| [Print](javascript:window.print()) | [Close](javascript:parent.window.close()) |

**Course Transcript**

Agile Programming and Testing

**Overview of Agile Programming**

| [1. Working as an Agile Programmer](http://xlibrary.skillport.com/courseware/Content/cca/sd_agsd_a04_it_enus/output/html/course_transcript.html#t4) |

| [2. Software Development Using Agile Methods](http://xlibrary.skillport.com/courseware/Content/cca/sd_agsd_a04_it_enus/output/html/course_transcript.html#t7) |

**Implementing TDD and BDD in an Agile Project**

| [1. Applying TDD and BDD in an Agile Project](http://xlibrary.skillport.com/courseware/Content/cca/sd_agsd_a04_it_enus/output/html/course_transcript.html#t10) |

**Agile Testing Strategies**

| [1. Implementing an Agile Testing Lifecycle](http://xlibrary.skillport.com/courseware/Content/cca/sd_agsd_a04_it_enus/output/html/course_transcript.html#t16) |

| [2. User Interface Testing in Agile Projects](http://xlibrary.skillport.com/courseware/Content/cca/sd_agsd_a04_it_enus/output/html/course_transcript.html#t19) |

| [3. Working with Data in an Agile Project](http://xlibrary.skillport.com/courseware/Content/cca/sd_agsd_a04_it_enus/output/html/course_transcript.html#t22) |

| [4. Using TDD, BDD, and Agile Testing Strategies](http://xlibrary.skillport.com/courseware/Content/cca/sd_agsd_a04_it_enus/output/html/course_transcript.html#t231) |

Working as an Agile Programmer

Learning Objectives

*After completing this topic, you should be able to*

* *identify tips for implementing pair programming*
* *identify tips for implementing collective ownership of code*

**1. Pair programming**

The concept of *pair programming* is an important part of many agile development methodologies. It refers to the practice of having two programmers share at each workstation so that they can readily collaborate with one another during the development process.  
  
Initially, pair programming can be difficult for a team used to traditional development processes to adopt – but it can have specific advantages.

Although the use of pair programming can take some adjustment, it enables programmers to

**produce better code**

Using pair programming generally results in higher quality code, with fewer errors. With two programmers working together, skills are combined and the likelihood of errors being spotted is greater.

**mentor junior developers**

Pair programming enables more experienced developers to mentor and train others as they work, and encourages programmers to learn from one another.

**share knowledge, and**

By using pair programming, you ensure that programmers work together, sharing ideas and knowledge.

**minimize rule shrinking**

Because pair programming involves a commitment between two people, each programmer is more likely to abide by rules and correct procedures during development.

When using pair programming, the programmer who's actively coding is referred to as the *driver*. Their partner programmer is commonly referred to as the *navigator*.  
  
It's generally best to assign a junior programmer as the driver to ensure that both programmers are productive. This way, the senior programmer can guide the junior programmer rather than taking over.

Partners should be encouraged to help drive if their drivers get stuck, or if ideas they have can be better expressed practically than verbally.  
  
They should also try to adapt to the pace and style of their drivers so as not to distract them.

It's important for drivers to acknowledge that their partners have more time than they do to analyze the code as a whole because they're not busy coding specific statements. For this reason, they might have better ideas throughout the process.

To make the best use of pair programming, you should

**introduce it gradually**

Because pair programming is a difficult concept to adopt, it should be introduced slowly. You should begin pairing only a few programmers for smaller projects, and build up to additional pairings and larger projects.  
  
You can also apply side-by-side programming, referenced in the Crystal Clear methodology as a gradual approach to pair-programming. Side-by-side programming involves programmers working beside each other and commenting on each other's work.

**avoid assigning new team members tasks of their own**

Pair programming can be used effectively to train new team members. Instead of assigning new members to their own tasks, you should pair them up with senior programmers so that they can learn through collaborating with others.

**consider pairing programmers with analysts or customers, and**

You should consider pairing programmers with analysts or customers in special circumstances. Additionally, you can pair programmers who use different programming languages. This enables the integration of various skills and types of knowledge.

**pair programmers in remote locations if necessary**

Remote pairing is possible if you have a phone line, broadband connection, or software that enables multiple users to share a single computer. You should use remote pairing if programmers in different locations have already established good working relationships, or if no local programmer is available for pairing.

Question

How should you implement pair programming?

**Options:**

1. Consider pairing programmers with analysts
2. Introduce pair programming quickly
3. Consider using remote pairing
4. Assign new team members tasks of their own

Answer

***Option 1:****Correct. You should consider pairing programmers with analysts or customers in special circumstances so that different skills and areas of knowledge are integrated.*

***Option 2:****Incorrect. You should introduce pair programming gradually because it can be a difficult practice to adopt.*

***Option 3:****Correct. Remote pairing can be useful if programmers in different locations already have good working relationships, or if no local programmer is available for pairing.*

***Option 4:****Incorrect. You should avoid assigning new team members tasks of their own. Instead you should encourage them to collaborate with others on tasks.*

**Correct answer(s):**

1. Consider pairing programmers with analysts  
3. Consider using remote pairing

**2. Collective ownership of code**

Co-ownership of code is another approach that's commonly used in agile development methodologies. It involves encouraging multiple programmers to work on a single piece of code, ensuring that all programmers understand all aspects of the code during development.

Because all programmers are empowered to work on any piece of the code, co-ownership reduces time wastage. You don't need to wait to allocate tasks to specific programmers, code is never duplicated, and if a problem needs to be fixed quickly, any programmer can be called in.  
  
In addition, the quality of the code produced is greater because skills are combined and multiple programmers look over each line of code. This ensures that more errors are identified and better standards are used.

To implement code co-ownership effectively, you should

**ensure that all programmers feel like they own all the code**

It's important that you make all programmers feel they own the entire body of code, rather than just specific sections of code they develop. It's essential that programmers work on many different parts of the code so that their skills and knowledge are spread throughout the code as a whole.

**encourage short iterations and small releases**

If you use short iterations and small releases, it's easier for programmers to take on new tasks within different sections of the code.

**ensure that programmers feel comfortable changing other programmers' code, and**

One of the benefits of co-ownership is that code can be modified by multiple programmers. It's important that programmers realize that they're free to change code created by others if doing this will improve the code.

**accept only modified code that passes testing**

Modifications of code can cause conflict between programmers. To avoid unnecessary arguments, you should stipulate that any modified code should be accepted as long as it passes testing and integrates cleanly.

Question

How should you implement collective ownership of code?

**Options:**

1. Accept changes if the code passes testing and integrates easily
2. Ensure that programmers feel comfortable modifying code that was created by other programmers
3. Encourage the use of long iterations, as well as small releases
4. Ensure that all programmers feel like they own the specific sections of code they develop

Answer

***Option 1:****Correct. To avoid unnecessary arguments about changed code, you should make it clear that any changes to code must be accepted, provided that the modified code passes testing and integrates cleanly.*

***Option 2:****Correct. To enhance development, programmers should be encouraged to change and improve code, even if they didn't originally write it.*

***Option 3:****Incorrect. You should encourage short iterations and small releases so that programmers can easily pick up tasks in different sections of the code.*

***Option 4:****Incorrect. You should ensure that all programmers feel they own all the code, rather than just particular sections of it.*

**Correct answer(s):**

1. Accept changes if the code passes testing and integrates easily  
2. Ensure that programmers feel comfortable modifying code that was created by other programmers

**3. Summary**

Pair programming involves having two programmers work at each workstation so that they can readily collaborate during development. This enables programmers to produce better code, mentor junior developers, share knowledge, and minimize rule shrinking. You should introduce pair programming gradually, avoid assigning new team members tasks of their own, consider pairing programmers with analysts or customers, and use remote pairing when needed.  
  
Encouraging co-ownership of code saves development time and increases the quality of the code produced. To use co-ownership effectively, you should ensure that all programmers feel like they own all code, encourage short iterations and small releases, and ensure that programmers feel comfortable changing other programmers' code. You should also specify that all changes to code must be accepted provided that the altered code passes testing and integrates cleanly.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_agsd_a04_it_enus/output/html/course_transcript.html#top)

Software Development Using Agile Methods

Learning Objectives

*After completing this topic, you should be able to*

* *recognize factors to consider when implementing agile practices*
* *identify the features of agile practices used in software development*

**1. Features of agile software development**

Agile software development methodologies include various practices that can be combined in a cohesive agile approach.

**Drill Down Home Page**

Five important practices comprise an agile approach.

**Page 1 of 2: Build automation**

Automation refers to the process of automating the compiling of all completed code. This reduces repetitive manual coding tasks, replacing them with a simple push-of the-button task that can be easily performed by all programmers.   
  
The automated build process should be extremely quick to learn and perform, and should be easily accessed on any programmer's workstation.

**Page 2 of 2: Build automation**

Build automation reduces reliance on the skills of individual programmers, which creates greater flexibility of development and better response times. It also minimizes unnecessary and mundane tasks, speeding up development and increasing motivation.

The build automation process also speeds up production because developers don't need to deal with convergence and compilation issues.

**Page 1 of 2: Automated deployment**

Automated deployment refers to the automation of tasks leading up to and including software deployment. The aim of this process is to simplify the management of code, from development through to testing and production.

**Page 2 of 2: Automated deployment**

Automated deployment enables you to provide access to code more quickly and to reduce deployment time in comparison to manual deployments. Because it's an automated process, it also reduces the risk of errors and isn't reliant on programmers having specific knowledge.

It's important that you perform regular automated deployments if you're using the agile approach, to comply with the need for short iterations.

**Page 1 of 1: Continuous integration**

Continuous integration is a process in which you continually integrate new functions with a clean code base. Functions are added as soon as they are complete, so the original code is regularly updated with smaller modifications.

Small updates reduce the amount of errors that occur and ensure that if errors do occur, they can be more easily identified and corrected.

You should always use continuous integration with build automation and automated testing practices. You should also check code regularly and ensure that you don't move too far from the clean code base so that it's integrity is maintained.

**Page 1 of 1: Simple design**

An agile approach emphasizes the need for simple design that focuses on meeting current needs, rather than on anticipating possible future needs. This ensures that coding is always relevant and that time isn't wasted on developing features to meet needs that will change or become redundant before software is released.

Ensuring simple design enables you to modify code more quickly, and provides a gentler learning curve for new team members.

**Page 1 of 1: Refactoring**

Refactoring involves making changes to the design of code. In an agile approach, programmers are encouraged to do this at any stage during development when design improvements can be made. This results in better quality code because programmers revisit code and make improvements once they've gained more insight into the functioning of a final software product.

Refactoring also makes other agile practices work better, encourages the use of iterations, and can be used to recover systems that have been poorly coded.

Question

Match each agile practice to the correct description.

**Options:**

1. Build automation
2. Automated deployment
3. Continuous integration
4. Refactoring
5. Simple design

**Targets:**

1. Repetitive tasks are minimized and replaced with simple automated tasks
2. Code management is simplified throughout a project
3. Small updates are made to a clean code base
4. Code design is modified throughout development
5. Design is implemented based on present needs

Answer

*Build automation reduces repetitive manual coding tasks, replacing them with simple tasks that can be performed quickly by all programmers.*

*Automated deployment is an automated process that minimizes the effort required to manage code, from development through to testing and production.*

*When you use continuous integration, you add functions to a code base as soon as they are complete. So the original code is regularly updated.*

*Refactoring involves changing code to make design improvements. Agile programmers are encouraged to do this throughout the development process.*

*The agile approach focuses on simple design, with developers encouraged to focus on meeting current needs and to avoid planning too far in advance.*

**Correct answer(s):**

Target 1 = Option A

Target 2 = Option B

Target 3 = Option C

Target 4 = Option D

Target 5 = Option E

**2. Implementing agile practices**

To make the most of the agile approach, it's important that you implement the practices it comprises effectively. There are various recommendations and guidelines that you can follow to do this.

To implement build automation, you should choose a build tool and create build scripts early on in the development process.

Note

*It's recommended that you test build scripts by first writing the "Hello World" program.*

It's important to keep compile times for build scripts as short as possible. Longer compile times slow down development, especially if developers must build frequently or are unable to leave builds to compile in the background while they attend to other tasks.  
  
In addition, if build times are too long, programmers might be tempted to perform them less frequently. This can reduce the benefits of build automation.

To decrease compile times, you can

* use incremental compilation
* use cleaner code
* use faster compilers, and
* upgrade system memory

Build automation isn't suitable in all situations. For example, it's unlikely to be suitable if the code being developed is quite complex.

You not only need to consider what coding can be automated, you also need to consider how development of the code will be shared across the project team.

If you use build automation, it's likely that you'll also use an automated deployment approach – and many of the same tools that are used for building code may be used to perform deployment activities.

If no functioning code has yet been written for a project, you can create an automatic deployment process simply by using a test script like the "Hello world" program – ensuring that the process satisfies any requirements specific to the project.  
  
You should then test the automated process you develop on a target machine that mimics the production environment, to identify and resolve any compilation or functional errors.

If code has already been written for a project, you should perform a test deployment and record the results.

Once you have developed an automated deployment process, you should enable non-programming team members to run it. This enables tasks to be performed without the assistance of a programmer, which speeds up development.

Question

Which guidelines should you follow when implementing build automation and automated deployment?

**Options:**

1. Keep compile times as long as possible when using build automation
2. Enable non-programming team members to access the automated deployment process
3. Use build automation only when developing complex code
4. Create a target machine that mimics the production environment when testing an automated deployment

Answer

***Option 1:****Incorrect. It's essential that you keep compile times as short as possible when using the build automation process. Long compile times can slow down development.*

***Option 2:****Correct. If you allow non-programmers to use the automated deployment process, development will run more smoothly.*

***Option 3:****Incorrect. Build automation isn't suitable for the development of more complex code. It should be used only in certain situations.*

***Option 4:****Correct. To effectively test your deployment, you need to test it on a machine that is as similar to the production environment as possible.*

**Correct answer(s):**

2. Enable non-programming team members to access the automated deployment process  
4. Create a target machine that mimics the production environment when testing an automated deployment

Continuous integration can be either

**serialized or**

With serialized continuous integration, programmers integrate their code with the original code base in a sequential order. Only one programmer is allowed to make changes to the code at any given time. To manage this process, you can use a specified workstation or a token that indicates who is allowed to integrate code at a specific time.

**simultaneous**

With simultaneous continuous integration, the entire code base is automatically checked and compiled on a regular basis. This should be done at least once every day to ensure that the system runs effectively. Programmers working using this method should use their own workstations to perform builds and test them before they integrate their code with the code base. This process is well suited to larger teams, as well as situations in which programmers do not share a common working location.

To support continuous integration, developers should aim to check in their code every few hours. It's important to make relatively small modifications at a time to the original code base so that changes can be tracked and don't introduce errors. So each developer should write a small piece of code, build it, and ensure that it passes all tests before continuing to write more code.

To ensure that programmers can check in code regularly, you need to break programming work into small, manageable tasks.

After using continuous integration for an extended period, you might find that the process requires updating. You may need to implement a more advanced process if you want to use

* a build machine that mimics the production environment
* a configuration that enables the build machine to start automatically after each code check-in, or
* an open-source application

Agile methodologies emphasize simplicity of design because this helps ensure faster, more reliable development and fewer errors.  
  
One aspect of keeping design simple is ensuring that software will do exactly what's required and no more. So agile programmers are encouraged to develop software to meet current needs – and not to design functionality to meet possible or anticipated needs, which may change before software is released.

To ensure simplicity, all new code that developers write should

* compile effectively and pass all tests
* produce no duplicated logic in your system
* achieve all required functionalities, and
* include as few classes and methods as possible

Additional guidelines for writing simple code are

* always to determine the simplest method of achieving what's required
* when possible, to avoid writing code that won't definitely be needed, and
* when possible, to write code only once

To help ensure simple design, you should encourage programmers to collaborate. Shared knowledge and experience can simplify the development process – as well as the resulting code – and save time in the long run.

In addition, when new team members join a project, it's important that they understand the required process. They also shouldn't misinterpret the concept of simplicity to involve easier – or less disciplined – coding.

Also to simplify the design process, programmers should reuse code when possible.  
  
If code creating a function that's required has been written before, the team should reuse the existing code. Similarly, if a simplified process has been developed, the team should follow it instead of wasting time attempting to create a new one from scratch.

Refactoring is the last practice that you need to consider when implementing an agile approach. It involves regularly making changes to code to improve its design. It typically relies on automated testing, build automation, and continuous integration.

There are three main activities that you should perform when using refactoring. You should

* simplify more complicated sections of code
* create reusable code for similar operations, and
* minimize duplicate code

Note

*For each of these tasks, you need to perform automated tests in the section of the system that you'll be refactoring. For older code, this might require that you write tests before you begin refactoring.*

It's important to understand that refactoring isn't the same as just reworking code. By improving code design, refactoring results in better maintenance of code, which is a useful goal and outcome.

When implementing refactoring, a team should aim to make ongoing small changes rather than larger once-off changes.

The process of refactoring also provides you with a good platform on which to develop a design pattern. However, it's important to remain aware of how changes to any part of the design process might affect other programmers working on the project. The agile approach is a team effort and requires collaboration and cooperation to succeed.

Note

*A design pattern is a general solution to a problem to a commonly occurring software design problem that you can reuse.*

Question

Match the agile practices with the factors that you should consider when implementing them.

**Options:**

1. Build automation
2. Automated deployment
3. Continuous integration
4. Simple design
5. Refactoring

**Targets:**

1. Ensure that compile times are as short as possible
2. Identify compilation and functional errors before production
3. Check in your code every few hours so that regular, small updates are made
4. Write code for today instead of planning for tomorrow
5. Implement small changes instead of large modifications

Answer

*When using build automation, you need to make sure that compile times are short to prevent slowing down the development process.*

*To establish an effective automated deployment process, you need to test the process before production to ensure that no errors or compilation issues occur. When doing this, you should use a test environment that mimics the production environment.*

*When using continuous integration, you should write a small piece of code, and then run the build to ensure that it passes all tests. You should then check in the piece of code before continuing to write more code.*

*The simple design process encourages you to focus on current needs so that you do not plan too far ahead with the possibility of wasting time on redundant coding.*

*If you use refactoring, you should make small changes rather than large changes so that the process is ongoing and does not stall the development process.*

**Correct answer(s):**

Target 1 = Option A

Target 2 = Option B

Target 3 = Option C

Target 4 = Option D

Target 5 = Option E

**3. Summary**

Key practices in an agile approach include build automation, automated deployment, continuous integration, simple design, and refactoring.  
  
For build automation to succeed, you need to keep compile times as short as possible. You should also ensure you use build automation only when it's really suitable. Automated deployments should be tested in an environment similar to the production environment, and non-programming team members should be allowed to run automated tasks to save programmers time. Continuous integration of code with a code base may be either serialized or simultaneous. Agile teams should focus on simplicity, designing software to meet current needs rather than possible future needs. They should also regularly refactor code to make design improvements.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_agsd_a04_it_enus/output/html/course_transcript.html#top)

Applying TDD and BDD in an Agile Project

Learning Objectives

*After completing this topic, you should be able to*

* *identify potential issues when implementing TDD in an agile project*
* *identify the features of BDD*

**1. TDD components**

Test Driven Design, or TDD, is a design technique that emphasizes thorough unit testing of source code, with unit tests written before the code itself is written. TDD should be used in conjunction with other agile testing techniques, such as agile acceptance testing and investigative testing.

As with traditional testing, a successful TDD test discovers mistakes in code. So a failed test is an opportunity for progress – it shows you where a mistake is so that you can fix the mistake and move on. You can also then re-run the same test on the same code to verify that the mistake really has been fixed.

Your aim in testing shouldn't be to test for a perfect system, but to test that the system does what's required. You should always be sure what you're testing a system for and which level of testing is necessary to achieve your goals. For example, more comprehensive tests are required for systems with higher risk profiles or higher criticality. TDD produces 100% coverage testing, which isn't necessarily the case with traditional testing.

Using TDD should involve writing and testing software in small increments. This is generally a more productive approach than attempting to code in large steps.  
  
If you create new functional code, add it, and run a test on it, it's likely that the test results will show some mistakes that need to be fixed. It's much easier to find and fix these mistakes in two lines of code than in two thousand lines of code.  
  
Ideally, you should aim to add and test less than ten lines of functional code at a time.

Benefits of TDD are that

* it ensures that all code is thoroughly tested
* it helps ensure that developers write code to be testable, because they create tests before developing the code
* it enables developers to change and update code without hesitation, knowing that thorough testing will prevent the introduction of errors to the code base, and
* it facilitates the development of clean, simple code because it encourages developers to write every object with the view that it has to be accessed and tested by another object

The components of TDD are

Graphic

*The diagram shows a test harness that contains a test fixture. The test fixture, in turn, contains a test suite with 2 test cases, a customer test and an order test, for a test harness.*

**test cases**

Test cases are the basic starting point for TDD. A test case specifies the sequence of actions necessary to achieve the goal state for a task.   
  
It is the code that you initially write in order to exercise an interface and drive implementation. Test cases include the basic assertions that say what kind of behavior should be the result of implementation. You use test cases to write the code most important for TDD, and then to drive the interface of the implementation.

**test fixtures**

Test fixtures are test objects shared by a number of test cases that specify the pre- and post-conditions for a test. Usually test fixtures are used to prepare the data against which tests will be run, and later to dispose of this data. You can use test fixtures to create test objects, setting up dependencies or creating mock-ups.

**test suites, and**

Test suites are groups of test cases designed to be executed at the same time. Test cases may be grouped in this way based, for example, on functional groups, application tiers, or common fixture needs.  
  
Usually test suites are hierarchical, so one test suite might be made up of several more test suites, and so on.

**test harnesses**

Test harnesses are TDD's highest level component. Harnesses provide a foundation for all other testing components, enabling the components to form an executable set.  
  
At its simplest, a harness provides an environment in which to run tests and capture their output. It may also provide features that simplify the testing process – for example, providing supporting services for fixture building or generating formatted reports of test results. The XUnit frameworks are the most commonly used harnesses for TDD, and include JUnit, NUnit, and CPPUnit.

Question

Which of the TDD components can you use to create test objects?

**Options:**

1. Test cases
2. Test fixtures
3. Test suites
4. Test harnesses

Answer

***Option 1:****Incorrect. Test cases are the basic starting point for TDD. They specify the sequence of actions necessary to achieve the goal state for a task.*

***Option 2:****Correct. You can use test fixtures to create test objects, setting up dependencies or creating mock-ups.*

***Option 3:****Incorrect. Test suites are groups of test cases meant to be executed at the same time. They don't enable you to create test objects.*

***Option 4:****Incorrect. Test harnesses provide environments in which to run tests and capture their output.*

**Correct answer(s):**

2. Test fixtures

**2. Implementing TDD in an agile project**

Although TDD has definite advantages, it also has some disadvantages. It isn't always easy to implement. For developers, doing this requires a change in mindset. If you're used to using business code, it might feel awkward to put so much time and thought into the initial implementation of design code. Before you have seen the benefits inherent in TDD for yourself, the process can seem daunting, so it may be difficult to get started.

It can be a challenge to use TDD on large volumes of existing code, because the tests have to be carried out on such small sections of code at a time for the best results.

Also, writing test cases and conducting thorough testing can be time consuming.

However, there are further advantages involved in using TDD that outweigh the disadvantages. It's possible that well-written unit tests, in creating a working specification of your functional code, can also be used as an important part of your technical documentation.

By the same token, acceptance tests can become an important part of your requirements documentation. This is logical because acceptance tests by nature provide a definition of what stakeholders' requirements are. You can then use your regression suite as detailed executable specifications.

After testing, you might still need user, system overview, operations, and support documentation. In addition, it's possible that you might need summary documentation as an overview of a system's business process.

Nevertheless, tests themselves can help a team stay as agile as it can with regards to documentation.

Misconceptions surrounding TDD include the ideas that

**you can always create a 100% regression test suite**

A 100% regression test suite is a good goal but isn't always attainable. This is because you often download or buy components without a test suite, and sometimes even without source code. A user interface could also be difficult to test, and some developers might not have the skills needed. Finally, database regression testing is a new idea and not yet optimally supported by tools.

**unit tests form 100% of the design specification**

Agile software development does require modeling and documenting. These occur as part of the process when developers think about the production code before writing it. Unit tests may make up a large part of the design specification, but so do models and documentation.

**only unit tests are necessary**

Employing unit tests as the sole type of testing is an option only for extremely simple systems.

**TDD is sufficient for testing, and**

Although TDD can be sufficient as confirmatory testing, it's necessary to employ further investigative tests at both the unit and customer test levels.

**TDD doesn't scale**

It can be difficult to adjust TDD for a larger project or team. The scalability problems you might have could include a test suite that takes too long to run, not all developers may know how to test the system, and TDD can run into problems if some testers are not taking a TDD approach. These problems can be overcome through effective management, coaching, and training of team members.

While TDD is useful if you need specification and validation, it isn't as helpful in providing an overview of a system's design, of how a system will be used, or of its user interface design. In these cases, agile model-driven development, or AMDD, is a good option.

TDD and AMDD differ fundamentally in several ways:

* TDD creates a shorter programming feedback loop, and AMDD creates a shorter modeling feedback loop
* TDD focuses on detailed tests whereas AMDD looks at the bigger picture
* TDD encourages high-quality code development, whereas AMDD encourages good communication among developers and with stakeholders, and
* TDD proves definitely that a system works, whereas AMDD helps a team gain a common understanding

Some further differences between TDD and AMDD are:

* TDD is suited to programmers, whereas AMDD is suited to business analysts, stakeholders, and data professionals
* TDD results in finely grained concrete feedback on the order of minutes, whereas AMDD results in verbal feedback on the order of minutes
* TDD makes sure that a system design is clean, whereas AMDD's focus is on addressing design and architectural issues before coding starts, and
* TDD is non-visually oriented, whereas AMDD is visually oriented

Some similarities shared by TDD and AMDD are that both are new and possibly intimidating to traditional developers, and both support evolutionary – or incremental – development.

Whether you decide to use TDD or AMDD depends on you and your team members' preferences. You will have to consider whether you and your team prefer a visual or text-driven approach, and also which technique seems best suited to the system you're working on.

You can also choose to combine the two approaches. First use AMDD to design models with project shareholders. This can help you to find out exactly what stakeholders require, and to represent these requirements through architectural and design models. Then use TDD when you start building the system, to make sure that you create clean, working code.

Combining AMDD and TDD in this way can help ensure that your team develops a high-quality system that meets stakeholders' requirements perfectly.

To implement TDD, you follow these steps:

Graphic

*The TDD workflow diagram starts with the add a test stage. You ten run the test. If it passes the test, you return to the add a test stage. If the test fails, you then make a little change to the test and run the adjusted test. If the test fails, you then return to the make a little change stage of the process. If it passes the test, you return to the add a test stage at the beginning of the process to continue the TDD process. Once you have tested all the code, and passed all the tests, development stops.*

* write a test and ensure that it fails
* write just enough code so that the test and all other tests pass, applying a simple design that doesn't plan for any future coding, and
* repeat and refactor as necessary, until the feature has been completed

Some tools you can use when implementing an agile project are CPPUnit, CSUnit, CUnit, DBFit, DBUnit, HTMLUnit, HTTPUnit, JMock, and JUnit.

Question

What are some potential issues involved in using TDD in agile project implementation?

**Options:**

1. Writing test cases is time-consuming
2. Tests have to be carried out on very small sections of code at a time
3. Test cases can't be used as technical documentation
4. Code written for test cases isn't testable

Answer

***Option 1:****Correct. The TDD approach can be time-consuming.*

***Option 2:****Correct. TDD requires designers to test small sections of code at a time. This can be daunting if a large body of code has already been written and needs to be tested.*

***Option 3:****Incorrect. It's possible to use test cases as part of your technical documentation.*

***Option 4:****Incorrect. Developers using TDD ensure that the code they write is testable. They create test cases before writing this code.*

**Correct answer(s):**

1. Writing test cases is time-consuming  
2. Tests have to be carried out on very small sections of code at a time

Question

Which steps should you complete when implementing TDD?

**Options:**

1. Apply a simple design that doesn't call for any future coding
2. Write a test and ensure that it doesn't fail
3. Repeat and refactor as necessary
4. Continue refining and adding to test code on an ongoing basis

Answer

***Option 1:****Correct. A simple design ensures that you write just enough code so that the test and all other tests pass.*

***Option 2:****Incorrect. In order to pick up any mistakes and fix them, the test must fail.*

***Option 3:****Correct. You repeat and refactor the test until there are no more mistakes in the code.*

***Option 4:****Incorrect. You need to write only enough code for the test and all other tests to pass.*

**Correct answer(s):**

1. Apply a simple design that doesn't call for any future coding  
3. Repeat and refactor as necessary

**3. BDD features**

Behavior Driven Development, or BDD, is an approach that incorporates aspects of TDD and domain-driven design.

BDD's aim is to improve a development team's ability to deliver prioritized, verifiable business value, through the use of a shared language known as Ubiquitous Language. Its purpose is to bridge the gap between business and technology.

The very specific vocabulary of Ubiquitous Language prevents miscommunication. It helps ensure that everyone, including business representatives, developers, testers, analysts, and managers, have a shared understanding of system specifications, design, implementation, and testing.

BDD is designed to accompany TDD, and doesn't alter the mechanics – or practices – that TDD involves. TDD is always at the core of BDD.

BDD provides the structure for a story that comes about as the result of a discussion between several people, and that has the following template – "As a Role, I request a Feature, to gain a Benefit."

First a business analyst talks to a stakeholder to determine the stakeholder's requirements. The analyst then reframes the requirements as a story.

Next the analyst speaks to a tester, who creates a scope for the story by deciding which scenarios are most important.

Finally a technical representative estimates how long it will take to develop the system described by the story. This person may also propose alternative ideas.

You need to use probing questions when you implement BDD analysis. For example, when a stakeholder states requirements, you ask why the stakeholder has those requirements. Although a requirement may seem self-evident, it's important to uncover the motivations behind it. This helps ensure that the requirement isn't misinterpreted.

BDD is often an iterative process. For example, a stakeholder may identify requirements without knowing the likely cost of meeting them. Technical and testing experts may then discuss the costs and benefits of various alternatives with the stakeholder – who then returns to and modifies the initial requirements.

If a development team struggles to come up with a ballpark estimate of the work required by a project, the team has to carry out investigative work – also called a spike.

BDD is different from use cases in that you can't use cases as if they represent all your thought. Instead, you have to put the use cases aside, and redefine stories using business outcomes as a base. You can then use the knowledge gained from the use cases to give you an understanding to create an outline of the work needed.

Note

*A use case describes what a system does in response to a request by a user. In order words, it describes who can do what with the system.*

Implementing BDD encourages you to think of the behavior of the component that you're developing, and of the roles and responsibilities of the other objects it interacts with.

Question

Which statements about BDD are correct?

**Options:**

1. BDD's aim is to improve a team's ability to deliver business value
2. BDD is the same, in practice, as use cases
3. BDD is often an iterative process
4. BDD combines aspects of TDD and domain-driven design

Answer

***Option 1:****Incorrect. BDD's aim is to improve a development team's ability to deliver business value, through the use of a shared language designed to bridge the gap between business and technology.*

***Option 2:****Incorrect. BDD is different from use cases in that you can't use cases as if they represent all your thought.*

***Option 3:****Correct. BDD is an iterative process. The initial requirements for the project may be modified as the project progresses or through further discussion with the stakeholders.*

***Option 4:****Correct. BDD combines aspects of TDD and domain-driven design, effectively bridging the gap between business and technology through a shared language.*

**Correct answer(s):**

3. BDD is often an iterative process  
4. BDD combines aspects of TDD and domain-driven design

**4. Summary**

TDD involves regular unit testing of small sections of source code, with tests typically created before the code is written. The components that make up TDD are test cases, test suites, test fixtures, and test harnesses.  
  
AMDD is useful for providing an overview – or model – of a system's design, and can be used with TDD to help ensure that stakeholder requirements are fully met.  
  
BDD is often used in conjunction with TDD. Its aim is to improve a development team's ability to deliver business value, through the use of a shared language designed to bridge the gap between business and technology.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_agsd_a04_it_enus/output/html/course_transcript.html#top)

Implementing an Agile Testing Lifecycle

Learning Objective

*After completing this topic, you should be able to*

* *recognize the features of agile testing practices*

**1. Testing during an agile life cycle**

Effective testing is an important part of the agile software development process. It helps to ensure the quality of the software you're developing so you can meet your customer's requirements.

During agile development, teams need to adhere to several principles.

**Test early**

It's important to test as early as you can to help minimize the impact of potential defects. The impact of these defects increases over the course of a project.

**Test often**

You should test as often as you can and as effectively as possible to increase the chance of finding defects. Running frequent tests increases project costs in the short term, but reduces the total cost of ownership because it helps to improve quality.

**Test whenever it's called for**

You should test as much as the situation calls for. Software for a commercial bank, for instance, requires far more testing than membership administration software for a small local company.

**Use pair testing**

The use of pair testing helps to reduce risk. Like pair programming and modeling alongside others, it helps team members spot defects early and test more effectively.

Agile software development projects are broken into several testing phases:

**initiation**

The initiation phase is usually of relatively short duration and involves establishing a project's foundations.

**construction**

In the construction phase, a system is developed in an evolutionary – or iterative and incremental – manner.

**end game, and**

In the end game phase, the system is transitioned into production.

**production**

The production phase involves operating the system and providing support for its users.

During the initiation phase, you need to perform initial setup tasks.   
  
Some of the setup tasks you might need to perform include

* selecting people for an external testing team
* selecting and possibly installing testing tools, and
* scheduling scarce resources – such as usability testing labs – needed for testing purposes

For projects that have fixed deadlines, you should also use the initiation phase to determine when a project needs to enter the end game phase.

Most of the testing conducted during an agile software development project occurs during the construction phase.

In agile projects, you should begin testing early, and you should test often. This is prioritized above planning, which may be important but isn't considered the chief focus. This differs from traditional approaches like the waterfall method, which place far greater importance on planning and documentation during development.

During iterations in the construction phase, a development team must perform

**confirmatory testing and**

Confirmatory testing verifies that the system fulfills the stakeholders' original requirements.

**investigative testing**

Investigative testing looks for problems that the development team hasn't considered.

Confirmatory testing verifies whether the project fulfills the stakeholder's intent by testing project specifications.  
  
There are two types of confirmatory tests.

**Acceptance tests**

Acceptance tests are concerned with the project's requirements specification. Agile acceptance tests are a mixture of traditional functional testing and traditional acceptance testing. During agile acceptance tests, the development team collaborates with the project customer.

**Developer tests**

Developer tests are concerned with the project's design specification. They are a mixture of traditional unit testing and traditional class, component, and service integration testing. Development tests seek to verify both the application's code – by seeking out coding errors, and its development schema – by performing path testing.

Confirmatory tests are usually automated because of the increased need for regression testing throughout the life cycle of a project.

During the end game phase, life-critical systems like medical software often require full system and acceptance tests.

If you've performed effective tests during the construction phase, your final testing efforts are normally straightforward and won't be too time-consuming.

If you've planned to do vigorous testing during the end game phase, you may need to rethink your planning if your team won't have enough time to act on any defects it finds.

Question

Match the phases of the agile project life cycle with the activities they involve.

**Options:**

1. Initiation
2. Construction
3. End game
4. Production

**Targets:**

1. Project foundations are set
2. The system is developed
3. The system is transitioned into production
4. The system is operated

Answer

*During the initiation phase, you set the foundations by identifying the external testing team, installing testing tools, and scheduling needed resources.*

*During the construction phase, a system is developed and tested in an iterative and incremental manner.*

*During the end game phase, a system is transitioned into production. You might need to conduct full system tests or acceptance tests at this point.*

*During the production phase, the system is operated. Teams might need to provide support for users during this phase.*

**Correct answer(s):**

Target 1 = Option A

Target 2 = Option B

Target 3 = Option C

Target 4 = Option D

**2. Agile testing practices**

Agile teams need to conduct a variety of tests to ensure the success of their development efforts.

**Drill Down Home Page**

Five main types of testing are typically used in an agile development environment.

**Page 1 of 6: Automated testing**

Automated testing, or auto-testing, is testing that's performed without – or with very little – human interaction. Some tests may be only partially automated.

**Page 2 of 6: Automated testing**

Various test activities may be automated. These include

* unit testing
* data setup
* installation
* execution of any type of test, and
* validation of test results

**Page 3 of 6: Automated testing**

If development projects already have legacy code, its not always necessary for the team to stop and write tests. It's often more useful simply to cordon off the existing code, and write unit tests only for the new code that the team adds.

**Page 4 of 6: Automated testing**

Although automation is sometimes difficult to achieve initially, it's important and saves a great deal of time in the long run. If teams invest in automation early on in a project, changes later are easier to deal with.

**Page 5 of 6: Automated testing**

In projects with short iterations, automating tests may be the only way to generate the number of tests necessary to create a fully tested and shippable product within given time constraints.

**Page 6 of 6: Automated testing**

With automated testing, testers should still be included in a development team to ensure continuous communication with developers. Testers and developers may even use pair testing to ensure collaboration.  
  
The development team must also work closely with the customer to help define acceptance tests for each of the customer's requirements.

**Page 1 of 5: Unit testing**

Unit tests are designed to check individual components within a system to ensure that specific functionality is working appropriately.  
  
In agile projects, development is test-driven, with tests written before unit code is developed. This is referred to as Test Driven Development, or TDD. Using TDD, developers add functionality to an application only once tests have been written to test this functionality.

**Page 2 of 5: Unit testing**

Unit tests should be run on restorable data. This is so that if someone wipes the database during a unit test, the data can be restored.

**Page 3 of 5: Unit testing**

Several unit testing frameworks are available for development teams to use in running unit tests.  
  
For instance, the automated test suite for the Java language, JUnit, uses red and green bars to present test progress. The green bar progresses as the team runs the test suite, indicating a passed test. If the green bar progresses all the way to the end, it indicates a passed test. If a single test is failed, the whole progress bar goes red.  
  
JUnit also provides a list of all the tests passed and failed. Even a single failed test denotes a broken build.

**Page 4 of 5: Unit testing**

Failed tests, including those failed for reasons beyond a team's control, can negatively affect morale. During unit testing, it's important to take steps to prevent this from occurring.

**Page 5 of 5: Unit testing**

Some projects don't include any legacy code, so developers need to write unit tests for all the code they develop. The thought of writing tests while coding can seem daunting at times, especially if a team doesn't have any experience writing automated unit tests. It's important that the team spends time learning unit testing techniques from an experienced programmer.

**Page 1 of 6: Investigative testing**

Investigative testing is completed by independent test teams during the construction phase of an iterative project.  
  
Investigative tests help to determine the quality of the development team's work by testing working software at intervals throughout a project.

**Page 2 of 6: Investigative testing**

Unlike confirmatory testing, which determines whether software meets written specifications, investigative testing looks for any defects that the development team might have missed.  
  
It helps determine whether the system that the development team has created is good enough by exploring the development team's blind spots.

**Page 3 of 6: Investigative testing**

Successful investigative testing has two main benefits:

* it helps identify problems before they become too expensive to address, and
* it provides feedback for management about whether a team is providing working software of a high quality

**Page 4 of 6: Investigative testing**

Investigative testers use defect stories to describe the potential problems that they identify.   
  
Once a potential problem has been described, the development team treats it as a requirement, which is estimated, prioritized, and added to the requirements stack so that it can be addressed.

**Page 5 of 6: Investigative testing**

There is no correct way to conduct investigative testing. The way it's conducted depends on a project's goals and the context or purpose of the testing.  
  
For example, tests seeking to determine whether software is ready to be shipped will be approached differently to those with the goal of determining whether software interpolates with other systems.  
  
It's also important to remember that the context for testing might change over the course of a project.

**Page 6 of 6: Investigative testing**

Investigative testing helps to address issues that may arise during various other types of testing, including

* load or stress testing
* integration testing
* security testing
* scenario testing, and
* usability testing

**Page 1 of 5: Regression testing**

Regression testing helps ensure that any changes made to an application don't effect its existing functionality.  
  
A small change to a program, for instance, could affect another part of the program. Without regression testing, this might go unnoticed and result in the program failing once it goes into production.

**Page 2 of 5: Regression testing**

Regression testing is conducted throughout the life cycle of a project. It's very important during incremental development projects because it helps ensure that the applications you release provide all the functionality they were designed to include.

**Page 3 of 5: Regression testing**

To conduct regression testing, you need to

* run previous test cases and
* test your code

**Page 4 of 5: Regression testing**

You should run your previous test cases against the new version of your application.  
  
If you've changed part of the application's design, you might need to modify some of your previous test cases.

**Page 5 of 5: Regression testing**

If a feature you've added to the application affects only a component of the system, you could potentially save time and money by simply testing this single component.  
  
However, this involves some risk because the impact of your changes may be greater than you think.

**Page 1 of 6: Acceptance testing**

Acceptance tests – also known as functional tests – focus on testing features. They're often conducted during the construction phase of a project's life cycle. They provide the opportunity for development teams to show customers that the new features they've programmed work. In turn, customers have the opportunity to accept or reject these new features.

Note

*Acceptance tests sometimes also serve as a way of capturing and documenting requirements.*

**Page 2 of 6: Acceptance testing**

Acceptance tests typically involve testing documented features, user stories, and use cases.  
  
As much as possible, acceptance tests should be automated.

**Page 3 of 6: Acceptance testing**

Acceptance tests that are correctly implemented provide two main benefits:

* effort expended remains steady and
* defects are reduced

**Page 4 of 6: Acceptance testing**

When you're running automated acceptance tests, you can expect the level of effort necessary on testing activities to remain steady over time.  
  
Conversely, other forms of testing often require increased effort for every new test that is run.

**Page 5 of 6: Acceptance testing**

Acceptance tests help reduce defects because developers can't consider their work complete until their code has passed the associated tests.

**Page 6 of 6: Acceptance testing**

The most beneficial acceptance tests are specified and written by the project customer. They should cover everything the customer wants verified before signing off for implementation.  
  
If a customer can't write the necessary acceptance test, the test may be written by an analyst or domain expert assigned to the development team.

Question

Match the agile testing practices with the types of tests they involve using.

**Options:**

1. Automated testing
2. Unit testing
3. Investigative testing
4. Regression testing
5. Acceptance testing

**Targets:**

1. Tests that are done without any human interaction
2. Tests that focus on small bits of functionality
3. Tests that determine the quality of the development team's work
4. Tests that ensure changes don't effect an application's existing functionality
5. Tests that enable the customer to accept new features

Answer

*Automated tests are conducted without any human interaction. Some tests are only partially automated. Automated testing is important during projects with short iterations.*

*Unit testing looks at individual components within a system to ensure that small bits of functionality are working properly.*

*Investigative tests are conducted by independent test teams who aim to determine the quality of the development team's work.*

*Regression testing runs previous test cases against the new version of an application to ensure that the changes made don't effect its existing functionality.*

*Acceptance tests are automated tests that work alongside features, user stories, and use cases, to enable customers to accept new features that have been added to an application.*

**Correct answer(s):**

Target 1 = Option A

Target 2 = Option B

Target 3 = Option C

Target 4 = Option D

Target 5 = Option E

**3. Summary**

Types of testing used by agile development teams include automated, unit, investigative, regression, and acceptance testing.  
  
The agile testing life cycle has four phases, namely initiation, construction, end game, and production. Most tests – including investigative and confirmatory tests – occur during the construction phase. However, regression testing occurs throughout the life cycle of a project. During the end game phase, a team may need to perform full system and acceptance tests.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_agsd_a04_it_enus/output/html/course_transcript.html#top)

User Interface Testing in Agile Projects

Learning Objectives

*After completing this topic, you should be able to*

* *identify the elements of an agile GUI testing process*
* *identify how test cases and oracles can be developed for agile GUI tests*

**1. Conventional GUI testing**

Software testing is an important component of any software development process. It enables the development team to detect errors in programs so that these errors can be fixed.

Errors are detected when

* a program's output isn't consistent with its specifications, or
* tests determine that a program's specifications are incorrect

Once an error is detected, developers might need to change software or its specifications. In turn, these changes often result in the need to re-test.

In a conventional development process, tests are executed using

**test cases and**

A test case is a set of variables that testers use to decide whether an application is functioning correctly.

**test oracles**

A test oracle is a mechanism that testers and developers use to determine whether an application has passed a test.

To test an application, you need to run test cases on it and use a test oracle to compare the test's output against the output you were expecting.

GUIs require different testing techniques to other software because they use user events and inputs, and produce graphical outputs.   
  
The conventional process for testing GUIs has six steps.

**Determine what to test**

The first step is to determine what you need to test. You can use coverage criteria to act as a set of rules for determining what needs to be tested.

**Generate test input**

The second step is to generate the test input. Test input is built either from software specifications or based on the way the software is structured. In GUI testing, test inputs include events like mouse clicks, menu selections, and object manipulation actions.

**Generate expected output**

The third step is to generate your expected output using test oracles. In GUI testing, expected outputs often include screen snapshots and the position and title of windows.

**Execute test cases and verify output**

The fourth step is to execute test cases and verify your output. To execute a test case, you need to perform all the input events you specified in step two. Once you've executed the test, you can compare the output with the expected output you generated using test oracles in step three.

**Determine whether the GUI has been adequately tested**

The fifth step is to decide whether the GUI has been tested sufficiently. Once you've run all your test cases, you can determine how well you've tested an application by analyzing the software to check which parts of it you've tested. Once you know what you've tested, you can see whether you've missed any important areas that should have been tested.

**Perform regression testing**

The last step occurs once you've identified and addressed any issues you detected during the previous steps. You should perform regression testing to make sure the modifications you've made don't adversely effect any other parts of the application.

Question

Sequence the steps for traditional GUI testing.

**Options:**

1. Determine what to test
2. Generate test input
3. Generate expected output
4. Execute test cases and verify output
5. Determine if the GUI is adequately tested
6. Perform regression testing

Answer

**Correct answer(s):**

**Determine what to test is ranked**

The first step is to use coverage criteria as a set of rules to help you determine what you need to test.

**Generate test input is ranked**

The second step is to use the software's structure or its specifications to generate test input.

**Generate expected output is ranked**

The third step is to use test oracles to generate expected test output.

**Execute test cases and verify output is ranked**

The fourth step is to execute test cases by performing the input events you specified in step two, and verifying your output by comparing it with the expected output you generated in step three.

**Determine if the GUI is adequately tested is ranked**

The fifth step is to analyze the software to determine whether you've missed any important areas and to determine whether the software has been adequately tested.

**Perform regression testing is ranked**

The last step is to perform regression testing to determine whether your modifications affect any other areas of the GUI. You should perform regression testing only once you've executed all test cases and addressed any issues you detect.

**2. The agile GUI testing process**

In agile projects, a different set of GUI testing techniques is needed so that every increment of the GUI can be tested efficiently.

To conduct continuous integration testing of GUI-based applications, you need to use both modern model-based GUI testing techniques and agile techniques.  
  
You can achieve this using a concentric testing loop method.

Graphic

*Three concentric loops are shown testing different aspects of the GUI. The smallest loop, crash testing, tests code check-ins. The middle loop, smoke testing, tests daily builds, and the biggest loop, comprehensive GUI testing, tests major versions. Code check-ins are contained within each daily build, and several daily builds make up a major release.*

Using this method, *crash tests* are run whenever GUI code is checked in. Crash tests are automated, inexpensive, and quick. They report software crashes within minutes of a code check-in.

*Smoke tests* are run on every GUI build's daily progress, and are designed to complete within eight to ten hours. Smoke tests run reference tests on the day's newly integrated version of the GUI, comparing the previous day's output to the new build's output. Any differences are reported to developers.

*Comprehensive GUI testing* is performed only once a major version of the GUI is ready. Comprehensive GUI tests are in-depth and expensive integration tests that detect and report any major software and GUI issues.

**Drill Down Home Page**

So the concentric testing loop method involves three types of testing.

**Page 1 of 5: Crash testing**

Crash testing involves using test cases to test major parts of the GUI quickly and automatically, without any human interaction.

**Page 2 of 5: Crash testing**

Crash testing includes test cases that

* are generated and executed quickly, and aren't saved or maintained as a suite
* cover the GUI's entire functionality
* restart the crash testing process whenever a GUI change is checked in, and
* detect major problems in short intervals

**Page 3 of 5: Crash testing**

Crash test oracles simply determine whether the tested software has crashed. A crash occurs when software terminates unexpectedly while test cases are being executed.

**Page 4 of 5: Crash testing**

Feedback from crash testing is given to the developer who checked in the last GUI changes. It's this developer's responsibility to debug the code and resubmit the changes so that issues don't affect any other developer's progress.

**Page 5 of 5: Crash testing**

The crash testing process is considered efficient because it's fully automated. It's also useful because it helps to detect any major GUI integration issues.

**Page 1 of 5: Smoke testing**

Smoke testing is more complex than crash testing, and takes more time to execute. It requires input from the tester.  
  
Whereas crash testing simply looks for crashes, smoke testing seeks to determine whether issues detected are the result of the latest modifications to a GUI.

**Page 2 of 5: Smoke testing**

Smoke testing test cases

* are generated and executed quickly
* remain usable even if the GUI is modified, and
* provide adequate coverage of the GUI's functionality and raise an alarm if they detect an issue

**Page 3 of 5: Smoke testing**

During smoke testing, test suites can be divided into parts that can be run in parallel on different machines.  
  
Test oracles compare the current version's output with the previous version, and report any differences.

**Page 4 of 5: Smoke testing**

Smoke testing is an effective way of detecting a wide variety of different faults. However, testers need to examine test results and to eliminate any false positives that might occur because of modifications to the GUI.

**Page 5 of 5: Smoke testing**

Crash bugs often result in a large number of failed and unexecuted test cases, which can cause delays. To eliminate this issue and ensure that crash bugs aren't transmitted to the smoke testing loop, you can combine smoke and crash testing.

**Page 1 of 3: Comprehensive GUI testing**

Comprehensive GUI testing is expensive and used only for testing major versions. You can conserve your resources during comprehensive GUI tests using a model-based approach.

**Page 2 of 3: Comprehensive GUI testing**

Test cases for comprehensive GUI testing

* cover the entire functionality of the GUI
* are generated using a model of the GUI that's updated as the GUI evolves, and
* detect any deviations from the specifications in the GUI model during test execution

**Page 3 of 3: Comprehensive GUI testing**

Because test cases are generated from a model of the GUI, so too is the test oracle. The comprehensive GUI test oracle encodes expected behavior.

Question

Identify the types of testing included in a concentric testing loop method.

**Options:**

1. Crash testing
2. Smoke testing
3. Comprehensive GUI testing
4. Output testing
5. Oracle testing

Answer

***Option 1:****Correct. Crash tests are fully automated, quickly executed tests that detect crashes within the GUI.*

***Option 2:****Correct. Smoke tests are complex tests that determine whether problems are the result of previous modifications to the GUI.*

***Option 3:****Correct. Comprehensive GUI tests use a model of the GUI to detect major software and GUI issues.*

***Option 4:****Incorrect. The concentric testing loop method involves crash, smoke, and comprehensive GUI testing.*

***Option 5:****Incorrect. The concentric testing loop method involves crash, smoke, and comprehensive GUI testing. Test oracles are used to specify whether an application has passed a test.*

**Correct answer(s):**

1. Crash testing  
2. Smoke testing  
3. Comprehensive GUI testing

**3. Developing test cases and oracles**

In the agile GUI testing process, test cases and test oracles are often automatically generated using the GUI event-flow model.  
  
The GUI event-flow model consists of *objects*and *properties*. Objects are widgets and containers, and properties refer to the state of the GUI's objects at a specified time.

In GUI testing, *events* are functions you execute to change the GUI's state.

The GUI event-flow model has two modes:

**preconditions and**

Preconditions are the state in which events can be executed.

**effects**

Effects are the changes that occur to properties after events have been executed.

The agile GUI testing method you use will determine how you generate

**test cases and**

During the comprehensive GUI testing loop, a task is by an initial and a goal state. Test cases are the sequence of actions necessary to achieve the goal state.   
  
Crash and smoke testing test cases are generated automatically using tools like the GUI Ripper. Crash and smoke test cases need to include all the events in the GUI model and all the possible event interaction pairs.   
  
The GUI Ripper constructs event graphs of all the possible event interactions in the GUI and creates integration trees that represent all the possible interactions between them. The Ripper then traverses each graph to generate test cases.

**test oracles**

Test oracles test whether software has functioned correctly for a given test case. To create a test oracle in a comprehensive GUI testing loop, the output of a test case is compared with the expected output. The GUI model helps you to determine the expected state, and you can get the actual result state while you're executing a test case.  
  
The GUI model isn't used to generate crash and smoke test cases. During a crash testing loop, the test oracle needs to determine only whether the system has crashed. During the smoke testing loop, you contrast the current and previous outputs.

Question

Identify how test cases and test oracles are generated in the agile GUI testing process.

**Options:**

1. Using the GUI event-flow model
2. Using regression testing
3. Using crash testing

Answer

***Option 1:****Correct. The GUI event-flow model generates test cases and oracles. Test cases are generated by determining the sequence of actions needed to achieve a goal state. A test oracle generates the expected output for a specified test case.*

***Option 2:****Incorrect. The GUI event-flow model generates test cases and oracles for the agile GUI testing process. Regression testing is performed after traditional GUI testing to ensure that modifications haven't effected the system.*

***Option 3:****Incorrect. The GUI event-flow model generates test cases and oracles for the agile GUI testing process. You perform crash testing as part of the agile concentric testing loop to test for software crashes.*

**Correct answer(s):**

1. Using the GUI event-flow model

**4. Summary**

To test GUIs, you need to determine what to test, generate test input and expected output, execute test cases and verify the output, determine whether the GUI has been adequately tested, and perform regression testing.  
  
In the agile GUI testing process, you can use the agile concentric loop method of testing. The method executes crash testing on every code check-in during a project, smoke testing on daily builds, and comprehensive GUI testing when a major version is ready.  
  
Test cases are generated using the GUI event-flow model. The event-flow model also generates test oracles for comprehensive GUI testing.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_agsd_a04_it_enus/output/html/course_transcript.html#top)

Working with Data in an Agile Project

Learning Objectives

*After completing this topic, you should be able to*

* *identify approaches to test data management*
* *recognize how to apply TDD to agile database development*
* *identify how BDD can be applied to agile database development*

**1. Testing data management**

Many systems use databases to collect, store, retrieve, and reference data.

Databases aren't ideal for agile programming, however, because they limit the agility of development teams. This is a result of their inability to deal with changes to data quickly and easily.  
  
Instead, agile teams require data management systems that can deal efficiently with multiple, changing datasets.

One option that increases agility is to start a project without a database.  
  
In this case, you might choose to use flat files, XML, or anything else that simplifies data management.  
  
This approach enables you to alter structures quickly, and work towards an appropriate data structure as development proceeds.

All data, including test data, needs to be managed. Two simple approaches to managing test data are

* authorizing tests to represent and delete their own data, and
* adding reloadable datasets to the database before tests are run

Alternatively, you can manage test data with spreadsheet-based tools.  
  
These tools can be simple, focusing only on data entering and exiting the database.  
  
They can also be specialized, containing knowledge of the system and allowing test cases to be created directly from a spreadsheet.

Spreadsheet-based solutions have both

Graphic

*Test data management contains three characteristics: It authorizes tests to represent and delete their own data, it add reloadable datasets to the database before tests are run, and it uses spreadsheet-based solutions. The two advantages of spreadsheet based solutions are simple data handling and that it provides a method for managing test data. The two disadvantages are primary key collisions and that the structure can get out of sync.*

**advantages and**

Spreadsheet-based solutions are simple to use and allow for quick data entry and navigation. They can be a useful method for programmers, analysts, and users to manage test data.

**disadvantages**

There are two main disadvantages of spreadsheet-based solutions. The first is that problems will arise if the structure in a spreadsheet isn't in sync with the system database. The second is that the spreadsheet and the database sometimes use the same set of primary keys, which results in primary key collision.  
  
Both these problems are a result of the data being managed apart from the database.

**Drill Down Home Page**

Two approaches have been developed to simplify the process of coding and maintaining test objects.

**Page 1 of 5: ObjectMother**

The ObjectMother approach is a programming pattern that addresses the issue of test data management by removing the need to create test data for individual cases.  
  
Instead it proposes a simple framework that constructs and facilitates the customization of objects.  
  
ObjectMother starts with the abstract factory pattern and adds the elements that are required for testing.

Note

*An ObjectMother framework is very simple to create at the onset of coding and is most effective if it includes iteration zero.*

**Page 2 of 5: ObjectMother**

An ObjectMother framework is able to

* produce fully formed data structures
* deliver requested data structures at any point in their life cycle
* customize delivered data structures as requested
* update data structures throughout the testing process, and
* terminate data structures and related objects at the end of the test process

**Page 3 of 5: ObjectMother**

The ObjectMother framework simplifies and standardizes the creation of test data. It also saves time by making it unnecessary to code individual test data or fix test cases.   
  
By using the ObjectMother pattern, you can reduce the activity required to obtain reliable test data to a handful of method calls. Ultimately, this is likely to encourage members of a development team to write more tests.

**Page 4 of 5: ObjectMother**

Writing an ObjectMother pattern involves locking small, simple objects together to create larger, more complex ones.  
  
You will use two main methods when creating an ObjectMother – creation methods and attachment methods.  
  
Creation methods return all data structures, whereas attachment methods assist in tailoring the created objects.

**Page 5 of 5: ObjectMother**

An ObjectMother creation method might look something like this.

Code

public static Invoice createNewInvoice() {  
Invoice invoice = new Invoice();  
invoice.setInvoiceNumber("InvTest001");  
Address address = createAddress();  
invoice.setBillToAddress(address);  
attachInvoiceLineAsCharge(invoice,  
new Money("4999.95","USD"));  
invoice.setStatus(InvoiceStatus.NEW);  
return invoice;  
}

**Page 1 of 4: Mock Objects**

The Mock Objects approach addresses the issue of test data management by removing the need to generate test data at all.  
  
Instead it proposes building "mock" test objects that mimic real objects. These objects are put through an application as real objects and then reviewed to see how they were treated.

**Page 2 of 4: Mock Objects**

Mock objects are used to

* isolate and test specific behavior within a system
* reach and test functions that are buried within an application
* test functions that produce unpredictable results
* tackle tests that would otherwise require large amounts of fabricated data
* test conditions that are hard to reproduce, and
* take the place of actual objects that still have to be developed

**Page 3 of 4: Mock Objects**

Say you're working on a system for an online bookstore. You create a mock object called "MockBook" that's coded to resemble an online book file.   
  
You then put MockBook through the application. MockBook will be dropped into the shopping cart and taken through different steps of the checkout process.  
  
In each test, the system will ask MockBook questions about its price, author, and publishing label.

**Page 4 of 4: Mock Objects**

As it goes through the system, MockBook records the interactions in which it's involved. It notes whether it has been asked certain things and give answers that will keep the test going.  
  
Finally you review MockBook's records to make sure the system performed the correct actions and didn't perform any unexpected ones.

Question

Identify the approaches to test data management.

**Options:**

1. Using mock objects
2. Using ObjectMother
3. Starting the project with a fully functioning database
4. Using TestDataBuilder

Answer

***Option 1:****Correct. One solution for test data management is to use mock objects. This approach attempts to remove the need to generate test data altogether.*

***Option 2:****Correct. One solution for test data management is to use the ObjectMother programming pattern. This approach removes the need to create test data for individual cases.*

***Option 3:****Incorrect. Databases present problems for testing, so it's best to start an agile project without a database.*

***Option 4:****Incorrect. Agile teams have replaced the TestDataBuilder programming pattern with the ObjectMother pattern.*

**Correct answer(s):**

1. Using mock objects  
2. Using ObjectMother

**2. The agile DBA**

Database administrators, also known as DBAs, determine the flexibility of the databases they run.

The position of DBA can be filled by different people at different stages of a project. Typically, the responsibilities and duties of the DBA are transferred among team members.

These responsibilities include bridging the gap between code and data, writing tools, and developing techniques.

An agile development project requires a DBA to

**perform traditional development-related duties**

The agile DBA needs to perform traditional development-related duties for the project team. These duties include creating and modifying new data sources, structures, queries, and  
procedures.

**provide extra assistance to the project team**

The agile DBA needs to provide extra assistance to the project team. The DBA will have to work with team members when completing tasks such as designing new data structures and testing datasets.

**understand the software development process**

An agile DBA needs to have a good understanding of the software development process in order to respond quickly to the changing needs of the team. However, the DBA doesn't need to know a lot about production database issues. Another production DBA can be brought in if necessary.

**serve as an emissary, and**

An agile DBA may need to serve as an emissary in a number of situations. This person will need to act as a representative between the development team and other parties within a large project, and between users of a shared data source, such as members of other development teams and report writers.

**act as a full-fledged member of the project team**

The agile DBA needs to act as a full-fledged member of the project team. This will encourage the DBA to improve the team's development environment. It will also encourage the DBA to recognize potential data-related issues before the team does.

An agile DBA doesn't need to be formally trained in database administration.  
  
For example, the DBA may be a team programmer, analyst, or tester who's asked to take on the additional task of managing the team's data sources.  
  
The DBA will, however, need time to train in the database the system is using.

Many programmers feel that database management is more complicated than code management. This is due to

**context complexity and**

An application records its code in one location, namely source control. This system records only one version of a code at any time. Source control makes use of a development timeline, which means that code can be found easily according to when it was completed. The same application can, however, have a dozen different sets of data that it needs to maintain. Context becomes an issue because each different set of data exists for a different purpose in a different context. Unlike the development timeline, context is a non-linear and noncontinuous dimension, and is far more difficult to manage.

**cultural differences**

Two types of people work on program development projects. Programmers may be referred to as "object people", whereas data professionals may be called "data people." Problems can arise because programmers and data professionals have different needs of a database. This can make data management a complicated task.

Question

What is the role of the DBA in an agile project?

**Options:**

1. To perform traditional development-related duties
2. To provide an additional level of assistance to the project team
3. To understand production database issues
4. To serve as an emissary
5. To act as a part-time consultant for the team

Answer

***Option 1:****Correct. An agile DBA will need to perform traditional development-related duties for the project team. These duties include creating and modifying new data sources, structures, queries, and procedures.*

***Option 2:****Correct. The DBA needs to provide an additional level of assistance to the project team. They will have to collaborate with team members on the design of new data structures and test data sets.*

***Option 3:****Incorrect. The DBA doesn't need to know a lot about production database issues because a real production DBA can be brought in if necessary.*

***Option 4:****Correct. The DBA needs to serve as an emissary or representative in situations such as meetings between the development team and other parties within a large project.*

***Option 5:****Incorrect. The DBA needs to act as a full-fledged member of the project team.*

**Correct answer(s):**

1. To perform traditional development-related duties  
2. To provide an additional level of assistance to the project team  
4. To serve as an emissary

**3. Applying Test Driven Development**

**Drill Down Home Page**

There are at least two common approaches to program development.

**Page 1 of 3: Test Driven Development**

Test Driven Development, or TDD, is an approach to program development that combines refactoring and test-first development.  
  
Refactoring involves making a simple change to a database schema while retaining its semantics, and test-first development involves writing the test code before you write the functional code to fulfill that test.

Some developers view TDD as a way to think through a design before writing functional code, while others view it as a programming technique that results in clean code.

**Page 2 of 3: Test Driven Development**

TDD enables you to take small steps when programming. Usually one step involves one test and a small piece of corresponding functional code.  
  
When the test code is complete, you start working on the functional code to ensure that the code passes the test you have just written.

**Page 3 of 3: Test Driven Development**

TDD is useful if you require detailed specification and validation. If you need to deal with broader considerations, such as how people will use a system, TDD isn't the best option.

**Page 1 of 2: Agile Model Driven Development**

Agile Model Driven Development, or AMDD, requires you to create models before you write any functional code.  
  
The models don't have to be perfect – they just need to be good enough to drive your development efforts before you begin to write any functional code.

**Page 2 of 2: Agile Model Driven Development**

AMDD is best suited to dealing with broad development concerns. It's a means of scaling agile software development beyond the small, focused approach that's advocated by TDD.

To date, TDD and AMDD have not been used extensively in database development.  
  
However, both the TDD and AMDD approaches are highly suitable for use in agile database development and should be used where appropriate.

Implementing TDD in database development involves three basic steps:

**database refactoring**

In database refactoring, developers make simple changes to a database without changing its semantics.

**database regression testing, and**

In database regression testing, developers regularly run a test suite that validates the database. If possible, each change to the database is validated.

**continuous database integration**

In continuous database integration, developers rebuild and retest the database schema whenever it changes.

A database refactoring is a simple change to a database schema that changes its design but retains its behavioral and informational semantics.

Database refactorings never add new features to a database.

So a key feature of refactoring is that it preserves both a database's

**informational semantics and its**

Informational semantics refers to the meaning of the information within a database. The meaning is taken from the point of view of the user. Preserving the informational semantics of a database means that any changes to the values of the data stored in a column won't affect the users of that information.

**behavioral semantics**

Behavioral semantics refers to the way a database acts as a result of the information within it. Preserving the behavioral semantics of a database means ensuring that after code is reworked, the database functions in the same way as it did before the change.

Say you notice that the FirstDate column of your database is being used for two distinct purposes. It stores the date of birth of any customer whose details are entered into the system, and it stores the date of hiring for any new employees.

Your realize that the system also needs to support customers who happen to be employees.

The first thing you do is change your database schema by replacing the FirstDate column with two columns titled "BirthDate" and "HireDate."

To maintain the database's informational semantics, you write a migration script that loops through the table, determines the type, and then copies the existing date into the appropriate column.

To maintain the database's behavioral semantics, you update all source code that accesses the FirstDate column to work with the two new columns.

There are five categories of database refactoring:

**data quality**

Data quality refactorings are a major category of database refactorings. These refactorings improve the quality of the data within a database. Examples include "Introduce Column Constraint" and "Replace Type Code with Booleans."

**structural**

Structural refactorings are a major category of database refactorings. These refactorings alter the database schema. Examples include "Rename Column" and "Separate Read-Only Data." A database refactoring is considered only a structural refactoring when it doesn't fall into the architectural, performance, or referential integrity subcategories.

**architectural**

Architectural refactorings make up a subcategory of structural refactoring. These refactorings involve one type of database item being changed to another type. Examples include "Encapsulate Calculation with a Method" and "Encapsulate Table with a View."

**performance, and**

Performance refactorings make up a subcategory of structural refactoring. These refactorings involve cases where alterations are made with the intention of improving database performance. Examples include "Introduce Calculated Data Column" and "Introduce Alternate Index."

**referential integrity**

Referential integrity refactorings make up a subcategory of structural refactoring. These types of database refactorings involve cases where alterations are made with the intention of ensuring referential integrity. Examples include "Introduce Cascading Delete" and "Introduce Trigger(s) for Calculated Column."

Two types of database changes are often confused as refactorings:

**a small change to a database to extend it, and**

A small change to a database schema to extend it isn't a refactoring. This change extends the design of the database, which is something that refactorings never do. An example of such a change would be the addition of a new column or table.

**a large number of small changes applied simultaneously**

A large number of small changes that are applied to a database schema simultaneously doesn't constitute a refactoring specifically because it isn't a single, small change. An example of such a transformation would be the renaming of several columns.

Question

Identify the steps involved in applying TDD to database development.

**Options:**

1. Undertaking database refactorings
2. Completing database regression testing
3. Continually integrating databases
4. Deleting databases
5. Creating models

Answer

***Option 1:****Correct. Database refactoring is involved in implementing TDD. In this step, developers make a simple change to a database to improve the design without changing its semantics.*

***Option 2:****Correct. Database regression testing is involved in implementing TDD. In this step, developers run a comprehensive test suite that validates the database regularly.*

***Option 3:****Correct. Continuous database integration is involved in implementing TDD. In this step, developers rebuild and retest the database schema whenever it changes.*

***Option 4:****Incorrect. Database deletion isn't involved in TDD. The steps involved are database refactoring, database regression testing, and continual database integration.*

***Option 5:****Incorrect. Model creation is involved in AMDD not TDD. The steps involved in TDD are database refactoring, database regression testing, and continual database integration.*

**Correct answer(s):**

1. Undertaking database refactorings  
2. Completing database regression testing  
3. Continually integrating databases

**4. Applying Behavior Driven Development**

Behavior Driven Development, or BDD, is an approach that is well equipped to deal with the friction between XP and Scrum processes. BDD is commonly thought of as an extension of TDD.

The focus in BDD is on the behavior or performance of systems or data objects. In other words, what systems *do*is of greater concern to programmers than the *data* they act on.

BDD includes four key principles:

* behavior is required
* a shared language should be used to discuss requirements, abstractions, specifications, documentation, and conversations
* acceptance criteria should be executable, and
* design constraints should be made into executable tests

All of these principles can be applied when implementing BDD in database development.

As with TDD, BDD uses any problems detected in the testing phase to work through any design problems in the database.

The applicability of BDD to database development is highlighted by the fact that database development relies on the expected behavior of database objects in the same way that BDD relies on the expected behavior of any system.

Some expected behavior is universal. For example, when you insert a row in a table, you expect that the same row can be retrieved later.

Some expected behavior isn't universal. For example, the expectation that a person table should have at least firstname or lastname columns is specific to a particular project.

Say that you make a column "Not Nullable." You can assume that the database server will throw an exception when a Null value is inserted in this column.

Making the column "Nullable" can alter the behavior of the database. This database change will mean that all the assumptions that the system made about the database not allowing Null values in the column are no longer true.

To avoid these problems, you can use the TDD core of BDD to test the database's behavior. The test will ensure that the database throws an exception when Null values are put in the column.

Question

The BDD technique can be applied to database development to develop and design databases in an iterative and incremental way.  
  
Identify the steps for implementing BDD in database development.

**Options:**

1. BDD is applied to database development through the use of a shared language
2. BDD is applied to database development via a mutual reliance on expected behavior
3. BDD is applied to database development only in cases where expected database behavior is universal
4. BDD is applied to database development via the process of database refactoring

Answer

***Option 1:****Correct. BDD is applied to database development through the use of a shared language. This language is used to describe database columns as well as to discuss requirements, abstractions, specifications, documentation, and conversations.*

***Option 2:****Correct. BDD is applied to database development via a mutual reliance on expected behavior. Some expected behaviors are universal while others aren't.*

***Option 3:****Incorrect. BDD is applied to database development in cases where expected database behavior is universal or non-universal.*

***Option 4:****Incorrect. Database refactoring is involved in the application of TDD to database development. BDD is applied to database development because of a mutual reliance on expected behavior.*

**Correct answer(s):**

1. BDD is applied to database development through the use of a shared language  
2. BDD is applied to database development via a mutual reliance on expected behavior

**5. Summary**

Test data management is sometimes undertaken with spreadsheet-based approaches. The ObjectMother and Mock Objects approaches simplify the process of coding and maintaining test objects.  
  
An agile DBA performs duties that include traditional development-related duties and providing extra assistance to the project team.  
  
Two approaches to program development are TDD and AMDD. Both approaches can be applied to database development. A refactoring is a simple change to a database schema that changes its design but retains its behavioral and informational semantics.  
  
BDD is an approach to programming in which the behavior or performance of systems or data objects is the focus.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_agsd_a04_it_enus/output/html/course_transcript.html#top)

Using TDD, BDD, and Agile Testing Strategies

Learning Objectives

*After completing this topic, you should be able to*

* *recognize BDD and TDD approaches to agile development*
* *recognize considerations when implementing an agile testing lifecycle*
* *recognize considerations when implementing UI testing in agile projects*
* *work with data in an agile project*

**1. Exercise overview**

In this exercise, you're required to recognize how to use TDD, BDD, and agile testing strategies.

This involves the following tasks:

* recognizing BDD and TDD approaches to agile development
* recognizing considerations when implementing an agile testing lifecycle
* implementing GUI testing in agile projects, and
* working with data in an agile project

**2. Recognizing BDD and TDD approaches to agile development**

TDD – Test Driven Development, and BDD – or Behavioral Driven Development, form two distinct but complementary design techniques which, when appropriately integrated, lead to the establishment of an effective agile programming development strategy.

Question

TDD, or Test Driven Development, is a design technique that emphasizes thorough unit testing of source code, with unit tests written before the code itself is written. What are the benefits of TDD?

**Options:**

1. It ensures that all code is thoroughly tested
2. It helps ensure that developers write code to be testable
3. It’s particularly effective on large volumes of existing code
4. It enables developers to change and update code without hesitation
5. It’s an easy technique to implement

Answer

***Option 1:****Correct. When followed diligently, TDD minimizes the risk of untested code making it into production.*

***Option 2:****Correct. As developers create tests before developing the code, it helps them write code that is testable.*

***Option 3:****Incorrect. In fact the opposite is true. Tests should be carried out on small sections of code at a time for the best results.*

***Option 4:****Correct. Following TDD, developers are more like to change and update code as they know that thorough testing will prevent the introduction of errors to the code base.*

***Option 5:****Incorrect. There can be some difficulty when the technique is first adopted as it requires developers to change their mindsets and to put more time and thought into the initial implementation of design code. Also, learning TDD may initially slow a team’s productivity.*

**Correct answer(s):**

1. It ensures that all code is thoroughly tested  
2. It helps ensure that developers write code to be testable  
4. It enables developers to change and update code without hesitation

Question

Which statement about TDD is true?

**Options:**

1. You can always create a 100% regression test suite
2. Unit tests form 100% of the design specification
3. TDD alone is sufficient for testing
4. TDD should be used in conjunction with other agile testing techniques

Answer

***Option 1:****Incorrect. This statement is false. While a 100% regression test suite is a good goal, it isn't always attainable. For example, you often download or buy components without a test suite, and sometimes even without source code.*

***Option 2:****Incorrect. This statement is false. While unit tests may make up a large part of the design specification, so do models and documentation. These occur as part of the process when developers think about the production code before writing it.*

***Option 3:****Incorrect. This statement is false. Although TDD can be sufficient as confirmatory testing, it's necessary to employ further investigative tests at both the unit and customer test levels.*

***Option 4:****Correct. This statement is true. TDD should be used in conjunction with techniques such as agile acceptance testing and investigative testing.*

**Correct answer(s):**

4. TDD should be used in conjunction with other agile testing techniques

Question

BDD, or Behavioral Driven Development, is an approach that incorporates aspects of TDD and domain-driven design. Which statements best describe the aim of BDD in an agile project?

**Options:**

1. To improve a development team's ability to deliver prioritized, verifiable business value, through the use of a shared language known as Ubiquitous Language
2. To facilitate the development of clean, simple code by encouraging developers to write every object in the knowledge that it has to be accessed and tested by another object
3. To facilitate scaling agile software development beyond a small, focused approach
4. To bridge the gap between business and technology
5. To reduce repetitive, manual coding tasks by replacing them with simple tasks that can be easily performed by all programmers

Answer

***Option 1:****Correct. BDD uses the very specific vocabulary of Ubiquitous Language to help deliver prioritized business value. It can help prevent miscommunication by ensuring that everyone has a shared understanding of system specifications, design, implementation, and testing.*

***Option 2:****Incorrect. This statement describes one of the aims of TDD. It also ensures that all code is fully tested and that developers write code to be testable – as they create tests before developing the code.*

***Option 3:****Incorrect, This statement describes AMDD (Agile Model Driven Development), which requires you to create models before you write any functional code. It contrasts with TDD, which tends to favor a more focused approach.*

***Option 4:****Correct. BDD aims to bridge the gap between business and technology through the use of Ubiquitous Language.*

***Option 5:****Incorrect. This describes build automation, the process of automating coding tasks, which is one of the features of agile programming. Ideally, these are push-of-a-button tasks, which are extremely quick to learn and perform, and that can be easily assessed on any programmer's workstation.*

**Correct answer(s):**

1. To improve a development team's ability to deliver prioritized, verifiable business value, through the use of a shared language known as Ubiquitous Language  
4. To bridge the gap between business and technology

**3. Recognizing considerations when implementing an agile testing lifecycle**

Each agile testing lifecycle demands specific and appropriate testing methods and techniques. Recognizing the attributes, applicability, and appropriate use of each type of testing in the lifecycle is essential to an effective agile programming strategy.

Question

Which guideline should agile teams avoid in relation to testing?

**Options:**

1. Test early and test often
2. Test at predefined milestones during the project
3. Use pair testing

Answer

***Option 1:****Incorrect. Agile teams should follow this guideline. It's important to test as early as possible to help minimize the impact of potential defects. Teams should also test as often as and as effectively as possible to increase the chance of finding defects.*

***Option 2:****Correct. Agile teams should avoid this guideline. Instead teams should test whenever it’s called for and as much as the situation calls for. Software for a commercial bank, for instance, requires far more testing than membership administration software for a small local company.*

***Option 3:****Incorrect. Agile teams should not avoid this guideline and should in fact use pair testing. This helps to reduce risk and helps team members to spot defects early.*

**Correct answer(s):**

2. Test at predefined milestones during the project

Question

This testing is performed without – or with very little – human interaction. While it may be difficult to achieve initially, it's important and saves a great deal of time in the long run. Which type of testing does this best describe?

**Options:**

1. Automated testing
2. Regression testing
3. Investigative testing

Answer

***Option 1:****Correct. Automated testing is performed with little, or no, human interaction. It can save time and in projects with short iterations, automating tests may be the only way to generate the number of tests necessary to create a fully tested and shippable product within given time constraints.*

***Option 2:****Incorrect. Regression testing helps ensure that any changes made to an application don't affect its existing functionality. It is conducted throughout the life cycle of a project.*

***Option 3:****Incorrect. Investigative testing is completed by independent test teams during the construction phases of an iterative project. It helps to determine the quality of the development team's work by testing working software at intervals throughout a project.*

**Correct answer(s):**

1. Automated testing

Question

Why are acceptance tests important?

**Options:**

1. They provide the opportunity for development teams to show customers that the new features they've programmed work
2. Customers have the opportunity to accept or reject new features
3. Defects are reduced
4. They help ensure that any changes made to an application don't impact on its existing functionality
5. They help to determine the quality of the development team's work by testing working software at intervals throughout a project

Answer

***Option 1:****Correct. Acceptance tests typically involve testing documented features, user stories, and use cases, allowing customers to see that features work.*

***Option 2:****Correct. Customers get the opportunity to see features working and to accept or reject based on that. However, acceptance tests should not be used to snap random defects.*

***Option 3:****Correct. Defects are reduced because developers can't consider their work complete until their code has passed the associated tests.*

***Option 4:****Incorrect. This describes a benefit of regression tests. Acceptance tests are important because they provide an opportunity for teams to show customers that features work and for customers to accept or reject those features. Acceptance tests also help reduce defects.*

***Option 5:****Incorrect. This describes investigative testing. It is completed by independent test teams during the construction phases of an iterative project and looks for any defects that the development team might have missed.*

**Correct answer(s):**

1. They provide the opportunity for development teams to show customers that the new features they've programmed work  
2. Customers have the opportunity to accept or reject new features  
3. Defects are reduced

**4. Implementing UI testing in agile projects**

The requirements of the user necessitate different techniques being applied to GUI testing. A full appreciation of the concentric testing loop method implemented during the six-step GUI-testing cycle is a prerequisite to the successful implementation of an agile development program.

Question

GUIs require different testing techniques to other software because they use user events and inputs and produce graphical outputs. There are six steps in conventional GUI testing. Sequence the steps in the correct order.

**Options:**

1. Determine what to test
2. Generate test input
3. Generate expected output
4. Execute test cases and verify output
5. Determine whether the GUI has been adequately tested
6. Perform regression testing

Answer

**Correct answer(s):**

**Determine what to test is ranked**

The first step is determining what to test: for this you can use coverage criteria to act as a set of rules.

**Generate test input is ranked**

The second step is to generating the test input. Test input is built either from software specifications or based on the way the software is structured.

**Generate expected output is ranked**

The third step is generating expected output using test oracles. In GUI testing, expected outputs often include screen snapshots and the position and title of windows.

**Execute test cases and verify output is ranked**

The fourth step is executing test cases and verifying your output. To execute a test case, you need to perform all the input events you specified. Once you've executed the test, you can compare the output with the expected output you generated using test oracles.

**Determine whether the GUI has been adequately tested is ranked**

The fifth step is deciding whether the GUI has been tested sufficiently. You can determine how well you've tested an application by analyzing the software to check which parts of it you've tested. You can then see whether you've missed any important areas that you should have tested.

**Perform regression testing is ranked**

The last step, performing regressing testing, occurs once you've identified and addressed any issues you detected during the previous steps. The tests should make sure that modifications you've made don't adversely affect any other parts of the application.

Question

To conduct continuous integration testing of GUI-based applications, you need to use both modern model-based GUI testing techniques and agile techniques. You can achieve this using a concentric testing loop method. Which statements describe this testing loop method?

**Options:**

1. Crash tests, the smallest loop in the method, run whenever GUI code is checked
2. Smoke tests, the middle loop in the concentric testing loop method, are run on every GUI build's daily progress, and are designed to complete within eight to ten hours
3. Comprehensive GUI testing, the biggest loop, is performed only when a major version of the GUI is ready
4. Oracle tests, the smallest loop in the concentric testing loop method, test whether software has functioned correctly for a given test case
5. In the concentric testing loop method, tests are executed using test cases and test oracles

Answer

***Option 1:****Correct. Crash tests are automated, inexpensive, and quick. They report software crashes within moments of code check-in.*

***Option 2:****Correct. Smoke tests run reference tests on the day's newly integrated version of the GUI, comparing the previous day's output to the new build's output. Any differences are reported to developers.*

***Option 3:****Correct. Comprehensive GUI tests are performed only for major versions of the GUI because they are in-depth and expensive integration tests that detect and report any major software and GUI issues.*

***Option 4:****Incorrect. Test oracles are performed as part of the GUI event-flow model, not the concentric testing loop method. Test cases are generated by determining the sequence of actions needed to achieve a goal state. A test oracle generates the expected output for a specified test case.*

***Option 5:****Incorrect. This explains how tests are conducted in a conventional development process.*

**Correct answer(s):**

1. Crash tests, the smallest loop in the method, run whenever GUI code is checked  
2. Smoke tests, the middle loop in the concentric testing loop method, are run on every GUI build's daily progress, and are designed to complete within eight to ten hours  
3. Comprehensive GUI testing, the biggest loop, is performed only when a major version of the GUI is ready

**5. Working with data in an agile project**

Database design and development, while limiting the agility of development teams, makes unique demands of them. Ascertaining the roles of database administrators or DBAs and establishing appropriate testing approaches for each database is key to successful and ongoing agile program development.

Question

Agile teams sometimes start a project without a database because databases limit the agility of teams. Two approaches – ObjectMother and MockObjects – have been developed to simplify the process of coding and maintaining test objects. Match each approach with its characteristics.

**Options:**

1. ObjectMother
2. MockObjects

**Targets:**

1. Removes the need to create test data for individual cases
2. Proposes a simple framework that constructs and facilitates the customization of objects
3. Simplifies and standardizes the creation of test data and saves time
4. Removes the need to generate any test data
5. Proposes building test objects that mimic real objects
6. Isolates and tests specific behavior within a system and test functions that produce unpredictable results

Answer

**Correct answer(s):**

Target 1 = Option A

Target 2 = Option A

Target 3 = Option A

Target 4 = Option B

Target 5 = Option B

Target 6 = Option B

Question

Which duties are DBAs expected to perform during an agile development project?

**Options:**

1. To perform traditional development-related duties
2. To serve as an emissary
3. To understand production database issues
4. To act as a full-fledged member of the project team
5. To be a formally trained database administrator

Answer

***Option 1:****Correct. The agile DBA needs to perform traditional development-related duties, such as creating and modifying new data sources, structures, queries, and procedures.*

***Option 2:****Correct. An agile DBA may need to act as a representative between the development team and other parties within a large project, and between users of a shared data source, such as members of other development teams and report writers.*

***Option 3:****Incorrect. A production DBA can be brought in if necessary. However, the DBA does need to have a good understanding of the software development process to respond quickly to the changing needs of the team.*

***Option 4:****Correct. This will encourage the DBA to improve the team's development environment and will encourage the DBA to recognize potential data-related issues before the team does.*

***Option 5:****Incorrect. The DBA may in fact be a team programmer, analyst, or tester who takes on the additional task of managing the team's data sources. The DBA will, however, need time to train in the database the system is using.*

**Correct answer(s):**

1. To perform traditional development-related duties  
2. To serve as an emissary  
4. To act as a full-fledged member of the project team

Question

TDD and AMDD or Agile Model Driven Development are two complementary but distinct approaches to program development. Which statements describe the AMDD approach?

**Options:**

1. It combines refactoring and test-first development
2. It’s useful if you require detailed specification and validation
3. It requires you to create models before you write any functional code

Answer

***Option 1:****Incorrect. This describes TDD where refactoring involves making a simple change to a database schema while retaining its semantics, and test-first development involves writing the test code before you write the functional code to fulfill that test.*

***Option 2:****Incorrect. This describes a feature of TDD as it enables you to take small steps when programming – usually one step involves one test and a small piece of corresponding functional code.*

***Option 3:****Correct. While the models don’t have to be perfect, they should be good enough to drive your development efforts before you begin to write any functional code.*

**Correct answer(s):**

3. It requires you to create models before you write any functional code

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_agsd_a04_it_enus/output/html/course_transcript.html#top)

© 2016 Skillsoft Ireland Limited