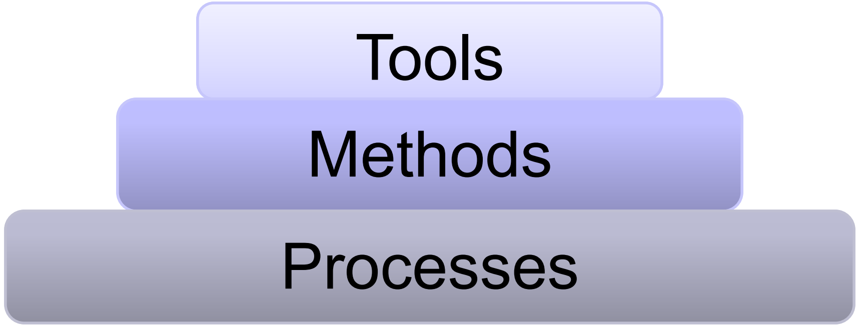
Introduction to Software Engineering

ECSE-321

Unit 3 – Software Processes

Software Engineering Layers



Tools

Methods

Processes

- ***Processes***: frameworks for specifying the required tasks (e.g., waterfall, extreme programming)
- ***Methods***: how the tasks are done (e.g., design review, code review, testing)
- ***Tools***: automating processes and methods

Software Process

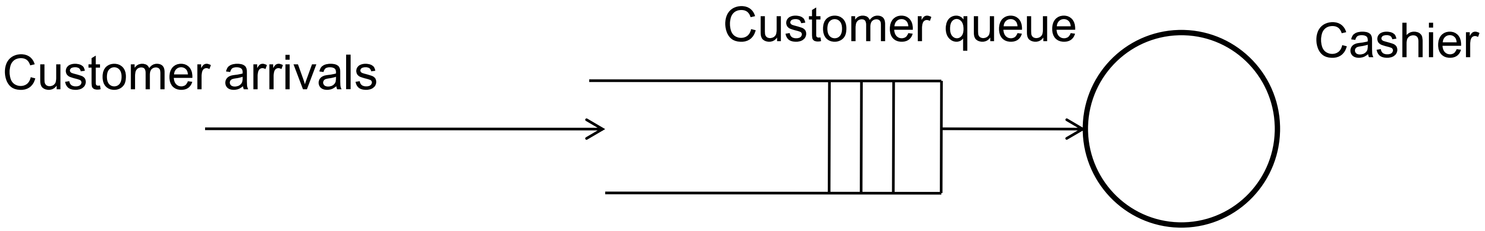
- Large-scale projects follow recognized stages (start to finish)
  - allows to progress tracking
  - resource management
- In software development, these “stages”
  - arrived at by trial and error
  - leveraging accumulated wisdom

Aside: What is a Process?

Customers

Cashier

Physical view of a checkout process


Customer arrivals

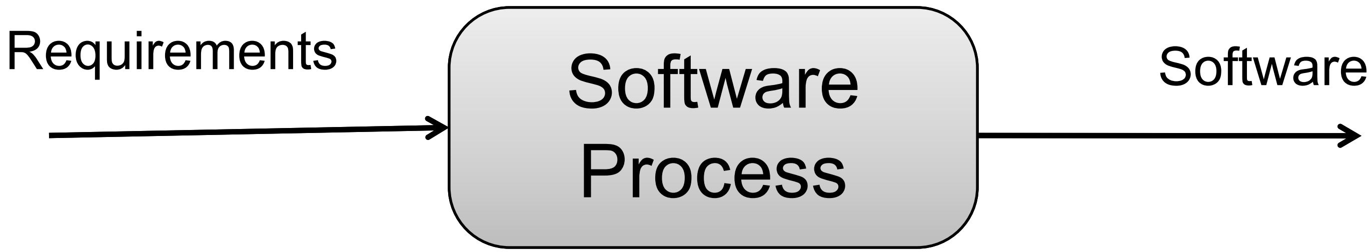
Customer queue

Cashier

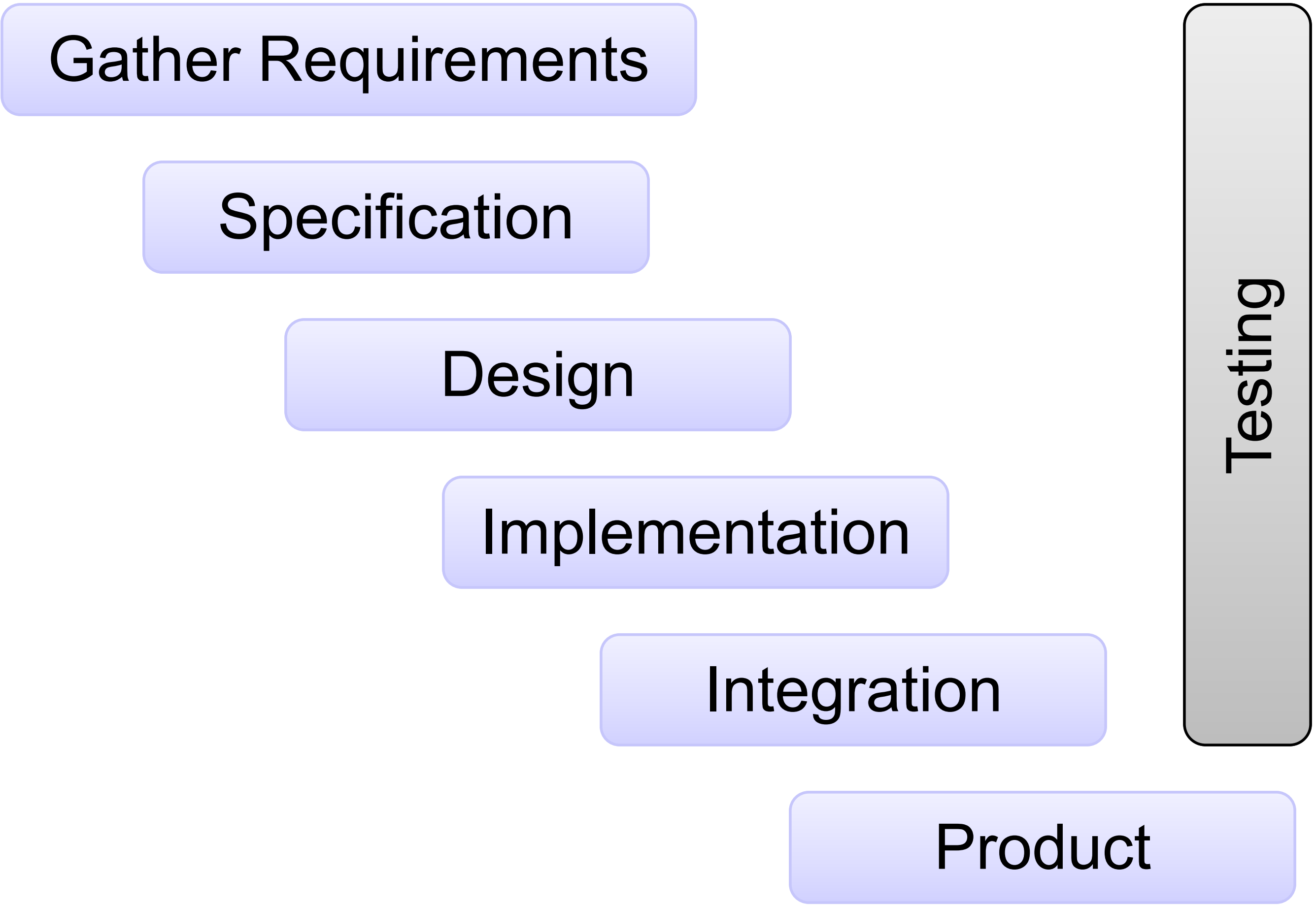
Abstract view of the same checkout process

Software Process

- A definition
  - Set of software engineering activities needed to transform requirements to software



Waterfall Process Phases



Gather Requirements

Specification

Design

Implementation

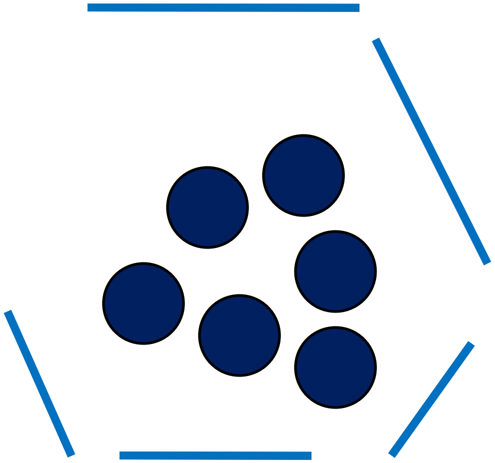
Integration

Product

Testing

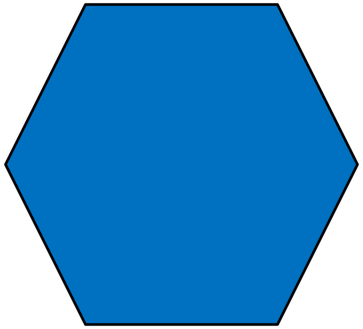
Gather Requirements

- Figure out what the "system" is supposed to do
  - Write down the list of features
  - List of constraints.. etc
- Good idea to talk to users, clients
  - They may not know the exact requirements either
- Purpose
  - Ensure we are building the right thing
  - Gather information for planning



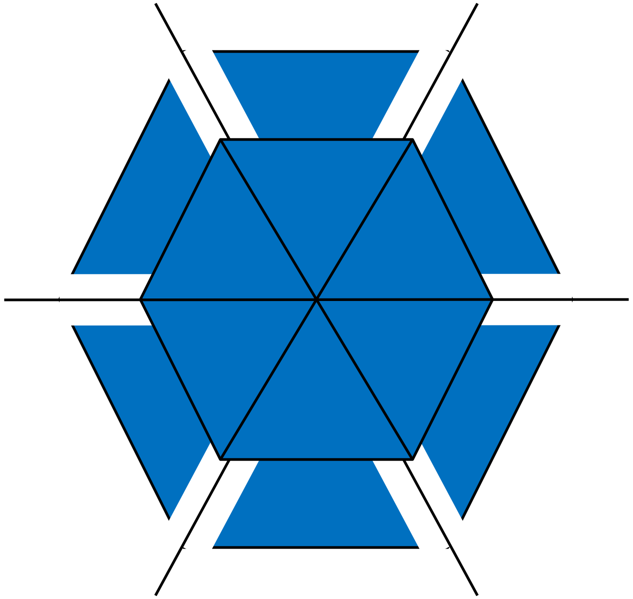
Specification

- A written document
- Contains
  - what the system does under all conceivable conditions
  - should cover all inputs and possible states
- More complete than requirements



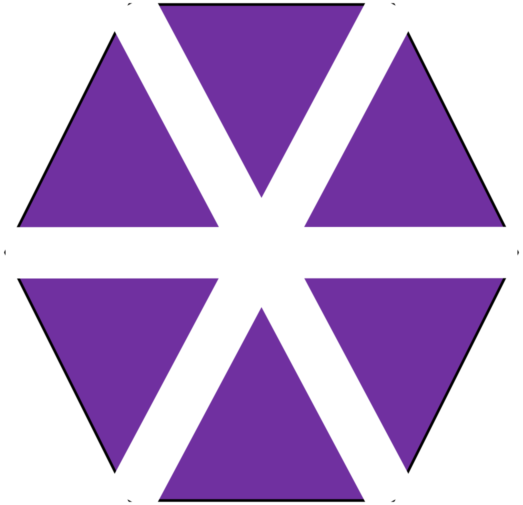
Design

- Develop a system architecture
- Decompose system into modules
- Specify interfaces between modules
- Based more on how "*system works*"



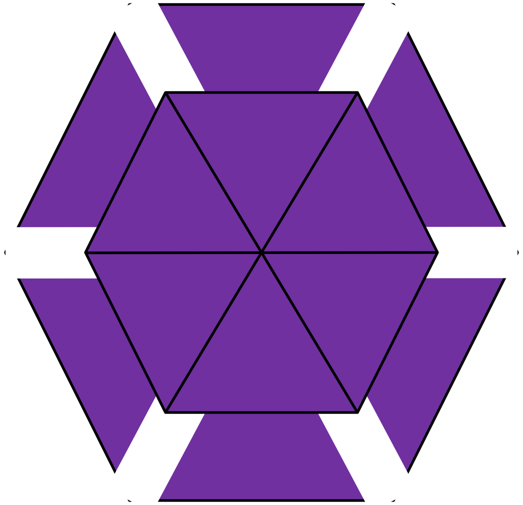
Implementation

- Code the design
- Coding is an extensive process
  - need a plan
  - prioritize activities
  - testability is a consideration in prioritization
- Test as modules are built



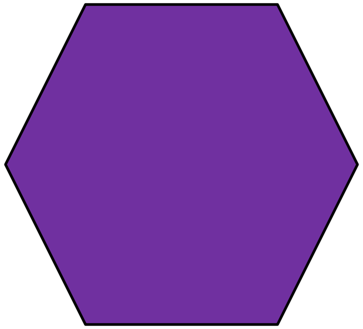
Integration

- Put the pieces together
- Test the entire system



Product

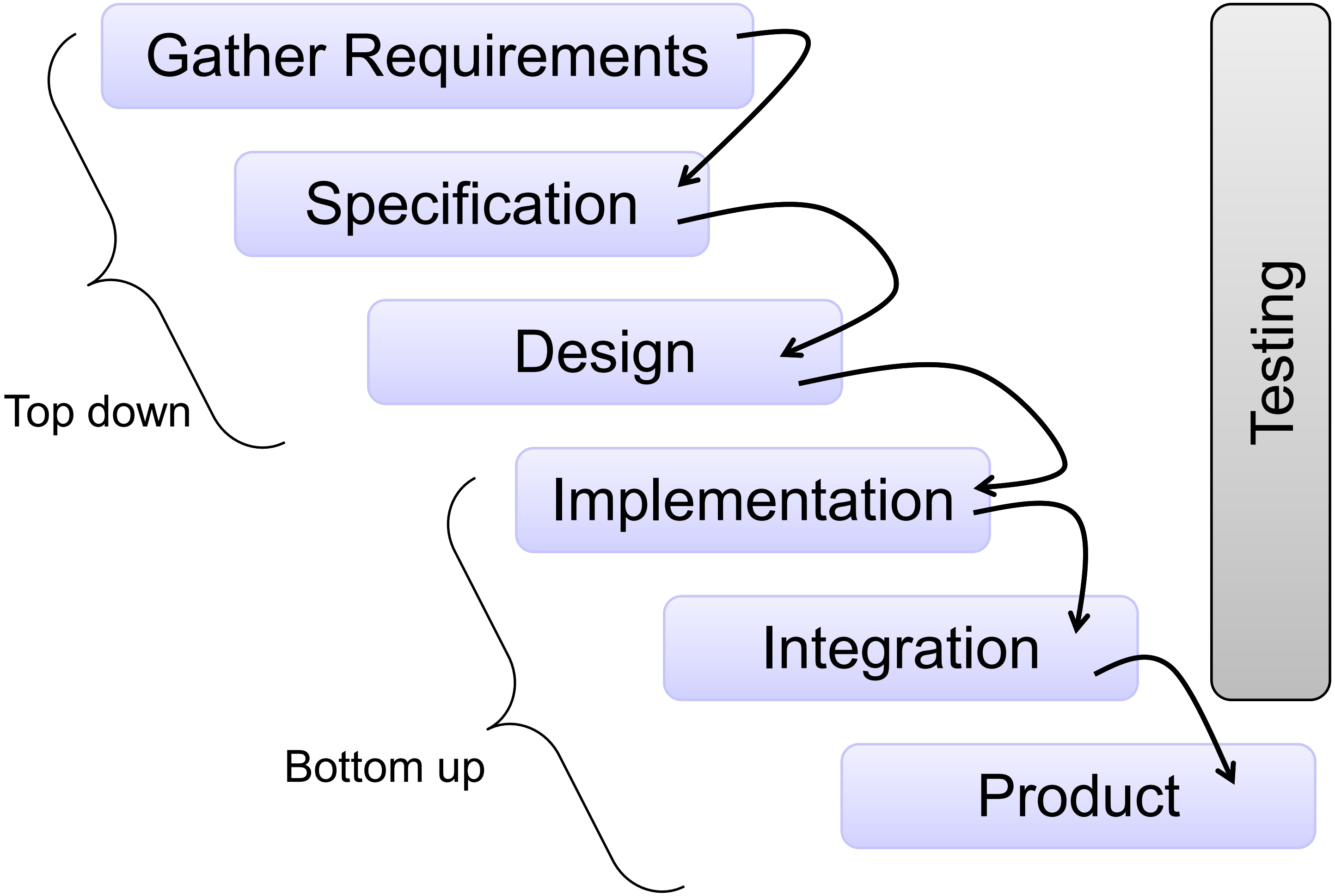
- Ship product – development phase of the project done!
- Maintenance begins



Above Software Process

- Called the waterfall model
  - one of the standard models for developing software
- Each stage leads to the next
  - No iteration or feedback between stages

Waterfall Model



Waterfall Model – Discussion

- What are major drawbacks of waterfall?
  - Relies heavily on accurate requirement assessment
  - Little feedback from users until product developed
  - User feedback arrives very late for large projects
  - Sequential project planning

Waterfall Model – Discussion

- Waterfall model closer to the process of building a skyscraper/bridge
- Your ideas on applicability of waterfall for software development

Good Aspects of Waterfall

- Emphasis on specification, design, and testing
- Emphasis on communication through written documents

Bad Aspects of Waterfall

- Time
  - Delay in getting feedback from users
  - Delay in incorporating changes
- Software development process need to factor in change
  - changes in underlying changes
  - changes in competing products
  - availability of 3rd party software/hardware elements
- ***Reducing development time*** is one way to reduce the number of changes!

“Fast” Software Development

- Short time scales
  - world changes less.. requirements remain valid
- Fast simplifies planning
  - have short-term predictions that are more reliable
  - cost overruns can be controlled
- Waterfall model is not suitable for fast development

Faster Processes: Rapid Prototyping

- Write a quick prototype
- Use the prototype to get user feedback
  - requirements can be refined using the prototype
- Then proceed as in waterfall model
  - throw away the prototype
  - do spec, design, coding, integration, etc

Problems with Rapid Prototyping

- Hard to throw away the prototype
  - “Prototype is the product”
  - Happens more often than you think!
- Advantages:
  - useful in refining the requirements
  - provides a non-abstract framework to get feedback
  - exposes design mistakes
  - experience of building the prototype can improve the logistics of building the final product

Reality Can be Different

- Reality – feedback is pervasive
  - Specification stage might provide feedback for refining requirements
  - Design stage can provide feedback to refine specification
  - Coding issues can affect design
  - Final product characteristics can lead to requirement changes
- Waterfall model with "feedback loops"

What is the Answer?

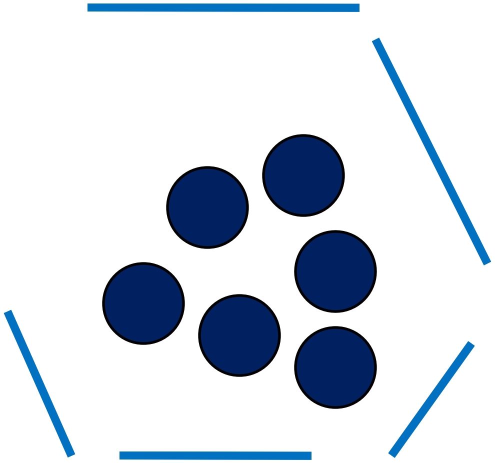
- Accept that feedback from later stages can change earlier decisions
- Build a flexible process – that accepts late feedback

Iterative Models: Plan for Change

- Adapt waterfall model for changes
- Plan to iterate the whole cycle several times
  - each cycle leads to a “build”
  - each cycle is smaller and lightweight compared to a normal “waterfall”
- Break the normal system into a series of progressively complete system

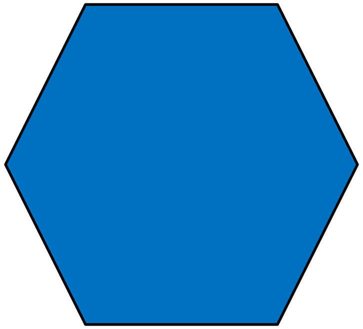
Gather Requirements

- Same idea as the waterfall
- Talk to customers and find out what is needed
- Recognize *diminishing returns*
- Important to show something to elicit full requirements



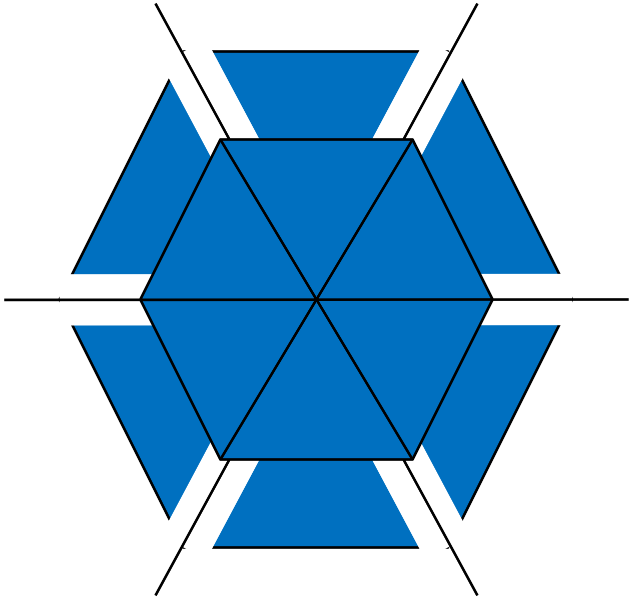
Specification

- Written document containing:
  - what the system does under all conceivable conditions
  - should cover all inputs and possible states
- Still important
- Can evolve with time
  - important to recognize aspects of the specifications that can change



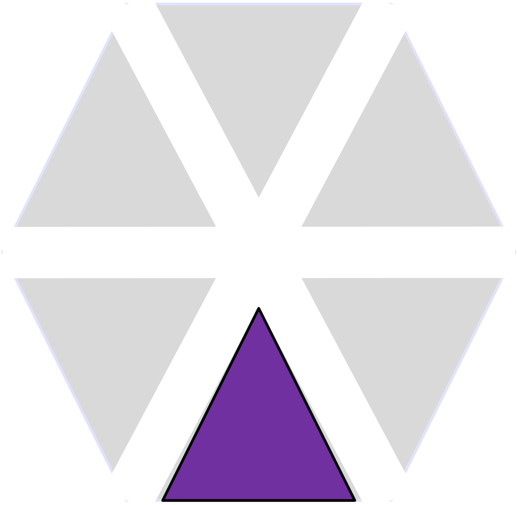
Design

- Decompose system into modules
- Design for change
- Which parts are most likely to change?
  - put suitable abstractions there



Incremental System Design

- Plan incremental development of each module
- Skeletal component to full functionality
- From most critical to least critical features



Implementation: Build 1

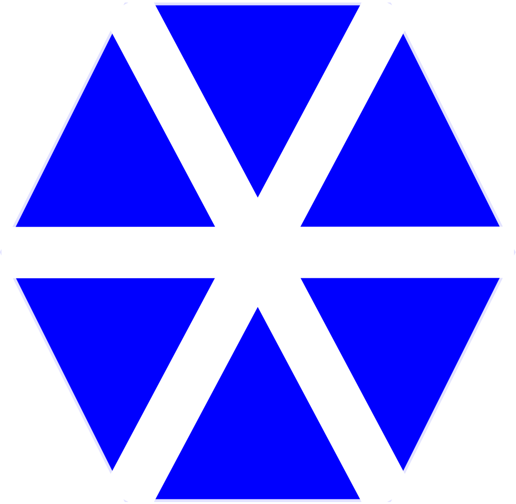
- Get skeletal system working
- Components are there.. but none of them are complete
- Interfaces between components are implemented

Implementation: Build 1

- Well defined interfaces allow
  - complete system to be built in an iterative manner
  - individual components rely on all interfaces of other components

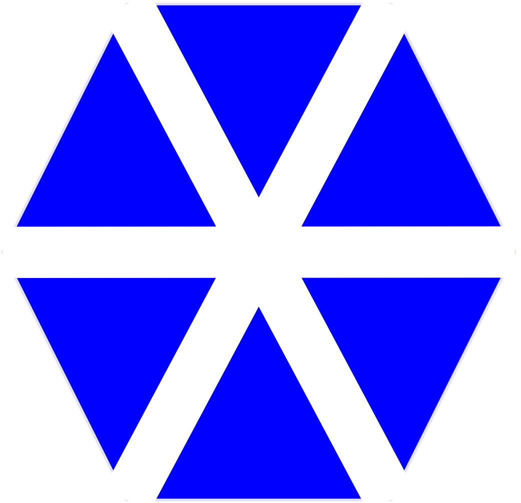
Implementation: After Build 1

- Have a demo to show
  - to customers
  - to team for communication
- Each subsequent build adds more functionality



Integration

- Integration and major test for each build
- Stabilization point might be releasing point of each build
- Iterate until the last build
- Earlier builds can ship to customers



Advantages of Iterative Builds

- Problems found sooner
  - Get earlier feedback from users
  - Ger earlier feedback on whether spec/design are feasible
- More quantifiable project management
  - Build 3 or 4 means X% progress in product development
  - Difficult to quantify the completion in different stages of the waterfall (resource requirements can be highly non-uniform)

Disadvantages of Iterative Builds

- Risk of making a mistake in requirements, spec, or design exists
  - Time before build 1 is reduced – less time is spent in those processes
  - Implementation starts earlier
  - (To a limited extent requirements, spec, or design can be refined – large changes can be costly)
- Trade-off against the risks of being too slow
  - better to get something working

Iterative Builds used in Practice

- Many actual projects use the iterative model
  - Daily builds
  - System is always working
  - Mozilla, Microsoft are examples
- Systems that are hard to test use something like a waterfall model
  - E.g., space probes

Conclusions

- Important to follow a good process
- Waterfall
  - top-down design, bottom-up implementation
  - lots of up front thinking, slow, hard to iterate
- Iterative building
  - build a prototype quickly, then evolve it
  - postpone some of the thinking