**Intel Interview questions and answers:**

**Difference between ArrayList and LinkedList**

ArrayList and LinkedList both implements List interface and maintains insertion order. Both are non synchronized classes.

|  |  |
| --- | --- |
| ArrayList | LinkedList |
| 1) ArrayList internally uses dynamic array to store the elements. | LinkedList internally uses doubly linked list to store the elements. |
| 2) Manipulation with ArrayList is slow because it internally uses array. If any element is removed from the array, all the bits are shifted in memory. | Manipulation with LinkedList is faster than ArrayList because it uses doubly linked list so no bit shifting is required in memory. |
| 3) ArrayList class can act as a list only because it implements List only. | LinkedList class can act as a list and queue both because it implements List and Deque interfaces. |
| 4) ArrayList is better for storing and accessing data. | LinkedList is better for manipulating data. |

**There are several differences between HashMap and Hashtable in Java**

1. [Hashtable](http://java.sun.com/javase/7/docs/api/java/util/Hashtable.html) is synchronized, whereas [HashMap](http://java.sun.com/javase/7/docs/api/java/util/HashMap.html) is not. This makes HashMap better for non-threaded applications, as unsynchronized Objects typically perform better than synchronized ones.
2. Hashtable does not allow null keys or values. HashMap allows one null key and any number of null values.
3. One of HashMap's subclasses is [LinkedHashMap](http://java.sun.com/javase/7/docs/api/java/util/LinkedHashMap.html), so in the event that you'd want predictable iteration order (which is insertion order by default), you could easily swap out the HashMap for a LinkedHashMap. This wouldn't be as easy if you were using Hashtable.

Since synchronization is not an issue for you, I'd recommend HashMap. If synchronization becomes an issue, you may also look at [ConcurrentHashMap](http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ConcurrentHashMap.html).

**Data Structure used for a directory Structure:**

Every record in the database has two fields: ID and ParentID. IDs are 4-5 characters (Base36, a-z:0-9 or something similar). Parent IDs are a concatenation of the parent's complete structure...

So...

This structure:

Root

Folder1

Folder2

Folder3

Folder4

Folder5

Folder6

Would be represented like this:

ID ParentID Name

0000 NULL ROOT

0001 0000 Folder1

0002 0000 Folder2

0003 00000002 Folder3

0004 0000 Folder4

0005 00000004 Folder5

0006 000000040005 Folder6

I like this structure because if I need to find all the files under a folder I can do a query like:

SELECT \* FROM Folders WHERE ParentID LIKE '0000%' -- to find all folders under Folder1

To delete a folder and all its children:

DELETE FROM Folders WHERE ID='0004' AND ParentID LIKE '00000004%'

To move a folder and its children, you have to update all the records that use the same parent, to the new parent.

And I don't want to linit the folders or the subfolders levels

An obvious limitation to this is that the number of subfolders are limited to the size of your ParentID field.

All file systems are different, so there are a huge number of data structures that actually get used in file systems.

Many file systems use some sort of [bit vector](http://en.wikipedia.org/wiki/Bitvector) (usually referred to as a bitmap) to track where certain free blocks are, since they have excellent performance for querying whether a specific block of disk is in use and (for disks that aren't overwhelmingly full) support reasonably fast lookups of free blocks.

Many older file systems (ext and ext2) stored directory structures using simple linked lists. Apparently this was actually fast enough for most applications, though some types of applications that used lots of large directories suffered noticeable performance hits.

The XFS file system was famous for using [B+-trees](http://en.wikipedia.org/wiki/B+_tree) for just about everything, including directory structures and its journaling system. From what I remember from my undergrad OS course, the philosophy was that since it took so long to write, debug, and performance tune the implementation of the B+-tree, it made sense to use it as much as possible.

Other file systems (ext3 and ext4) use a variant of the B-tree called the [HTree](http://ext2.sourceforge.net/2005-ols/paper-html/node3.html) that I'm not very familiar with. Apparently it uses some sort of hashing scheme to keep the branching factor high so that very few disk accesses are used.

**How to ensure that program works across various compilers without any errors**

pre-processor directives

## Definition - What does *Preprocessor Directive* mean?

Preprocessor directives are lines included in a program that being with the character #, which make them different from a typical source code text. They are invoked by the compiler to process some programs before compilation. Preprocessor directives change the text of the source code and the result is a new source code without these directives.  
  
Although preprocessing in C# is conceptually similar to that in C/C++, it is different in two aspects. First, preprocessing in C# does not involve a separate step for preprocessor execution before compilation. It is processed as a part of the lexical analysis phase. Second, it cannot be used to create macros. In addition, the new directives #region and #unregion have been added in C# along with the exclusion of some directives used earlier (#include is a notable directive whose use is replaced with "using" to include assemblies).   
  
Java does not support preprocessor directives.

**In languages such as C++ and Java, what is the shorthand for an if statement?**

In [computer programming](https://en.wikipedia.org/wiki/Computer_programming), ?: is a [ternary operator](https://en.wikipedia.org/wiki/Ternary_operator) that is part of the syntax for a basic [conditional expression](https://en.wikipedia.org/wiki/Conditional_(programming)) in several [programming languages](https://en.wikipedia.org/wiki/Programming_language). It is commonly referred to as the **conditional operator, inline if (iif), or ternary if.**

It originally comes from [CPL](https://en.wikipedia.org/wiki/CPL_(programming_language)), in which equivalent syntax for *e*1 ? *e*2 : *e*3 was *e*1 → *e*2, *e*3.[[1]](https://en.wikipedia.org/wiki/%3F:#cite_note-1)[[2]](https://en.wikipedia.org/wiki/%3F:#cite_note-2)

**When a derived class is instantiated, which constructor is called first that of the parent or child?**

So when we instantiate an instance of Derived, first the Base portion of Derived is constructed (and the Base constructor is called). Once the Base portion is finished, the Derived portion is constructed (and the Derived constructor is called). At this point, there are no more derived classes, so we are done.

**What is database normalization? When is normalization not desirable?**

Database normalization, or simply normalization, is the process of organizing the [columns](https://en.wikipedia.org/wiki/Column_(database)) (attributes) and [tables](https://en.wikipedia.org/wiki/Table_(database)) (relations) of a [relational database](https://en.wikipedia.org/wiki/Relational_database) to reduce [data redundancy](https://en.wikipedia.org/wiki/Data_redundancy) and improve data integrity.

Normalization involves arranging attributes in tables based on [dependencies](https://en.wikipedia.org/wiki/Dependency_theory_(database_theory)) between attributes, ensuring that the dependencies are properly enforced by database integrity constraints. Normalization is accomplished through applying some formal rules either by a process of synthesis or decomposition. Synthesis creates a normalized database design based on a known set of dependencies. Decomposition takes an existing (insufficiently normalized) database design and improves it based on the known set of dependencies.

**Database normalization is unnecessary in situations where we are storing immutable data such as financial transactions or a particular day's price list.**

#### When Multiple Joins are Needed to Produce a Commonly Accessed View

The biggest problem with normalization is that you end up with multiple tables representing what is conceptually a single item. For example, consider this normalized set of tables which represent a user profile on a typical social networking site.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| user table | | | | | | | | |
|  |  |  |  |  |  |  |  |  |
| user\_id | first\_name | last\_name | sex | hometown | relationship\_status | interested\_in | religious\_views | political\_views |
| 12345 | John | Doe | Male | Atlanta, GA | married | women | (null) | (null) |

|  |  |  |
| --- | --- | --- |
| user\_affiliations table | | |
|  |  |  |
| user\_id (foreign\_key) | affiliation\_id (foreign key) |  |
| 12345 | 42 |  |
| 12345 | 598 |  |

|  |  |  |
| --- | --- | --- |
| affiliations table | | |
|  |  |  |
| affiliation\_id | description | member\_count |
| 42 | Microsoft | 18,656 |
| 598 | Georgia Tech | 23,488 |

|  |  |  |
| --- | --- | --- |
| user\_phone\_numbers table | | |
|  |  |  |
| user\_id (foreign\_key) | phone\_number | phone\_type |
| 12345 | 425-555-1203 | Home |
| 12345 | 425-555-6161 | Work |
| 12345 | 206-555-0932 | Cell |

|  |  |  |
| --- | --- | --- |
| user\_screen\_names table | | |
|  |  |  |
| user\_id (foreign\_key) | screen\_name | im\_service |
| 12345 | geeknproud@example.com | AIM |
| 12345 | voip4life@example.org | Skype |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| user\_work\_history table | | |
|  |  |  |
| user\_id (foreign\_key) | company\_affiliation\_id (foreign key) | company\_name | job\_title |
| 12345 | 42 | Microsoft | Program Manager |
| 12345 | 78 | i2 Technologies | Quality Assurance Engineer |

This is the kind of information you see on the average profile on [Facebook](http://www.facebook.com/). With the above design, it takes **six** [SQL Join](http://en.wikipedia.org/wiki/Join_%28SQL%29) operations to access and display the information about a single user. This makes rendering the profile page a fairly database intensive operation which is compounded by the fact that profile pages are the most popular pages on social networking sites.

The simplest way to fix this problem is to denormalize the database. Instead of having tables for the user’s affiliations, phone numbers, IM addresses and so on, we can just place them in the user table as columns. The drawback with this approach is that there is now more wasted space (e.g. lots of college students people will have null for their work\_phone)  and perhaps some redundant information (e.g. if we copy over the description of each affiliation into an affiliation\_name column for each user to prevent having to do a join with the affiliations table).

However given the very low costs of storage versus the improved performance characteristics of querying a single table and not having to deal with SQL statements that operate across six tables for every operation. This is a small price to pay.

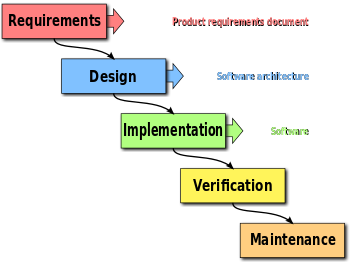
As Joe Gregorio mentions in his blog post about [the emergence of megadata](http://bitworking.org/news/158/ETech-07-Summary-Part-2-MegaData), a lot of the large Web companies such as Google, eBay and Amazon are heavily into denormalizing their databases as well as eschewing transactions when updating these databases to improve their scalability.

UPDATE: Someone pointed out in the comments that denormalizing the affiliations table into user's table would mean the member\_count would have to updated in thousands of user's rows when a new member was added to the group. This is obviously not the intent of denormalization for performance reasons since it replaces a bad problem with a worse one. Since an affiliation is a distinct concept from a user, it makes sense for it to have it's own table. Replicating the names of the groups a user is affiliated with in the user table is a good performance optimization although it does mean that the name has to be fixed up in thousands of tables if it ever changes. Since this is likely to happen very rarely, this is probably acceptable especially if we schedule renames to be done by a [cron job](http://en.wikipedia.org/wiki/Cron) during offpeak ours On the other hand, replicating the member count is just asking for trouble.

Explain a full-outer join, left/right join.

Explain the singleton design pattern/ the factory method design pattern.

Explain the waterfall development methodology/ Agile development methodology.

[](https://en.wikipedia.org/wiki/File:Waterfall_model.svg)

The unmodified "waterfall model". Progress flows from the top to the bottom, like a cascading waterfall.

The waterfall model is a [sequential](https://en.wikipedia.org/wiki/Sequence) (non-iterative) [design](https://en.wikipedia.org/wiki/Design) process, used in [software development processes](https://en.wikipedia.org/wiki/Software_development_process), in which progress is seen as flowing steadily downwards (like a [waterfall](https://en.wikipedia.org/wiki/Waterfall)) through the phases of conception, initiation, [analysis](https://en.wikipedia.org/wiki/Analysis), [design](https://en.wikipedia.org/wiki/Software_design), construction, [testing](https://en.wikipedia.org/wiki/Software_testing), [production/implementation](https://en.wikipedia.org/wiki/Implementation) and [maintenance](https://en.wikipedia.org/wiki/Software_maintenance). Despite the development of new software development process models, the Waterfall method is still the dominant process model with over a third of software developers still using it.[[1]](https://en.wikipedia.org/wiki/Waterfall_model#cite_note-1)

The waterfall development model originates in the [manufacturing](https://en.wikipedia.org/wiki/Manufacturing) and [construction](https://en.wikipedia.org/wiki/Construction) industries: highly structured physical environments in which after-the-fact changes are prohibitively costly, if not impossible. Because no formal software development methodologies existed at the time, this hardware-oriented model was simply adapted for software development.[[2]](https://en.wikipedia.org/wiki/Waterfall_model#cite_note-benington-2)

One of the differences between agile and waterfall is the approach to quality and testing. In the [waterfall model](https://en.wikipedia.org/wiki/Waterfall_model), there is always a separate *testing phase* after a *build phase*; however, in agile development testing is completed in the same iteration as programming.

Because testing is done in every iteration—which develops a small piece of the software—users can frequently use those new pieces of software and validate the value.

After the users know the real value of the updated piece of software, they can make better decisions about the software's future. Having a value retrospective and software re-planning session in each iteration—[Scrum](https://en.wikipedia.org/wiki/Scrum_(software_development)) typically has iterations of just two weeks—helps the team continuously adapt its plans so as to maximize the value it delivers

#### Agile principles[[edit](https://en.wikipedia.org/w/index.php?title=Agile_software_development&action=edit&section=3)]

The Agile Manifesto is based on twelve principles:[[13]](https://en.wikipedia.org/wiki/Agile_software_development#cite_note-ManifestoPrinciples-13)

1. Customer satisfaction by early and continuous delivery of valuable software
2. Welcome changing requirements, even in late development
3. Working software is delivered frequently (weeks rather than months)
4. Close, daily cooperation between business people and developers
5. Projects are built around motivated individuals, who should be trusted
6. Face-to-face conversation is the best form of communication (co-location)
7. Working software is the principal measure of progress
8. Sustainable development, able to maintain a constant pace
9. Continuous attention to technical excellence and good design
10. Simplicity—the art of maximizing the amount of work not done—is essential
11. Best architectures, requirements, and designs emerge from self-organizing teams
12. Regularly, the team reflects on how to become more effective, and adjusts accordingly

Agile software development describes a set of principles for [software development](https://en.wikipedia.org/wiki/Software_development) under which requirements and solutions evolve through the collaborative effort of self-organizing [cross-functional teams](https://en.wikipedia.org/wiki/Cross-functional_team).[[1]](https://en.wikipedia.org/wiki/Agile_software_development#cite_note-Collier_2011-1) It advocates adaptive planning, evolutionary development, early delivery, and continuous improvement, and it encourages rapid and flexible response to change.[[2]](https://en.wikipedia.org/wiki/Agile_software_development#cite_note-2) These principles support the definition and continuing evolution of many [software development methods](https://en.wikipedia.org/wiki/Software_development_methodologies).[[3]](https://en.wikipedia.org/wiki/Agile_software_development#cite_note-LarmanGuide-3)

**Agile development model** is also a type of [**Incremental model**](http://istqbexamcertification.com/what-is-incremental-model-advantages-disadvantages-and-when-to-use-it/). Software is developed in incremental, rapid cycles. This results in small incremental releases with each release building on previous functionality. Each release is thoroughly [**tested**](http://istqbexamcertification.com/why-is-testing-necessary/) to ensure [**software quality**](http://istqbexamcertification.com/what-is-software-quality/) is maintained. It is used for time critical applications.  Extreme Programming (XP) is currently one of the most well known agile [**development life cycle model**](http://istqbexamcertification.com/what-are-the-software-development-models/).

**Diagram of Agile model:**



**Advantages of Agile model:**

* Customer satisfaction by rapid, continuous delivery of useful software.
* People and interactions are emphasized rather than process and tools. Customers, developers and testers constantly interact with each other.
* Working software is delivered frequently (weeks rather than months).
* Face-to-face conversation is the best form of communication.
* Close, daily cooperation between business people and developers.
* Continuous attention to technical excellence and good design.
* Regular adaptation to changing circumstances.
* Even late changes in requirements are welcomed

**Disadvantages of Agile model:**

* In case of some software deliverables, especially the large ones**, it is difficult to assess the effort required at the beginning of the software development life cycle.**
* **There is lack of emphasis on necessary designing and documentation.**
* **The project can easily get taken off track if the customer representative is not clear what final outcome that they want.**
* **Only senior programmers are capable of taking the kind of decisions required during the development process. Hence it has no place for newbie programmers, unless combined with experienced resources.**

**When to use Agile model:**

* **When new changes are needed to be implemented.** The freedom agile gives to change is very important. New changes can be implemented at very little cost because of the frequency of new increments that are produced.
* To implement a new feature the developers need to lose only the work of a few days, or even only hours, to roll back and implement it.
* Unlike the [**waterfall model**](http://istqbexamcertification.com/what-is-waterfall-model-advantages-disadvantages-and-when-to-use-it/) in agile model very limited [**planning**](http://istqbexamcertification.com/what-is-the-purpose-and-importance-of-test-plans/) is required to get started with the project. **Agile assumes that the end users’ needs are ever changing in a dynamic business and IT world.** Changes can be discussed and features can be newly effected or removed based on feedback. This effectively gives the customer the finished system they want or need.
* Both system developers and stakeholders alike, find they also get more freedom of time and options than if the software was developed in a more rigid sequential way. Having options gives them the ability to leave important decisions until more or better data or even entire hosting programs are available; meaning the project can continue to move forward without fear of reaching a sudden standstill.

Here’s my list of potential disadvantages with agile:

* [**Active user involvement**](http://www.allaboutagile.com/2007/02/principle-1-active-user-involvement-is.html)**and**[**close collaboration**](http://www.allaboutagile.com/2007/04/agile-principle-10-no-place-for-snipers.html)**are required throughout the development cycle. This is very engaging, rewarding and ensures delivery of the right product.** It’s the fundamental principle in agile that ensures expectations are well managed. And since the definition of failure is not meeting expectations, these are critical success factors for any project. However these principles are very demanding on the user representative’s time and require a big commitment for the duration of the project.
* [**Requirements emerge and evolve**](http://www.allaboutagile.com/2007/03/agile-principle-3-time-waits-for-no-man.html)**throughout the development. This creates the very meaning of agile – flexibility. Flexibility to change course as needed and to ensure delivery of the right product. There are two big flip sides to this principle though. One is the potential for scope creep, which we all know can create the risk of ever-lasting projects.** The other is that there is much less predictability, at the start of the project and during, about what the project is actually going to deliver. This can make it harder to define a business case for the project, and harder to negotiate fixed price projects. Without the maturity of a strong and clear vision, and the discipline of fixing timescales and trading scope, this is potentially very dangerous.
* [Agile requirements are barely sufficient](http://www.allaboutagile.com/2007/03/agile-requirements-just-in-time-and.html). This eliminates wasted effort on deliverables that don’t last (i.e. aren’t part of the finished product), which saves time and therefore money. Requirements are clarified just in time for development and can be documented in much less detail due to the timeliness of conversations. However this can mean less information available to new starters in the team about features and how they should work. It can also create potential misunderstandings if the teamwork and communication aren’t at their best, and difficulties for team members (especially testers) that are used to everything being defined up front. The belief in agile is that it’s quicker to refactor the product along the way than to try to define everything completely up front, which arguably is impossible. And this risk is managed closely through the [incremental approach](http://www.allaboutagile.com/2007/03/agile-principle-5-how-dyou-eat-elephant.html) to development and [frequent delivery of product](http://www.allaboutagile.com/2007/03/agile-principle-6-focus-on-frequent.html).
* [**Testing is integrated throughout the lifecycle**](http://www.allaboutagile.com/2007/04/agile-development-agile-testing-is-not.html)**. This helps to ensure quality throughout the project without the need for a lengthy and unpredictable test phase at the end of the project.** However it does imply that testers are needed throughout the project and this effectively increases the cost of resources on the project. This does have the effect of reducing some very significant risks, that have proven through research to cause many [projects to fail](http://www.allaboutagile.com/2007/08/why-most-it-projects-fail-and-how-agile.html). The cost of a long and unnpredictable test phase can, in my experience of waterfall, cause huge unexpected costs when a project over-runs. However there is an additional cost to the project to adopt continuous testing throughout.
* [**Frequent delivery**](http://www.allaboutagile.com/2007/03/agile-principle-6-focus-on-frequent.html)**of product and the need to sign off**[**each feature as *done***](http://www.allaboutagile.com/2007/04/agile-principle-7-done-means-done.html)**before moving on to the next makes UAT (user acceptance testing) continuous and therefore potentially quite onerous.** The users or product owner needs to be ready and available for prompt testing of the features as they are delivered and throughout the entire duration of the project. This can be quite time-consuming but helps drastically to ensure a quality product that meets user expectations.
* Finally, common feedback is that agile development is rather intense for developers. The need to really *complete*each feature 100% within each iteration, and the relentlessness of iterations, can be mentally quite tiring so it’s important to find a sustainable pace for the team.

I believe these trade-offs are well worthwhile. Software is complex. People are complex. And the only thing that’s certain in projects is change. This lethal combination of unpredictability is more often than not helped by agile principles. So, in my view, for many project situations, the advantages of agile development far outweigh the disadvantages.

# Advantages of Agile

## **Customer Satisfaction by Rapid, Continuous Delivery of Useful Software**

Your customers and users will be satisfied because you are continuously delivery value to them with usable software.

This is a stark contrast compared to that of a traditional waterfall product delivery, that if your customers are used to waterfall, they may find it strange adjusting to having working software sooner.

The big downside of waterfall is that you deliver large pieces of functionality towards the end of the project life-cycle. This means that all throughout the development stages of waterfall, your project is incurring costs with no return on investment.

By delivering working pieces of functionality sooner and more regularly you are giving your users an opportunity to get a return on their investment sooner. Sure, they may not have all the functionality they need upfront, but they can start to make use of the solution to make their lives easier and start realising the benefits sooner.

## People and Interactions are Emphasized Rather Than Process and Tools

Agile is focused very heavily around people and the interactions between people, rather than process and tools. This is a core value of the agile manifesto.

The reason this is important is because it is the input from your team and customers that will ultimately make your project a success as opposed to what tools you use. Continual collaboration throughout the entire development cycle of your project enables everyone involved to build up a good working relationship that will be based on trust. This trust based working relationship is crucial when building software incrementally.

## Continuous Attention to Technical Excellence and Good Design

When working agile your work in shorter iterations and only build what is necessary to satisfy the requirements for that iteration and nothing else. This forces you to keep you designs simple which is very important as simplicity helps you design testable and therefore more reliable systems.

Developers understand, and choose from, many possible technical ways to satisfy business need and these are choices that reflect a craft that balances design, use, and support. They provide the technical underpinnings that enable us always to move forward at a steady pace, and keep code quality consistently high. They do this using principles of truly simple design, combined with a grasp of technical debt and the means to keep it under control. Developers like to use the best techniques for keeping the design under control without excessive work or rework.

Some of these techniques include :

* **Refactoring** : Refactoring is the process of improving the design of existing code without changing its externally observable behavior. In order to make big changes to the structure of the code, refactoring uses a quick succession of small, well-known steps that can each be verified as safe (functionally equivalent). Refactoring is most often done in the context of Test-Driven Development/Design where extensive tests and simple design make it easy to refactor safely.
* **Simple Design** : Keeping your design simple and not repeating code helps you keep your code maintainable. If you design your code to be modular and interface driven then you can reduce coupling between objects which leads to more robust systems.
* **Test Driven Development** : Test-Driven Development is a way of driving the design of code by writing a test which expresses what you intend the code to do, making that test pass, and continuously refactoring to keep the design as simple as possible. TDD can apply at multiple levels, e.g. Customer Tests, Integration Tests, Unit Tests.

Test-Driven Development/Design follows a rigorous cycle. Start by writing a failing test. Implement the simplest solution that will cause the test to pass. Then search for duplication and remove it. This is often called RED-GREEN-REFACTOR has become almost a mantra for many test driven design practitioners. Understanding and even internalizing this cycle is key to being able to use test driven design to solve problems.

## Regular Adaptation to Changing Circumstances

Your customer or business sponsor may change their mind about what is being built. This may be because you have given them new ideas from the software you delivered in a previous iteration. It may be because the companies priorities have changed or a new regulatory change comes into force. They key thing here is that you should embrace it. Yes, some code may get thrown away and some time has been lost, but if you are working in short iterations then this lost time is minimized Change is a reality of software development, a reality that your software process must reflect.

Change can be scary at first, for clients and partners alike, but when both sides are prepared to take the leap, it can be mutually rewarding. It’s fair to say that Agile is a journey and Web Technology Group is working hard to offer clients in the private and public sectors ways to adopt Agile practices.

* **Get used to prioritising your Product backlog or Work Items** : Clients need to be heavily focused from the outset on prioritising the Product Backlog in order to get the most ‘bang for their buck’. To keep delivery focused and optimise the output, the Product Manager needs to be able to have some ‘tough conversations’ about what features are must-haves and when they need to be delivered.
* **Make sure you’re ‘bought-in’ to the Agile process** : Your team and customers need to be bought into the agile process. The reality of Agile is that once the sprints or iterations are complete, the delivery can be less – or more, of course than what was expected for the budget. Friction can occur if the budget is spent and further sprints are needed. Agile is about clear, frequent and open communication.
* **Encourage frequent, open communication** : It can’t be overstated, but regular engagement needs to take place between the development team and customer or users. At the end of each sprint or iteration, there will be a formal opportunity to inspect the increment, ensuring that the delivered software is consistent with requirements. Through the whole process, though, discussions need to be open and realistic.
* **Value cross-functional team members** : Your development team should be self-organising and cross-functional. Both of these characteristics are key strengths for Agile delivery. In some ways Agile is a simple idea but the reality is that it can mean different things to different people – especially depending on their role in the software development process. One of the key things, though, is to be open to change. Not just in order to move from traditional ways of organising projects but to adapt your use of Agile itself.

# Disadvantages of Agile

## Difficult to Assess the Effort Required at the Beginning of the Software Development Life Cycle

A complaint I have often heard from business leaders and project managers alike is that compared to waterfall it is hard to quantify the total effort and cost to deliver a project. On one hand I can see why they would think this, especially when they come from a regimented waterfall process world.

It is indeed harder to fully quantify how long the total project will take, but the mitigation for this is that features product will be delivered incrementally by giving the users the most valuable requirements first.

## Can be Very Demanding on the Users Time

Active user involvement and close collaboration are required throughout the development cycle. This is very engaging, rewarding and ensures delivery of the right product. It’s the fundamental principle in agile that ensures expectations are well managed. And since the definition of failure is not meeting expectations, these are critical success factors for any project. However these principles are very demanding on the user representative’s time and require a big commitment for the duration of the project.

## Potential for scope creep

Requirements emerge and evolve throughout the development. This creates the very meaning of agile – flexibility. Flexibility to change course as needed and to ensure delivery of the right product. There are two big flip sides to this principle though. One is the potential for scope creep, which we all know can create the risk of ever-lasting projects. The other is that there is much less predictability, at the start of the project and during, about what the project is actually going to deliver. This can make it harder to define a business case for the project, and harder to negotiate fixed price projects. Without the maturity of a strong and clear vision, and the discipline of fixing time-scales and trading scope, this is potentially very dangerous.

## Harder for new Starters to Integrate in the Team

Agile requirements are normally barely sufficient. Requirements are clarified just in time for development and can be documented in much less detail due to the timeliness of conversations. This can mean less information available to new starters in the team about features and how they should work.

It can also create potential misunderstandings if the teamwork and communication aren’t at their best, and difficulties for team members (especially testers) that are used to everything being defined up front.

## Costs can Increase as Testers Required all the Time Instead of at the End

Testing is integrated throughout the lifecycle. This helps to ensure quality throughout the project without the need for a lengthy and unpredictable test phase at the end of the project.

However it does imply that testers are needed throughout the project and this effectively increases the cost of resources on the project. This does have the effect of reducing some very significant risks, that can cause many projects to fail. The cost of a long and unpredictable test phase can, in my experience of waterfall, cause huge unexpected costs when a project over-runs. However there is an additional cost to the project to adopt continuous testing throughout.

## Agile can be Intense for the Team

Finally, common feedback is that agile development is rather intense for developers. They need to really complete each feature 100% within each sprint or iteration, and the relentlessness of iterations, can be mentally quite tiring so it’s important to find a sustainable pace for the team.

**The Disadvantages**

**No Fixed Pricing:**Software product pricing is not fixed. In other words, clients cannot know how much the software product will really cost them until the project starts and even then, they just get an estimate.

Agile Methodologies require that clients participate proactively and as a project requirement. Resource availability (on the client side) is sometimes scarce, putting a strain on the client company.

**The Lingo:**Agile Methodologies require that the client company use Agile terminology such as “[scrum](http://www.nearshoreamericas.com/small-software-firms-agile-development/),” “sprints” and “points,” etc. This is often confusing and not perceived as necessary.

Working with various clients can be challenging from a software development company perspective. All clients have different backgrounds, products, teams and [expectations](http://www.nearshoreamericas.com/small-software-firms-agile-development/), etc. The learning curve is constant. What may be obvious to one client is not obvious to the next, yet all clients are what they are and the software product has to be developed regardless of what the party involved may be like.

**The Advantages**

**Transparency:**Agile methodologies require transparency, making a project less cumbersome from a communication point of view. Agile methodologies require software development companies to explicitly communicate functionality costs and development times.

**Collaborative Effort:**Clients are obligated to actively participate in the requirements definition process, thus reducing unknowns and making the overall process more efficient.  And the method requires software development companies to basically “rent out” a software team at a fixed amount until the client chooses to end the project. This pay-as-you-go system makes project finances very easy to manage from a revenue and expense point of view.

Top of Form

Bottom of Form

**The Disadvantages**

**Complicated Sales Cycle:**Clients like fixed amount contracts. This is true in software development as in other types of services. When bidding on a project, clients like to know “how much is this going to cost me more or less”. Selling software with Agile Methodologies makes answering this question very difficult. First of all, it can never be answered until iteration ends and frankly clients don’t like hearing this.

**Training the Client:**Clients do have to go through a training process to be successful in the product development. The lack of client [participation](http://www.nearshoreamericas.com/small-software-firms-agile-development/) and understanding will affect the software quality and therefore reflect on the software development company.

Small projects and maintenance projects are a big part of software development services, yet Agile Methodologies are not ideal for these kinds of projects. This forces software development companies to have hybrid solutions.

**Explain the singleton design pattern/ the factory method design pattern.**

Asked me about the projects in my resume, basic Linux questions and then the interviewer went on to talk about chips and how they were fabricated. It turned out that the position that they interviewed me for was not entirely related to Software.

**Interview Questions**

* If you could improve on one of your projects, what would you improve?

Write a function in python that will return true if a passed in string has a capital letter as its first character.

 Technical questions about C++ template, multithreading, python code, git and unix commands and more questions about the latest job experiences. There were a lot of behavior questions too.Show Less

**Interview Questions**

* Explain C++ template and multithreading and examples? How does git store data and manage access control? Fast-forward merging in Git? Unix command to find patterns in directory hierarchy.

video color format, allocate aligned memory. basic c++ questions.

Do you know the technology that our current team is working on

Had an initial Hackerrank challenge and problems were on Linked Lists, bit manipulation and few questions on OS and in the interview was asked details about my projects and experience.

**What's the difference between C and Java in terms of compilation?**

The C compiler generates platform dependent code .Therefore the generated program runs only on the architecture (Intel,Motorola,etc.) and OS(Windows,Linux,AIX)it was compiled for.Therefore it can be as fast as it can be.

Due to platform independence ,a Java compiler will interpret Java source code into Java Byte code and pass to the JVM ,which will pass machine understandable code through to cpu .

Wanted to know what I would do if I had been asked to do too many projects at once, and how I would deal with the situation.

what is the difference between a process and a thread?

## Differences Between C and Java

If you are a C or C++ programmer, you should have found much of the syntax of Java--particularly at the level of operators and statements--to be familiar. Because Java and C are so similar in some ways, it is important for C and C++ programmers to understand where the similarities end. There are a number of important differences between C and Java, which are summarized in the following list:

**No preprocessor**

Java does not include a preprocessor and does not define any analogs of the #define, #include, and #ifdef directives. Constant definitions are replaced with staticfinal fields in Java. (See the java.lang.Math.PI field for an example.) Macro definitions are not available in Java, but advanced compiler technology and inlining has made them less useful. Java does not require an #include directive because Java has no header files. Java class files contain both the class API and the class implementation, and the compiler reads API information from class files as necessary. Java lacks any form of conditional compilation, but its cross-platform portability means that this feature is very rarely needed.

**No global variables**

Java defines a very clean namespace. Packages contain classes, classes contain fields and methods, and methods contain local variables. But there are no global variables in Java, and, thus, there is no possibility of namespace collisions among those variables.

**Well-defined primitive type sizes**

All the primitive types in Java have well-defined sizes. In C, the size of short, int, and long types is platform-dependent, which hampers portability.

**No pointers**

Java classes and arrays are reference types, and references to objects and arrays are akin to pointers in C. Unlike C pointers, however, references in Java are entirely opaque. There is no way to convert a reference to a primitive type, and a reference cannot be incremented or decremented. There is no address-of operator like &, dereference operator like \* or −>, or sizeof operator. Pointers are a notorious source of bugs. Eliminating them simplifies the language and makes Java programs more robust and secure.

**Garbage collection**

The Java Virtual Machine performs garbage collection so that Java programmers do not have to explicitly manage the memory used by all objects and arrays. This feature eliminates another entire category of common bugs and all but eliminates memory leaks from Java programs.

**No goto statement**

Java doesn't support a goto statement. Use of goto except in certain well-defined circumstances is regarded as poor programming practice. Java adds exception handling and labeled break and continue statements to the flow-control statements offered by C. These are a good substitute for goto.

**Variable declarations anywhere**

C requires local variable declarations to be made at the beginning of a method or block, while Java allows them anywhere in a method or block. Many programmers prefer to keep all their variable declarations grouped together at the top of a method, however.

**Forward references**

The Java compiler is smarter than the C compiler, in that it allows methods to be invoked before they are defined. This eliminates the need to declare functions in a header file before defining them in a program file, as is done in C.

**Method overloading**

Java programs can define multiple methods with the same name, as long as the methods have different parameter lists.

**No struct and union types**

Java doesn't support C struct and union types. A Java class can be thought of as an enhanced struct, however.

**No enumerated types**

Java doesn't support the enum keyword used in C to define types that consist of fixed sets of named values. This is surprising for a strongly typed language like Java, but there are ways to simulate this feature with object constants.

**No bitfields**

Java doesn't support the (infrequently used) ability of C to specify the number of individual bits occupied by fields of a struct.

**No typedef**

Java doesn't support the typedef keyword used in C to define aliases for type names. Java's lack of pointers makes its type-naming scheme simpler and more consistent than C's, however, so many of the common uses of typedefare not really necessary in Java.

**No method pointers**

C allows you to store the address of a function in a variable and pass this function pointer to other functions. You cannot do this with Java methods, but you can often achieve similar results by passing an object that implements a particular interface. Also, a Java method can be represented and invoked through a java.lang.reflect.Method object.

**No variable-length argument lists**

Java doesn't allow you to define methods such as C's printf() that take a variable number of arguments. Method overloading allows you to simulate C varargs functions for simple cases, but there's no general replacement for this feature.

**Implement a queue using two stacks.**

**Design ecommerce using oops**