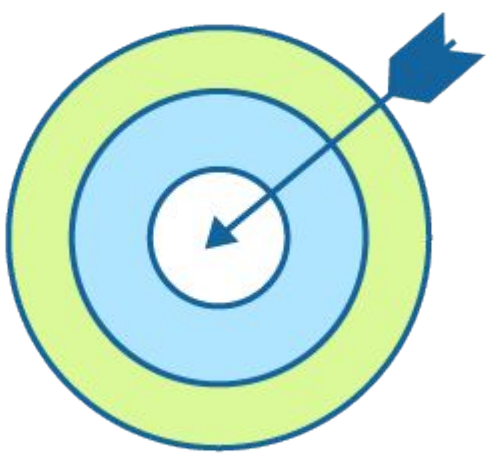
English Course for Developers

Prof. Carlos Arriaga

Understand the Fundamentals of Software Development

Goals for this lesson



1. Customer's inception
2. User stories
3. Iterations as part of a development process

Software development

Software development is all about…

Creating great software that makes your customer happy.

Customer’s inception

Customer’s inception

The moment when your customer has an original idea and needs you to materialize it—having a deadline and budget.

Customer's inception

The customer needs software.

It is your job to take steps to transform an intangible idea into working, deliverable software.

Constant communication with your client is "key".

User stories

User stories

A user story is an individual task the software has to do, and obviously you have to code it.

User stories

Before implementing the idea, think about the individual tasks (user stories) the software has to do.

User stories

Build the initial requirement - a group of actions the software has to perform.

Listen to your customer!

User stories

Title Catalog creation

Description

The website will show a catalog of all available bicycles.

Priority= 25

User stories are composed of individual tasks, they contain a title, a description and a priority rating.

25-Urgent

15-20-Important, include in the next sprint.

6-12- Moderately important

1-5 Nice-to-have, Low priority

User stories

User stories simplify that idea into smaller, actionable tasks.

Example:
1- Create the product catalog.

2- Users can select their favorite model.

3- Users can select their favorite color, etc.

Title Color selection

Description

Clicks on her favorite color.

25

Title Bicycle selection

Description

The user clicks on her favorite model

25

Iterations

What is an iteration?

The repetition of a process; it’s a period after which you get together with the client and discuss your progress.

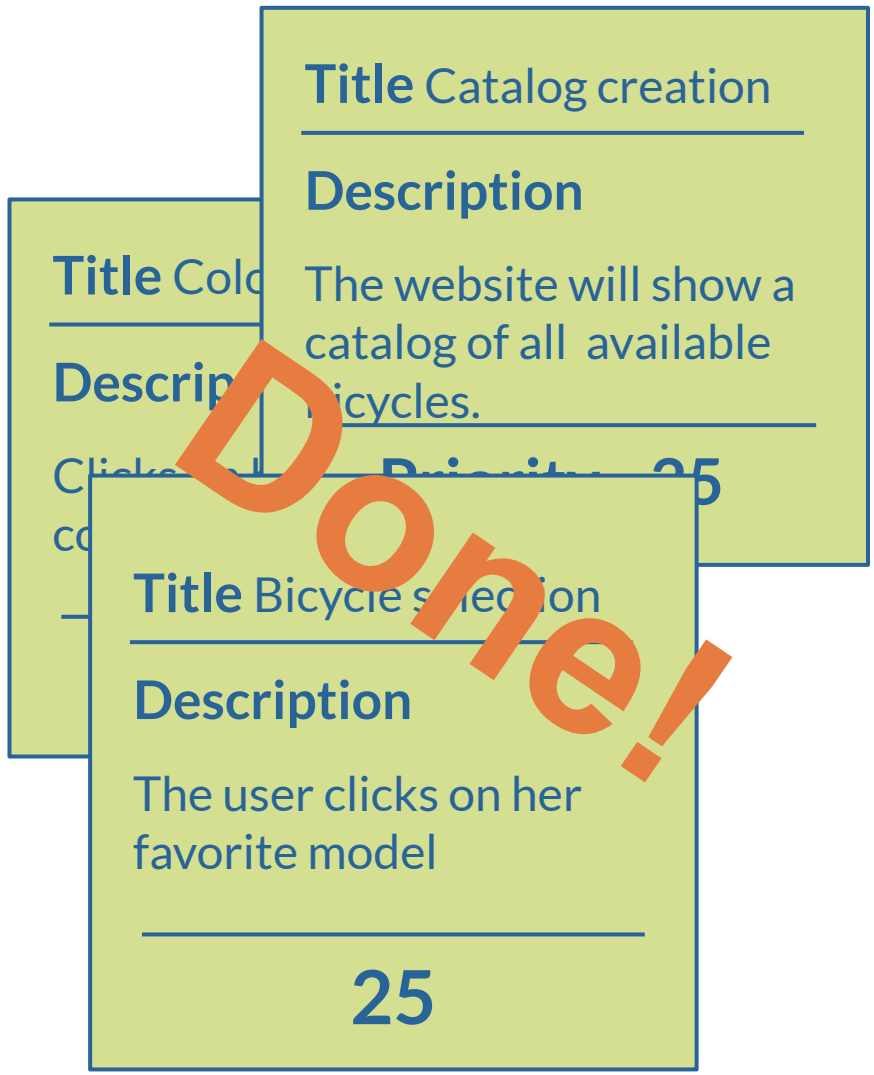
A milestone or checkpoint.

Iterations

Have an iteration once you have completed the user stories in the period (normally one month).

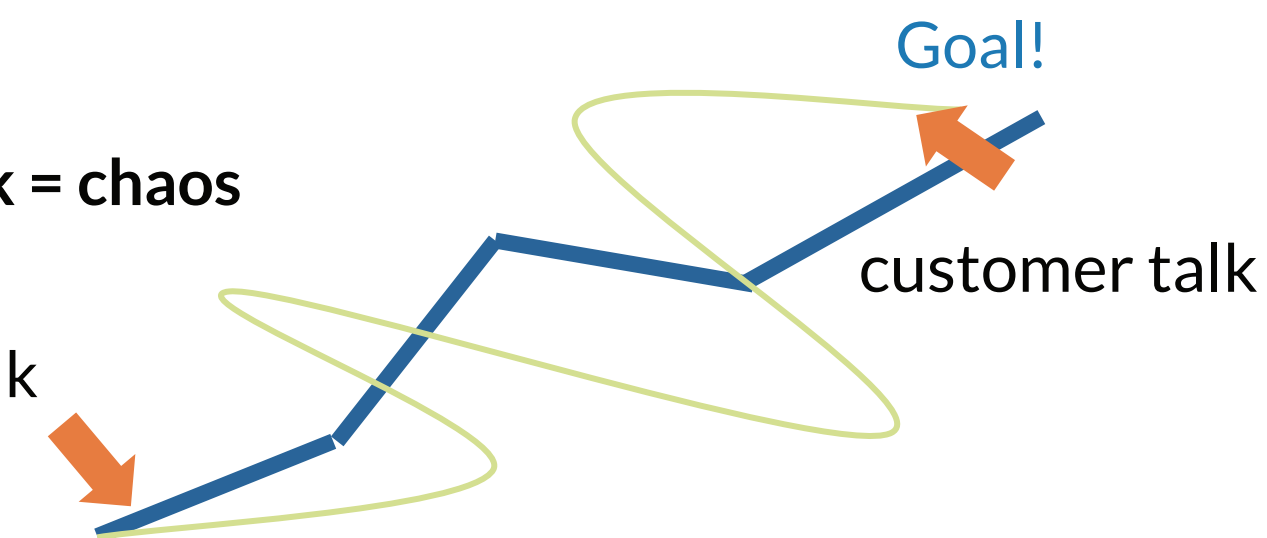
Meet with the customer and discuss your progress.

Show your customer your working code.



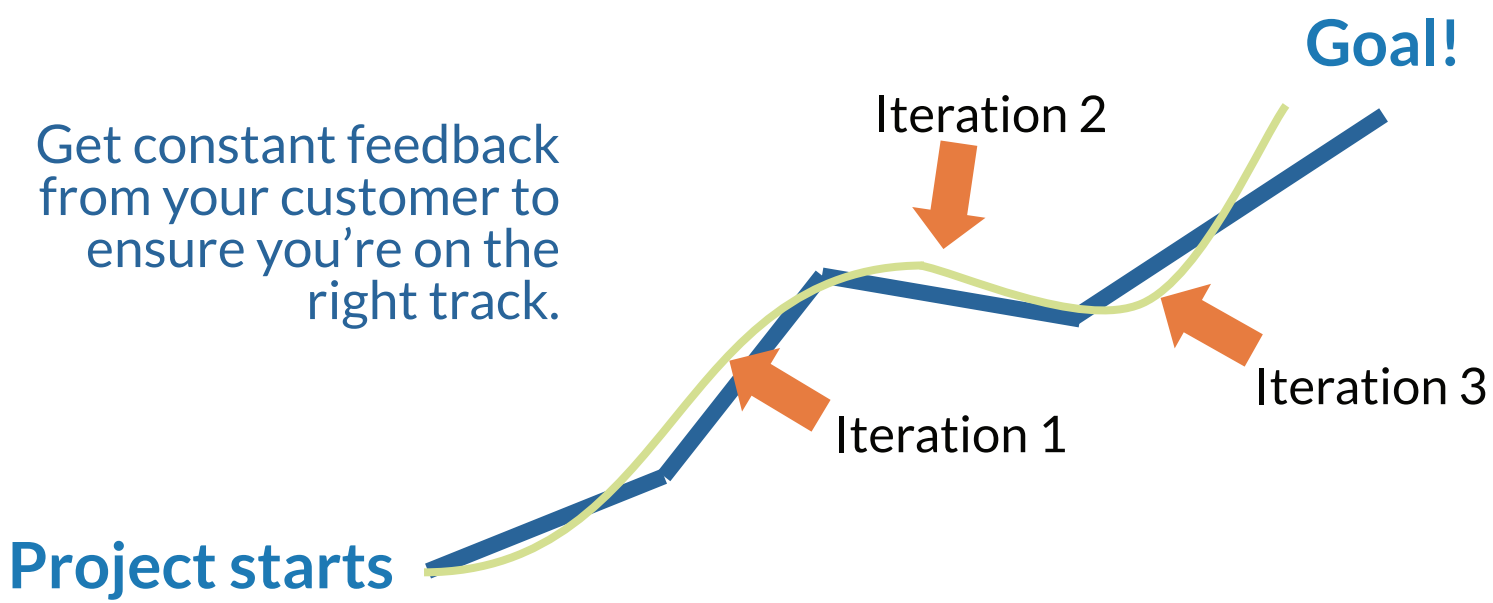
Iterations are important because...

They help you control your progress and make sure you're following your customer's instructions. Developing without iterations results in chaos because...

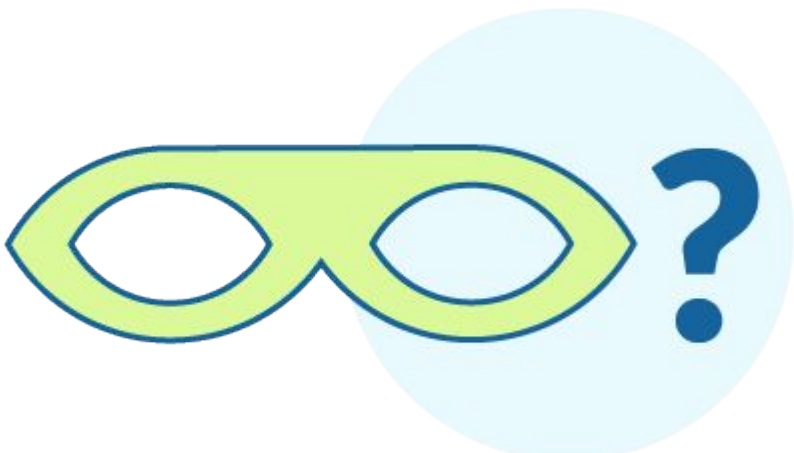


Iterations are recommendable because...

They allow you to make quick adjustments that will create great results. Every time you make significant progress, have an iteration with the customer.

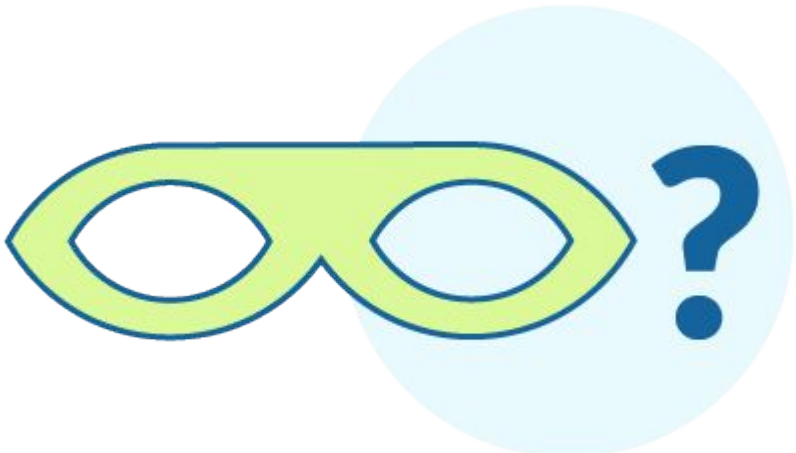


Who am I?



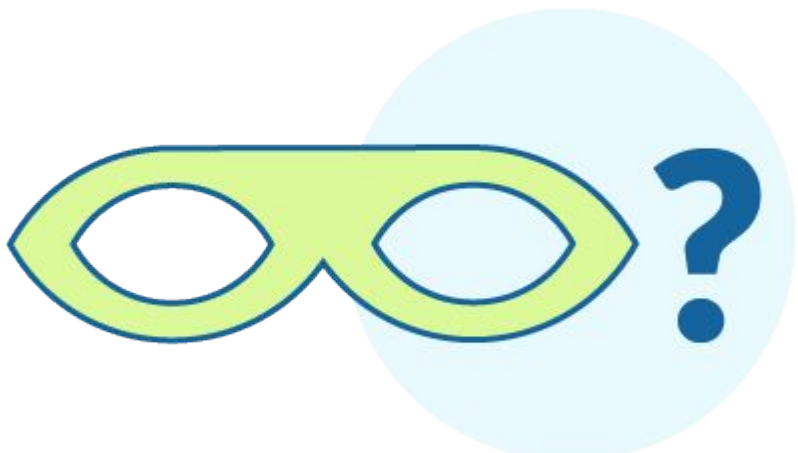
Let’s play a little game. Can you guess the concepts if I give you some clues?

Who am I?



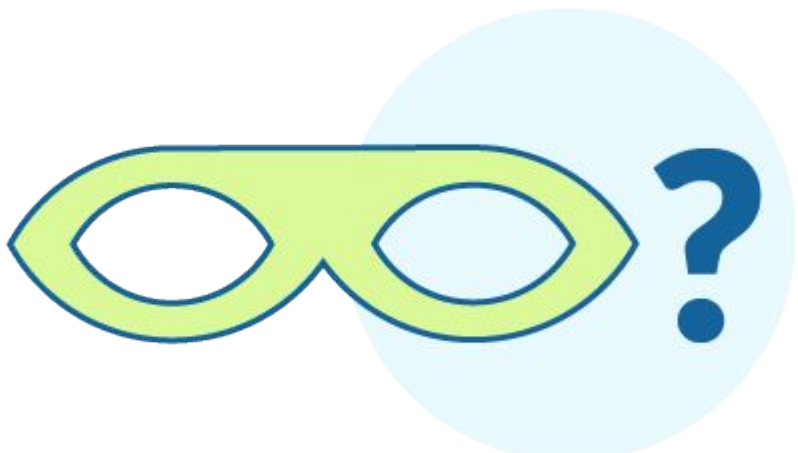
I am the description of a individual action the software performs. I have a title, a description and a priority...

Who am I?



I am a frequent check-up
to talk about progress,
I take place once a month.

Who am I?

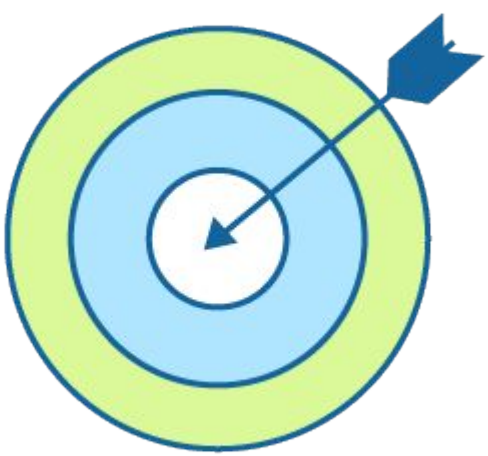


I am the initial idea a customer has, I am also the name of a famous sci-fi movie...

This is the end of this lesson

Communicate with your customer accurately

Goals for this lesson



1. Accurate communication
2. Idea brainstorming
3. User stories
4. Estimates

Accurate communication

Accurate communication

Customer requirements / Understanding the customer

At the beginning some information might still be unclear - go back to the customer and clarify.

Get info on requirements. Think of everything you need before you start building the software.

Accurate communication

What questions do you think you could ask if we take the bicycle online sales as an example?

Can you think of some questions?

Let me know in the chat

Accurate communication

Examples:

How many different types of bicycles does the software have to support?

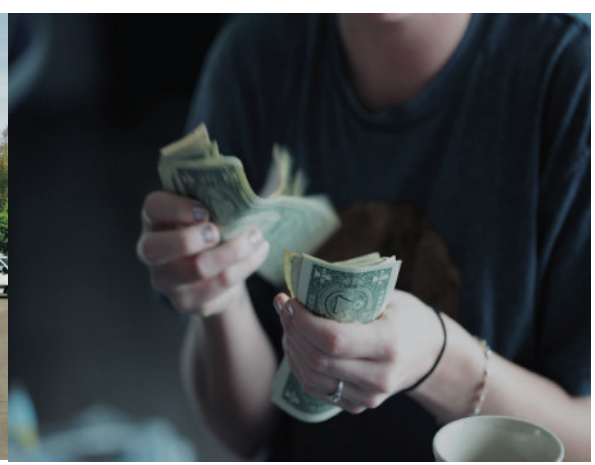
Should the software allow any changes to color or model once the bicycles have been purchased?

Accurate communication

Does the software have an administrator interface to make changes?

Will the software have to talk to other systems? Paypal or others?



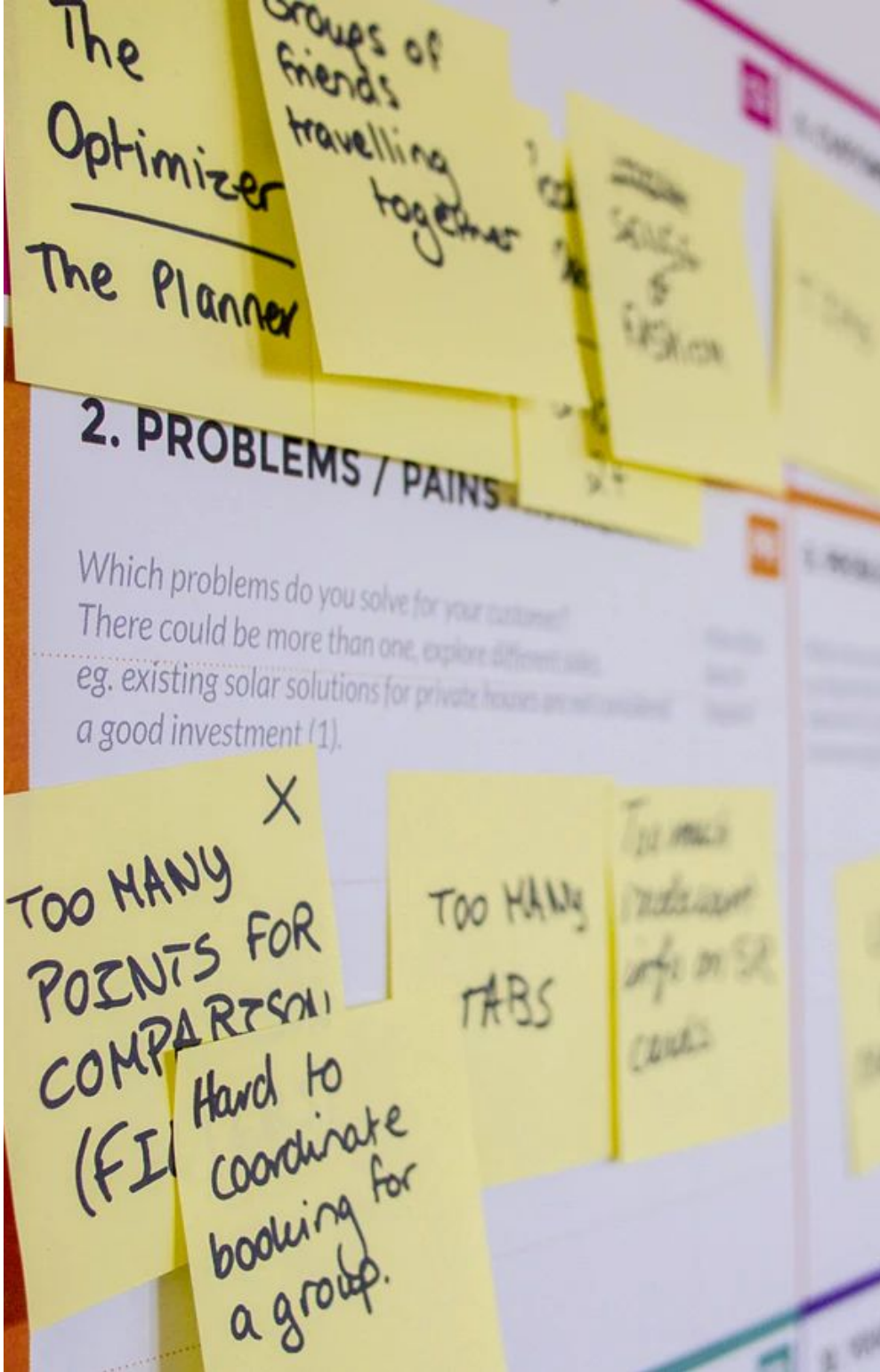


Idea brainstorming

Brainstorming

A session designed to express ideas about the project design. Any idea can be expressed.

Promote the right atmosphere to avoid having a foggy or muffled session.



“

Two heads are better than one and four heads are better than two, as long as people can contribute without criticism.

”

Carlos Arriaga

Brainstorming

The objective of Brainstorming is getting good requirements that will create great results and great software.

You can also try Brainstorming and observation as a complement to Brainstorming to gather good requirements

User stories

User stories

Working, functional software is the primary measure of progress.

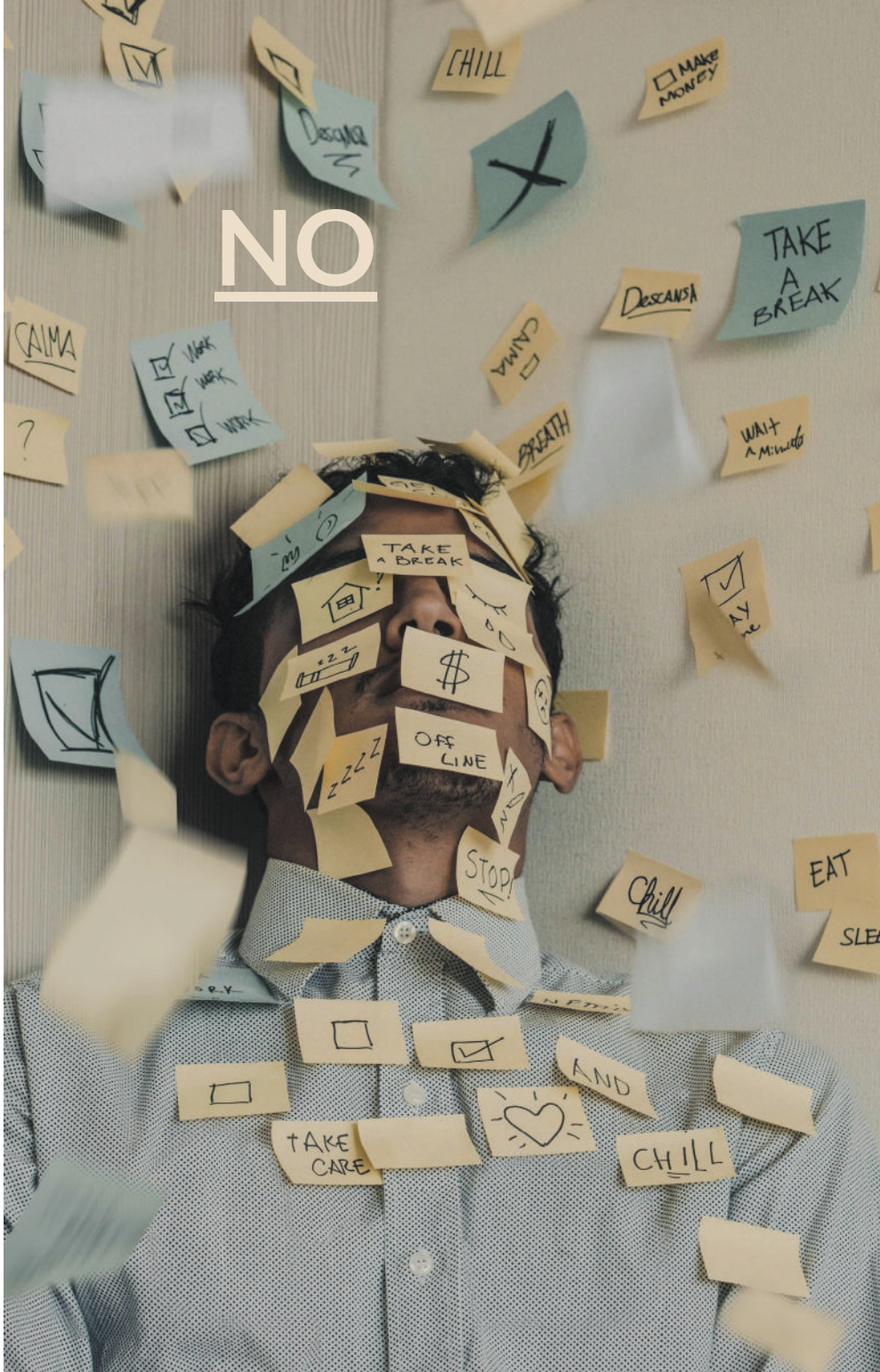
User stories help you sort out what is most important.

Your priority is to satisfy your customer through early and continuous delivery of valuable software.

User stories

User stories should describe one thing that the software needs to do for the customer, they must be short.

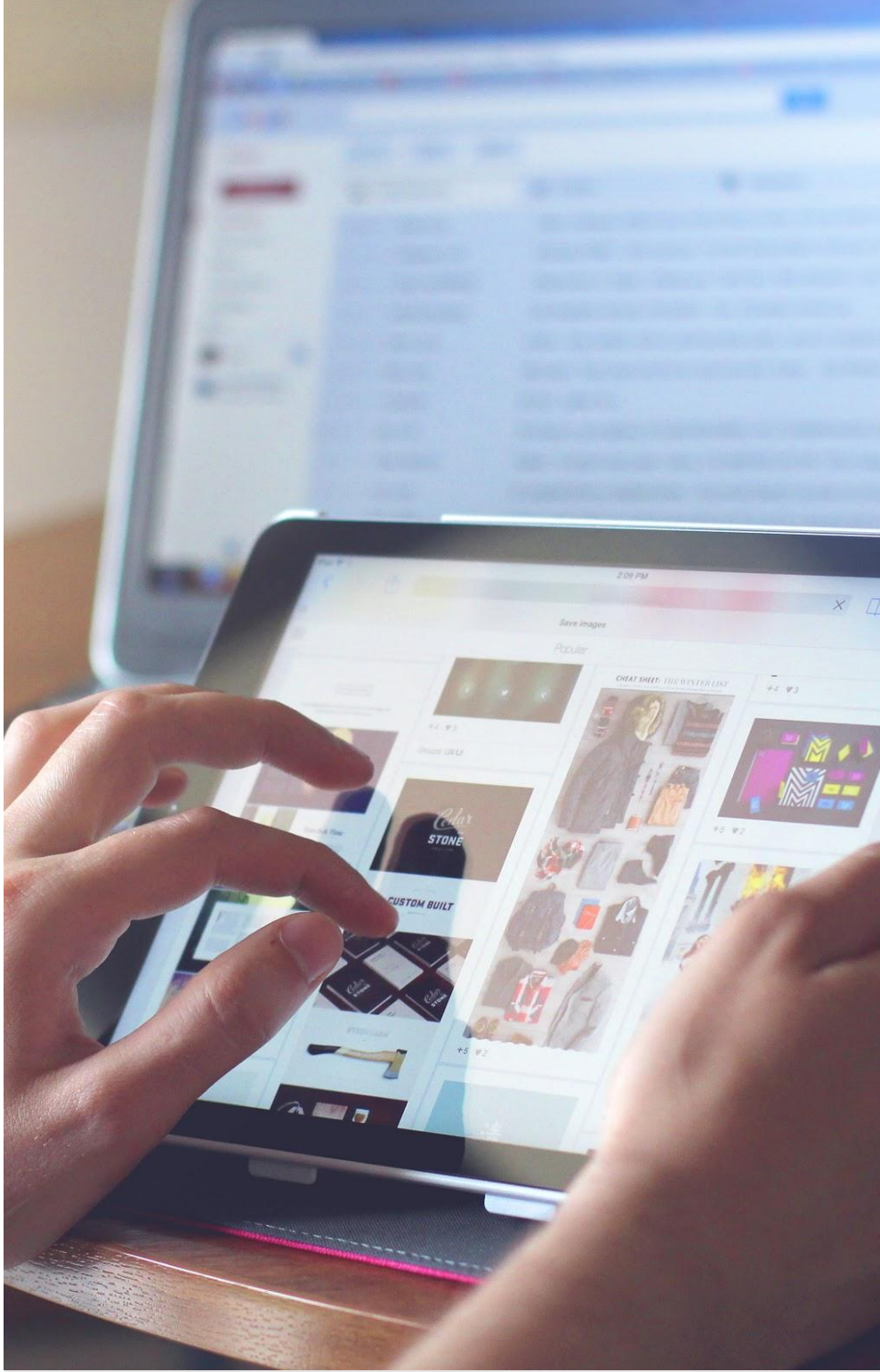
Written using language that the customer understands…avoid using technical terms.



User stories

User stories must be written from the customer's perspective

Example:
AS A USER
I WANT TO….change my password
SO I CAN…..keep my account secure



Estimates

Estimates

After your initial requirement capture stage you will have clear user stories.

The customer will then want to know when those stories will be built.

How long will it take to complete the project? (Estimation)

Estimates

Your estimate is the sum of all the time you take to build the user stories.

Example
User story 1. Catalog creation **(5 days)**
User story 2. Selecting a bicycle **(2 days)**
User story 3. Selecting a color **(2 days)**

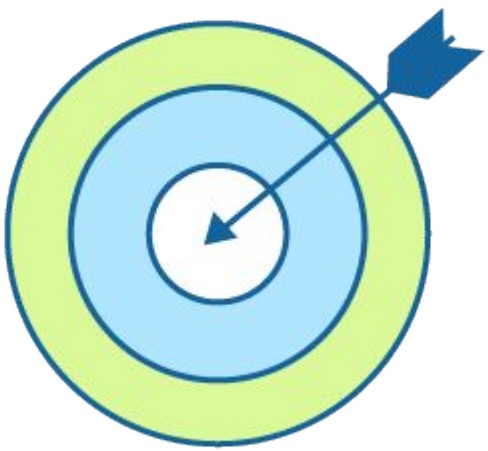
TOTAL= 9 days

This is the end...
...for now

BONUS!
Design thinking PDF

Communicate with your customer accurately

Goals for this lesson



1. Iteration cycles
2. Estimating the whole project
3. Reaching consensus in estimations

Iteration cycles

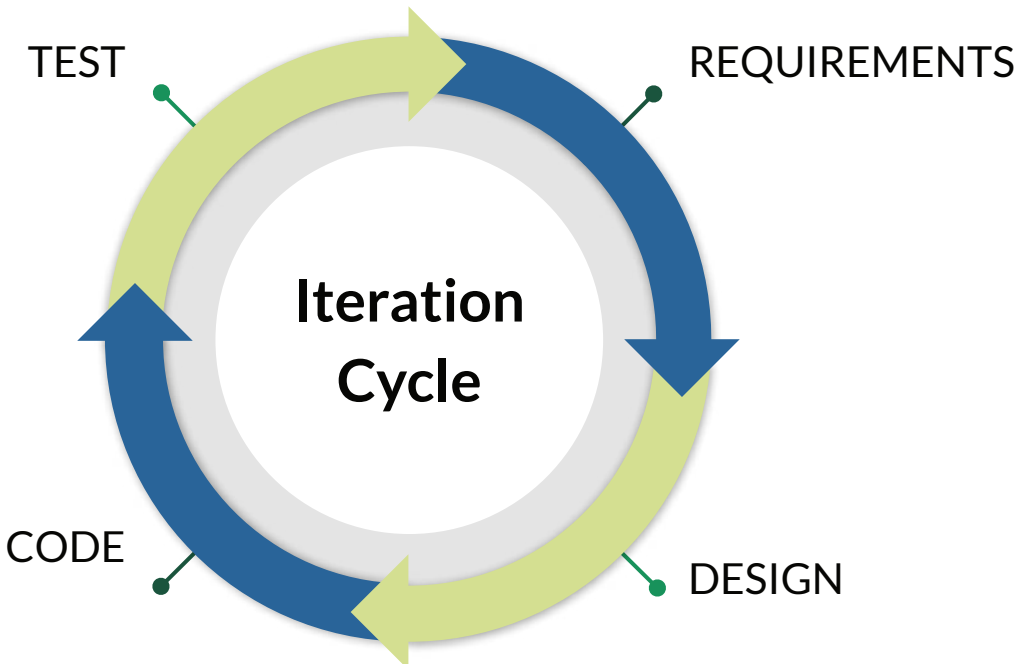
Iteration cycles

The secret to great software is Iterations, and constant communication.

You can’t simply ignore the customer during development.

You get to ask the question: how am I doing?

Each iteration is a mini cycle to produce quality software with requirements, design, code and test.



Each cycle produces working quality software.
A process is really just a sequence of steps.

Estimating the whole project

Estimating the whole project

Your iteration length should be at the right tempo for your project.

User stories define the”WHAT” of your project.

Estimates define the “WHEN”.

This is where the customer asks the big question: how long will it take?

Estimating the whole project

Your project estimate is the sum of the estimates for your user stories.

To figure out how long it will take to complete all of the requirements captured in your user stories,you need to reach consensus in your estimations.

Reaching consensus in estimations

Reaching consensus in estimations

Let's analyze the next example:

NEXT USER STORY: You need to enable the software to receive payments.

Think about people paying with a credit / debit card.

Developers may have different estimations based on their individual assumptions.

Reaching Consensus in Estimations

Let’s analyze the next example:

NEXT USER STORY: You need to enable the software to receive payments.

Think about people paying with a credit / debit card.

Developers may have different estimations based on their individual assumptions.

Mary
5 days

John
13 days

Jane
2 days

ASSUMPTIONS

Reaching consensus in estimations

Assumptions =

The action of accepting something without proof.

Examples:

People will pay with a Visa or MasterCard

But what about Paypal? Or Bitcoin?
Or American Express?

Reaching consensus in estimations

Developers might disagree about estimates.

It is essential your team comes to a consensus, only then can you start estimating accurately.

Eliminate assumptions, have a meeting and agree on Estimates by playing planning poker.

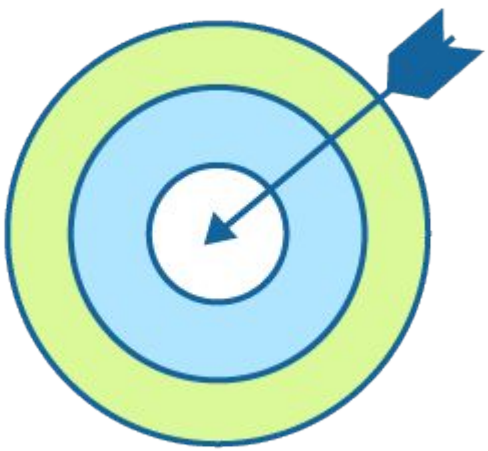
BONUS!
Playing planning poker PDF

All good things….

…must come to an end.

Communicate with your customer accurately

Goals for this lesson



1. Planning considering priorities
2. Milestones

Planning considering priorities

Planning considering priorities

We previously played planning poker and the total time is: 300 days.

However, after informing the customer she says:

Listen, I don't have that much time, I will give you 90 days…

What do you do?

Planning considering priorities

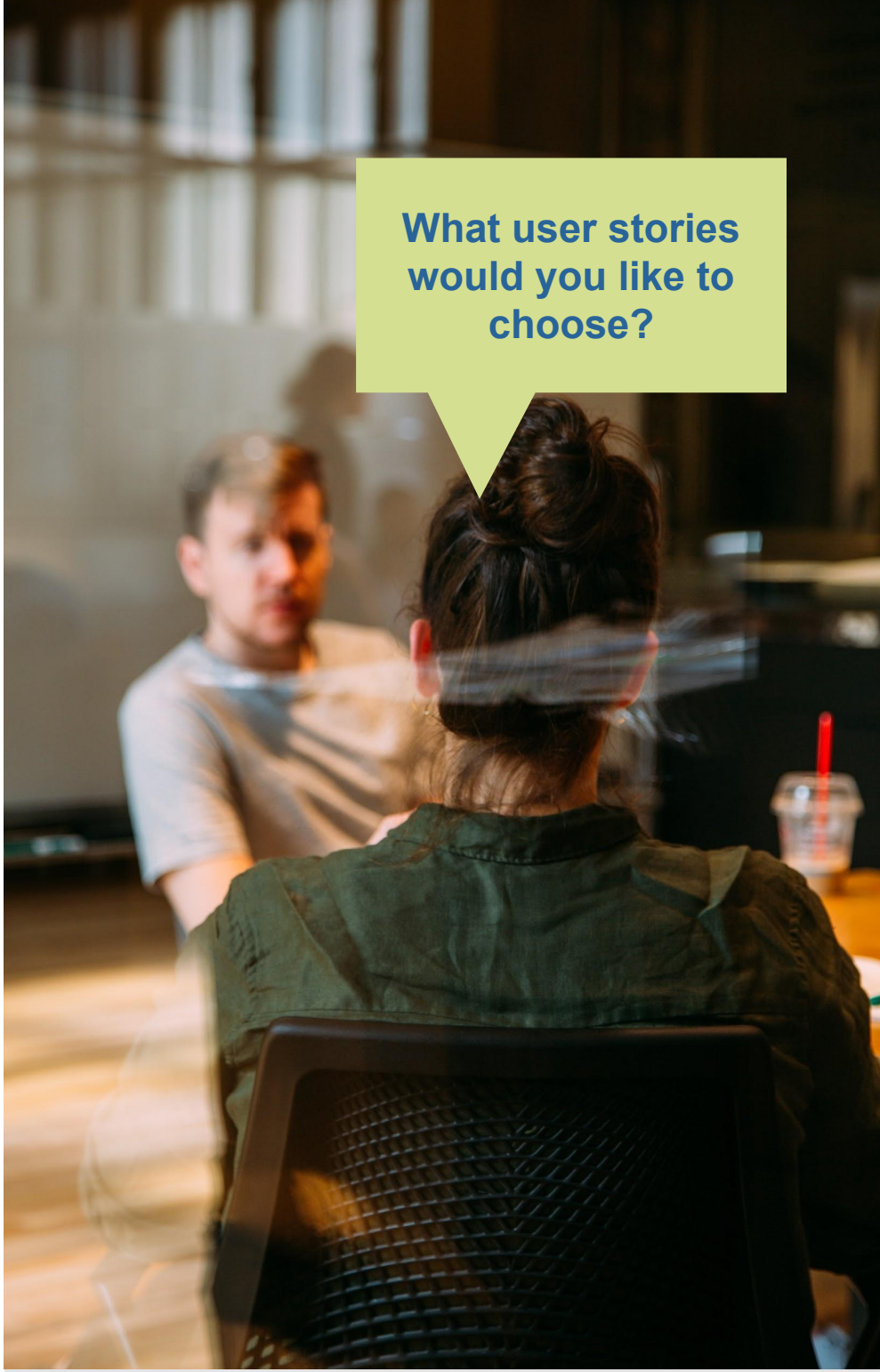
Go back and prioritize your stories with the client.

The client is responsible for selecting the most relevant user stories, it’s her decision.

Planning considering priorities

You should have a meeting with your client, lay out all the user stories and ask her about the order of the priorities.

Then select the set of features to be delivered in milestone 1.0



Milestones

Milestones

A Milestone is a major release.

You deliver your software (and you expect to get paid for it).

This differs from iterations because during an iteration you show your customer the software for your feedback

Milestones

A Milestone is about delivering software with baseline functionality.

Don’t try to include fancy functions, instead focus on delivering software that will cover baseline functionality.

This is the end of this lesson.

Remember the concepts we have covered so far

Review I

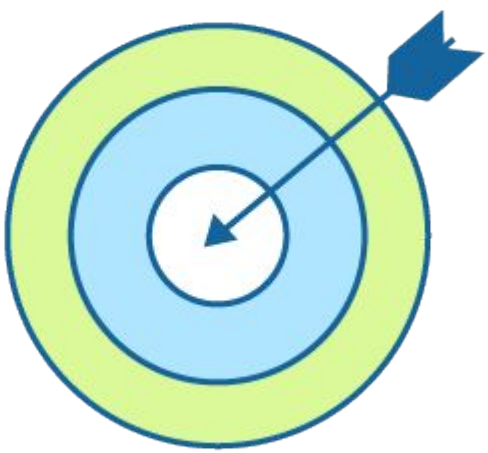
- Customer’s Inception
- User stories
- Iterations as part of a development process
- Accurate communication
- Idea brainstorming

Review I

- Estimates
- Iteration cycles
- Estimating the whole project
- Reaching consensus in estimations
- Planning considering priorities
- Milestones

Understand your customers and their requirements

Goals for this lesson



1. Achievable development plan.
2. Defining iterations
3. VELOCITY - Productive time

Achievable development plan

Achievable development plan

Setting ambitious goals is always important, however if your goals and the customer’s goals are not aligned then there’s a great chance something will go wrong.

Achievable development plan

At the beginning of each iteration, make sure you have the right set of features and that you’re doing what you’re supposed to do.

Stay customer-focused at all times, only the customer knows what is needed.

Achievable development plan

User stories take priority based on your customer’s decisions.

Help the customer make decisions, lay out all your user stories in front of him/her.

This influences the set of features that your software will contain.

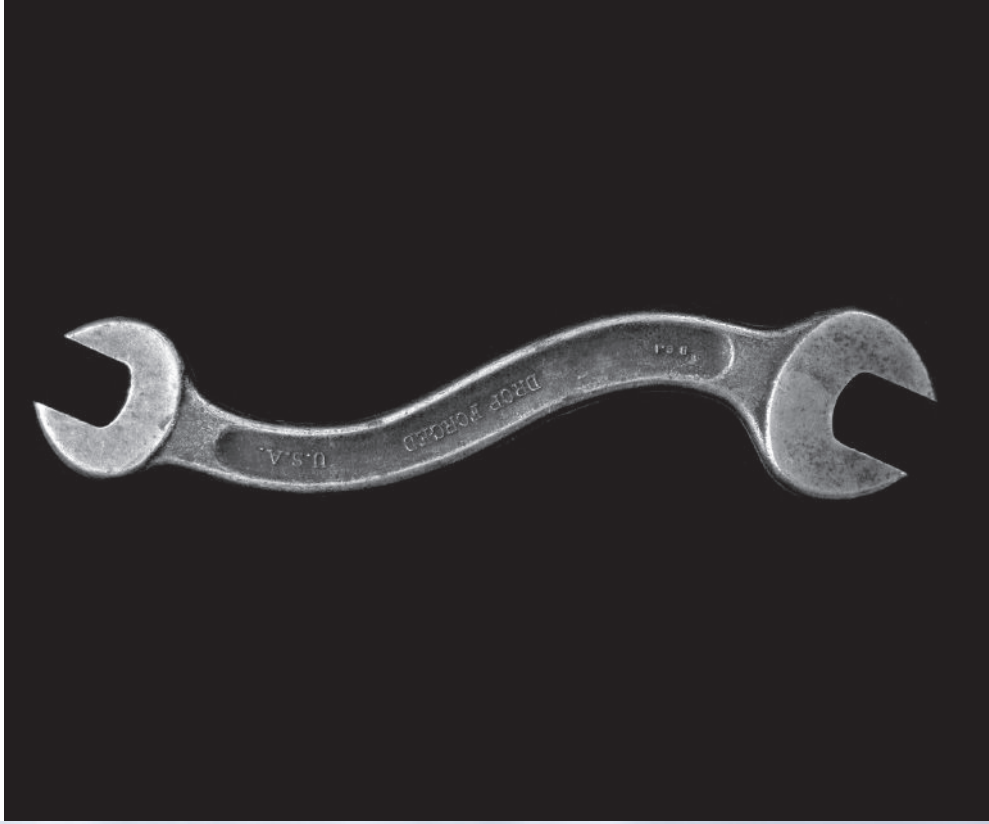
Defining iterations

Defining iterations

The shorter your iterations are the better the chance you have to catch any modifications or any unexpected details quickly.

Defining iterations

This way you can adjust your plans and change what you're doing if necessary before reaching a milestone.



VELOCITY - Productive time

VELOCITY
Productive time

People are not bots and you can’t expect them to be working 100% of the time.

People have different personal issues to deal with such as:

Holidays,

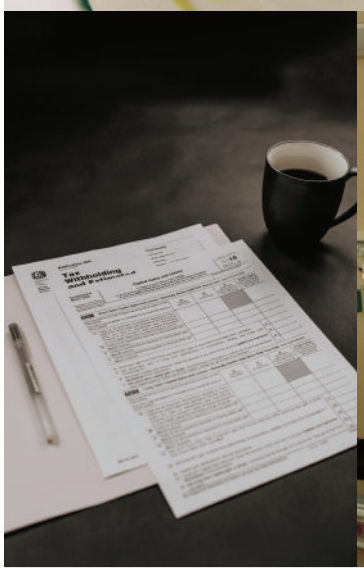
Sickness,

Paperwork.

VELOCITY
Productive Time

Add a little reality into the plan.

Factor in all those little things that can happen and prevent you from reaching a 100% velocity.



VELOCITY
Productive time

Velocity is how fast your team can actually work. It is a percentage.

Given X number of days, how much of that time is Productive work.

Your team will be working about 70% of their available time.

This means your team has a velocity of 0.7 . I.e. For every 10 days of work about three will be taken up personal events.

VELOCITY
Productive Time

At the beginning a 0.7 velocity estimate should be enough to start and adjust.

This will give you a realistic estimate.

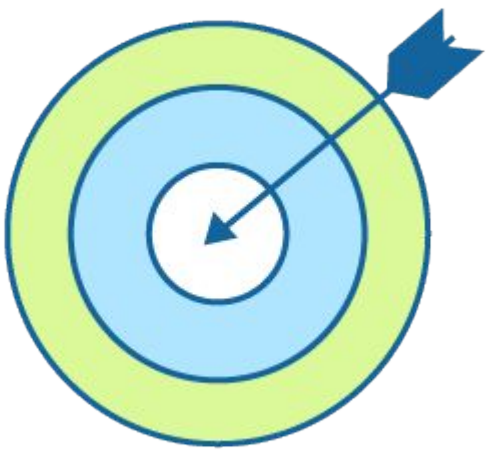
Be confident in your plans by applying velocity and not overworking yourself and your team

BONUS!
Prioritizing with the customer PDF

This is the end...
...for now

Understand your customer and their requirements

Goals for this lesson



1. Backlog
2. Milestone 1.0

Backlog

Backlog

It’s time to set up your software development dashboard aka backlog for iteration I.

Backlog

The backlog is actually just a big board on the wall of your office.

Use it to keep tabs on what work's in the pipeline, what's in progress, and what's done.

Milestone 1.0

Milestone 1.0

Reach it as early as you can, during Milestone 1.0 try to iterate around once a month to keep your development work on track.

Milestone 1.0

When you don’t have enough time to build everything ask the customer to reprioritize.

Milestone 1.0

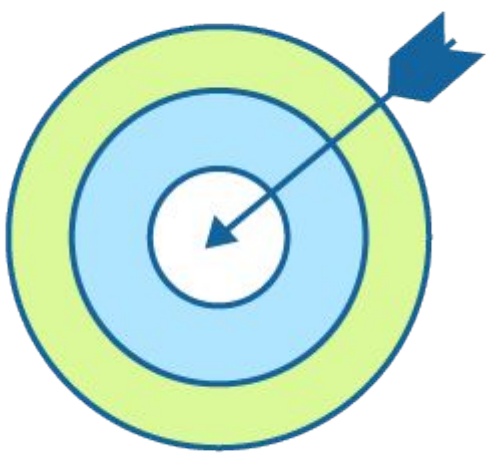
Once you have agreed-upon an achievable set of user stories for milestone 1.0, set up your backlog and get developing!

All good things….

…must come to an end.

Organize your tasks

Goals for this lesson



1. Break user stories into tasks
2. Use estimates to track your project
3. Update your backlog

Break user stories into tasks

Break user stories into tasks

Breaking user stories into tasks adds confidence and accuracy to your estimates and your plan.

User stories are like the ingredients of a bigger more complex “salad”.

Break user stories into tasks

Tasks really add another level of detail specific to the actual coding you'll do for a user story.

Use estimates to track your project

Use estimates to track your project

Track your project from Inception to completion.

It’s often best to break out tasks from your user stories right at the beginning of the estimation process, if you have time.

Use estimates to track your project

This will add even more confidence to the plan that you give your customer.

It’s always best to rely on the task estimates.

Use estimates to track your project

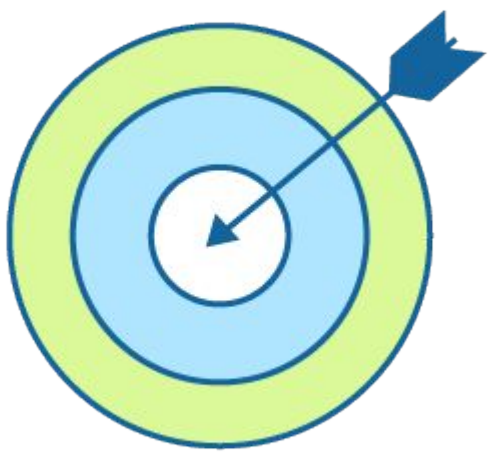
Tasks describe the actual software development work that needs to be done and are far less of a guesstimate than a coarse-grained user story estimate.

BONUS!
Updating your backlog PDF

This is the end of this lesson.

Organize your tasks

Goals for this lesson



1. Standup meetings
2. Analyze and design
3. Modeling your design

Stand up meetings

Stand up meetings

You have some tasks in progress now, to keep everyone in the loop, you conduct a quick stand up meeting everyday.

Stand up meetings

Stand up meetings should

- Track your progress
- Update your burn-down rate
- Update tasks
- Talk about what happened yesterday and what’s going to happen today

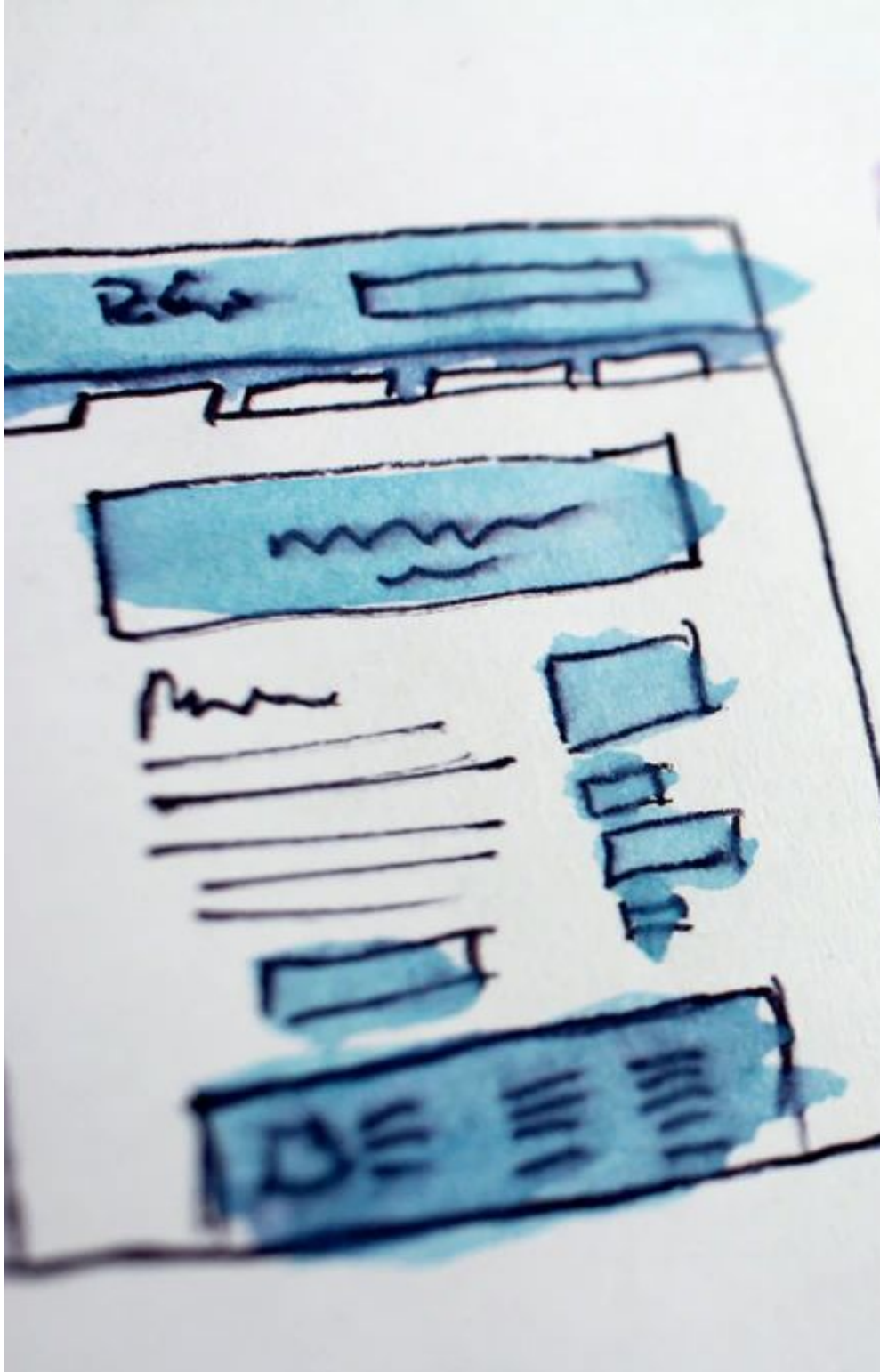
Stand up meetings

- Bring up any issues
- Last between 5 and 15 minutes
- A daily stand up meeting should keep everyone motivated, keep your board up to date and highlight any problems early.

Analyze and design

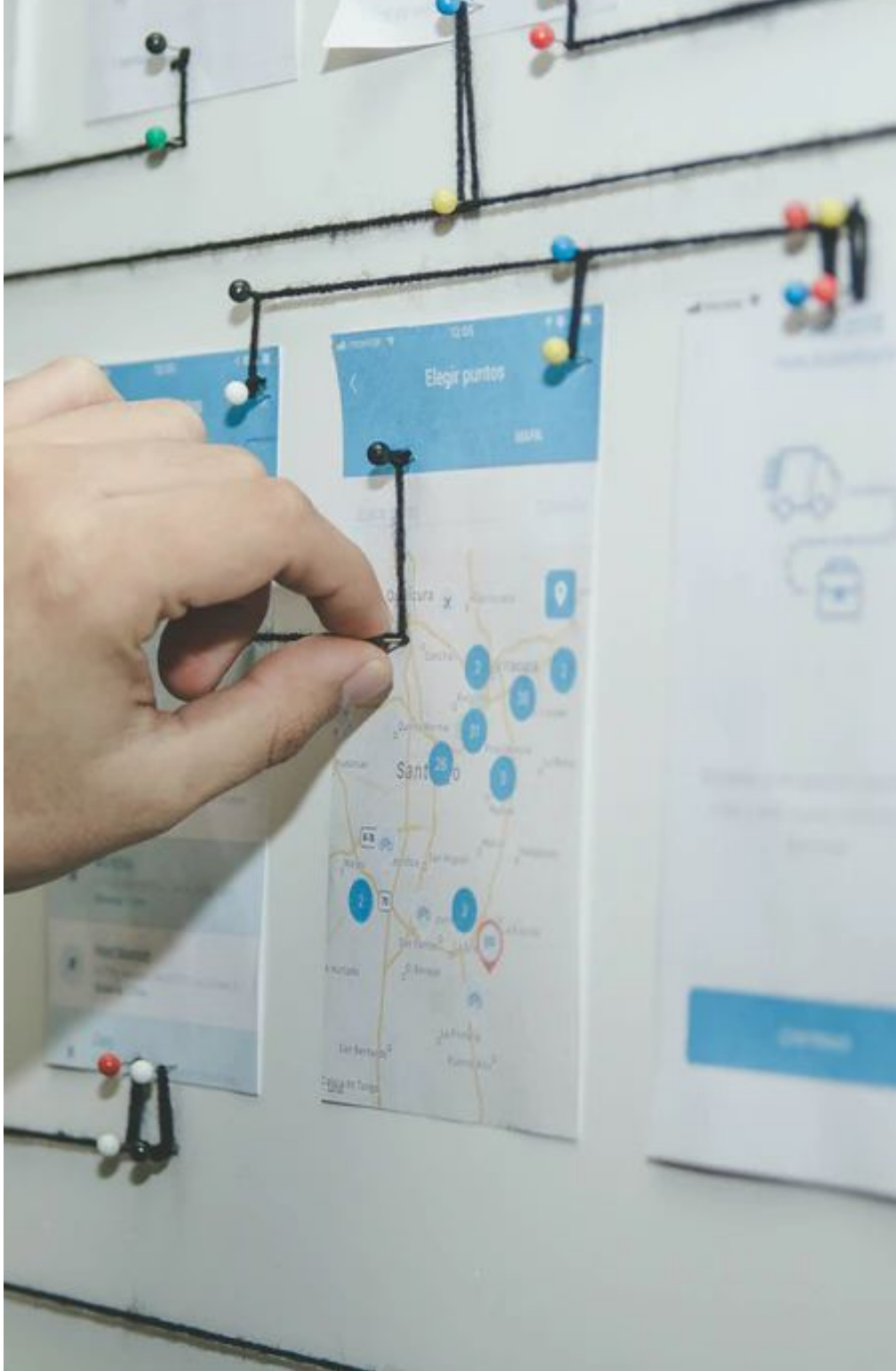
Analyze and design

Analyzing and designing your software as things start coming up is an integral part of the software development process.



Analyze and design

The ability to pivot after seeing necessities that may arise on the fly will demonstrate your readiness and adaptability.



Modeling your design

Modeling your design

Once you realize there are things you need to change then you will need to model your design to fit the new requirements.

Modeling your design

Modifications might force you to add or delete tasks from your backlog.

They might have an impact on the overall estimate for the project.

BONUS!
Burndown rate PDF

This is the end...
...for now

Remember the concepts and definitions we have covered so far

REVIEW II

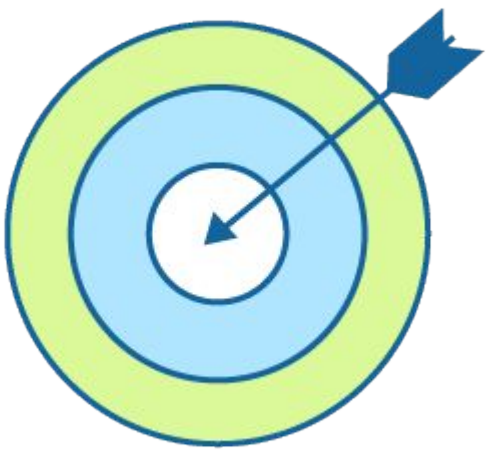
- Achievable development plan
- Defining iterations
- VELOCITY - Productive time
- Backlog
- Milestone 1.0
- Break user stories into tasks

REVIEW II

- Use estimates to track your project
- Update your backlog
- Standup meetings
- Analyze and design
- Modeling your design

Create deliverable design

Goals for this lesson



1. Refactoring your design
2. SRP Single Responsibility Principle
3. DRY Don’t Repeat Yourself

Refactoring your design

Refactoring your design

Refactoring is the process of modifying the structure of your code, without modifying its behavior.

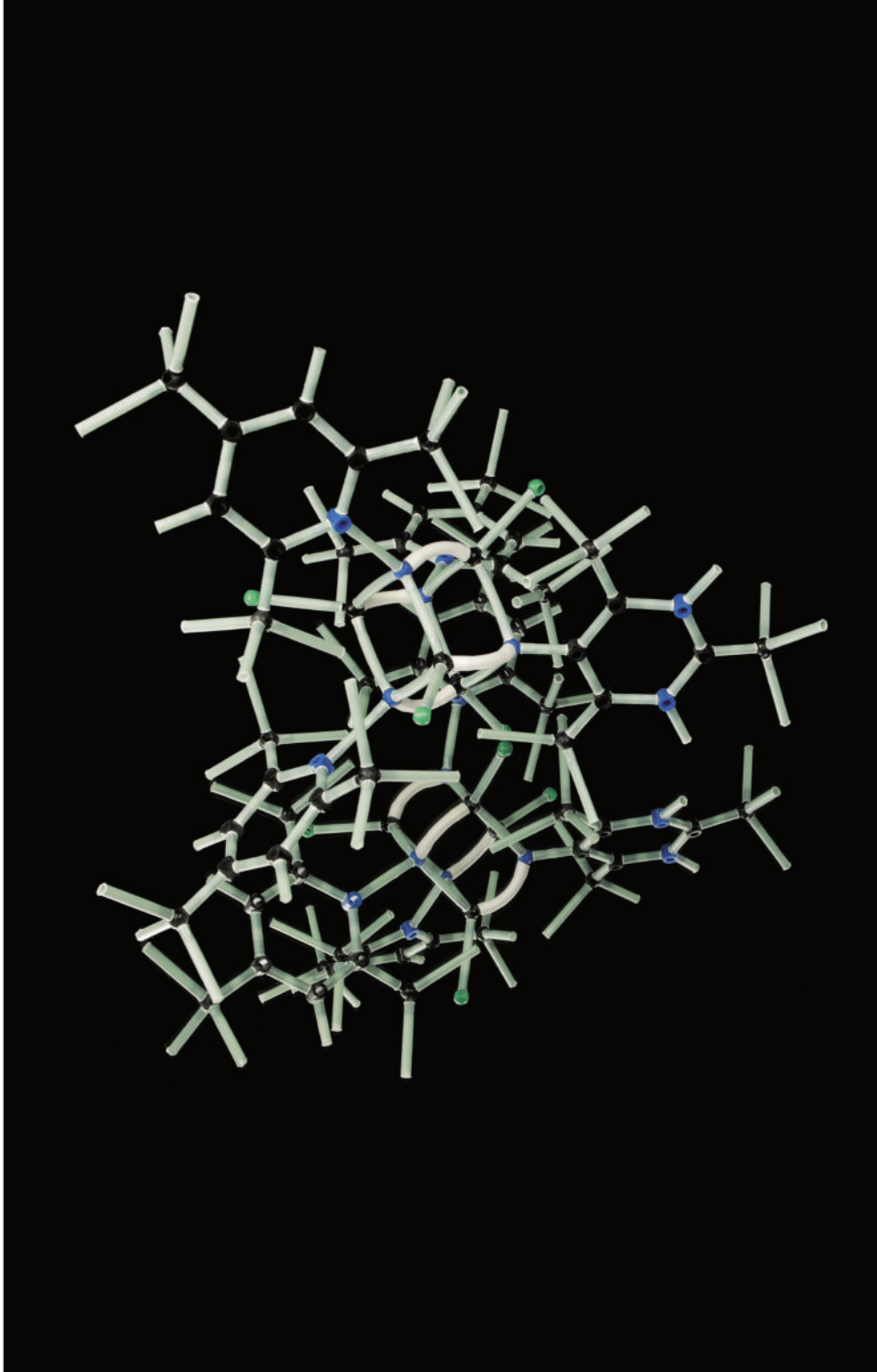
Refactoring is done to improve your code and your design.

Refactoring your design

Refactoring allows you to have cleaner, more readable code.

Clean up any duplication, ugliness, old code, etc.

Once you're done coding, go back and clean up your code for the future.



You code it, you own it.

Carlos Arriaga

SRP Single Responsibility Principle

SRP Single Responsibility Principle

An idea of the SRP is that all of your classes, modules and functions should have only one responsibility and one reason to change.

E.g. Car-point A to point B

SRP Single Responsibility Principle

You have implemented the Single Responsibility Principle correctly when each of your objects has only one reason to change.

DRY

Don’t Repeat Yourself

DRY
Don’t Repeat Yourself

What is DRY?

DRY is about having each piece of information and behavior in your system in a specific, single place.

The DRY principle is aimed at reducing repetition of information of all kinds.

DRY
Don’t Repeat Yourself

Imagine you have a section of your code with:

1. A password request when you enter the first page.
2. The same password request at the end of the shopping cart page.
3. Again at the end of the address confirmation page.

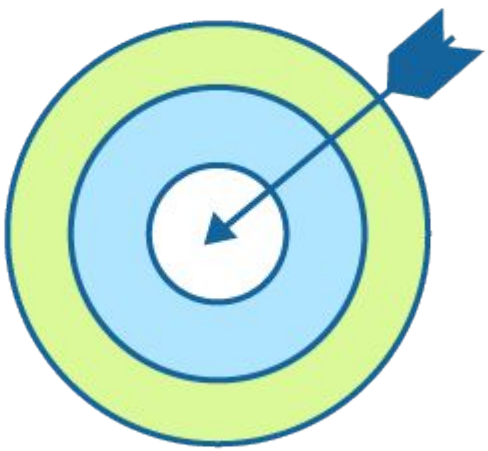
Avoid repetition. Make it DRY

Refactoring, SRP and DRY are all related to improving your design.

It’s been fun but...

Create deliverable design

Goals for this lesson

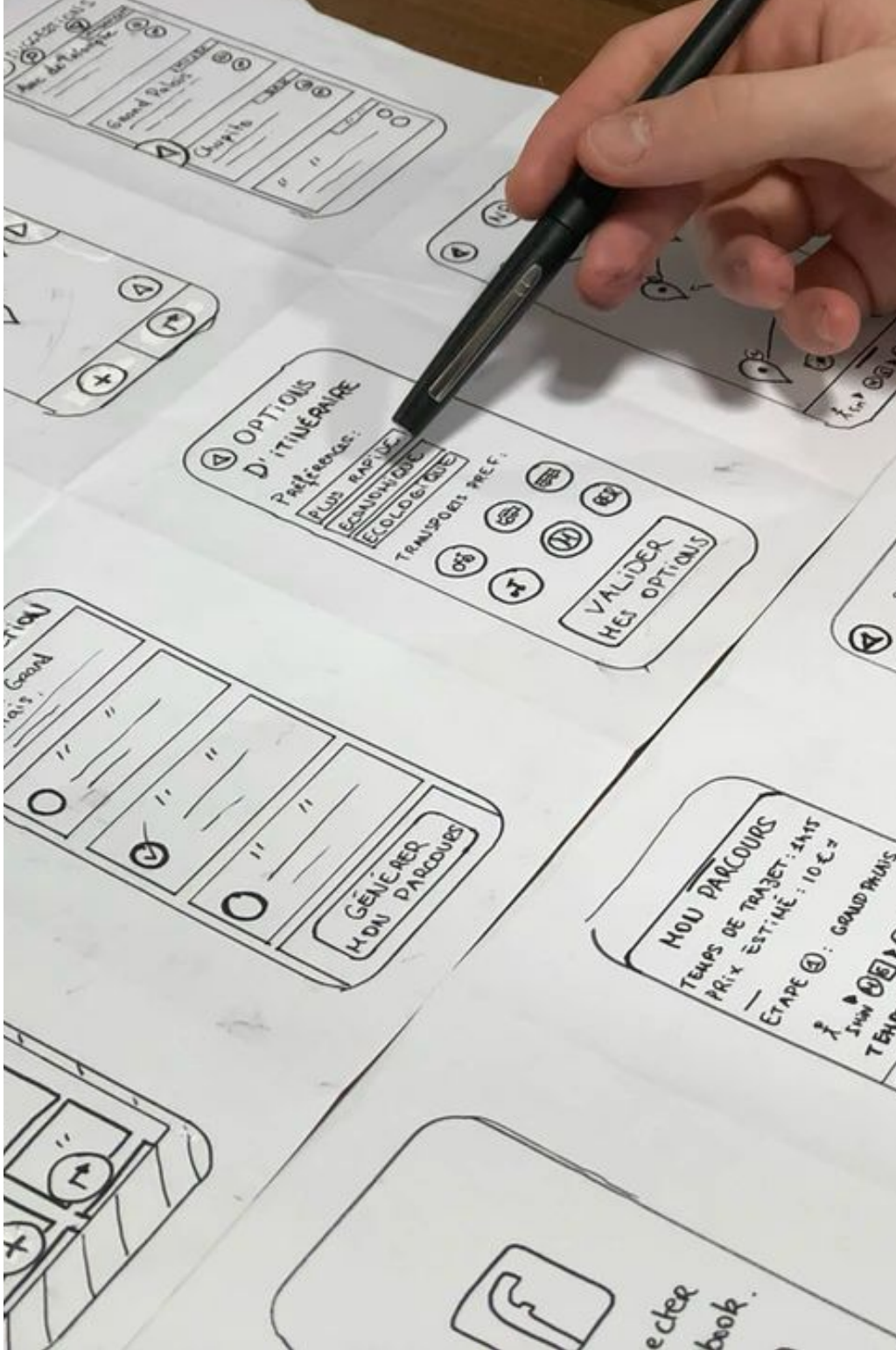


1. Refactoring and stand up meetings
2. Definition is done
3. Ship out / Release software with quality and value

Refactoring and stand up meetings

Refactoring and stand up meetings

Refactoring is the process of modifying the structure of your code, without modifying its behavior.



Refactoring and stand up meetings

Refactoring is done to increase the cleanness, flexibility, and extensibility of your code, and usually is related to a specific improvement of your design.

Definition is done

Definition is done

When everything is complete , then you have definition, it’s done.

Definition is done

You have

- Finished all your tasks,
- Done your refactoring,
- Done any demos the iteration is complete.

When all the work is done, so is your iteration…

Ship out / Release software with quality and value

Release software with quality and value

Your software must always have awesome quality and value.

You should aim at perfection...

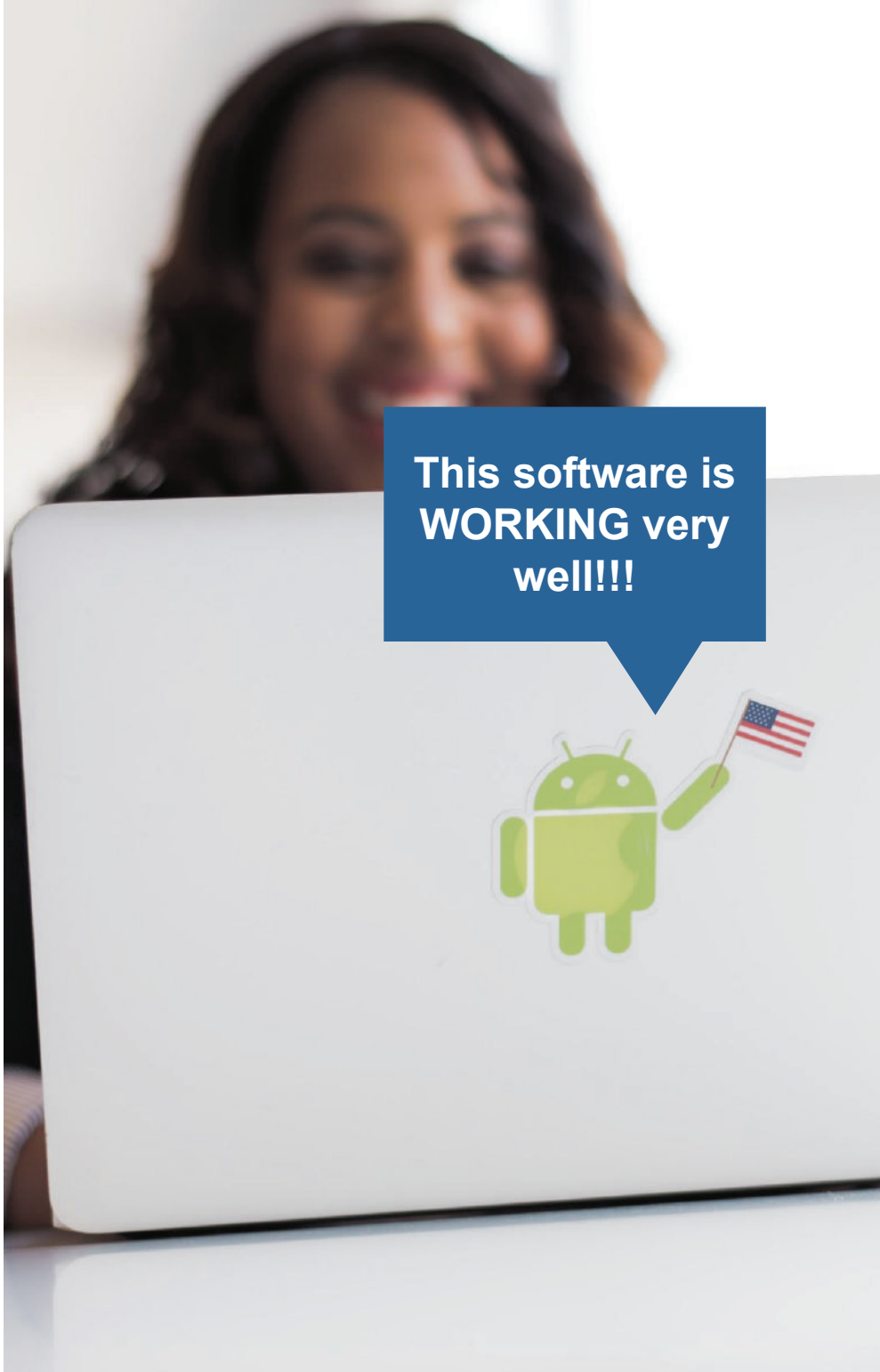
Release software with quality and value

However, let’s not get caught up in the idea of only releasing software when it’s “perfect”.

Your design should be “good-enough” to work well.

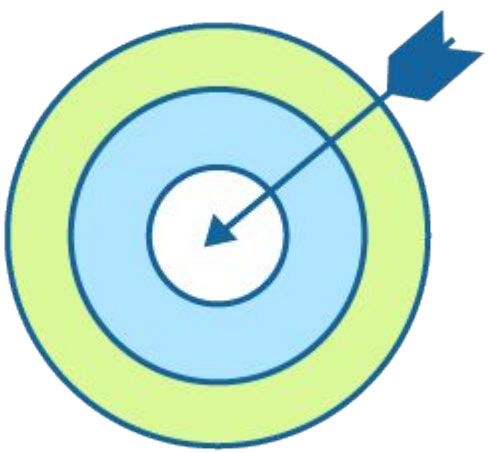
Release software with quality and value

Perfect design is just marvelous and fantastic, but delivering working, effective software is what pays the bills.



Protect your very valuable software

Goals for this lesson



1. Defend your software from yourself and your peers.
2. Technique #1 - version control with your code repository.
3. Technique #2 - control your dependencies.

Defend your software from yourself and your peers

Defend your software from yourself and your peers.

Imagine you are playing the Super Bowl.

You've just scored a touchdown that will give you the advantage over the other team.

What would you do?

Defend your software

Talk to your teammates and make sure their number one priority is protecting that advantage that you have just gotten.

Defend your software

This is where the concept of Defensive Development comes from.

Once you have reached a point where your software is working you need to protect it and make sure it continues working.

Let’s look at techniques to do that...

Technique #1 - Version Control with your code repository

Version Control with your code repository

Version Control is a super cool technology you can use, it’s one of the best defensive tools.

Version Control with your code repository

VC software helps you keep track of changes made to your code.

VC allows you to control different versions of your files, different versions of your code, different versions of what you have committed.

Version control with your code repository

Committing is a very important word here - saving your files onto the version control software.

You are committing new versions of your files into your VC repository.

Version control with your code repository

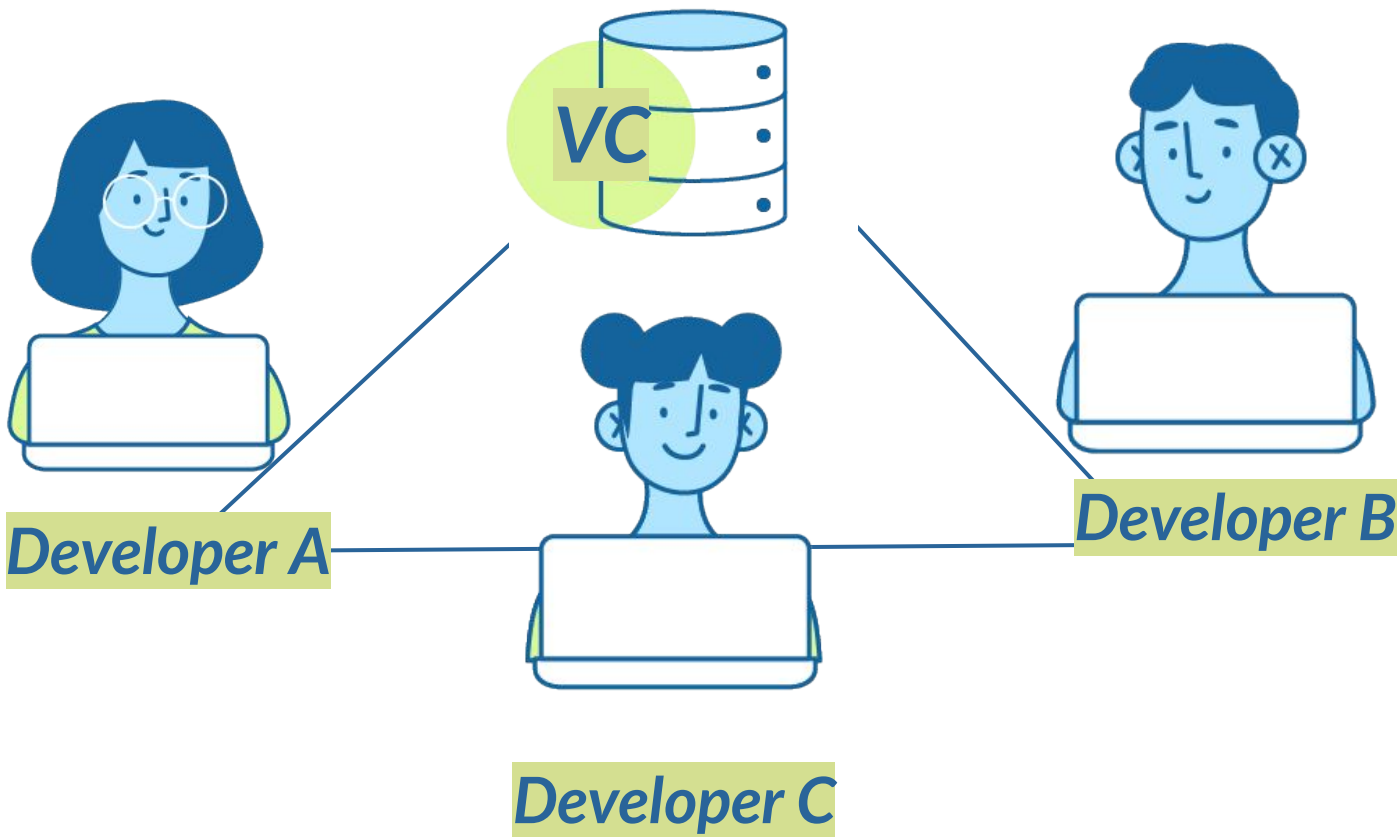
You commit those changes to your branch or branches.

The branch is the file of code you are working on .

Within this branch you can have additional branches, for different uses and modifications on different branches.

Original file and modified versions

Version Control with your code repository



Version Control with your code repository

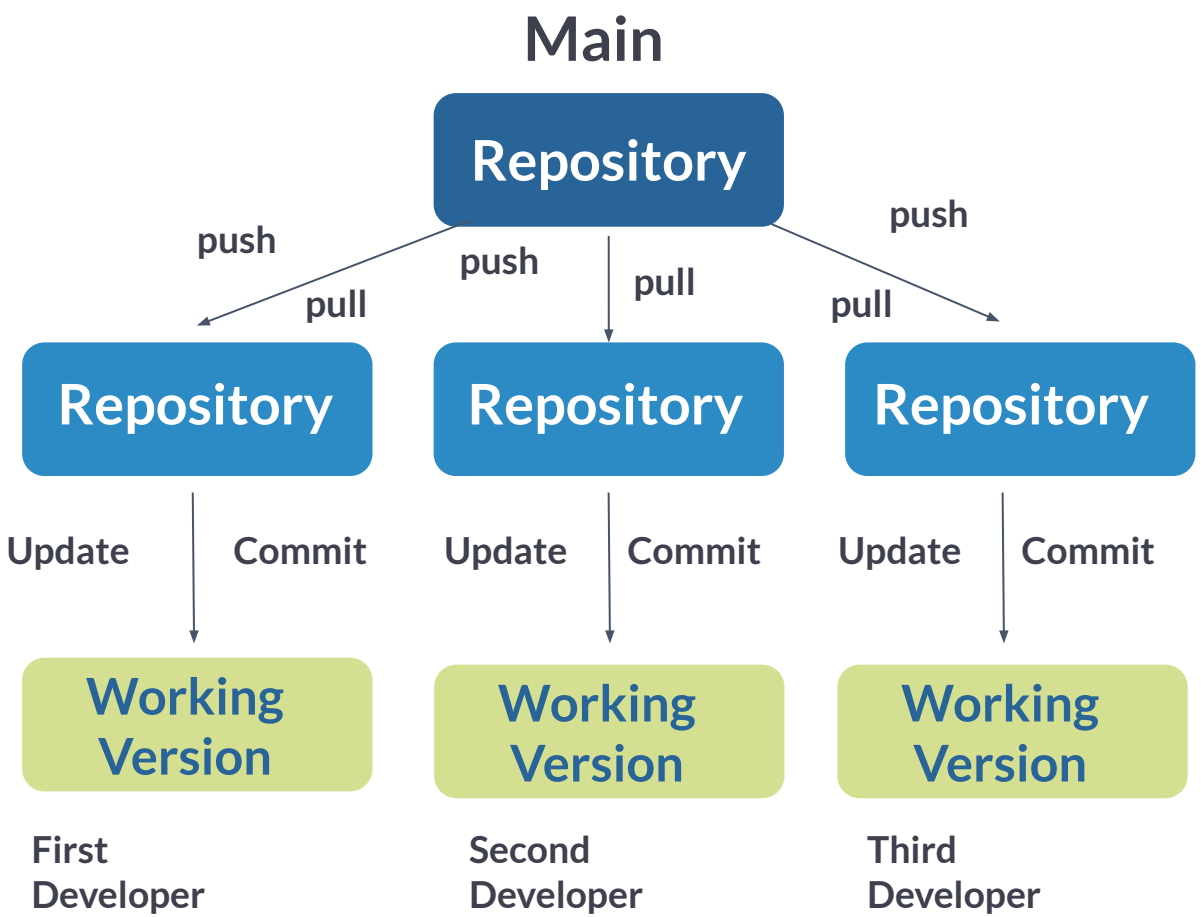
It lets multiple people check changes back into the repository and distribute them.

It keeps track of the history - who changes what, when and why.

It tags so you can find versions of your code from “way back when”.

Version control works mainly as a repository, it doesn’t test your code.

Version Control



Technique # 2 - Control your dependencies

Control your dependencies

Dependencies make your code more complex.

Sometimes you want to test some functionality of your code, but that functionality is tied to another task you haven’t done yet.

What do you do in that case?

Control your dependencies

Real-World code always has dependencies, it's never isolated.

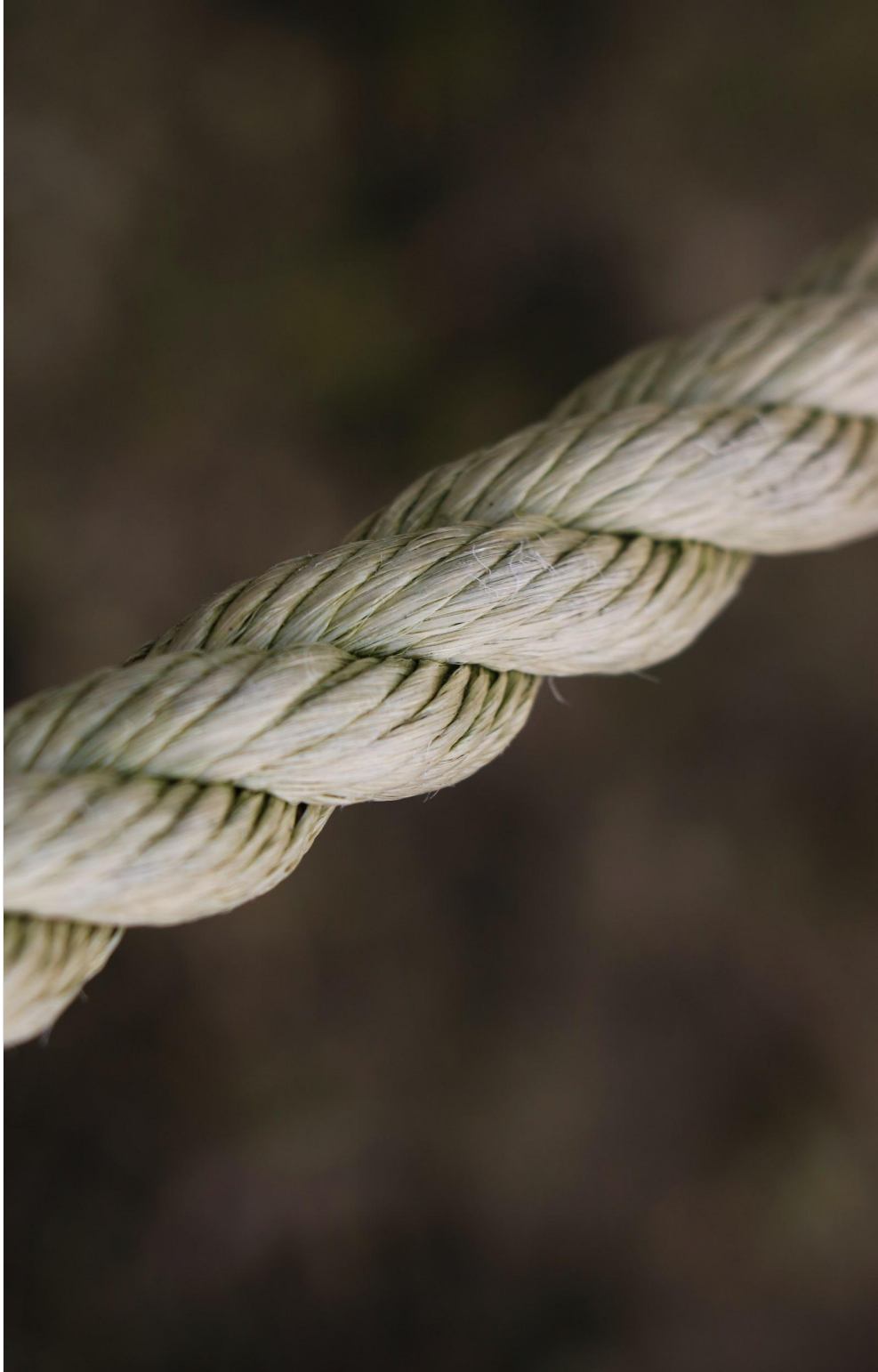
Sometimes you have code that depends on something external, like a database.

You have to figure out a way to test that functionality independent of those dependencies.

Control your dependencies

Always check your dependencies for connectivity necessities.

In other words, you need to find a way to keep things independent, but working together at the same time to make testing easier.



Control your dependencies when building and packaging your software

You can use a tool like ANT for your JAVA Projects and it all revolves around an XML Build Script that is a file you build for ANT and tells the tool what to do when you need to build.

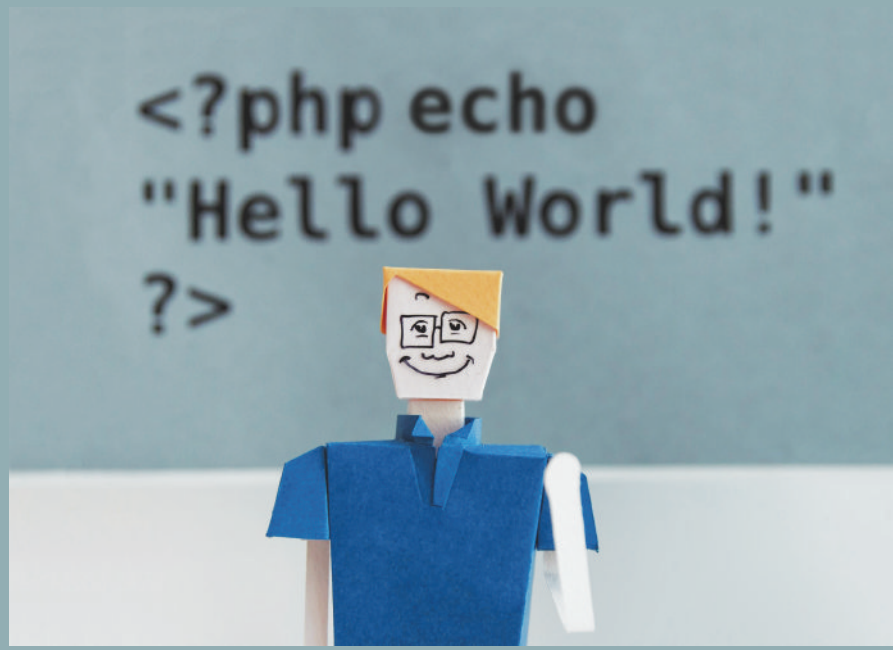
Packaging...

It’s not enough to use version control to ensure your code stays safe.

You also need to worry about compiling your code and packaging it into a deployable unit.

Packaging your software is really important, remember that your software must be usable.

In a nutshell -Take care of your work!

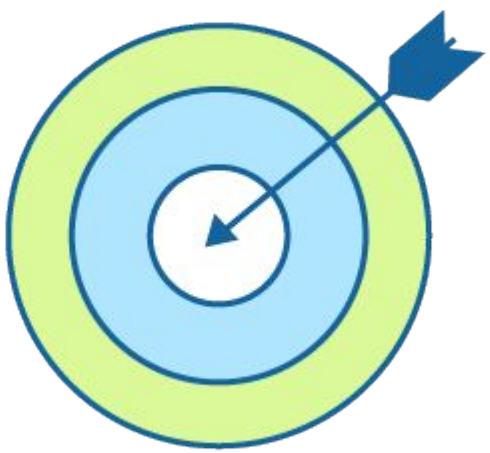


BONUS
Test-Driven Development PDF

See you soon!

Protect your very valuable software

Goals for this lesson



1. Functional testing
2. Technique #3 - Unit testing.
3. Technique #4 -Let your peers understand your code - Document it!

Functional testing

Functional testing

What things can go wrong in a development project?

Different problems will arise when working.

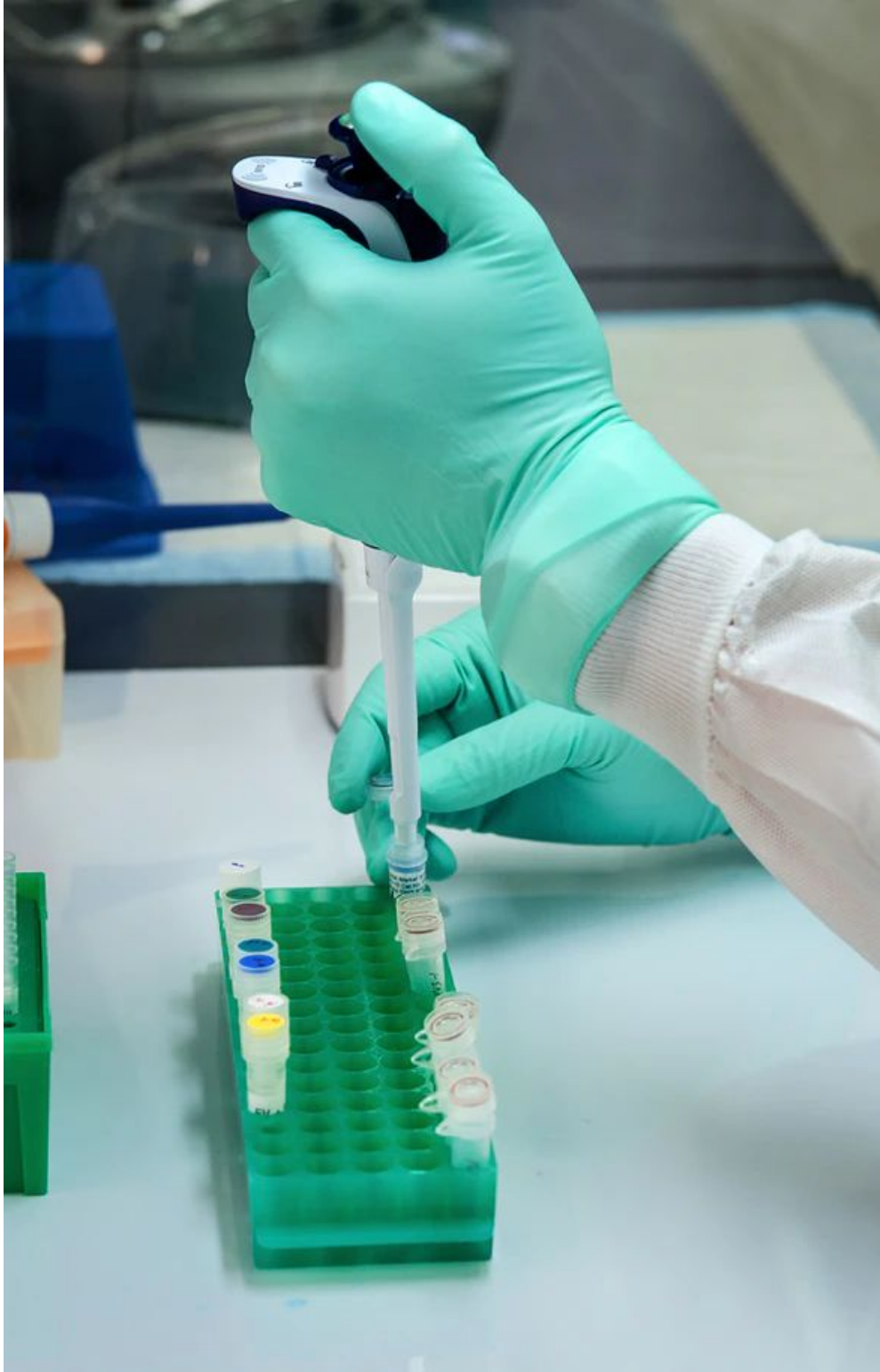
Sometimes even the best developer breaks the build.

Functional testing

Coming up with tests is your job.

There are a lot of good frameworks out there.

Remember they only run your tests, they don’t build them for you.



Functional testing

A testing framework is a collection of tools that help you build your Tests.

Your tests need to guarantee your software's optimal functionality.

Always think about functional testing.

Technique # 3 - Always exercise your code - Unit testing

Testing Methods - Unit testing

Testing methods

Manual testing- a developer or group of developers that manually start testing the code's functionality

Automated testing- If you have a more complex piece of software then it will be much better to perform automated testing.

Technique # 3

Unit testing

Unit testing is all about creating automatic tests for the smallest components of the code to test their business logic.

That is going to be essential to the development and the effectiveness of your testing.

Unit testing

Unit testing ties in together with the continuous integration process many companies have, whenever new code is submitted to source control the software will test the build and email people to let them know there is a problem with their code.

Unit testing vs System testing

Things may work great in isolation, but when different components of the code interact or when users use the system bugs can be overlooked.

System testing: hooking everything together and then treating it like a black box. It focuses on the overall functionality, making sure the system handles interactions well.

Technique # 4- Let your peers understand your code - Document it!

Technique # 4

Debriefing. Let your peers understand your code - Document it!

Have stand up meetings to

- Inform your customers of any changes you have made to the code
- Document your changes
- Analyze them along with your peers
- Keep them informed-reasons

Technique # 4

It’s important you become an expert communicator and standup meetings will be the place where you will be required to shine.

Find ways to adapt your speech depending on your listeners.

This is the end of this lesson.

Remember the concepts and definitions we have covered so far

REVIEW III

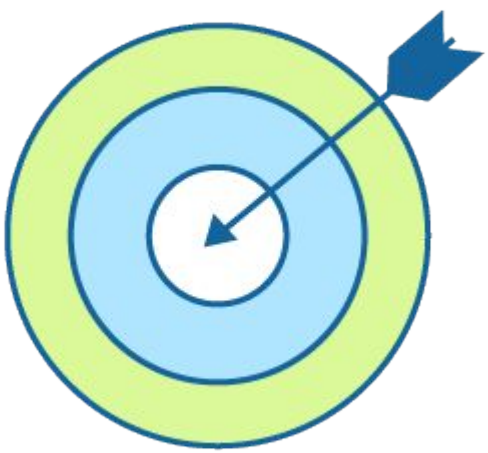
- Refactoring your design
- SRP Single Responsibility Principle
- DRY Don’t Repeat Yourself
- Refactoring and Stand up meetings
- Definition is done
- Ship out / Release software with quality and value
- Defend your software from yourself and your peers

REVIEW III

- Technique #1 - Version control with your code repository
- Technique #2 - Control your dependencies.
- Functional Testing
- Technique #3 - Always exercise your code-unit testing
- Technique #4 - Let your peers understand your code - Document it!

Understand Continuous Integration (CI)

Goals for this lesson



Understand

1. Black-box
2. Gray-box
3. White-box testing

Black-box testing

Black box Testing

There are different approaches to testing and different viewpoints.

Users see the system from the outside, they don’t look at the database tables they don’t evaluate the algorithms.

Black-box testing

Users want functionality

For an end user a software program is a black-box

- No code evaluation
- No database evaluation
- No algorithm evaluation
- Input-output
- State transitions (e.g. GPS)
- Boundary case errors (Month 13, day 32)

Gray-box testing

Gray-box testing

Testers are looking for functionality.

They do a little bit more of a deep dive to make sure things are really happening the way you said they would be.

This is gray-box testing.

Gray-box testing

Testers might be interested in

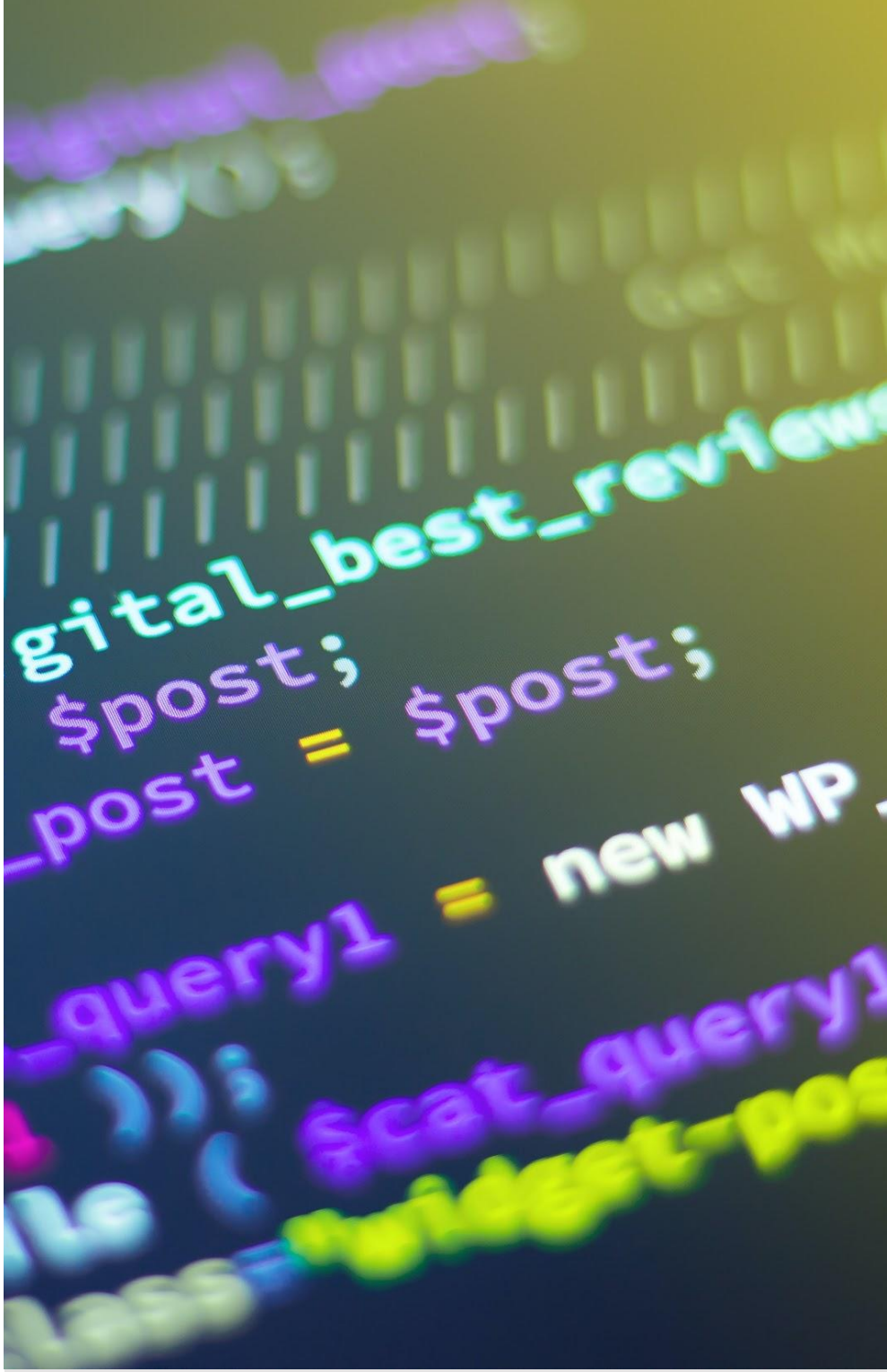
- Auditing, not available through user interface
- Data designed for other systems (output), delayed orders for example
- System added information
- Security risks, memory leaks, clean uninstalling

White-box testing

White-box testing

Testing done by other Developers

The deepest level of testing

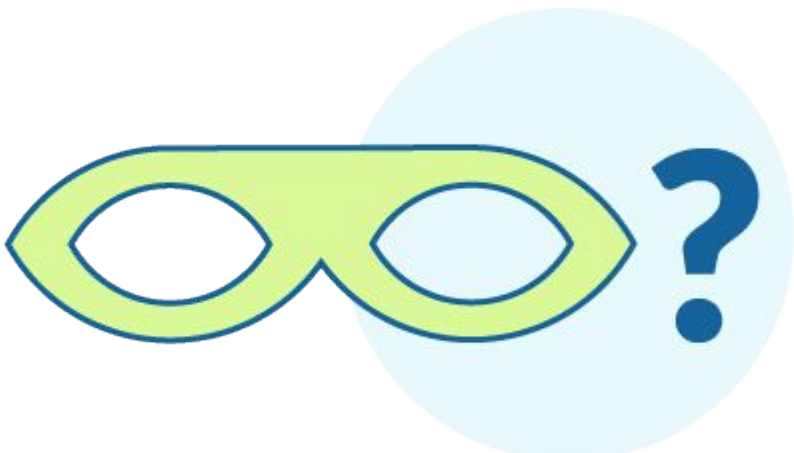


White-box testing

They really look at all the details and look for:

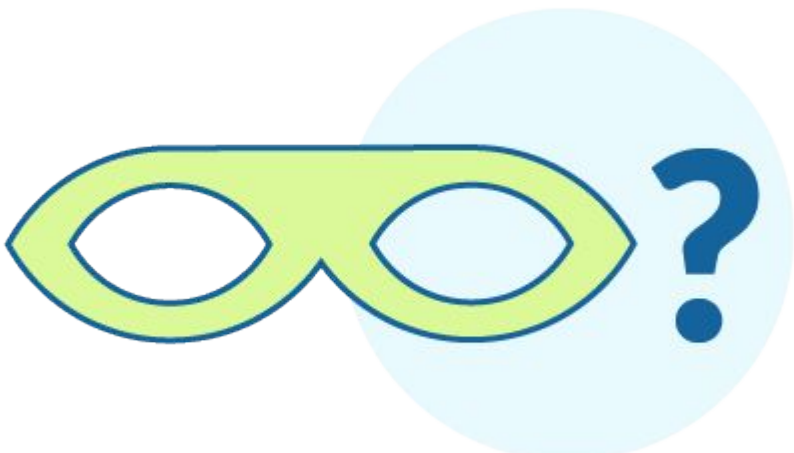
- Class designs
- Duplicated code
- Representation inconsistencies, etc.
- All the branches of code
- Error handling
- Code-on-code

Who am I?



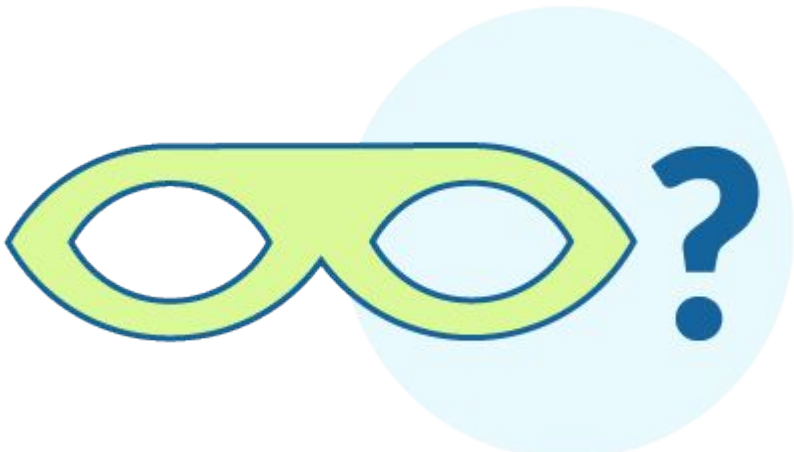
Let’s play a little game. Can you guess the concept if I give you some clues?

Who am I?



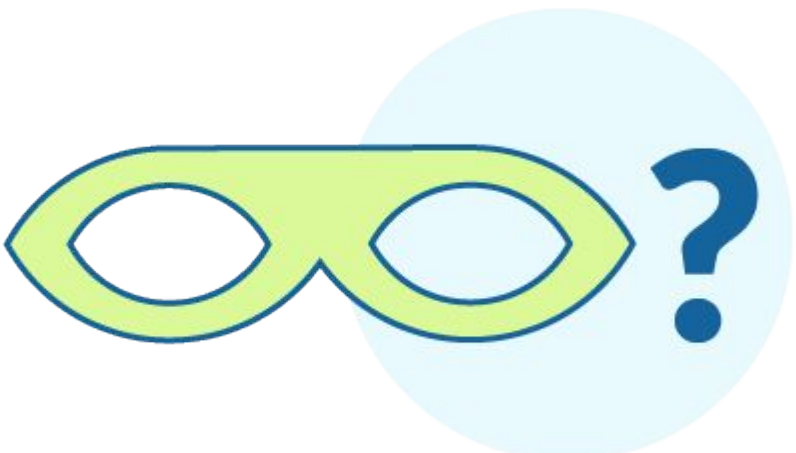
I am the kind of testing done by Developers and I am looking for glitches, bugs, and code duplicates...

Who am I?



I am the kind of testing done by end users, I am looking for functionality...

Who am I?

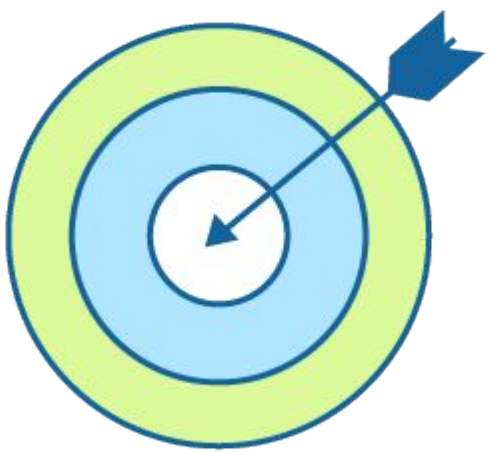


I am the kind of testing done by testers, I look at the database, and other software building principles, but I don’t go soooo deep...

This is the end...
...for now

Understand Continuous Integration (CI)

Goals for this lesson



1. Handle accidents when building the code
2. Continuous Integration (CI)

Handle accidents when building the code

Handle accidents when building the code

You are sure your code compiles, you have tested it and committed it into the repository.

But something happens and your code doesn't compile. So the next person who checks it out of the repository is going to be in trouble.

How's your Continuous Integration?

Handle accidents when building the code

There are many things that can go wrong with software development.

Can you think of one example of problems that might arise when developing new software?

Do you have the same problems when you are working with small or big teams?

Leave a comment

Continuous Integration (CI)

Continuous Integration (CI)

Continuous Integration is a technique that guarantees the reduction in the impact of conflicts by doing the following:

Core principles of CI

- Continuously submitting working, runnable software into source control that will be built and tested.
- Using automation to enable source control to build and test the latest versions of the code submitted by the developers.

Continuous Integration (CI)

- Continuous Integration tools run your tests as soon as you submit your code.
- Source control emails the developers last involved in conflicts that appear when submitting.
- Continuous Integration wraps version control, compilation and testing into a single repeatable process.

Continuous Integration (CI)

Source Control guarantees the impact reduction of conflicts.

The benefits of using Source Control:

-Avoid Merge Hell

-Fix bugs quickly, testable build

-Ensure Ownership

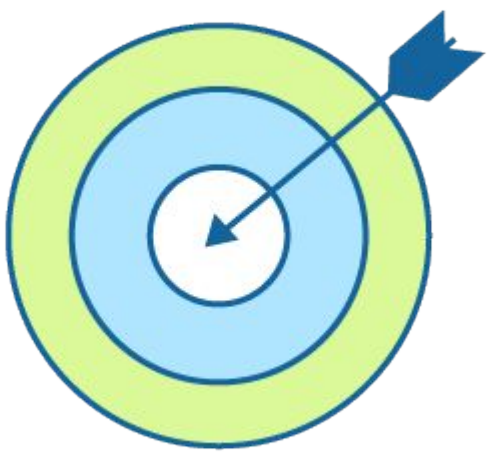
All done through constant communication with your fellow Developers.

All good things….

…must come to an end.

Test your software

Goals for this lesson



1. Test-Driven Development (TDD) is all about designing tests first, then code.
2. Make your code go from RED to GREEN
3. Never Skip Tests!

Test-Driven Development (TDD) is all about designing tests first, then code.

Test-Driven Development

TDD is all about designing tests first, then code.

Unit testing, system testing, black-box, gray-box and white-box testing, but what is the real philosophy behind all this?

Well, the idea is to think about testing as a driver.

Test-Driven Development

Testing is so important that you want to use it as effectively as you possibly can and to do this you need to follow the Test-Driven Development philosophy.

Test-Driven Development

TDD is all about writing tests before writing the code, and will allow you to have solid, specific ideas about what your software needs to do.

This gives you clarity.

Make your code go from RED to GREEN

Make your code go from RED to GREEN

Most of the time you will be writing tests before you even start writing code.

Those tests will fail the first time you run them and that is only logical.

Before we write any new code we need to write a failing test.

RED!

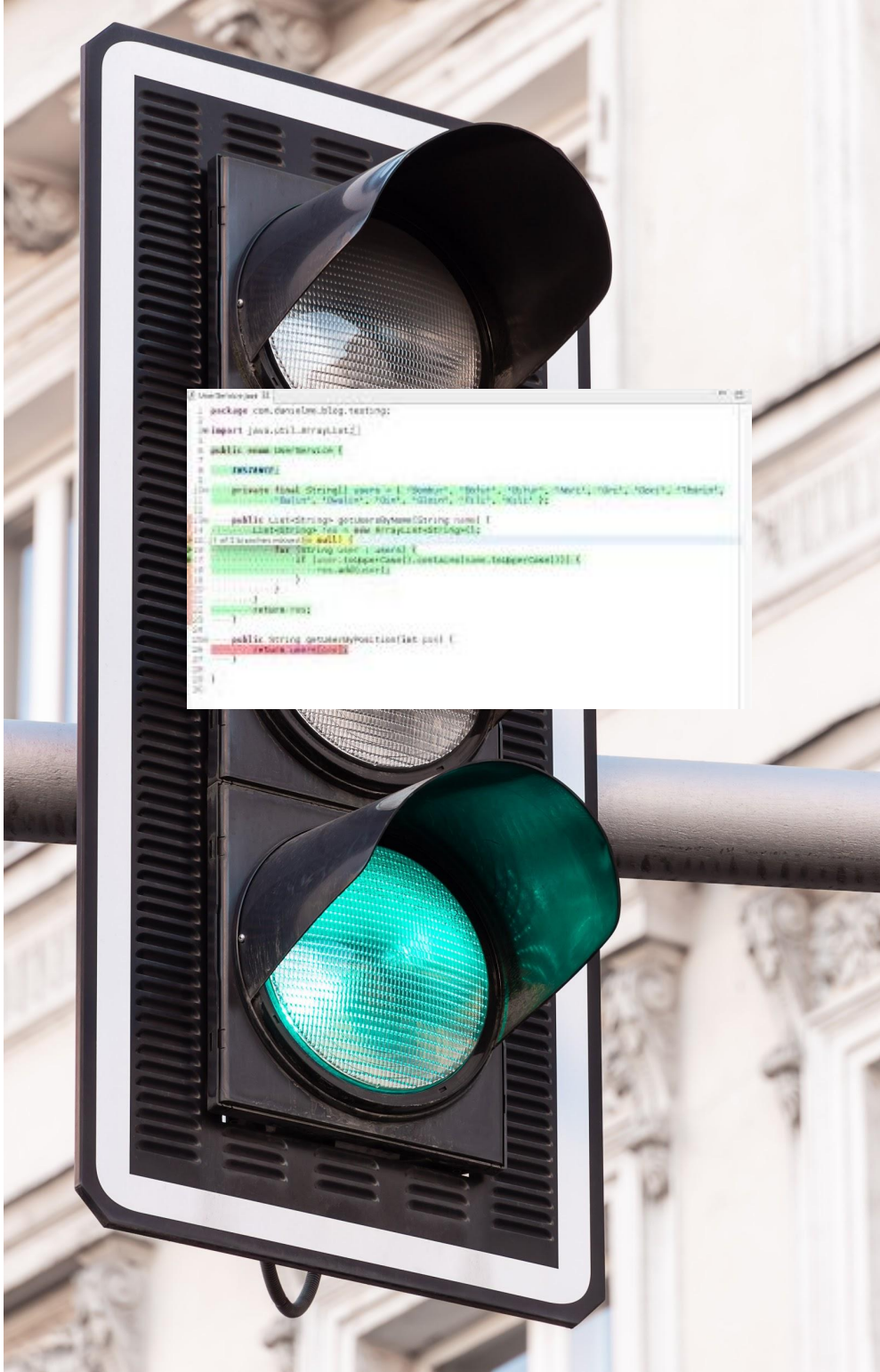
Make your code go from RED to GREEN

Your goal is to get your tests to pass. That is to go “GREEN”.

“GREEN” refers to the green bar that JUNIT / Visual Studio display when the test passes

RED= Failing

GREEN= Passing



Make your code go from RED to GREEN

Testing a single element promotes implementing the simplest code possible to make your tests pass.

RED-GREEN-REFACTOR!!

Never skip tests!

Never skip tests!

Remember to automate your tests whenever possible.

- Repetitive tasks
- People vs Computers

Never skip tests!

Do unit testing and system testing.

System testing exercises the functionality of the system from front to back in real world, black box scenarios.

This is the end of this lesson.

Remember the concepts and definitions we have covered so far

REVIEW IV

- Black-box testing
- Gray-box testing
- White-box Testing
- Handle accidents when building the code
  -

REVIEW IV

- Continuous Integration (CI)
- Test-Driven Development (TDD) is all about..?
- Make your code go from RED to GREEN
- Never skip tests!

Be ready for the end.
You are almost done.

Goals for this lesson

1. Correct and report your bugs!
2. Add more user stories to fix your bugs (Always be adjusting)
3. Reprioritize your user stories

Correct and report your bugs!

Correct and report your bugs!

Eventually your testers are going to find a bug. Actually they'll probably find lots of them.

SO What happens then?

DO you just report the bugs to the team and fix them? Cycle?

Correct and report your bugs!

The actual life cycle of a bug:

- A tester finds the bug
- The tester files a bug report
- You create a story to fix the bug*
- You fix the bug
- Check the fix and verify that it works
- Update the bug report
- Reprioritize**

Add user stories to fix bugs

Add user stories to fix bugs

Now you are sure you have bugs in your code.

- Fixing your bugs will take some time
- This creates extra tasks
- Remember all the tasks need to be....

Add user stories to fix bugs

All that time needs to be taken into consideration when estimating the length of your iterations…

More User stories=
more tasks=
more work=
more time.

Reprioritize your user stories

Reprioritize your user stories

The time has come to reprioritize your stories based on the bugs.

Reprioritization is critical because you want to make sure you are doing the right thing on the project at all times.

Reprioritize your user stories

Constant communication is key.

Reprioritizing user stories will call for adjustments on the fly or once iterations have finished.

Take this chance to adjust, adapt and overcome.

Reprioritize your user stories

Getting feedback and recommendations from other developers can truly make a difference.

Ask questions like:

What do you suggest we do here?

What are your thoughts on this?

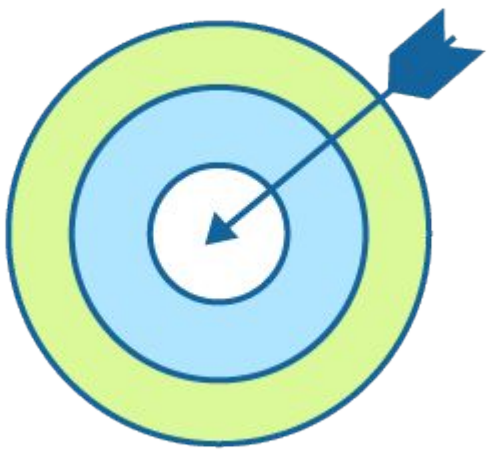
What do you think should be our number one priority right now?

BONUS Retrospective session PDF

This is the end...
...for now

Be ready for the end.
You are almost done.

Goals for this lesson



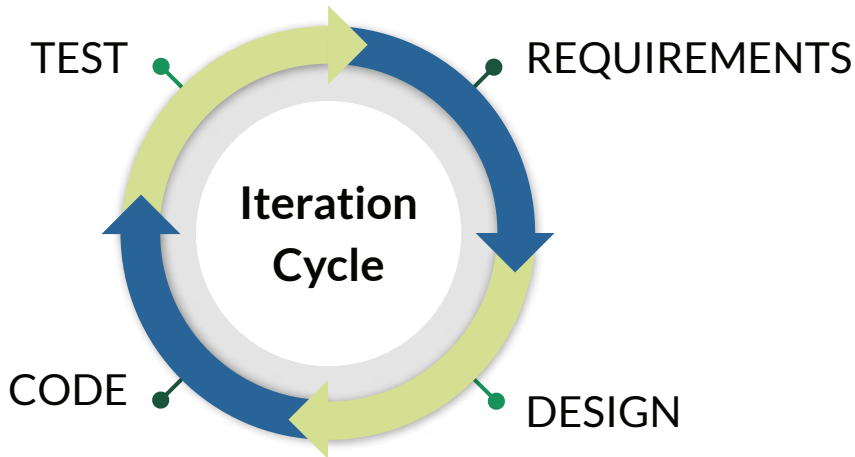
1. Plan your next iteration
2. Estimate again and learn from your data (VELOCITY)

Plan your next iteration

Plan your next iteration

The end of your Iteration is here.

You have already finished the work for this iteration so it's time for the retrospective session.



Iteration Length?

30 Days?
45 Days?
60 Days?

30 Days!!!!!

Plan your next iteration

It’s important you have a standard set of questions to review, think about the future, calculate your metrics.

What questions do you think you can ask after the Iteration?

Plan your next iteration

Ask questions like:

What did we do well in this Iteration?

What are our takeaways after this Iteration / Sprint?

What are our areas of opportunity and how can we improve?

What are our main pain-points?

Plan your next iteration

Once that is done, start again.
Follow the standard procedure for Iterations.

- Speaking to the customer
- Analyzing the requirements
- Coming up with user stories
- Estimating user stories--learn from your VELOCITY
- Playing planning poker to reach consensus
- Start again

Estimate again and learn from your data (VELOCITY)

Estimate again and learn from your data (VELOCITY)

Calculate your metrics and make sure you are readjusting after the iteration, your team may be working faster than you thought or more slowly.

Perhaps you could check your task breakdown approach.

Adjusting is good.

Initial Project Velocity

=

0.7

Estimate again and learn from your data (VELOCITY)

You have finished an Iteration and your team..

...Finished in a rush.

...Finished early.

...Finished late.

What would you do in these cases?

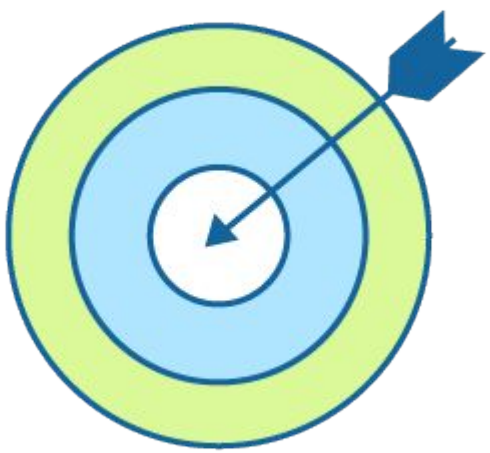
Leave your comments in the chat

All good things….

…must come to an end.

Fix your bugs!

Goals for this lesson



1. The Lifecycle of a bug
2. Release your code to the real world
3. Continuous delivery - fix functional bugs
4. Estimate bug- fixing effort

The life cycle of a bug

The life cycle of a bug

You have now started your next Iteration, built a new backlog, adjusted your burndown rate, your velocity but….

You find BUGS!!!!!

Eventually your testers are going to find a bug. Actually they'll probably find lots of them.

What happens now?

The life cycle of a bug

The actual life cycle of a bug is as follows:

- A tester finds the bug
- The tester files a bug report
- You create a story to fix the bug
- Fix the bug
- Check the fix and verify that it works
- Update the bug report

Continuous delivery - fix functional Bugs

Continuous delivery - fix functional bugs

It is necessary to create user stories to fix bugs.

Fixing bugs takes time and all that time needs to be taken into consideration when estimating the length of your iterations.

Continuous delivery - fix functional bugs

Fix functional bugs first!

Remember that whenever you get bugs you just need to fix the Bugs that will affect the functionality that you need
In other words only fix code to fix your user stories.

Continuous delivery - fix functional bugs

Everything revolves about customer-oriented functionality
Only fix what is broken, and you know what is broken because you have tests that fail.

Writing beautiful, perfect code is great but writing good-enough code is better because it pays the bills.

Spike test

Spike test

Spike testing is a period in which there is an “explosion” in the testing activities.

Test continuously, take random bits of code and functionalities to ensure functionality before delivery.

Spike test

Weed out bugs, later you can use the results to estimate adjustments.

1. Take a week to do your spike test
2. Pick a random sample from the tests that are failing and try to fix just those tests.

Act based on the results of the Spike testing effort. At the end of the week calculate your bug fix rate.

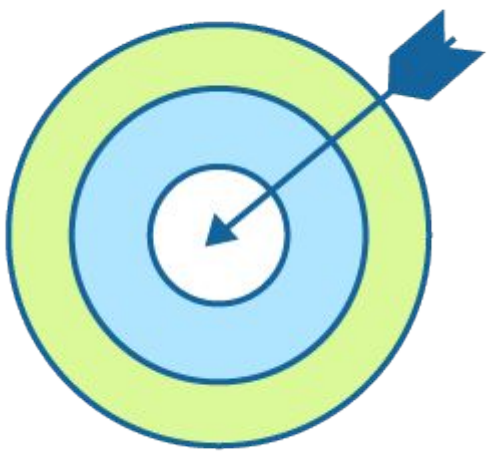
Fix functionality only

BONUS
Bug fix rate
PDF

This is the end of this lesson

Fix your bugs!

Goal for this lesson



Continuous delivery as a method for functionality

Continuous Integration Test Delivery Method

Continuous Integration Test Delivery Method

Continuous Integration Test Delivery is a cloud enabled fast delivery method for functionality.

It works better when two or more people are working on code, and committing new code to source control.

Frequency ensures fast response when a bug comes up.

Continuous Integration Test Delivery Method

This method is really just a process for delivering functionality.

It lets your team be successful because you are delivering functional software that has been tested and delivered on time, with quality that is present in all your processes.

Continuous Integration Test Delivery Method

Remember this very counter - intuitive principle of software Development:

If it hurts do it often and it won't hurt as much.

Beautiful code is nice but tested and readable code is delivered on time.

Real success comes from delivering functionality.

This is the end...
...for now

Remember the concepts and definitions we have covered so far

REVIEW V

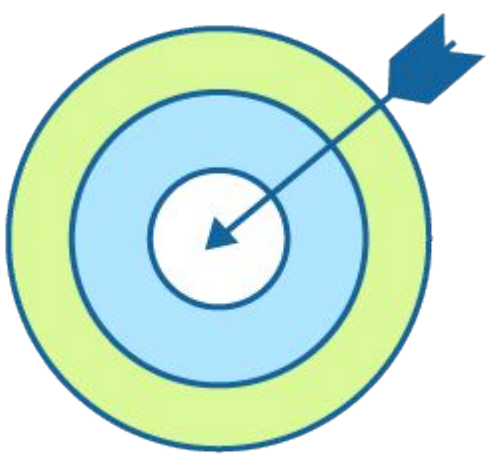
- Correct and report your bugs!
- Add more user stories to fix your bugs (Always be adjusting)
- Reprioritize your user stories
- Plan your next Iteration
- Estimate again and learn from your data (VELOCITY)

REVIEW V

- The lifecycle of a bug
- Release your code to the real world
- Continuous delivery - fix functional bugs
- Estimate bug- fixing effort
- Continuous delivery as a method for functionality

See the real world! / Review

Final Review



1. The definition of a software development process
2. Developing iteratively
3. The backlog technique
4. The user stories technique
5. The version control technique
6. The continuous integration technique
7. Test-Driven Development

The definition of a software development process

A structure imposed on the development of a software product that produces great, functional software.

Developing iteratively

Working following mini project cycles - Iterations, to promote customer feedback and deliverability.

The backlog technique

It's a big board in your office where you can track the progress of your User stories and tasks.

There are three task classes on the backlog

1. To do
2. Doing
3. Done

   Opt. Burn down rate /Calorie Burning Rate

The user stories technique

A User Story is the representation of an individual task the software has to do, it is composed of smaller tasks and it contains a title, a description, an estimate, and a priority value.

The version control technique

VC is a defensive tool (multiple access repository) that keeps track of the different versions of your code.

The continuous integration technique

It's a technique that guarantees the reduction of conflict impact by putting together version control, compilation and testing into a single repeatable process.

Test-Driven Development

TDD is all about writing tests before you even start writing the code that will give the software its functionality, and will allow you to have clear, solid, specific ideas about what your software needs to do.

The end

Final exam