Project Phase Template 1.2 Component Specification

1. Design

All changes performed when synchronizing documentation with the version 1.1 of the source code of this component are marked with **purple**.

All changes made in the version 1.2 are marked with **blue**.

All new items in the version 1.2 are marked with **red**.

A project is usually executed in a predefined set of phases for a particular customer. Requiring the user to manually define the phase hierarchy is laborious and unnecessary. The component provides a template mechanism to handle this scenario. Template storage is pluggable and can be added without code changes. XML storage is provided with this release.

There're two things need to be pluggable in this component – the template persistence logic and the default project start date generation logic. Two interfaces are defined to fulfill those requirements, so that plugging a persistence mechanism or start date generation algorithm is very simple.

PhaseTemplatePersistence interface defines the contract to generate a set of project phases from a specific persistence mechanism, a built-in implementation based on XML persistence is designed as the default persistence logic, a schema and a sample persistence file are provided in the docs directory. Inside the XmlPhasePersistence, the template is stored as a Map of Document DOM objects, keyed on the template names.

StartDateGenerator interface defines the contract to generate a default project start date according to a specific generation algorithm, a built-in implementation which generates a relative time in a week is designed as the default start date generation logic, it can generate a relative time in any week based on the configurations. One thing to note is that, a concept of "week offset" is introduced to describe the relationship of weeks. For example, if we say the target week has a week offset of 1 from current week, then the target week is the next week, if we say the target week has a week offset of 0 from current week, then the target week is current week itself.

Version 1.1 adds the following:

- database persistence
- a category classification for templates
- extra fields to templates
- a new way of applying the templates.
- a migration utility to translate XML templates to DB templates.

The new version is backwards compatible, i.e. if this version is used instead of version 1.0 in an existing application, it reads the old XML template file and the application code shouldn't be changed in order to continue working.

New methods were added to PhaseTemplate interface. Two new applyTemplate() overloads can accept variable phase ID, fixed phase ID and fixed phase start date. This way of generating phases enables the user to additionally specify when a phase (other than the first) starts. It is used, for example, to specify both registration and review start dates, and let the submission length be automatically adjusted.

Also PhaseTemplate interface now contains methods those allow the user to retrieve some information about templates (creation date, description and category – an integer value, representing categories like design, development, architecture, specification and so on, but this component doesn't care about the actual meaning of this value).

PhaseTemplatePersistence interface was extended to support additional template parameters (category, creation date and description).

DBPhaseTemplatePersistence class was designed to add database persistence functionality to the component. It also provides a method that allows the user to remove a template from

database.

Some XML templates will be migrated to DB. Thus ConverterUtility class is provided to be used as a standalone utility that reads an XML file and generate SQL insert commands. Those commands are sent to standard output so that the user can copy-paste and run them. In the automatic mode this utility can automatically execute all generated SQL queries.

In the version 1.2 this component was updated to support flexible selection of phases from the template to be put to the created project. Now the user can pass the list of IDs of phases to be left out to one of new overloads PhaseTemplate#applyTemplate() of method.

Please note that currently phase IDs are inconsistent between XmlPhaseTemplatePersistence and DBPhaseTemplatePersistence

1.1 Design Patterns

Strategy Pattern – The component has decoupled the logic for phase template persistence (PhaseTemplatePersistence interface and its implementations) and project start date generation (StartDateGenerator interface and its implementations).

1.2 Industry Standards

XML, XSD schema, JDBC, SQL

1.3 Required Algorithms

1.3.1 Generate project phases from XML-based phase template persistence

The XML schema for the template persistence is fairly simple, there're three main concepts:

- λ **PhaseType definitions** : defines the phase types information needed in this template, type id and type name is included(note that in this XML schema, attribute "id" is reserved for internal reference, so attribute "typeId" is used to store the id of the type, while "id" attribute is used to store the internal reference point so that the phase definitions can specify the type conveniently).
- λ **Phase definitions** : defines the phases in this template, the information included are – phase length(in "length" attribute), phase type (in "type" attribute") and dependency information (in "Dependency" child elements). An internal "id" is reserved for internal reference so that we can specify the dependency information conveniently.
- λ **Dependency descriptions** : defines a single dependency of a phase, the information included are – the dependency phase (in "id" attribute), whether the dependency is a start (in "isDependencyStart" attribute, optional, default to false if missing) , whether the dependent is a start (in "isDependentStart" attribute, optional, default to true if missing) and lag time(in "lagTime" attribute, optional, default to 0 if missing) between the dependency and dependent. Note that zero or more "Dependency" child elements may exist, each one describes a single dependency information.

Please see docs/xml_phase_template.xsd and docs/sample.xml for more details.

Inside the XmlPhasePersistence, the template is stored as a Map of Document DOM objects, keyed on the template names.

Given a template name, the phases generation algorithm is as following:

XmlPhaseTemplatePersistence.generatePhases(String templateName , Project project)

0. Getting the template DOM Document.

First, retrieve the DOM Document object for the template associated with the given templateName.

Document templateDOM = (Document) templates.get(templateName);

1. Phase types generation

Before generating the phases, all phase types defined in the template will be generated and cached in a Map with the id for further reference. (NOT typeId) as the key (The developer may choose lazy generation to improve the efficiency):

For each "PhaseType" element, create a PhaseType object with the "typeId" and "typeName" attributes, then put the PhaseType object into a map for caching.

```
Map phaseTypes = new HashMap();
NodeList list = templateDOM.getElementsByTagName("PhaseType");
for (int i = 0; i < list.getLength(); i++) {
    Element typeElement = (Element) list.item(i);
    phaseTypes.put(typeElement.getAttribute("id"),
 new PhaseType(Long.parseLong(typeElement.getAttribute("typeId")),
typeElement.getAttribute("typeName"));
}
```

2. Phases generation

It is recommended to perform a 2-pass scan to generate the full hierarchy(however the developer may choose more efficient approach where appropriate):

In the 1st pass, create a Phase object for each "Phase" element with the "length" and "type" attributes, ignore the dependencies information, and cache the Phase objects in a Map, with the "id" attribute as the key.

In the 2nd pass, go through the dependencies information for each "Phase" element, add the dependency relationship to the created Phase objects.

In the version 1.2 this algorithm was updated to check whether all phases specified in leftOutPhaseIds parameter exist in the template, ignore phases specified in leftOutPhaseIds and “redirect” all dependencies to left out phases. See sample in the section 4.3.14 to understand how redirection of dependencies works. Note that leftOutPhaseIds equal to null must be treated as an empty array.

Finally, return the created phase objects.

```
// 1st pass
Map phases = new HashMap(); // key is phase ID string (name)
Set leftOutPhaseNames = new HashSet();
Map dependenciesToLeftOutPhases = new HashMap();
// keys in both maps are Longs for dependent phase IDs
// values in both maps are Lists of Dependency instances
list = templateDOM.getElementsByTagName("Phase");
for (int i = 0; i < list.getLength(); i++) {
    Element phaseElement = (Element) list.item(i);
    String id = Long.parseLong(phaseElement.getAttribute("id"));
    // do this only if phaseId attribute is present:
    long phaseId = Long.parseLong(phaseElement.getAttribute("phaseId"));
    If phaseId is specified and leftOutPhaseIds contains phaseId then
        leftOutPhaseNames.add(id);
    long length = Long.parseLong(phaseElement.getAttribute("length"));
    Phase phase = new Phase(project, length);
    phase.setId(phaseId);
     phase.setPhaseType((PhaseType)
  phaseTypes.get(phaseElement.getAttribute("type")));
    phases.put(id, phase);
}
If leftOutPhaseNames.size() != leftOutPhaseIds.length then
```

```
        Throw IllegalArgumentException (at least one of elements in leftOutPhaseIds is not
                a valid phase ID)
    // 2nd pass
    for (int i = 0; i < list.getLength(); i++) {
        Element phaseElement = (Element) list.item(i);
         NodeList dependencyElements =
              phaseElement.getElementsByTagName("Dependency");
        Phase phase = (Phase) phases.get(phaseElement.getAttribute("id"));
        for (int j = 0; j < dependencyElements.getLength(); j++) {
            Element dependencyElement = (Element) dependencyElements.item(j);
            String dependencyId = dependencyElement.getAttribute("id");
             Phase dependencyPhase = (Phase) phase.get(dependencyId);
            // isDependencyStart flag, optional attribute, default to false if missing.
            boolean isDependencyStart = false
            // isDependentStart flag, optional attribute, default to true if missing
            boolean isDependentStart = true;
             // lagTime between the dependent and the dependency, optional attribute,
    default to
             //0  if missing
            int lagTime = 0;
            // temp variable to cache attribute value
            String tmp = dependencyElement.getAttribute("isDependencyStart");
            if (tmp != null) {
                if (tmp.equals("true") {
                    isDependencyStart = true;
      } else if (tmp.equals("false") {
          isDependencyStart = false;
      } else {
          throw new IllegalArgumentException("cannot parse isDependencyStart");
      }

    }
            tmp = dependencyElement.getAttribute("isDependentStart");
            if (tmp != null) {
                if (tmp.equals("true") {
                    isDependentStart = true;
      } else if (tmp.equals("false") {
          isDependentStart = false;
      } else {
          throw new IllegalArgumentException("cannot parse isDependentStart");
      }
            }
            tmp = dependencyElement.getAttribute("lagTime");
            if (tmp != null) {
                lagTime = Integer.parseInt(tmp);
    }
         Dependency dependency = new Dependency(dependencyPhase,
phase,  isDependencyStart, isDependentStart, lagTime);
          If leftOutPhaseNames contains dependencyId then
              long depPhaseId = dependencyPhase.getId();
              List dependenciesToLeftOutPhase =
                  dependenciesToLeftOutPhases.get(depPhaseId);
              If dependenciesToLeftOutPhase == null then
                 dependenciesToLeftOutPhase = new ArrayList();
                  dependenciesToLeftOutPhases.put(depPhaseId,
dependenciesToLeftOutPhase);
              dependenciesToLeftOutPhase.add(dependency);
          phase.addDependency(dependency);
        }
    }
    // process all dependencies for left out phases
    // this algorithm replaces each dependency from not left out phase to left out phase
    // with dependencies from the original not left out phase to not left out phases that
```

```
        // are dependencies of the original left out phase
        // Note that since dependency phases of left out phases can be also left out,
        // the breadth-first search approach is used in this algorithm to locate all new
dependencies
        // to be used instead of the old dependency
        For each (leftOutPhaseId:long; dependencies:List) from
dependenciesToLeftOutPhases do:
            For each dependency:Dependency from dependencies do:
                Phase dependentPhase = dependency.getDependent();
                If leftOutPhaseIds contains dependentPhase.getId() then continue;
                LinkedList dependenciesToBeProcessed = new LinkedList();
                Set allProcessedDependencies = new HashSet();
                dependenciesToBeProcessed.add(dependency);
                While dependenciesToBeProcessed is not empty do:
                    Dependency curDependency = dependenciesToBeProcessed.removeFirst();
                    If allProcessedDependencies contains curDependency then continue;
                    allProcessedDependencies.add(curDependency);
                    If leftOutPhaseIds contains curDependency.getDependency().getId() then
                        Set moreDependencies =
curDependency.getDependency().getDependencies();
                        For each dependencyToProcess:Dependency from moreDependencies do:
                            Combine two dependencies – curDependency and dependencyToProcess
– as
                                described in the section 1.3.10. Save result to
combinedDep:Dependency.
                            dependenciesToBeProcessed.add(combinedDep);
                    Else
                        If dependentPhase.getDependencies() contains a Dependency with ID equal
                            to curDependency.getDependency().getId(), then continue.
                        Create a copy of curDependency (create a new instance of Dependency and
                            copy all attributes to it). Set result to dependencyCopy:Dependency.
                        dependencyCopy.setDependent(dependentPhase);
                        dependentPhase.addDependency(dependencyCopy);
        // add the phases to the project
        for (Iterator itr = phases.values().iterator(); itr.hasNext();;) {
            Phase phase = (Phase) itr.next();
            If leftOutPhaseIds doesn't contain phase.getId() then
                project.addPhase(phase);
        }
```

If any exception is caught in the above process, wrap it into PhaseGenerationException and throw out. Generally the possible errors would be:

(1) dependency reference doesn't exist

(2) phase type reference doesn't exist

(3) cyclic dependency exists in the template(certain exception will be thrown from Project Phases component).

(4) dependency phase cannot be left out when dependent phase is not left out

1.3.2 *Generate a relative time in a week*

1. Obtain a Calendar instance for current time:

Calendar cal = Calendar.getInstance();

2. Adjust the time to the week according to the weekOffset

cal.add(Calendar.WEEK_OF_YEAR, weekOffset);

3. Adjust the time to the day of week according to the dayOfWeek

cal.set(Calendar.DAY_OF_WEEK, dayOfWeek);

4. Adjust the time to the exact time according to the hour, minute, second.

cal.set(Calendar.HOUR_OF_DAY, hour);

cal.set(Calendar.MINUTE, minute);

cal.set(Calendar.SECOND, second);

5. Return the adjusted time.

return cal.getTime();

1.3.3 Generate a project with phases from a given template and a start date

DefaultPhaseTemplate.applyTemplate(String template, Date startDate):

1. Create an empty project : *Project project = new Project(startDate, workdays);*
2. Generate a set of phases and add them to the project by calling *persistence.generatePhases(template, project, null);*

1.3.4 Generate a project with phases from a given template, without a specific start date provided

DefaultPhaseTemplate.applyTemplate(String template):

1. Generate a start date by calling *startDateGenerator.generateStartDate()*,
2. Create an empty project with the date generated from 1 :

   Project project = new Project(startDate, workdays);

3. Generate a set of phases and add them to the project by calling *persistence.generatePhases(template, project, null);*

1.3.5 Adjust intermediate phase start date

This algorithm corresponds to DefaultPhaseTemplate#adjustPhases(project:Project, varPhaseId:int, fixedPhaseId:int, fixedPhaseStartDate:Date) method.
Each phase may have many dependencies. Thus in some cases it's impossible to adjust some phase start date only by changing the length of some other phase. This algorithm is based on binary search:

- Get all phases of project.
- In the retrieved array find varPhase with id equal to varPhaseId and fixedPhase with id equal to fixedPhaseId.
- Get difference between current fixed phase start date and desired one (in minutes):
  long diff = (fixedPhaseStartDate.getTime() - fixedPhase.calcStartDate().getTime()) / 60000;
- Save the variable phase length (in minutes):
  long oldLength = varPhase.getLength() / 60000;
- If diff is equal to 0 then return.
- Use the binary search to find the best correction:
- long left=0, right=oldLength + 7*24*60 + diff; // this will take less than 25 iterations
- While left <= right do:
  long mid = (left + right) / 2;
  varPhase.setLength(mid*60000);
  diff = (fixedPhaseStartDate.getTime() - fixedPhase.
  calcStartDate().getTime()) / 60000;
  If diff = 0 then return;
  If diff > 0 then left = mid+1 else right = mid-1.
- Throw exception.

1.3.6 Generate project phases from database-based phase template persistence

This algorithm corresponds to DBPhaseTemplatePersistence#generatePhases() method.
In this algorithm some queries are executed to retrieve corresponding information from the database. Please read section 1.3.8 for details on how to perform SQL queries.
- Create a database connection with use of DBConnectionFactory.
- Execute query "SELECT id FROM template WHERE name={1}", where {1}=templateName.
- Save returned value to templateId. If zero or more than one row is present in the query

result then throw an exception.
- Create a types map (keys are global type ids in DB, values are PhaseType objects).
- Execute query “SELECT id, type_id, name FROM phase_type WHERE template_id={1}”, where {1}=templateId.
- For each row in the result:
  Extract id, typeId and name values.
  Create PhaseType(typeId, name).
  Add created object to types map: types.put(id, phaseType).
- Create a phases map (keys are global phase ids in DB, values are Phase objects).
- int skippedPhasesNum = 0;
- Map dependenciesToLeftOutPhases = new HashMap();
  // keys in both maps are Longs for dependent phase IDs
  // values in both maps are Lists of Dependency instances
- Execute query “SELECT id, phase_type_id, phase_id, time_length FROM phase WHERE template_id={1}”, where {1}=templateId.
- For each row in the result:
  Extract id, typeId, phaseId and timeLength values.
  If leftOutPhaseIds contains id then
    skippedPhasesNum++;
  Create Phase object: new Phase(project, timeLength).
  Set phase type: phase.setPhaseType(types.get(typeId)).
  Set phase ID: phase.setId(phaseId).
  Add phase object to phases map: phases.put(id, phase).
- If skippedPhasesNum != leftOutPhaseIds.length then
  Throw IllegalArgumentException (at least one of elements in leftOutPhaseIds is not a valid phase ID)
- For each phase from phases:
  Execute query “SELECT dependency_id, dependent_start, dependency_start, lag_time FROM dependency WHERE dependent_id={1}”, where {1}= phase id in database.
  For each row from the result:
    Extract dependencyId, dependentStart, dependencyStart and lagTime.
    Create a dependency: new Dependency(phases.get(dependencyId), phases.get(dependentId), dependencyStart, dependentStart, lagTime).
    If leftOutPhaseIds contains dependencyId then
      List dependenciesToLeftOutPhase = dependenciesToLeftOutPhases.get(dependencyId);
      If dependenciesToLeftOutPhase == null then
        dependenciesToLeftOutPhase = new ArrayList();
        dependenciesToLeftOutPhases.put(dependencyId, dependenciesToLeftOutPhase);
      dependenciesToLeftOutPhase.add(dependency);
    Add a dependency to a phase: phases.get(dependentId).addDependency(dependency);
// Process all left out dependencies
// The same algorithm as in the section 1.3.1 is used here, thus developers
// need to perorm a proper code refactoring to avoid duplication
- For each (leftOutPhaseId:long; dependencies:List) from dependenciesToLeftOutPhases do:
  For each dependency:Dependency from dependencies do:
    Phase dependentPhase = dependency.getDependent();
    If leftOutPhaseIds contains dependentPhase.getId() then continue;
    LinkedList dependenciesToBeProcessed = new LinkedList();
    Set allProcessedDependencies = new HashSet();
    dependenciesToBeProcessed.add(dependency);
    While dependenciesToBeProcessed is not empty do:
      Dependency curDependency = dependenciesToBeProcessed.removeFirst();
      If allProcessedDependencies contains curDependency then continue;
      allProcessedDependencies.add(curDependency);
      If leftOutPhaseIds contains curDependency.getDependency().getId() then
        Set moreDependencies = curDependency.getDependency().getDependencies();
        For each dependencyToProcess:Dependency from moreDependencies do:
          Combine two dependencies – curDependency and dependencyToProcess – as described in the section 1.3.10. Save result to combinedDep:Dependency.

dependenciesToBeProcessed.add(combinedDep);

Else

If dependentPhase.getDependencies() contains a Dependency with ID equal to curDependency.getDependency().getId(), then continue.

Create a copy of curDependency (create a new instance of Dependency and copy all attributes to it). Set result to dependencyCopy:Dependency.

dependencyCopy.setDependent(dependentPhase);

dependentPhase.addDependency(dependencyCopy);

- For each phase from phases:

If leftOutPhaseIds doesn't contain phase.getId() then

Check cyclic dependencies: phase.calcStartDate();

Add phase to the project: project.addPhase(phase);

1.3.7 *Generate a list of INSERT SQL command for specified XML phase template files*

This algorithm corresponds to ConverterUtility#convert(templateFileNames) method.

In the following algorithm if some error occurs (e.g. attribute is absent in XML file, needed element is absent in a map) a corresponding message must be printed out to System.out and execution must be terminated with return statement.

- If connection is not null then start a transaction.
- Create DocumentBuilderFactory factory instance:
  DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
- Create DocumentBuilder builder from factory:
  DocumentBuilder builder = factory.newDocumentBuilder();
- For each fileName from templateFiles:
  templatesId++;
  Parse the given file:
  Document doc = builder.parse(new File(fileName));
  Get current template name:
  String templateName = doc.getDocumentElement().getAttribute("name");
  Read category of the current template to String templateCategory (use "category" attribute), set to "0" if not specified.
  Read creation date to String creationDate (use "creationDate" attribute), set to "NULL" if not specified.
  Read description to String description (use "description" attribute), set to "NULL" if not specified.
  Read, parse as boolean and save "isDefault" attribute (can be "true"/"yes" or "false"/"no") to boolean templateIsDefault, set to false if attribute is absent.
  Output to System.out line "INSERT INTO template (id, name, category, creation_date, description) VALUES ({1}, '{2}', {3}, '{4}', '{5}');", replace {X} with templatesId, templateName, category, creationDate and description values correspondingly. Execute this query if connection is not null.
  If templateIsDefault then output to System.out line "INSERT INTO default_template (template_id, category) VALUES ({1}, {2});", replace {X} with templatesId and category values correspondingly. Execute this query if connection is not null.
  Create a map of phase types (key will be an internal id in XML file, value - id in a database table):
  Map phaseTypes = new HashMap();
  Get a list of phase type nodes:
  NodeList list = doc.getDocumentElement().getElementsByTagName("PhaseTypes");
  For each Element typeElement from list:
  phaseTypesId++;
  Read "id", "typeId" and "typeName" attributes of an element:
  String id = typeElement.getAttribute("id"); // must be an integer though
  int typeId = Long.parseLong(typeElement.getAttribute("typeId"));
  String typeName = typeElement.getAttribute("typeName");
  Add an item to phaseTypes:
  phaseTypes.put(id, phaseTypesId);
  Output to System.out line "INSERT INTO phase_type (id, template_id, type_id, name) VALUES ({1}, {2}, {3}, '{4}');", replace {X} with phaseTypesId, templatesId, typeId and typeName values correspondingly. Execute this query if connection is not null.
  Create a map of phases (key is an internal id in XML file, value - id in a database table):
  Map phases = new HashMap();

int curPhaseId = 0; // is used if "phaseId" attribute is absent
Get a list of phase nodes:
list = doc.getDocumentElement().getElementsByTagName("Phases");
For each Element phaseElement from list:
phasesId++; curPhaseId++;
Get "id", "phaseId", "length" and "type" attributes of an element to String id, phaseId, length and type correspondingly, if "phaseId" attribute is absent, set phaseId to curPhaseId instead.
Add an item to phases:
phases.put(id, phasesId);
Get typeId for type:
long typeId = (long) phaseTypes.get(type);
Output to System.out line "INSERT INTO phase (id, template_id, phase_type_id, phase_id, time_length) VALUES ({1}, {2}, {3}, {4}, {5});", replace {X} with phasesId, templatesId, typeId, phaseId and length values correspondingly. Execute this query if connection is not null.
For each Element phaseElement from list:
Get dependentId: read "id" attribute of phaseElement to String id, long dependentId = (long) phases.get(id).
Get dependencies list:
NodeList dependencyElements = phaseElement.getElementsByTagName("Dependency");
For each Element dependencyElement from dependencyElements:
dependenciesId++;
Get dependencyId: read "id" attribute of dependencyElement to String id, long dependencyId = (long) phases.get(id).
Get "isDependencyStart", "isDependentStart" attributes and set String isDependencyStart to "f" if corresponding attribute is absent or equal to "false", else set it to "t", set String isDependentStart to "t" if corresponding attribute is absent or equal to "true", else set it ti "f".
Get "lagTime" attribute of dependencyElement and set it to String lagTime.
Output to System.out line "INSERT INFO dependency (id, dependent_id, dependency_id, dependent_start, dependency_start, lag_time) VALUES ({1}, {2}, {3}, '{4}', '{5}', {6});", replace {X} with dependenciesId, dependentId, dependencyId, isDependentStart, isDependencyStart and lagTime values correspondingly. Execute this query if connection is not null.
- Discard or commit DB changes if connection is not null.

For details of parsing XML file see XMLPhaseTemplatePersistence.initializeTemplates() method source.

1.3.8 *Perform database SQL query*

SQL queries are performed by DBPhaseTemplatePersistence and ConverterUtility classes.
- Use DBConnectionFactory instance to create a Connection.
- Set autocommit to false.
- For each query in the current transaction PreparedStatement instance must be created.
- Start a transaction: Execute the PreparedStatement(s).
- Extract all required data from ResultSet object.
- If error occurred then rollback, else commit.
- Connection, PreparedStatement and ResultSet must be properly closed.
- Transaction is not needed for the simple query statement.

1.3.9 *Generate a project phases with some phases being left out*

This is performed in the following methods:

DefaultPhaseTemplate#applyTemplate(templateName, leftOutPhaseIds)

DefaultPhaseTemplate#applyTemplate(templateName, leftOutPhaseIds, startDate)

DefaultPhaseTemplate#applyTemplate(templateName, leftOutPhaseIds, varPhaseId, fixedPhaseId, fixedPhaseStartDate)

DefaultPhaseTemplate#applyTemplate(templateName, leftOutPhaseIds, varPhaseId, fixedPhaseId, fixedPhaseStartDate, startDate)

The algorithms are the same as for respective overloads without leftOutPhaseIds parameter, but this parameter is simply passed to PhaseTemplatePersistence#generatePhases() methods instead of null. See details in method docs provided in TCUML.

1.3.10 Combine dependencies

This algorithm is used when generating phases and some phases are left out.
Input of this algorithm is two dependencies (D1 and D2) that match the following condition: dependency phase in D1 is a dependent phase in D2. I.e. these two dependencies indicate that phase A depends on phase B, and phase B depends on phase C.
Output of this algorithm is a dependency D3 that occurs when phase B is left out.
This algorithm must create a new Dependency instance and set its properties as specified below:

D3.dependent = D1.dependent
D3.dependency = D2.dependency
D3.dependentStart = D1.dependentStart
D3.dependencyStart = D2.dependencyStart
D3.lagTime = D1.lagTime + D2.lagTime

1.4 Component Class Overview

Note: a more thorough description of all the classes is available in the Documentation tabs within TC UML Tool.

Interface PhaseTemplate
PhaseTemplate interface defines the contract to access the phase templates, generally the implementations will manage several different templates which are used to generate different project phase hierarchies. It provides the API to generate a set of project phases from a given predefined template, and compose a Project object with those phases, with or without a specified project start date, it also provides the API to retrieve names of all templates available to use.
In the version 1.1 new methods were added. They allow the user to retrieve additional information about each template: category, description and creation date, retrieve all template names and default one for each category. Also methods for applying templates with intermediate phase start time specified were added.
Changes in 1.2:
- Added applyTemplate() overloads that allow to left out some phases from the template being applied.
- Fixed type of varPhaseId and fixedPhaseId parameters.

Class DefaultPhaseTemplate
DefaultPhaseTemplate is a default implementation of PhaseTemplate interface.
It manages two underlying variables - persistence(of type PhaseTemplatePersistence) which is used as the actual template storage logic; and startDateGenerator(of type StartDateGenerator) which is used as the default project start date generation logic if there's no specific start date provided during the phase generation. By this, we can easily change the persistence and start date generator implementation so that the client is able to swap out for different storage and start date generation strategies without code changes - all we need to do is adding new PhaseTemplatePersistence and/or StartDateGenerator implementations.
In the version 1.1 new methods were added. They allow the user to retrieve additional information about each template: category, description and creation date, retrieve all template names and default one for each category. Also methods for applying templates with intermediate phase start time specified were added.
Changes in 1.2:
- Added applyTemplate() overloads that allow to left out some phases from the template being applied.
- Fixed type of varPhaseId and fixedPhaseId parameters.

Interface PhaseTemplatePersistence
PhaseTemplatePersistence interface acts as the persistence layer of the phase templates, so that the persistence is pluggable and can be added without code changes. It manages a set of templates, provides the API to generate an array of Phases from a template it

manages.
Note that this interface generates only phases, the Project generation is out of the scope of this interface.
In this initial release, the persistence is read-only, all templates are not modifiable with this component.The template authoring functionalities may be added in the future versions.
In the version 1.1 new methods were added. They allow the user to retrieve additional information about each template: category, description and creation date, retrieve all template names and default one for each category.
Change in 1.2:
- Added leftOutPhaseIds parameter to generatePhases() method.

Class XmlPhaseTemplatePersistence
XmlPhaseTemplatePersistence is an XML based persistence implementation.
Phase templates are stored in XML files, each XML file defines one template, the XML schema is defined in docs/xml_phase_template.xsd, and also a sample template is provided in docs directory.
Each template is assigned a template name, inside the XmlPhaseTemplatePersistence, templates are stored in a Map with the template name as the key, and with org.w3c.dom.Document objects parsed from the XML document as the value.
In the version 1.1 new methods were added. They allow the user to retrieve additional information about each template: category, description and creation date, retrieve all template names and default one for each category.
generatePhases() method was modified so that it now reads phaseId attribute of Phase elements.
Change in 1.2:
- Updated generatePhases() method to support skipping specified phases from the template.

Class DBPhaseTemplatePersistence
This is a database based persistence implementation. It uses DBConnectionFactory to establish a connection with DBMS. Structure of the database can be seen with use of DBDesigner (see DBModel.xml file) or by SQL tables creation script in database.sql file. Also it provides the user with the method that removes a template from the database.
Change in 1.2:
- Updated generatePhases() method to support skipping specified phases from the template.

Interface StartDateGenerator
StartDateGenerator interface defines the contract to generate a default start date for a project according to specific generation logic, so that the DefaultPhaseTemplate can employ different start date generation logic without code changes.
It will be used in the DefaultPhaseTemplate as the default project start date generation logic if no start date specified in the phase generation process.

Class RelativeWeekTimeStartDateGenerator
RelativeWeekTimeStartDateGenerator is the initial built-in implementation of StartDateGenerator interface.
It generates a relative time in a week as the default start date, for example, it can be configured to generate 9:00 AM next Thursday as the default start date.

ConverterUtility
This class is a persistence migrating utility for XML phase templates files. It can be used as a main class in a standalone utility which reads XML files with templates and outputs to console SQL INSERT queries those must be executed by the user to configure template phases database. This utility uses Configuration Manager component to retrieve all input data. Namespaces of XmlPhaseTemplatePersistence and optionally DBPhaseTemplatePersistence configurations must be specified as command line arguments.

1.5 Component Exception Definitions

IllegalArgumentException
It may be thrown from many places in this component where the argument is null or empty string, or the numeric argument is out of range.

PhaseTemplateException:
It is the base class for custom exceptions thrown from this component.

PhaseGenerationException:
It may be thrown from PhaseTemplatePersistence and PhaseTemplate implementations if there're errors in the phase generation process, e.g. the cyclic dependency, reference to an undefined project phase, etc.

StartDateGenerationException:
It may be thrown from StartDateGenerator implementations if there're errors in the start date generation process, e.g. connection errors in database-based start date generation implementations. In the initial release, this exception is not used at all, it is designed for future extension.

ConfigurationException:
It may be thrown from many places where the outside configuration is needed but there're errors in the configuration. for example, the expected namespace is not loaded, the required configuration property is missing, etc.

PersistenceException
PersistenceException indicates that there're errors while accessing the template persistence, e.g. persistence file doesn't exist for file-based persistence layer.

PersistenceRuntimeException
PersistenceRuntimeException is the runtime version of exception for the PhaseTemplatePersistence interface. This exception is used for DBPhaseTemplatePersistence's implementation on the methods defined in version 1.0. These methods in version 1.0 defined in PhaseTemplatePersistence interface would not allow to throw non-runtime exception.

1.6 Thread Safety

This component should be thread safe, implementations of the interfaces defined in this component must be thread safe.

RelativeWeekTimeStartDateGenerator and XmlPhaseTemplatePersistence are immutable and therefore they're thread safe.

DefaultPhaseTemplate is mutable(persistence and startDateGenerator may be modified), so appropriate locking mechanism is required in the applyTemplate method implementations, however it is fairly simple : lock on the persistence while generating phases from a template, lock on the startDateGenerator while generating the default project start date, lock on this in while changing the object references in setters.

New classes and changes in version 1.1 don't break thread safety of the component. Locking is used in new methods of DefaultPhaseTemplate, thus it is still thread safe. DBPhaseTemplatePersistence is immutable and thread safe class. ConverterUtility is used as a standalone utility, thus it is not required to be thread safe.

Thread safety of this component was not changed in the version 1.2.

2. Environment Requirements

2.1 Environment

Development language: Java 1.4
Compile target: Java 1.4, Java 1.5
QA Environment: Solaris 7, RedHat Linux 7.1, Windows 2000, Informix 10, Windows 2003

2.2 TopCoder Software Components

- **Configuration Manager Version 2.1.5** – loads configuration for all classes
- **Project Phases Version 2.0.1** – defines project phase entities used in this component
- **Object Factory Version 2.1** – creates instances of predefined pluggable objects
- **Object Factory Config Manager Plugin 1.0** – allows Object Factory to be configured with Configuration Manager component

start_date_generator.class	class of the StartDateGenerator	Full qualified class name

- λ **Workdays Version 1.0.1** – define Workdays instance used in this component
- λ **Base Exception Version 2.0** – provides a base class for custom exceptions
- λ **DB Connection Factory 1.1** – creates a predefined DB connection
- λ **Command Line Utility 1.0** – simplifies command line arguments parsing

2.3 Third Party Components

- λ **JAXP 3.0 Implementation (Xerces2-J , version 2.8.0 for example)**

3. Installation and Configuration

3.1 Package Name

com.topcoder.project.phases.template
com.topcoder.project.phases.template.persistence
com.topcoder.project.phases.template.startdategenerator
com.topcoder.project.phases.template.converter

3.2 Configuration Parameters

3.2.1 Configuration for XmlPhaseTemplatePersistence

Parameter	Description	Values
template_files	A list of template files from which the templates will be loaded	XML persistence file names. **Required**

3.2.2 Configuration for RelativeWeekTimeStartDateGenerator

Parameter	Description	Values
week_offset	The week offset of the date to generate, from current week.	Any integer value.e.g. 0 means current week, 1 means the next week **Required**
day_of_week	the day of the week	Integer values from 1 to 7, representing SUNDAY to SATURDAY respectively. **Required.**
hour	hour in 24-hour clock	Integer values in [0, 23]. **Required**
minute	Minute value	Integer values in [0, 59] **Required**
second	Second value	Integer values in [0, 59] **Required**

3.2.3 Configuration for DefaultPhaseTemplate

Parameter	Description	Values
persistence	Configuration property for persistence, should have “class” and “namespace” sub-properties	See persistence.class and persistence.namespace
persistence.class	class of the PhaseTemplatePersistence	Full qualified class name **Required**
persistence.namespace	namespace from which the PhaseTemplatePersistence will be created	The configuration namespace **Optional**, if not provided, no arg ctor will be used to create the persistence
start_date_generator	Configuration property for start date generator, should have “class” and “namespace” sub-properties	See start_date_generator.class and start_date_generator.namespace

start_date_generator.class	class of the StartDateGenerator	Full qualified class name **Required**
start_date_generator.namespace	namespace from which the StartDateGenerator will be created	The configuration namespace **Optional**, if not provided, no arg ctor will be used to create the startDateGenerator.
workdays	Configuration property for workdays variable, should have “object_specification_namespace” and “object_key” sub-properties, “object_identifier” sub-property is optional.	See workdays. object_specification_namespace, workdays.object_key and workdays.object_identifier
workdays. object_specification_namespace	Configuration namespace of the object specification configuration of the workdays	The configuration namespace. **Required**
workdays.object_key	key of the workdays object	The key of the workdays object in the Object specification configuration, **Required**
workdays.object_identifier	Identifier of the workdays object	The key of the workdays object in the Object specification configuration, **Optional.** If missing, the workdays will be created without identifier.

3.2.4 Object Specification Configuration for Workdays

```
<CMConfig>
  <Config name="com.topcoder.project.phases.template.DefaultPhaseTemplate.workdays">
    <Property name="workdays:default">
      <Property name="type">
        <Value>com.topcoder.date.workdays.DefaultWorkdays</Value>
      </Property>
      <Property name="params">
        <Property name="param1">
          <Property name="type">
            <Value>String</Value>
          </Property>
          <Property name="value">
            <Value>test_files/workdays.xml</Value>
          </Property>
        </Property>
        <Property name="param2">
          <Property name="type">
            <Value>String</Value>
          </Property>
          <Property name="value">
            <Value>XML</Value>
          </Property>
        </Property>
      </Property>
    </Property>
  </Config>
</CMConfig>
```

3.2.5 XML phases template format

In the version 1.1 Phase element can have new category, isDefault, creationDate and description attributes. All these attributes are optional. Category attribute must be integer, when it's not specified then assume that template belongs to zero-category (category = 0). IsDefault attribute specifies whether the current template is default for its category. The user is responsible to ensure that only one template (or none) is specified as a default for each category. CreationDate and description attributes can contain template creation date in format "yyyy-MM-dd HH:mm:ss" and template description correspondingly. In the new version Phase element can contain an optional phaseId attribute value of which must be an integer.

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<xs:schema xmlns:xs=http://www.w3.org/2001/XMLSchema
          elementFormDefault="qualified">
  <xs:element name="Template">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="PhaseTypes"/>
        <xs:element ref="Phases"/>
      </xs:sequence>
      <xs:attribute name="name" use="required"/>
      <xs:attribute name="category" use="optional" type="xs:integer"/>
      <xs:attribute name="isDefault" use="optional" type="xs:boolean"/>
      <xs:attribute name="creationDate" use="optional"/>
      <xs:attribute name="description" use="optional"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="PhaseTypes">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" ref="PhaseType"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="PhaseType">
    <xs:complexType>
      <xs:attribute name="id" use="required" type="xs:NCName"/>
      <xs:attribute name="typeId" use="required" type="xs:integer"/>
      <xs:attribute name="typeName" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="Phases">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" ref="Phase"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="Phase">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="0" ref="Dependency"/>
      </xs:sequence>
      <xs:attribute name="id" use="required" type="xs:NCName"/>
      <xs:attribute name="phaseId" use="optional" type="xs:integer"/>
      <xs:attribute name="length" use="required" type="xs:integer"/>
      <xs:attribute name="type" use="required" type="xs:NCName"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="Dependency">
    <xs:complexType>
      <xs:attribute name="id" use="required" type="xs:NCName"/>
      <xs:attribute name="isDependencyStart" use="optional" type="xs:boolean"/>
      <xs:attribute name="isDependentStart" use="optional" type="xs:boolean"/>
      <xs:attribute name="lagTime" use="optional" type="xs:integer"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

3.2.6 Configuration for DBPhaseTemplatePersistence

Parameter	Description	Values
connection_name	A String, which represents the database connection name	String **Optional**
connection_factory_class_name	A String, which represents the class name, which is used to create the database connection	String **Required**
object_factory_namespace	A String, which represents the object factory namespace	String **Required**

3.3 Dependencies Configuration

All dependency components must be properly configured. See their docs for details.

4. Usage Notes

- λ Extract the component distribution.
- λ Follow Dependencies Configuration.
- λ Execute ‘ant test’ within the directory that the distribution was extracted to.

4.1 Required steps to test the component

Preload the configuration file into Configuration Manager. Follow demo.

4.2 Required steps to use the component

Preload the configuration file into Configuration Manager. Follow demo.
Please see the configuration files in docs directory for information on how to configure this component.

4.3 Demo

4.3.1 Create the PhaseTemplate instance from configuration

```
DefaultPhaseTemplate template = new
    DefaultPhaseTemplate(
        "com.topcoder.project.phases.template.DefaultPhaseTemplate");
```

4.3.2 Create the PhaseTemplate instance with specific PhaseTemplatePersistence and StartDateGenerator.

```
// create PhaseTemplatePersistence instance
PhaseTemplatePersistence persistence = new
    XmlPhaseTemplatePersistence(
        "com.topcoder.project.phases.template." +
        "persistence.XmlPhaseTemplatePersistence");
// create StartDateGenerator instance
StartDateGenerator startDateGenerator = new
    RelativeWeekTimeStartDateGenerator(
        "com.topcoder.project.phases.template." +
        "startdategenerator.RelativeWeekTimeStartDateGenerator");
// create PhaseTemplate instance with the persistence and startDateGenerator
DefaultPhaseTemplate template =
    new DefaultPhaseTemplate(persistence, startDateGenerator, new DefaultWorkdays());
```

4.3.3 Apply a template to generate project phases

```
// apply a template with name "TCS Component Project" with a given start date
Project project = template.applyTemplate("TCS Component Project",
                                         Calendar.getInstance().getTime());
// apply a template with name "TCS Component Project" without a specific start date
project = template.applyTemplate("TCS Component Project");

// apply a template with name "TCS Component Project" with a given start date,
// adjust the length of the phase with id=2 to make the phase with id=1 start
// a fixed startTime
project template.applyTemplate("TCS Component Project", 2, 1, startTime,
                               Calendar.getInstance().getTime());

// apply a template with name "TCS Component Project" with auto detected start date,
// adjust the length of the phase with id=2 to make the phase with id=1 start
// a fixed startTime
project template.applyTemplate("TCS Component Project", 2, 1, startTime);
```

4.3.4 Retrieve names of all available templates

```
// retrieve names of all available templates
String[] templateNames = template.getAllTemplateNames();
```

4.3.5 Change template persistence or start date generator dynamically

```
// change template persistence(say we have a SQLPhaseTemplatePersistence
template.setPersistence(new SQLPhaseTemplatePersistence(
  "com.topcoder.project.phases.template.persistence.SQLPhaseTemplatePersistence"));
// change start date generator(say we have a RelativeMonthTimeStartDateGenerator)
template.setStartDateGenerator(new RelativeMonthTimeStartDateGenerator(
    "com.topcoder.project.phases.template.startdategenerator." +
    "RelativeMonthTimeStartDateGenerator"));
```

4.3.6 Create the PhaseTemplate instance with use of specific DBPhaseTemplatePersistence and StartDateGenerator.

```
// create DBPhaseTemplatePersistence instance
PhaseTemplatePersistence persistence = new
```

```
    DBPhaseTemplatePersistence("com.topcoder.project.phases.template." +
    "persistence.DBPhaseTemplatePersistence");
// create StartDateGenerator instance
StartDateGenerator startDateGenerator = new
    RelativeWeekTimeStartDateGenerator("com.topcoder.project.phases.template." +
        "startdategenerator.RelativeWeekTimeStartDateGenerator");
// create PhaseTemplate instance with the persistence and startDateGenerator
DefaultPhaseTemplate template =
    new DefaultPhaseTemplate(persistence, startDateGenerator, new DefaultWorkdays());
```

4.3.7 Retrieve names of all available templates from specific category

```
// retrieve names of all available templates from category 2
String[] templateNames = template.getAllTemplateNames(2);
```

4.3.8 Retrieve the name of the default template for specific category

```
// retrieve the name of the default template for category 2
String templateName = template.getDefaultTemplateName(2);
```

4.3.9 Retrieve template information

```
// retrieve the category of the template with name "New_Design"
int category = template.getTemplateCategory("New_Design");
// retrive the description of the template with name "New_Design"
String description = template.getTemplateDescription("New_Design");
// retrive the creation date of the template with name "New_Design"
Date creationDate = template.getTemplateCreationDate("New_Design");
```

4.3.10 Use XML templates migration utility

The following command line can be used to output SQL queries those are intended for fulfilling an empty database:

```
java ConverterUtility.class -inamespace xml_phase_template_namespace
```

The following command line can be used to output SQL queries those are intended for adding templates to non-empty database:

```
java ConverterUtility.class -inamespace xml_phase_template_namespace -onamespace
db_phase_template_namespace
```

The following command line can be used to fulfill a database (queries are executed automatically):

```
java ConverterUtility.class -inamespace xml_phase_template_namespace -onamespace
db_phase_template_namespace -auto
```

If users want to specify the config files in the command line instead of ConfigManager.properties,
they can use the following command line(suppose the xml_phase_template_namespace is in file “xml_config.xml”, and db_phase_template_namespace is in "db_config.xml"):

```
java ConverterUtility.class -inamespace xml_phase_template_namespace -onamespace
db_phase_template_namespace -ifile xml_config.xml -ofile db_config.xml -auto
```

4.3.11 Delete a template from database

```
// create DBPhaseTemplatePersistence instance
DBPhaseTemplatePersistence dbPersistence = new
    DBPhaseTemplatePersistence(
        "com.topcoder.project.phases.template." +
        "persistence.DBPhaseTemplatePersistence");
// delete the template with name "New_Design"
dbPersistence.removeTemplate("New_Design");
```

4.3.12 XML-to-DB conversion sample

This section contains a sample XML phase template file content and a database content which must be generated by migration utility.

Xml_Phase_Template_Persistence.xml

```
<?xml version="1.0"?>
<CMConfig>
  <Config name=
    "com.topcoder.project.phases.template.persistence.DemoXmlPhaseTemplatePersistence">
    <!-- XML files from which the templates will be loaded. -->
    <Property name="template_files">
      <Value>My_Project.xml</Value>
    </Property>
  </Config>
</CMConfig>
```

My_Project.xml

```
<?xml version="1.0"?>
<!-- A template which defines a set of project phases -->
<Template name="Design" category="1" isDefault="true"
          creationDate="2007.12.04 14:45:02"
          description="This is a design phases template">
  <!-- An enumeration of project phase types defined in this template -->
  <PhaseTypes>
    <PhaseType id="firstPhaseType" typeId="1" typeName="FirstPhaseType"/>
    <PhaseType id=”secondPhaseType" typeId="2" typeName="SecondPhaseType"/>
  </PhaseTypes>
  <!-- A set of project phases defined in this template. -->
  <Phases>
    <Phase id="firstPhase" phaseId="1" length="604800000" type="firstPhaseType"/>
    <Phase id="secondPhase" phaseId="2" length="86400000" type="secondPhaseType">
      <Dependency id="firstPhase" isDependencyStart="false" isDependentStart="true" lagTime="0"/>
    </Phase>
  </Phases>
</Template>
```

ProjectPhaseTemplate database

Table “template”

id	category	name	creation_date	description
1	1	Design	2007.12.04 14:45:02	This is a design phases template

Table “default_template”

id	template_id	category
1	1	1

Table “phase_type”

id	template_id	type_id	name
1	1	1	FirstPhaseType
2	1	2	SecondPhaseType

Table “phase”

id	template_id	phase_type_id	phase_id	time_length
1	1	1	1	604800000
2	1	2	2	86400000

Table “dependency”

id	dependent_id	dependency_id	dependent_start	dependency_start	lag_time
1	2	1	t	f	0

4.3.13 *Generate project with phase being left out*

Assume that persistence contains a template definition described in the section 4.3.12. Then the following call can be used to create a project with only one phase (with ID=1) configured (with phase with ID=2 left out):

```
// apply a template with name "TCS Component Project" with a given start date
// left out a phase with ID=2
project = template.applyTemplate("TCS Component Project",
                new long[]{2}, Calendar.getInstance().getTime());
// project will contain only one phase (with ID=1) and no phase dependencies
```

Note that when {1} is passed to this method as leftOutPhaseId, an exception should be thrown since phase 2 depends on phase 1. But it’s possible to pass {1, 2} as leftOutPhaseIds. In this case a project without any phases will be created.

4.3.14 *Samples of “redirection” of dependencies to left out phases*

Assume that we have phases A, B, C, D, E and F with the following dependencies:
Phase C depends on phases A and B.
Phase D depends on phase C.
Phase E depends on phase D.
Phase F depends on phase D.

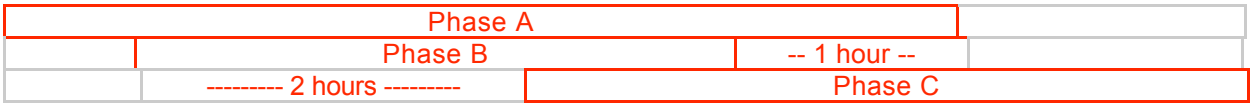
In this case if phases C and D are left out, this component must create phases A, B, E and F with the following dependencies:
Phase E depends on phases A and B.
Phase F depends on phases A and B.

The redirection logic is the following:
Since phase D is left out, dependency from E to D can be transformed into dependency from E to the phase which phase D depends on – i.e. phase C. In turn, since phase C is also left

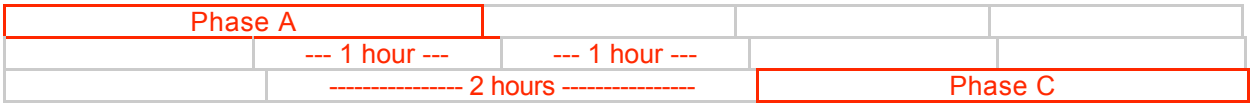
out, dependency from E to C is transformed into dependencies from E to all phases which C depends on, i.e. from E to A and from E to B. The same applies to dependencies of phase F.

One more example with different dependency types:
Assume that we have phases A, B, C with the following dependencies:
Phase A ends in 1 hour after phase B ends.
Phase B starts 2 hours before phase C starts.

We can show this schematically:



Then after phase B is left out, phase A should depend on phase C:



I.e. phase A should end 1 hour before phase C starts.

To understand why we get this result try to imagine that the length of phase B became equal to 0.

5. Future Enhancements

- λ Implement new StartDateGenerator implementations to add new project start date generation logic.
- λ Add template authoring/editing functionality
- λ Enhance the XmlPhaseTemplatePersistence to support dynamic template management (adding/removing).
- λ Enhance the DBPhaseTemplatePersistence to support dynamic template management (adding).