Customizable Workflow Repository

Welcome to YouTrack Workflow Repository!

To use a ready workflow file in your YouTrack, please:

- Download a workflow zip file.
- If you are a project administrator, then import the workflow to your project in Administration > Projects > *project* > Workflow tab.
- If you have system administrator permissions, then import the workflow to your YouTrack in Administration -> Workflow page and attach it to one or multiple projects.

You can also try any of the custom workflows listed below in our YouTrack InCloud test instance. All registered users get project admin permissions.

Continue

- New to workflow in YouTrack? Please take a look at the workflow guide.
- Ready to create your own workflow rule? Use YouTrack Workflow Editor.
- Want to share your workflow? Contact us or create an issue.

Custom Workflow Examples

This table contains various use cases and programmed workflow rules ready to be used in your YouTrack.

Votes and Watchers (download zip file)	Source Code

Add the project lead to an issue watchers list once the issue collects a certain number of votes.	`rule watch if lots of votes`
	
	`when votes == 10 {`
	`  project.leader.watchIssue(issue);`
	`}`

Default Visibility Group (download zip file)	Source Code

Set the default visibility group for all new issues in a project.
 	
Requirements: group 'Reporters'
	`rule set permitted group for new issues`
	
	`when !isReported() && reporter.isInGroup("Reporters") {`
	`  permittedGroup = {developers};`
	`}`

User Comment(s) Awaiting Reply Notification (download zip file)	Source Code

If an issue has a recent non-developer comment, send a notification to a specific user (support team member). This check is performed at a given time (daily at 12pm by default).

Requirements: group 'developers-team'

```
schedule rule Notify on no comments during time period

daily at 12 : 00 : 00 [!issue.State.isResolved] {
  if (comments.last != null &&
!comments.last.author.isInGroup("developers-team")) {
    project.leader.notify("[Youtrack, Notifier] Issue " +
    getId() + " needs your attention",
    "Issue " + summary + " has a recent comment added by a
non-developer.");
  }
}
```

Reopen Duplicate (download zip file)	Source Code
Reopen an issue if all duplicate links are removed.	

```
rule Reopen issue on removing last duplicates link

when duplicates.changed && duplicates.isEmpty {
  State = {Open};
}
```

Clear 'Fix For Version' (download zip file)	Source Code
Clear 'Fix for version' field when an issue is reopened.	

```
rule Clear fix for version on issue reopen

when State.oldValue != null && State.oldValue.isResolved &&
!State.isResolved {
  Fix versions.clear;
}
```

'Unblocked' State (download zip file)	Source Code
Provide an ability to have the issue unblocked (denoted with a "is blocked by" - "blocks" relationship) when the last remaining blocker is resolved and the issue has no remaining blockers.	

Requirements: link types 'blocks' - 'is blocked by'

```
rule unblock on blockers resolving

when issue.State.isResolved &&
!issue.State.oldValue.isResolved {
  for each dependant in issue.blocks {
    if (dependant.is blocked by.first == dependant.is blocked
by.last) { //e.g. it's source issue
      dependant.State = {Unblocked};
    }
  }
}
```

Semi-Automatic 'Wait for Reply' State Management (download zip file)	Source Code
When a comment from an external user is required, internal user sets issue state to 'Wait for reply' manually. When a comment is added, issue is automatically reopened. Assignee gets a notification if no comment is added within 5 days.	Statemachine with 'Waiting for reply' state:

Requirements:

- state 'Wait for Reply'
- group 'developers'

```
statemachine States for field State {

  initial state Submitted {
    on wait[always] do {<define statements>} transit to Wait
for Reply
    ...
  }

  state Open {
    on wait[always] do {<define statements>} transit to Wait
for Reply
    ...
  }

  state Wait for Reply {
    on fix[always] do {<define statements>} transit to Fixed

    on open[always] do {<define statements>} transit to Open

    on in progress[always] do {<define statements>} transit
to In Progress

    ...
  }

}
```

Changes issue state from 'Wait for reply' to 'Open' when comment is added by an external user

```
rule Wait for reply: reopen on answer

when State == {Wait for Reply} && comments.added != null
&& comments.added.isNotEmpty &&
  comments.added.last != null &&
!comments.added.last.author.isInGroup("developers") {

  State = {Open};
}
```

Notify assignee if no answer is received within 5 days

```
schedule rule Wait for reply reminder

daily at 12 : 05 : 00 [issue.State == {Wait for Reply}] {
  var lastComment = comments.last;
  if (lastComment != null &&
lastComment.author.isInGroup("developers")
      && lastComment.created + 5 days < now) {
    if (Assignee != null) {
      Assignee.notify("[Youtrack, 'Wait for reply'
reminder] Unanswered comment within 5 days",
      "Hi, " + Assignee.fullName + "! <br><br> Issue <a
href=\"" + getUrl() + "\">" + getId()
      + " " + summary + "</a>" + " is in state \"Wait for
reply\" more than 5 days."
      + " It has unanswered comment:<br><b>" +
lastComment.text
      + "</b><br><br> Please answer or resolved
issue.<br><br>"
      + "<p style=\"color: gray;font-size:
12px;margin-top: 1em;border-top: 1px solid #D4D5D6\">"
      + "Sincerely yours, YouTrack" + "</p>");
    }
  }
}
```

Notify User Mentioned in Comment (download zip file)	Source Code
Send notification to a user mentioned in a comment.	

```
schedule rule send notification to a user mentioned in
comment

daily at 12 : 00 : 00 [<expression>] {
  var mentionedUser;
  for each comment in comments {
    if (comment.text.contains("remind", opts) &&
comment.created + 1 day > now) {
      mentionedUser = comment.text.substringBetween("remind
@", " ");
    }
  }

  for each user in permittedGroup.getUsers() {
    if (user.login.eq(mentionedUser, opts)) {
      user.notify("You were mentioned", "Your name was
mentioned in a comment in " + getId());
    }
  }
}
```

'Waiting for Reply' Tag Management (download zip file)	Source Code

Replace 'waiting for reply' tag with 'answered' when a developer adds a comment.

Requirements: group 'developers'

```
rule Set tag waiting for reply (answered)

when comments.added.isNotEmpty {
  if (!comments.added.last.author.isInGroup("developers")) {
    removeTag("waiting for reply");
    addTag("waiting for reply (answered?)");
  }
}
```

Last Submitted Date for Duplicate Exceptions (download zip file)	Source Code

Show the last submitted date for the main issue when a new 'is duplicated by' link is added to it.

Requirements: field 'Last duplicate' of 'date' type

```
rule Display last submitted date for duplicated exceptions

when issue.is duplicated by.added.isNotEmpty {
  var lastDuplicate = issue.Last duplicate;
  for each source in issue.is duplicated by.added {
    var addedDuplicate = source.created;
    if (lastDuplicate < addedDuplicate) {
      lastDuplicate = addedDuplicate;
      issue.Last duplicate = lastDuplicate;
    }
  }
}
```

Restrict Fields Changing (download zip file)	Source Code

Make some fields editable only by the issue assignee. Display an error message to any other user who attempts to modify the fields.

```
rule assert assignee can change field

when Priority.changed || Fix versions.changed || Fixed in
build.changed {
  assert loggedInUser == Assignee: "Only the Issue Assignee
can set this field";
}
```

Duplicate Issues Only With Same Visibility Groups (download zip file)	Source Code

Disable duplicating issues with different visibility groups.

```
rule Different duplicates issues visibility

when issue.duplicates.added.isNotEmpty {
  for each duplicate in issue.duplicates.added {
    assert permittedGroup == duplicate.permittedGroup:
       "Issues can only be duplicated if they have the same
visibility groups";
  }
}
```

'Open' Assigned Issues (download zip file)	Source Code

Change an issue state to Open when an assignee is added.	

```
rule Open issue on set Assignee

when Assignee != null && Assignee.changed &&
Assignee.oldValue == null {
  State = {Open};
}
```

QA Verification Process (download zip file)	Source Code

- Notifies a user when he/she is set to verify an issue
- Requires setting 'Verified in build' value when changing state to 'Verified', and sets current user name for 'Verified by' field value.
- Requires adding a comment when changing state to 'W/O verification'.

Requirements:
- field 'Verified by' of 'user' type
- field 'Verified in build' of 'build' type
- state 'W/O verification'

```
rule Notify on set Verify by user

when isReported() && Verified by.changed {
  if (Verified by != null && loggedInUser != Verified by) {
    Verified by.notify("[Youtrack, Verify by] You became the
'Verified by' user for issue "
    + issue.getId() + " " + issue.summary, "Hi, " + Verified
by.fullName +
    "! <br><br> You became the 'Verified by' user for issue
<a href=\"" + issue.getUrl()
    + "\">" + issue.getId() + "</a>" + " " + " <a href=\"" +
issue.getUrl() + "\">"
    + issue.summary + "</a>" + "<br> Please verify it or
close as \"W/O verification\".<br><br>"
    + "<p style=\"color: gray;font-size: 12px;margin-top:
1em;border-top: 1px solid #D4D5D6\">"
    + "Best regards, YouTrack" + "</p>");

  }
}
```

Set 'verified in build' value

```
rule Set Verified State on specifying Verified In Build

when Verified in build != null && State != {Verified} {
  State = {Verified};
}
```

Set 'verified by' user name when adding 'verified in build' value

```
rule Set verified by user on specifying Verified in build

when Verified in build != null && Verified in
build.changed {
  Verified by = loggedInUser;
}
```

States machine

```
...
state W/O verification {
  enter {
    Verified by = loggedInUser;
  }

  on reopen[always] do {<define statements>} transit to
Open
}
state Verified {
  on reopen[always] do {<define statements>} transit to
Open

  enter {
    Verified in build.required("Specify 'Verified in
build'");
    Verified by = loggedInUser;
  }

  exit {
    Verified in build = null;
  }
}
...
```

Require adding a new comment when changing an issue state

```
rule Require comment on w/o verification

when State.becomes({W/O verification}) {
  assert comments.added.isNotEmpty: "Add a comment about
the missing details";
}
```

Advanced States Machine (download zip file)	Source Code
Manage states transitions. **Requirements**: • field 'Verified by' user of type 'user' • field 'Verified in build' of type 'build' • value 'W/O verification in bundle 'State' • state 'Wait for Reply'	Lifecycle of the State field

```
statemachine States for field State {

  initial state Submitted {
    in 3 days[State == {Submitted}] do {
      var user = Assignee;

      if (user == null) {
        user = project.leader;
      }

      user.notify("[Youtrack, State reminder] Issue " +
issue.getId()
      + " is still Submitted: " + issue.summary, "Hi, " +
user.fullName
      + "! <br><br> Issue <a href=\"" + issue.getUrl() +
"\">" + issue.getId()
      + "</a>" + " is still in Submitted state:" + " <a
href=\"" + issue.getUrl()
      + "\">" + issue.summary + "</a>"
      + "<br> Please start working on it or specify a
more accurate state.<br><br>"
      + "<p style=\"color: gray;font-size:
12px;margin-top: 1em;border-top: 1px solid #D4D5D6\">"
      + "Sincerely yours, YouTrack" + "</p>");
```

```
    }

    on fix[always] do {<define statements>} transit to
Fixed

    on open[always] do {<define statements>} transit to
Open

    on in progress[always] do {<define statements>}
transit to In Progress

    on discuss[always] do {<define statements>} transit
to To be discussed

    on can't reproduce[always] do {<define statements>}
transit to Can't Reproduce

    on obsolete[always] do {<define statements>} transit
to Obsolete

    on duplicate[always] do {<define statements>} transit
to Duplicate

    on as designed[always] do {<define statements>}
transit to As designed

    on invalidate[always] do {<define statements>}
transit to Invalid
  }

  state Open {
    on in progress[always] do {<define statements>}
transit to In Progress

    on discuss[always] do {<define statements>} transit
to To be discussed

    on fix[always] do {<define statements>} transit to
Fixed

    on obsolete[always] do {<define statements>} transit
to Obsolete

    on duplicate[always] do {<define statements>} transit
to Duplicate

    on can't reproduce[always] do {<define statements>}
transit to Can't Reproduce

    on as designed[always] do {<define statements>}
transit to As designed

    on invalid[always] do {<define statements>} transit
to Invalid

    on wait[always] do {<define statements>} transit to
Wait for Reply
  }

  state Reopened {
    on open[always] do {<define statements>} transit to
Open
  }

  state Obsolete {
    on reopen[always] do {<define statements>} transit to
Open
  }

  state Duplicate {
```

```
    on reopen[always] do {<define statements>} transit to
Open
  }

  state In Progress {
    on reopen[always] do {<define statements>} transit to
Open

    on fix[always] do {<define statements>} transit to
Fixed

    on can't reproduce[always] do {<define statements>}
transit to Can't Reproduce

    on obsolete[always] do {<define statements>} transit
to Obsolete

    on as designed[always] do {<define statements>}
transit to As designed
  }

  state To be discussed {
    on in progress[always] do {<define statements>}
transit to In Progress

    on duplicate[always] do {<define statements>} transit
to Duplicate

    on obsolete[always] do {<define statements>} transit
to Obsolete
  }

  state Can't Reproduce {
    on reopen[always] do {<define statements>} transit to
Open
  }

  state As designed {
    on reopen[always] do {<define statements>} transit to
Open
  }

  state Won't fix {
    on reopen[always] do {<define statements>} transit to
Open
  }

  state Invalid {
    on reopen[always] do {<define statements>} transit to
Open
  }

  state Incomplete {
    on reopen[always] do {<define statements>} transit to
Open
  }

  state Fixed {
    on reopen[always] do {<define statements>} transit to
Open

    on verify[always] do {<define statements>} transit to
Verified

    on can't verify[always] do {<define statements>}
transit to W/O verification
  }

  state W/O verification {
    enter {
```

```
      Verified by = loggedInUser;
    }

    on reopen[always] do {<define statements>} transit to
Open
  }

  state Verified {
    on reopen[always] do {<define statements>} transit to
Open

    enter {
      Verified in build.required("Specify verified in
build");
      Verified by = loggedInUser;
    }

    exit {
      Verified in build = null;
    }
  }

  state Wait for Reply {
    on fix[always] do {<define statements>} transit to
Fixed

    on open[always] do {<define statements>} transit to
Open

    on in progress[always] do {<define statements>}
transit to In Progress

    on discuss[always] do {<define statements>} transit
to To be discussed

    on can't reproduce[always] do {<define statements>}
transit to Can't Reproduce

    on obsolete[always] do {<define statements>} transit
to Obsolete

    on duplicate[always] do {<define statements>} transit
to Duplicate

    on as designed[always] do {<define statements>}
transit to As designed

    on invalidate[always] do {<define statements>}
transit to Invalid
```

```
    }

  }
```

'On Duty' Management (download zip file)	Source Code
Manage duties. Requires a separate project to store duties. Duty is stored in the 'Assignee' field. The chain of duties is implemented as linked by 'is required for' link issues. **Requirements**: • states: 'On duty', 'Active', 'On a rest' • new field 'day of week' of type 'integer'	Set today's person-on-duty based on 'is required for' link

```
schedule rule Set Today's Duty Based on is required for

daily at 10 : 00 : 00 [issue.State == {On duty} && issue.Day of week != 5
 && issue.Day of week != 6] {
  if (updated + 10 seconds < now) {
    // set new duty
    var nextDuty = is required for.first;
    var stop = 0;

    if (nextDuty == null || nextDuty.Assignee == null) {
      project.leader.notify("Duty",
        "Duties chain should be implemented via 'is required for' links,
         and 'Duty' store field must be non-empty.",
        true);
    }

    while (stop != -1 && stop < 50) {
      if (nextDuty.State == {Active}) {
        nextDuty.Day of week = Day of week;
        nextDuty.State = {On duty};

        Assignee.notify("Duty: You are no longer on duty", Assignee.fullName
                        + " you just came off duty.<br><br>The new person or
                        + nextDuty.Assignee.fullName + ".", true);

        State = {Active};
        debug("Duty changed to: " + nextDuty.Assignee.fullName);

        stop = -1;
      } else if (nextDuty.State == {On a rest}) {
        nextDuty = nextDuty.is required for.first;
        stop = stop + 1;

      } else {
        project.leader.notify("Duty", "States should be only 'Active' or 'Or
rest'", true);

        stop = -1;
      }
    }

    nextDuty.Assignee.notify("Duty: You are on duty", nextDuty.Assignee.full
"
    + "you are on duty today.<br><br> Don't forget to process "
    + "<a href=\"http://youtrack.jetbrains.net/issues/JT?"
    +
"q=%23Unassigned+%23Unresolved+created%3A+Yesterday%2C+Today+sort+by%3A+Pric
    + "any new issues in YouTrack." + "</a>" + "<br>"
    + "<p style=\"color: gray;font-size: 12px;margin-top: 1em;border-top: 1p
#D4D5D6\">"
    + "Sincerely yours, YouTrack" + "</p>", true);

    var curIssue = nextDuty;
    var preventInfinite = 0;
    while (curIssue != issue && preventInfinite < 50) {
      if (curIssue.State == {Active}) {
        assert curIssue.Assignee != null: "Assignee cannot not be null!";

        curIssue.Assignee.notify("Duty: Person on duty has changed", "Today'
on duty is "
        + nextDuty.Assignee.fullName + ".", true);
      }
      curIssue = curIssue.is required for.first;
      preventInfinite = preventInfinite + 1;
    }
  }
}
```

Manage day of week:

```
schedule rule Set day of week

daily at 00 : 00 : 00 [issue.State == {On duty}] {
  Day of week = (Day of week + 1) % 7;
}
```

Disable dropping Assignee:

```
rule Assert Assignee not null

when Assignee.changed {
  assert Assignee != null: "Assignee cannot be null!";
}
```

Change Visibility Warning (download zip file)	Source Code
Warn on change of issue visibility. **Requirements**: group 'developers'	

```
rule issueVisibility

when permittedGroup.changed {
  var msg = "To post non-public data, please make the issue
public
  instead and add protected attachments and/or comments.";
  assert loggedInUser.isInGroup("developers"): msg;
}
```

Assignee and Multiple Subsystems (download zip file)	Source Code
When a Subsystem is changed, set issue assignee to the subsystem owner. **Requirements**: field 'Subsystems' of type 'ownedField[*]'	

```
rule Set assignee from Subsystems

when Subsystems.changed {
  var owner = null;
  for each subsystem in Subsystems {
    if (owner == null) {
      owner = subsystem.owner;
    } else if (subsystem.owner != owner) {
      owner = null;
      break;
    }
  }
  if (owner != null && Assignee == null) {
    Assignee = owner;
  }
}
```

Feedback Management (download zip file)	Source Code

Handle feedback getting from mailbox.
 	
Requirements:
• states: 'Answered', 'Unanswered'
• group 'developers'
• type 'Spam'
	Set the state to 'Answered' if a new comment is posted:
	
	`rule issue is answered if new comment posted`
	
	`when comments.added.isNotEmpty {`
	`  if (comments.last.author.isInGroup("developers")) {`
	`  State = {Answered};`
	`  } else {`
	`  State = {Unanswered};`
	`  }`
	`}`
	
	Issue becomes answered if a new comment posted
	
	`rule issue is answered if new comment posted`
	
	`when comments.added.isNotEmpty {`
	`  if (comments.last.author.isInGroup("developers")) {`
	`  State = {Answered};`
	`  } else {`
	`  State = {Unanswered};`
	`  }`
	`}`
	
	SPAM issues become answered
	
	`rule resolve on mark as SPAM`
	
	`when Type.becomes({Spam}) {`
	`  State = {Answered};`
	`}`
	
	Assign an issue to the author of the last 'developer group' comment
	
	`rule set assignee on developer comment`
	
	`when comments.added.isNotEmpty &&`
	`comments.added.last.author.isInGroup("developers")`
	`  && comments.added.last.author != Assignee {`
	`  Assignee = comments.added.last.author;`
	`}`

Comment Won't Fix State (download zip file)	Source Code

Require adding a comment when changing state to Won't fix.	Require adding comment when moving to Won't Fix state
	
	`rule Comment when won't fix`
	
	`when issue.State.becomes({Won't fix}) {`
	`  assert comments.added.isNotEmpty: "Please comment!";`
	`}`

Default Workflows

This table contains workflows included in YouTrack by default.

Additional Notifications (download zip file)	Source Code

Notify the previous assignee when issue is reassigned.	Notify the previous issue assignee
	`rule Notify on issue reassign`
	`when isReported() && Assignee.changed {`
	`  var oldAssignee = Assignee.oldValue;`
	`  var newAssignee = Assignee.login;`
	`  if (newAssignee == null) {`
	`  newAssignee = "nobody";`
	`  }`
	`  if (oldAssignee != null) {`
	`  oldAssignee.notify("[YouTrack, Reassigned] Issue " +`
	`getId() + ": " + summary,`
	`  "Issue " + "<a href=\"" + issue.getUrl() + "\">" +`
	`issue.getId() + "</a>`
	`was reassigned from you to " + newAssignee);`
	`  }`
	`}`

No Comments for Verified Issues (download zip file)	**Source Code**

Disable adding comments to a verified issue.	No Comments for Verified Issues
	`rule No Comments for Verified Issues`
	
	`when State == {Verified} && !State.becomes({Verified}) {`
	`  assert comments.added.isEmpty: "Commenting fixed and`
	`verified issues is disabled.";`
	`}`

New Issue Description Template (download zip file)	**Source Code**

Provide external reporters with a description template for New Issue.	Description template for external users
	`rule Description template for external users`
	
	`when !isReported() && description == null {`
	`  description = "What are the steps to reproduce the`
	`problem?\n" + "1.\n2.\n3.\n\n" +`
	`"What is the expected result?\n\n" + "What happens`
	`instead?\n\n" +`
	`"Please provide any additional information below.\n" +`
	`"Attach a screenshot if possible\n";}`

Fixing Linked Issues, (download zip file)	**Source Code**

Prohibit moving an issue to Fixed state if it 'depends on' unresolved issue(s).	Disable fixing issues with unresolved dependencies
	`rule Do not allow fix issue with unresolved dependencies`
	
	`when State.becomes({Fixed}) && depends on.isNotEmpty {`
	`  for each dep in depends on {`
	`assert dep.State.isResolved: "The issue has unresolved dependencies and cannot be Fixed!";`
	`  }`
	`}`

Voting for Resolved Issues (download zip file)	Source Code

Disable voting for resolved issues.	Disable voting for resolved issues
	`rule jetbrains-youtrack-doNotVoteForResolvedIssue`
	
	`when isResolved() {`
	`  assert !votes.changed: "Voting for resolved issues is disabled.";`
	`}`

Due Date Management (download zip file)	Source Code

Disable submitting an issue without a Due Date set. Notify the issue assignee about overdue issues.
 	
Requirements: field 'Due date' is attached to project
	Disable submitting an issue without a due date
	`rule Don't allow to submit issue without Due Date set`
	
	`when !issue.isReported() {`
	`  Due date.required("Please set the Due Date!");`
	`}`
	
	Notify an issue assignee about overdue issues
	`schedule rule Notify assignee about overdue issues`
	
	`daily at 10 : 00 : 00 [now > issue.Due Date] {`
	`  var user;`
	`  if (issue.Assignee == null) {`
	`  user = issue.project.leader;`
	`  } else {`
	`  user = issue.Assignee;`
	`  }`
	`  user.notify("Issue is overdue", "Please see the issue: " + issue.getId());`
	`}`

Intelligent Duplicates Management (download zip file)	Source Code
	Users cannot remove all duplicate links from issue in state Duplicate

- Automatically relink all duplicates directly to the main duplicated issue.
- User can't remove all duplicate links from issue in Duplicate state.
- Set issue state to duplicate when a Duplicate link is added.
- Raise the Priority of a duplicated issue.
- Require adding a duplicate link when changing issue state to Duplicate.

```
rule User can't remove all duplicate links from issue in
state Duplicate

when issue.State == {Duplicate} &&
duplicates.removed.isNotEmpty {
 assert duplicates.isNotEmpty: "Issue in Duplicate state
should have at least one duplicate.";
}
```

Set issue state to Duplicate when duplicate link is added

```
rule When have duplicate link set issue state to
Duplicate

when duplicates.added.isNotEmpty && State != {Duplicate}
{
  if (State.canBeUpdatedBy(loggedInUser)) {
    State = {Duplicate};
  }
}
```

Require adding a link to the main issue when issue state is set to Duplicate

```
rule When issue becomes duplicate it must have duplicate
link

when State.becomes({Duplicate}) {
  duplicates.required("Add link to duplicate issue.");
}
```

Don't allow duplicates to form a tree structure with height greater than one (target)

```
rule Don't allow duplicates to form a tree structure with
height greater than one (target)

when issue.is duplicated by.added.isNotEmpty {
  info("Processing duplicate-target issue " +
issue.getId());
  var target = issue.duplicates.first;
  if (target != null) {
    // ban several duplicates for one issue
    issue.duplicates.clear;
    issue.duplicates.add(target);
    for each incoming in issue.is duplicated by {
      incoming.duplicates.remove(issue);
      if (incoming == target) {
        info("Target cycle resolved for issue " +
target.getId());
      } else {
        incoming.duplicates.add(target);
      }
    }
  }
}
```

Don't allow duplicates to form a tree structure with height greater than one (source)

```
rule Don't allow duplicates to form a tree structure with
height greater than one (source)

when issue.duplicates.added.isNotEmpty {
  info("Processing duplicate-source issue " +
issue.getId());
  var target = issue.duplicates.added.first;
  for each incoming in issue.is duplicated by {
    incoming.duplicates.remove(issue);
    if (incoming == target) {
      info("Source cycle resolved for issue " +
target.getId());
    } else {
      incoming.duplicates.add(target);
    }
  }
}
```

When 'is duplicated by' link is added, try to raise priority

```
rule When is duplicated by link is added try to raise
priority

when issue.is duplicated by.added.isNotEmpty {
  var priorityOrdinal = issue.Priority.ordinal;
  for each source in issue.is duplicated by.added {
    var sourcePriority = source.Priority;
    var sourcePriorityOrdinal = sourcePriority.ordinal;
    if (priorityOrdinal > sourcePriorityOrdinal) {
      priorityOrdinal = sourcePriorityOrdinal;
      issue.Priority = sourcePriority;
    }
  }
}
```

Deny Certain Combination of Field Values (download zip file)	**Source Code**
Disable a certain combination of field values.	Denied Combinations

```
rule deniedCombinations

when <issue created or updated> {
  assert !(Priority == {Show-stopper} && State ==
{Submitted}):
 "Denied fields combination detected (Submitted
Show-stopper)";

  assert !(State == {Open} && Assignee == null):
 "Denied fields combination detected (Open Unassigned)";
}
```

Notify Reporter To Approve Fix (download zip file)	**Source Code**

- Send a specific notification to the reporter to approve the fix.
- Introduce State life-cycle including reporter verification.

Send a notification to the reporter to approve fix

```
rule Send specific notification to reporter to approve fix

when State.becomes({Pending verification}) {
  reporter.notify("Please approve fix for the issue" +
getId(), "You have reported issue"
 + getId() + "Please review the applied fix for your issue
and set the appropriate state.");
  Assignee = reporter;
}
```

State life-cycle with verified by reporter value

```
statemachine State lifecycle with verification by
reporter for field State {

  initial state Submitted {
    on Open[always] do {<define statements>} transit to
Open
  }

  state Open {
    on Fix[always] do {<define statements>} transit to
Fixed
  }

  state Fixed {
    enter {
      Fixed in build.required("Please set the 'Fixed in
build' value.");
    }

    on Send for verification[always] do {<define
statements>} transit to Pending verification
  }

  state Pending verification {
    enter {
      Assignee = reporter;
      reporter.notify("Please approve fix for the issue"
+ getId(), "You have reported issue"
 + getId() + "Please review the applied fix for the issue
and set the appropriate state.");
    }

    on Approve[always] do {<define statements>} transit
to Verified

    on Reopen[always] do {<define statements>} transit to
Open
  }

  state Verified {
    on Reopen[always] do {<define statements>} transit to
Open
  }

}
```

Fix Version and Fixed State (download zip file)	Source Code

- Require Fix Version value for Fixed state.
- Set an issue state to Fixed when Fixed Version is added.

Require Fix Version value for Fixed state

```
rule Assert Fix version is set for Fixed issues

when State.becomes({Fixed}) {
  Fix versions.required("Please set the 'Fix versions'
field!");
}
```

Set Fixed state on specifying Fixed in version

```
rule Set Fixed state on specifying Fixed in version

when State != {Fixed} && Fix versions.added.isNotEmpty {
  State = {Fixed};
}
```

Set Assignee to Subsystem Owner (download zip file)	Source Code
Set subsystem owner as assignee for new issues when subsystem is specified.	Set subsystem owner as assignee for new issues

```
rule Set subsystem owner as assignee for new issues

when Assignee == null && (Subsystem.changed ||
project.changed) {
  if (issue.Subsystem != null) {
    issue.Assignee = issue.Subsystem.owner;
  }
}
```

Time Management (download zip file)	Source Code

- Change issue state in time automatically.
- Notify an issue assignee, project lead or subsystem owner when issue life-cycle is violated.

Requirements:
- subsystem 'Support'
- state 'Overdue'
- state 'Wait for reproduce'
- state 'Approved'

Time Management for State field

```
statemachine Time Management for field State {

  initial state Submitted {
    enter {
      Subsystem = {Support};
      Assignee.required("Responsible support engineer is
required!");
    }

    in 1 hour[always] do {<define statements>} transit to
Overdue

    on reproducing[always] do {<define statements>}
transit to Open

    on incomplete[always] do {<define statements>}
transit to Incomplete
  }

  state Overdue {
    enter {
      var user;
```

```
      if (Assignee != null) {
        user = Assignee;
      } else if (Subsystem.owner != null) {
        user = Subsystem.owner;
      } else {
        user = project.leader;
      }
      user.notify("Acknowledgement needed", "Issue" +
getId() + "is waiting for acknowledgement.");
    }

    on incomplete[always] do {<define statements>}
transit to Incomplete

    on reproducing[always] do {<define statements>}
transit to Open
  }

  state Open {
    in 4 hours[always] do {<define statements>} transit
to Wait for reproduce

    on approved[always] do {<define statements>} transit
to Approved

    on incomplete[always] do {<define statements>}
transit to Incomplete

    on can't reproduce[always] do {<define statements>}
transit to Can't Reproduce
  }

  state Wait for reproduce {
    in 1 day[always] do {
      var user;

      if (Subsystem.owner != null) {
        user = Subsystem.owner;
      } else {
        user = project.leader;
      }
      user.notify("Issue is not reproduced in 1 day",
 "Issue " + getId() + " is still waiting for reproduction
steps.");
      // Notify sales?
    }

    in 3 days[always] do {
      var user;

      if (Subsystem.owner != null) {
        user = Subsystem.owner;
      } else {
        user = project.leader;
      }
      user.notify("Issue is not reproduced in 4 days",
"Issue " + getId() +
" is not reproduced. You may want to visit the customer
at their site.");
      // Notify sales?
    }

    on approved[always] do {<define statements>} transit
to Approved

    on can't reproduce[always] do {<define statements>}
transit to Can't Reproduce

    on incomplete[always] do {<define statements>}
transit to Incomplete
```

```
  }

  state Can't Reproduce {
    on reopen[always] do {<define statements>} transit to
Open
  }

  state Incomplete {
    on reopen[always] do {<define statements>} transit to
Open
  }

  state Approved {
    enter {
      Assignee.required(fail message);
    }

    on fixed[always] do {<define statements>} transit to
Fixed

    on obsolete[always] do {<define statements>} transit
to Obsolete
  }

  state Fixed {
    on verify[always] do {<define statements>} transit to
Verified

    on reopen[always] do {<define statements>} transit to
Open
  }

  state Obsolete {
    on reopen[always] do {<define statements>} transit to
Open
  }

  state Verified {
    on reopen[always] do {<define statements>} transit to
Open
```

```
    }

  }
```

Fix in Build Management (download zip file)	Source Code
• Clear Fixed in build value when issue becomes unresolved. • Copy Fixed in Build value from duplicate. • Set Fixed in Build value for duplicate when it's specified in the main issue.	Clear Fixed in build on issue unresolve

```
rule Clear Fixed in build on issue unresolve

when issue.isReported() && !(issue.State.isResolved)
 && issue.State.oldValue != null &&
issue.State.oldValue.isResolved {
  if (issue.Fixed in build != null) {
    issue.Fixed in build = null;
  }
}
```

Copy Fixed in build to duplicate from the main issue

```
rule Copy Fixed in build from duplicated issue
when State.becomes({Duplicate}) && duplicates.isNotEmpty
{
  for each duplicatedIssue in duplicates {
    if (duplicatedIssue.project == project &&
duplicatedIssue.Fixed in build != null) {
      Fixed in build = duplicatedIssue.Fixed in build;
      break;
    }
  }
}
```

Copy Fixed in build to duplicates when it is set in the main issue

```
rule Copy Fixed in build to duplicate issues when it is
set

when Fixed in build.changed && Fixed in build != null {
  var duplicatedBy = is duplicated by;
  for each duplicate in duplicatedBy {
    if (duplicate.project == project) {
      duplicate.Fixed in build = Fixed in build;
    }
  }
}
```

Fixed in Build for Won't Fix (download zip file)	Source Code
Set Fixed in build to 'Never' for 'Won't fix' state. **Requirements**: fixed in build 'Never'	Set 'Never' fixed in build for 'Won't fix' state

```
rule Set 'Never' fixed in build for 'Won't fix' state

when State.becomes({Won't fix}) {
  Fixed in build = {Never};
}
```