consider how to report on tests for projects which use Jenkins, but which have not been written in Java. For example, projects written in C++ or Python may need a different way of viewing test results.

9. **Project 9**: Make Jenkins backup and manage the history of configuration files and present the information in these files in a more human readable way. Jenkins users may write configuration files in order to tell Jenkins how to build/run their projects. Jenkins doesn't currently backup such configuration files. Neither does it keep a history of changes made to such files. The goal of this project is to address these limitations, and also present the configuration files, and their evolution, in a more human-readable form than the current (XML-like) form.

10. **Project 10**: Implement a rich text editor for typing in shell commands. Typing shell commands in the Jenkins default interface consists only of typing text into a plain (HTML) textarea. This project will entail the development of a rich text editor that can better assist Jenkins users whenever they enter shell commands in Jenkins.
11. **Project 11:** Making Jenkins to produce more detailed test run times when running tests in parallel. Currently, after Jenkins runs many tests in parallel, it will only output the total (CPU) time taken by all the tests. The goal of this project is to produce finer grained information from running many tests in parallel. For example, it should show, in addition to the total time, the test(s) which took the longest time to run.

12. **Project 12:** Extend TestInProgress plugin to communicate with Jenkins slaves in real time. One way of running tests in parallel is to configure Jenkins to run on a Master instance which can spawn multiple slave instances for running the tests. In this project, you will be extending TIP, or similar, to monitor tests on both the master and all its slaves, and present real time reports on the progress of each.

13. **Project 13:** Alternate ways of computing progress in TestInProgress (TIP) plugin. This project will investigate how TIP currently computes the progress of tests that are currently being run, and seek to add other (possibly better) ways of computing progress. For example, progress may be computed based on number of tests already run versus total number of tests available. It may also be computed based on the amount of time used so far versus the amount of time used for running all the tests in previous runs. Finally, this should also work for cases where the tests are being run in parallel.

14. **Project 14:** Dealing with “data races” when tests are being run in parallel. A number of tests may depend on the same data in order to execute. When all the tests are run sequentially, this is not a problem. However, when Jenkins is configured to run tests in parallel, some form of data race may result when tests have to wait for each other to access some common resource. The goal of this project is to come up with a better way to deal with such scenarios.

15. **Project 15:** Extend TestInProgress (TIP) to report as soon as a test fails. Currently, the only ways of finding out about test failures is either when the Jenkins build job completes or by watching the console of the build job. Thus, Jenkins users spend/waste time to find out about tests which may have failed early in the build process. The goal here is to help save developer’s time. There are three parts to this: extending TIP to (a) report to the user as soon as a test fails (b) give the user an option to stop the build process immediately or continue to wait on the build process (c) modify the left menu on the default build job console page to dynamically populate with links to reports about failing tests as soon as any tests fail.

16. **Project 16:** Making Jenkins to produce produce “diffs” of a project between successive runs. After running a Jenkins build job, the resulting reports have to be compared offline with that from previous runs. Can we make Jenkins produce these diffs?

17. **Project 17:** To understand branches of a project. Imagine a project with many branches, created at different times in the past. Currently, managing this in Jenkins is not a very straightforward process. Jenkins users usually have to duplicate each branch as a separate project in Jenkins. This project aims to fix this problem by integrating knowledge of branching structure into Jenkins so that Jenkins can automatically manage them. It should also be possible to mark branches as active/inactive.

18. **Project 18:** Better parallelization of tests in Jenkins. Sometimes, tests have to be run on a finite amount number limited resources. For example, each test may need to be run on its own dedicated processor core, and will therefore need to compete with other tests for this. This can lead to races among such tests. What is needed is to build better test parallelization algorithms into Jenkins and that is the aim of this project.

19. **Project 19:** Better reporting of intermittent failures in Jenkins. When building a large-scale software in Jenkins, test may sometimes fail as a result of external factors such as hardware failures and not some error that the developer made. In order words, the tests may be flaky. The idea in this project is to make Jenkins create better reports such that developers may see, at a single glance, whether their tests are failing due to their own mistakes or due to intermittent failures of external dependencies.

20. **Project 20:** Extend TestInProgress (TIP) to report as soon as a test fails. Currently, the only ways of finding out about test failures is either when the Jenkins build job completes or by watching the console of the build job. Thus, Jenkins users spend/waste time to find out about tests which may have failed early in the build process. The goal here is to help save developer’s time. There are three parts to this: extending TIP to (a) report to the user as soon as a test fails (b) give the user an option to stop the build process immediately or continue to wait on the build process (c) modify the left menu on the default build job console page to dynamically populate with links to reports about failing tests as soon as any tests fail.

21. **Project 21:** Better reporting of intermittent failures in Jenkins. When building a large-scale software in Jenkins, test may sometimes fail as a result of external factors such as hardware failures and not some error that the developer made. In order words, the tests may be flaky. The idea in this project is to make Jenkins create better reports such that developers may see, at a single glance, whether their tests are failing due to their own mistakes or due to intermittent failures of external dependencies.

22. **Project 22:** Better parallelization of tests in Jenkins. Sometimes, tests have to be run on a finite amount number limited resources. For example, each test may need to be run on its own dedicated processor core, and will therefore need to compete with other tests for this. This can lead to races among such tests. What is needed is to build better test parallelization algorithms into Jenkins and that is the aim of this project.

23. **Project 23:** Use stubs and mocks to test a project. In order words, the tests may be flaky. The idea in this project is to make Jenkins create better reports such that developers may see, at a single glance, whether their tests are failing due to their own mistakes or due to intermittent failures of external dependencies.

---

**Jenkins Projects suggested by current CS 427 students during MP2**

These unfiltered projects ideas were suggested by students in this class. Before choosing any of these, we strongly recommend that you first check that they are NOT proposing things that have already been implemented in the Jenkins online community.

1. Extend the testInProgress-plugin to work with Jenkins CLI. Jenkins UI's performance is not that good and so CLI is preferred by some users. We are not sure if the testInProgress-plugin supports this feature. If not, then this is a potential enhancement.

2. Add mocktupp is growingly popular unit testing for C++. We can develop Jenkins plugin that supports mocktupp unit tests. With the resource and time constraints, we may focus on a plugin with basic support.

3. Better Jenkins UI (Back button, Better JDK selection, Better error output, Automatic screen displaying errors in console output, Cleaner list of previous builds)

4. Rejection of code changes which cause failures in the tests. Possibly include a threshold, e.g. accept only if 90% of unit tests pass.

5. Allow Jenkins to “watch” a directory and build everything under it.

6. Have Jenkins monitor irregular build times based on statistics, e.g. builds that take too long, and notify developer before the build is even done, such that developers might be able to just stop the build and fix the code if it is indeed a code error.

7. Have a chart showing what tests passed on which builds.

8. Have duration statistics and its visualization for each Junit tests in each build.

9. Pick UI updates to implement from https://wiki.jenkins-ci.org/display/JENKINS/UI+Enhancements
10. Have Jenkins automatically commit code only if a "pre-commit" repository passes the build and/or tests. [https://issues.jenkins-ci.org/browse/JENKINS-1682](https://issues.jenkins-ci.org/browse/JENKINS-1682)
11. Show better maven build error messages. Jenkins doesn't show all the output from Maven. Would be good to include maven path and java path.
12. Have a chatroom/IM feature in Jenkins so people in the team can use it
13. Allow editing files in Jenkins
14. Show more useful info, such as how to fix a failed build
15. Show detailed output report for unit tests, including console output, which tests were run, which passed, which failed, and include code coverage.
16. Finer grain info in the build history box while building, e.g. show when it's compiling, when it's testing, how many tests are scheduled to run
17. The test result trend graph in the project homepage should include a legend and label for X and Y axes.
18. Better Jenkins configuration screen. Each tool (java, mvn, etc) should have 3 choices: (1) system default (2) manually hardcoded version (3) "local" version only used by Jenkins
19. Have Jenkins check compatibility between selected JRE and version required for the maven project and display error as soon as this is known 21) Use AJAX in the web UI when possible.
20. "Blame" who broke a build. Finer grained email options, such as emailing everyone when a build fails after a long time of build succeeds, or only email relevant developers if a build is fixed after a break. Or email based on importance of project.
21. Have Jenkins be more aware of system notifications such as out of disk space on linux machines
22. Jenkins mobile app (or mobile web version) that shows less info but still useful
23. Allow for job priorities, so jobs with higher priorities get built first. Automatically infer priorities based on file changes
24. Have Jenkins look for licensing information in source code. Fail the test (maybe with a different color) if license information is omitted.
25. Code profiling feature
26. Show trend of time to run tests (not including build time)
27. GUI for starting up Jenkins instead of java -jar jenkins.war. It should give more debugging information.
28. Allow svn functionality (tagging, committing, reverting versions, etc) directly in Jenkins
29. TestInProgress improvement. Showing code alongside the test results.
30. Have plugins for windows phones. (Currently only has iOS and Android.)
31. Notify users for missing dependency errors
32. Use buttons instead of links
33. Have a command line console to simplify configuration rather than doing it through the UI
34. Automatically create github issue including failed build log
35. Build projects in a "virtual container" by combining Jenkins with Docker
36. Allow Eclipse to use Jenkins, e.g. to monitor, build jobs
37. Provide Jenkins as a SAAS
38. Have the test in progress report and console output be in one page
39. Use Jenkins in conjunction with Phabricator ([https://phabricator.org/](https://phabricator.org/)). e.g. extend the github plugin to allow commenting on github commits
40. A page that shows a diff between revisions of build
41. Interactive Jenkins tutorial
42. Notify build systuses on different social media platforms, e.g. facebook groups, twitter, groupme. Also allow the platforms to command jenkins builds, e.g. tweet "@jenkins build job1"
43. Create an iOS monkey test runner plugin for Jenkins.
44. Create plugin to support haskell.oaml other languages
45. Allow building eclipse projects
46. Infer proper build commands/configurations based on source code
47. Better statistics, e.g. total number of daily, weekly, monthly builds by group as well as individual member
48. Chrome add-on to monitor Jenkins projects
49. Exchange calendar plugin that publishes build records on an exchange calendar
50. A Jenkins plugin that allows for the use of GNU Bazaar as a SCM
51. Coloring the console output for warnings, errors, etc.
52. Export build/test report(s) into JSON/XML/PDF/HTML
53. Text message/phone notification
54. Linter integration
55. Visualize build/test statistics
56. Schedule builds at specific time of day
57. Visualize data structures used by tests
58. Improve test summary page. Something that shows a matrix of the tests over time like the "Continuous Integration Display" that was shown in Lecture 7.
59. Exclude tests that are not affected by changes in new commits
60. Add project management functionality, e.g. assigning tasks, giving due dates, sending reminders for due dates
61. Add auto online documentation for successful builds, auto create a webpage, like javadocs
62. Dynamically scale nodes on distributed builds
63. Build (a) specific revision(s)
64. Add functionality to identify outdated dependencies and suggest updates in pom files
65. Use distributed file system or multiple servers or SVN instead of local disk for jod data, build history, etc
66. Show how long test code took to run
67. Allow Jenkins to deploy code after a successful build
68. Better error messages. show call graph of failed tests.
69. Add a random input tester. Users config input parameters. Jenkins randomly chooses input and feed it to the tests
70. All grouping of jobs. support shared and individual configuration
71. Import/export configuration settings