



ARIZONA STATE UNIVERSITY

GENERAL STUDIES COURSE PROPOSAL COVER FORM

Course information:

Copy and paste current course information from Class Search/Course Catalog.

Academic Unit College of Technology & Innovation Department Engineering and Computing Systems
Subject SER Number 216 Title Software Enterprise II: Testing and Quality Units: 3
Is this a cross-listed course? No
If yes, please identify course(s)
Is this a shared course? No If so, list all academic units offering this course
Course description:

Requested designation: Literacy and Critical Inquiry-L
Note- a separate proposal is required for each designation requested

Eligibility:

Permanent numbered courses must have completed the university's review and approval process.
For the rules governing approval of omnibus courses, contact the General Studies Program Office at (480) 965-0739.

Area(s) proposed course will serve:

A single course may be proposed for more than one core or awareness area. A course may satisfy a core area requirement and more than one awareness area requirements concurrently, but may not satisfy requirements in two core areas simultaneously, even if approved for those areas. With departmental consent, an approved General Studies course may be counted toward both the General Studies requirement and the major program of study.

Checklists for general studies designations:

- Complete and attach the appropriate checklist
Literacy and Critical Inquiry core courses (L)
Mathematics core courses (MA)
Computer/statistics/quantitative applications core courses (CS)
Humanities, Fine Arts and Design core courses (HU)
Social and Behavioral Sciences core courses (SB)
Natural Sciences core courses (SQ/SG)
Global Awareness courses (G)
Historical Awareness courses (H)
Cultural Diversity in the United States courses (C)

A complete proposal should include:

- Signed General Studies Program Course Proposal Cover Form
Criteria Checklist for the area
Course Syllabus
Table of Contents from the textbook and list of required readings/books

Contact information:

Name Dr. Timothy Lindquist Phone 480-727-2783
Mail code E-mail: Timothy.Lindquist@asu.edu

Department Chair/Director approval: (Required)

Chair/Director name (Typed): Dr. Ann McKenna Date: 10/10/13
Chair/Director (Signature):

SER 216 - Software Enterprise II: Testing & Quality

Course description: Project-centered course covering testing and quality in software engineering; concepts, tools, and methods in testing and quality management; teamwork and communication in software engineering. Project based.

Arizona State University Criteria Checklist for
LITERACY AND CRITICAL INQUIRY - [L]

Rationale and Objectives

Literacy is here defined broadly as communicative competence in written and oral discourse. **Critical inquiry** involves the gathering, interpretation, and evaluation of evidence. Any field of university study may require unique critical skills which have little to do with language in the usual sense (words), but the analysis of spoken and written evidence pervades university study and everyday life. Thus, the General Studies requirements assume that all undergraduates should develop the ability to reason critically and communicate using the medium of language.

The requirement in Literacy and Critical Inquiry presumes, first, that training in literacy and critical inquiry must be sustained beyond traditional First Year English in order to create a habitual skill in every student; and, second, that the skills become more expert, as well as more secure, as the student learns challenging subject matter. Thus, the Literacy and Critical Inquiry requirement stipulates two courses beyond First Year English.

Most lower-level [L] courses are devoted primarily to the further development of critical skills in reading, writing, listening, speaking, or analysis of discourse. Upper-division [L] courses generally are courses in a particular discipline into which writing and critical thinking have been fully integrated as means of learning the content and, in most cases, demonstrating that it has been learned.

Students must complete six credit hours from courses designated as [L], at least three credit hours of which must be chosen from approved upper-division courses, preferably in their major. Students must have completed ENG 101, 107, or 105 to take an [L] course.

Notes:

1. ENG 101, 107 or ENG 105 must be prerequisites
2. Honors theses, XXX 493 meet [L] requirements
3. The list of criteria that must be satisfied for designation as a Literacy and Critical Inquiry [L] course is presented on the following page. This list will help you determine whether the current version of your course meets all of these requirements. If you decide to apply, please attach a current syllabus, or handouts, or other documentation that will provide sufficient information for the General Studies Council to make an informed decision regarding the status of your proposal.

Proposer: Please complete the following section and attach appropriate documentation.

ASU - [L] CRITERIA			
TO QUALIFY FOR [L] DESIGNATION, THE COURSE DESIGN MUST PLACE A MAJOR EMPHASIS ON COMPLETING CRITICAL DISCOURSE--AS EVIDENCED BY THE FOLLOWING CRITERIA:			
YES	NO		Identify Documentation Submitted
<input checked="" type="checkbox"/>	<input type="checkbox"/>	CRITERION 1: At least 50 percent of the grade in the course should depend upon writing, including prepared essays, speeches, or in-class essay examinations. <i>Group projects are acceptable only if each student gathers, interprets, and evaluates evidence, and prepares a summary report</i>	Course Syllabus, Assignment Description & Grading Rubric
1. Please describe the assignments that are considered in the computation of course grades--and indicate the proportion of the final grade that is determined by each assignment.			
2. Also: <div style="border: 1px solid black; border-radius: 50%; padding: 10px; text-align: center; margin: 10px 0;"> <p style="background-color: yellow;">Please circle, underline, or otherwise mark the information presented in the most recent course syllabus (or other material you have submitted) that verifies this description of the grading process--and label this information "C-1".</p> </div> <p style="text-align: center;">C-1</p>			
<input checked="" type="checkbox"/>	<input type="checkbox"/>	CRITERION 2: The composition tasks involve the gathering, interpretation, and evaluation of evidence	Testing tools survey activity of Final Project (described in CourseProjectDescription)
1. Please describe the way(s) in which this criterion is addressed in the course design			
2. Also: <div style="border: 1px solid black; border-radius: 50%; padding: 10px; text-align: center; margin: 10px 0;"> <p style="background-color: yellow;">Please circle, underline, or otherwise mark the information presented in the most recent course syllabus (or other material you have submitted) that verifies this description of the grading process--and label this information "C-2".</p> </div> <p style="text-align: center;">C-2</p>			
<input checked="" type="checkbox"/>	<input type="checkbox"/>	CRITERION 3: The syllabus should include a minimum of two substantial writing or speaking tasks, other than or in addition to in-class essay exams	Course Syllabus
1. Please provide relatively detailed descriptions of two or more substantial writing or speaking tasks that are included in the course requirements			
2. Also: <div style="border: 1px solid black; border-radius: 50%; padding: 10px; text-align: center; margin: 10px 0;"> <p style="background-color: yellow;">Please circle, underline, or otherwise mark the information presented in the most recent course syllabus (or other material you have submitted) that verifies this description of the grading process--and label this information "C-3".</p> </div> <p style="text-align: center;">C-3</p>			

ASU - [L] CRITERIA			
YES	NO		Identify Documentation Submitted
<input checked="" type="checkbox"/>	<input type="checkbox"/>	<p>CRITERION 4: These substantial writing or speaking assignments should be arranged so that the students will get timely feedback from the instructor on each assignment in time to help them do better on subsequent assignments. <i>Intervention at earlier stages in the writing process is especially welcomed</i></p>	<p>GradingRubric is used for all activities. These are used for grading and additional feedback will be provided to the students within 2 weeks of assignment submission (attached and referenced rubrics and course syllabus). The project has multiple deliverables that occur at different times during the second half of the semester. The project description clearly states the deliverables and due dates for each task. The reading & summarizing assignments, homework assignments happen during the first half of the semester.</p>
<p>1. Please describe the sequence of course assignments--and the nature of the feedback the current (or most recent) course instructor provides to help students do better on subsequent assignments</p>			
<p>2. Also:</p> <div style="border: 1px solid black; border-radius: 50%; padding: 10px; text-align: center;"> <p style="background-color: yellow;">Please circle, underline, or otherwise mark the information presented in the most recent course syllabus (or other material you have submitted) that verifies this description of the grading process--and label this information "C-4".</p> </div>			
C-4			

Course Prefix	Number	Title	Designation
SER	216	Software Enterprise II: Testing & Quality	L

Explain in detail which student activities correspond to the **specific** designation criteria.
Please use the following organizer to explain how the criteria are being met.

Criteria (from checksheet)	How course meets spirit (contextualize specific examples in next column)	Please provide detailed evidence of how course meets criteria (i.e., where in syllabus)
C1	<p>51% of the grade is towards Literacy and Critical Inquiry activities. These activities are as follows: Readings (include writing a summary) (6%) Homework Assignments (10%) Course Project (35%)</p> <p>The readings & summarizing assignments, homework assignments happen during the first half of the semester.</p>	<p>Activity description and grading rubric attached.</p> <p>Course syllabus is also attached.</p> <p>The homework assignment includes 2 labs that require students to fill out a number forms and use templates provided to document the process followed for implementing the specified program. The lab description is attached with highlighting that indicates these tasks.</p>
C2	<p>As part of the Course project, students have a to find, use, and evaluate a number of softare testing tools. They write an evaluation report on one of the tools and present the evaluation of a second tool to the class. This activity addresses the criteria about having a composition task that involves gathering, interpretation, and evaluation of evidence. The project has multiple deliverables that occur at different times during the second half of the semester. The project description clearly states the deliverables and due dates for each task.</p>	<p>Course project description and grading rubric for presentation & testing tool report and software test plan template are attached.</p>
C3	<p>This course has three substantial writing assignments (listed below): 1. Reading, critiquing, and summarizing published articles 2. Creating a test plan for a given software product 3. Writing a report on a software testing tool</p>	<p>Activity descriptions along with grading rubric are attached.</p>

SER 294/216 – Software Enterprise II (Testing & Quality) Spring 2014

Summary:

Project-centered course covering testing and quality in software engineering; concepts, tools, and methods in testing and quality management; teamwork and communication in software engineering.

Course Details:

Time: Tuesday & Thursday 3:00pm - 4:15pm

Location: Peralta 213

Credits: 3

Pre-requisites: SER 215;

Course Description:

SER 216 is the second course taught in the Software Enterprise; Students learn in a hybrid lecture-lab-project environment in which concepts are learned in a project context. Projects are team-based and include multiple deliverables and presentations, with a specific emphasis on testing, validation, and quality assurance.

Instructor:

Name: Srividya Bansal

Email: srividya.bansal@asu.edu

Office: Peralta 230G

Office Hours: Tue: 1:30pm-2:30pm; Thu: 1:30 - 2:30pm; and by appointment on other days.

Teaching Assistant:

Name: TBA

Email: TBA

Office: Peralta 235

Office Hours: TBA

Grading:

Assessment Type	Weight	Points
Mid-term test	15%	150
HWs/Assignments → C-3	35%	350 (7 assignments; 50 points each) 2 assignments satisfy L component (PSP and Test Case generation assignments) comprising 10% of the grade → C-1
Readings/Quizzes → C-3	10%	100 (5 readings; 20 points each) 3 readings satisfy L component comprising 6% of the grade → C-1
Final Project → C-2, C-3	40%	400 A number of activities in the project satisfy L component and comprise 35% of the final grade → C-1
Total		1000 points

Text Book:

["Software Engineering"](#) (9th edition) by Ian Sommerville; Publisher: Addison Wesley; ISBN-10: 0137035152; ISBN-13: 978-0137035151;

Reference Books:

["Software Testing: Principles and Practices"](#) by Srinivasan Desikan, Gopalaswamy Ramesh; Publisher: Addison Wesley; ISBN-10: 817758295X; ISBN-13: 978-8177582956

["Introduction to Java Programming, Comprehensive Version"](#) (8th edition) by Y. Daniel Liang; Publisher: Prentice Hall; ISBN-10: 0136012671; ISBN-13: 978-0136012672;

Weekly Course Schedule (Tentative):

<i>Week #</i>	<i>Lecture Content</i>
Week 1	Class Overview & Introduction
Week 2	Software Planning (PSP1)
Week 3	Software Quality - reviews and inspections (PSP2)
Week 4	Team Software Process (TSP)
Week 5	UML (Use case, Class)
Week 6	UML (Activity, State diagrams)

Week 7	Software Maintenance and Evolution
Week 8	Principles of Software Testing
Week 9	Unit Testing
Week 10	White Box and Black Box testing
Week 11	Tools for Testing
Week 12	Software Test Plan
Week 13	Integration testing
Week 14	System and Acceptance Testing

Course Policy:

- Students are expected to participate in the educational process and not be a disruptive element with regard to the learning of others. Safety, self-discipline and respect for others are necessary elements in the educational processes employed in this course. All students should be familiar with the Student Code of Conduct, which can be found at <http://www.asu.edu/studentlife/judicial/>.
- Ample time will be provided to complete homework assignments. The assignments should be turned in by the specified deadline. Late programming assignments will not be accepted unless prior arrangements have been made with the instructor. The only legitimate reasons are business or university related travel or illness for more than half the assignment period with appropriate documentation.
- Cell phones must be either set to vibrate, turn the ringer volume off, or turn off the phone completely. Use of computer or cell phone for chat, texting, and personal (non-emergency) calls are not allowed. You will be marked absent from class if found using computers or cell phones for these activities during class.
- It is the student's responsibility to keep a backup of all your assignments and projects.
- Feedback on assignments will be provided within 2 weeks of submission. Grading rubric will be provided for the assignments along with the specification. This rubric sheet along with additional feedback on student's work will be provided. Students have the right to appeal a grade in writing. Submit your typed appeal with the graded item, stating the reason for your appeal. All appeals must be turned in no later than one week after the material has been returned in class. → C-4
- Any students who need special needs or accommodations in this course are encouraged to communicate these as soon as possible to make appropriate arrangements for these accommodations.

Course Ethics:

Plagiarism or academic dishonesty in any form will not be tolerated. Punishment can include a record on the student's transcripts, an E in the course, and/or dismissal from the department.

The following exemptions are valid for this course:

- You can discuss the homework with other students. ***But you are not allowed to copy someone else's code.***
- You are encouraged to help other students fix their syntax errors.
- You can discuss the methods and the algorithm with other students. ***But do not write the code (share the code) with the other students.***

All the code you submit must be yours. A software tool may be used at times to check for similarities between submitted assignments. In this class, any cases of suspected violations will be turned over to the department who will track violations and determine additional punishment for students and repeat offenders. Punishment can include a record on the student's transcripts, an E in the course, and/or dismissal from the department. ASU's academic integrity policies (<http://provost.asu.edu/academicintegrity>) and the ASU Student Code of Conduct are provided on ASU's website. If you are not sure if something is really cheating, ask your professor.

Student learning outcomes:

Students completing SER216 will be able to:

1. appreciate the need for Software quality assurance
2. evaluate various software testing tools → C-3
3. learn fundamentals of Software testing
 - a. describe error, fault, failure, debugging, and validation correctly
 - b. create test cases in the correct format
 - c. use and demonstrate unit testing with JUnit effectively. understand and use code coverage tools such as EclEmma effectively
4. understand Integration, System, Acceptance testing
 - a. explain various approaches for Integration testing
 - b. explain the similarities and differences between integration and system testing
5. generate test cases using boundary value analysis and equivalence partitioning
6. create UML use case, class, state, and activity diagrams
7. design a Software test plan in the IEEE template format and conduct testing for a given Software product → C-3
 - a. choose appropriate approaches for functional testing
 - b. choose an appropriate approach for integration testing
 - c. choose appropriate tools for test automation
 - d. decide correctly which parts of the software product will be manually tested and which parts will be tested using automation
8. demonstrate working effectively in small teams
9. communicate effectively in writing a technical document, evaluating and presenting software testing tools, and project presentation → C-3

Mapping of Program outcomes to student learning outcomes in this course:

Program outcomes	SER 216 learning outcomes
Technical outcomes – Software design and process	Outcomes 2,3,4,5,6,7
Computing practice, Critical thinking, Decision-making	Outcomes 2,3,4,5,6,7
Problem-solving	Outcomes 5,7
Perspective	Outcomes 1
Professionalism, Communication	Outcomes 8, 9

Mapping of Course topics to student learning outcomes in this course:

#	Topics	Supports learning outcomes
1	Software Planning	Outcomes 1
2	Software Quality	Outcomes 1,3
3	UML Modeling	Outcomes 6
4	Software Maintenance	Outcomes 7
5	Principles of Software Testing	Outcomes 3
6	Unit Testing	Outcomes 3,5,7
7	Functional Testing (Black box & White box)	Outcomes 3,5,7
8	Integration, System, Acceptance Testing	Outcomes 4,7
9	Software Testing tools	Outcomes 3,7
10	Software Test Plan	Outcomes 7

Mapping of Assessments to student learning outcomes in this course:

#	Assessments	Assesses learning outcomes
1	Software Planning - Lab	Outcomes 1
2	Design review - Lab	Outcomes 1, 3
3	Code review & Defect tracking - Lab	Outcomes 1, 3
4	UML: Use Case & Class diagram - Lab	Outcomes 6
5	UML: Activity & State diagram - Lab	Outcomes 6
6	Unit Testing: JUnit - Lab	Outcomes 2, 3.c
7	Testing & Debugging - Lab	Outcomes 3.a, 3.b
8	Code Coverage: EclEmma - Lab	Outcomes 2, 3.d
9	UML Quiz	Outcome 6
10	Unit Testing Quiz	Outcome 3.c
11	Semester Project (Summative)	Outcomes 1, 2, 5, 7, 8, 9
12	Mid-term test (Summative)	Outcomes 3, 4, 5, 6
13	Survey, Evaluate, & Present 2 testing tools	Outcomes 2, 8, 9

SER 216 Software Enterprise II: Testing and Quality

Summary of assignments related to literacy requirements/criteria. Criterion 4 is addressed in separate documents:

Writing Assignment: Reading on Software Review Process (C-1, C-3)

Each student in the class is required to complete the following assignment:

Read the article: "Painless improvements to the review process by IIsakka and Tervonen", Software Quality Journal 7 (1998), pp. 11-20 (available on Blackboard).

Summarize the article and evaluate its applicability to SER216 software projects. Your description should include 1-2 pages of narrative and be structured and formatted as discussed in class. Grading will be done using one of the referenced rubrics. Points: 20 points (2% of course grade.)

Writing Assignment: Reading on Finding Bugs Using Static Analysis (C1, C3)

Read the article: "Using Static Analysis to Find Bugs by Nathaniel Ayewah, William Pugh, David Hovemeyer, David Morgenthaler, and John Penix" (available on the course Blackboard site).

Summarize the article and describe the applicability of the approach to finding bugs in your SER216 software development project. The narrative should be at least 2 pages in length. Grading will be done using one of the referenced rubrics. Points: 20 points (2% of course grade.)

Writing Assignment: Reading on Unit Testing (C1, C3)

Read the following article on "The Art of Unit Testing by Manning and Osherove" (available on the SER216 Blackboard site).

Summarize the article within 1-2 pages of narrative, including a description of how your team used this approach in testing your semester project. Grading will be done using the references rubric. Points: 20 points (2% of course grade.)

Presentation & Writing Assignment (Semester Project) (C1,C2).

Student teams are required to select and study two testing tools from a list provided. They are required to write a report and provide the class with a 20 minute presentation about the tool. Each member of the team must present at least one aspect of the tool. The presentation includes the following topics: (a) What testing purpose does it serve? (b) What programming languages and platforms does it support? (c) What is the cost and how can it be acquired? (d) Strengths and limitations of the tool (e) Usage, setup, and configuration of the tool (f) Demonstrations of the tool. Points: 80 points (10% of course grade.)

Presentation Assignment: Semester Project Presentation (C2).

Student teams are required to provide the class with a 20 minute presentation of the results of their semester project. Each member of the team must present at least one aspect of the presentation. The presentation includes the following topics: (a) Description and demonstration of software developed, (b) Software system design, (c) Software quality and testing requirements utilized in developing the project, (d) Defect analysis by software development phase, (e) Retrospect of the project. Points: 40 points (5% of course grade.)

Writing Assignment: Software Test plan (Semester Project) (C1, C2).

Student teams are required to create a test plan for a given project that includes a detailed

description of various testing activities (e.g., individual test cases), different types of testing, tools used for testing, description of coverage of the testing effort, etc. Points: 100 points (15% of course grade.)

Grading of written and oral presentations utilize widely available rubrics for writing and presentations such as:

Rubrics for Evaluating Writing:

http://www.readwritethink.org/files/resources/lesson_images/lesson782/Rubric.pdf

<http://www.smarterbalanced.org/wordpress/wp-content/uploads/2012/05/TaskItemSpecifications/EnglishLanguageArtsLiteracy/ELARubrics.pdf>

<http://www.middleweb.com/wp-content/uploads/2013/04/Student-Friendly-Writing-Rubric.pdf>

Oral Presentations

<http://pooh.poly.asu.edu/Ser401/ClassNotes/RubricPresentation.html>

http://www.readwritethink.org/files/resources/lesson_images/lesson416/OralRubric.pdf

Software Enterprise II - Course Project

Software Maintenance, Testing & Quality

Instructions:



Through this project you will learn about Software Testing (various tools, creating test plans, performing testing) and Software Maintenance activities of the software development lifecycle. You will work in groups of 2-3. Your project submissions should be made via Blackboard before the specified deadline. You will work through this project in multiple iterations. There are deliverables at the end of each iteration. Each team will be provided existing software products (an open-source product, a research software tool, a software game, etc.) that you will work on for Testing and Maintenance.

1: *Testing tools survey*

Each team is required to select TWO testing tools from the list provided, each from a different category of tools (such as Unit Testing, Functional Testing, Defect tracking, Performance, etc.). A tool not listed here may also be selected upon approval from the instructor. A consolidated list of open source software testing tools is available at <http://www.opensourcetesting.org/>. Each team will present one of the two tools to the class and will provide a written description for the other one via Blackboard. For each tool, you must download the tool and learn to use it. Your papers must answer the questions below in the order given.

1. Who (company or individual) developed the tool? What is the cost to the tool user? How do you acquire it?
2. What testing purpose does the tool serve? (i.e, what problem does it attempt to solve? How does it improve productivity?)
3. What programming language(s) does the tool support, if any?
4. In what phase of software testing is the tool useful?
5. What do you need to do in order to use the tool?
 - How do you install it?
 - How do you configure it?
 - How do you use it?
6. What are the strengths and limitations of the tool?

Deliverables:

- **Written Report on one of the selected tools (submit via Blackboard)** 
- **Presentation to class on the second tool selected (in class)** 

Due Date: April 4, 2014 (11:59pm) Presentations in class during first week of April.

2: Setup and Execute the project

- Download the source code provided, create an Eclipse project and build it
- Understand how the software works
- Use the software as an end-user

Deliverables:

none

Due Date: April 4, 2014 (11:59pm)

3: Understand the code

- Analyze the code
- Create use case diagram and class diagram for the existing source code. Capture all classes in the code base. Use a UML tool of your choice to create these diagrams.

Deliverables:

Use Case Diagram and Class Diagram (via Blackboard)

Due Date: April 11, 2014 (11:59pm)

4: Create a Test Plan

- Create a test plan that includes a detailed description of the testing activities (e.g., individual test cases), a description of the coverage of the testing effort, etc. A template will be provided in class.

**Deliverables:****Submit a Test Plan document** (via Blackboard)**Due Date: April 23, 2014 (11:59pm)**

5: Perform Testing as per your plan

- Perform testing and create a list of bugs and possible enhancements. From this list you will identify a few bugs and enhancements (after discussing with your instructor) that you will fix .

Deliverables:

List of Bugs and Enhancements (via Blackboard)

Due Date: April 23, 2014 (11:59pm)

6: Project presentation

Each team will present their project to the class. Your presentations must answer the following questions:

1. What is the software product your are working on? What does it do? Who is using it?
2. What are the goals of your testing efforts?
3. What is your test plan?
4. What were the results of your testing? What coverage did you achieve?
5. What did you discover?
6. What are your conclusions?
7. How would you improve your process in the future?

Deliverables:

In-class presentation - Your presentation should be about 15 minutes, and must answer the above questions. You should provide a brief, convincing demonstration of the tool.

Due Date: During last week of classes

7: Implementation of Bug fixes and enhancements

Implement bug fixes and enhancements that you have identified in the previous step and test the software thoroughly.

Deliverables:

- Compressed folder that contains the complete source code
- Executable jar file
- Summary of bug-fixes (status of bug-fixes, if an error is not fixed then explain why)

Due Date: Final Exam date (11:59pm)

Grading Rubric:

Deliverable - Testing tool presentation: 40 points
Deliverable - Testing tool report: 40 points
Deliverable - UML Diagrams: 40 points
Deliverable - Software Test plan: 100 points
Deliverable - List of Bugs/Enhancements: 60 points
Deliverable - Presentation: 40 points
Deliverable - Implementation of fixes: 40 points
Individual Contribution report: 40 points

Total: 400 points

Your Name: _____

Team #: _____

Criteria	Levels of Achievement		
	Good	Moderate	Poor
Addressed the question: Who (company or individual) developed the tool? What is the cost to the tool user? How do you acquire it?	4	2	0
Addressed the question: What testing purpose does the tool serve? (i.e., what problem does it attempt to solve? How does it improve productivity?)	4	2	0
Addressed the question: What programming language(s) does the tool support, if any?	4	2	0
Addressed the question: In what phase of software testing is the tool useful?	4	2	0
Addressed the question: What do you need to do in order to use the tool? i.e., How do you install it? How do you configure it? How do you use it?	8	4	0
Addressed the question: What are the strengths and limitations of the tool?	4	2	0
Collaboration: Did everyone contribute to the presentation? Did everyone seem well versed in the material?	4	2	0
Organization: Was the presentation well organized and easy to follow?	4	2	1
Presentation: Did presenters speak clearly, engage audience, and seem well prepared?	4	2	1

Total: _____(out of 40 points)

Name: _____

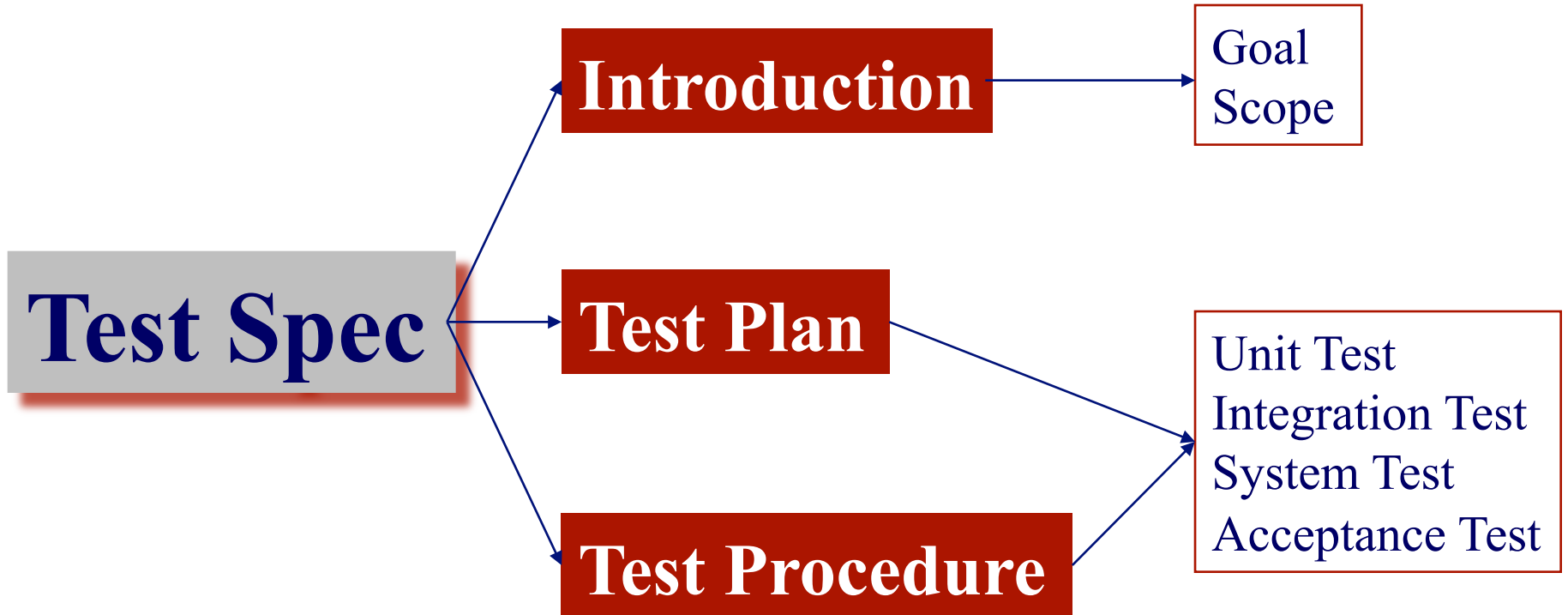
Team #: _____

Criteria	Levels of Achievement		
	Good	Moderate	Poor
Formatting	4	2	0
Organization	4	2	0
Grammar	4	2	0
Addressed the question: Who (company or individual) developed the tool? What is the cost to the tool user? How do you acquire it?	4	2	1
Addressed the question: What testing purpose does the tool serve? (i.e., what problem does it attempt to solve? How does it improve productivity?)	4	2	1
Addressed the question: What programming language(s) does the tool support, if any?	4	2	0
Addressed the question: In what phase of software testing is the tool useful?	4	2	0
Addressed the question: What do you need to do in order to use the tool? i.e., How do you install it? How do you configure it? How do you use it?	8	4	1
Addressed the question: What are the strengths and limitations of the tool?	4	2	1

Total: _____ (out of 40 points)

Software Test Plan

What Must be Included?



Test Plan – Major Items

1.0 Introduction

This section provides an overview of the entire test plan document. This document describes both the test plan and test procedures.

1.1 Goals and objectives

Overall goals and objectives of the test process are described.

1.2 Statement of scope

A description of the scope of software testing. Functionality/features/behavior to be tested is described. In addition any functionality/features/behavior that is not to be tested is also noted.

Test Plan – Major Items

1.3 Major Constraints

Any business, product line, or technical constraints that will impact the manner in which the software is to be tested are noted here.

2.0 Test Plan

This section describes the overall testing strategy and project management issues that are required to properly execute effective tests.

2.1 Software to be tested

The software to be tested are identified by name. Exclusions are noted explicitly.

2.2 Testing Strategy

The overall strategy for software testing is described.

Test Plan – Major Items

2.2.1 Unit Testing

The strategy for unit testing is described. This includes an indication of the components that will undergo unit tests or the criteria to be used to select components for unit test. Test cases are NOT included here.

2.2.2 Integration Testing

The integration testing strategy is specified. This section includes a discussion of the order of integration by software function. Test cases are not included here.

2.2.3 System Testing

The system testing strategy is specified.

2.2.4 Acceptance Testing

The validation testing strategy is specified. This section includes a discussion of the order of validation by software function. Test cases are NOT included here.

Test Plan – Major Items

3.0 Test Procedure

This section describes the detailed test procedure including test tactics and test cases for the software.

3.1 Software to be tested

The software to be tested are identified by name. Exclusions are noted explicitly.

3.2 Testing Procedure

The overall procedure for testing is described.

3.2.1 Unit Test cases

The procedure for unit testing is described for each software component (that will be unit tested). This section is repeated for all components (i).

3.2.1.1 Stubs/Drivers for component i

Test Plan – Major Items

3.2.2 Integration Testing (The integration testing procedure is specified)

3.2.2.1 Testing Procedure for integration

3.2.2.2 Stubs/Drivers required

3.2.2.3 Test cases and their purpose

3.2.2.4 Expected results

3.2.3 System Testing (The system testing procedure is specified)

3.2.3.1 Testing Procedure

3.2.3.2 Test cases and their purpose

3.2.3.3 Expected results

3.2.4 Acceptance Testing (The acceptance testing procedure is specified)

3.2.4.1 Testing Procedure

3.2.4.2 Test cases and their purpose

3.2.4.3 Expected results

Lab 3
Personal Software Process - PSP2
(Process Measurement, Software Quality & Planning)
Due Date: January 31, 2013
Submission: via Blackboard or hard-copy

Program requirements

JOLLY JUMPERS

A sequence of n integers ($n > 0$) is called a *jolly jumper* if the absolute values of the differences between successive elements take on all possible values 1 through $n - 1$. For instance,

1 4 2 3

is a jolly jumper, because the absolute differences are 3, 2, and 1, respectively. The definition implies that any sequence of a single integer is a jolly jumper. Write a program to determine whether each of a number of sequences is a jolly jumper.

Input

Each line of input contains an integer n (where $n < 500$) followed by n integers representing the sequence.

Output

For each line of input generate a line of output saying “Jolly” or “Not jolly”.

Sample Input

4 1 4 2 3
5 1 4 2 -1 6

Sample Output

Jolly
Not jolly

Assignment instructions:

Assignment instructions

Before starting program 3, review the top-level PSP2 process script below to ensure that you understand the “big picture” before you begin. Also, ensure that you have all of the required inputs before you begin the planning phase.

PSP2 Process Script

Purpose	To guide the development of module-level programs	
Entry Criteria	<ul style="list-style-type: none"> - Problem description - Blank PSP2 Project Plan Summary form - Blank Size Estimating worksheet and PSP Design Form - Task Planning and Schedule planning templates - Blank Time and Defect Recording logs - Defect Type and size counting standards - Stopwatch (optional) 	
Step	Activities	Description
1	Planning	- Follow the attached planning script.
2	Development	- Follow the attached development script.
3	Postmortem	- Follow the attached postmortem script.
Exit Criteria	<ul style="list-style-type: none"> - A thoroughly tested program - Completed Project Plan Summary form with estimated and actual data - Completed Size Estimating Worksheet & Design form - Completed Task planning and schedule planning templates - Completed Time and Defect Recording logs 	

Planning phase

Conduct project planning following the PSP2 planning script.

PSP2 Planning Script

Purpose	To guide the PSP planning process	
Entry Criteria	<ul style="list-style-type: none"> - Problem description - Blank PSP2 Project Plan Summary form - Blank Size Estimating worksheet - Task Planning & Schedule Planning templates - Blank Time and Defect Recording logs 	
Step	Activities	Description
1	Form Setup	<ul style="list-style-type: none"> - Complete form headers. - Enter start time for PLAN phase in Time Recording Log. - Enter your available hours on the Schedule Planning template
2	Program Requirements	<ul style="list-style-type: none"> - Produce or obtain a requirements statement for the program. - Ensure that the requirements statement is clear and unambiguous. - Resolve any questions.
3	Size Estimate	<ul style="list-style-type: none"> - Produce a program conceptual design. - Use the informal estimation procedure to estimate the size of this program. - Complete the size estimating worksheet.
4	Defect Estimate	<ul style="list-style-type: none"> - Use data from your most recent previous program to estimate the number of defects for this program. - Follow directions for Summary form and complete the estimate column for “Defects injected” and “Defects removed” part of the summary form.

5	Resource Estimate	<ul style="list-style-type: none"> - Follow the directions for completing the planning portion of the Project Summary form. - Enter the stop time for the PLAN phase in the Time Log.
---	-------------------	---

Exit Criteria	<ul style="list-style-type: none"> - Documented requirements statement. - Program conceptual design (design form). - Completed Size Estimating Worksheet - Project Summary form contains estimated program size, defect, and development time data. - Time recording log contains entry for PLAN phase. - Defect Recording log header completed. - Schedule planning template with planning columns filled.
----------------------	--

Verify that you have met all of the exit criteria for the planning phase, and then proceed to the development phase.

Development phase

Conduct development of the project following the PSP2 development script.

PSP2 Development Script

Purpose	To guide the development of small programs
Entry Criteria	<ul style="list-style-type: none"> - Same as exit criteria from Planning Script. - PSP Design Form.

Step	Activities	Description
1	Design	<ul style="list-style-type: none"> - Record start time in the Time Recording Log. - Review the requirements and produce a design to meet them. - Record any design work you do in the PSP Design form. - Record in the Defect Recording log any requirements defects found. - Record time in the Time Recording log.
2	Design Review	<ul style="list-style-type: none"> - Review your design (use the Design Review checklist provided). - Record time in the Time Recording log under Design phase.
3	Code	<ul style="list-style-type: none"> - Record start time in the Time Recording Log. - Implement the design. Write the entire source code for the solution. - Record in the Defect Recording log any requirements or design defects found. - Record time in the Time Recording log.
4	Code Review	<ul style="list-style-type: none"> - Follow the code review script provided on the next page.
5	Compile	<ul style="list-style-type: none"> - Record start time in the Time Recording Log. - Compile the program until error-free. - Fix all defects found. - Record defects in the Defect Recording log. - Record time in the Time Recording log.
6	Test	<ul style="list-style-type: none"> - Record start time in the Time Recording Log. - Test until all tests run without error. - Fix all defects found. - Record defects in the Defect Recording log. - Record time in the Time Recording log. - Based on the recorded times, compute earned values and record them in actual columns of task and schedule planning templates.

Exit Criteria	<ul style="list-style-type: none"> - A thoroughly tested program. - Completed PSP Design Form. - Time Log entries for Plan through Test Phases. - Completed Defect Recording log. - Task and Schedule Planning templates
----------------------	---

Verify that you have met all of the exit criteria for the development phase, and then proceed to the postmortem phase.

Continued on next page

PSP2 Code Review Script

Purpose	To guide you in reviewing programs	
Entry Criteria	<ul style="list-style-type: none"> - A completed and reviewed program design - Source program listing - Code Review checklist - Defect Type standard - Time and Defect Recording logs 	
Step	Activities	Description
1	Review	<ul style="list-style-type: none"> - Record start time in Time Recording Log - Follow the Code Review checklist. - Review the entire program for each checklist category; do not try to review for more than one category at a time! - Check off each item as it is completed. - For multiple modules or programs, complete a separate checklist for each.
2	Correct	<ul style="list-style-type: none"> - Correct all defects. - If the correction cannot be completed, abort the review and return to the prior process phase. - To facilitate defect analysis, record all of the data specified in the Defect Tally instructions for every defect.
3	Check	<ul style="list-style-type: none"> - Check each defect fix for correctness. - Re-review all design changes. - Record any fixed defects as new defects. - Record stop time in Time Recording Log.
Exit Criteria	<ul style="list-style-type: none"> - A fully reviewed source program - One or more Code Review checklists for every program reviewed - All identified defects fixed - Completed Time and Defect Recording log 	

Verify that you have met all of the exit criteria for the code review phase, and then continue the remaining steps in the development phase.

Postmortem phase

Conduct the postmortem following the PSP2 postmortem script.

PSP2 Postmortem Script

Purpose	To guide the PSP postmortem process	
Entry Criteria	- Same as exit criteria from Development Script.	
Step	Activities	Description
1	Defect Recording	<ul style="list-style-type: none"> - Review the Project Plan Summary to verify that all of the defects found in each phase were recorded. - Using your best recollection, record any omitted defects.
2	Defect Data Consistency	<ul style="list-style-type: none"> - Record start time for Postmortem in the Time Recording Log. - Check that the data on every defect in the Defect Recording log are accurate and complete. - Verify that the numbers of defects injected and removed per phase are

		reasonable and correct. - Using your best recollection, correct any missing or incorrect defect data.
3	Defect Summarizing	- Summarize defect tally data on the Project Summary Form.
4	Size	- Count the size of the completed program (can use LOC counter programs available online). Don't count comments. - Enter this data in the Project Summary form.
5	Time	- Review the completed Time Recording log for errors or omissions. - Using your best recollection, correct any missing or incomplete time data. - Compute the delta time for all completed log entries. - Guess how long it will take to complete the Project Summary calculations and enter you guessed stop time in the Time Log. Compute the delta time. - Summarize time data on Project Summary form. - Finish remaining calculations on Project summary form.
Exit Criteria		- A thoroughly tested program - Completed Project Plan Summary form - Completed Time and Defect Recording logs

Verify that you have met all of the exit criteria for the PSP2 postmortem phase, and then submit your assignment.

Submitting your assignment

When you've completed the postmortem phase, submit your assignment via Blackboard or hard-copy to your instructor. The submission package should have the following:

- PSP2 Project Plan Summary form
- PSP Estimation worksheet
- PSP Design form **with reviews/comments from 1 of your classmates**
- Time Recording log
- Defect Recording log
- Design Review and Code Review checklists (marked)
- Task Planning and Schedule Planning templates (with earned value columns completed)
- Source program listing
- **Test results for 4 test cases**

PSP2 Project Summary

Programmer _____ Program ID _____
 Program _____ Date _____

Time in Phase (min.)	Est.	Actual	To Date	To Date %
Planning	_____	_____	_____	_____
Design	_____	_____	_____	_____
Code	_____	_____	_____	_____
Code Review	_____	_____	_____	_____
Compile	_____	_____	_____	_____
Test	_____	_____	_____	_____
Postmortem	_____	_____	_____	_____
Total	_____	_____	_____	_____

Defects Injected	Est.	Actual	To Date	To Date %
Planning	_____	_____	_____	_____
Design	_____	_____	_____	_____
Code	_____	_____	_____	_____
Code Review	_____	_____	_____	_____
Compile	_____	_____	_____	_____
Test	_____	_____	_____	_____
Total Development	_____	_____	_____	_____
After Development	_____	_____	_____	_____

Defects Removed	Est.	Actual	To Date	To Date %
Design	_____	_____	_____	_____
Code	_____	_____	_____	_____
Code Review	_____	_____	_____	_____
Compile	_____	_____	_____	_____
Test	_____	_____	_____	_____
Total Development	_____	_____	_____	_____
After Development	_____	_____	_____	_____

Summary	Est.	Actual	To Date
Program Size (LOC)	_____	_____	_____
LOC/Hour	_____	_____	_____
Defects/KLOC	_____	_____	_____
Yield	_____	_____	_____
A/F ratio	_____	_____	_____

PSP Estimating Worksheet

1. Conceptual Design (Sketch your design here)

2. Module Estimates

Module Description	Estimated Size

TOTAL ESTIMATED SIZE: _____

PSP Design Form

Programmer _____

Program ID _____

Use this form to record whatever you do during the design phase of development. Include notes, diagrams, formal design notation, or anything else you consider to be part of designing a solution that happens BEFORE you write program source code. Attach additional pages if necessary.

PSP2 Design Review Checklist

Student _____	Date _____
Program _____	Program # _____
Instructor _____	Language _____

Purpose	To guide you in conducting an effective design review
General	<ul style="list-style-type: none"> - Review the entire design for each checklist category; do not attempt to review for more than one category at a time! - As you complete each review step, check off that item in the box at the right. - Complete the checklist for one program or program unit before reviewing the next.

Complete	Verify that the design covers all of the applicable requirements. <ul style="list-style-type: none"> - All specified outputs are produced. - All needed inputs are furnished. - All required includes are stated. 				
External Limits	Where the design assumes or relies upon external limits, determine if behavior is correct at nominal values, at limits, and beyond limits.				
Logic	<ul style="list-style-type: none"> - Verify that program sequencing is proper. <ul style="list-style-type: none"> Stacks, lists, and so on are in the proper order. Recursion unwinds properly. - Verify that all loops are properly initiated, incremented, and terminated. - Examine each conditional statement and verify all cases. 				
Internal Limits	Where the design assumes or relies upon internal limits, determine if behavior is correct at nominal values, at limits, and beyond limits.				
Special Cases	<ul style="list-style-type: none"> - Check all special cases. - Ensure proper operation with empty, full, minimum, maximum, negative, and error values for all variables. - Protect against out-of-limits, overflow, and underflow conditions. - Ensure "impossible" conditions are absolutely impossible. - Handle all possible incorrect or error conditions. 				
Functional Use	<ul style="list-style-type: none"> - Verify that all functions, procedures, or methods are fully understood and properly used. - Verify that all externally referenced abstractions are precisely defined. 				
System Considerations	<ul style="list-style-type: none"> - Verify that the program does not cause system limits to be exceeded. - Verify that all security-sensitive data are from trusted sources. - Verify that all safety conditions conform to the safety specifications. 				
Names	Verify that <ul style="list-style-type: none"> - all special names are clear, defined, and authenticated - the scopes of all variables and parameters are self-evident or defined - all named items are used within their declared scopes 				
Standards	Ensure that the design conforms to all applicable design standards.				

PSP Defect Recording Log

Defect Types	
10 Documentation	60 Checking
20 Syntax	70 Data
30 Build, Package	80 Function
40 Assignment	90 System
50 Interface	100 Environment

Student _____	Date _____
Program _____	Program # _____
Instructor _____	Language _____

Project	Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.
Description: _____							

Project	Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.
Description: _____							

Project	Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.
Description: _____							

Project	Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.
Description: _____							

Project	Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.
Description: _____							

Project	Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.
Description: _____							

Project	Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.
Description: _____							

Project	Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.
Description: _____							

PSP Defect Recording Log Instructions

Purpose	<ul style="list-style-type: none"> - Use this form to hold data on the defects that you find and correct. - These data are used to complete the Project Plan Summary form.
General	<ul style="list-style-type: none"> - Record each defect separately and completely. - If you need additional space, use another copy of the form.
Header	<ul style="list-style-type: none"> - Enter your name and the date. - Enter the program name and number. - Enter the instructor's name and the programming language you are using.
Project	<ul style="list-style-type: none"> - Give each program a different name or number. - For example, record test program defects against the test program.
Date	Enter the date on which you found the defect.
Number	<ul style="list-style-type: none"> - Enter the defect number. - For each program or module, use a sequential number starting with 1 (or 001, etc.).
Type	<ul style="list-style-type: none"> - Enter the defect type from the defect type list summarized in the top left corner of the form. - Use your best judgment in selecting which type applies.
Inject	<ul style="list-style-type: none"> - Enter the phase when this defect was injected. - Use your best judgment.
Remove	Enter the phase during which you fixed the defect. (This will generally be the phase when you found the defect.)
Fix Time	<ul style="list-style-type: none"> - Enter the time that you took to find and fix the defect. - This time can be determined by stopwatch or by judgment.
Fix Ref.	<ul style="list-style-type: none"> - If you or someone else injected this defect while fixing another defect, record the number of the improperly fixed defect. - If you cannot identify the defect number, enter an X.
Description	Write a succinct description of the defect that is clear enough to later remind you about the error and help you to remember why you made it.

PSP Defect Type Standard

Type Number	Type Name	Description
10	Documentation	Comments, messages
20	Syntax	Spelling, punctuation, typos, instruction formats
30	Build, Package	Change management, library, version control
40	Assignment	Declaration, duplicate names, scope, limits
50	Interface	Procedure calls and references, I/O, user formats
60	Checking	Error messages, inadequate checks
70	Data	Structure, content
80	Function	Logic, pointers, loops, recursion, computation, function defects
90	System	Configuration, timing, memory
100	Environment	Design, compile, test, or other support system problems

Task Planning Template Instructions

Purpose	<ul style="list-style-type: none"> • To estimate the development time for each project task • To compute the planned value for each project task • To estimate the planned completion date for each task • To provide a basis for tracking schedule progress even when the tasks are not completed in the planned order
General	<ul style="list-style-type: none"> • Expand this template or use multiple pages as needed. • Include every significant task. • Use task names and numbers that support the activity and are consistent with the project work breakdown structure.
Header	<p>Enter the following:</p> <ul style="list-style-type: none"> • Your name • Today's date • The project number • The instructor's name
Task	<ul style="list-style-type: none"> • Enter a task number and name. List the tasks in the order in which you expect to complete them. • Select tasks that have explicit completion criteria, for example, planning completed, program compiled and all defects corrected, testing completed and all defects corrected, and so on.
Plan – Hours	Enter the planned hours for each task.
Plan – Planned Value	<ul style="list-style-type: none"> • Total the planned hours for all the tasks. • For each task, calculate the percent its planned hours are of total hours. • Enter this percentage as the planned value for that task. • The total planned value should equal 100.
Plan – Cumulative Hours	Enter the cumulative sum of the plan hours down through each task.
Plan – Cumulative Value	<ul style="list-style-type: none"> • Sum the planned values down through each task. • Before proceeding, complete the Schedule Planning Template down through Plan–Cumulative Hours. • Then complete the Schedule Planning and Task Planning templates together.
Plan Date – Monday	<ul style="list-style-type: none"> • For each cumulative hours entry, find the plan cumulative hours entry on the Schedule Planning Template that equals or just exceeds it. • Enter the date from that row (of the Schedule Planning Template) as the plan date on the Task Planning Template. • If several weeks on the Schedule Planning Template have the same cumulative value, enter the earliest date. • Unless you made daily plans, pick the plan date as the Monday of the week during which completion for that task is planned.
Actual Date	As each task is completed, enter the completion date.
Earned Value	For each completed task, enter the planned value.
Cumulative Earned Value	As each task is completed, total all the earned value entries and enter that total beside the latest task that was completed.

Schedule Planning Template Instructions

Purpose	<ul style="list-style-type: none"> • To record the estimated and actual hours expended by calendar period • To relate the task planned value to the calendar schedule • To calculate adjusted planned and earned values when tasks change
General	<ul style="list-style-type: none"> • Expand this template or use multiple pages as needed. • Complete it in conjunction with the Task Planning Template.
Header	<p>Enter the following:</p> <ul style="list-style-type: none"> • Your name • Today's date • The project number • The instructor's name
Week Number	<ul style="list-style-type: none"> • From the project start, enter a week number, typically starting with 1. • For very small projects, it may be more convenient to use days instead of weeks.
Date (Monday)	<ul style="list-style-type: none"> • Enter the calendar date for each week. • Pick a standard day in the week, for example, Monday.
Plan-Direct Hours	<ul style="list-style-type: none"> • Enter the planned number of direct project hours you expect to spend each week. • Consider non-work time such as vacations, holidays, and so on. • Consider other committed activities such as classes, meetings, and other projects.
Plan-Cumulative Hours	Enter the cumulative planned hours through each week.
Plan-Cumulative Planned Value	<p>For each week, do the following:</p> <ul style="list-style-type: none"> • Take the plan cumulative hours from the Schedule Planning Template. • On the Task Planning Template, find the task with nearest equal or lower plan cumulative hours and note its plan cumulative value. • Enter this cumulative value in the Schedule Planning Template for that week. • If the cumulative value for the prior week still applies, enter it again.
Actual	<ul style="list-style-type: none"> • During development, enter the actual direct hours, cumulative hours, and cumulative earned value for each week. • Determine status against plan by comparing the cumulative planned value and the actual cumulative earned value.
Adjusted Earned Value	Proportionately adjust the earned value up or down as tasks are added or deleted. The adjusted earned value compensates for these changes without requiring a complete new plan.

Contents

Part 1 Introduction to Software Engineering

Chapter 1: Introduction

- 1. Professional software development
- 1.2 Software engineering ethics
- 1.3 Case studies

Chapter 2: Software processes

- 2.1 Software process models
- 2.2 Process activities
- 2.3 Coping with change
- 2.4 The Rational Unified Process

Chapter 3: Agile software development

- 3.1 Agile methods
- 3.2 Plan-driven and agile development
- 3.3 Extreme programming
- 3.4 Agile project management
- 3.5 Scaling agile methods

Chapter 4: Requirements engineering

- 4.1 Functional and non-functional requirements
- 4.2 The software requirements document
- 4.3 Requirements specification
- 4.4 Requirements engineering processes
- 4.5 Requirements elicitation and analysis
- 4.6 Requirements validation
- 4.7 Requirements management

Chapter 5: System modeling

- 5.1 Context models
- 5.2 Interaction models
- 5.3 Structural models
- 5.4 Behavioral models
- 5.5 Model-driven engineering

Chapter 6: Architectural design

- 6.1 Architectural design decisions
- 6.2 Architectural views

- 6.3 Architectural patterns
- 6.4 Application architectures

Chapter 7: Design and Implementation

- 7.1 Object-oriented design using the UML
- 7.2 Design patterns
- 7.3 Implementation issues
- 7.4 Open source development

Chapter 8: Software testing

- 8.1 Development testing
- 8.2 Test-driven development
- 8.3 Release testing
- 8.4 User testing

Chapter 9: Software Evolution

- 9.1 Evolution processes
- 9.2 Program evolution dynamics
- 9.3 Software maintenance
- 9.4 Legacy system management

Part 2 Dependability and Security

Chapter 10: Socio-technical Systems

- 10.1 Complex systems
- 10.2 Systems engineering
- 10.3 System procurement
- 10.4 System development
- 10.5 System operation

Chapter 11: Dependability and Security

- 11.1 Dependability properties
- 11.2 Availability and reliability
- 11.3 Safety
- 11.4 Security

Chapter 12: Dependability and Security Specification

- 12.1 Risk-driven requirements specification
- 12.2 Safety specification
- 12.3 Reliability specification
- 12.4 Security specification

12.5 Formal specification

Chapter 13: Dependability Engineering

13.1 Redundancy and diversity
13.2 Dependable processes
13.3 Dependable systems architectures
13.4 Dependable programming

Chapter 14: Security Engineering

14.1 Security risk management
14.2 Design for security
14.3 System survivability

Chapter 15: Dependability and Security Assurance

15.1 Static analysis
15.2 Reliability testing
15.3 Security testing
15.4 Process assurance
15.5 Safety and dependability cases

Part 3 Advanced Software Engineering

Chapter 16: Software Reuse

16.1 The reuse landscape
16.2 Application frameworks
16.3 Software product lines
16.4 COTS product reuse

Chapter 17: Component-based Software Engineering

17.1 Components and component models
17.2 CBSE processes
17.3 Component composition

Chapter 18: Distributed Software Engineering

18.1 Distributed systems issues
18.2 Client-server computing
18.3 Architectural patterns for distributed systems
18.4 Software as a service

Chapter 19: Service-oriented Architecture

- 19.1 Services as reusable components
- 19.2 Service engineering
- 19.3 Software development with services

Chapter 20: Embedded Systems

- 20.1 Embedded systems design
- 20.2 Architectural patterns
- 20.3 Timing analysis
- 20.4 Real-time operating systems

Chapter 21: Aspect-oriented software engineering

- 21.1 The separation of concerns
- 21.2 Aspects, join points and pointcuts
- 21.3 Software engineering with aspects

Part 4 Software management

Chapter 22: Project management

- 22.1 Risk management
- 22.2 Managing people
- 22.3 Teamwork

Chapter 23: Project planning

- 23.1 Software pricing
- 23.2 Plan-driven development
- 23.3 Project scheduling
- 23.4 Agile planning
- 23.5 Estimation techniques

Chapter 24: Quality management

- 24.1 Software quality
- 24.2 Software standards
- 24.3 Reviews and inspections
- 24.4 Software measurement and metrics

Chapter 25: Configuration management

- 25.1 Change management
- 25.2 Version management
- 25.3 System building
- 25.4 Release management

Chapter 26: Process improvement

- 26.1 The process improvement process
- 26.2 Process measurement
- 26.3 Process analysis
- 26.4 Process change
- 26.5 The CMMI process improvement framework

Glossary