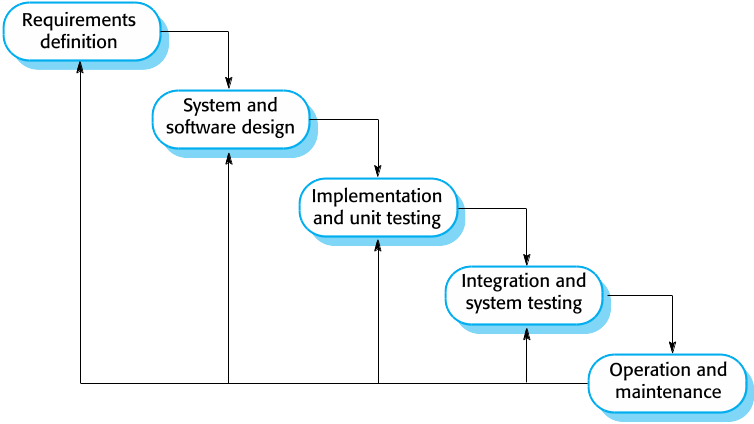
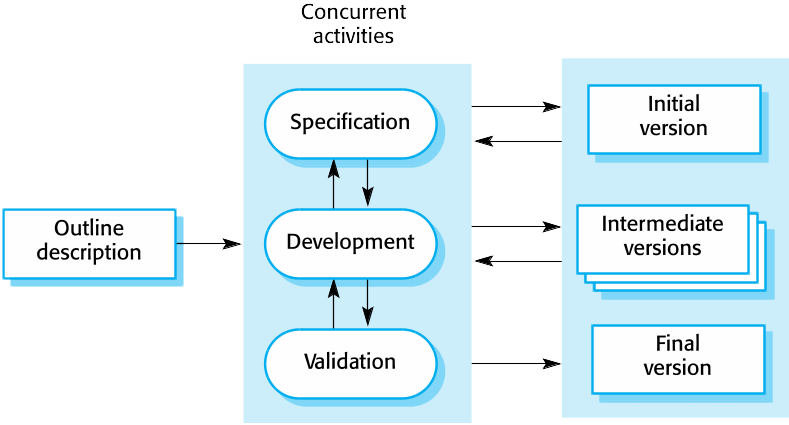
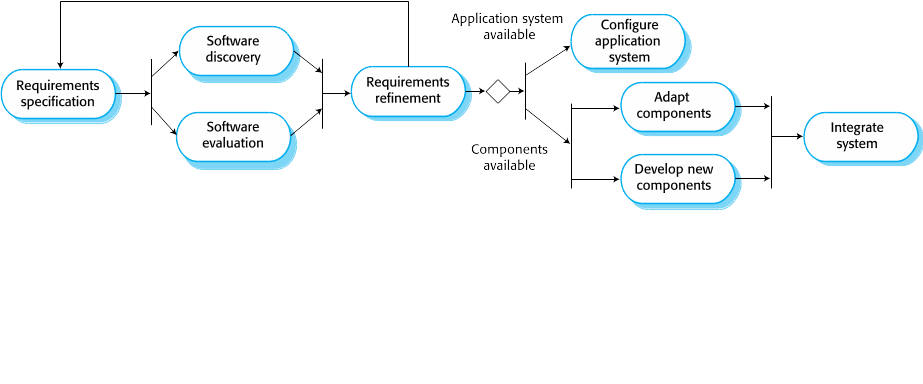
* Software costs
  + More than hardware
  + More to maintain than to develop
    - Especially for systems with a long life
  + Cost-effective development is the main concern of software engineering
* Software project failure
  + Increasing system complexity
    - New techniques require more complex systems.
      * Capabilities that used to be impossible
* FAQ
  + What is software
    - Computer programs and associated documentation
  + What are attributes of good software
    - Deliver required functionality and performance. Maintainable, dependable, and usable
  + What is software engineering
    - Engineering discipline that is concerned with all aspects of software production
  + What are the fundamental software engineering activities
    - Software specs, dev, validation, and evolution
  + What is the difference between software and engineering computer science
    - CS is theory and fundamentals, SE is practicalities of developing and delivering useful software
  + Diff between SE and systems engineering
    - SE is all aspects (hardware, software, process). SE is part of this
  + Key challenges
    - Increasing diversity, demands, shorter time, trustworthy, budget
  + Costs?
    - 60% are development costs, 40% testing, for custom, evolution costs often exceed dev costs
  + Best SE techniques and methods
    - Many types for different projects
  + Differences made by the web?
    - More available services, developing highly distributed service-based systems. Advances in programming languages, and software reuse
* Software products
  + Generic
    - Standalone systems (sold to customer)
    - Graphics programs, project management, CAD
  + Customized
    - Software commissioned by specific customer to meet their own needs
      * Embedded control systems, air traffic control, traffic monitoring
* Product spec
  + Generic
    - Specs of what the software should do is owned by the developer, and decisions are made by dev
  + Customized
    - specs and decisions are made by the customer
* Good software
  + Maintainable
  + Dependable and secure
    - No damage in event of failure
    - No unauthorized access
  + Efficient
    - No wasting memory or processor cycles
    - Responsiveness, processing time, memory utilisation...
  + Acceptability
    - Understandable, usable, compatible with other systems in use
* Software engineering
  + Discipline that is concerned with all aspects of software production from early stages of system specification through to maintaining the system after it has gone into use
  + Engineering discipline
    - Appropriate theories and methods to solve problems with organizational and financial constraints
  + All aspects of software production
* Software process activities
  + Specs: define software and constraints
  + Development: designed and programmed
  + Validation: checked for what was required
  + Evolution: modified for features and fixes
* General issues
  + Heterogeneity
    - Systems required to operate across networks that include different types of computer and mobile devices
  + Business and social change
    - Everything is changing very quickly
    - Need to be able to change existing software and to rapidly develop new software
  + Security and trust
    - Have to trust
  + Scale
    - Wide scale, from embedded systems to internet-scale, cloud-based systems that serve global audience
* SE Diversity
  + Many types of systems and no universal set of software techniques that is applicable to all of these
* Fundamentals
  + Some principles apply to all types of systems
    - Need managed and understood development process.
    - Dependability and performance
    - Understand and manage software specification and requirements
    - Reuse software that has already been developed rather than write new software
* Internet SE
  + Web based systems are more common
  + Web services allow functionality to be accessed over the web
  + Cloud computing
    - Users don't buy, you just rent or use it for free
  + Software reuse
    - Try to use pre-existing software components
  + Incremental and agile development
    - Generally recognized that agile is best for web, impractical to specify all requirements
  + Service-oriented systems
    - Components are stand alone web services
  + Rich interfaces
    - AJAX, HTML5
* Ethics
  + Wider responsibilities than application of technical skills
  + SEs must behave in an honest and ethically responsible way if they are to be respected as professionals
  + More than just the law, but also principles
* Issues of professional responsibility
  + Confidentiality
    - Even if you don't sign an NDA, respect confidentiality
  + Competence
    - Engineers should not misrepresent competence
  + IP rights
    - Engineers should be aware of IP rights. Protect IP of employers and clients
  + Computer Misuse
    - Shouldn’t use technical skills to misuse other people’s computers.
      * Playing games on employer’s machine
      * Sendesing virus
* ACM/IEEE code of ethics
  + Public - act with public interest
  + Client and employer - act with client and employer (w/ public interest)
  + Product - highest standards
  + Judgement - integrity and independence in professional judgment
  + Management - managers shall promote ethics
  + Profession - advance integrity and reputation consistent with public interest
  + Colleagues - fair and supportive of colleagues
  + Self - lifelong learning of profession and promote ethical approach to profession
* Ethical Dilemmas
  + Disagreement in principle with policies of senior management
  + Employer acts unethically - releases safety-critical system without finishing testing
  + Participation in development of military weapons systems or nuclear systems
* 9/11/17
* Software Processes
  + Structured set of activities required to develop a software system
  + Many processes, but all involve
    - Specification
      * What it does
    - Design and implementation
      * Organize and implement
    - Validation
      * Test to make sure it works and does what it is supposed to
    - Evolution
      * Change system in response to changing customer needs
  + Software process description
    - Includes products, roles, pre and post conditions
  + Plan-driven processes
    - All activities are planned in advance and progress is measured against the plan
  + Agile processes
    - Planning in incremental and easier to change the process to reflect changing customer requirements
  + Most processes include both
    - No right or wrong processes
* Software process models
  + Waterfall model
    - Plan-driven: separate and distinct phases of specification and development
  + Incremental development
    - Specification, development, and validation are interleaved. Plan-driven or agile
  + Integration and configuration
    - System assembled from existing configurable components. Plan-driven or agile
  + In practice, most large systems are developed using a process that incorporates elements from all these models
* Waterfall Model



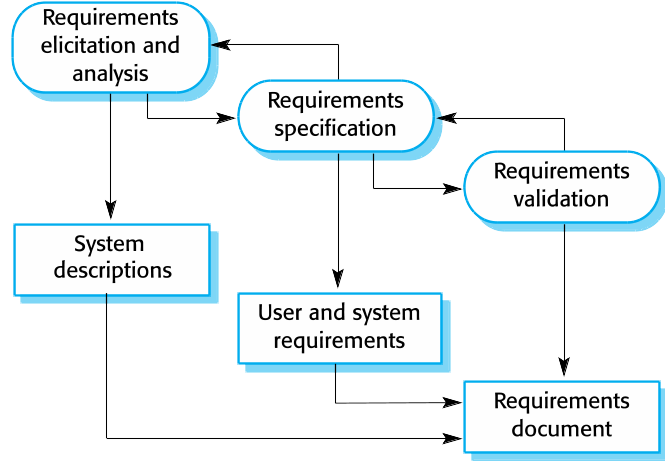
* + Can return to any step after deployment
    - If something needs to be changed SoL
  + Very change resistant
    - Only appropriate where requirements are well defined, and static
    - Not realistic for most business systems
  + Mostly for large systems developed at several sites
    - Helps coordinate large group
* Incremental Model



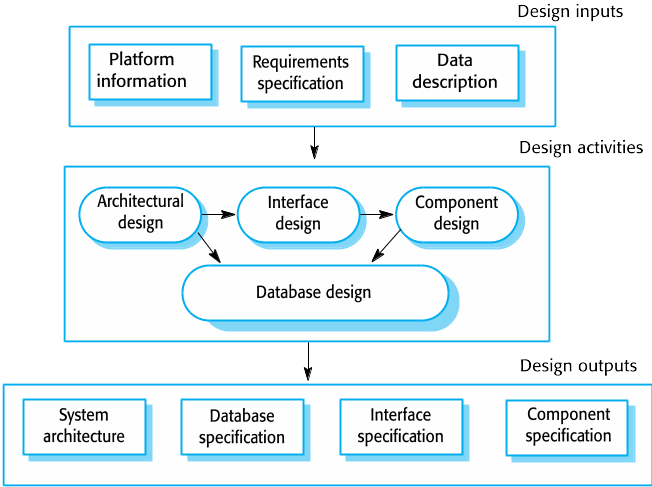
* + Benefits
    - Easier to change
    - Can get intermediate customer feedback
      * Clarification
      * Demonstration
    - Rapid delivery of minimum viable product
      * Customer may be able to use software earlier and benefit from it
      * Feedback
  + Problems
    - Process is not visible
      * Harder to measure progress
      * Not cost-effective to document every single version of the system
    - Structure degrades
      * Have to spend time and money to refactor
      * More changes become more difficult and costly
* Integration



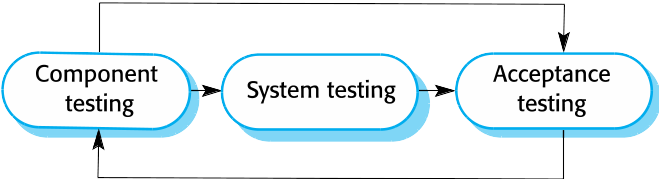
* + Stand-alone Application Systems(COTS) are configured for particular environment
  + Collections of objects that are developed as a package to be integrated with a component framework like .NET or J2EE
  + Web services hosted by someone else
  + Advantages
    - Reduced cost(less dev costs)
    - Faster delivery and deployment
  + Disadvantages
    - Requirements compromises are inevitable
    - May not meet all the needs
    - Loss of control over evolution of reused system elements
* Process Activities
  + Real processes are
  + 4 basic process activities
    - ***Specification, Development, Validation, Evolution***
    - Specs

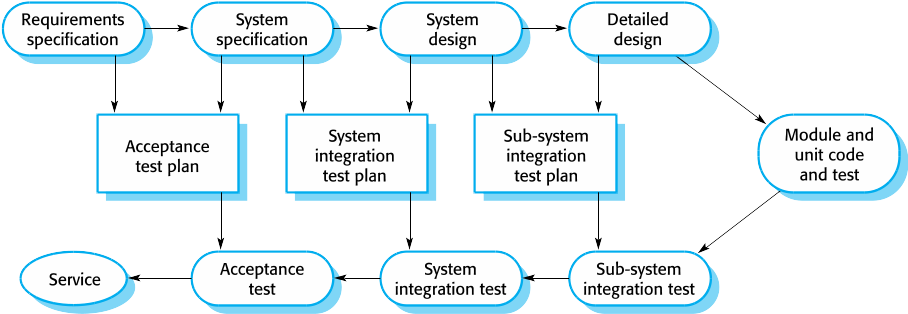


* + - Development/Implementation

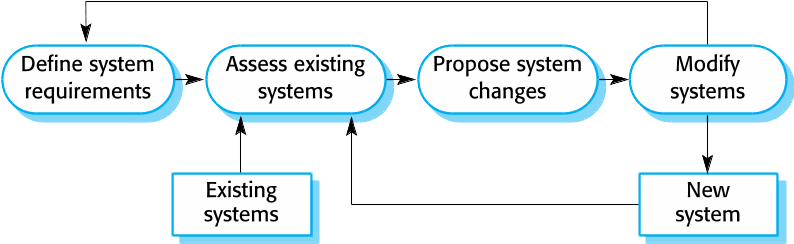


* + - Validation





* + - Evolution



* 9/13/17
* Coping with change
  + All large projects will change
    - Business changes will change requirements
    - New tech, new possibilities
    - Change platform
  + Changes lead to rework, so costs include new requirements and new implementation
* Reducing rework Costs
  + Change anticipation
    - Framework+key features have room to slot new things
  + Change tolerance
  + Change avoidance
* Coping with new requirements
  + System prototyping: check customer’s requirements with prototype
    - Good for change anticipation
  + Incremental delivery: small parts are delivered to customer for comment and experimentation
    - Good for change avoidance and change tolerance
  + Refactoring: improve structure and organization of program
    - Good for change tolerance
* Prototyping
  + Initial version of system used to demonstrate concepts
  + Prototype can be used in:
    - Help with requirements elicitation and validation
    - Explore options and develop a UI design
    - Back-to-back tests
  + Benefits
    - Improved usability
    - Closer match to users real needs
    - Improved design quality
    - Improved maintainability
    - Reduced Development effort
      * Less wasted
  + Process
    - Establish objectives
      * Prototype plan
      * What are you testing, is this specific feature what you want
        + Ignore all else, need specific things to be tested
    - Define prototype functionality
      * Outline definition
      * What function should this prototype have
        + Start with fake data, just very basic
    - Develop prototype
      * Executable
  + Development
    - May be based on rapid prototyping languages or tools
    - May leave out functions
      * Focus on areas of the product that are not well understood
      * Error checking and recovery may not be included
      * Focus on functions, not reliability and security
        + Functional requirement: thing it does, can write test cases for
        + Non-function requirements: memory, cpu requirements, performance…
  + Throwaway prototypes
    - Prototypes should be discarded
      * May be impossible to tune for non-functional requirements
      * Usually undocumented
      * Bad structure from rapid change
      * Probably won’t meet quality standards
* Incremental Delivery
  + Break delivery down into increments where each increment has part of the required functions
  + User requirements are prioritised, high priority are included early
    - Allows better testing of high priority features
  + Once development on an increment is started, requirements are frozen.
    - Requirements for later increments can continue to evolve
  + Incremental development
    - Develop system in increments and evaluate each increment before proceeding
    - Normal approach used by agile methods
    - Evaluation done by proxy
  + Incremental Delivery
    - Deploy increment for use by end-users
    - More realistic evaluation of practical use of software
    - Difficult to implement as a replacement system because it is missing some features
  + Advantages
    - More feedback, more accurate feedback
    - Customer can benefit from development earlier
    - Lower risk of overall project failure
    - Highest priority system services tend to receive most testing
      * In use longest
  + Disadvantages
    - Most systems have basic features common to many specific features
      * Will have to refactor and retrofit many things to be more general
    - Specification is developed in conjunction with software
      * Hard to predict specification
      * Hard to know what the final project will look like
* Process Improvement
  + Improving the process of development enhances software quality
    - Reduces cost, accelerate development
  + Process measurement
    - Need metrics to measure improvement
      * Time for completion
      * Bug reports
        + Not number found, but number in code
    - Care about objective, not metric
      * Don’t hide/pad to make it look better
  + Maturity approach
  + Agile approach
* 9/15/17 ***START HERE FOR QUIZ 2***
* Agile Software Development
  + Rapid development is now usually the most important requirement for software systems
    - Fast changing environment makes it almost impossible to produce stable software requirements
  + Agile development
    - Specification, design, and implementation are interleaved
    - Series of versions or increments
    - Frequent delivery of new versions
    - Extensive tool support (automated testing)
    - Minimal documentation - focus on code, not docs
* Agile Methods
  + Core theme:
    - Focus on code rather than the design
    - Based on an iterative approach
    - Intended to deliver working software quickly and evolve this quickly to meet changing requirements
  + 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
    - Customer involvement
    - Incremental Delivery
    - People not process - work your way, not under a set process
    - Embrace change
    - Maintain Simplicity
  + Applicability
    - Developing small or medium products
      * Almost all software
    - System development within an organization where customer is involved
* Agile Techniques
  + Extreme programming
    - New versions several times a day
    - Incremental delivery every 2 weeks
    - All tests for all builds, build only accepted if all tests are successful
    - Release cycle
      * Select user stories
      * Break stories into tasks
      * Plan release
      * develop/integrate/test
      * Release software
      * Evaluate system
    - Practices
      * Incremental planning
        + Requirements recorded on story cards
      * Small releases
      * Simple design
      * Test-first development
        + Write the tests before the code
      * Refactoring
        + All developers refactor often
        + As soon as you see a way to improve, you do it
      * Pair programming
        + Developers work in pairs, check each other’s work
      * Collective ownership
        + Anyone can change anything, no islands of expertise
      * Continuous integration
        + As soon as a task is complete, it is integrated to the system

All tests in system must pass

* + - * Sustainable pace
        + No large amounts of overtime

Usually reduces code quality

* + - * On-site customer
        + Should always be available for developers
        + Member of the team
    - Influence of Extreme Programming
      * User stories
      * Refactoring
        + Slightly less
      * Test-first development
      * Pair programming
    - User Stories (BOOK EXAMPLE)
      * Customer gives an example of interaction with the software
        + Write in in plain english describing what you do and what the software does in response
      * Pull out each individual task from the story
        + Put it on a task card
      * Create test card
        + Write test cases

Write code

* + - User Scenario (HOW IT USUALLY WORKS/TEACHER EXAMPLE)
      * Like stories but more formal
      * Stories become:
        + As a \_\_\_\_\_\_\_\_\_\_\_\_ (role)
        + I want to \_\_\_\_\_\_\_\_\_ (suggestion on how to accomplish)
        + So i can \_\_\_\_\_\_\_\_\_ (goal)
      * Point is to open a dialog between customer and developer
      * User scenario vs user story vs use case
        + Use case: role and action
        + User story: reason and goal of task with suggestion of methods/reason
        + User scenario: user action and system response
    - Refactoring
      * According to most systems: Design for change
        + Worth spending time to anticipate change, reduces costs later
      * XP maintains that is not worthwhile, can’t anticipate the changes
        + Constant refactoring to make it easier to change later
      * Team looks at possible improvements and makes them even if there is no immediate need
      * Improves understandability of software
        + Reduces need for documentation
      * Changes are easier because code is structured and clear
      * Some changes require architecture refactoring and that is more expensive
    - Test-first development
      * Testing features
        + Incremental test development
        + Automated test harnesses to run all component tests each time that a new release is built
    - Test-driven development
      * Clarifies requirements to be implemented
      * Tests written as programs so they can be automated
        + Usually a testing framework
      * All previous and new tests are run automatically when new functionality is added
        + No new errors
* 9/18/17
  + - Customer Involvement
      * During testing, help develop tests for stories
      * Actually writes the tests
      * Some customers may have limited time and will give requirements, but not tests
    - Test automation
      * Programmers are lazy, rather not write code
      * UI tests are difficult
      * Hard to judge completeness of a set of tests
        + Tests tell you if there is an issue, not if there isn’t one
    - Pair programming
      * Work with someone else, both know the code, both will spot bugs
  + Agile Project management
    - Scrum
      * Agile method focuses on managing iterative development rather than specific agile practices
      * 3 phases
        + Outline planning phase, general objectives for project
        + Series of sprint cycles, each cycle develops an increment
        + Evaluation and documentation. Evaluate sprint progress
      * Terminology
        + Development: 5-7 people
        + Potentially shippable Product: product at end of sprint, is stable, with everything planned for the sprint, could ship
        + Product Backlog: list of ToDo items for final product
        + Product Owner: customer, defines requirements. Sets priorities. Can be customer, or a stakeholder
        + Scrum or Standup: daily meeting of team to review progress and priority, should be short and face-to-face
        + ScrumMaster: Manager of the scrum process
        + Sprint: development iteration, usually 2-4 weeks
        + Velocity: estimate of how much backlog that can be covered in a single sprint
      * Sprint Cycle
        + Fixed length (2-4 weeks)
        + During this period team is isolated from customer and organization. Scrum master protects from external distractions
        + Daily meetings (Scrums) members share info, describe progress and challenges

Re-plan short term work

* + - * Benefits
        + Product is broken down into manageable chunks
        + Unstable requirements don’t slow down progress
        + How team can see everything
        + Customer gets increments on-time, we get feedback
        + More trust between customers and developers
* Scaling agile methods
  + Agile works great with small teams
  + Attributed to improved communication
  + Scaling Out
    - Agile across large organization
  + Scaling Up
    - Developing large systems not feasible for small teams
  + Problems
    - Informality of agile doesn’t work with legal contracts used by most companies
    - Agile designed for teams closeby
      * Many projects have worldwide teams
    - Agile is most appropriate for new software, not maintenance
      * Majority of software costs for large companies come from maintenance
  + Contractual Issues
    - Most contracts for software are based around a specification
      * Defines what has to be implemented for customer
    - Doesn’t allow for interleaving specification and development
    - Contract paying for time rather than functions is needed
  + Software Maintenance
    - More expensive than development usually
    - 2 issues
      * Are agile systems maintainable?
        + Emphasis on development process of minimizing formal documentation
      * Can agile methods be used effectively for evolving systems?
    - Key problems
      * Lack of documentation
      * Keeping customers involved in development process
      * Maintaining continuity of the development team
    - Agile relies on team knowing and understanding system
    - Long-lifetime systems, original developers may not be working on the code
  + Most projects are a combination of plan-driven and agile
    - Is it important to have detailed specification before implementation
      * Lean plan-driven
    - Is incremental deliver with rapid feedback needed
      * Lean agile
  + Agile methods for large system
    - System of systems
      * Most large projects are just a collection of smaller systems
        + Each team develop independently
    - Brownfield Development
      * Brownfield system is a system that includes and interact with a bunch of other existing systems
    - System Configuration
      * Need to configure everything to be consistent, even if you don’t know the features
    - Regulatory constraints
      * Must fit all regulations
    - Diverse stakeholders
      * Everyone has a different priority
    - Prolonged procurement
      * Big systems take longer, will lose and gain people over time
  + Scaling up to large systems
    - Not possible to be completely incremental
    - Multiple owners/ customer representative
    - Need mechanisms for communicating with different teams
  + Multi-team scrum
    - Role replication
      * Each team has a scrum master
      * Each team has a product owner
    - Product Architects
      * Each team has one, works with other to collaborate to design and evolve entire system
    - Release alignment
      * Release dates need to be aligned
    - Scrum of Scrums
      * Daily meeting where reps from each team meet
* 9/20/17
* 9/22/17
  + Kanban
    - Group each task into one of 6 columns
      * To do, plan, develop, test, deploy, done
      * Can use other columns
    - Color code tasks into a category
      * User story, defect, task, feature
      * Give each a point value (usually time in hours)
    - Swimlane
      * Each person moves the tasks they are working on they move them into their own swimlane
    - Work in Progress Limit (WIP Limit)
      * Max number of tasks per column
      * Helps prevent bottlenecks
    - Blocker
      * Task can’t continue without other work being finished
        + Can move it to not affect WIP limit
      * After dependency is finished, it can be moved back in(with high priority)
  + Scrum vs Kanban
    - Scrum
      * Small teams (cross-functional)
      * User stories sorted by priority
      * Potentially shippable code at end of sprint
      * Limits WIP by using batch tasks
      * inspection/review at end of each sprint
        + Optimize plan and change priority
    - Kanban
      * Visual workflow
      * Limits based on column, not total project
      * Measures lead time
        + Lead time: Average time to complete an item (from:to-do to:done)
        + Cycle time: in progress to done
    - Both
      * Uses cycle and lead time to optimize the process
      * Both metrics should be small and predictable
      * Are both agile (Kanban is more agile)
  + Scrumban
    - Product vision stays the same
    - Product Backlog
      * Becomes to-do column
      * Global backlog for entire project
    - Sprint backlog
      * ready/plan column
      * Tasks to be executed next
    - Daily standup
      * Shorter, everyone can see what is happening
      * Mostly for blocked items
    - Sprint Cycle
      * Gone
      * Continuous flow