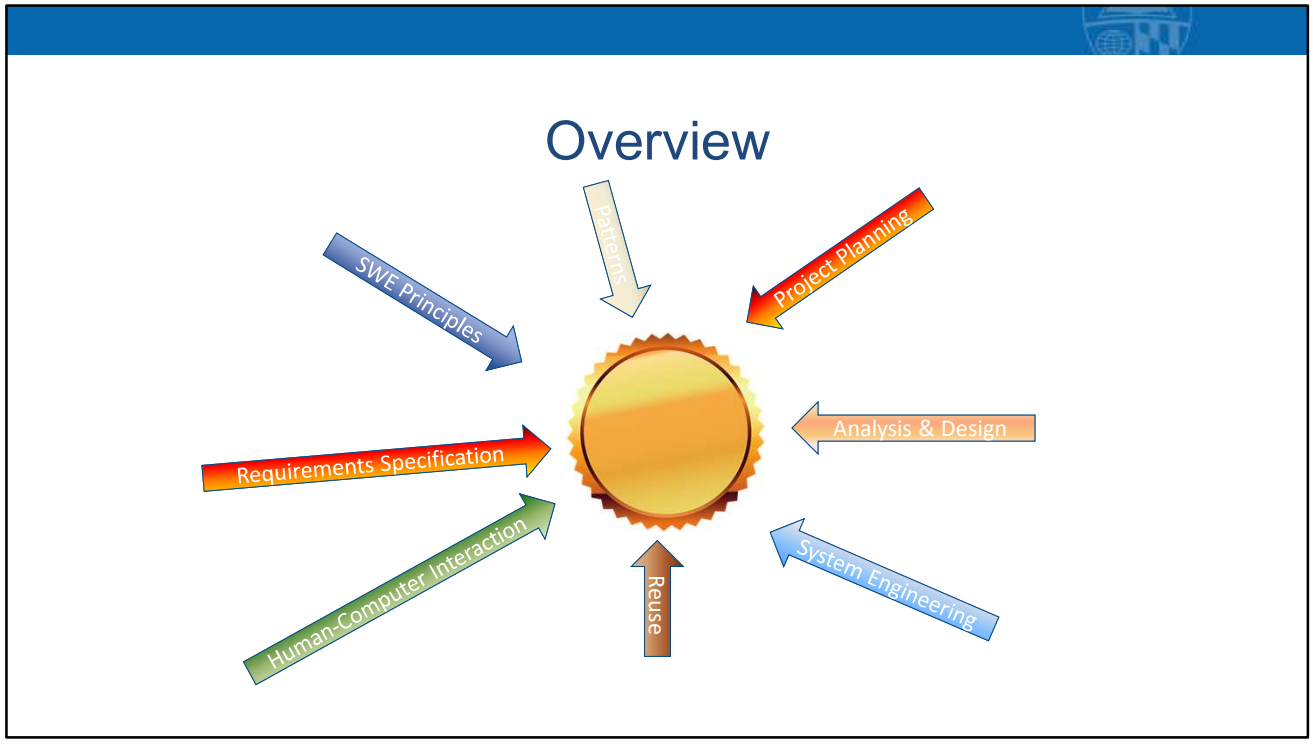
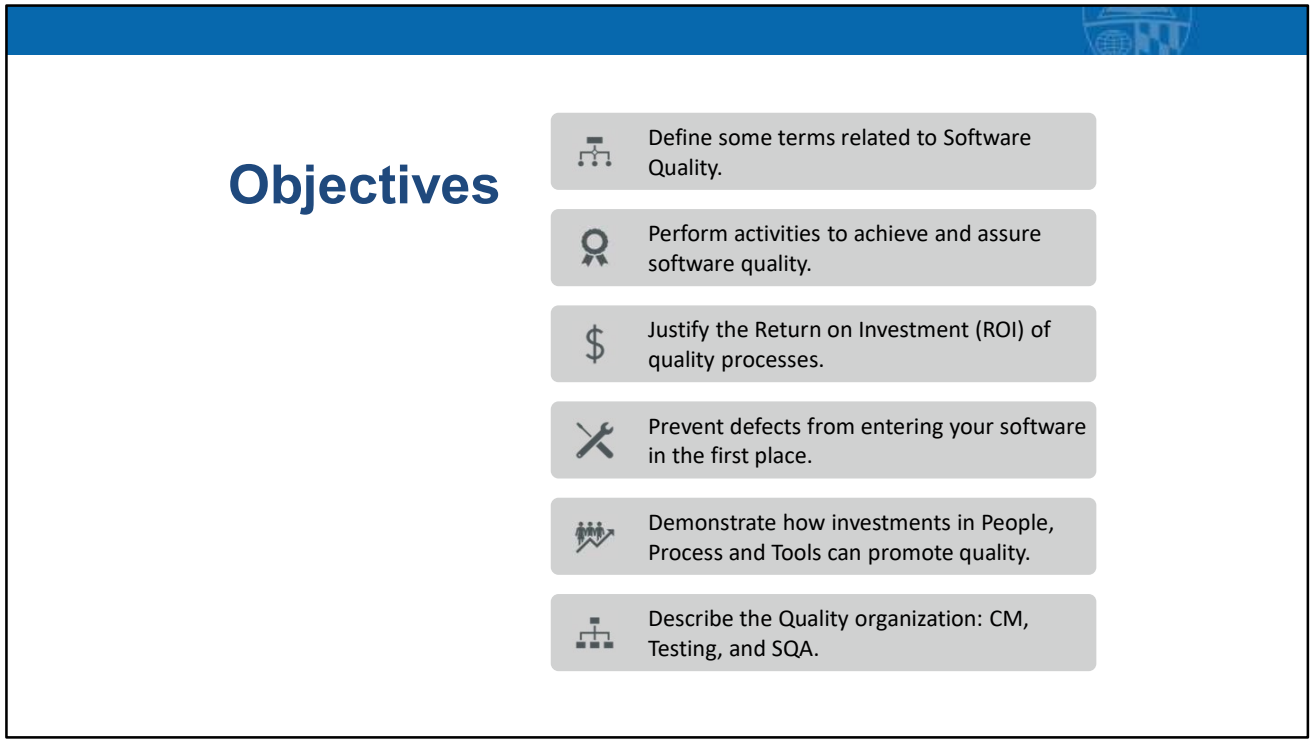
# Software Quality

Welcome to the module on Software Quality.




Everything we have been talking about all semester has had something to do with software quality. This is perhaps the most central and important topic of the course. We will discuss the meaning of quality, and how to achieve and how to assure it. We will then turn our attention to defects: where they come from, what to do with them, and how to prevent them. We then discuss the Quality Triangle: People, Processes, and Tools. We conclude with a discussion of software quality assurance.




Here is what you should be able to do upon completion of this module:

- Define some terms related to Software Quality.
- You should be able to perform activities to achieve and assure software quality.
- Considering that activities to achieve and assure quality are an investment, you should be able to demonstrate the Return on Investment (ROI) of performing these quality processes.
- You will be able to demonstrate techniques to prevent defects from entering your software in the first place.
- You should be able to show how investments in People, Process and Tools can promote quality.
- Finally, you should be able to describe the Quality organization: CM, Testing, and Software Quality Assurance (SQA).

# Outline

- Software quality
  - What is it? How to get it?
- Defects
  - What to do about them?
- The quality triangle
  - People, Processes, Tools
- Software Quality Assurance

There are four more videos in this module.

First, we will examine three questions:
1. What is software quality?
2. How do we achieve it?
3. How do we assure it?

Then we will discuss defects and what to do about them.

Next, we will talk about the quality triangle: People, Processes and Tools. The idea is that investing in any of these three dimensions will result in improved quality.

Then we will zoom in on people, processes and tools that relate specifically to software quality assurance.

## Next

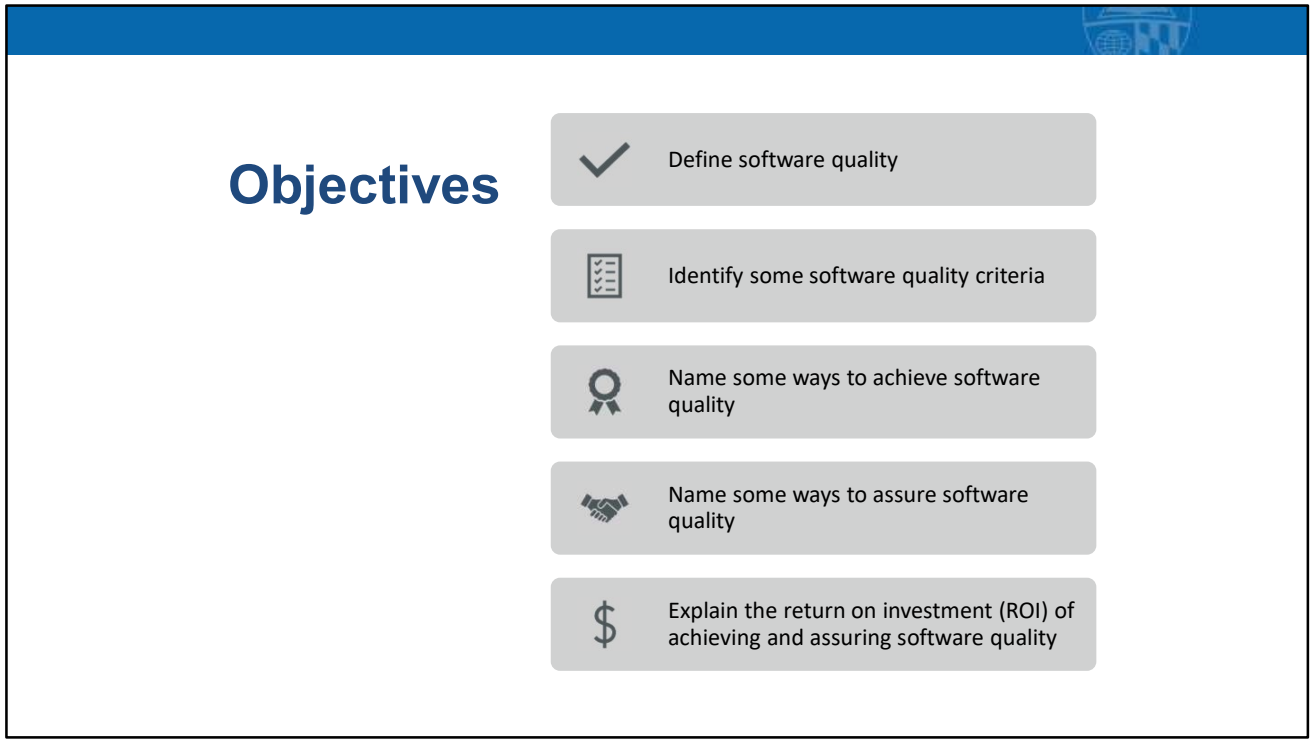
- Software Quality
  - What is it?
  - How do we get it?

There's a lot here, so let's get started….

# Software Quality

## What is it?
## How do we get it?


This is the first video on software quality. We will be discussing the definition of quality and ways in which we can achieve it in our software




More specifically, here are the things we expect you should be able to do once we are done here.

You should be able to define software quality.

You should be able to identify some of the criteria that we are looking for in quality products. These are similar to the software engineering goals we discussed in the first module of this course.

You will be able to name some ways in which we can achieve software quality. You will also be able to name some ways to assure ourselves and others that we are building a quality product.

Lastly, you will be able to explain why it is a good investment to do the things to achieve and assure quality.

## What is Quality?

- One definition: Meets requirements
- Better definition: Satisfies the needs of the customer


The traditional definition of quality is “meets requirements.”

This was useful for situations in which the requirements were stated in terms of things that could be measured. There would typically be quality inspectors at various points in the assembly line. If the inspectors noticed that things were beginning to get out of spec, they could shut the line down so they could figure out what was going wrong.

Consider the manufacturing of an automobile or a desktop computer. These types of products are made up of lots of subsystems and parts, each of which must adhere to a specification. The parts can be manufactured by several different vendors, so adherence to the specification is important.

Similarly, most software is made up of subsystems and components, each of which must interoperate with the others. So, specification of the interface and behavior of the components and subsystems is important. These specs are not measurable because they are not physical, but they can be inspected and tested for conformance. We will revisit this later in the course when we cover software reuse.

Now, what about the requirements for the entire software system product? As we have seen, these requirements are usually written in the form of stories or use cases. We can certainly perform tests to ensure that the software does what we say it should do.

But maybe the requirements are wrong.

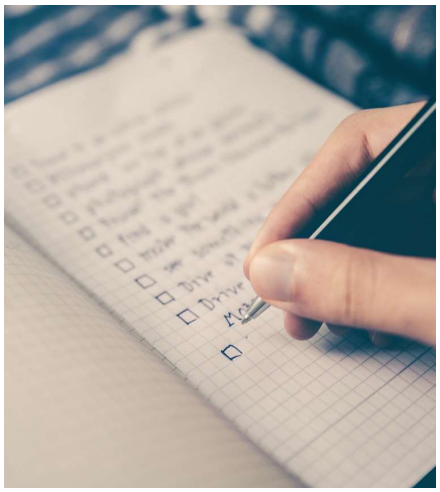
It is important to consider the goals of customers or other stakeholders before thinking about the software requirements.  There should be traceability between the requirements and the goals of the stakeholders.

In that way, we will have a better chance of building a product that satisfies the needs of the customer or other stakeholders.

This is the reason we write Vision documents.

## Some Software Quality Criteria

- Correct
- Efficient
- Flexible
- Robust
- Maintainable
- Reliable
- Reusable
- Testable
- Usable
- Understandable



Remember the first module of this course when we were discussing the software crisis and the goals of software engineering? This list is an expansion of the list of goals we had in that module.

You will recognize most of these as non-functional requirements. It is possible to write non-functional requirements in a measurable way, so it should be possible to write test cases that show that the software meets them.

## Achieving Software Quality

- What can we do to build quality into our product?
  - Materials
  - Workers
  - Standards and conventions
  - Tools

Now we will address two categories of actions: Achieving and Assuring quality. They both start with A, so we need to distinguish between them.

Achieving quality means doing things that build a quality product. Assuring quality means convincing ourselves and others that we are building, or have built a quality product.

Here are some things we might do in order to achieve quality.

We ought to build our product out of high-quality materials, components or subsystems. We want to avoid any component that could be a single point of failure for the entire system.

We should use skilled workers to build our product. More experienced or highly trained workers may be a bit more expensive but in the long run they can save you (as we shall see in a couple of minutes).

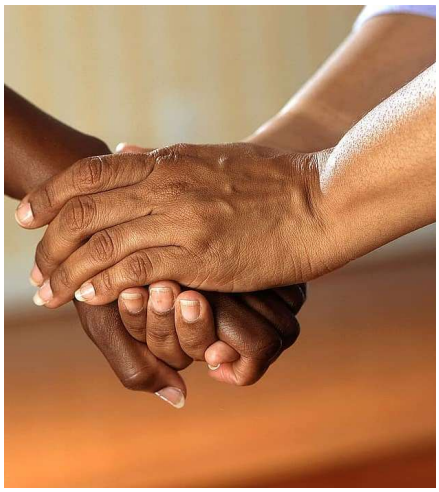
They should follow good practices, standards and conventions. For programming, this might entail following coding conventions.

Using good tools can both make it easier to build our product and enable a higher quality production as well.

Note that these suggestions apply to building cars, computers, and houses, as well as they do to building software.

## Assuring Quality

- How do we know we are building a quality product?
- How do we convince others?
  - Testing
  - Reviews
  - Processes and standards



Assuring quality means convincing ourselves and others that we are building a quality product.

We will cover testing in a later module of this course. For now, let us agree that as we run tests, if we are encountering fewer and fewer defects, that this shows that quality is improving

What do we do before we have enough software in place to do testing? We can run inspections and reviews. We can review requirements, designs, code, and even testing scenarios. We should also perform reviews of our processes and standards.

Following standards and conventions and other processes not only can help us achieve quality, but the mere fact that we have good processes in place, and that they are being followed, gives us assurance that we are more likely to build a quality product.

Putting this another way, you could say, “If you’ve got quality and you know it… clap your hands” <clap> <clap>.

# Return On Investment (ROI)

- There is a cost to achieving and assuring quality
- What is the cost of not doing these things?

Now wait a minute. Isn't all this going to drive up my costs and take more time to develop my product?

Well, yes, there is a cost to doing the things we talked about. And yes, it does take time to perform them.

But what is the cost of not doing these things?

One of the obvious costs is scrap and rework. If you are manufacturing small parts such as computer chips and they start failing their tests, it is cheaper to just scrap the defective pieces. This is a cost. If we are building larger systems, such as TVs or computers, and they fail the tests, it is more cost effective to send the defective products be repaired. These refurbished products are usually sold as refurbished, or what Amazon calls renewed, at a discounted price. There is a cost to fix the product, combined with the loss of profit on the discounted price, but the cost penalty is not as high as what you would have if you scrapped the product.

If we continue to sell defective products, we will get low customer satisfaction ratings. This will result in a loss of business.

Some companies have even had to go out of business because of poor performance.

What if you get sued? Once you get the lawyers involved, that can get really expensive.

But wait, it gets worse. What if your defective product ends up doing something bad such as destroying property, injuring someone, or even killing them? You are now going to have even bigger legal problems.

Hopefully you will agree that the cost of doing things to achieve and assure quality is much less than the cost of not doing them.

Quality pays for itself.

## Next

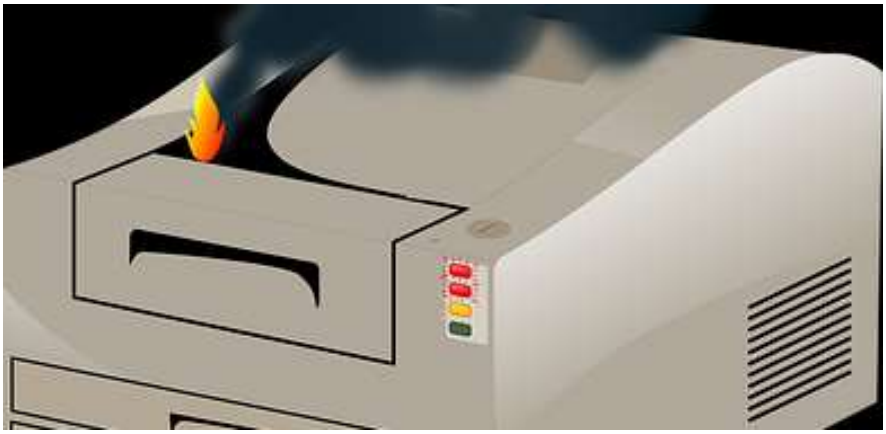
- Defects

Coming up in the next video is a discussion of defects.

One thing I think we can agree on is that defects are bad.  Not having defects is good.

There's more to it than that, but that's the bottom line.

# Defects



In this video, we will discuss defects. Where do they come from? What do we do with them? How do we prevent them?

# Objectives

- Upon completion of this video, you should be able to:
  - Define the terms defect, error, fault, and failure
  - Describe ways of finding defects
  - Classify defects into categories
  - Prevent defects from entering your work products
  - Explain what happens when we remove defects

Here's what you should be able to do once you've finished this video.

You should be able to define the terms defect, error, fault, and failure.

You should be able to describe ways of finding defects.

You should be able to classify defects into categories.

You should be able to prevent defects from entering your work products.

You should be able to explain what happens when we remove defects

## Some Definitions

- Error: A conceptual, syntactic or clerical discrepancy which results in one or more faults in the software.
- Fault: A specific manifestation of an error. A discrepancy in the software which can impair its ability to function as intended. An error may be the cause for several faults.

Glossary of Software Engineering Terminology, IEEE Std 610.12-1990

https://softwaretestingfundamentals.com/error-defect-failure/

Let's first agree on the definitions of several terms.

According to the Gossary of Software Engineering Terminology, the difference between an error and a fault is the fault is the bug in the code and the error is the bug in the process that generated the code.

For example, if a programmer misunderstood how to use a certain feature of the programming language or IDE tool, that one misunderstanding is an error that could possibly manifest itself as multiple faults in the software. These faults would need to be individually detected and repaired.

# More Definitions

- Failure: A software failure occurs when a fault in the computer program is evoked by some input data, resulting in the computer program not correctly computing the required function in an exact manner

2Lloyd, D.K., and M. Lipow, Reliability, Management, Methods and Mathematics, 2nd Edition, published by the authors, 1977

- Defect: Either a fault or discrepancy between code and documentation that results in mischief in the testing, installation, maintenance, or use of software

Dunn, Robert, and Ullman, Richard, Quality Assurance for Computer Software, McGraw-Hill, New York, 1982

When we run software that contains faults, sometimes it will fail. This failure may be as minor as computing the wrong result or as major as bringing down the entire system.

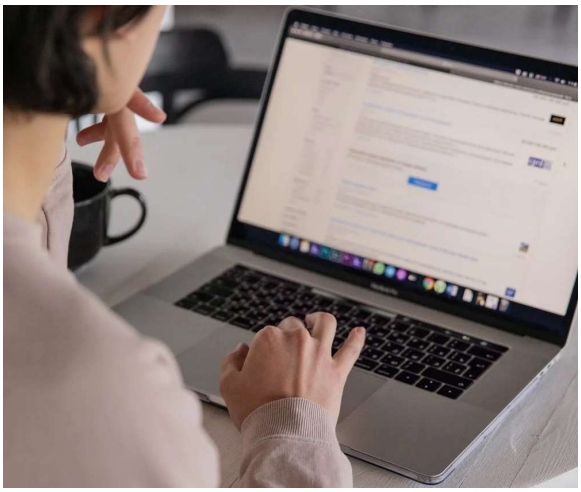
Sometimes we use the term, defect, in place of fault. The definitions are very close.

## Errors Lead to Defects Lead to Failures


To summarize,, errors cause defects, which, in turn, can result in failures.

# Finding Defects

- Ways to find defects:
  - Reviews
  - Walkthroughs
  - Formal Inspections
  - Testing
  - Bug reports



Because software is written by humans, there is always the chance that defects will be introduced into the code.

We have devised multiple ways to detect defects.

Reviews are performed by a group of people, including the developer of the code. There are multliple levels of formality in these reviews. An informal review is called a walkthrough. We basically walk through the design or code in the group. Usually, the developer is the one leading the walkthrough. Because there are multiple sets of eyes looking at the work product, there is a higher probability that if there is a defect, someone will spot it.

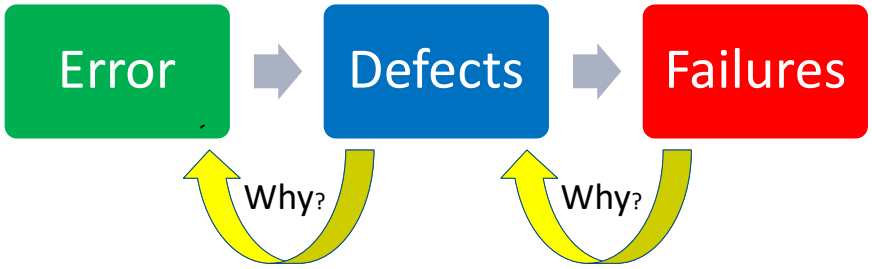
A more formal type of review is a formal inspection. These types of reviews are performed under well defined procedures and rules. The members of the inspection team have assigned roles to play. The number and types of defects found are analyzed as Quality Assurance data.

Much of the testing we do on our software has the objective of finding defects, particularly at lower levels of testing.

All of this is done internally. What we don't prefer to see is for the customer to find defects. When they submit a bug report, what they are describing is a failure. It is up

to the developers to figure out what defects led to the failure.

# Finding Errors – Root Cause Analysis




What we really need to do is to figure out why the defects happened in the first place.

This is called root cause analysis. Basically, we just keep asking the question, "Why?"

Starting with a failure, we ask why did it happen? What was the defect that led to the failure?

Then we ask, why did the defect occur? What error led to the defect?

Here's a little example. Suppose an airplane is flying and loses part of its rudder. The pilot is a hero and is able to land the plane safely, What happens next?

Well, if the airplane was run by a software company, we might just duct tape the broken part back together and continue flying the plane and hope nobody notices.

What the airine does, though, is figure out what caused the part to fail, fix it, and then inspect all other aircraft of the same type. This may require grounding the planes until this inspection is done, depending on the severity of the failure. If necessary, if the same defect is found in other planes, they would be fixed as well.

But the process doesn't stop there. We would ask the question, "Why did the defect occur in the first place?" What was the error that led to the defect? Suppose the part

broke because of rusted bolts.  What could cause the bolts to rust?  Maybe it was due to environmental conditions.  The planes weren't designed to fly in salty sea air.

Okay, why did we use bolts that could rust?  What was wrong with our design process that we didn't ask this question.

For a real world example of this, look up what happened after the space shuttle Challenger blew up soon after launch.

# Handling Defects

- What to do once a defect is found:
  - Record and categorize the defect
  - Identify and record the error
  - Look for similar defects and errors

We have this tendency to hide our defects and hope nobody notices them.

A better approach would be to acknowledge the defects and errors. We should look for patterns of defects and errors. If we keep making similar mistakes, we need to modify our process to avoid making these errors.

Maybe we need checklists to make sure we do all the steps in the right order Maybe we need refresher training on languages or tools. Maybe we just need to beef up our review and testing processes.

## Categories of Defects

- Nature
  - functional, performance, usability, compatibility, security
- Severity
  - Critical, High, Medium, Low
- Priority
  - Urgent, High, Medium, Low

Sidorova, Tatyana. “Software Testing Basics: Types of Bugs and Why They Matter" ScienceSoft, 29 June 2020, https://www.scnsoft.com/blog/types-of-bugs

When we record our defects, it is helpful to categorize them. This will help us when we are looking for similar defects.

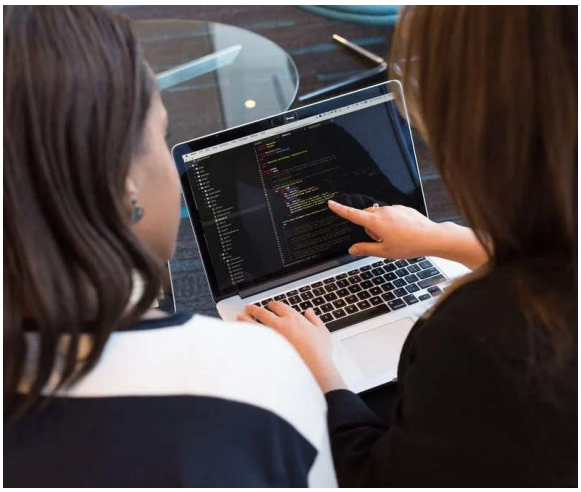
One type of categorization is by the nature of the defect. Examples are functional, performance, usability, and so forth.

We also need to identify the severity of the defect. What is the severity of the failures caused by the defects? Does it shut things down completely, or is it simply a minor annoyance, or somewhere in between?

When we assign a priority we are deciding the timing of repair of the defect. Urgent defects wil be repaired as soon as possible. The repair of low priority defects can wait until we have time.

# Removing Defects

- Change control
- Regression testing
- Ripple effect



When we fix a defect, we are changing the software. These changes need to be made in a controlled manner. There needs to be a strict change control process in place. You don't want someone going in and making unapproved changes.

If you make a change to the software, you need to re-run some of the test cases to make sure you haven't accidentally broken something else. This is called regression testing.

We need to do a careful job of analyzing the changes we make to the software to avoid ripple effects. You change one thing and something else breaks. You fix that and something else doesn't work. It's like a game of whack-a-mole.

# Preventing Defects

- Defect tracking and cataloging
- Continuous Process Improvement

George Santayana said, “Those who cannot remember the past are condemned to repeat it.”

To avoid making the same defects over and over again, we need to figure out the errors that lead to the defects and avoid the errors.

We have aready discussed defect tracking and cataloging and root cause analysis.

A good way to avoid errors is to modify our processes and procedures. This should be done on a continuing basis. This is referred to as continuous process improvement.

If we don’t’ continually improve our processes, we risk making the same mistakes over and over again.

There is a tremendous return on investment of doing this.

It’s a simple idea, embodied in Santayana’s aphorism as well as other wise sayings:

An ounce of prevention is worth a pound of cure.

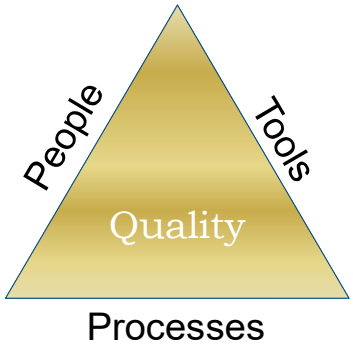
A stitch in time saves nine.

## Next

- The Quality Triangle: People, Processes, Tools

In our next video, we will examine what we call the Quality Triangle: People, Processes, and Tools

# The Quality Triangle



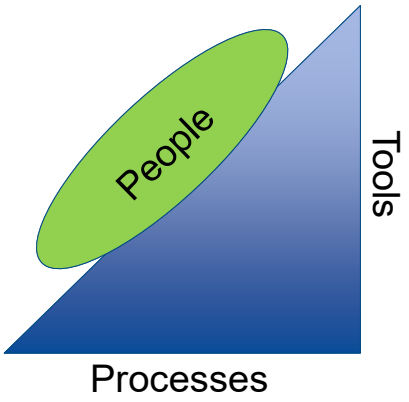

There are three things that we have control over that can lead to quality software: People, Processes, and Tools.

If we invest in any one (or all) of them, we should see increased software quality.

I've depicted this relationship as a triangle. The area inside the triangle represents quality. The lengths of the sides of the triangle represent the amount of investment we have placed in people, processes and tools.

We can increase the area inside the triangle by making any of the sides longer.

# People




Let's start with People.

Software is a labor-intensive operation. It is difficult to automate, although we have been trying to figure that one out since the mid 70's.

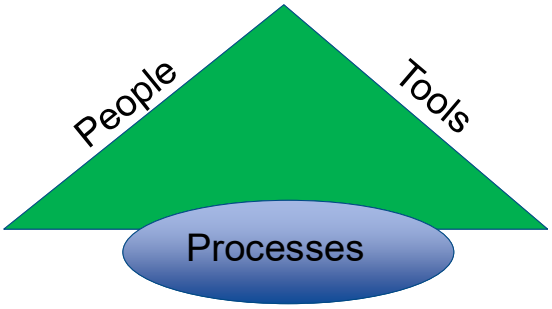
- If we invest in our people, we will enjoy higher software quality.
  - Hire good people.
  - Retain good people.
  - Train your staff to keep their skills up to date.
- These things come with a cost, but they should lead to a positive return on investment (ROI).

# Types of Jobs

What people (sometimes called human resources) are we talking about?

- Management:
  - Project managers.
  - People managers.
- Technical staff:
  - Analysts.
  - Designers.
  - Programmers.
- Quality Assurance Function:
  - Testers.
  - Configuration Management.
  - Software Quality Assurance.

# Processes




An investment in processes can lead to higher software quality. We discussed the benefit of repeatable, defined processes earlier in the course.

## Processes and Standards

- Document Best practices
  - Based on our own experience
  - Based on industry standards
- Maintain
- Train
- Ensure we are following the standards
  - This is the role of SQA

We need to document our processes and standards.

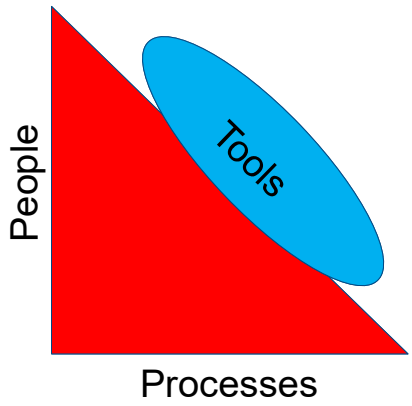
- Base our processes and standards on best practices. We can do this based on our own experience. What have we found that works well in the past? Capture that experience and share it. There are also plenty of government and industry standards that we can use as the basis for our standards and practices.

We should maintain our processes (encourage the things that work, eliminate what doesn't work). Put the processes and standards under configuration control just like software, ensuring that we make changes to the standards in a controlled way.

We should train people in the standards and processes, so they know the correct way to do things.

We should ensure they are following the standards. This is the role of Software Quality Assurance.

## Tools




Finally, we can invest in tools. Now we're talking about software tools here, but the principle is the same as for any tools. For example, carpenters use power tools such as circular saws to help them do their work. In either case (software or carpentry) the use of tools and automation have many benefits:

## Tools -- Benefits

- Reduces errors
- Increases productivity and efficiency
- Makes job easier
- Saves money in the long run

They reduces errors If we accept the premise that people make mistakes, the less that people are involved in the production of things the less likelihood of errors leading to defects. (This is one reason why manufacturing has invested so heavily in robots.)

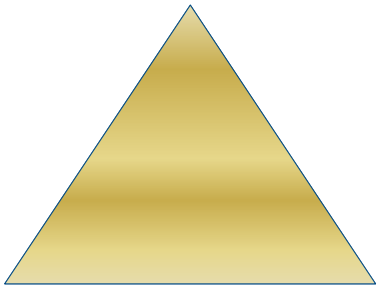
Tools Increases productivity. (It takes less time to cut a board using a power saw than with a hand saw.) We can get more work done in a shorter amount of time. This is another reason why manufacturing has invested in robots. Robots can perform tasks with perfection in shorter time than humans.

Tools make the job easier: Let's let the tools do more of the work. Think about how much effort it would take to build today's kind of software if we only had assembler language to work with. High level languages shift a lot of the effort of programming from the programmer to the compiler.

Tools save money in the long run. If we weigh the cost of the tools against the benefits we receive from the things we just mentioned, we again realize a positive return on our investment. Tools pay for themselves.

# Next

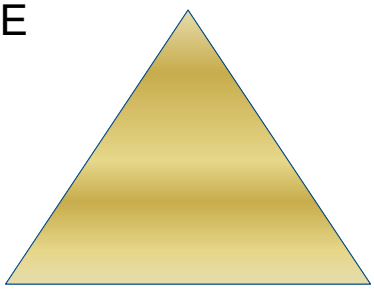
- Quality Assurance



In the next video, we will take a closer look at the quality triangle as it applies to software quality assurance people processes and tools.

# Quality Assurance

- People: Quality Organization
- Processes: Standards
- Tools: CASE

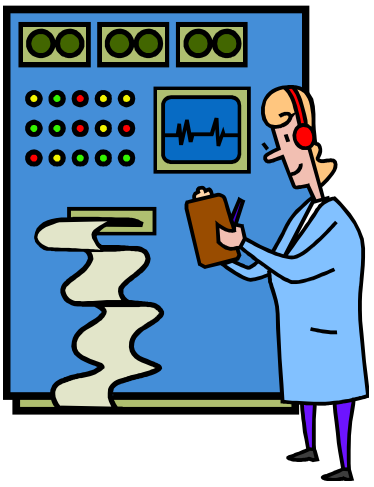


1

In this video we will discuss the quality triangle as it applies to the assurance of quality. After you watch it, you will be able to name the three main functions in the quality organization. You will be able to identify the organizations that provide standards for software engineering. You will be able to define the acronym CASE and name some tools that support software production.

## Quality Functions

- Configuration Management
- Testing
- Software Quality Assurance



Let's talk about the "people" dimension of quality assurance. There are three main functions related to quality assurance.

One is Configuration Management. These people are responsible for version control of the artifacts being produced by the developers. This includes code, of course, but also requirements and design documentation. Also the standards and procedures that are used by the developers should be kept under configuration control.

Testing is another function in the quality assurance organization. Now the developers themselves are responsible for unit testing of their own pieces of work, but there needs to be a separate group of people responsible for larger scale testing. We have an entire module related to testing later in the course.

The third function is sometimes called Software Quality Assurance. These people are managerially Independent from the developers, as well as configuration management and testing.
SQA is concerned with the software processes that regulate the work of these other groups. First, SQA ensures that standards are in place and are approved. They ensure that the standards are placed under configuration control. Then SQA runs periodic reviews to ensure that the standards are being followed. Thus, SQA assures all stakeholders of the quality of the workmanship of the rest of the team.

## Standards

- Government standards
  - http://everyspec.com/
- IEEE standards
  - https://standards.ieee.org/
- ISO standards
  - http://www.iso.org/iso/

The second side of our triangle is Processes. Standards are embodiments of best practices.

Government organizations have historically been the largest originators of standards documents. This would include the US Military, NASA, FAA, and other government agencies, as well as NATO and governments of other countries. Typically, for government contracts, the customer will provide a set of standards to be followed.

There are also two big non-governmental organizations that sponsor standards activity in software engineering: the IEEE (Institute of electric and electronic engineers) and the ISO (International Standards Organization).

If you follow the links, you will get a sense of the magnitude of standardization efforts.

Much of the time, these standards will be tailored to meet the needs of each project. Some small projects may not need a lot of standards, while large projects are likely to fail if there are no standards applied to the work effort.

## Tools

- CASE – Computer Aided Software Engineering

The third side of the triangle is Tools. The acronym CASE, for Computer Aided Software Engineering, is frequently used to refer to thee types of tools. CASE tools aid in all aspects of software engineering:

One group of tools are the so-called Upper CASE tools. They support earlier phases in the software development process such as

- Planning and estimating.
- Project management.
- Requirements analysis.
- Analysis and design.

And then there are Lower CASE tools that support the production of the software

- Coding and reuse.
- Analyzing code.
- Testing.
- Configuration management.
- Metrics.
- Documentation.
- Maintenance.

# Summary

You should now be able to:

- Define Software Quality.
- Demonstrate how to achieve and assure software quality.
- Justify the Return on Investment (ROI) of quality processes.
- Explain how to prevent defects.
- Demonstrate how investments in People, Process and Tools can promote quality.
- Describe the Quality organization: CM, Testing, and SQA.

Here are the topics we have covered in this module. You should now be able to

- Define the term, Software Quality.
- Demonstrate how to achieve quality in our efforts and assure ourselves and others that we are producing a quality product.
- Justify the Return on Investment (ROI) of quality processes of achieving and assuring quality.
- Explain how to manage defects, most importantly how to prevent them.
- Demonstrate how investments in People, Process and Tools can promote quality.
- Describe the Quality organization: CM, Testing, and SQA.