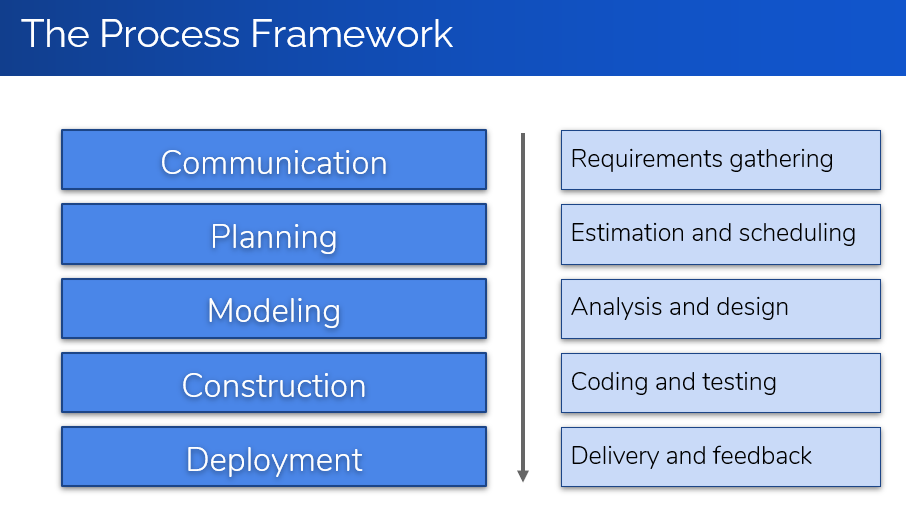
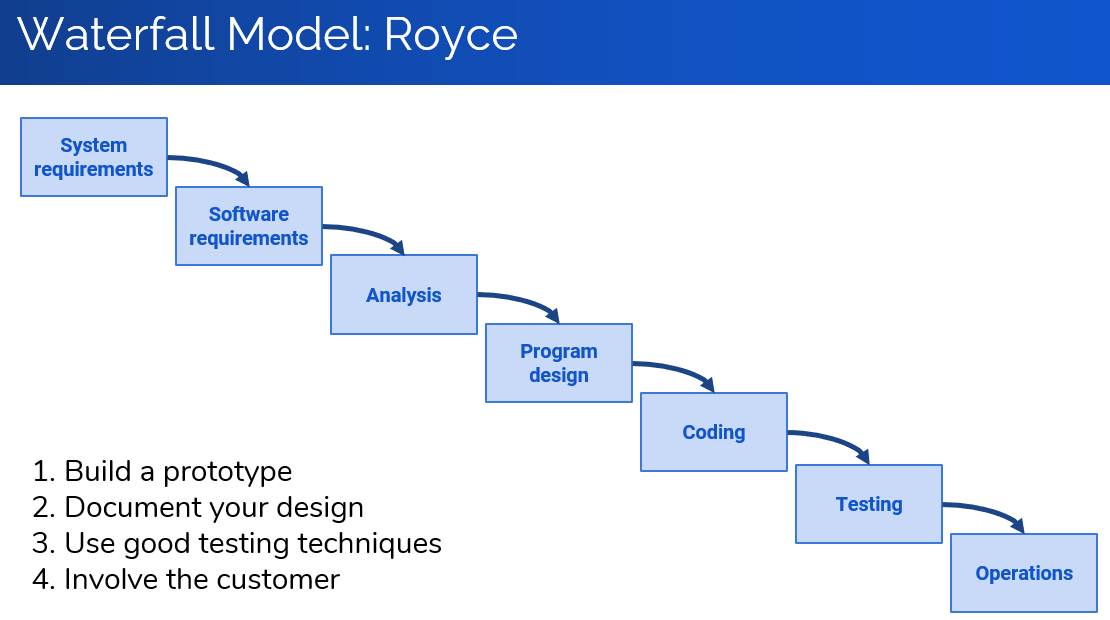
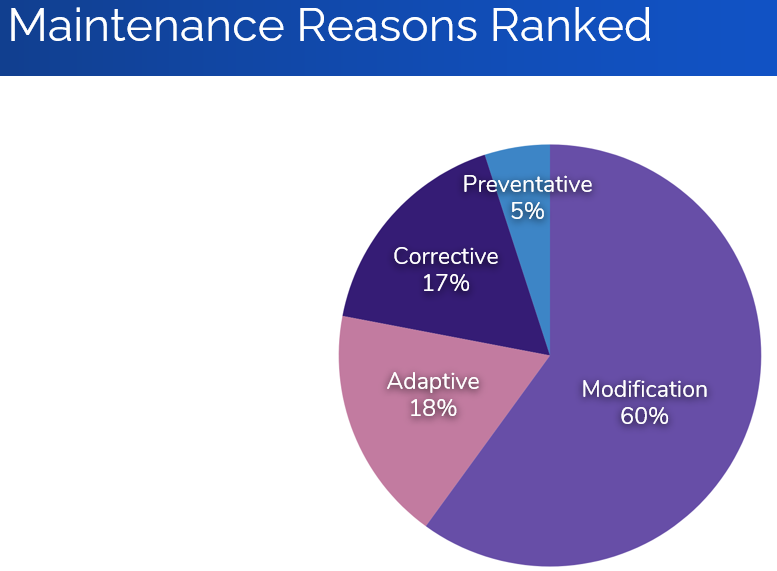
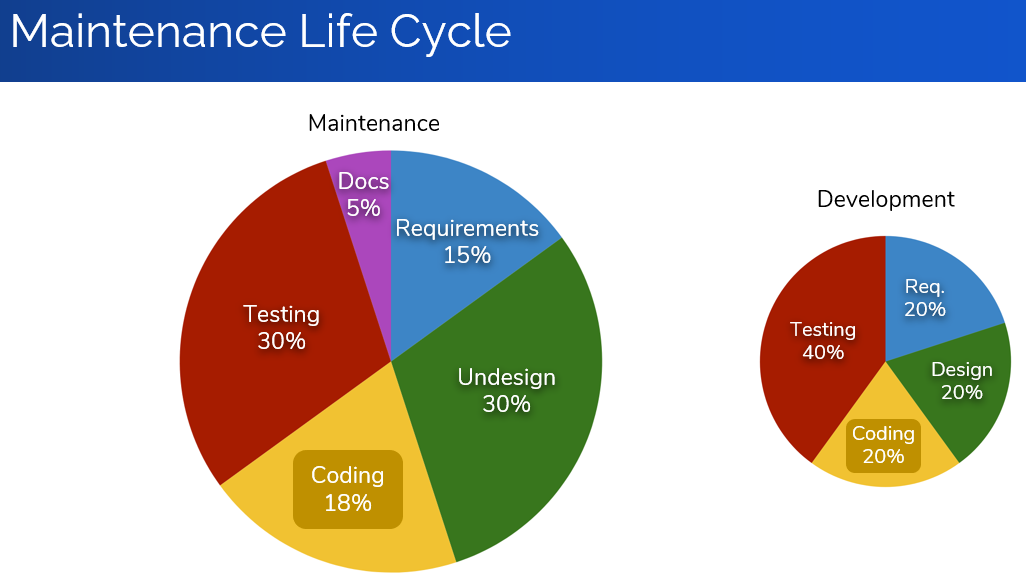
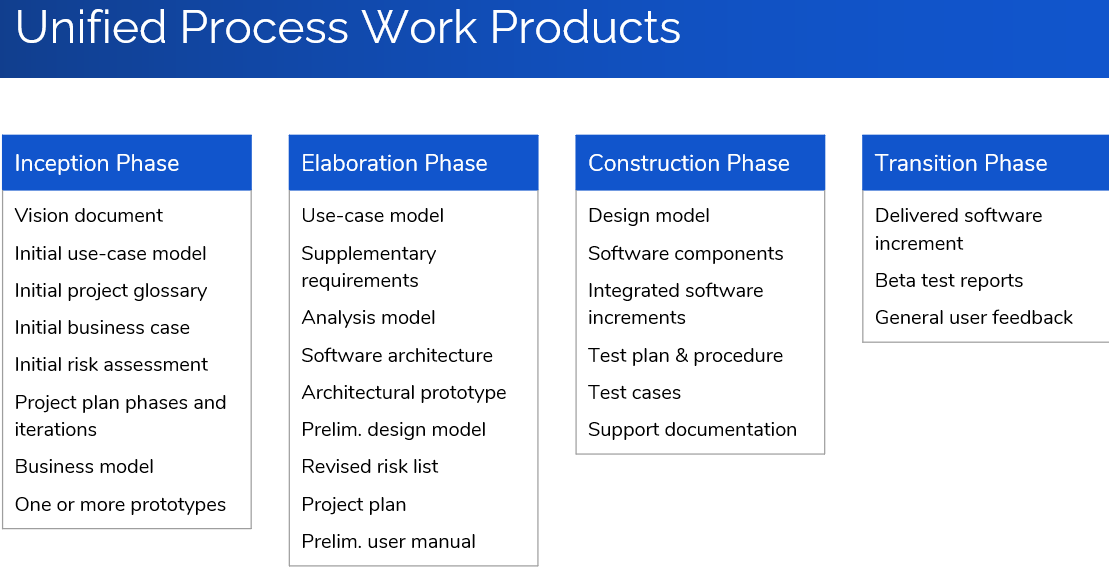
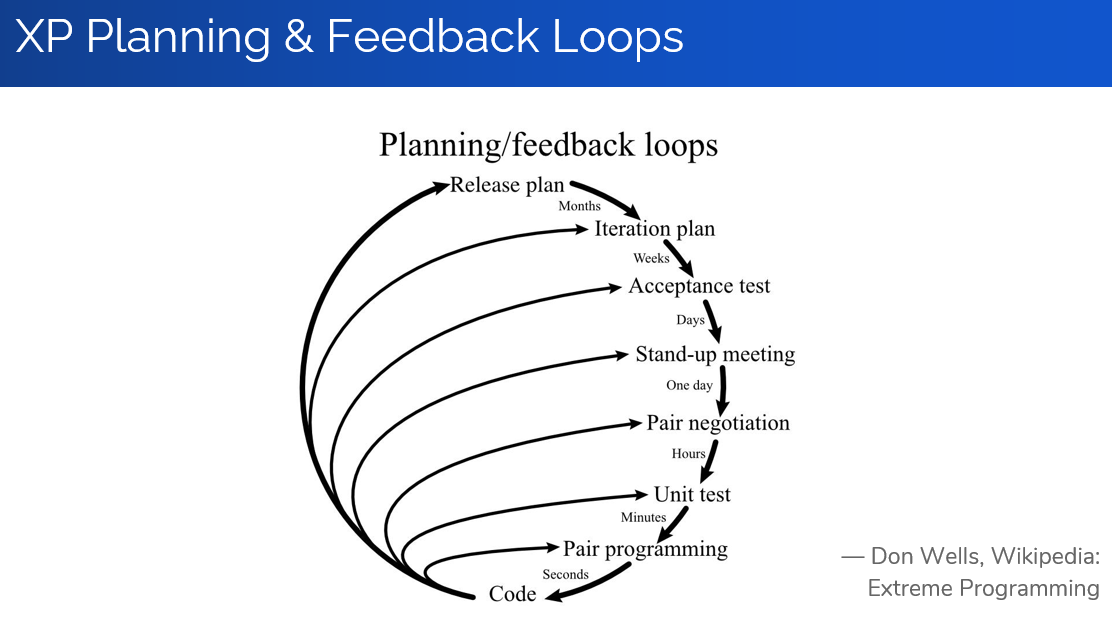
**Module 1**

* \*quality software\* is the foundation of software engineering.
* Software engineers want a disciplined approach to software development because they want to be able to build a quality product. But they also want an agile approach that allows them to respond effectively to change.
* 
* General Principles:
  + Provide value to users
  + Keep it simple
  + Maintain the vision
  + What you produce,  
    others consume
  + Be open to the future
  + Plan ahead for reuse
  + Think before you act
* The Essence of Practice:
  + Understand the problem  
    (communication and analysis)
  + Plan a solution  
    (modeling and design)
  + Carry out the plan  
    (code generation)
  + Examine result for accuracy  
    (testing and quality assurance)
* Layered Approach to Software Engineering:
  + Tools
  + Methods
  + Process
  + Quality Focus
* Questions Asked by Software Engineers:
  + Why does it take so long?
  + Why are costs so high?
  + Why can’t we find errors earlier?
  + Why is maintenance so hard?
  + Why is measuring progress so hard?
* Brooks Law: Adding more programmers to a late project will make it even later

**Module 2**

* Waterfall model disadvantages:
  + Real projects rarely follow a sequential workflow.
  + It’s hard of a customer to explicitly state all requirements at the beginning of the project.
  + The customer may need to wait a long time before they get a working program.
  + Major mistakes may not be detected until the working program is reviewed.
* 
* The most time-consuming phase of the software development life cycle is **Maintenance**
* Second time-consuming is **Testing**
* Reasons for Maintenance:
  + Adaptive maintenance
  + Corrective maintenance
  + **Modifications** – the highest ranking
  + Preventive maintenance
* 
* 60 percent of software’s dollar is spent on maintenance, and 60 percent of maintenance is enhancement.
* 
* Prototyping Potential Problems:
  + Customers see what appears to be working software.
  + Design compromises made for the prototype become part of the system.
* **Rapid prototype** (throwaway) – Simple model of system to visually show users what their requirements may look like
* **Evolutionary prototype** – Robust, structured prototype that can continually be refined and rebuilt
* Spiral Model Misconceptions:
  + The spiral is simply a sequence of waterfall increments.
  + All project activities follow a single spiral sequence.
  + Every activity in the diagram must be performed, and in the order shown.
* Unified Process Model Characteristics:
  + Use-case driven
  + Architecture centric
  + Iterative
  + Aligned with UML
* 
* Spiral model has an evolutionary flow

**Module 3**

* Agile Manifesto:
  + Individuals and interactions over processes and tools
  + Working software over comprehensive documentation
  + Customer collaboration over contract negotiation
  + Responding to change over following a plan
* Principles of Agility:
  + Priority: Early and continuous delivery of valuable software
  + Welcome changing requirements, even late in development
  + Deliver working software frequently (every few weeks)
  + Customers and developers work together daily
  + Build projects around motivated individuals
  + Face-to-face conversation is best
  + Working software is the primary measure of progress
  + Agile processes promote sustainable development
  + Technical excellence and good design enhance agility
  + Simplicity is essential
  + Use self-organizing teams
  + Periodically assess your team and change accordingly
* XP Planning:
  + Begins with the creation of user stories
  + Customers assign a value, and developers assign a cost
  + Stories are grouped to form a deliverable increment
  + A commitment is made on a delivery date
  + After first increment project velocity is used to help define subsequent delivery dates for other increments
* XP Design: CRC cards: Class, Responsibility, and Collaboration
* Acceptance tests are defined by the customer and executed to assess customer visible functionality
* 
* Kanban Practices:
  + Visualize your workflow
  + Limit work-in-progress
  + Make policies explicit
  + Manage workflow
  + Implement feedback loops
  + Improve collaboratively, evolve experimentally
* DevOps Stages:
  + Continuous development
  + Continuous testing
  + Continuous integration
  + Continuous deployment
  + Continuous monitoring

**Module 4**

* Principles that Guide Practice:
* Core:
  + Be agile.
  + Focus on quality.
  + Be ready to adapt.
  + Build an effective team.
  + Establish mechanisms for communication and coordination.
  + Manage change.
  + Assess risk.
  + Create work products that provide value for others.
* Principles:
  + Divide and conquer.
  + Understand abstraction.
  + Strive for consistency.
  + Focus on transfer of information.
  + Build modular software.
  + Look for patterns.
  + Show the problem and solution from different perspectives.
  + Remember that someone will maintain your software.
* Communication Principles:
  + Listen.
  + Prepare before you communicate.
  + Use a facilitator.
  + Talking face-to-face is best.
  + Take notes & document decisions.
  + Strive for collaboration.
  + Stay focused.
  + Draw a picture to clarify.
  + When you agree, move on;  
    When you disagree, move on;  
    When you’re blocked, move on.
  + Negotiation is not a contest.
* Planning Principles:
  + Understand the project scope.
  + Involve stakeholders in planning.
  + Planning is iterative.
  + Estimate based on what you know.
  + Consider risk when planning.
  + Be realistic.
  + Adjust granularity in the plan.
  + How will you ensure quality?
  + How will you handle change?
  + Track the plan; adjust as required.
* Modeling Principles:
  + The primary goal of your team is to build software, not models.
  + Don’t create unnecessary models.
  + Keep your models simple.
  + Build models that can change.
  + Each model has a purpose.
  + Adapt the model to the system.
  + Build useful (not perfect) models.
  + A model should communicate content, syntax is secondary.
  + If a model feels wrong, it is.
  + Get feedback as soon as possible.
* Construction Principles:
  + Understand the problem.
  + Understand design principles.
  + Pick an appropriate language.
  + Pick appropriate tools.
  + Create unit tests before code.
  + Follow structured programming.
  + Consider pair programming.
  + Use appropriate data structures.
  + Create appropriate interfaces.
  + Conduct code walkthroughs.
  + Unit test to find errors.
  + Refactor to improve quality.
* Testing Principles:
  + Tests should be traceable to requirements.
  + Test should be planned.
  + Pareto principles: 80% of all errors are in 20% of components.
  + Test from small to large.
  + Exhaustive testing is not possible.
  + Focus tests on complex code.
  + Consider testing documentation.
  + Track defects; look for patterns.
  + Include tests that show correct behavior.
* Deployment Principles:
  + Manage customer expectations.
  + Test the complete deliverable.
  + Establish a support regimen.
  + Provide instructional materials.
  + Do not deliver buggy software!
* Requirements Engineering Activities:
  + **Inception** – Establish a basic understanding of the problem and the nature of a solution.
  + **Elicitation** – Draw out the requirements from stakeholders.
  + **Elaboration** – Create analysis model: represent information, functions, and behavior.
  + **Negotiation** – Agree on a deliverable system that is realistic for both developers and customers.
  + **Specification** – Describe requirements formally or informally.
  + **Validation** – Review the specification for errors, ambiguities, omissions, and conflicts.
  + **Requirements management** – Manage changing requirements.
* Establishing the Groundwork:
  + Identifying stakeholders
  + Recognizing multiple viewpoints
  + Working toward collaboration
  + Asking the first questions
  + Nonfunctional requirements
  + Traceability
* The hardest single part of building a software system is deciding what to build. No part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later.

— Fred Brooks

* The seeds of major software disasters are usually sown within the first three months of commencing the software project.

— Capers Jones

* We spend a lot of time—the majority of the project effort—not implementing or testing, but trying to decide what to build.

— Brian Lawrence

* Collaborative Requirements Gathering:
  + Meetings are attended by developers and other stakeholders.
  + Rules for preparation and participation are made.
  + The agenda is formal but allows exchanges of ideas.
  + A facilitator is used.
  + A definition mechanism is used.

**Module 5**

* Classes should be immutable unless there is a very good reason to make them mutable.
* Singleton classes represent objects for which only one single instance should exist.
* Types implement Comparable:
  + String
  + LocalDate
  + Sex (enum type)