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

**Course Transcript**

Planning an Agile Software Development Project

**Identifying the Project Requirements**

| [1. Overview of Agile Project Planning](http://xlibrary.skillport.com/courseware/Content/cca/sd_agsd_a02_it_enus/output/html/course_transcript.html#t5) |

| [2. User Stories and Features](http://xlibrary.skillport.com/courseware/Content/cca/sd_agsd_a02_it_enus/output/html/course_transcript.html#t7) |

| [3. Estimating User Stories](http://xlibrary.skillport.com/courseware/Content/cca/sd_agsd_a02_it_enus/output/html/course_transcript.html#t9) |

| [4. Creating and Estimating a User Story](http://xlibrary.skillport.com/courseware/Content/cca/sd_agsd_a02_it_enus/output/html/course_transcript.html#t11) |

**Iteration Planning and Modeling**

| [1. Release Planning](http://xlibrary.skillport.com/courseware/Content/cca/sd_agsd_a02_it_enus/output/html/course_transcript.html#t18) |

| [2. Iteration Planning](http://xlibrary.skillport.com/courseware/Content/cca/sd_agsd_a02_it_enus/output/html/course_transcript.html#t20) |

| [3. Agile Architectures and Modeling](http://xlibrary.skillport.com/courseware/Content/cca/sd_agsd_a02_it_enus/output/html/course_transcript.html#t22) |

| [4. Planning and Modeling Tasks in an Agile Project](http://xlibrary.skillport.com/courseware/Content/cca/sd_agsd_a02_it_enus/output/html/course_transcript.html#t24) |

Overview of Agile Project Planning

Learning Objectives

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

* *identify the tasks to perform before executing iterations*
* *recognize additional approaches to agile planning*

**1. The agile customer**

Any project requires an initiation phase, which includes the once-off activities performed to set the project in motion. During this phase, a broad project vision is defined. Also, overall customer requirements are gathered, and these are shaped into a plan or model that will guide development efforts.

In traditional development, initial planning may be highly detailed and prescriptive – whereas in agile development, it's likely to be left open for further elaboration as work proceeds.

In both development models, it's customers who define the initial project vision or overall requirements that determine the functionality a team must build.

A key difference in agile development is that the customer continues to play a vital role, both in planning – during the initiation phase and beyond – and in development.

It's expected that the customer will be readily available to the development team throughout a project, and face-to-face communication is prioritized.

In an agile project, the responsibilities of an agile customer typically include

* defining and clarifying requirements, from a business perspective
* specifying and prioritizing features, and
* conducting ongoing reviews and tests

The customer's role is important throughout all phases of an agile project. However, it's especially important during release planning, when the customer actually shapes the plan for the functionality that's to be produced.

The customer involved in an agile project may represent the interests of a group, such as stakeholders in the organization for which software is being produced.

This person may consult colleagues or supervisors before making decisions, and should be an effective facilitator – bringing stakeholders together and ensuring their business needs are effectively represented.

In agile development, the emphasis on customer involvement increases the chances that a project will succeed, by

**reducing superfluous analysis activity and**

The initiation phase for a traditional project involves analyzing customers' business needs and planning how to meet these in detail before development work proceeds. But business needs may change – or not be clearly understood at the outset.  
  
In an agile project, requirements and features are elaborated and planned based on customers' current needs, as development proceeds and only once it's clear they're relevant.

**ensuring that developed functionality is relevant**

Ongoing feedback and reviews from the customer ensure that a development team properly understands and can focus on meeting business needs. The team focuses on these needs throughout the development process, instead of addressing customer satisfaction only during initial planning and again once development work has completed.  
  
So less effort is wasted on planning or building functionality that's not required, and less time is spent redoing development work.

Although all agile methodologies emphasize customer involvement, various methodologies view the role of the customer differently.

**ASD**

In Adaptive Software Development – or ASD, the customer acts as a member of the development team. All team members are viewed as agents, whose interaction over the course of a project results in the evolution of the final software product.  
  
It's not expected that the customer – or any other team member – will have a complete understanding of the product to begin with. So the customer and other team members act as peers during the progressive elaboration of requirements and features.

**DSDM**

In the Dynamic Systems Development Method – or DSDM, the main focus is on the fitness of the system that's being developed, given its business purpose. It's ultimately only the end user – or customer – who can assess this.  
  
As a result, the customer takes a leading role, directing the project team and verifying the results of each completed step of the project. It's seen as critical that all team members collaborate and cooperate, with one another and with the customer.

**XP**

In Extreme Programming – or XP – the customer must be located in the same working area as developers to enable continuous interaction. This person creates, prioritizes, and validates the user stories that the team implements, effectively controlling what's included in each release.

**Lean**

In a Lean software development process, the customer is responsible for determining the perceived integrity of the developed product. This is the extent to which the built system is the correct system and meets business requirements.  
  
So the customer plays a vital role in determining requirements and providing information to the project team.

**Scrum**

In Scrum, either the customer or someone who can accurately represent the customer's interests takes the role of product owner.  
  
This person prioritizes backlog items, effectively defining project direction by determining what functionality will be included in each development period and release. The customer also determines whether delivered functionality is acceptable.

Because customers play a role in determining and prioritizing requirements throughout the agile development process, contracts for agile projects must allow more flexibility than those for traditional projects.

Contract options for agile projects include using

**optional scope contracts or**

In an optional scope contract – also known as a negotiated scope contract or delivered-feature contract – the vendor agrees to provide a fixed number of developers, either for a specified number of iterations or for a fixed period of time. This makes costs predictable but doesn't ensure a project will have a particular scope.  
  
Optional scope contracts may be altered to give the customer slightly more control over what functionality is included, or which requirements must be met.

**target-cost contracts**

A target-cost contract binds the customer and the vendor to complete a project within a specified total cost – and if relevant by a given release date. They're then both responsible for ensuring adequate functionality is built, to meet the project goal while staying at or below the agreed cost.  
  
This type of contract can be prepared to satisfy a specific schedule for a customer who requires a speedy delivery.

Question

Match each agile methodology with a description of the role it assigns the customer.

**Options:**

1. ASD
2. DSDM
3. XP
4. Scrum

**Targets:**

1. The customer and all other team members act as peers, shaping a product as it evolves
2. The customer is responsible for assessing the fitness of a system
3. The customer continuously interacts with developers, and creates, prioritizes, and validates user stories
4. The customer is the product owner and determines what functionality to include in each release

Answer

*In ASD, all team members – including the customer – are seen as agents, whose interaction drives the evolution of the final product. So the customer and other team members act as peers.*

*In DSDM, the customer – or end user – determines the fitness of a system given its business purpose. Close collaboration with developers is required, but the customer plays a leading role.*

*XP specifies that the customer must be physically located with developers, providing feedback and direction throughout the development process. The customer also determines and validates the user stories that the team implements.*

*In Scrum, the customer prioritizes backlog items to determine what functionality will be included in each development period and release.*

**Correct answer(s):**

Target 1 = Option A

Target 2 = Option B

Target 3 = Option C

Target 4 = Option D

**2. Agile project initiation**

In agile development, planning occurs during project initiation – as it does in traditional development. However, planning also continues throughout development, with plans revisited or new plans created before the beginning of each iteration.

This is in contrast with a traditional project, in which a considerable period is spent on initial planning, and the rest of the time is spent on managing any deviations from the plan.

**Drill Down Home Page**

Four main activities mark the initiation of an agile project.

**Page 1 of 4: Defining the project mission**

At the outset of an agile project, stakeholders collaborate to create a project mission. This is the broad objective that the project must achieve, and is designed to keep the development process on track.  
  
The mission statement should be broad enough that it won't have to change even as specific requirements are elaborated or changed during development. It should also be easy for all stakeholders to understand.

Graphic

*Examples of possible mission statements are "build a company-wide knowledge portal", "replace a legacy invoicing system", and "upgrade a call center application from Java EE 5 to Java EE 6"*

Many agile methodologies advise posting the mission statement somewhere highly visible for the duration of a project. The statement may also be regularly reviewed.

**Page 2 of 4: Defining the project mission**

A well-developed mission statement has three characteristics:

* it sets a clear path for development by specifying broadly what a project must accomplish
* it inspires all stakeholders to commit to accomplishing the specified goal, and
* it provides a framework for decision-making and the development process

**Page 3 of 4: Defining the project mission**

Methodologies like XP encourage the creation of a metaphor – a simple, concise description of how a system must function, expressed using familiar, everyday objects or concepts. An example is the metaphor of a "shopping cart" for an e-commerce application that lets users select items to buy and add these to a repository.  
  
The use of a simple metaphor helps ensure that a project's overriding mission is easy for all stakeholders to understand.

**Page 4 of 4: Defining the project mission**

A workshop attended by all stakeholders – including developers, managers, and the customer – is often used to produce the mission statement or metaphor for an agile project.  
  
During the workshop, attendees form small groups and each group writes a brief statement, of no more than 25 words, about the purpose of a system. All the groups then share the statements they've developed.  
  
The most popular words and phrases should be merged into one statement, which the entire group can then review and approve.

**Page 1 of 2: Compiling a list of features**

A vital part of agile project planning is compiling a list of the features a customer initially identifies as required. These features can then be prioritized and the development effort to implement them can be estimated, as the basis for planning each iteration.  
  
The form a features list takes depends on which agile methodology is used. For example, Feature Driven Development – or FDD – uses the term features list, Scrum uses the concept of a Product Backlog, and XP uses user stories.

**Page 2 of 2: Compiling a list of features**

A story-writing workshop can be used to gather user stories or features for inclusion in a software product. The workshop typically takes the form of a single meeting in which all stakeholders – including programmers, users, domain experts, and managers – participate.

**Page 1 of 3: Creating a plan**

Once a features list has been compiled, the initial, basic planning for an agile project is comparatively straightforward. It involves allocating the listed features among the project's planned release points.

Note

*More traditional planning and project management processes may be incorporated with an agile approach.*

**Page 2 of 3: Creating a plan**

An agile project includes three main types of planning – release planning, iteration planning and daily planning.

In addition, planning may occur at the strategy, portfolio and product levels.

Note

*A portfolio includes a number of suite of products, while a strategy defines a vision that will be used to determine the products included in a portfolio.*

**Page 3 of 3: Creating a plan**

Release planning focuses on determining a project's scope, schedule, and needed resources, based on required features. It occurs during the initiation phase, but should be updated at the start of each iteration.

Iteration planning occurs at the start of each iteration. It involves prioritizing features and determining which of these to implement – and how – in an iteration, based on what work was completed in the last iteration.

During daily planning, team members meet to coordinate their work and plan which development tasks to undertake until the next daily meeting.

**Page 1 of 2: Modeling**

Creating a model involves planning exactly how a system will function and how its components will integrate.  
  
Not all agile methodologies include a distinct modeling phase. However, modeling can be useful, especially if multiple teams must collaborate in developing system components.

**Page 2 of 2: Modeling**

FDD is one methodology that does require early modeling, with a domain model created before development of required features begins. This model then drives all further development work.

Teams using other agile methodologies can use a similar approach where up-front modeling is required, to create an overall roadmap of a system. Once this is complete, the team can return to focusing on analysis and design activities at the feature – or user story – level.  
  
A more flexible form of modeling can also be incorporated with any agile methodology.

Question

Which tasks are typically performed during agile project initiation?

**Options:**

1. Creating a broad, guiding statement about the project's purpose
2. Allocating required features among planned development iterations
3. Gathering customer requirements in the form of required features
4. Determining exactly how to implement customer requirements
5. Defining measures for tracking deviations from planned progress

Answer

***Option 1:****Correct. At the outset of a project, all stakeholders participate in defining the project's mission – which must be a broad and easily understood statement about what the project should accomplish.*

***Option 2:****Correct. During the initiation phase, required features are allocated among the planned iterations. This plan is then revisited at the start of each iteration.*

***Option 3:****Correct. In agile methodologies, customer requirements are translated into discrete, required features. These can then be prioritized and estimated.*

***Option 4:****Incorrect. Teams plan how to implement required features largely during iteration planning – which occurs at the start of each iteration – and during daily planning.*

***Option 5:****Incorrect. Traditional development – rather than agile development – focuses on tracking and preventing deviations from initial plans. In agile development, plans are expected to change and evolve as work proceeds.*

**Correct answer(s):**

1. Creating a broad, guiding statement about the project's purpose  
2. Allocating required features among planned development iterations  
3. Gathering customer requirements in the form of required features

**3. Additional approaches to agile planning**

Most agile methodologies share an emphasis on delivering small, regular, and incremental releases of software, and on time-boxed development.  
  
A "timebox" is a fixed period of development – often two to four weeks – that results in working software. A release may be delivered after one or more of these periods.

In an agile project, the team assists the customer in assigning requirements to timeboxes, based on the number of developers available and the estimated time it will take to implement each requirement.  
  
This process occurs at the start of each "timebox", or iteration. The form it takes depends on the methodology that's used.

Three of the common agile methodologies differ in their approaches to time-boxed release development:

**DSDM**

In contrast with other methodologies, there is only one release to the customer in a DSDM project. What also sets DSDM apart from other methodologies is that it categorizes timeboxes depending on their functions, which include investigate, refine, and consolidate.  
  
Most investigation occurs at the start of a project, while consolidation occurs more at the end.  
  
A meeting is held at the start of each time-boxed period of development to establish its objectives. The customer reassesses priorities, and the team determines quality criteria and agrees on the minimum criteria that should be delivered.

**Scrum, and**

In a Scrum project, the customer – or Product Owner – prioritizes requirements, and developers estimate the time needed to implement them.  
  
Each time-boxed development period is known as a sprint and has a definite goal. At the start of each sprint, a Scrum team decides which items from a product backlog are achievable and how to reach the sprint goal.

**XP**

In an XP project, the customer writes user stories, each of which describes a discrete, required function. In what's known as the planning game, developers estimate the time needed to implement each user story and, on this basis, assign each story a number of points. They also estimate how many story points can be completed per iteration.  
  
The customer determines which user stories to include in an iteration based on their relative priority, but ensuring that the maximum number of story points isn't exceeded.

The Crystal Clear methodology uses what's known as "blitz planning." This is a variation of the planning game used in XP, but involves more extensive planning – for a project's first release and follow-up releases, and including technical as well as functional issues.

Blitz planning usually covers about three months of activity, which is an acceptable period in which to begin a small project and expect results.

In a blitz planning meeting, team members use index cards and write down the required development tasks they can think of. All the cards are then put together and organized. Team members use the cards to add estimates and notes, and to plan strategies for managing obstructions, costs, time, and resources.

A project team can use the blitz planning approach to complete several project initiation activities in one meeting. For example, participants may develop a features list, create a release-based plan, and perform an informal value assessment.

Participants may also address issues that with other planning approaches aren't considered until later. For instance, they may decide on the distribution of work among programmers and identify problems that may affect the first few releases of a product.

A blitz planning meeting has a collaborative nature, and the plan developed during the meeting should be accepted by every important individual or group involved in the project.

Question

Match each term associated with agile project planning to its description.

**Options:**

1. Release
2. Timebox
3. Blitz planning
4. Planning game

**Targets:**

1. A delivery of software to the customer
2. A fixed development period, resulting in working software
3. Rapid, comprehensive planning of a project during a collaborative meeting
4. A process for allocating work among iterations, based on customer priorities and development capacity

Answer

*Releases are deliveries of software. In agile development, releases are small and may be delivered after one or more timeboxes.*

*Each timebox – or iteration – is a fixed development period, planned to result in working software.*

*Blitz planning – as used in the Crystal Clear methodology – involves creating a comparatively detailed plan, addressing activities over the first few months of a project. The plan is created through a collaborative meeting.*

*The planning game used in XP enables the allocation of user stories – or required features – among multiple fixed-length development cycles. Stories are allocated by customers based on their priorities and on developers' estimates of how many story points can be covered in each iteration.*

**Correct answer(s):**

Target 1 = Option A

Target 2 = Option B

Target 3 = Option C

Target 4 = Option D

**4. Summary**

In agile development, the customer plays a central role during project initiation, as well as throughout the development process.  
  
Activities typically involved in project initiation include defining the project's mission, compiling a list of required features, and creating an initial plan of which features must be built during each of multiple iterations. A modeling phase may also be included.  
  
Most agile methodologies base project planning on releases, with development broken into "timeboxes" – or fixed-length periods. Examples of specific planning methods are the planning game used in XP and the blitz planning associated with the Crystal Clear methodology.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_agsd_a02_it_enus/output/html/course_transcript.html#top)

User Stories and Features

Learning Objectives

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

* *recognize how to write user stories*
* *recognize the purpose of user roles and proxies*

**1. Writing user stories**

In agile programming, the primary artifacts of development are known as features. Each feature is a discrete functionality that a system must provide.  
  
Features are used throughout the development process. They should be realistic and easy for both team members and customers to understand.

Features must also be estimable. This means developers must be able to estimate how long they'll take to implement. These estimates enable planning of the development process.

A user story is one example of the form a feature may take. It's a description of a discrete functionality the user requires of a system, written from the user's perspective – rather than from the developer's perspective.

Graphic

*An example of a user story for a database system might be that the customers can pay with a credit card.*

User stories are central to Extreme Programming – or XP, but have also been adapted for use in many other agile methodologies.

A user story consists of three components:

**a written description of a required functionality**

A customer produces a written description of something a system must do – for example, enable customers to make payments with credit cards. This is recorded on a story card, with a story title and number.

**conversations about the story, and**

Ongoing conversations about a user story occur between the customer and developers, throughout the release and iteration planning. This is in keeping with an iterative, incremental approach. Developers ask questions and listen as the customer elaborates the original story. Elaboration often adds detail or uncovers additional stories. It may also lead developers to restate their understanding of what the customer requires. Notes regarding the conversations are added to the story card.

**tests**

Each user story should be accompanied by tests, for verifying that the customer need it describes has been met. The tests can be listed on the back of story card. Developers are responsible for defining technical aspects and providing feedback to the customer. This enables the creation of acceptance and usability tests. Acceptance tests should always be written before programming begins.

Question

Match each component of a user story to its description.

**Options:**

1. A written description of a story
2. Conversations about a story
3. Tests

**Targets:**

1. Describes a discrete functionality required by the customer
2. Elaborates a customer's requirement
3. Confirms that a customer's requirement has been met

Answer

*At the outset of a project, the customer writes user stories on story cards. Each user story describes a discrete functionality that the customer requires in the developed software product.*

*Conversations between the customer and developers allow for elaboration of a user story throughout the development process. Understanding of the story and how it can be implemented technically evolves through this interaction.*

*For each user story, the customer and developers create an appropriate test. This is designed to verify that the requirement described in the story has been met, and should be defined before programming begins.*

**Correct answer(s):**

Target 1 = Option A

Target 2 = Option B

Target 3 = Option C

**Drill Down Home Page**

The process of using user stories to plan a development project can be broken into five main steps.

**Page 1 of 4: Step 1: Write user stories**

The process begins with customers writing user stories. At this stage, developers assist by ensuring that all stakeholders' requirements are elicited and clarified, and that each story describes a discrete functionality.  
  
User stories are usually recorded on small paper cards called "story cards." These cards are also used to record other important information throughout the development process.

**Page 2 of 4: Step 1: Write user stories**

Common techniques for eliciting user stories – or "trawling" for requirements – include user interviews, questionnaires, observation, and story-writing workshops.  
  
For the best results, a combination of these techniques is typically used. Also, the techniques are applied iteratively to accommodate the evolving nature of stories.

**Page 3 of 4: Step 1: Write user stories**

In a story-writing workshop, which is attended by customers, other stakeholders, and development team members, participants are asked to write as many stories as they can. This is an effective way of gathering a large number of stories in a limited time.

Questions used to elicit or clarify user stories should be open-ended and context-free – they shouldn't indicate assumptions on the part of developers.

**Page 4 of 4: Step 1: Write user stories**

There is no one way in which user stories should be structured. Some examples of user stories include

* "as a user, I can enter my employment details"
* "a company can pay for a job posting with a credit card", and
* "a project manager must be able to access all project data"

**Page 1 of 2: Step 2: Select an iteration length**

Next the development team and customer select an appropriate iteration length, based on project requirements. In agile development, this is typically a development period of two to four weeks.  
  
Each iteration will result in working software, which builds incrementally on what was developed in the previous iteration.

**Page 2 of 2: Step 2: Select an iteration length**

Short, incremental iterations ensure that customer feedback about working software can be obtained regularly throughout development.  
  
Also, plans can be revisited after each iteration – so adjustments can be made as a system evolves.

Note

*There are overhead costs associated with each iteration, which can make shorter iterations a slightly more expensive option.*

**Page 1 of 3: Step 3: Estimate work**

Once an iteration length has been decided, the development team needs to plan which user stories to implement in each iteration.

To do this, it first estimates how long it will take to build each of the features described in user stories and, on this basis, assigns each story a number of story points.

Then the team determines a velocity for the project. This determines how many story points can be allocated to each iteration, based on the number of available developers.

**Page 2 of 3: Step 3: Estimate work**

To determine velocity, a team may

* use historical values, if they're available for similar development projects
* run an initial iteration and use the actual velocity of that iteration for those that remain, or
* make an estimate – which is the most common approach

**Page 3 of 3: Step 3: Estimate work**

Say a small team of eight programmers who work on a project divided into iterations of two weeks each – with 10 working days per iteration. This means eight programmers will be available for 10 days per iteration – which gives you a total of 80 days of programming work.  
  
If one story point is equal to one day of programming, ideally the programmers will be able to complete 80 story points every iteration. Velocity will therefore be calculated as a value of 80.

However, the actual number of programming days available should be estimated between a third and a half of ideal days available to take into account down time such as sick leave and so on. Therefore, the velocity should be calculated as being roughly 35.

**Page 1 of 2: Step 4: Prioritize and allocate stories**

Once velocity is established, the customer prioritizes all user stories and records a priority rating on each story card.  
  
The cards can then be sorted into piles based on priority, and developers will allocate higher-priority stories to the earliest iterations.

Note

*The number of story cards assigned to each iteration will depend on the velocity established for the team.*

**Page 2 of 2: Step 4: Prioritize and allocate stories**

Say a team of developers divides all user story cards into three piles – "Must have", "Should have", and "Could have" – based on the priority ratings assigned by the customer.  
  
The team's velocity means that it can complete roughly 35 story points per iteration. So when planning what work to include in an iteration, the team starts by including stories in the "Must have" pile.  
  
Once that pile has been allocated and there's still capacity left in an iteration, story points from the "Should have" pile – which has lower priority – can be selected, until the total for an iteration reaches 35. This approach continues in the development planning for each successive iteration.

**Page 1 of 2: Step 5: Update estimates and iterations**

The final step of creating user stories is an ongoing one. It involves continually updating story estimates and the planned work for iterations, as a project proceeds.

**Page 2 of 2: Step 5: Update estimates and iterations**

As development proceeds, user stories are elaborated – and new stories may be uncovered. Also, a team may find that user stories take either more or less time to implement than initially expected.

So the planning for each successive iteration must be revised to reflect actual progress. For example, some story points initially assigned to one iteration may be moved to the next, if they can't be completed in time.

The team may also revise its velocity if user stories are proving either more or less time-consuming to implement than originally anticipated.

Question

How is a user story developed?

**Options:**

1. The customer writes a description of the story
2. The story undergoes elaboration during the development process
3. The customer and developers specify how they'll test that the story has been implemented properly
4. The customer estimates the work needed to implement the story
5. Developers assign the story a priority rating

Answer

***Option 1:****Correct. The story writing process begins with the customer writing a description of a user story, outlining a required system feature.*

***Option 2:****Correct. The story undergoes elaboration during development, as developers gain understanding of its technical implications and interact with the customer.*

***Option 3:****Correct. Before development begins, the customer and developers establish how they'll test that the requirement identified by a story has been fully met.*

***Option 4:****Incorrect. Developers – rather than the customer – need to estimate the time and work required to implement the feature identified by a user story.*

***Option 5:****Incorrect. The customer – rather than a developer – must be responsible for assigning each user story a relative priority.*

**Correct answer(s):**

1. The customer writes a description of the story  
2. The story undergoes elaboration during the development process  
3. The customer and developers specify how they'll test that the story has been implemented properly

**2. User roles and proxies**

Planning in an agile development process is driven by user stories.  
  
This makes it crucial that the stories accurately represent users' requirements.

A challenge in writing accurate user stories is that often there isn't just one type of system user. A developed system may have a variety of types of users, each with different requirements of the system.

This means it's often impossible to write a good user story from a single user's perspective. A story like this won't reflect the experiences and goals of all relevant users, and can cause developers to miss requirements from users who don't fit the profile of the primary user type.

Creating user roles can help developers elicit accurate user stories that fully represent the requirements of all end users of a system.

A user role is a collection of defining attributes that characterizes a population of users and their intended interactions with a system. For example, you might create a user role for first-time users and one for returning users.

Each user role highlights needs specific to a particular group of end users – but user roles often overlap. For example, say you identify five user roles for a job recruitment web site – Job Poster, Head Hunter, First Time User, Returning User, and Job Seeker.

In this example, users in the Head Hunter and Job Poster roles both have the goal of finding appropriate job candidates – so their needs of the system overlap. Similarly, job seekers – including both first-time users and returning users of the site – will all use the search features of the site.  
  
However, the way users in each role will use the site – and how often – will differ, and it's important to reflect this in user stories.

In agile development, you should create each user role based on information you gather, either from

**a real user or**

Gathering information from real users is ideal because only users can know exactly what their system requirements are.  
  
Many agile methodologies emphasize the need for a customer or an end user to be onsite with a development team and to participate throughout the development process.

**a proxy**

A proxy is someone who can represent the interests of a user role when real members of a user group aren't available – for example, because of geographical or confidentiality constraints. Using proxies isn't ideal, but may be necessary.

Question

What is the purpose of user roles and proxies in agile development?

**Options:**

1. To accurately represent the perspectives of all end users
2. To break user requirements into discrete, required features
3. To help a development team calculate its velocity
4. To ensure ongoing customer feedback during development

Answer

***Option 1:****Correct. User roles and proxies represent the perspectives of different types of end users so that these can be accurately reflected in the user stories for a project.*

***Option 2:****Incorrect. User stories – rather than user roles or proxies – break user requirements into discrete, required features, for which the needed development work can be estimated.*

***Option 3:****Incorrect. A team's velocity establishes how much work it can complete per iteration. This isn't affected by user roles or proxies, which represent the perspectives of different types of end users.*

***Option 4:****Incorrect. User roles and proxies represent the perspectives of different types of end users so that these can be accurately reflected in the user requirements captured for a project.*

**Correct answer(s):**

1. To accurately represent the perspectives of all end users

**3. Summary**

Features are the primary artifacts that drive planning and development in agile methodologies. User stories are descriptions of required system features, created from the perspective of users. A user story includes a written description, conversations about the story that result in its elaboration during development, and tests.  
  
Identifying user roles can help developers ensure that user stories accurately reflect the requirements of all system users. They should be derived based on input from real users or – where this isn't possible – from proxies who represent their interests.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_agsd_a02_it_enus/output/html/course_transcript.html#top)

Estimating User Stories

Learning Objectives

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

* *identify techniques for estimating stories*
* *recognize how to split stories in given examples*

**1. Story estimating techniques**

Customers generally want to know when a product will be ready. At the outset of a project, however, there's no way an agile team can provide an exact answer to this question.  
  
Instead, an agile team provides an estimate of how long each user story – or required feature – is likely to take to develop. The point isn't to be precise, but rather to provide a reasonable and educated guess.

Note

*The precision of an estimate is likely to decrease as the size of a story increases.*

A good technique for estimating stories has these characteristics:

* it can be used to estimate stories of different size or complexity
* it can be completed quickly so that not too much time is wasted
* it accounts for inevitable imprecision in original estimates
* it allows for adjustments as a project proceeds and more accurate information about progress becomes available, and
* it provides estimates that can be used as a basis for release planning

**Drill Down Home Page**

Four methods are commonly used to estimate stories.

**Page 1 of 3: Story points**

Estimates are commonly calculated in terms of developer hours. The story point approach doesn't, however, always relate directly to hours.  
  
It can be better understood as a measurement of the effort required to implement a story.

**Page 2 of 3: Story points**

A story point is a relative unit of measurement. This means that a point defined by one team doesn't necessarily equate to a point defined by another.  
  
One team might, for example, define a story point as one ideal day of work while another defines it as a week, or as a perceived level of complexity.

**Page 3 of 3: Story points**

Once each user story has been assigned a certain number of story points, an agile team calculates how many story points it can handle per fixed-length iteration. This will depend on the number of developers available, as well as on iteration length.

The customer can prioritize user stories and decide which to include in each iteration, provided the maximum number of story points for an iteration isn't exceeded.

**Page 1 of 3: Triangulation**

After a few estimates have been made, a team can begin to triangulate the estimates.  
  
Triangulation is a process of estimation based on the relationships between stories. Estimates are produced by measuring stories against one another.

**Page 2 of 3: Triangulation**

If two stories have been estimated, for example, you can consider the stories in relation to one another.  
  
If one of the stories is estimated at 8 story points and the other at 4 points, you'd note that the first story is roughly twice the size of the second. If a third story is estimated at 6 story points, you'd note that it's smaller than the first story but larger than the second.

**Page 3 of 3: Triangulation**

Most commonly, an agile team triangulates story estimates by pinning story cards into columns on a poster. Each column represents a particular number of story points, and as each new story is estimated, it's pinned in the appropriate column.  
  
Each newly estimated story should then be compared to other stories in the same column to determine whether it's of a similar size.  
  
This enables a team to verify that it's not gradually altering the meaning of a story point.

**Page 1 of 3: Themes and epics**

It's easiest to estimate user stories of similar sizes. However, different development situations call for different story sizes.

For example, a customer may be unsure about whether particular functionality is required, preferring to wait until later to make this decision. In this case, it's preferable to write one large story rather than several smaller, more detailed stories. This type of large story is called an epic.

In other situations, small stories that are related may be combined. These combinations of user stories are called themes.

Note

*The size of an epic means that it often constitutes a theme on its own.*

**Page 2 of 3: Themes and epics**

Using epics and themes can reduce the time and effort you spend on estimating user stories. However, the estimates are likely to be much more imprecise than for smaller stories.

**Page 3 of 3: Themes and epics**

One way to estimate epics and themes is to give all participants in a meeting of developers a set of cards representing possible estimates.  
  
A description of each story is read and then considered by the participants. Developers then all select the cards that represent their estimates of how long the story will take to implement.  
  
At this stage, developers' estimates tend to differ quite widely. Variations are discussed and the process is repeated until the estimates converge, and the resulting single estimate is accepted.

Note

*An alternative approach can be used when dealing with large numbers of stories. Stories can be represented on cards that are then ordered in a horizontal scale according to estimate size.*

**Page 1 of 4: Planning poker**

Planning poker is a way of estimating user stories that results in fairly reliable estimates within a short space of time.  
  
It's usually played before a project officially begins. Once the project is underway, it's used to estimate any new stories that emerge.

**Page 2 of 4: Planning poker**

All project developers, including programmers, testers, database engineers, analysts, and so on, participate in playing planning poker.  
  
The customer also participates in the exercise but doesn't make any estimates.

**Page 3 of 4: Planning poker**

Planning poker begins when participants are gathered together and given packs of cards that represent possible estimates.  
  
A moderator reads a story description and the customer – or product owner – answers any questions that developers may have.

Each of the participants then selects a card representing their estimate.

Variations among the selected estimates are discussed and the process is repeated until the estimates converge, resulting in acceptance of a single estimate.

Note

*Teams can be split into smaller groups of at least three members each. Each group then needs to ensure that it makes estimates consistent with those of other groups.*

**Page 4 of 4: Planning poker**

Planning poker has several advantages:

* it involves a cross-functional team, with multiple expert opinions, in the task of estimating stories
* it promotes an involved discussion that helps compensate for the lack of detailed information at the beginning of a project, and
* group discussion leads to an averaging of individual developer's estimates, which results in more reliable estimates

Question

Match techniques for estimating stories to their descriptions.

**Options:**

1. Story points
2. Triangulation
3. Themes and epics
4. Planning poker

**Targets:**

1. Estimates a story based on its relationship to one or more other stories
2. Uses a relative unit of measurement to estimate the development effort required to implement a user story
3. Saves time spent on estimating by aggregating stories
4. Brings together multiple expert opinions to produce reliable estimates quickly

Answer

*Triangulation involves estimating stories after they've been assigned story points, by comparing stories with the same numbers of points. It's an effective way for a team to verify that it's not gradually altering the meaning of a story point.*

*A story point is a relative unit of measurement, representing a fixed amount of development effort.*

*Themes and epics are large, or combined, user stories. Estimating these instead of larger numbers of smaller stories saves time but results in less precise estimates.*

*Planning poker is a game in which a cross-functional team produces reliable estimates in a short time.*

**Correct answer(s):**

Target 1 = Option B

Target 2 = Option A

Target 3 = Option C

Target 4 = Option D

**2. Recognizing how to split stories**

Sometimes it's necessary to split an epic or a theme into smaller, more discrete user stories.  
  
For example, you may split a description of an epic like "As a user, I can order a camera online" into multiple user stories. These might specify that the user must be able to "browse camera types by brand, price, model, and description", "pay with a Visa credit card", and "enter a residential address for delivery."

Note

*You should be careful not to create stories that are too small when splitting an epic. For example entering a phone number is too small for a user story. Like epics, small stories make prioritization and estimation difficult.*

There are two main reasons why you might need to split a story:

**the story is too large to fit in one iteration or**

A single user story must be split if it will take longer to implement than the fixed period assigned to each iteration in the development process. Otherwise an iteration won't result in working software, and iterations will be difficult to plan.

**a more accurate estimation is needed**

The estimate for a large story will be more imprecise than one for a smaller story. If the customer requires a more accurate estimate before deciding whether to include a story, the story should be split. It may also be necessary to split stories during release and iteration planning to allow for more accurate estimates.

**Drill Down Home Page**

There are a number of ways to split stories.

**Page 1 of 2: Across data boundaries**

One of the easiest and most frequently used approaches is to split stories based on the different types of data to which they relate.

**Page 2 of 2: Across data boundaries**

Say you're developing a product that must collect customer feedback. You may begin with the epic user story, "As a customer, I can enter feedback."

You might then decide to split the story based on the type of data that the user could enter. For instance, the first story may be "As a customer, I can enter personal information", and the second may be "As a customer I can enter feedback on a product."

By splitting the story in this way, you can create stories that are small enough to estimate accurately and that will each fit within a single iteration.

**Page 1 of 2: Across operational boundaries**

Splitting by operation is useful for stories that cover a number of different operations.  
  
The most common approach is to split a story based on the create, read, update, and delete operations that must be developed to implement it.

**Page 2 of 2: Across operational boundaries**

For example, you might split a large user story such as "As a user, I can manage my theater bookings" into these user stories:

* "I can add new reservations"
* "I can update information about existing reservations", and
* "I can delete reservations to cancel them"

**Page 1 of 2: Based on priority**

A story may include multiple sub-stories, each with a different priority. In this case, the story can be split based on priority levels.

**Page 2 of 2: Based on priority**

Say a user story for an online shopping portal is "A user can select product requirements and then view a list of matching products, with the system highlighting the products that are the best matches."  
  
In this case, the product owner wouldn't ship the product if it didn't support the core goal of enabling customers to navigate to a page containing products that match their specifications. However, the product could be shipped if it doesn't highlight the best matches.  
  
The story can therefore be split into two smaller stories, with the story that houses the core goal of the product going into the first iteration and the second story possibly left for later development.

In some situations, it's necessary to combine multiple stories into a larger story, rather than estimating each of the smaller stories separately.

For example, it's very common to combine stories requiring bug reports, which are very small on their own.

Also, combining related stories can make the task of prioritizing user stories easier.

Question

A user story for an online shopping portal for an airline company currently reads "As a user, I can search for flights."  
  
In which ways could you choose to split this user story?

**Options:**

1. Based on the different types of data the user must be able to use to retrieve flight listings
2. Based on the different system operations required
3. Based on the priorities of sub-stories

Answer

***Option 1:****Correct. This story can be split along data boundaries, based on the types of search criteria that must be supported. For example, it may be split into stories such as "I can search for a flight by airline", "I can search for a flight by date", and "I can search for a flight by destination."*

***Option 2:****Incorrect. This story doesn't require distinctly different types of operations, like create, read, update, and delete operations.*

***Option 3:****Incorrect. In this example, the original user story doesn't include sub-stories with differing priority levels.*

**Correct answer(s):**

1. Based on the different types of data the user must be able to use to retrieve flight listings

**3. Summary**

Techniques for estimating user stories include using story points, using triangulation, merging small stories into themes or epics, and playing planning poker.  
  
User stories may be split if they're too big to fit within a single iteration or more accurate estimates are required. Stories are most commonly split based on data, operational, or priority boundaries.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_agsd_a02_it_enus/output/html/course_transcript.html#top)

Creating and Estimating a User Story

Learning Objectives

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

* *create a user story*
* *evaluate user stories*
* *estimate a user story*

**1. Exercise overview**

In this exercise, you're required to recognize considerations for creating, evaluating, and estimating user stories in given scenarios.

This involves the following tasks:

* recognizing how to create user stories
* evaluating user stories, and
* identifying ways to estimate user stories

**2. Creating user stories**

Question

Which components should a user story include?

**Options:**

1. A written description of functionality required by the customer
2. Ongoing conversations about the story
3. A test for verifying that the story has been properly implemented
4. An appropriate iteration length for the story
5. An estimation of the required work

Answer

***Option 1:****Correct. A customer produces a written description of a user story, outlining specific functionality that the developed software product should include from a business or user perspective.*

***Option 2:****Correct. Ongoing conversations about a user story occur between the customer and developers, throughout the development process. Understanding of the story evolves through this interaction.*

***Option 3:****Correct. Each user story should be accompanied by one or more tests for verifying that the requirement described in the story has been met.*

***Option 4:****Incorrect. Iteration lengths are decided independently of the sizes of user stories. Once stories have been gathered, however, they're assigned to iterations based on customers' priorities and on the time it's estimated they'll take to complete.*

***Option 5:****Incorrect. User stories are estimated once they've been gathered, but the estimates don't form part of the stories themselves.*

**Correct answer(s):**

1. A written description of functionality required by the customer  
2. Ongoing conversations about the story  
3. A test for verifying that the story has been properly implemented

Question

Which techniques can agile development teams use to gather user stories?

**Options:**

1. User interviews
2. Observation
3. Story-writing workshops
4. Developer interviews
5. Tests

Answer

***Option 1:****Correct. A team can interview users individually to elicit information about their requirements of a system.*

***Option 2:****Correct. Observing users completing business tasks or using existing software is a common way to elicit the users' requirements. It can result in identification of user stories that would otherwise be overlooked.*

***Option 3:****Correct. Story-writing workshops are useful for obtaining user stories from a group of stakeholders. They're ideal when a large number of stories is needed and there's a limited amount of time to collect them.*

***Option 4:****Incorrect. In an agile process, customers or users – rather than developers – provide the requirements outlined in user stories.*

***Option 5:****Incorrect. User stories should include appropriate tests – but tests aren't generally a way to gather users' requirements of a system.*

**Correct answer(s):**

1. User interviews  
2. Observation  
3. Story-writing workshops

Question

An agile development team needs to develop an online reservation portal for a hotel situated within the theater district.  
  
Which of these user story descriptions is appropriate?

**Options:**

1. "A user can make a reservation."
2. "A user can deal with reservations online."
3. "A user can manage a reservation."
4. "A user can confirm theater show times."

Answer

***Option 1:****Correct. This story is relevant and will add business value to the portal for the hotel.*

***Option 2:****Incorrect. This story is too large and needs to broken down into separate stories for creating and managing a reservation.*

***Option 3:****Correct. This story can add business value by adding flexibility to a guests' online reservation experience with the hotel's portal.*

***Option 4:****Incorrect. Arguably this functionality wouldn't add business value as it's as simple for the user to select the required reservation dates as to have them optionally matched to a theater reservation. Additionally if the functionality is required it should form part of the user story to create a reservation.*

**Correct answer(s):**

1. "A user can make a reservation."  
3. "A user can manage a reservation."

**3. Evaluating user stories**

Once an agile development team has recorded user stories, it needs to evaluate these and then to split or combine stories where necessary.

Question

An agile team is improving the product search feature for an online store that sells classical music in MP3 format. An epic story for the search feature is recorded as "As a user, I can search for music."  
  
Which user stories can be derived from this epic?

**Options:**

1. "As a user, I can search for classical music."
2. "As a user, I can search for music by artist."
3. "As a user, I can search for music by title."
4. "As a user, I can search for all types of music."

Answer

***Option 1:****Incorrect. This is also an epic story – it doesn't describe a discrete, required functionality, and it may not be possible to implement it in one iteration.*

***Option 2:****Correct. This user story describes one of the required functionalities indicated by the epic story description.*

***Option 3:****Correct. This user story can be derived from the epic. It describes one of the ways in which users must be able to search for music, so it identifies a discrete, required functionality.*

***Option 4:****Incorrect. This is also an epic story. It's likely to be difficult to implement in one iteration and doesn't describe a discrete, required functionality.*

**Correct answer(s):**

2. "As a user, I can search for music by artist."  
3. "As a user, I can search for music by title."

Question

A user story for an online shopping portal for a clothing retailer is "As a user, I can buy clothes online."  
  
Which user stories can be derived from this epic?

**Options:**

1. "As a user, I can browse by garment, color, and size."
2. "As a user, I can pay for my purchase with a Visa credit card."
3. "As a user, I can buy shoes online."
4. "Common classes will be used to manage errors."

Answer

***Option 1:****Correct. This is an appropriate user story because it identifies discrete, required functionality. It could be implemented in one iteration or – if necessary – be easily split into smaller stories that could be.*

***Option 2:****Correct. This is an effective user story because it identifies discrete, required functionality for which development work can be estimated and planned.*

***Option 3:****Incorrect. Like the original story, this is an epic. It's only slightly more specific and doesn't identify discrete, required functionality that could be implemented in a single iteration.*

***Option 4:****Incorrect. This information is relevant only to developers, not customers. The user story could be rewritten as "Errors will be displayed and logged in a consistent fashion." to make it more relevant for customers.*

**Correct answer(s):**

1. "As a user, I can browse by garment, color, and size."  
2. "As a user, I can pay for my purchase with a Visa credit card."

Question

Brocadero, an international baggage shipping company, requires an online data management system. One user story for the system is "As a user, I can enter my shipping requirements." Users must be able to specify the required routes, date and time of arrival, and various other delivery details.  
  
How should this user story be split?

**Options:**

1. Across data boundaries
2. Along operational boundaries
3. According to priority

Answer

***Option 1:****Correct. This story could be split along data boundaries, based on the types of data users must be able to enter.*

***Option 2:****Incorrect. The original user story doesn't indicate the need for different types of operations. It specifies that the user must be able to enter data but not, for example, to update or delete data.*

***Option 3:****Incorrect. The original user story doesn't indicate the existence of a set of sub-stories that are of different priority to the customer.*

**Correct answer(s):**

1. Across data boundaries

**4. Estimating a user story**

After splitting or combining user stories where necessary, a development team needs to estimate the time or effort required to implement each of the stories.

Question

An agile team needs to develop a web site for the publisher of a weekly newspaper. The team gathers user stories, records them on cards, and assigns them to columns based on their estimated sizes. It then compares the size and complexity of all the stories in a single column, making adjustments to original estimates on this basis.   
  
Which method is the team using to estimate the stories?

**Options:**

1. Triangulation
2. Themes and epics
3. Planning poker
4. Story points

Answer

***Option 1:****Correct. Triangulation is a process of estimation based on the relative size and complexity of stories. Estimates are produced by measuring stories against one another.*

***Option 2:****Incorrect. In this example, triangulation is being used. Themes and epics are broad user stories, in which multiple smaller stories are combined. Estimates can then occur at a less detailed level.*

***Option 3:****Incorrect. In this example, triangulation is being used. Planning poker doesn't involve producing estimates by comparing the relative size and complexity of stories.*

***Option 4:****Incorrect. User stories aren't assigned story points based on their relative size and complexity. However, triangulation may be used after story points are assigned.*

**Correct answer(s):**

1. Triangulation

Question

A development team needs to estimate user stories for an online reservation system quickly. All developers and the customer meet, and they're given packs of cards representing possible estimates. For each story, all the developers select a card. They then discuss variations in the estimates they've chosen and repeat the process until they converge on a single estimate.  
  
Which method is the team using to estimate the user stories?

**Options:**

1. Planning poker
2. Themes and epics
3. Triangulation
4. Story points

Answer

***Option 1:****Correct. In planning poker, all developers select estimates for a user story and then discuss variations in the estimates they've chosen. They then repeat the process and collaborate to arrive at a single estimate. This can result in fairly reliable estimates within a short space of time.*

***Option 2:****Incorrect. Estimating themes and epics, rather than smaller user stories, saves time. However, it doesn't involve having developers select estimates, discuss variations, and repeat their selections until they arrive at a single estimate per user story.*

***Option 3:****Incorrect. Triangulation involves producing estimates by comparing the relative size and complexity of stories. It doesn't involve a collaborative process in which developers select estimates, discuss variations, and repeat their selections to arrive at a single estimate per user story.*

***Option 4:****Incorrect. Although possible estimates may have been expressed using story points, the process of having developers select estimates, discuss variations, and repeat their selections until they arrive at a single estimate per user story is known as planning poker.*

**Correct answer(s):**

1. Planning poker

Considerations for creating, evaluating, and estimating user stories have been identified.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_agsd_a02_it_enus/output/html/course_transcript.html#top)

Release Planning

Learning Objectives

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

* *recognize how to create a release plan*
* *calculate project buffer in a given scenario*

**1. Release planning activities**

A release is a delivery of production-ready software to the customer. In agile development, which is time-boxed, a release may be delivered after one or more iterations.

Through release planning, an agile team determines what work to complete and when, in order to deliver software with required features within an acceptable timeframe.

An emphasis in agile development is that each iteration should result in working, fully tested software.

Although the project team delivers software that's ready for production on a regular basis, the decision for a release to go into production lies with the customer.

Before a release can be planned in detail, a target for determining customer acceptance of the release must be set. The customer – or product owner – decides this, in collaboration with developers.

Graphic

*A detailed, sequential release plan calls for customer collaboration in establishing project success criteria, estimating, arranging and prioritizing stories, choosing iteration lengths and team velocity, and setting the release date and further project stories.*

Typically, agile projects are either

**date-driven or**

In a date-driven project, a release is scheduled for a fixed date. Features included in the release are negotiable – lower-priority features may be left out if it's not possible to build them by the agreed date.

**feature-driven**

In a feature-driven project, the overriding goal is to deliver a set of required features. If these features take more or less time to build than anticipated, the estimated release date may be changed.

Once the overall target for a release has been agreed and an initial set of customer requirements – in the form of user stories or a required features list – has been gathered, the customer and development team collaborate to plan the release.

Release planning includes four key activities:

**establish project success criteria**

It's important to identify the criteria that will be used to determine a project's success or failure. These are set by the customer and help determine the schedule, scope, and resources required for a project.  
  
Often criteria for success relate to financial goals. For example, a project's success may be judged based on the amount of money saved or generated.

**estimate features or user stories**

The development team and customer begin by assessing the initial list of features or user stories for a release. They update the items based on any relevant information gathered during the previous release.

**choose iteration length and velocity and prioritize user stories**

Before development work begins, the team plans an appropriate iteration length – typically from one to four weeks. Each iteration will then be kept to this length. If the customer has set a fixed release date, the number and length of iterations will be planned accordingly.  
  
Next the team determines its velocity – or the number of story points it can complete per iteration. It also estimates the time or effort required to implement each of the required features, or user stories.  
  
With assistance from the team, the customer prioritizes the user stories to indicate their relative importance.

**choose user stories and set release date**

The customer typically selects the features or user stories for the team to develop in each iteration, while ensuring that these don't exceed the team's prediction of how much work it can complete per iteration. High-priority features are usually allocated to the earliest iterations.  
  
In a feature-driven project, the customer and developers then estimate a flexible release date, usually as a range of acceptable dates.  
  
In a date-driven project, it may already be clear at this point that some compromises are needed – with low-priority features left for future releases so that the target date can be met.

Question

Sequence the key activities that occur during release planning for an agile project.

**Options:**

1. Estimate user stories
2. Estimate velocity
3. Allocate features to iterations and estimate or confirm a release date
4. Establish project success criteria

Answer

**Correct answer(s):**

**Establish project success criteria is ranked**

The product owner's conditions of satisfaction can be determined by a combination of schedule, scope, and resources.

**Estimate user stories is ranked**

The development team and customer assess the initial list of user stories for a release then they update the items based on information gathered during the previous release.

**Estimate velocity is ranked**

The team estimates its velocity to determine how much work – or how many features – it can complete per iteration.

**Allocate features to iterations and estimate or confirm a release date is ranked**

As the final step, the customer typically determines which features or stories to develop in each iteration, taking team velocity and iteration length into account.

**2. Prioritizing stories**

When prioritizing user stories, the customer may simply categorize them as being of "Low", "Medium", or "High" priority.  
  
However, some agile methodologies outline more specific techniques for assigning stories priority levels.

One of the prioritization techniques – which is used in Dynamic Systems Development Management, or DSDM – is referred to as MoSCoW. In the acronym, each capital letter represents a particular priority level.

**Must have**

Must-have features are crucial to the system. Typically they should be allocated for development in the earliest iterations.

**Should have**

Should-have features are important but it's possible to implement short-term workarounds for them. They're viewed as mandatory if a project has no time limitations. Otherwise, compromises may be made.

**Could have**

Could-have features can be left out of a release if the project team runs out of development time.

**Would have**

Would-have features are desired but marked for probable inclusion in a later release.

Whichever technique is used, various factors should be taken into account when priority levels are assigned.

Customers' criteria may include the relative popularity of each feature among a broad base of users or customers, or among a smaller group of key stakeholders.

Customers may also consider how each story fits in with other stories. For example, a "zoom out" feature may not be high priority but be treated as such because it complements a "zoom in" feature, which does have high priority.

While customers ultimately decide on priority levels, the development team needs to ensure that technical factors are considered. These include

**the risk that a story won't be completed by the desired time**

High-risk stories, like those that involve functionality that's complex to develop, should generally be handled earlier – and therefore given higher priorities – so there's time to compensate for delays.

**the impact that a story may have on other stories, and**

If the implementation of one or more stories depends on another story, this story may require a higher priority setting.

**infrastructure or performance-related requirements**

If a feature will have a significant effect on the architectural design and performance of a system, it should be assigned a higher priority.

If a customer finds it difficult to prioritize a story, it's sometimes possible to split the story into smaller stories with more obvious, relative priorities.  
  
For example, a user story may initially be described as "Users can search for songs by artist, title, genre, album, or a combination of these."  
  
This could be split into three stories – a high-priority story for searching by artist or title, a medium-priority story for searching by genre or album, and a low-priority story for searching using a combination of the criteria.

Developers then estimate user stories, determining the number of days – or story points – each will take to complete.

Question

Which technical factors may influence the way a user story for a release is prioritized?

**Options:**

1. The risk that the story can't be completed as desired
2. The impact that the story may have on other stories if it's postponed
3. The story's popularity among a broad base of users or customers
4. The estimated time it will take to develop the story

Answer

***Option 1:****Correct. Typically, high-risk stories should be handled earlier – and therefore given higher priorities – so there's time to compensate for delays.*

***Option 2:****Correct. A technical factor to consider is that a story on which one or more other stories depend should be given a higher priority.*

***Option 3:****Incorrect. The popularity of a user story among stakeholders isn't a technical factor, although it's one a customer may apply when prioritizing stories.*

***Option 4:****Incorrect. The estimated development time for implementing user stories doesn't affect the way they're prioritized – although it does influence the way they're allocated to specific iterations.*

**Correct answer(s):**

1. The risk that the story can't be completed as desired  
2. The impact that the story may have on other stories if it's postponed

**3. Estimating duration using velocity**

The developers and customer decide together on an iteration length. Shorter iterations allow more frequent course corrections and make it easier to track progress. However, this can be more expensive because it means using a greater number of iterations, and some operating cost is associated with each iteration.

The development team then estimates its velocity in one of three main ways, by

**using a historical value**

If the same team worked on a recent project that's similar to the new one, it may use its actual velocity from the previous project. It's not always possible to do this, but it results in a fairly reliable velocity estimate.

**running an initial iteration and using the velocity of that iteration, or**

The team may complete a first iteration and use its actual velocity from that iteration as the basis for further release planning. However, it's possible that the initial iteration won't be representative of other iterations.

**establishing an estimate based on a simple calculation**

The team may base its velocity calculation on estimates of the ideal number of days required to complete each user story, and then factor in likely delays. This is the most common approach.

Based on the velocity it estimates, the team can calculate how many iterations it will take to complete a project.  
  
Say the estimates for all user stories for a project add up to 150 story points, and a story point represents one ideal day of work for the development team. If velocity is estimated at 25, you divide 150 by this value to get six iterations.

Note

*In a date-driven project, the number of iterations is fixed. Iteration length and velocity will determine how many user stories can be completed – and others may be left for inclusion in later releases.*

Question

It's estimated that the user stories for a project will take 100 days of development time to implement, and the team's velocity is set at 25.   
  
How many iterations will it take to complete the project?

**Options:**

1. 4
2. 0.25
3. 5

Answer

***Option 1:****Correct. To calculate the number of iterations it will take to complete the project, you divide the number of ideal days by the velocity.*

***Option 2:****Incorrect. To calculate the number of iterations it will take to complete the project, you divide the number of ideal days by the velocity – you don't divide velocity by the number of ideal days.*

***Option 3:****Incorrect. To calculate the number of iterations it will take to complete the project, you divide the number of ideal days by the velocity.*

**Correct answer(s):**

1. 4

**4. Creating a release plan**

Based on the estimated velocity, the length and number of iterations, and user story estimates, the customer and development team collaborate to create a release plan.

This involves determining what to include in the release. The team may indicate exactly what it expects to develop during each iteration, or only what will be developed during the overall release.

It takes more time and effort to assign specific features to particular iterations. However, this extra detail is useful, especially if work needs to be coordinated among several teams.

The process of determining which stories to allocate to which iterations is driven by the stories' priorities, which are determined largely by the customer.

However, the development team's velocity and iteration length determine how many stories can be allocated to each iteration.

It's accepted that the plan will change as the project progresses and the team learns more. For this reason, it's often considered best to allocate stories only to the first few iterations. Planning can then be revisited for subsequent iterations.

To simplify the planning process, the team may pin actual user story cards to columns representing iterations on a wall chart.

This makes it easy to compare the relative amount of work in each iteration – and cards can easily be moved as work progresses, to enable all team members to track progress.

Note

*A team may also use a spreadsheet to track the features for development in each iteration.*

Question

Which activities may occur as part of finalizing a release plan?

**Options:**

1. Allocating user stories among iterations
2. Determining whether a project should be accepted
3. Calculating how many iterations it will take to complete the project
4. Estimating and prioritizing user stories

Answer

***Option 1:****Correct. To create a release plan, the customer and development team determine which user stories to include and how to allocate these among iterations.*

***Option 2:****Correct. Based on initial release planning, the customer and team can determine whether a project is likely to meet acceptance criteria – with enough key features implemented within an acceptable timeframe.*

***Option 3:****Incorrect. The number of required iterations is established earlier, before a release plan is finalized.*

***Option 4:****Incorrect. This is one of the first release planning activities. Stories are estimated and prioritized before a release plan can be finalized.*

**Correct answer(s):**

1. Allocating user stories among iterations  
2. Determining whether a project should be accepted

Based on initial release planning, the customer and developers can determine whether a project will meet criteria for its acceptance.

For example, the team can estimate a release date – and therefore likely cost – for a feature-driven project.

For a date-driven project, it can determine whether a core set of user stories that will provide business value can be implemented by the fixed release date.

**5. Calculating a project buffer**

The scope of an agile project often changes as it progresses – and this is expected. For example, a customer may want to add functionality to a release, with implications for a team's ability to meet a fixed release date. Certain features may also be omitted if development progresses slower than planned.

In a DSDM project, the team stipulates that during planning, a maximum of 60% of the features – or user stories – allocated to an iteration can be "must-have" features. Ensuring lower-priority features are included adds flexibility. It means these can be omitted if new, higher-priority features are added after the initial planning phase.

In Feature-Driven Development – or FDD – the acceptable amount of change to a project's scope within a given cycle is quantified. If scope increases by more than 10%, the project manager informs the customer and requests that either features be removed from the plan or that the delivery date be extended.

If a development team realizes it's irreversibly behind schedule, it should notify the customer as soon as possible. The customer can then make changes to the required functionality or the delivery date of the project.

In agile planning, a release schedule is likely to be particularly uncertain if, for example,

* a release is planned far in advance
* a release has a fixed deadline as well as a fixed set of required features, or
* development work for a release is outsourced to another company

To account for uncertainty when committing to a particular release date, an agile team may add buffers to the initial estimates it makes. This enables the team to absorb unexpected delays or increases in scope.

Two types of buffers may be used:

**feature buffers and**

A feature buffer is a percentage of features that may be omitted if development time runs short or new feature requirements are introduced. Typically it's set at 25% to 40% of the features allocated per iteration.  
  
For example, feature buffering is used in DSDM, where features are prioritized according to MoSCoW rules. Approximately 70% of the planned effort for each iteration is targeted at must-have requirements. This leaves a feature buffer equivalent to 30%.

**schedule buffers**

A schedule buffer helps ensure that an agile team can meet a fixed release date. Extra time is added to the release date that's initially estimated to enable the team to absorb unexpected delays or increases in scope.

To calculate an appropriate buffer, you need to quantify the uncertainty involved in the development process. One way to do this is to determine two estimates for each user story – an average or "most likely" estimate, and a worst-case estimate.

It's important to apply these estimates at the level of user stories – rather than to an over-all release – because each story may represent a different level of risk, or uncertainty.

It can also help to express your level of confidence in each estimate, to further quantify the uncertainty it involves. For example, you may determine there's a 50% likelihood that a user story can be completed within two days, but a 90% likelihood that it could be completed within five days.

Your 50% estimate applies to the average case, reflecting what's most likely given an average number of obstacles. Your 90% estimate applies to the worst case, if you don't believe it's plausible that the story could take longer than five days to develop.

Note

*The percentage for the worst-case estimate acknowledges that although uncertainty remains, it's extremely unlikely that every possible risk or delay will be realized.*

To begin determining an appropriate schedule buffer based on your average and worst-case estimates, you subtract the average case from the worst case and square the result for each story.

For example, if a user story is likely to take two days to complete but could take up to five days, you subtract 2 from 5 to get 3, and then square the result to get 9.

You repeat this process for each story in turn, and then add the results for all the stories.

Once you've calculated the results for each story, you add them. You then find the square root of the total.

Say the results for all the stories in a release have added up to 81. The square root of 81 is 9. This means that nine days is an appropriate schedule buffer for the release.

To get the total project duration, you add the totals of the average-case estimates to the buffer – for example, adding 9 days to a total average development time of 55 days. As a rule of thumb, you should arrive at a schedule buffer that constitutes at least 20% of the total duration.

Question

Say you have 2 user stories in a release, one with an average time estimate of 6 days and a worst-case estimate of 10 days and the other with an average time estimate of 8 days and a worst-case estimate of 11 days.  
  
What is an appropriate schedule buffer for the release?

**Options:**

1. 25 days
2. 5 days
3. 7 days

Answer

***Option 1:****Incorrect. To calculate the buffer, you need to calculate the square root of the sum of the differences squared between the worst and average case estimates for each user story.*

***Option 2:****Correct. The difference between the 2 estimates in the first task is 4, squared is 16.  The difference between the 2 estimates in the second task is 3, squared is 9.  The sum of 16 and 9 is 25, and the square root of 25 is 5.*

***Option 3:****Incorrect. To calculate the buffer, you need to calculate the square root of the sum of the differences squared between the worst and average case estimates for each user story.*

**Correct answer(s):**

2. 5 days

**6. Summary**

Release planning in an agile project includes four key activities – prioritizing and estimating user stories, setting an iteration length, estimating the team's velocity, and then allocating required features among iterations.  
  
User stories are typically assigned priorities by the customer, although with developers providing input about technical factors that must be considered. Higher-priority stories will then be allocated to earlier iterations. Developers also estimate the development time or effort required to develop each story.  
  
A development team estimates its velocity to indicate how much work – or how many features or user stories – it can complete per iteration. Based on this, total project duration can be predicted.  
  
In the final stage, stories are allocated to each iteration and the likelihood that a release will satisfy acceptance criteria can be judged.  
  
To account for uncertainty, an agile team may add both feature and schedule buffers to the estimates it gives the customer.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_agsd_a02_it_enus/output/html/course_transcript.html#top)

Iteration Planning

Learning Objectives

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

* *sequence the steps in velocity-driven iteration planning*
* *recognize the activities of commitment-driven iteration planning*

**1. Iteration planning**

During the release planning phase, an agile development team prioritizes features or user stories, determines iteration length and velocity, and creates an initial release plan. When it's time for an iteration to start, it takes this planning further with iteration planning*.*

Iteration planning generally consists of updating and elaborating on the release plan. Its objective is to generate the kind of detail necessary to drive the work that will be done during a particular iteration.

Before an iteration planning meeting, you need to adjust the release plan for the imminent iteration. You do this by

**determining the amount of work the team can complete**

To determine the amount of work the team can complete in the iteration, you need to consider the team's velocity and the iteration's length. You should also factor in things like time for which developers will be unavailable – for example, due to vacations, training seminars, or company meetings.

**updating the list of work for the release**

The customer and the team need to ensure that the user stories associated with the current release are completely up to date.

**setting a goal for the iteration, and**

Each iteration needs to have a set goal so that the team maintains focus as development work proceeds. Normally the customer sets the goal. It could be to develop a specified set of features or user stories, or it could be a broader objective such as reducing risk in a specified area.

**selecting the functionality to develop**

The functionality to be developed during each iteration may be determined during the release planning stage. If the iteration time has been estimated, the customer selects a functionality that's appropriate for that period of time. If the iteration time hasn't been estimated, the customer simply identifies stories of the highest priority.

An iteration planning meeting should be attended by the entire team, including developers, testers, database engineers, and so on. It's also useful to have the customer present at the meeting so that this person can answer questions about user stories as they arise.

Iteration planning meetings can last anywhere from a couple of hours to a whole day, depending on the size of the team and the length of the relevant iteration.

The iteration plan itself can be compiled on a spreadsheet, a wiki, or even as a set of cards on a board. Any medium will do, as long as team members find the plan easy to follow.

During the iteration planning meeting, you

**present user stories**

Ideally, the customer should be the one to present the user stories and answer developers' questions about them.

**translate stories into development tasks, and**

Once the user stories allocated to an iteration have been explained, the stories must be translated into discrete, technical development tasks. Breaking each story down into smaller steps has further benefits, such as making it easier to divide work among developers and to track their progress.

**confirm a reasonable commitment by the team**

All developers need to confirm their commitment to completing the tasks that have been identified. You also need to ensure that developers don't take on more work than they can handle.

Question

Specific activities are completed prior to an iteration planning meeting.  
  
Which one of these activities doesn't usually involve the customer?

**Options:**

1. Determining the amount of work the team can do during the iteration
2. Updating the user stories for the current release
3. Selecting the functionality to be developed during the iteration
4. Selecting a goal for the iteration

Answer

***Option 1:****Correct. The customer isn't usually in a position to determine how much work the team can complete. This depends on an understanding of the technical work involved in implementing what the customer requires.*

***Option 2:****Incorrect. The customer needs to be involved in updating the user stories, together with the team.*

***Option 3:****Incorrect. The customer needs to play the main role in selecting the functionality to develop, based on the relative priority of user stories.*

***Option 4:****Incorrect. The goal for the iteration is usually set by the customer, based on business needs.*

**Correct answer(s):**

1. Determining the amount of work the team can do during the iteration

**2. Velocity-driven iteration planning**

Iteration planning can be said to be *velocity-driven*if the team's velocity is the most important factor in determining how much work will be allocated for each iteration.

**Drill Down Home Page**

The velocity-driven iteration planning process can be divided into six steps.

**Page 1 of 2: Adjust priorities**

The priorities assigned to user stories should be adjusted as required at the start of each iteration. This is because the priority levels of different user stories are liable to change as a project progresses.

Reprioritization typically occurs during an iteration review, after the customer has had a chance to view the most recently completed features.

It's useful to schedule the planning meeting for the next iteration and the review of the previous iteration on the same day, to ensure that reprioritization happens quickly. This also helps ensure that any unfinished work from the previous iteration can be prioritized for the next one.

**Page 2 of 2: Adjust priorities**

The user stories associated with a release are usually prioritized according to these criteria:

* their development cost, in terms of both money and time
* their business value to the customer
* the amount of risk involved in creating them
* the significance of what the team will learn when creating them, and
* dependencies among the stories

**Page 1 of 2: Determine target velocity**

Velocity refers to the amount of work the team believes it can complete per day or per iteration.  
  
The usual point of reference is the amount of work completed in the previous iteration. This is known as the concept of "yesterday's weather" – a title originating from the idea that the best factor for predicting today's weather conditions is conditions from the day before.

**Page 2 of 2: Determine target velocity**

Another approach is to use the average amount of work created over the last few iterations.

**Page 1 of 2: Identify an iteration goal**

The goal for the iteration should be a succinct description of what the team and customer would like to accomplish.  
  
For example, a team creating a fitness club web site might decide that "complete all search features" is a suitable goal for an iteration.

**Page 2 of 2: Identify an iteration goal**

However, the goal doesn't need to be highly specific. It could, for example, simply be to make progress with the search features or to reduce risk in a certain area.

**Page 1 of 2: Select user stories**

The customer and the team select user stories that, when combined, reflect the iteration goal.

**Page 2 of 2: Select user stories**

For example, if the goal for an iteration is to finish all the search features for a fitness club web site, the user stories that reflect this goal could be

* "as a client, I can search for training sessions according to time and date parameters"
* "as a client, I can search for group classes using keywords", and
* "as a trainer, I can enter and update the data necessary to make my classes searchable"

**Page 1 of 3: Split user stories into tasks**

Each user story selected must be broken down into the separate development tasks required to implement the corresponding functionality.

**Page 2 of 3: Split user stories into tasks**

The user story "As a trainer, I can upload images of my classes", for example, might be translated into a list of tasks such as

* design the user interface
* get user interface feedback from the customer
* determine image size limits
* code the user interface
* test the user interface, and
* determine the amount of storage space available for uploading images

**Page 3 of 3: Split user stories into tasks**

The list of tasks identified and estimated for each user story needs to be comprehensive. The team should ensure that it includes

* only tasks that add value to the project
* unit testing tasks
* meetings, including required preparation for these
* tasks associated with fixing bugs
* tasks necessary for handling dependencies, and
* tasks that are necessary to determine or support other tasks

**Page 1 of 1: Estimate tasks**

Estimating tasks involves making an educated guess about how much time will be spent performing each task. Generally, it's recommended that tasks shouldn't be designed to involve more than one day's work.

The time needs to be estimated by the team as a whole. This helps to minimize the possibility of developers taking on more work than they can handle or estimating too much time for a given task.

It may be necessary for some design discussion to take place as the team estimates tasks. However, this shouldn't be detailed or too long. For example, it shouldn't include drawing class diagrams or other models – this can be done after the meeting.

Question

An agile development team is currently planning its third iteration in a project to develop a hospital administration program. Its planning is based on a velocity-driven approach.  
  
Sequence examples of the steps that the team needs to take to plan the iteration.

**Options:**

1. Ask the hospital administrator whether his priority for this iteration has changed
2. Determine how much work the team did during the second iteration
3. Establish the customer's goal for the iteration
4. Select user stories that reflect the goal of the iteration
5. Identify all the development tasks that need to be completed during the iteration
6. Estimate how long each of the required development tasks will take to complete

Answer

**Correct answer(s):**

**Ask the hospital administrator whether his priority for this iteration has changed is ranked**

First the team needs to find out if the priorities of the customer have changed and adjust them accordingly.

**Determine how much work the team did during the second iteration is ranked**

As a second step, the team needs to estimate how much work it can complete during the iteration, based on the amount of work completed in the previous iteration.

**Establish the customer's goal for the iteration is ranked**

The customer needs to set the goal for the iteration. This typically occurs once the priorities of user stories have been adjusted and the team knows how much work it can complete during the iteration.

**Select user stories that reflect the goal of the iteration is ranked**

Once a goal for the iteration has been set, the team needs to select user stories that relate to this goal.

**Identify all the development tasks that need to be completed during the iteration is ranked**

As the fifth step, the specific development tasks that will result in the functionality described by the user stories must be identified.

**Estimate how long each of the required development tasks will take to complete is ranked**

The last step in the process involves estimating how long each required development task will take to complete.

**3. Commitment-driven iteration planning**

Commitment-driven iteration planning is an alternative to velocity-driven planning.  
  
It involves many of the same steps as velocity-driven iteration planning, but places emphasis on commitment as a basis for how many features the team can create during an iteration.

Essentially, the team decides how many user stories it can commit to completing by looking at each one individually, and then adding them one by one until an iteration is "full."

Like for velocity-driven planning, the team and customer start by adjusting priorities and setting an iteration goal. Then the team starts selecting stories to add to the iteration.

One story is selected, it's broken into a set of development tasks, and the tasks are estimated. The team then asks whether it can commit to delivering that story.

If it can, it selects and expands another story. The team then discusses whether it can commit to the next story, and the process continues until the team can't commit to further work.

If the team finds that it can't commit to a story that it has just expanded, it selects a different one and tries again until an iteration is full.  
  
It should be noted that the second step featured in velocity-driven planning, where the team determines its velocity by referring to past iterations, is dispensed with altogether.

Question

How does commitment-driven iteration planning differ from velocity-driven planning?

**Options:**

1. There's less emphasis on velocity
2. No more than one story is expanded at a time
3. The team uses the "yesterday's weather" concept to estimate the work it can complete
4. The customer doesn't play a significant role in the planning process

Answer

***Option 1:****Incorrect. The commitment-driven approach doesn't involve using the velocity of the team at all.*

***Option 2:****Correct. The team selects and expands one story, and decides whether to commit to it before it proceeds with the next story.*

***Option 3:****Incorrect. The concept of "yesterday's weather" is used to estimate the team's velocity, which is not a feature of commitment-driven planning.*

***Option 4:****Incorrect. The customer takes part in the process by helping to set the goal for the iteration and adjusting priorities. The customer is also involved in the selection of stories.*

**Correct answer(s):**

2. No more than one story is expanded at a time

**4. Planning releasable iterations in Scrum**

In the Scrum methodology, an iteration is defined as a sprint. As with an iteration, each sprint ends with either a software result, or a release that can be shown or deployed in the case of the final sprint, to the customer.

Before each sprint, the team holds a sprint planning meeting. This is similar to an iteration planning meeting.

Before the meeting starts, the entire list of features assigned to the project and the priority levels of these features are updated by the customer. This list is called a *product backlog*.

The sprint planning meeting is broken into two parts, each with a specific objective.

Graphic

*Each part of the meeting typically lasts for about four hours.*

**Defining the sprint objective**

The first part of the meeting is attended by the entire project team and the customer. The customer describes the highest-priority items to the team. Then the team and the customer decide together on the functionality to be completed during the sprint. This is called the *sprint objective*.

**Defining the sprint backlog**

The second part of the meeting is attended only by the team, with the customer on standby. The team breaks the features identified as part of the sprint objective into a list of estimated development tasks. This list is called the *sprint backlog.*

Question

Match activities to the part of a sprint planning meeting in which they're performed. Not all of the activities are valid.

**Options:**

1. The customer describes high-priority features
2. The team estimates the time necessary for each task
3. The team defines a mission statement
4. The customer adjusts the list of features for the release

**Targets:**

1. Defining the sprint objective
2. Defining the sprint backlog

Answer

*To define the sprint objective, the customer needs to describe the features that are currently the most important to include.*

*To define the sprint backlog, the team expands each required feature into a list of tasks and estimates the time required to complete each task.*

**Correct answer(s):**

Target 1 = Option A

Target 2 = Option B

**5. Summary**

Iteration planning elaborates on the planning done during the release planning phase. The release plan is adjusted for the impending iteration, and then an iteration planning meeting takes place. During this meeting, user stories are broken up into estimated tasks.  
  
Iteration planning can be said to be velocity-driven when the velocity of the team is the biggest factor in determining how to allocate work to an iteration. The team's velocity is determined based on work completed in the previous iteration.  
  
Commitment-driven planning is an alternative to velocity-driven planning. Teams deconstruct and estimate each user story before committing to it, and repeat this process until an iteration is full.  
  
The Scrum methodology features sprints, each of which defines both a release and an iteration. Teams hold sprint planning meetings, which are similar to iteration planning meetings, to determine the objective and a backlog of development tasks for each sprint.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_agsd_a02_it_enus/output/html/course_transcript.html#top)

Agile Architectures and Modeling

Learning Objectives

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

* *identify features of agile modeling*
* *recognize how AMDD activities fit into the agile software development process*

**1. Features of agile modeling**

Agile modeling is a methodology for the effective modeling and documentation of software-based systems.  
  
Model Driven Development, or MDD, is an approach that involves creating extensive models of projects before any source code is written. The agile version of MDD involves creating incremental models just sufficiently detailed to support development throughout the project lifecycle.

Graphic

*Model Driven Development, or MDD, requires that Iteration 0, comprising the initial high-level requirements envisioning and architectural envisioning processes, take place simultaneously and inform each other over several days.  
  
The modeling of the first and succeeding iterations' development is an agile and self-contained process. Specific iteration modeling, measured in hours, leads to model storming, measured in minutes. Both processes are informed by a third process, Test Driven Development, or TDD, measured in hours.*

The team can begin by developing a high-level model of the project requirements and go on to develop more low-level models for each iteration requirement.

Generally, agile methodologies don't specifically include a modeling phase. Those that do, such as Feature Driven Development – or FDD – and the Agile Unified Process – or AUP – are exceptions. These methodologies are prescriptive about what type of model to create and how to create it.

However, agile modeling describes a set of values and principles for agile modeling that can be incorporated in any agile methodology.   
  
Agile modeling is sometimes described as "chaordic", meaning that it combines the chaotic system of simple modeling practices with the more ordered system of software modeling artifacts.

Question

Which are characteristics of agile models?

**Options:**

1. The Modeling phase is included by default in all agile methodologies
2. Agile modeling can be described as chaordic
3. Modeling in agile projects is completed at the planning phase
4. FDD includes a prescriptive modeling phase

Answer

***Option 1:****Incorrect. Most agile methodologies do not include a modeling phase by default. Exceptions include FDD and AUP.*

***Option 2:****Correct. Agile modeling can be described as chaordic, meaning that it is both a chaotic system of simple modeling practices and an ordered system of software modeling artifacts.*

***Option 3:****Incorrect. In Agile development, incremental models that are sufficiently detailed are created to support development throughout the project lifecycle.*

***Option 4:****Correct. FDD includes a modeling phase, while most other agile methodologies don't specifically include a modeling phase.*

**Correct answer(s):**

2. Agile modeling can be described as chaordic  
4. FDD includes a prescriptive modeling phase

Using agile modeling before starting development work can increase productivity by preventing setbacks from occurring later. Its success, though, will depend on the way the principles it defines are applied in practice.

For modeling to be truly agile, it must be used with an overall development methodology that's also agile. However, a team using a more traditional approach may still benefit from using agile modeling techniques.

There are several principles associated with agile modeling that should inform the development process from beginning to end. These include

**assuming simplicity**

This entails assuming that the simplest solution to a problem is always the best solution. It also means that you keep your models simple, and don't include unnecessary features.

**modeling with a purpose**

You need to ensure that you always have a clear idea of why and for whom you are designing an artifact. You must know what the audience really needs and what the purpose of the system as a whole will be.

**valuing content above representation**

Any piece of modeling can be represented in a number of different ways. Placing more value in content means that you have the benefit of modeling without being too concerned with documentation.

**creating quality work**

This principle has benefits for everyone, from the developers and refactoring team to the stakeholders and users.

**communicating openly, and**

Open communication means that every member of the team feels confident about making suggestions and sharing insights about the project. It also means that stakeholders are kept informed regularly of the status of the project.

**traveling light**

Imagine the project as a suitcase full of components that needs to be carried and maintained over time, and you will see the benefit of dispensing with unnecessary artifacts. Keeping your work simple and not overburdening it will keep it agile.

The remaining principles of agile modeling are

**embracing change**

Requirements change as people's understanding of the requirements change. You need to ensure that the development environment isn't resistant to those changes by making it more agile.

**developing multiple models**

Different models and artifacts serve different purposes. Having a wide range of models available will help you to meet challenges better, just like having a range of tools in a toolbox.

**maximizing stakeholder involvement**

Stakeholders invest money and resources into getting the best result possible for their particular needs. They should be able to comment on how those resources are used.

**rapid feedback, and**

It's important to have a clear idea of what the customer needs from the system, but an inconvenient reality is that requirements may change over time. In order to eliminate lost work hours and avoid changes, it helps to get feedback from the stakeholder early on in the project and keep getting more feedback often.

**incremental change**

Agile development is incremental by its very nature, so your models don't need to be perfect the first time you develop them. You should develop simple models that are just good enough at first, and improve them over time.

Modeling should be done with the purpose of communicating with people external to your team.  
  
You increase communication by maximizing stakeholder involvement and displaying your models in a public space. Modeling with others also enables someone else to spot any mistakes you've made before you work too far ahead of them.

On a team, applying modeling standards and patterns ensures that work is consistent, and can improve quality and productivity. However, you need to ease into the application of patterns gently.

Other practices that increase the overall quality of work are considering testability throughout the process, iterating to the next modeling artifact whenever you get stuck, and modeling in small increments.

Simplicity in modeling can be achieved by reusing existing resources, discarding temporary models, using the simplest tools for each task, and updating models only when it's absolutely necessary. Your content and your depictions should be simple, and you should always model to understand.

It can be helpful to create several different models in parallel and always try to choose the right one to get each job done. There must be a sense of collective ownership among team members, and the effectiveness of every model should be proven with code.

In the context of agile modeling, models could take the form of documentation, diagrams, textual descriptions of business processes, or collections of cards. It doesn't matter which tools you use to create models – it's the way you use them that's important.

Question

Which are characteristics of agile models?

**Options:**

1. They include a high level of detail
2. They're as simple as possible
3. They fulfil clear purposes
4. They're technical and designed to be understood chiefly by programmers

Answer

***Option 1:****Incorrect. An agile model should be just detailed enough to fulfil its purpose.*

***Option 2:****Correct. Simplicity is one of the defining features of agile models.*

***Option 3:****Correct. An agile model should be simple while fulfilling a clear purpose.*

***Option 4:****Incorrect. An agile model should be easy for the target audience – including stakeholders like the customer – to understand.*

**Correct answer(s):**

2. They're as simple as possible  
3. They fulfil clear purposes

**2. AMDD in the development process**

While traditional MMD takes a serial form, Agile Model Driven Development – or AMDD – takes an iterative form. Each iteration includes activities that are part of the agile modeling process.

Graphic

*In AMDD, iteration 0 includes identifying the initial requirements, creating a high-level architectural plan, and initial setup and planning.   
  
The succeeding development iterations could each include iteration modeling, model storming, Test Driven Development – or TDD, and investigative testing.*

The goal of the envisioning process in iteration 0 is to determine the scope of a system and the framework in which you can build it. When using AMDD, you should avoid writing detailed specifications and aim to create a very broad and flexible strategy.   
  
Each iteration that follows iteration 0 includes modeling activities both before and during coding.

**Drill Down Home Page**

AMDD includes five main phases.

Graphic

*During Iteration 0, the steps to be carried out include initial requirements envisioning and initial architecture envisioning.  
  
During each successive development iteration, the steps to be carried out include iteration modeling, model storming, and Test Driven Development.*

**Page 1 of 1: Initial requirements envisioning**

The initial requirements envisioning phase should give you a good idea of what your system needs to do and how users will work with it. It could take a few days to a week, but you should try to make it as short as possible to avoid the risk of over-modeling.

It should include an initial user interface model if development of a user interface is part of the project, and a brief look at usability issues. It should also contain an initial domain model, which will help you identify business entity types and the relationships between them.  
  
Stakeholders should be closely involved in this phase so that you generate a genuinely shared understanding.

**Page 1 of 4: Initial architecture envisioning**

In the initial architecture envisioning phase, you should try to create an infrastructure that has a good chance of succeeding while being as simple as possible. You use this infrastructure to organize teams and their tasks in a broad manner.

Because of its simplicity, you could use the architectural vision for subsequent releases, and for those releases you could minimize or even dispense with iteration 0.

**Page 2 of 4: Initial architecture envisioning**

You should aim to create an initial architecture model that's not sophisticated, but just barely enough to get the ball rolling. The evolutionary nature of agile development means that this infrastructure can and most likely will change with time.

**Page 3 of 4: Initial architecture envisioning**

During the envisioning stages, activity diagrams can be used to represent and understand a business model in terms of its required activities.

Activity diagrams can also be divided into swim lanes to describe how components share tasks.

**Page 4 of 4: Initial architecture envisioning**

Throughout the project lifecycle, other representation of the business model that can be used to facilitate development include

* component diagrams which show individual software components and how they are linked together
* interaction diagrams that show how components and their activities interact with each other, and
* class diagrams are used to describe the equivalent classes required to build the system

**Page 1 of 2: Iteration modeling**

The iteration modeling phase is the phase in each iteration during which work is estimated. You create more detailed models in this phase than in the previous ones, but they shouldn't take long to create because each concerns only one iteration.

You estimate what you need to build and then estimate the amount of work required to implement specific requirements. You could make use of records of how much your team accomplished in previous projects per iteration. This typically takes up to a few hours.  
  
The feature list or product backlog that you compile can be represented by a use-case diagram. This can be used to show how stories are grouped and what dependencies are between stories, which helps to see clearly the effects of moving or adding a story.

**Page 2 of 2: Iteration modeling**

Software modeling systems enable you to generate partial coded solutions, documentation and schemas and so on from models that you have created. This can expedite development and facilitates teams to incorporate changes in requirements.

**Page 1 of 1: Model storming**

Model storming sessions can be planned, or could happen spontaneously as needed. A handful of people would typically gather around a whiteboard, or a similar tool, and briefly explore an issue before continuing.

These sessions should typically last only a few minutes, and could take half an hour at the most.

**Page 1 of 1: Test Driven Development (TDD)**

TDD phases should take up the majority of your team's time. They consist of tests that implement what has just been modeled.

You do this through executive specifications such as customer tests or development tests. TDD allows you to work through very focused issues, which supplements the detailed modeling you do during model storming sessions and iteration development. The refactoring process helps to evolve a design in small steps.  
  
You could use visual modeling tools such as Rational Software Architect during this phase.

Question

Which one of these phases of agile modeling should be the least time-consuming?

**Options:**

1. Iteration modeling
2. Executive specifications
3. Storm modeling
4. Initial requirements envisioning
5. Initial infrastructure envisioning

Answer

***Option 1:****Incorrect. Estimating the work for each iteration should take a few hours.*

***Option 2:****Incorrect. This phase involves testing, and should take up more time than any other phase. It could involve a few hours or even days' work within each iteration.*

***Option 3:****Correct. Storm modeling is intended to take the form of a quick brainstorming session between a few colleagues and should last no longer than half an hour.*

***Option 4:****Incorrect. Initial requirements envisioning may take up to a few days. Although this phase is important, it shouldn't result in specifications that are too detailed.*

***Option 5:****Incorrect. Envisioning the initial infrastructure is done only once, and should take a couple of days. Like initial requirements envisioning, it shouldn't be too detailed.*

**Correct answer(s):**

3. Storm modeling

**3. Summary**

Agile modeling is a methodology for successful documentation and modeling of software development projects. It's comprised of a collection of practices guided by specific principles and values.   
  
AMDD activities are incorporated into every iteration of an agile development project. In iteration 0, the envisioning process involves identifying high-level requirements and creating a high-level infrastructure for building the system. Subsequent iterations include iteration modeling, model storming, and TDD activities.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_agsd_a02_it_enus/output/html/course_transcript.html#top)

Planning and Modeling Tasks in an Agile Project

Learning Objectives

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

* *complete activities common to the release planning phase of an agile software development project*
* *complete activities common to the iteration planning phase of an agile software development*
* *implement agile modeling*

**1. Exercise overview**

In this exercise, you're required to recognize how to plan and model tasks in an agile development project.

This involves the following tasks:

* identifying activities common to the release planning phase of an agile software development project
* identifying activities common to the iteration planning phase, and
* recognizing the phases of agile modeling

**2. Planning a release**

An agile development team is currently planning a release for a project that involves developing a new computer game.  
  
Together with the customer, the team has already gathered and estimated user stories for inclusion in the release.

Question

Which factors should influence the priority levels assigned to user stories for the release?

**Options:**

1. The priorities of the customer and of the stakeholders this person represents
2. Implications of particular stories for system infrastructure and performance
3. The chosen iteration length for the release
4. The team's estimated velocity
5. Any dependencies among the stories
6. The level of risk associated with each of the stories

Answer

***Option 1:****Correct. In an agile project, it's largely the customer who decides on the relative priorities of user stories – based on the needs of users and business stakeholders.*

***Option 2:****Correct. Developers should ensure that if failing to implement particular user stories could significantly compromise system infrastructure and performance, these stories are given higher priorities.*

***Option 3:****Incorrect. Iteration length will help determine how many user stories the team can develop, but it doesn't affect the relative priorities of the stories.*

***Option 4:****Incorrect. The team's velocity indicates how much work it can complete per time interval. This doesn't affect the relative priorities of user stories.*

***Option 5:****Correct. User stories on which one or more other stories depend should generally be given higher priorities.*

***Option 6:****Correct. User stories with the highest risk should typically be implemented early on in the development process, so they should be given higher priorities.*

**Correct answer(s):**

1. The priorities of the customer and of the stakeholders this person represents  
2. Implications of particular stories for system infrastructure and performance   
5. Any dependencies among the stories  
6. The level of risk associated with each of the stories

Question

The team and customer decide to use MoSCoW rules to prioritize the user stories. One example of a user story for the release is "As a player, I can delete a game from inside the game." The customer wants the corresponding functionality to be included, but it has a fairly low priority and it's agreed that it may be left out if development time runs short.  
  
To which priority category should the user story be assigned?

**Options:**

1. Must have
2. Should have
3. Could have
4. Would have

Answer

***Option 1:****Incorrect. A "must-have" story is one critical to the system and that has a very high priority. Without it, a release won't be viable.*

***Option 2:****Incorrect. A "should-have" story has a high priority and, ideally, shouldn't be left out of a release.*

***Option 3:****Correct. A "could-have" story is one describing functionality it would be preferable to include, but that may be left out if development time runs short. A release will be viable without it.*

***Option 4:****Incorrect. "Would-have" stories are marked for development in a later release – although they may be added back into the current release if extra development time is available.*

**Correct answer(s):**

3. Could have

Question

The team estimated the development time required to implement each user story using story points, where one story point represents one ideal day of work. The total number of story points for all the stories is 200, and the team has estimated its velocity as 25.  
  
How many iterations will it take to complete the release, given that the project is feature-driven?

**Options:**

1. 8 iterations
2. 64 iterations
3. This depends on iteration length and the required release date

Answer

***Option 1:****Correct. The velocity value specifies the amount of work the team estimates it can complete – in this case, 25 story points per iteration. So to calculate how many iterations it will take to complete the release, you simply divide the total number of story points by the team's estimated velocity.*

***Option 2:****Incorrect. To calculate how many iterations it will take to complete the release, you divide the total number of story points by the team's velocity. You don't then square the result.*

***Option 3:****Incorrect. In a feature-driven project, the release date is generally flexible – the focus is on completing the development of required features.*

**Correct answer(s):**

1. 8 iterations

Question

The team needs to give the customer an estimated release date. Because the release requires some types of work that are new to the team, it's important to add a buffer period to the release date – to account for possible delays. Average estimates indicate that the user stories for the release are likely to be completed within 15 work days. Worst-case estimates suggest the stories could take up to 40 days to complete.  
  
How long should the buffer be?

**Options:**

1. 25 days
2. 5 days
3. 27.5 days

Answer

***Option 1:****Correct. To calculate an appropriate schedule buffer, you first determine the difference between the average and worst-case estimates – which in this case is 25. However, you should then calculate the square root of this value.*

***Option 2:****Incorrect. To calculate an appropriate schedule buffer, you determine the square root of the difference between the average and worst-case estimates. In this case, the difference is 25 – and the square root of this value is 5.*

***Option 3:****Incorrect. To calculate an appropriate schedule buffer, you determine the square root of the difference between the average and worst-case estimates. You don't simply average the two estimates.*

**Correct answer(s):**

1. 25 days

**3. Planning an iteration**

Immediately after creating a release plan, the customer and development team planned the first development iteration. This iteration was completed successfully – so the team now needs to plan the second iteration for the release.

Question

The team's approach to iteration planning is velocity-based. Sequence the steps that the team needs to take to plan the second iteration.

**Options:**

1. Update and adjust the priorities assigned to user stories, based on input from the customer
2. Establish how many story points the team can complete during the iteration
3. Establish the customer's goal for the iteration
4. Select user stories that reflect the goal of the iteration
5. Break user stories down into discrete development tasks
6. Estimate the time needed to complete each development task

Answer

**Correct answer(s):**

**Update and adjust the priorities assigned to user stories, based on input from the customer is ranked**

First the team needs to ensure that all user stories for the release are still prioritized appropriately, based on the customer's input.

**Establish how many story points the team can complete during the iteration is ranked**

As a second step, the team needs to estimate its velocity. This is how much work – or how many story points – it can complete during the second iteration, typically estimated based on the amount of work it completed in the first iteration.

**Establish the customer's goal for the iteration is ranked**

The customer needs to set the goal for each iteration. This typically occurs once the priorities of user stories have been adjusted and the team has estimated how much work it can complete during the iteration.

**Select user stories that reflect the goal of the iteration is ranked**

Once a goal for the iteration has been set, the team and customer needs to select user stories that relate to this goal.

**Break user stories down into discrete development tasks is ranked**

Once user stories have been selected, each story must be converted into a list of the specific development tasks needed to implement it.

**Estimate the time needed to complete each development task is ranked**

The last step in the process involves estimating how long each required development task will take to complete.

Question

How does commitment-driven iteration planning differ from velocity-driven planning?

**Options:**

1. The step of estimating velocity is omitted
2. No more than one story is expanded into a set of estimated tasks at a time
3. No definite goal for an iteration is set
4. It's unnecessary to estimate development tasks

Answer

***Option 1:****Correct. The commitment-driven approach doesn't involve using the velocity of the team as a basis for planning. Instead the team selects, expands, and commits to one user story at a time, until the development time available in an iteration is full.*

***Option 2:****Correct. In commitment-driven iteration planning, a team selects and expands one story, and decides whether to commit to it before it proceeds with the next story.*

***Option 3:****Incorrect. Like in velocity-driven planning, a step in commitment-driven planning is setting an over-all goal for an iteration.*

***Option 4:****Incorrect. Irrespective of which planning approach is used, it's necessary to estimate the time needed to complete each development task – to ensure that all the required tasks can be completed within the given iteration.*

**Correct answer(s):**

1. The step of estimating velocity is omitted  
2. No more than one story is expanded into a set of estimated tasks at a time

**4. Agile modeling**

The team chooses to integrate Agile Model-Driven Development – or AMDD – techniques in its development approach.

Question

Match each activity to the phase of the agile modeling development process in which it occurs.

**Options:**

1. Implementing executive specifications
2. Designing the user interface model
3. Generating ideas about a development-related issue
4. Estimating work for an iteration

**Targets:**

1. Test-driven development
2. Initial requirements envisioning
3. Model storming
4. Iteration modeling

Answer

*A test-driven development – or TDD – phase involves running tests that implement what has just been modeled, by applying executive specifications – such as customer tests or development tests.*

*The initial requirements envisioning phase should indicate how a system will function, how users will work with it, and an initial user interface model.*

*Model storming sessions are brainstorming sessions in which a few team members briefly explore an issue. They may be planned or occur spontaneously as needed.*

*The iteration modeling phase is the phase in each iteration during which work is estimated.*

**Correct answer(s):**

Target 1 = Option A

Target 2 = Option B

Target 3 = Option C

Target 4 = Option D

Question

What occurs during the iteration modeling phase?

**Options:**

1. More detailed models are created for an iteration than in previous phases
2. A few people spontaneously gather around a whiteboard to discuss an issue and generate ideas
3. The team attempts to create an infrastructure that is likely to succeed while being as simple as possible
4. The team applies executive specifications such as customer or development tests

Answer

***Option 1:****Correct. In the iteration modeling phase, you create more detailed models that are specific to a given iteration.*

***Option 2:****Incorrect. This is a description of model storming. In iteration modeling, you create detailed models that are specific to a given iteration.*

***Option 3:****Incorrect. This is a description of initial architecture envisioning. In iteration modeling, you create detailed models that are specific to a given iteration.*

***Option 4:****Incorrect. This is a description of test-driven development, or TDD. In iteration modeling, you create detailed models that are specific to a given iteration.*

**Correct answer(s):**

1. More detailed models are created for an iteration than in previous phases

Activities included in agile release and iteration planning, and in agile modeling, have been identified.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_agsd_a02_it_enus/output/html/course_transcript.html#top)

© 2016 Skillsoft Ireland Limited