# Software Development Life Cycles	start time:

Software development activities are grouped into 4 general categories:

analysis, **design**, **coding**, and **testing**

For small programs (e.g. homework), one person (or a small team) can do all of the work, and stop when the program is finished or they reach a deadline and run out of time. However, this approach does not scale well to larger systems and development teams.

This activity explores ways to organize these 4 categories into a **software development process** or **life cycle (SDLC)**. This parallels how an animal is born, matures, and dies. This is different from (but related to) a **software release life cycle**, which describes how each software release is created, matures, and becomes obsolete.

Before you start, complete the form below to assign a role to each member. If you have 3 people, combine Speaker & Reflector.

Team	**Date**
Hat Trick + 1	
Team Roles	**Team Member**
Recorder: records all answers & questions, and provides copies to team & facilitator.	Scott
Speaker: talks to facilitator and other teams.	Lewis

Manager: keeps track of time and makes sure everyone contributes appropriately.	Kyle
Reflector: considers how the team could work and learn more effectively.	Anastasia

Reminders:

1. *Note the time whenever your team starts a new section or question.*
2. *Write legibly & neatly so that everyone can read & understand your responses.*

A. Finding & Fixing Errors	

1. Estimate how long (seconds, minutes, hours, days, weeks, months, or years) it typically takes to correct an error in software when it is found by:

a.	a **compiler**, seconds after the file was edited	seconds
b.	a **compiler**, later the same day or during a nightly build	hours
c.	a **pair programming** partner, seconds after the error was made	seconds
d.	a **code review**, days or weeks after the file was edited	days
e.	a **customer or other user**, months after the software is released	months
f.	a **unit test**, minutes after the file was edited	minutes
g.	a **unit test**, later the same day or during a nightly build	hours
h.	a **system test** shortly before software is released, weeks or months after the file was edited	weeks

2. Describe (or sketch a graph) of the relationship between the time to **find an error** and the time and cost to **repair the error**.

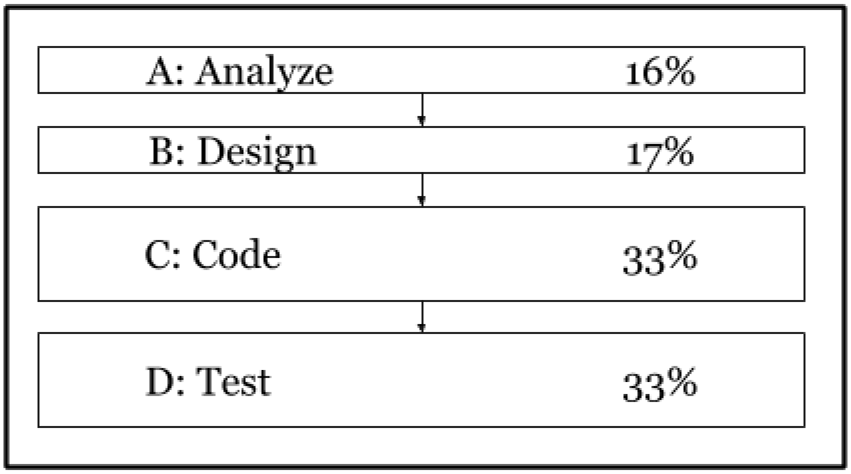
The longer it takes to find the error, the longer and more costly it takes to fix the error.

3. Explain why we should design SDLCs to find & fix errors as quickly as possible.

Reduction of risk, functionality can be added later in cycle.

The next sections will explore some different SDLCs to help you understand the advantages, disadvantages, and tradeoffs.

B. Waterfall Model



1. The diagram above shows a **Waterfall Model**.

a.	How many stages are shown?	4
b.	Which stage is 1st?	Analyze
c.	Which stage(s) must be finished before **coding** starts?	Analyze and Design

2. The diagram also shows the typical **percentage** of **total cost & effort** for each stage, although these percentages vary widely by project. Answer each question with one of:
0% 25% (¼) 33% (⅓) 50% (½) 67% (⅔) 75% (¾) 100%

a.	What % of **total effort** is in:	the **last stage**?	33.00%

b.		the **first 2 stages**?	33.00%
c.	When the project is **25%** done:	what % of **analysis** is done?	100.00%
d.		what % of **coding** is done?	0.00%
e.	When the project is **50%** done:	what % of **coding** is done?	25.00%
f.		what % of **testing** is done?	0.00%

3. It is important to find and then fix errors in the software.

a.	If **coding** errors are found during **C:Code**, in which stage should they be fixed?	Coding Stage
b.	If **coding** errors are found during **D:Test**, in which stage should they be fixed?	Testing Stage
c.	If **analysis** errors are found during **B:Design**, in which stage should they be fixed?	Design Stage
d.	If **analysis** errors are found during **D:Test**, in which stage should they be fixed?	Test Stage
e.	Which stage focuses most on **finding** errors?	Testing
f.	Are major errors in analysis and design more likely when the project is **similar** to past projects, or **different**?	More error when its different

4. Explain why later stages often take more time, effort, & money than expected.

More error are found in the later stages, that is why they take longer.

C. Iterative Model	

1. The diagram above shows an **Iterative Model**. In this activity, assume that the **total cost & effort** is the same for each model - they differ only in how they are organized.

a.	How many **stages** are shown?	12
b.	Which stage is 7th?	Coding 2
c.	How many stages involve **design**?	3

2. The diagram shows the **percentage of cost & effort**. Answer each question with one of:

0% 17% (⅙) 25% (¼) 33% (⅓) 50% (½) 67% (⅔) 75% (¾) 100%

a.	What % of **total effort**:	is for the **first 4 stages**?	33.00%
b.		is for **testing**?	33.00%
c.		is for **analysis & design**?	33.00%
d.	When the project is **25% done**:	what % of **analysis** is done?	33.00%
e.		what % of **coding** is done?	33.00%
f.		what % of **testing** is done?	17.00%
g.	When the project is **50% done**:	what % of **design** is done?	67.00%
h.		what % of **coding** is done?	50.00%
i.		what % of **testing** is done?	33.00%

3. It is important to find and then fix errors in the software.

a.	If **analysis** errors are found during **A1:Analyze**, in which stage could they first be fixed?	A1
b.	If **analysis** errors are found during **B1:Design**, in which **analysis** stage could they be fixed?	A2
c.	If **coding** errors are found during **D2:Test**, in which **coding** stage could they be fixed?	C3
d.	If **analysis** errors are found after **B2:Design**, in which **analysis** stage could they be fixed?	A3
e.	Are **analysis errors** likely to cause **design errors**?	Yes it is likely
f.	Are **design errors** likely to cause **coding errors**?	Yes
g.	Is it better to have **one try** or **several tries** to remove all errors and make the project perfect?	Several tries

4. Explain why each **test** stage should try to find as many errors as possible from the prior **code** stage.

It is better to find as many error in the first test stage so that there are less error to fix later and to speed up the cycle

5. Explain why **Iterative** is less likely than **Waterfall** to run into problems late in the project.

You are less likely to run into problem late in the project because you would have run the testing 3 times and coding 3 times where as in waterfall you only do it once.

D. Agile Model	

1. The diagram above shows an **Agile Model**. Again, assume that models have the same total cost & effort, and differ only in how they are organized.

a.	How many **stages** are shown?	24
b.	Which stage is 9th?	A3
c.	How many stages involve **design**?	6

2. The diagram also shows the **percentage of cost & effort**. Answer each question with one of: *0%, 17% (⅙), 25% (¼), 33% (⅓), 50% (½), 67% (⅔), 75% (¾), or 100%.*

a.	What fraction of **total effort**:	is for the **first 4 stages**?	17.00%
b.		is for **testing**?	33.00%
c.	When the project is **50% done**:	what % of **design** is done?	50.00%

d.		what % of **coding** is done?	50.00%
e.		what % of **testing** is done?	50.00%

3. Explain why **Agile** is likely to find and fix errors faster than **Iterative**.

It is faster to find errors because you are spreading it out over a longer time.

E. Other SDLCs

1. Explain why the diagram above might be called a **Big Design Up Front (BDUF) Model**.

After you design once you move on to testing and coding until you solve all the problems.

2. Explain when & why this might be a good choice for a SDLC.

If you know what you want the design to be, you can make sure that all the bugs of the code are seamlessly fixed.

3. Explain why the diagram above might be called a **Spiral Model**.

You repeat every stage over and over until you solve all the errors.

4. Explain when & why this might be a good choice.

Its good for testing prototypes and to re-designing to final product.

F. Summary

1. Decide whether each statement below is true or false.

	Statement	**T/F**
a.	Waterfall is often the best SDLC.	F
b.	Agile is often the best SDLC.	T
c.	Starting a stage usually includes some extra work, which adds to the overall cost & time for SDLCs with many stages.	F
d.	If you spend too little time on analysis and design, you may have to make major changes to the software later in development.	T
e.	It's better (faster, cheaper) to get things right the first time, than to make errors that have to be corrected later.	T
f.	If you're not sure what users want or how they will use the software, it can be easy to design and develop features that are not needed.	T
g.	It's just as hard to fix a recent error as to fix an error made weeks or months ago.	F
h.	Major design decisions (system architecture, inheritance hierarchies) are usually easy to change during development.	F

2. Decide which SDLC models (Waterfall, Iterative, Agile, Spiral) best fits each scenario.

	Scenario	**Best Model**
a.	You are developing a radically new type of software, and during development you want to show many preliminary versions to potential users.	Spiral
b.	Your organization develops many similar systems (e.g. basic web sites, device drivers) so there are few surprises and you need to produce software as quickly & cheaply as possible.	Waterfall

3. Create a scenario for a SDLC model not listed above.

Agile: I have a huge team and I need to make software that is labour intensive .

4. Summarize the key characteristics of each SDLC model listed below.

SDLC	Key Characteristics
Waterfall	
Iterative	
Agile	
Spiral	
BDUF	