



Project & Tools Handbook



Citi - the leading global bank

Citi, the leading global bank, has approximately 200 million customer accounts and does business in more than 160 countries and jurisdictions. Citi provides consumers, corporations, governments and institutions with a broad range of financial products and services, including consumer banking and credit, investment banking, securities brokerage, transaction services and wealth management.

Formed in 1812, the company has been helping other organisations conduct their business, and individuals manage their finances for over 200 years. We are focused on providing simple, creative, and responsible financial solutions; and using fresh-thinking and the latest technologies to become established as the leading digital bank for the future.

TECHNOLOGY brings our bank genuine competitive advantage

How would you like to help shape the future of international banking?

Whether it's in the complex, international multi-million dollar trades we conduct every second, or the intricate trading strategies of the individual investor portfolios we support across the world, you can assist in advancing our technological capabilities day after day.

We aim to not only to deliver seamless services to the bank, but work towards building the flexible innovative systems that will define the digital bank of tomorrow.

Join us to find out how you can play your part.

Belfast Citi Technology Academy

The Citi Technology Academy provides an unrivalled opportunity to build a career in programming, application support or quality assurance. The Academy participants will benefit from a bespoke programme of "best in class" technical and business training which will provide a platform of knowledge and experience upon which they will build their career with Citi.

You'll benefit from 12 weeks' training in:

- Capital markets and investment banking training
- Software development – Java, .NET, SQL Server, UNIX, design patterns, testing & automation, Agile methodologies
- Application support – database technology, industry standards, incident/change management, monitoring & tools.

Visit www.uncampus.citi.com for more information.

Handbook Overview

Citi have created this handbook with the purpose of providing additional information which is designed to assist you, the student. The handbook is designed in such a way as to give you guidance not only as you work through your project but for any future IT projects you may work on.

In the following chapters you will find information on project management, communication and effective teamwork which are 3 key skills required for a successful project. Within these sections you will find tools designed to help you as well as do's and don'ts in these areas.

Insight will also be provided into a number of tools with a view to developing key IT skills in code management, code quality, testing and continuous integration, all of which are essential in developing any IT solution.

An overview of quality assurance will show where it fits in to the software development cycle and the types of testing that can be performed.

All of the subject matter covered has relevance to the project that you are about to embark on, the tools introduced will assist you in overcoming any issues along the way and provides information and guidance in areas key to the world of IT employment.

Contents

Contents.....	III
1 Project Management	1
1.1 Introduction to Project Management.....	1
a What.....	1
b Four Phases of a project.....	1
1. Initiation phase	1
2. Planning phase.....	1
3. Execution phase	2
4. Closure phase	2
c Two Project Methodologies	3
Waterfall methodology – four phases follow each other in turn.....	3
AGILE methodology – multiple Planning and Execution phases	3
d Breaking down Requirements	4
e Project Plans and Status Reports	4
Project Plans.....	4
Status Reports	5
f Common Project Management Mistakes.....	5
1.2 Communication and Teamwork	7
a Introduction to Communication	7
Definition.....	7
The Process	7
Types of Communication	7
Communication Barriers	9
Potential Communication Challenges and solutions	9
b The 7 Cs of effective communication.....	9
c Meetings.....	10
Checklist for a successful meeting.....	10
What not to Do in meeting	10
Conference Call Etiquette	10
d Teamwork.....	11
Effective Teamwork	11

Conflict resolution	11
Escalation	11
2 Coding Best Practice	13
a What.....	13
Code Review.....	13
B Example Coding Best Practice	14
C Code Review BEst Practice	14
D Code Review Tools – Code collaborator	16
Creating a Review	16
Action Items	16
Review Editor	17
E Further information on coding standards.....	18
3 Unit Testing.....	19
A What is Unit Testing.....	19
B Why Write Unit Tests.....	19
C When to Write Unit Tests.....	19
D How to Write Unit Tests.....	20
E Unit Test Frameworks.....	20
E Example: Java Unit Test Using JUnit	21
4 Tools.....	22
4.1 Eclipse	22
a What.....	22
b Key Concepts	22
c Key Perspectives	22
d Key Activities	22
Change Perspective	22
Creating A Java Project	22
Creating A Package	22
Creating A Class	23
Renaming A Project/Package/Class/File	23
Configuring the Project Classpath	23
Compiling Code.....	23
Running A “Main” Class	23
Setup A “Run Configuration”	24
Setting A Breakpoint	24

4.2	Subclipse.....	25
a	What.....	25
b	Key Concepts	25
c	General Principles.....	25
d	Key Activities	25
	Setup Connection to the repository	25
	Sharing A Project (Upload To The Repository)	26
	Checking Out A Project	26
	Synchronizing (Getting/Committing Changes & Resolving Conflicts).....	26
4.3	Ant.....	27
a	What.....	27
b	Key Concepts	27
c	Key Activities	27
	Creating a Build Script Manually	27
	Creating a Build Script Automatically	28
	Running a Build Script.....	28
4.4	Jenkins.....	29
a	What.....	29
b	Key Concepts	29
c	Key Activities	30
	Creating a Job.....	30
	Manually Running a Job.....	30
	Managing Jenkins	31
4.5	EclEmma – Eclipse Plugin	32
a	What.....	32
b	Key Concepts	32
c	General Principles.....	32
d	Key Activities	32
	Running ECL Emma	32
4.6	FindBugs – Eclipse Plugin	33
a	What.....	33
b	Key Concepts	33
c	General Principles.....	33
d	Key Activities	33
	Running FindBugs	33

Disabling A Rule	33
5 Quality Assurance	34
5.1 Introduction to Quality Assurance.....	34
a What.....	34
b Where QA fits in to the software development lifecycle.....	34
V-Model – Early test design	34
c Types of Testing	35
Overview of testing types	35
d Typical tester tasks.....	38
Key documents	38
Test management.....	39
e QA Career Avenues.....	39
Why testing?	39
Psychology of testing	40
Careers options.....	41
Notes	42

1 Project Management

1.1 Introduction to Project Management

A WHAT

Project Management is about creating an environment and conditions in which a defined goal or objective can be achieved in a controlled manner by a team of people. (www.projectsmart.co.uk)

A **Project** is a temporary activity with a defined beginning and end undertaken to meet unique goals and objectives, typically to bring about beneficial change. (Wikipedia)

A good **Project Manager** is one of the keys to a successful project. He or she carries the main responsibility for making sure the project delivers what is needed, on time and to budget.

They need to make sure the standards are being followed, that everyone is being kept up to date with the relevant information and that the project team is working well together.

B FOUR PHASES OF A PROJECT

1. INITIATION PHASE

Defines the overall objectives for the project and sets up the management structure.

Outputs

- Business case (*what* the project will do and *why*)
- Initial estimate of work involved i.e. cost and likely end date (*when*)
- Definition of the key stakeholders - the senior decision makers (*who*)

Tip: The work or spend involved is estimated at a high-level at this stage but it is important. If the cost of the project is going to be more than the resulting additional business revenue then it probably isn't worth doing!

2. PLANNING PHASE

Defines the detailed scope and requirements, the resources and any dependencies

Outputs

- Business requirements (e.g. GUI functionality, reports, projected volumes, performance)
- IT requirements (e.g. security needs, hardware, software/tools, testing environments)
- Risk and issue log
- Agree project communication plan
- Agree project team makeup
- Initial project plan (critical path, planned end date)

A *Risk* is a potential event or situation that, if it happened, would hinder your project e.g. if a key person was re-assigned to another project.

An *Issue* is a situation that is hindering your project or a question that has arisen (often relating to incomplete requirements).

The *Project Plan* pulls together the requirements, the estimates for the work and the people available and results in a proposed delivery date

Tip: Try to capture everything that could be impacted by your project e.g. other systems, screens, security requirements, failover¹ in case a server goes down, or user manuals.

3. EXECUTION PHASE

Implements the tested software along with any required documentation

Outputs

- Detailed technical design
- Review risks and issues
- Coding and unit testing
- Track progress against project plan, produce status reports
- Test plan and testing (integration, stress, user, automated)
- Implementation plan
- Training manuals and system documentation
- “Go live”

Unit testing is the testing of individual methods (or small pieces of code) in isolation.

Integration testing is when everyone’s code is built together and tested to ensure the end-to-end processing works.

User testing (or User Acceptance Testing) simulates “real-life” usage and checks the project meets the business requirements.

Stress testing may be carried out for high-volume or high-performing systems.

Tip: Plan out your project implementation so that the steps happen in the correct order. Also think about a rollback plan in case the deployment goes wrong!

4. CLOSURE PHASE

Ensures an orderly end to the project

Outputs

- Identify any outstanding tasks or actions

¹ Wikipedia: “In computing, **failover** is automatic switching to a redundant or standby computer server, system, hardware component or network upon the failure or abnormal termination of the previously active application, server, system, hardware component, or network.”

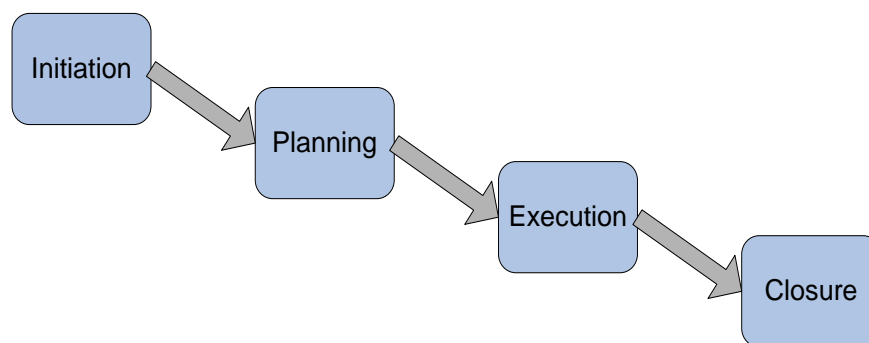
- Evaluate the project success e.g. compares the cost/time/deliverables against the business case
- Document any lessons learned
- Handover to Support team

If any requirements have not been delivered – e.g. an enhancement to a screen that was too complex to be fitted into the dates – then these should be recorded somewhere to be picked up by a subsequent project.

Tip: The Lessons Learned document is often overlooked but it is a very useful way of capturing what went well and what could be done better in the future.

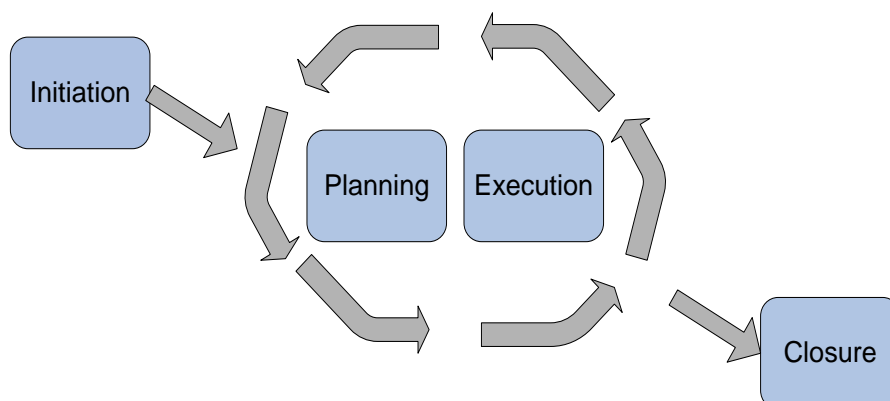
C TWO PROJECT METHODOLOGIES

WATERFALL METHODOLOGY – FOUR PHASES FOLLOW EACH OTHER IN TURN.



The traditional methodology works well when the requirements are clearly defined up front but any late or changing requirements can result in unplanned re-work – thus putting the agreed delivery date at risk. There are variations upon the model shown.

AGILE METHODOLOGY – MULTIPLE PLANNING AND EXECUTION PHASES



Agile uses a series of planning and execution phases called “sprints” each one resulting in the implementation of new or changed functionality.

In this way, the user will receive some new or amended product or service early and can come back with changes which will feed into later sprints. Also each sprint process can be refined as lessons are learnt along the way.

Agile methodology includes a range of concepts such as daily “scrum” meetings, “user stories” to record business requirements, “test-driven development” and “pairs programming.” You can pick and choose the different parts of Agile which seem the most beneficial to your project and which meet any standards already in place.

D BREAKING DOWN REQUIREMENTS

A key skill is being able to break down the project requirements into smaller pieces of work. This will help you estimate and share out the work.

For example: “Ensure only registered users can access the system via the GUI” could break down into:

- New login screen
- Process to check username and password against the database and return “success” or “failure”
- New or amended database tables to hold the user details
- New screen for users to change their password (if needed)
- Process to force the users to change their password (if needed)
- Process to update the passwords in the database (is encryption needed?)
- Audit trail of successful and failed logins

Tip: It can helpful to think through the processing from start to finish, step by step.

E PROJECT PLANS AND STATUS REPORTS

PROJECT PLANS

There are various tools which can be used to create a project plan e.g. Microsoft Project but a simple spreadsheet will suffice.

[Project Name] Workplan/Schedule								
% Complete	Task Name	Duration	Planned Start	Actual Start	Planned Finish	Actual Finish	Resource Name	De
II. Planning Phase								
100%	Planning workshop with team	1d	07/09/2012	07/09/2012	08/09/2012	08/09/2012	Sarah and Team	
100%	Agree project communication guidelines and responsibilities	1d	08/09/2012	08/09/2012	09/09/2012	09/09/2012	Sarah and Team	
100%	Define scope and requirements	3d	09/09/2012	09/09/2012	12/09/2012	12/09/2012	Sarah Lucas	
100%	Identify deliverables	2d	10/09/2012	10/09/2012	12/09/2012	12/09/2012	Simon Jackson	
100%	Break deliverables into tasks	2d	12/09/2012	13/09/2012	14/09/2012	15/09/2012	Simon Jackson	
100%	Estimate tasks	1d	13/09/2012	13/09/2012	14/09/2012	14/09/2012	Sarah Lucas	
100%	Assign tasks to team members	1d	14/09/2012	14/09/2012	15/09/2012	15/09/2012	Simon Jackson	
100%	Review plans and documents with team	0.5d	20/09/2012	21/09/2012	20/09/2012	21/09/2012	Sarah and Team	
100%	Circulate project documentation	0.5d	22/09/2012	22/09/2012	22/09/2012	23/09/2012	Sarah Lucas	
	Planning Phase Complete				22/09/2012	23/09/2012		
III. Execution Phase								
100%	Setup source control	3d	23/09/2012	23/09/2012	26/09/2012	26/09/2012	Simon Jackson	
50%	Coding and Unit testing							
	New login screen	3d	27/09/2012		30/09/2012		Simon Jackson	
	password verification	4d	24/09/2012		28/09/2012		Anna Knight	
	database changes	4d	24/09/2012		28/09/2012		John Black	
	logging	2d	24/09/2012		26/09/2012		Emily James	
	Review test cases	2d	26/09/2012		28/09/2012		Sarah Lucas	
	Integration testing	5d	01/10/2012		06/10/2012		Team	
	Prepare implementation plan	2d	06/10/2012		08/10/2012		Simon Jackson	
	Implementation date	1d	15/10/2012		15/10/2012			
	Execution Phase Complete				15/10/2012			
IV. Closure Phase								
	Handover session for Support	2d	16/10/2012		18/10/2012		Sarah and Team	

Tip: Remember to think about what tasks need to be completed before other tasks can start.

STATUS REPORTS

These are used to summarise progress, risks and issues for senior management and usually include a RAG status (compared against the Project Plan).

- **RED** – Project is running “significantly” late and the end date is at risk
- **AMBER** – Project has slipped a little and could be late if action is not taken
- **GREEN** – Project is progressing as per the Plan and is on target

F COMMON PROJECT MANAGEMENT MISTAKES

1. Taking shortcuts in the Planning Phase.

This is very tempting but often results in inaccurate delivery dates and/or in time being lost owing to re-work.

2. Lack of process in place to handle changing requirements or “scope creep”.

This often happens and the temptation will be to try to accommodate all the changes and yet still meet the same planned date.

The correct process is to estimate the extra work and then agree on one of these options:

- deliver the extra work but adjust the delivery date
- expand the project team (additional cost) to deliver the extra work and meet the planned date
- postpone the extra work to a later project

3. Under-estimating testing.

The more components and interfaces your project has, the longer it will take to test.
Realistic end-to-end testing is vital.

4. An incomplete Project Plan.

Everything needs to be captured in the Project Plan otherwise your end date will be unrealistic and your project will end up late.

“Scope creep” is when someone requests changes to a new area, system or component that wasn’t originally considered part of the project.

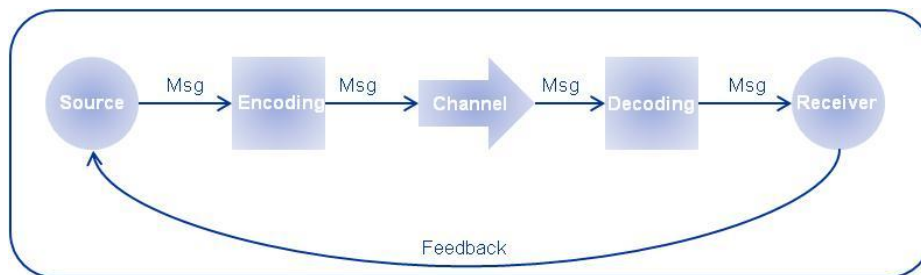
1.2 Communication and Teamwork

A INTRODUCTION TO COMMUNICATION

DEFINITION

Communication is the exchange of thoughts, messages, or information, as by speech, visuals, signals, writing, or behavior.

THE PROCESS



Source	• person relaying the message
Message	• information you want to communicate
Encoding	• process of transferring the information
Channel	• how the message is conveyed, verbal/non verbal
Decoding	• reading/listening carefully
Receiver	• your audience
Feedback	• reaction to the message

TYPES OF COMMUNICATION

There are two types of communication:

- **Verbal**
 - Oral
Message is transmitted verbally, by word of mouth
 - Written
Message is influenced by the vocabulary and grammar used, writing style, precision and the clarity of the language used
- **Non-verbal**
 - Appearance
Person - clothing, hairstyle or neatness
Surroundings - room size, decorations and furnishings.
 - Body Language
Facial expressions, gestures and postures
 - Sounds
Voice tone, volume and rate of speech

Communication Style	Verbal		Non Verbal		
	Oral	Written	Appearance	Body Language	Sounds
Face to Face	✓	✓	✓	✓	✓
Telephone	✓				✓
Video Conference	✓		✓	✓	✓
Email		✓			
Reports		✓			
Presentations	✓	✓	✓	✓	✓
Instant Message		✓			
Live Meeting/Web Ex	✓	✓			
Forum Boards		✓			

COMMUNICATION BARRIERS

- Overly Complex Messages
- Withholding Information
- Incorrect information
- Lack of Trust
- Surroundings
- Personality Types
- Body language & Rapport

POTENTIAL COMMUNICATION CHALLENGES AND SOLUTIONS

Challenges

- Time zones
- Language Barriers
- Cultural Differences
- Misinterpretation
- Isolation

Solutions

- Establish a routine
- Allow for Personal preferences (timings)
- Encourage the team to communicate in the most effective and comfortable way for them
- Encourage openness, build rapport
- Be aware of the language used and its possible interpretations

B THE 7 Cs OF EFFECTIVE COMMUNICATION

Completeness	<ul style="list-style-type: none">• Convey all the facts required by the audience
Conciseness	<ul style="list-style-type: none">• Underline and highlight the main message while avoiding jargon
Consideration	<ul style="list-style-type: none">• Take the audience into consideration, background, level, requirements
Clarity	<ul style="list-style-type: none">• Clear message using exact, appropriate and concrete words
Concreteness	<ul style="list-style-type: none">• Be precise, clear, support with facts & figures
Courtesy	<ul style="list-style-type: none">• Consider both viewpoints, show respect, not biased
Correctness	<ul style="list-style-type: none">• Exact, correct, well-timed, use the correct language

C MEETINGS

CHECKLIST FOR A SUCCESSFUL MEETING

In order to have a successful meeting you should:

- Check all attendees' availability (Be aware of time zones)
- Book room or conference facilities
- Clarify the objectives of the meeting
- Send out an agenda with the meeting invite
- Open the call early
- Ensure you get your points across clearly
- Keep on track and be aware of time constraints
- Ask questions
- Summarise any actions required
- Determine if a follow up meeting is required
- Close call
- Send out email containing meeting minutes and summary of any actions

WHAT NOT TO DO IN MEETING

- Arrive late
- Use inappropriate body language – slouching, rolling eyes, avoiding eye contact
- Leave abruptly without prior warning
- Talk over the top of people
- Be overly aggressive – interrogating, criticising, blaming, shaming
- Use excessive jargon
- Ignore poor personal care
- Sit on the sidelines, be reluctant to engage
- Ignore time zones, cultural differences and consideration of others

CONFERENCE CALL ETIQUETTE

- Be on time
- Introduce yourself
- Advise if you need to drop off the call early
- Mute your phone, but not everyone's
- Pay attention
- Don't eat and drink
- Don't talk over people
- Allow for breaks on long calls
- Be aware of time-zones

D TEAMWORK

EFFECTIVE TEAMWORK

A team's journey starts from being a group of strangers to becoming a united team with a **common goal**.

Effective teamwork arises from:

- A team whose size and resources match the task
- Good leadership and attention to team-building
- Commitment by the team to understand and identify goals
- Working together as a team to achieve those goals
- A shared sense of ownership and responsibility
- Co-ordinated effort and planned sharing of tasks evenly

Communication is key to a team's success

The purpose of communication in a team:

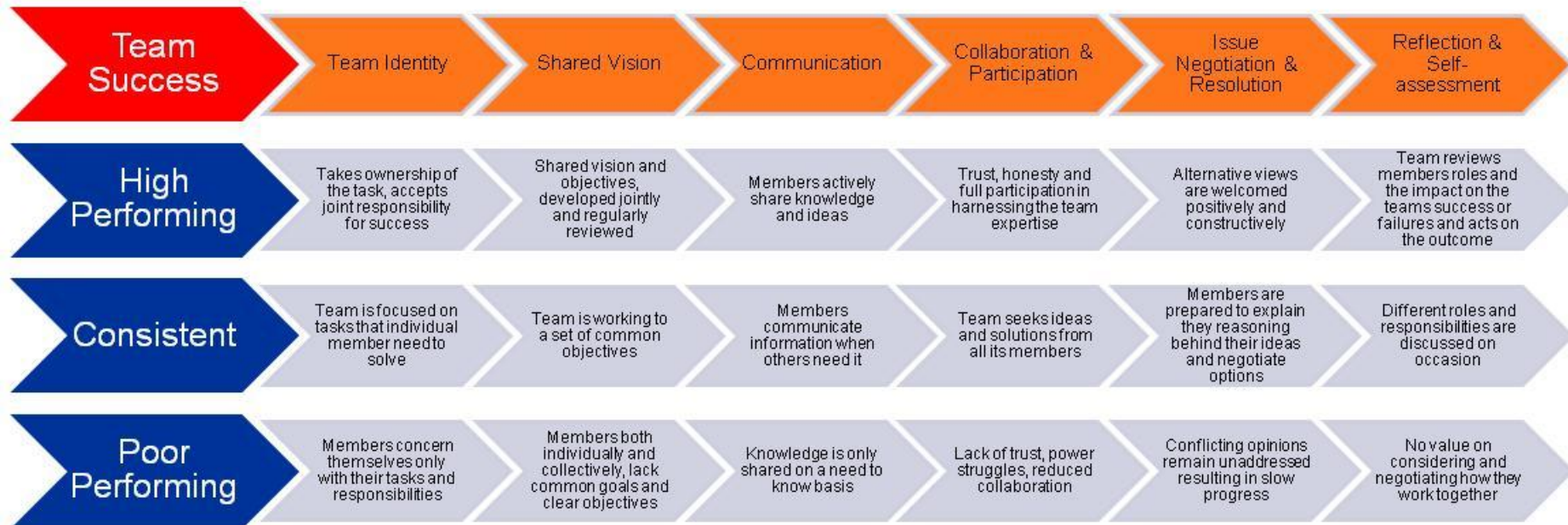
- Deliver a message
- Keep people informed
- Receive feedback
- Promote team unity
- Motivate
- Escalation
- Deal with issues

CONFLICT RESOLUTION

- Acknowledge the conflict
- Constructive questioning to identify the cause of the conflict
- Focus on joint team solutions
- Allow all team members to contribute
- Allow frustration to be heard
- Don't approach a conflict with a solution already in mind
- Celebrate the success of the resolution

ESCALATION

- What is it?
 - To raise project issues to a higher authority for awareness and resolution
- When to escalate
 - Timelines are at risk
 - Conflict cannot be resolved
 - Conflicting resource demands
 - Scope disagreements
- To whom
 - Peers
 - Team Lead
 - Senior Manager



Team Performance Matrix

2 Coding Best Practice

A WHAT

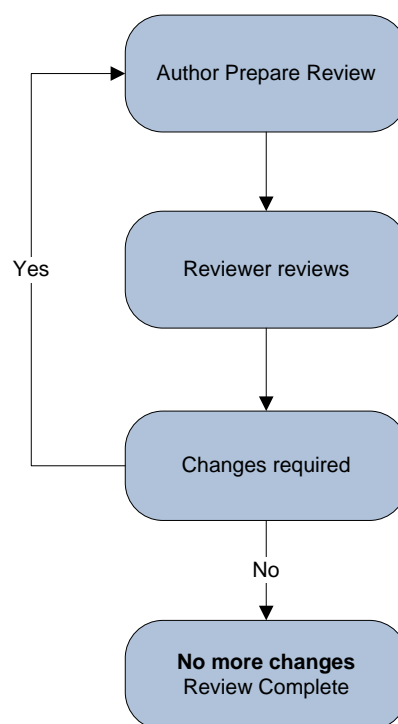
Coding Best Practice should be defined early in a project's life cycle. It provides a set of practices that will help with,

- Finding defects
- Maintaining code
- Readability
- Code review

The best practice should cover

- A common style
- Naming conventions
- Code review

CODE REVIEW



Code Review is a process that

- Finds defects early in the life cycle
- Reviews against Coding Best Practice
- Provides learning opportunities for junior and new members of teams.

Code Review can take place in the following ways:

- Formal – The team sit down in a meeting and review code as a team
- Light Weight Review – Ad-hoc review by another developer
- XP Pair Programming – review as you code

Before starting code reviews the project's Coding Best Practice should be defined

B EXAMPLE CODING BEST PRACTICE

Different languages have different Coding Best Practice. When working with a language, it's advisable to know what the standards are, as most projects will be based upon them. For Java standards see below,

- <http://www.oracle.com/technetwork/java/codeconv-138413.html>

These cover, filenames, package structure, indentation, declarations, white space and much more.

C CODE REVIEW BEST PRACTICE

Below is an example check list of items that should be looked for during code review.

General	Yes/No
Does the code meet the requirement?	
Does the code implement more than the requirement?	
What testing has taken place to prove that it meets the requirement?	
Variable and Constant Declarations	
Are the Coding and Naming conventions followed?	
Are the names used conveying how they are used?	
Are all variables correctly typed?	
Are all variables correctly initialised?	
Are correct access modifiers (private, protected, public) used?	
Are constant variables declared correctly?	
Are static and non-static variables declared correctly?	
Method Definitions	
Are the Coding and Naming conventions followed?	
Do the method names convey how the methods are used?	
Are correct access modifiers (private, protected, public) used?	
Are static and non-static methods declared correctly?	
Class Definition	
Does each class have the correct constructor?	
Does any subclass have a common class variable or method that should be in the superclass?	
Can the class inheritance hierarchy be simplified?	
Arrays	
Is every reference within the defined bounds of the array?	
For an object array, is every reference certain not to be null?	
Numeric Computation	
Are mixed data types used (Long, Double, etc.)?	
For expressions with more than one operator, is the ordering of the calculation correct?	
Are parentheses used to avoid ambiguity and make readable?	
Is rounding handled correctly?	
Is division by zero handled correctly?	

Comparisons	
For Boolean checks, is the correct condition being checked?	
Have all side effects of a comparison been considered?	
Are comparisons constructed correctly; is && used consistently for the logical operator AND?	
Are 'less than', 'greater than' and 'equal to' covered?	
Control Flow	
Is the best construct for loops being used?	
Will all loops terminate?	
When there are multiple exit points from a loop, are they all required and handled correctly?	
Does each switch statement have a default clause?	
Does the switch statement have any missing break clauses?	
Can the nesting of loops be simplified to make code more readable?	
Can if statements be refactored?	
Are all exceptions handled correctly?	
I/O	
Have all files been opened before use?	
Are the correct attributes used to open the file?	
Have files been closed after use?	
Is buffered data flushed?	
Are exception handled?	
Are files checked for existence before an attempt is made to access them?	
Comments	
Are comments added to aid in understanding of code? Can the reviewer understand?	
Are classes commented?	
Are class variables commented?	
Are methods commented?	
Do comments add value?	
Formatting	
Is the defined style used for formatting of code?	
Does each method have a single responsibility? (There can be exceptions).	
Does each class have a single responsibility (if not split)?	
Performance	
Could better or more efficient algorithms be used?	
Are logical tests arranged for performance?	
Can the cost of computing a value be reduced by computing once and storing the results?	
Are all results used?	
Can a computation be moved outside of a loop?	
Is all code required within a loop or could it be moved outside?	
Are there two loops operating on the same data that could be rolled into one?	
JAVA General	
'Is the code written to use the interface rather than the implementation?'	
Exception Handling	
Are exceptions handled correctly?	
Logging	
Is all logging required?	
Could any logging usefully be added?	

Is logging meaningful?	
Does logging provide all context required?	
Threading	
If a class is used by multiple threads, is it thread-safe?	
If a method is used by multiple threads, is it thread safe?	
Are thread-safe collections used, where required?	

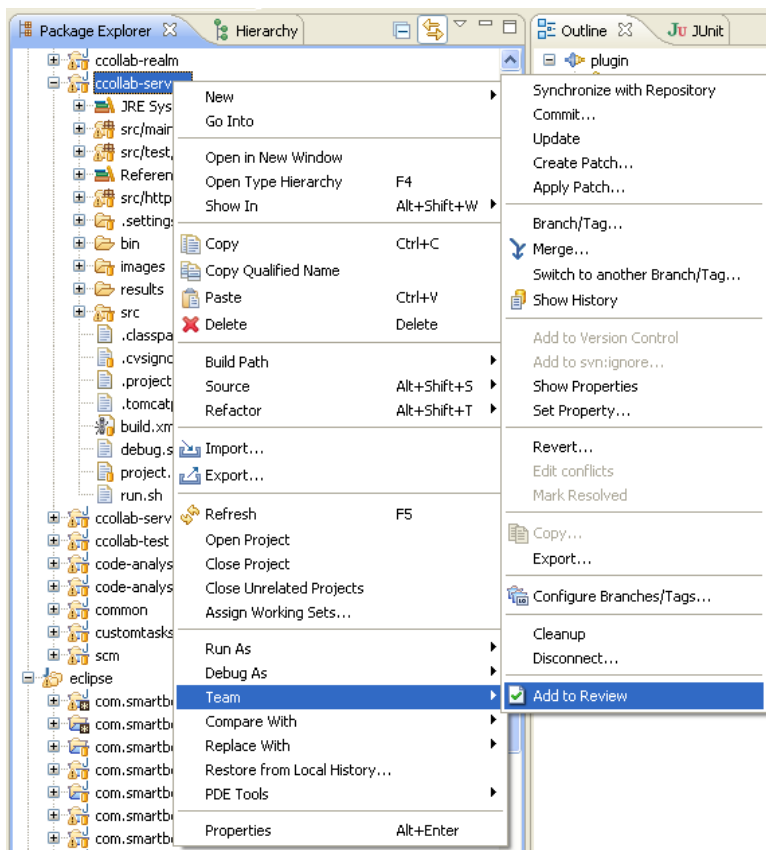
D CODE REVIEW TOOLS – CODE COLLABORATOR

Code Collaborator is a tool that provides an online process for code review. It also has a plug-in for Eclipse (see Section 4.1).

CREATING A REVIEW

The following looks at the steps that are taken to submit the review to Code Collaborator using the Eclipse Plug-in.

To start creating a review right-click on any **Working Sets, Projects, Folders, or Files** and select **Add to Review** from the **Team** menu.

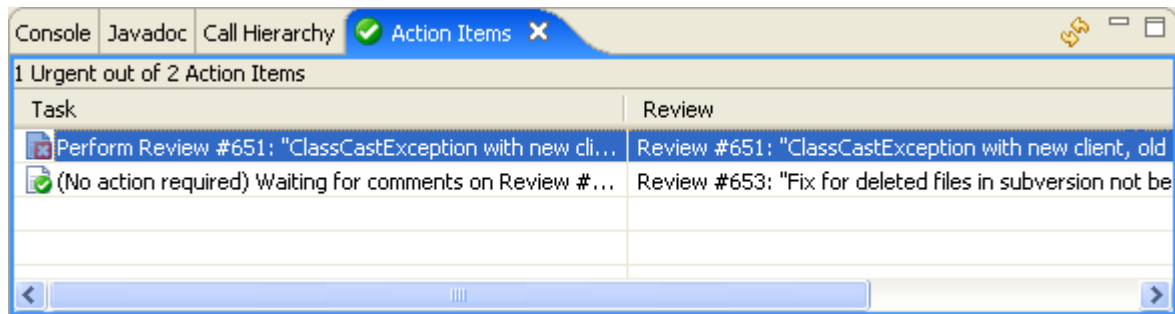


This will launch the **Add files to Review** Wizard. This allows you to create a new review or add or modify files to an existing review.

ACTION ITEMS

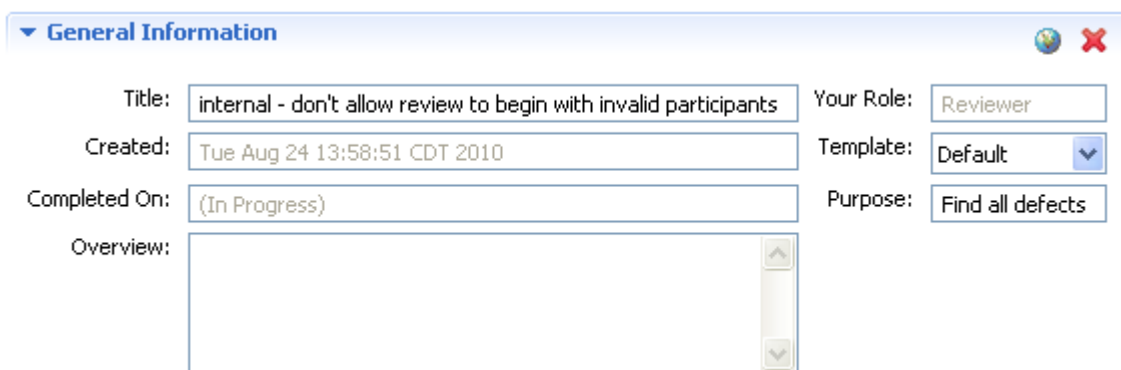
To open your Code Collaborator Action Items in Eclipse, select *Window -> Show View -> Action Items*.

Your Code Collaborator Action Items appear in Eclipse as a View:



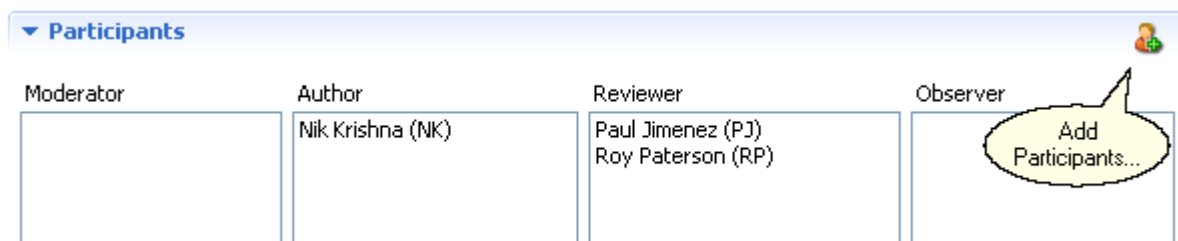
REVIEW EDITOR

Double-clicking an Action Item will open the Review in the Review Editor.



General Information			
Title:	internal - don't allow review to begin with invalid participants	Your Role:	Reviewer
Created:	Tue Aug 24 13:58:51 CDT 2010	Template:	Default
Completed On:	(In Progress)	Purpose:	Find all defects
Overview:			

The General Information section shows you information about the Review as a whole, and lets you edit certain fields.

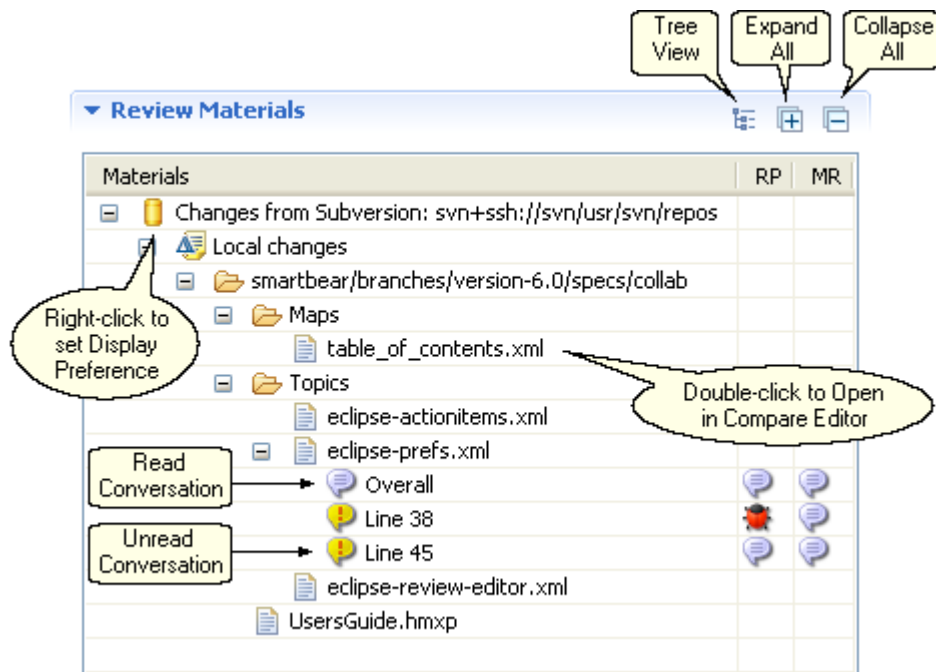


Participants			
Moderator	Author	Reviewer	Observer
	Nik Krishna (NK)	Paul Jimenez (PJ) Roy Paterson (RP)	Add Participants...

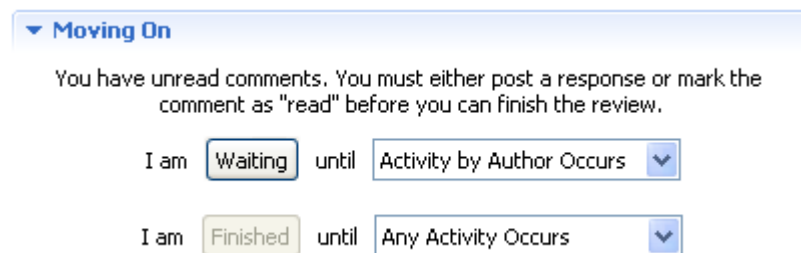
When creating a review you will need to add a set of Participants.

- Moderator – someone there to make sure that review is taking place
- Author – the developer who created the review
- Reviewer – the developer(s) who will be completing the review
- Observer – Developers who may want to see what actions arise from review.

Click the Add Participants... icon in the toolbar of the Participants section to toggle the section for adding a new Participant to the Review.



The Review Materials section shows the changelists and files which are part of the Review. Double-click on a file to open it in the Compare Editor.



The Moving On section allows you to set whether you are waiting for responses or finished with the review.

E FURTHER INFORMATION ON CODING STANDARDS

.NET 4.0

<http://msdn.microsoft.com/en-us/library/ms229042.aspx>

Windows 7 GUI Guide lines

<http://msdn.microsoft.com/en-us/library/windows/desktop/aa511440.aspx>

Google C++ Style Guide

<http://google-styleguide.googlecode.com/svn/trunk/cppguide.xml>

Recommended C Style and Coding Standards

<http://www.doc.ic.ac.uk/lab/cplus/cstyle.html>

PHP Coding Guidelines

<http://www.phpdeveloper.org.uk/articles/php-coding-guidelines/>

Google JavaScript Style Guide

<http://google-styleguide.googlecode.com/svn/trunk/javascriptguide.xml>

All Google Standards (includes CPP, C, python, java, java script, JSON, objectC, XML)

<http://google-styleguide.googlecode.com/svn/trunk/>

3 Unit Testing

A WHAT IS UNIT TESTING

- Unit testing means the testing of the smallest 'units' of code – classes, methods (as opposed to larger scale integration, system, and functional testing).
- Unit tests are written and run by software developers to ensure that code meets its design and behaves as intended.
- Should be code-based to remove manual error and easily permit repeat execution (though may be associated with an element of manual code review or 'static testing').

B WHY WRITE UNIT TESTS

- The sooner issues are found, the cheaper they are to fix. Bugs detected in QA cycles or after release require much greater effort to fix (further QA testing, release effort, associated planning).
- Aids a software developer in thinking through the changes they are making, encouraging them to explore the possibilities and implications for the system from the changes they make.
- Can help increase the modularity of software, as it's easier to write unit tests when the software is well modularised.
- A base of automated unit tests can be re-used in future, helping detect accidental introduction of bugs.

C WHEN TO WRITE UNIT TESTS

- Continually. It's not always possible to write unit tests for all sections of code, particularly those embedded within existing systems, however it should always be attempted and tests written whenever possible.

D HOW TO WRITE UNIT TESTS

- Analyse the code required to meet a requirement. Decide which methods and classes may need to be created or modified to meet that requirement. Look at what the code should do, and the boundary conditions and error conditions it should handle. Formulate tests for the methods, and implement them using a unit testing framework, preferably one that integrates to your IDE.
- Test Driven Development advocates the writing of test code prior to the actual program code.

E UNIT TEST FRAMEWORKS

- There are many unit testing frameworks available – most popular are the XUnit family (JUnit, NUnit, CPPUNIT, etc).
- It's advisable to choose one that
 - is widely adopted
 - offers good IDE integration
 - has good references and support groups online (email forums etc.)

E EXAMPLE: JAVA UNIT TEST USING JUNIT

```
public class PastryChefTest {  
  
    PastryChef chef;  
    List<String> ingredients;  
  
    @Before  
    public void hireChef() {  
        chef = new PastryChef();  
        ingredients = new LinkedList<String>();  
        ingredients.add("Apples");  
        ingredients.add("Flour");  
        ingredients.add("Sugar");  
        ingredients.add("Eggs");  
    }  
  
    @Test  
    public void testGoodPie() {  
        ApplePie pie = chef.bakePie(ingredients, 150, 60);  
        Assert.assertTrue(pie.isTasty());  
    }  
  
    @Test  
    public void testUndercookedPie() {  
        ApplePie pie = chef.bakePie(ingredients, 50, 60);  
        Assert.assertFalse(pie.isTasty());  
    }  
  
    @Test(expected=IngredientsException.class)  
    public void testMissingIngredients() {  
        ingredients.remove("Apples");  
        ApplePie pie = chef.bakePie(ingredients, 150, 60);  
    }  
  
    @Test(expected=OutOfRangeException.class)  
    public void test1MillionDegrees() {  
        ApplePie pie = chef.bakePie(ingredients, 100000, 60);  
    }  
}
```

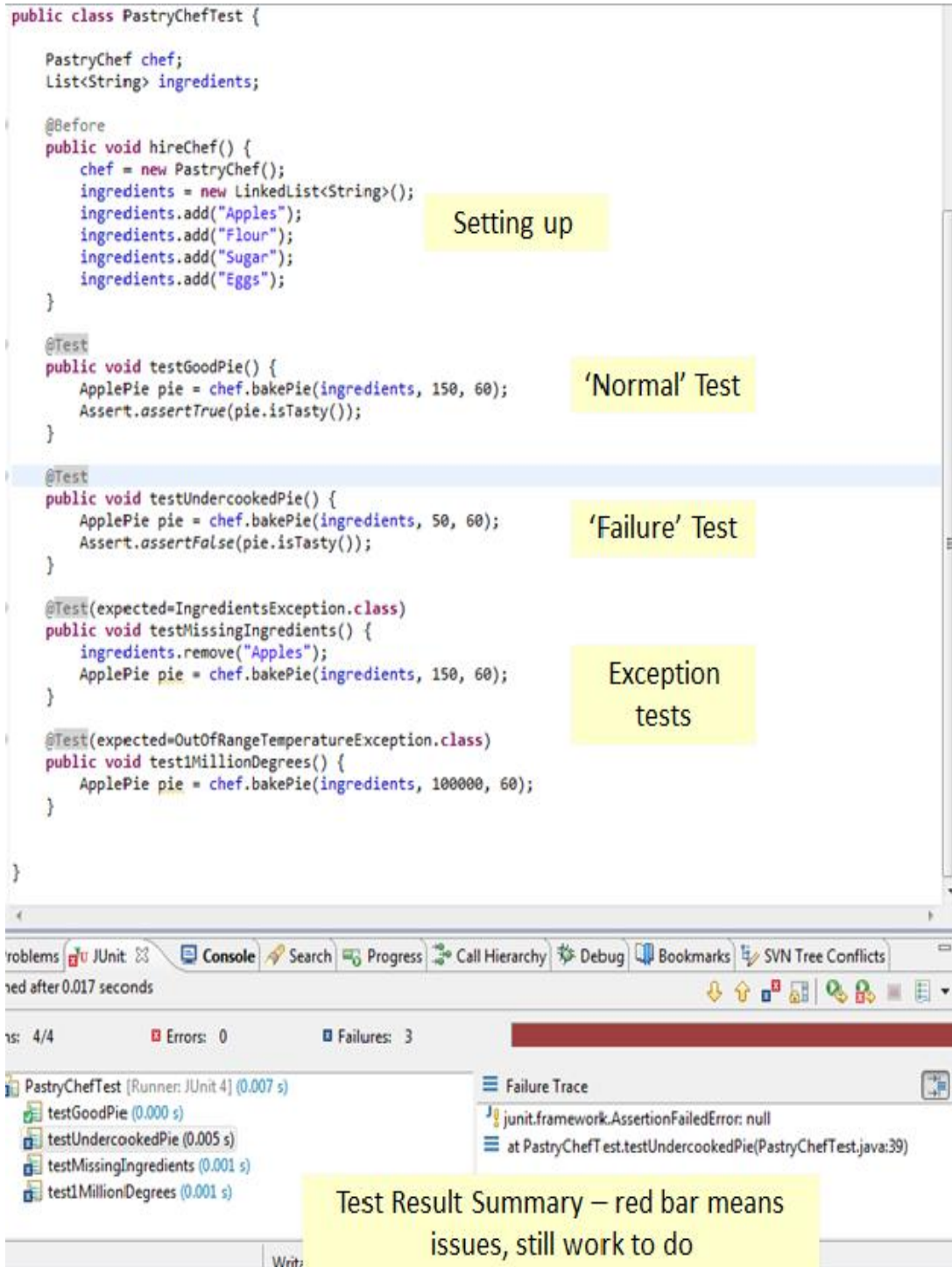
Setting up

'Normal' Test

'Failure' Test

Exception tests

Test Result Summary – red bar means issues, still work to do



Problems JUnit Console Search Progress Call Hierarchy Debug Bookmarks SVN Tree Conflicts

red after 0.017 seconds

Tests: 4/4 Errors: 0 Failures: 3

PastryChefTest [Runner: JUnit 4] (0.007 s)

- testGoodPie (0.000 s)
- testUndercookedPie (0.005 s)
- testMissingIngredients (0.001 s)
- test1MillionDegrees (0.001 s)

Failure Trace

junit.framework.AssertionFailedError: null

at PastryChefTest.testUndercookedPie(PastryChefTest.java:39)

4 Tools

4.1 Eclipse

A WHAT

- An Open-Source IDE used widely for Java Development
- Plug-in based, so it supports a wide variety of utilities and different languages

B KEY CONCEPTS

- Workspace - the Central Repository
- Perspective - a group of related views related to a specific purpose (e.g. java Development, Debugging, etc.)
- View - a single window with a dedicated purpose (e.g. Editor, Package Explorer, etc.)

C KEY PERSPECTIVES

- Java - Java Development, including the Project/Package Explorer, Console & Editor
- Debug - Application Debugging, including Breakpoint Manager, Variable Explorer, etc.
- SVN Repositories - SVN Repository Management
- Team Synchronizing - Source Code Control Management, synching with & committing to a repository

D KEY ACTIVITIES

CHANGE PERSPECTIVE

- “Window -> Open Perspective”
- Choose the perspective to open

Once a perspective has been opened a shortcut button is available in the top right of the window

CREATING A JAVA PROJECT

- “File -> New -> Java Project” **OR** “Right Click in Package Explorer -> New -> Java Project”
- Enter Project Name
- Choose Java Version/Runtime
- Choose Structure (recommended that “Create separate folders for sources and class files” is used)
- Create source folders (recommended that just “src” is used for simple projects.

CREATING A PACKAGE

- Click Source Folder, then “File -> New -> Package” **OR** “Right Click on Source Folder -> New -> Package”

- Enter Package Name

CREATING A CLASS

- Click Package, then “File -> New -> Class” **OR** “Right Click on Package -> New -> Class”
- Enter Class Name
- Choose Superclass & Inherited Interfaces if any

RENAMING A PROJECT/PACKAGE/CLASS/FILE

- Right click the project/package/class, then “Refactor -> Rename”
- Enter the new Name

CONFIGURING THE PROJECT CLASSPATH


Right click the project, then “Build Path -> Configure Build Path”

- To Add a JAR inside the project - From the “Libraries” Tab click “Add JAR”, browse to the JAR file, click OK
- To Add a JAR outside of the project - From the “Libraries” Tab click “Add External JAR”, browse to the JAR file, click OK
- To Remove a JAR - From the “Libraries” Tab click the JAR, click “Remove”
- To Add another Project - From the “Projects” Tab click “Add”, choose the Project, click OK
- To Remove another Project - From the “Projects” Tab click the Project, click “Remove”

COMPILING CODE


It’s recommended that you enable “Project -> Build Automatically”: when a file is changed, it will be compiled.

- To request a build use “Project -> Build All”
- To request a clean (remove all compiled resources), “Project -> Clean”


Compilations errors will be shown in the “Problems” window (also see error icons  in the Package Explorer)






RUNNING A “MAIN” CLASS

To run normally

- Right click the class and select “Run As -> Java Application”
OR (if the class is in the Editor) click the play icon 

To run in “Debug Mode”

- Right click the Class and select “Debug As -> Java Application”
OR (if the class is in the Editor) click the debug icon 
- You will be prompted to change to the “Debug” perspective
- When breakpoints are reached (see later) the code will freeze, and you can then view the stack and any variable values



- To move the code on
 -  = resume executing until next breakpoint or application finish
 -  = stop the application
 -  = “step in” i.e. step into the execution of the current line, where it is a method call
 -  = “step return” i.e. step out of the current method to where it was called from
 -  = “step over” i.e. step over the current line to the next one

Any “System.out.println” output will go to the “Console” window

SETUP A “RUN CONFIGURATION”

Run configurations are used when you want to pass parameters to the Java application, or customize the runtime (override the classpath, JVM² parameters, etc.)

- “Run -> Run Configurations”
- Click “Java Applications” (left menu), then Click the “New” button
- On the “Main” tab, set the Name (for display), the project and the “main” class
- On the “Arguments” tab, specify any arguments
- Click Run



Once run the configuration can be accessed quickly from the “Play”  or “Debug”  dropdowns

SETTING A BREAKPOINT

Open the Class in the Editor

Identify the line that you want to introduce the breakpoint at

Right-click in the margin to the left of the line and choose “Toggle Breakpoint”

-  = breakpoint active
-  = breakpoint disabled

² Java Virtual Machine: the component of the Java software platform that executes your code.

4.2 Subclipse

A WHAT

- An Eclipse Plug-in for working with Subversion Repositories

B KEY CONCEPTS

- “SVN Repositories” Perspective: the perspective for viewing/managing the repository
- “Team Synchronizing” Perspective: the perspective for managing “checked out” source (used if you want to get updates, commit changes, etc.)

C GENERAL PRINCIPLES

- Synchronize regularly to minimize conflicts (which, if they occur, will result in your manually having to resolve differences)
- Always synchronize before you need to commit changes
- Always update incoming changes & resolve conflicts before you commit your code
- Always make sure the code compiles before you commit (after incoming changes & conflicts are resolved)
- Always include a comment when you commit, to illustrate what your change is for
- Commit all your related changes in one go (select multiple files) so it commits as a single change
- If in doubt check the history (“Java” Perspective, Right-click, then “Team -> Show History”): this will let you see how the file has changed over time, who changed it and the comment they committed it with. You can even compare different versions to see the changes between them.

D KEY ACTIVITIES

SETUP CONNECTION TO THE REPOSITORY

- Open the “SVN Repositories” Perspective
- Right click in the “SVN Repositories” View, choose “New -> Repository Location”
- Enter the URL of the Repository
- When you next try to access the repository you will be prompted for your username and password
- Note that **if you choose to save the password in Eclipse it will be stored in clear text.**

SHARING A PROJECT (UPLOAD TO THE REPOSITORY)




- Open the “Java” Perspective
- Right-click on the Project (here we assume that you have selected one that has not been uploaded to the repository before)
- Choose “Team -> Share Project”
- Choose the Repository Type (SVN in this case)
- Choose the Repository
- Choose the location in the Repository
 - Its recommended to use the “Use Specified Folder Name” option
 - The location should be relative to the repository root e.g. “trunk/projects/MyProject”

Note that at this point you have only created the root directory in the repository, and linked your local project: **nothing has yet been committed/uploaded**. See “committing changes below” for details on how to upload/commit.

CHECKING OUT A PROJECT

- Open the “SVN Repositories” Perspective
- Browse through the repository to the location of the project
- Right-click the directory and choose “Checkout”
- Choose the option “Check out as a project in the workspace”
- Click “Finish”

SYNCHRONIZING (GETTING/COMMITTING CHANGES & RESOLVING CONFLICTS)

- Open the “Team Synchronizing” Perspective
- Click the “Synchronize SVN” button in the “Synchronize” view
- If this is the first attempt, you’ll be prompted to choose the repository type (SVN) and which projects to synch on (choose all projects.)
- Once complete, the “Synchronize” view will display all changes (you can double click the file to see a side-by-side comparison of changes)
 -  = Incoming, changes committed to the repository by others. Right-click and “Update” to retrieve the change and apply it locally
 -  = Outgoing, uncommitted changes made by you. Right-click and “Commit” to commit the change to the repository
 -  = Conflicting, the file has Incoming and Outgoing changes
 - Double click the file to get a side by side comparison
 - Apply any incoming changes to your local copy & save (Left = Local, Right = Remote)
 - Right-click the file and choose “Mark as Merged”
 - It will now show as an Outgoing change you can commit normally

For icons above, an imbedded +/- indicates the addition/deletion of a file.

4.3 Ant

A WHAT

- An Open-Source tool for compiling Java source code into executable code
- Provides means to perform many additional tasks such as executing unit tests, formatting test results and creating a distributable file of the compiled code

B KEY CONCEPTS

- Ant³ is a tool and an XML-based scripting language for creating build scripts to automate the compilation of code
- An Ant build script can be used to drive Continuous Integration jobs
- An Ant build script defines the Ant 'project'. The project is made up of a number of Ant 'targets'. Targets are made up of Ant 'tasks'
- There is a default target for the project. This is what runs when Ant runs the script (in the absence of an explicit target being supplied). The default target will have dependencies on other targets in the project.
- Ant tasks cover many different areas such as compilation tasks, file tasks, archiving tasks

C KEY ACTIVITIES

CREATING A BUILD SCRIPT MANUALLY

- In the Java project in Eclipse select "File -> New -> File"
- Enter File Name as `build.xml`
- Define the project name in the script with a default target
- Define the default target, for example a 'compile' target
- Add another target as a dependency of the default target

Here is a simple example that compiles Java source code into executable code. Prior to the compilation, any previously compiled code is cleared out:

```
<project basedir="." default="compile" name="SimpleAntBuild">

  <!-- Compiles the project -->
  <target name="compile" depends="clean">
    <javac destdir="target/classes">
      <src path="src/main/java" />
    </javac>
  </target>

  <!-- Cleans the project -->
  <target name="clean">
    <delete dir="target" />
    <mkdir dir="target/classes" />
    <copy includeemptydirs="false" todir="target/classes">
      <fileset dir="src/main/java" excludes="**/*.java" />
    </copy>
  </target>
</project>
```

³ <http://ant.apache.org/>

```
</target>  
</project>
```

CREATING A BUILD SCRIPT AUTOMATICALLY

- Eclipse can generate a standard Ant build script from an existing project
- In the Java project in Eclipse select “File -> Export -> General -> Ant Buildfiles”
- Enter File Name as `build.xml`
- Edit the generated file as appropriate

RUNNING A BUILD SCRIPT

- To run the default target, right click on the `build.xml` script in Eclipse, select “Run As -> Ant Build”

4.4 Jenkins

A WHAT

- An Open-Source tool for the continuous build of software projects, commonly referred to as 'Continuous Integration' or CI
- Browser based and supports a wide variety of languages



B KEY CONCEPTS

- CI improves developer and team productivity by
 - Removing manual build steps
 - Automating deployment
 - Detecting build breaks sooner
 - Reporting failing tests more clearly
 - Increasing visibility of progress
 - Providing build metrics over time
- Jobs - the software projects to be built through Jenkins via a tool such as Ant
- SCM – software configuration management tool, e.g. Subversion, that stores the software project to be built
- Notifications – how results of a build are relayed to the team
- Plug-ins – tools intended to extend Jenkins functionality by providing additional features (e.g. Twitter integration, etc.)

CREATING A JOB

- Click “New Job” on left-hand menu in Jenkins
- Enter a Job Name
- Select “Build a free-style software project”
- Enter a description of the job
- Click “Enable project-based security”
- Add user/group to the project with relevant security authorisations
- Select an appropriate SCM provider under “Source Code Management”, e.g. Subversion
- Enter “Repository URL” of the Subversion repository
- Enter authorisation credentials for Subversion
- Choose an appropriate build triggering approach under “Build Triggers”, e.g. “Build periodically”
- For “Build periodically”, enter a build schedule in standard CRON format, e.g. build every day at 6pm
- Click “Add build step”
- Select “Invoke Ant”
- Select Ant Version
- Leave “Targets” blank to run Ant default target
- Enter Recipients to receive build notifications under “Post-build Actions”
- Select “Send e-mail for every unstable build”
- Click “Save”

MANUALLY RUNNING A JOB

- In the Jobs list on Jenkins’ ‘dashboard’⁴, click  next to the job you wish to run
- To see the progress of the build, Click the running job on Build Queue, and in the Build History, click the progress bar to see the console output. If the job has finished, hold the cursor over the job in the Build History and click  Console Output .

⁴ The ‘home screen’ for the application, found under the Jenkins tab.

MANAGING JENKINS

- Click “Manage Jenkins” from the left-hand menu
- Click “Configure System” to select security options, identify the tools you are going to use in the build process (your Java JDK⁵ and Ant installations; your preferred version of Subversion), and set other important parameters (the URL for Jenkins; details of your email server)
- Click “Manage Plug-ins” to add, remove, enable or disable plug-ins to extend the functionality of Jenkins
- Click “Manage Nodes” to add, remove or monitor other nodes used to run Jenkins jobs
- Click “Disk usage” to monitor the space consumed by Jenkins and build jobs
- Click “Monitoring of Hudson/Jenkins master” to get statistics on the performance of the Jenkins CI server and any connected nodes

⁵ Oracle’s Java Development Kit

4.5 EclEmma – Eclipse Plugin

A WHAT

- An Eclipse Plug-in that generates “Code Coverage” Metrics for your unit-test suite
- Supports a drilldown from package-level, right down to method level
- Highlights covered/uncovered code

B KEY CONCEPTS


- Test coverage – a set of metrics produced against software code to identify how much of that code is covered by unit tests and more importantly how much is untested

C GENERAL PRINCIPLES

- Use EclEmma as part of your “check in” routine
- Address gaps in the coverage as soon as possible
- Set a **minimum** benchmark for coverage, and clearly define exceptions (e.g. Generated code)
- 100% coverage is fantastic but rarely possible given how EclEmma works, aim for the high 90s

D KEY ACTIVITIES

RUNNING ECL EMMA

- Ensure the EclEmma Plugin is installed
 - If not, it can be installed by
 - Help -> Eclipse Marketplace
 - Search for “EclEmma” and Install the Plugin
- Open the “Java” Perspective
- Right click the project/test package/test class, then “Run As -> JUnit Test”
- Re-Run the Unit Tests with ECL Emma Using the Launcher ( on the toolbar)
- The “Code Coverage” Window should now appear, if not open it with
 - Window -> Show View -> Other
 - Java -> Coverage
- You can now use the view to drill down and view the following per project/package/class/method
 - Coverage %
 - Instructions Covered/Missed/Total
- You can double click any class/method from this view to open it in the editor and see the covered (green highlight) & missed (red highlight) instructions. Yellow highlighted lines are “in doubt”

4.6 FindBugs – Eclipse Plugin

A WHAT

- An Eclipse Plug-in that performs static code analysis
- Generates a report with a list of reported issues, including the type, severity and a full description of why it is considered an issue

B KEY CONCEPTS

- Static code analysis – a set of rules run against software code to report on its quality, robustness, and adherence to standards

C GENERAL PRINCIPLES

- Use FindBugs as part of your “check in” routine
- Attempt to address any issues as soon as their found.
- Disable any rules you (really) consider to be unsuitable
- Set a **maximum** benchmark for bugs in terms of the number and severity detected

D KEY ACTIVITIES

RUNNING FINDBUGS

- Ensure the FindBugs Plugin is installed
 - If not, it can be installed by
 - Help -> Eclipse Marketplace
 - Search for “FindBugs” and Install the Plugin
- Open the “Java” Perspective
- Open the “Bug Explorer” Window if not already open
 - Window -> Show View -> Other
 - FindBugs -> Bug Explorer
- Right click the project/package/ class, then “FindBugs -> FindBugs”
- You can now use the view to view the bugs that were found, organised by project and type
- You can double click any of the warnings 🚩 to show the line/lines in your code that the bug relates to
- You can Right-Click -> “Show Bug Info” any of the warnings 🚩 to show details of why it is considered a bug.

DISABLING A RULE

- Only disable a rule if you’re sure its unsuitable and after review with your peers
- Window -> Preferences
- Java -> FindBugs
- Locate the rule in the “Detector Configuration” tab and uncheck it.

5 Quality Assurance

5.1 Introduction to Quality Assurance

A WHAT

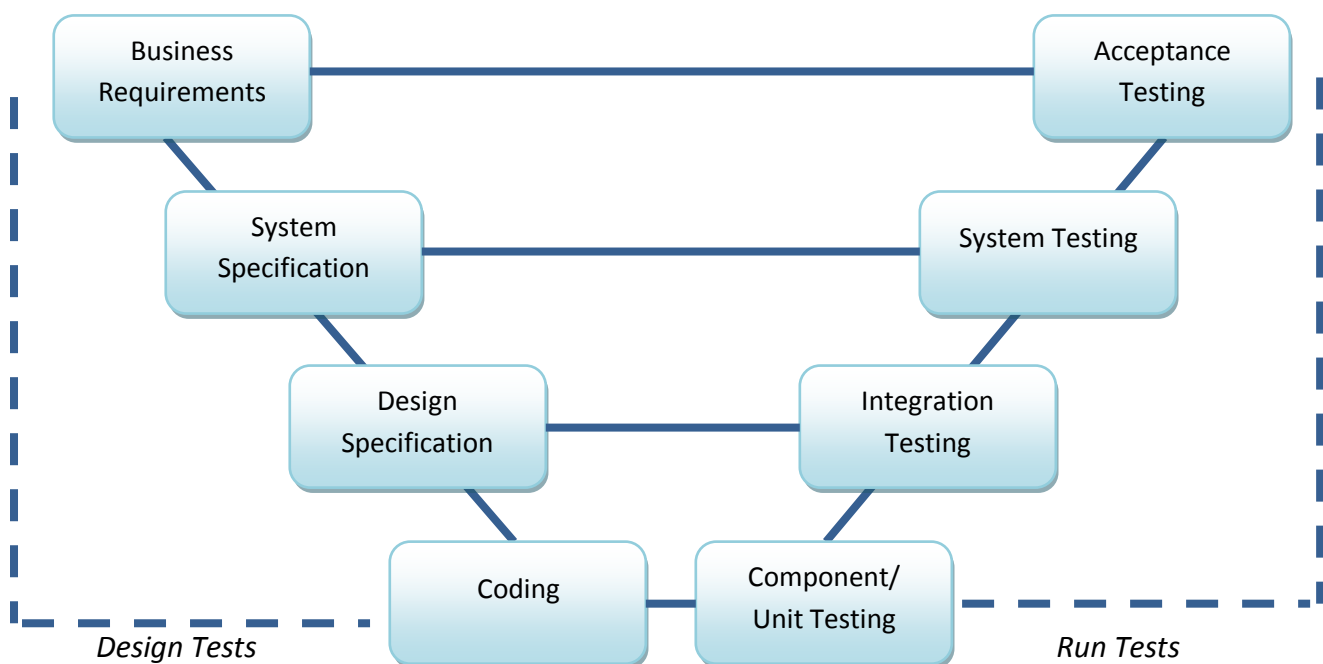
Testing is used to measure the **quality** of software both in terms of what it does (e.g. functionality) and how it does it (e.g. performance). It is also important that the test cases used in this process are of high quality i.e. if you have poor test cases it may indicate software quality is good when in reality there could be underlying issues.

Testing identifies issues in the code base i.e. defects. The earlier these defects can be found the more time and money is saved. By having a well tested application you hope to reduce the risk of issues in the 'live' production environment. If issues are found by the users they are referred to as bugs and have the potential to cause reputational risk and financial loss.

B WHERE QA FITS IN TO THE SOFTWARE DEVELOPMENT LIFECYCLE

V-MODEL – EARLY TEST DESIGN

The V-Model is often referred to in the SDLC and clearly emphasizes the importance of testing from a development (Unit), QA function and user testing perspective.



For the purposes of understanding the QA function the key here is that as soon as a requirement is known QA can start the creation of test cases. It is by starting to plan test cases that you can challenge any assumptions and gain a very in-depth understanding of a change. As mentioned previously test case quality is key for determining software quality

therefore it is essential that due time is given to writing test cases. It is also important that all test cases have very detailed expected results so that there is no ambiguity when it comes to determining the outcome from test execution.

C TYPES OF TESTING

OVERVIEW OF TESTING TYPES

Table follows on next page.

RACI : Responsible, Accountable, Consulted, Informed matrix – describes the participation by various roles in completing deliverables for a project. It is especially useful in clarifying roles and responsibilities in cross-functional/departmental projects and processes.

Test Type	Objective	On failure	RACI Matrix for Responsibilities				Automation
			Dev	QA	Prod Dev	Beta	
Unit	Ensure that code written by developers conforms to the user requirements and functional designs.	Developer fixes defect that caused unit test to fail.	A & R	I	I		Yes – Unit testing can not be effectively executed or reported on as a manual activity.
Smoke	When code is deployed into an environment for the first time a Smoke Test will be carried out with the objective of determining, at a high level, are all key components functioning as expected and whether the build is stable for further testing.	Defect should be raised against the requirement that failed. Build should be backed out of environment so that Testing may continue on previous build.	I	R	I		Yes – initially smoke testing is a manual process however as it is repeated process it should be automated as soon as possible.
Functional	Ensure that new functionality works as expected.	Defect should be raised against the requirement that failed.	I	R	A		No – new functionality is prone to change and therefore does not form a good basis for automation.
Non-functional	Ensure that the application can support the non functional aspects of the design. i.e. Performance, Scalability, Security and Usability.	Defect should be raised against the requirement that failed.	R	A	C	I	Yes –performance, scalability and security testing must be automated in order to simulate a production environment. Usability

	Note: Usability testing forms the core of User Acceptance Testing, see below.						is typically a manual testing exercise.
User Acceptance Testing	A form of functional Testing usually undertaken by a nominated set of Business Users who will verify the application is fit for purpose rather than validating that the application meets the documented requirements.	Defect should be raised against the requirement that failed.	I	C	R	A	No – UAT testing is a manual process that relies on the users experience to test to application.
Regression	This applies across ALL testing types and is not limited to any one group. Focus is on determining whether change has introduced a defect into a previously working component or process.	Defect should be raised against the requirement that failed.	R	R	R	R	Yes – as functionality matures and becomes stable this is the primary area for building out System Level automation.

KEY DOCUMENTS

Test Plan

A test plan is created at the start of the software development lifecycle. It is a detailed document which describes how the testing for a given project is going to be approached. This will incorporate the testing scope, the resources assigned to the project and a schedule of the different testing activities. It will also provide a list of the systems involved in the overall project or which interface with the application in question which need to be considered from an end to end testing perspective. This document will also detail the signoff criteria and who is required to provide signoff (often includes a number of business users). It is important that this document also highlights any known risks or assumptions. The purpose of this document is to ensure complete transparency on the testing process and to ensure everyone is aware of the breadth and depth to which it will cover. Given that this document is created at the outset any insufficiencies can be addressed earlier in the process to ensure the end target is not compromised.

Test Case

A test case is probably the most recognizable and familiar test artefact. It usually starts with a brief description of the change it relates to or the area of the application for which it addresses. It will also detail what pre-requisites are required to be able to actually execute the test case i.e. system access, database login etc. The main body of the test case has a simple structure i.e. Test Case Number, Summary, Description and Expected Results. It is essential when you are writing a test case to make it as detailed as possible. Always remember that a test case should be able to be taken and executed by anyone even if they are not familiar with the system. It is therefore good practice to include detailed navigation, Unix commands and sample SQL. Furthermore it should be explicitly clear from the expected results what is required to be considered as a Pass. It should never be left to personal interpretation as to whether something is actually working as expected as this opens things up to be missed. Many tools are available on the market to record test cases in however even a simple spreadsheet can be utilized effectively.

Status Report

In all software development it is essential that at any point in the process all stakeholders have a real sense of the quality of the underlying software. Given the Quality Assurance team's integral part in the overall testing of the application it is essential that their reporting is not only extremely accurate but also meaningful and concise for each stakeholder group. Given this, during the official testing phase of a project the QA team will send out a daily status report at the end of each day. This normally takes the form of an email which details the change list for a given release (Requirement/Change, Status, Test Complete %, Assigned

QA, Developer and any comments). Furthermore this status report will have a list of any open issues as well as insight into how much regression testing has been completed by the QA team on the current build. Towards the end of the test cycle these status reports are used as the central artefact for discussions with the business, development, QA and production support to determine if the release is at a high enough quality to be released into the live production environment.

TEST MANAGEMENT

Estimates

As is the case with developers estimating how much time it will take to code a new feature the Quality Assurance team also have to provide estimates on how much time they feel it will take to effectively test any new functionality for a given release. As well as this they need to determine how much regression coverage can be delivered in a given release cycle. It is essential that the QA team document and distribute this information to the key stakeholder groups to determine if the release cycle needs to be extended to ensure there is enough time to perform adequate testing.

Reporting

As mentioned above the Daily Status Report sent by the Quality Assurance Team is an essential aid to all the key stakeholders during a release cycle. However it is also key that the QA team use their experience and knowledge to escalate any potential blocking issues when they arise in the test environment. This may include escalating the fact that an interfacing application is down therefore blocking important end to end testing or could be an error string identified in the applications logs for example which is deemed critical to the overall functioning of the application.

Managing Risk

The Quality Assurance team always have to remain objective when looking at the quality of a release. The development team will naturally be under pressure to deliver to the business however the QA team are there to ensure that any associated risks are not only communicated to all stakeholders but more importantly acknowledged and understood. Ultimately if all stakeholders are willing to accept the risk then a release may proceed however formal signoffs noting this risk factor need to be in place. The QA team often have the benefit of having insight into and understanding multiple applications which interfere together. It often then comes down to the experience of the QA team to identify potential risks in the end to end flow which the development team may not have considered.

E QA CAREER AVENUES

WHY TESTING?

Testing exposes you to multiple aspects of the SDLC as well as develops many transferrable skills. Without doubt testing professionals have to be technical and understand complex concepts as well having excellent organization and management skills.

Key aspects of a job in QA:

- Exposure to technical concepts
- Problem solving
- Issue tracking/management
- Key function – understands the area, not just the components

PSYCHOLOGY OF TESTING

Testers have to have a different mindset to developers. Given that a developer writes a piece of code they can often test it in a manner which proves it 'works'. However a tester can look at this task differently given they have a separation from the underlying code base. Testers often add value and find defects by negatively testing a change. Testers also challenge the assumptions made by the developer to determine if the requirement has actually been completely met.

Testers also have the added benefit of having a wider exposure to the complete application therefore can test a given change in a number of different manners by utilizing their understanding of the overall application.

It is important that both developers and testers are aware that testing should not be seen as a negative process of fault finding. Instead it should be a collaboration with the ultimate goal being to release the best quality code for the users. If the presence of a test team can reduce the occurrences of production issues then it can most definitely be seen as a win for development as well. It is worth noting however that the earlier defects can be found in the process the more time/cost effective it is therefore developer testing is always essential.

The below can be considered as qualities of a "good" tester however there are many more:

- Curiosity
- Challenges assumptions
- Attention to detail
- Good communication skills
- Clear thinker under pressure
- Technical understanding

CAREERS OPTIONS

A career in testing is guaranteed to offer many exciting challenges and learning opportunities. Testing is a highly technical skill and the technologies used align directly with development and because of this both job functions are considered on equal par in the industry.

Possible career paths:

- Manual tester
- Technical specialist i.e. automation expert
- Team lead
- BA
- Longer term – Global Head of Testing across multiple applications or functions

Notes

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

This image shows a single sheet of white paper with horizontal blue ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]