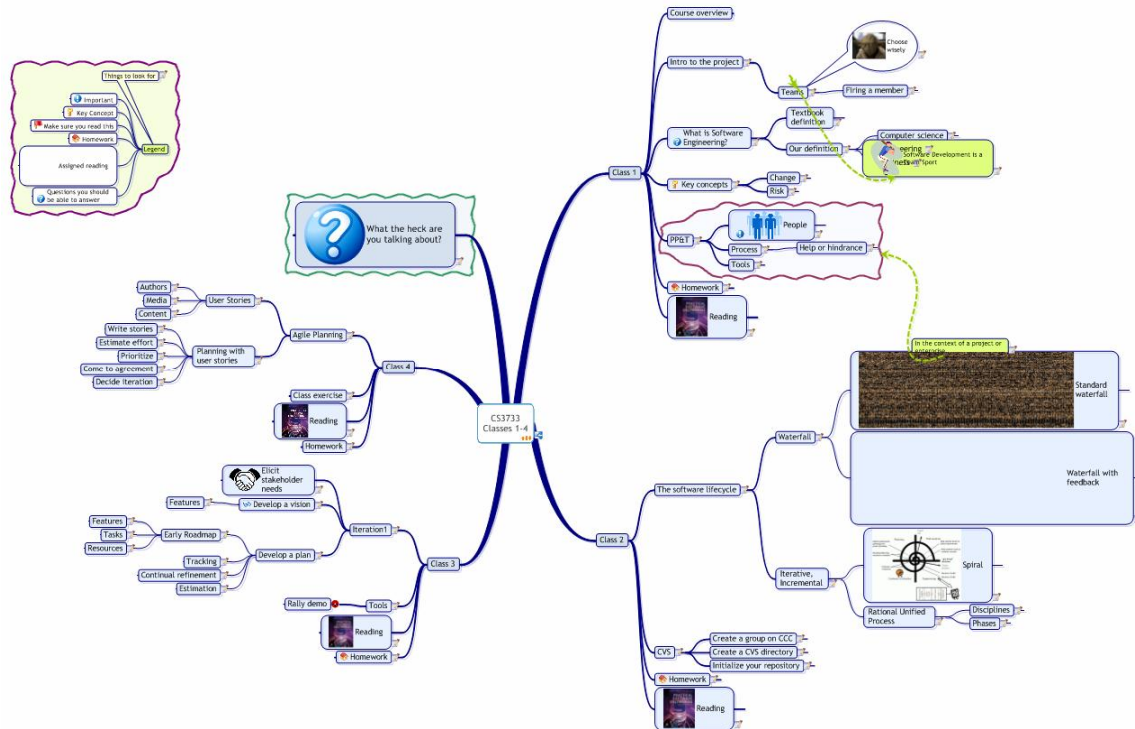


# CS3733 Classes 1-4



See document: [CS3733 SoftwareEngineering.mmap](#)

## Class 1

### Course overview

#### Intro to the project

The project for this course consists of your team producing a software system that satisfies your stakeholders. The system has already been conceptually divided into components (subsystems). Your job is to manage the construction of the subsystem that is assigned to you. You will be required to interact with the stakeholders to prioritize requirements, manage the scope, and ensure that you are building the right things.

Even though you are only responsible for building one of the subsystems, you are responsible for delivering a complete product. This means that you will need to integrate your subsystem with subsystems produced by other teams. Usually you will have more than one possible supplier for each subsystem. You must make the wise choice in selecting your vendor. You want to choose one that has working software, supports their customers, and understands your needs.

Conversely, you are a supplier for other teams and must provide them with an appropriate product (subsystem).

#### Teams

Teams will be made up of 3-4 students. You should have your teams identified by the next class.

There will be a forum on myWPI for you to use to seek out teams and team members and report the makeup of your teams.

You are encouraged to create a name for your team. This is your "company" name.

- **Choose wisely**

One of the surest ways to have difficulty with this class is to end up on an ineffective, or dysfunctional, team.

Choose your teammates wisely. Find those who have the same work ethic and set of values that you do. Then keep each other involved and responsible to the rest of the team.

## **Firing a member**

You can fire a team member who is not pulling his or her weight. All team members must agree that the firing is warranted and be willing to document the reasons for the dismissal.

Team members who are fired will be required to find another team who is willing to take them on, or create a team with other students who have been fired from their team. Individual projects will not be allowed.

## ***What is Software Engineering?***

This course is part of your computer science curriculum. However, software engineering is not computer science. What then is it?

### **Textbook definition**

The textbook describes software engineering in terms of five key observations. These do not, however, really define what software engineering is.

- the software system less than the enterprise information system
- is part of the business process
- it is different from traditional engineering
- more than programming
- it is about modeling
- it is complex

We will address many of these observations in our preferred definition of software engineering.

### **Our definition**

Software engineering is a combination of many different fields. The main contributors to software engineering are described in the following subsections.

### **Computer science**

Software engineering is intimately coupled with computer science. You cannot be a successful software engineer unless you have some in-depth knowledge of algorithms, programming methods, logic, discrete mathematics, and other basic computer science topics.

### **Engineering**

Engineering is:

*The profession in which a knowledge of the mathematical and natural sciences gained by study, experience, and practice is applied with judgement to develop ways to utilize, economically, the materials and forces of nature for the benefit of mankind.*  
(Accreditation Board for Engineering and Technology, 1996)

Engineering is different from a basic science in that it is concerned with the application of science to solve problems, rather than discovering new scientific principles.

## **Business**

Software engineering is done in a business, or enterprise context. We engineer solutions to satisfy the needs of the organization.

Software engineering is applied in many commercial settings. It is much more than providing Information Technology (IT) systems to an organization. Software has become an ubiquitous entity in every aspect of life today; whether it is manifested in the form of embedded software in appliances and communication devices, large data processing and data mining applications, scientific and engineering systems, and so on.

As the textbook observes, the software system is less than the enterprise's information system. Software engineering takes into consideration the whole enterprise.

## **Key concepts**

There are two key concepts that you need to understand and consider early and often throughout a project.

## **Change**

One constant in software development is that *things change*. Change is inevitable. An effective process allows you to address changes in requirements, business conditions, new technologies, ...

## **Risk**

Successful projects are those that constantly pay attention to potential risks and work at reducing them. One of the things you will need to do is maintain a risk list and keep it up-to-date.

You should prioritize your risks on a project according to the likelihood that they will happen and the possible damage that can happen if the risk materializes.

## **PP&T**

When we consider software engineering in the context of providing valuable software systems to stakeholders, we can identify three broad categories that must be addressed. These are *People, Process, and Tools*, as discussed in the following sections.

## **People**

The **most important** success factor in building effective, valuable software systems is people. People build software. People use software. People are affected by software.

There are many people issues that we will address in the course. You will experience many of these first-hand, in a small context, as you work on your project. Effective software engineers must be able to deal with people on their teams, and outside of the teams.

If people do not work well together, you have virtually no hope of succeeding in software development.

## Process

When you work on any team project, you need to identify how the team will work. You want to have enough structure so that everyone knows what they are supposed to do, how they contribute to the team's success, and what they can expect from their teammates.

In its simplest form a process describes: *Who* does *what* and *when*.

The "who" is described as a set of roles, or responsibilities. The "what" is described as a set of activities that produce results, often in the form of *artifacts*. The "when" describes the ordering, or temporal dependence, if any. For example, the programmer (who) write the code for a unit (what) after (when) the requirements specifier (who) produces a use case (what).

Some processes, like the Rational Unified Process (RUP), also provide guidance on *how* to perform the different activities.

## Help or hindrance

Process has gotten a bad name because there are numerous cases in the history of software engineering where process becomes the product, rather than software. When this happens, teams bog down doing what the process requires rather than focusing on solving their customers' problems by producing useful software. This type of bloated process leads to teams that are unable to deliver results. (One of the reasons why only 28% of IT projects are completed on-time, within budget, according to the Standish Group's CHAOS latest CHAOS report).

An effective process contains just enough formality to let the team know who does what, and when and lets the team adjust the process as needed. An effective process is also one that proceeds iteratively and incrementally.

## Tools

The third category that we will be concerned with is the *tools* category. Software is complex. We want to automate the software development process as much as possible. This requires a balanced approach to selecting the right tools for the job at hand. In this course we will use several tools. Most of them are quite appropriate for a course project, as well as for larger projects. Some of the tools, however, will be more complex and not deliver the full amount of benefit to you in this course. However, they are appropriate for industrial projects, and the lack of benefit received on your project is outweighed by learning the tools for possible use in your future career. The number of such tools will be minimal.

## Homework

Form teams and post your team members (optional team name) on the myWPI discussion forum for teams. Make sure that you identify the person who will be responsible for project management.

*Due, 28-Oct-2004*

## Reading

Chapter 1 of the textbook.

## Class 2

### ***The software lifecycle***

The software (development) lifecycle (SDLC) refers to the events that occur when producing a software product. There are two fundamental SDLC approaches with several variations on each.

## **Waterfall**

The waterfall lifecycle is one that was used extensively until the 1990s. It is often attributed to Winston Royce, however, Royce's views were widely misunderstood and he has gone down in history as the father of the waterfall.

The waterfall approach will work only if you have complete, or almost complete, information at the beginning of your project and if there is little or no change during the development of your project.

## **Standard waterfall**

The standard waterfall model is divided into a set of non-overlapping, sequential phases that correspond to:

- requirements analysis and specification
- design
- implementation
- integration and deployment
- (optional) operation and maintenance

## **Waterfall with feedback**

This model has the identical phases as the standard waterfall, but it depends upon feedback between phases that might cause the team to *back up* to a previous phase if a problem is encountered.

This model is still much too sequential. Defects are found late in the process. If you make a bad decision at an early phase, you often have to back up to that phase and rework a much of what you have done.

Rework is inevitable, but you want to minimize the amount that you have to do.

## **Iterative, Incremental**

Most modern software development lifecycles stress iteration through the process and adding incremental functionality in each iteration.

## **Spiral**

In response to the deficiencies of the waterfall SDLC, Barry Boehm from USC, described the spiral model of software development. This is the basis for most modern software development processes.

In the spiral model, you do some requirements, evaluate risks, implement (engineer) the product for that iteration, and get customer feedback on the iteration.

As we shall see in subsequent discussions, while the parts of the loop may change, the pattern of repetition and incremental delivery of product are repeated in many other methods.

## **Rational Unified Process**

The Rational Unified Process (RUP) is a commercial implementation of the Unified Process. RUP describes the software development process along two dimensions.

### ***Disciplines***

One dimension (vertical) is a description of disciplines (types of activities) a software development team performs when implementing a software-intensive system. The main disciplines in RUP are

- Business modeling. Understanding the enterprise in which the system will operate.
- Requirements. Understanding the customer needs, Analyzing the needs in order to ensure that the correct system is built.
- Analysis and design. Building an object model, designing an architecture, and detailing the system specification.
- Implementation. Creating the code artifacts that realize the requirements.
- Testing / quality. Ensuring that the requirements are correctly realized.
- Deployment. Packaging and delivering the product.

There are other disciplines that are considered to be supporting disciplines, such as Project Management, Environment, and so on.

### ***Phases***

RUP defines four phases along a horizontal axis that is a temporal scale.

Each phase is usually divided into one or more iterations; where working software is produced at the end of each iteration, except, perhaps, in the first couple of iterations.

The four phases are

- Inception. You determine what you are going to build, who wants it, and whether it is economically viable.
- Elaboration. You address the most important parts of the system, identify appropriate technical approaches, and implement an *executable architecture*.
- Construction. Build the system.
- Transition. Deliver the system.

The end of each phase marks a milestone. This is a point in the project where you make a decision whether to continue the project or not.

## **CVS**

CVS is a version (revision) control tool. It lets a software development team work on the same set of artifacts and maintain a record of all revisions, with the ability to *back out* unwanted revisions.

It also allow you branch and merge revisions -- a capability provided by several other version control tools. You will use CVS for your team projects. The steps to use CVS are described in the following sections.

### **Create a group on CCC**

Create a group on the CCC system for your team. See the CCC web pages for information on how to do this.

### **Create a CVS directory**

In one of the team members' home directories, create a CVS directory and set the group write permission bit on this and subdirectories. You will use the sticky bit setting for this.

### **Initialize your repository**

From the user account, initialize the CVS directory. If your CVS repository is in a directory named CVS as a subdirectory of your home directory, you would use the command:

```
cvs -d <path to the CVS directory> init
```

## **Homework**

Prepare a team communication form for your team. Post your communication form in the team file exchange for your team in myWPI. There is a template for the team communication form on the process web pages.

## **Reading**

There is a link to an essay about people, process, and tools on the course web pages.

## **Class 3**

### **Iteration1**

Your first iteration on a project will usually address inception issues. You want to quickly get an idea about what the project is all about and start putting together ideas about how long it will take, what you need to accomplish the job, and so on.

### **Elicit stakeholder needs**

In order to determine your stakeholder needs, you need to identify who the stakeholders are. Clearly, end users have a stake in the success of the project. The customer who is paying for the system is also a stakeholder.

But there are more stakeholders. The development team, management, company shareholders, and others. Each of these stakeholders may have some ideas about what your project should produce. You need to make sure that you pay attention to them with an appropriate amount of detail.

Initially, you will gather most of the stakeholder needs. But due to changes, you cannot stop there. Iterative processes require you to continually review the needs with the stakeholders and ensure that you're still on track, or that you adjust your plans to make sure you're building the right product.

### **Develop a vision**

Every project needs a vision. The vision provides a common view of what the team is trying to accomplish.

The vision can be a simple statement like:

*We are going to build the most advanced, and easy-to-use on-line shopping site for antiques.*

It can also be a structured document that can extend for several pages, describing major features, competition, business climate, and other pertinent information.

### **Features**

If the vision does not contain a list of features, you need to develop a feature list. A feature is a statement of some characteristic or capability that your system provides. For example:

*The system will allow customers to pay for their purchases by credit card, or electronic fund transfer.*

### **Develop a plan**

If you're going to get anywhere, you need to plan your trip. A software development project is no different than any other type of project or endeavor where you have to attain a goal.

Plans can be very extensive and detailed or quite small and informal. The important thing is to ensure that all project team members and stakeholders know what the project is trying to do and, as time goes by, what changes are made and the current status.

RUP considers the major plan for a project the Software Development Plan that consists of several other sub-plans like:

- Iteration plan
- Requirements management plan
- Measurement plan
- Risk management plan
- Product acceptance plan
- Configuration management plan
- Documentation plan
- Quality assurance plan
- Process improvement plan

Clearly you can become overrun with plans. On very large projects each of these may be necessary. However, for small to medium-sized projects, you don't need to think of separate plans, but you do need to think of things like quality, configuration management, and so on.

## **Early Roadmap**

When you start your plan, you try to determine what success will be for your team. Then you try to break your project into a set of *steps* that will get you to realize that vision. Once you know where you're going and what has to get done, you need to begin determining who will do the work.

### ***Features***

If you know the set of features that you want in the final product, you should prioritize them. Which are most important to the customer? Which are the riskiest? Which depend on others?

### ***Tasks***

For each of the features, how will you implement them in the product? Each feature will have some number of tasks related to their implementation.

### ***Resources***

How much time does your team plan on spending each week? How long will each task take? Who decides?

## **Tracking**

So you have a plan. Now what? Is it left to collect dust? Not if you want to keep to your plan. You will need to track the work to see if anyone is getting off track and then make the appropriate adjustments.

## **Continual refinement**

The plan is a living artifact. If you make a plan, you want it for a reason. Therefore you need to look at it regularly and, as you get more and better information, update it.

## Estimation

How do you know how long something will take you to do?

There are many ways to do this, some more accurate than others. Some easier to compute than others. Some more stable than others.

For now, develop some simple guesstimate that will give you a starting point.

We will revisit estimation later on in the course.

## Tools

There are many tools for project planning that you can use. One of the more popular in the marketplace is Microsoft Project. For this class we will use the Rally tool.

## Rally demo

See document: [rally1.rallydevelopment.com](http://rally1.rallydevelopment.com)

## Reading

Sections 4.1, 4.2, and 4.2

## Homework

Prepare your project plan in the Rally project management tool.

## Class 4

### Agile Planning

For small-to-medium and/or informal projects, techniques such as the *XP Planning Game* may be effective ways of planning and tracking projects. This section describes the basics of the XP Planning Game.

### User Stories

User stories provide the development team with a statement of requirements at a granularity that can be used for planning. They describe things the system needs to do. While this may seem to be identical to features that were discussed previously, they are a bit different in that they are usually described in a much smaller scope. For example, the feature might be:

*The system will allow customers to pay for their purchases by credit card, or electronic fund transfer.*

There may be several user stories that are specified to make the feature actually exist. These would include:

- The user must be able to enter their credit card information that consists of the card type, card number, expiration date, and security code if it applies.
- The valid credit card types must be a set that is defined by the system administrator. For the initial release, we will accept American Express, MasterCard, and Visa only.
- Each credit card type will be identified by the credit card supplier information that includes the internet connection to be used for account verification.
- The system must be able to verify the credit card account status before allowing a purchase to be completed.

- There must be the ability for management overrides to allow purchases to be made when communication to the credit card provider cannot be established.

## **Authors**

Who writes the user stories? In an XP project, the customer write the user stories. What if you don't have a customer who is available to write them? Then anyone can write the user story, as long as the team agrees that they want to use them to describe the system at this level of detail. You should review them with your stakeholders as much as possible.

## **Media**

In XP, user stories are written and maintained as a stack of index cards. This allows the stories to be easily created, removed, split, and provides an easy way of working with them that anyone can get used to.

You could use other means to maintain user stories, depending upon your project team.

## **Content**

What goes into a user story card? There are many different formats, but most of them will provide space for:

- Description of the story
- Estimate of effort to complete
- Actual effort
- Priority
- Risk
- Completion / final disposition
- Developer
- Iteration

## **Planning with user stories**

Planning with user stories is easy to describe. Don't be fooled, however, getting all of the information and coming to agreement can be quite tricky.

## **Write stories**

The customer or other authors write the stories in enough detail to allow the developers to estimate the effort.

If there is not enough detail, then the author must be available to discuss the story with the development team.

## **Estimate effort**

The developers take the user stories and estimate the effort it will take to implement the story. They record the estimates in the unit-of-measure the team adopts.

## **Prioritize**

The customer prioritizes the stories. This is typically identifying those stories that are absolutely essential to the success of the project, those that add the most value, and so on.

## **Come to agreement**

Developers and customer need to come to an agreement on what the user stories mean, whether there are hidden dependencies, or other things that could cause the estimates to be wrong, or the work to be misunderstood. Often user stories may be split or merged as a result of this iterative process of gaining a shared understanding.

## **Decide iteration**

The team estimates the total number of units of work they can do for the next iteration.

The customer selects user stories that add up to no more than the estimated number of units.

## ***Class exercise***

Write some user stories for your subsystem of the class project.

## ***Reading***

Sections 7.1-7.2

## ***Homework***

Assume that you are engaging a software development team to create a system that will let teachers test their students on- line. The teachers need to be able to "publish" tests, notify the students that a test is available, and retrieve the completed tests. Students need to take the tests and see their grades after the teacher grades them.

Write 10 good user stories for this system. A good user story is one that describes one thing in enough detail that a developer can estimate the effort required to implement the story.

Put your stories in a text or word document and submit it using turning.

## **What the heck are you talking about?**

Any time you don't understand something, it is **your responsibility** to speak up. It's perfectly reasonable at any time during a class (except when taking tests, etc.) to ask "**What the heck are you talking about?**" At that point the speaker has the responsibility to try and clarify the issues for you.