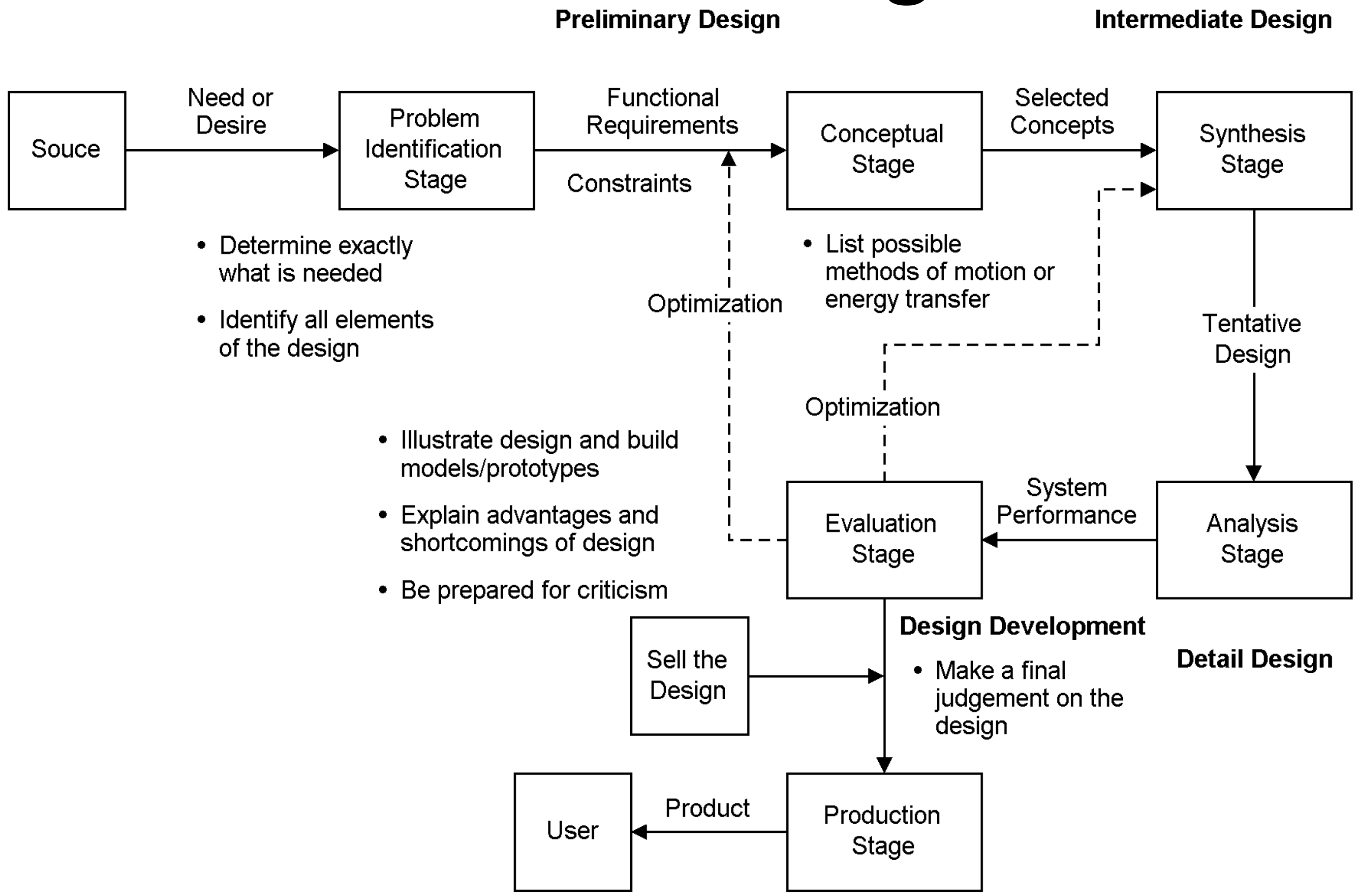
Elec 4309 Senior Design

Wendell H Chun

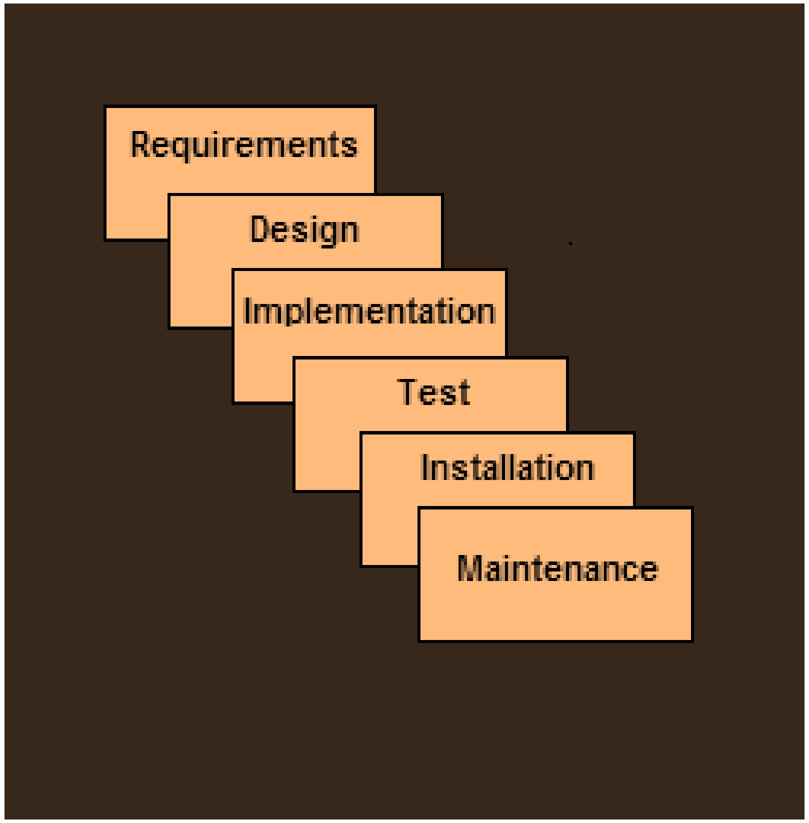
Sep. 26, 2017

The Traditional Design Process



Preliminary Design
Intermediate Design
Souce
Need or Desire
Problem Identification Stage
Functional Requirements
Constraints
Conceptual Stage
Selected Concepts
Synthesis Stage
• Determine exactly what is needed
• Identify all elements of the design
Optimization
• List possible methods of motion or energy transfer
Tentative Design
Optimization
• Illustrate design and build models/prototypes
• Explain advantages and shortcomings of design
• Be prepared for criticism
Evaluation Stage
System Performance
Analysis Stage
Design Development
• Make a final judgement on the design
Detail Design
Sell the Design
User
Product
Production Stage

Waterfall Model



- **Requirements** – defines needed information, function, behavior, performance and interfaces.
- **Design** – data structures, software architecture, electrical design, mechanical design, interface representations, algorithmic details.
- **Implementation** – source code, database, user documentation, testing.

Waterfall Strengths

- Easy to understand, easy to use
- Provides structure to inexperienced staff
- Milestones are well understood
- Sets requirements stability
- Good for management control (plan, staff, track)
- Works well when quality is more important than cost or schedule

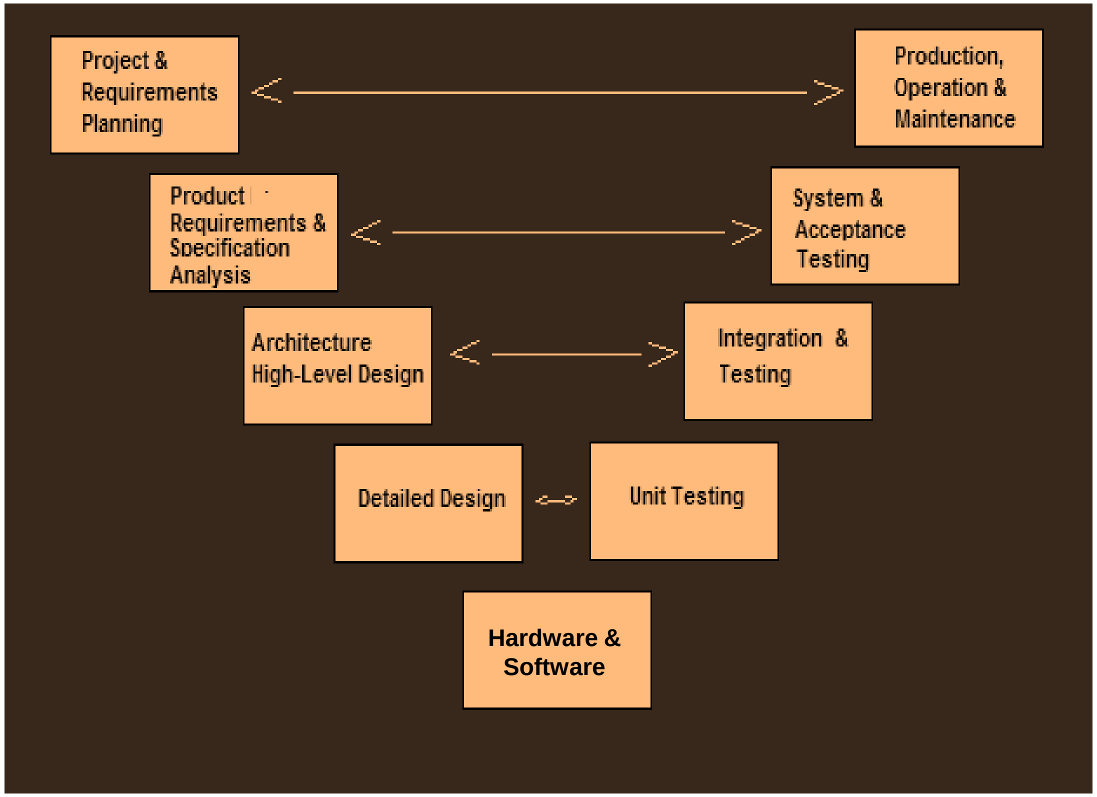
Waterfall Deficiencies

- All requirements must be known upfront
- Deliverables created for each phase are considered frozen – inhibits flexibility
- Can give a false impression of progress
- Does not reflect problem-solving nature of development – iterations of phases
- Integration is one big bang at the end
- Little opportunity for customer to preview the system (until it may be too late)

When to Use the Waterfall Model

- Requirements are very well known
- Product definition is stable
- Technology is understood
- New version of an existing product
- Porting an existing product to a new platform.

V-Shaped Model

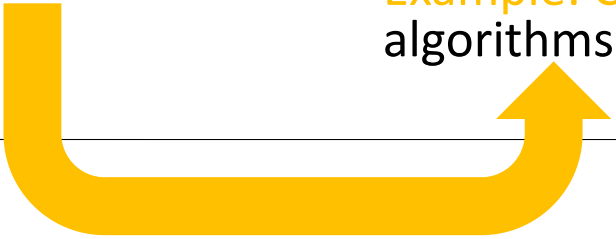

Project & Requirements Planning
Production, Operation & Maintenance
Product Requirements & Specification Analysis
System & Acceptance Testing
Architecture High-Level Design
Integration & Testing
Detailed Design
Unit Testing
Hardware & Software

- A variant of the Waterfall that emphasizes the verification and validation of the product.
- Testing of the product is planned in parallel with a corresponding phase of development

V-Shaped Steps

- Project and Requirements Planning – allocate resources
- Product Requirements and Specification Analysis – complete specification of the hardware/software system
- Architecture or High-Level Design – defines how hardware and software functions fulfill the design
- Detailed Design – develop drawings and algorithms for each architectural component

- Production, operation and maintenance – provide for enhancement and corrections
- System and acceptance testing – check the entire system in its environment
- Integration and Testing – check that modules , components, and subsystems interconnect correctly
- Unit testing – check that each module acts as expected
- Example: Coding – transform algorithms into software



V-Shaped Strengths

- Emphasize planning for verification and validation of the product in early stages of product development
- Each deliverable must be testable
- Project management can track progress by milestones
- Easy to use

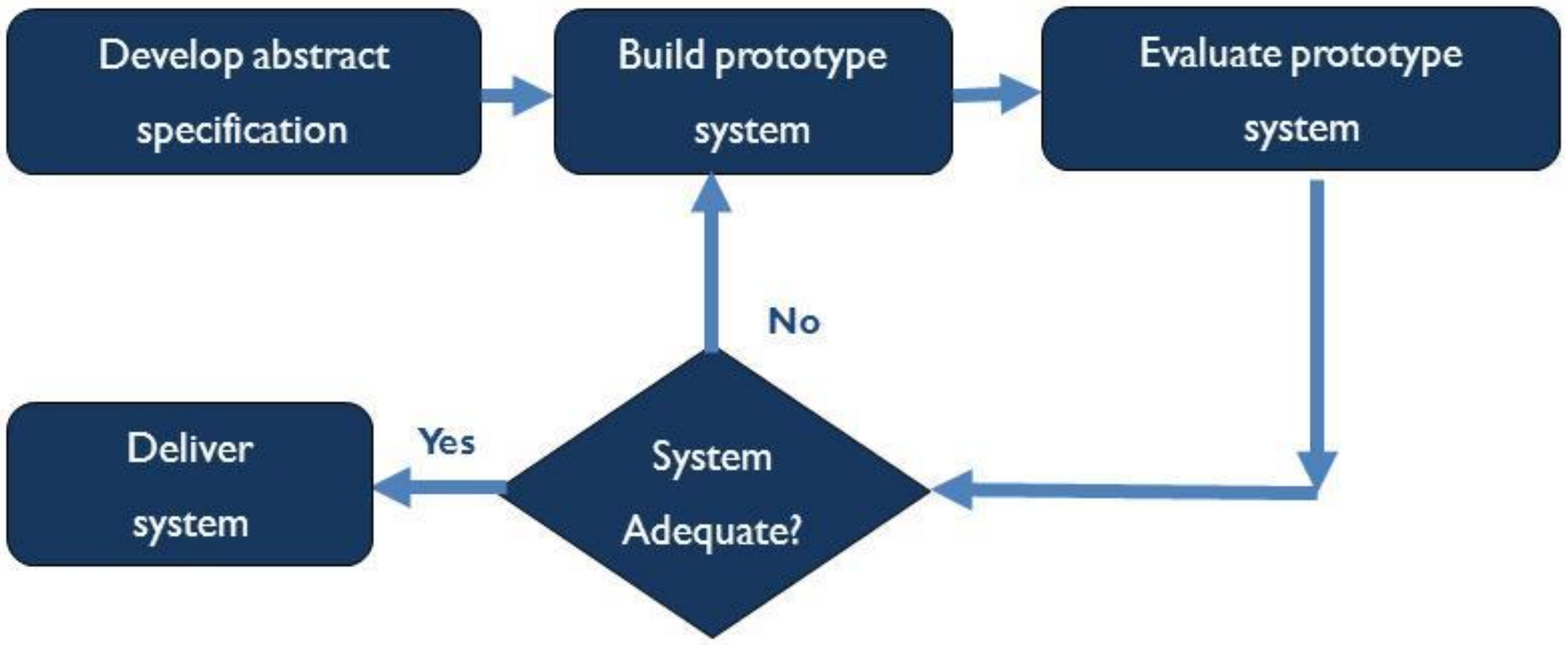
V-Shaped Weaknesses

- Does not easily handle concurrent events
- Does not handle iterations or phases
- Does not easily handle dynamic changes in requirements
- Does not contain risk analysis activities

When to Use the V-Shaped Model

- Excellent choice for systems requiring high reliability – hospital patient control applications
- All requirements are known up-front
- When it can be modified to handle changing requirements beyond analysis phase
- Solution and technology are known

Evolutionary Prototype Model



Structured Evolutionary Prototyping Model

- Developers build a prototype during the requirements phase
- Prototype is evaluated by end users
- Users give corrective feedback
- Developers further refine the prototype
- When the user is satisfied, the prototype is brought up to the standards needed for a final product.

Structured Evolutionary Prototyping Steps

- A preliminary project plan is developed
- An partial high-level paper model is created
- The model is source for a partial requirements specification
- A prototype is built with basic and critical attributes
- For software projects, the designer builds:
  - the database
  - user interface
  - algorithmic functions
- The designer demonstrates the prototype, the user evaluates for problems and suggests improvements.
- This loop continues until the user is satisfied

Structured Evolutionary Prototyping Strengths

- Customers can "see" the system requirements as they are being gathered
- Developers learn from customers
- A more accurate end product
- Unexpected requirements accommodated
- Allows for flexible design and development
- Steady, visible signs of progress produced
- Interaction with the prototype stimulates awareness of additional needed functionality

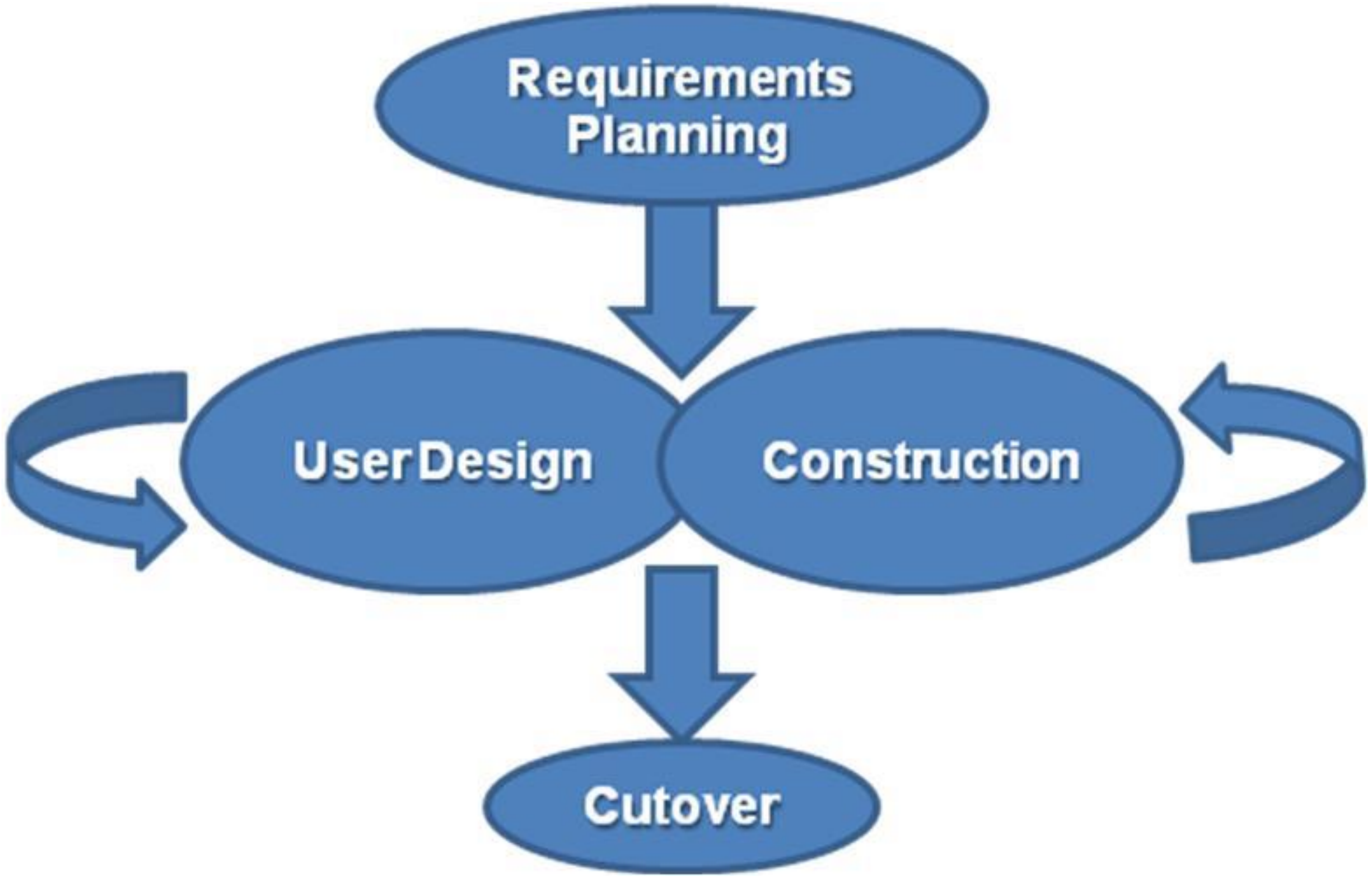
Structured Evolutionary Prototyping Weaknesses

- Tendency to abandon structured program development for "code-and-fix" development
- Bad reputation for "quick-and-dirty" methods
- Overall maintainability may be overlooked
- The customer may want the prototype delivered
- Process may continue forever (scope creep)

When to Use Evolutionary Prototyping

- Requirements are unstable or have to be clarified
- As the requirements clarification stage of a waterfall model
- Develop user interfaces
- Short-lived demonstrations
- New, original development
- With the analysis and design portions of object-oriented development.

Rapid Application Model



Rapid Application Model (RAD)

- Requirements planning phase (a workshop utilizing structured discussion of business problems)
- User description phase – automated tools capture information from users
- Construction phase – productivity tools, such as code generators, screen generators, etc. inside a time-box. ("Do until done")
- Cutover phase -- installation of the system, user acceptance testing and user training

RAD Strengths

- Reduced cycle time and improved productivity with fewer people means lower costs
- Time-box approach mitigates cost and schedule risk
- Customer involved throughout the complete cycle minimizes risk of not achieving customer satisfaction and business needs
- Focus moves from documentation to code, hardware, software (WYSIWYG)
- Uses modeling concepts to capture information about business, data, and processes.

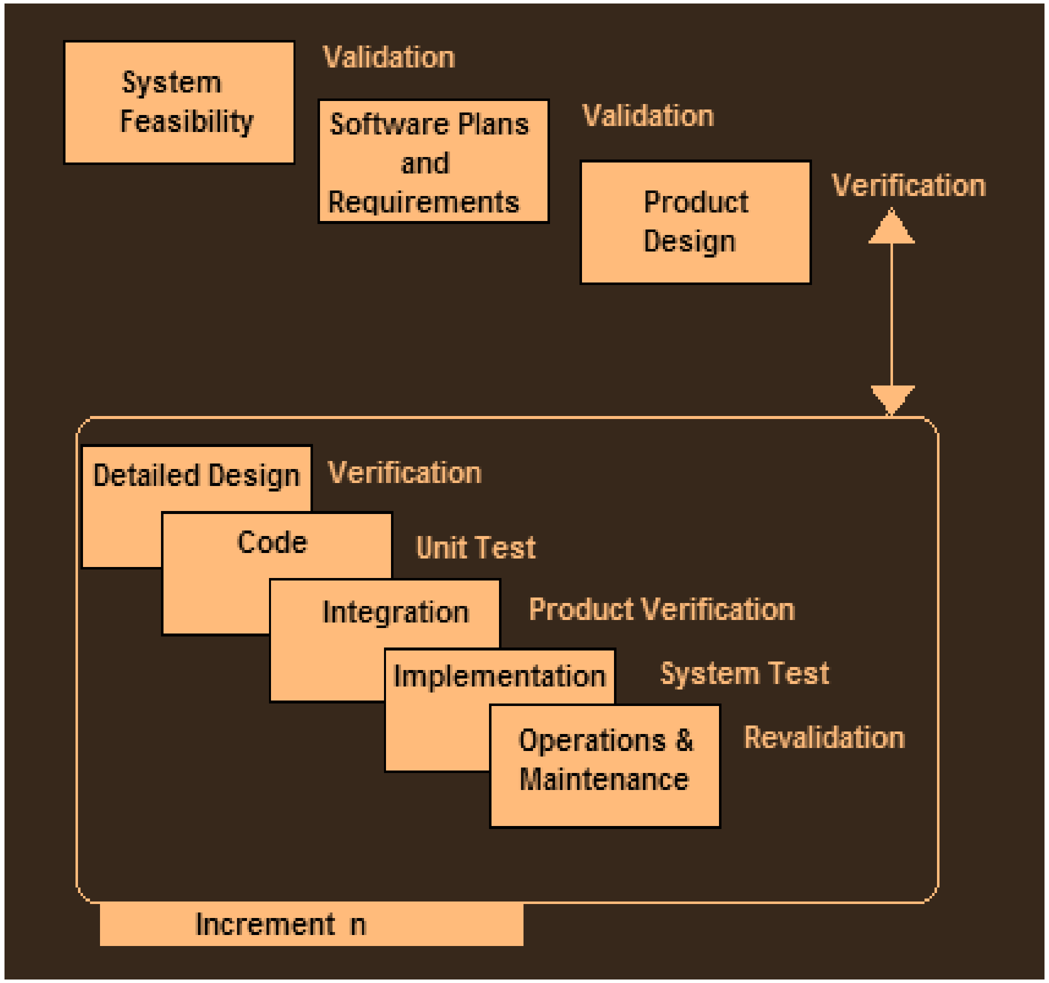
RAD Weaknesses

- Accelerated development process must give quick responses to the user
- Risk of never achieving closure
- Hard to use with legacy systems
- Requires a system that can be modularized
- Developers and customers must be committed to rapid-fire activities in an abbreviated time frame.

When to Use RAD

- Reasonably well-known requirements
- User involved throughout the life cycle
- Project can be time-boxed
- Functionality delivered in increments
- High performance not required
- Low technical risks
- System can be modularized

Incremental SDLC Model



- Construct a partial implementation of a total system
- Then slowly add increased functionality
- The incremental model prioritizes requirements of the system and then implements them in groups.
- Each subsequent release of the system adds function to the previous release, until all designed functionality has been implemented.

Incremental Model Strengths

- Develop high-risk or major functions first
- Each release delivers an operational product
- Customer can respond to each build
- Uses “divide and conquer” breakdown of tasks
- Lowers initial delivery cost
- Initial product delivery is faster
- Customers get important functionality early
- Risk of changing requirements is reduced

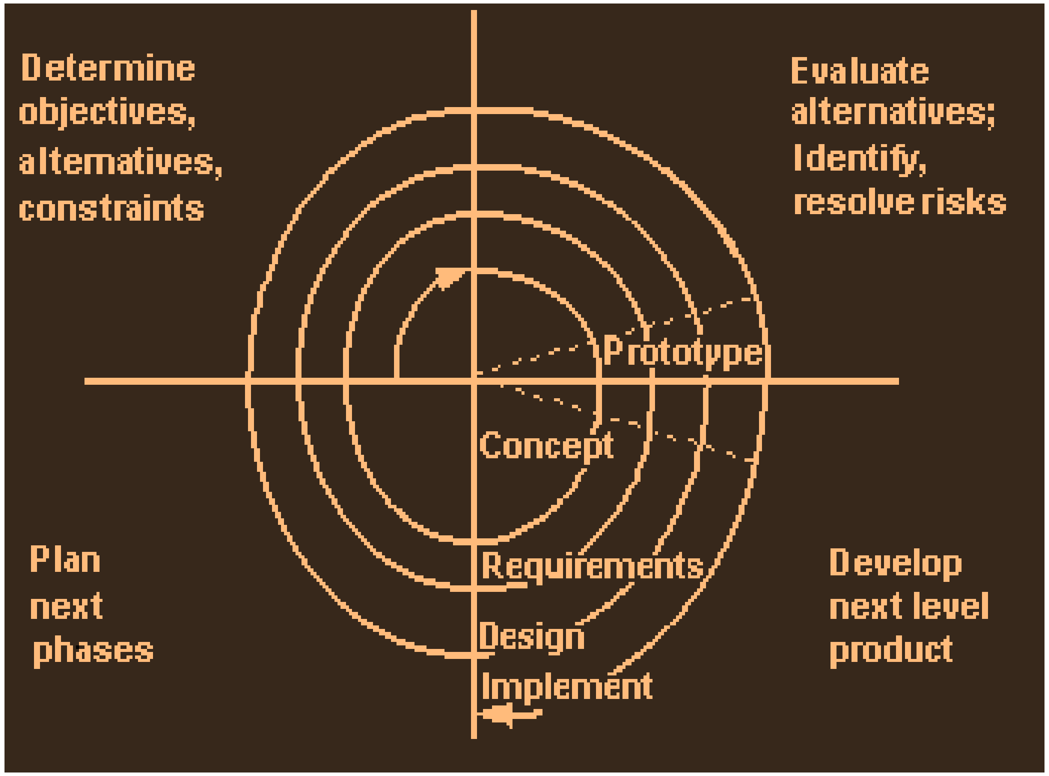
Incremental Model Weaknesses

- Requires good planning and design
- Requires early definition of a complete and fully functional system to allow for the definition of increments
- Well-defined module interfaces are required (some will be developed long before others)
- Total cost of the complete system is not lower

When to Use the Incremental Model

- Risk, funding, schedule, program complexity, or need for early realization of benefits.
- Most of the requirements are known up-front but are expected to evolve over time
- A need to get basic functionality to the market early
- On projects which have lengthy development schedules
- On a project with new technology

Spiral SDLC Model


Determine objectives, altematives, constraints
Evaluate alternatives; Identify, resolve risks
Prototype
Concept
Requirements
Design
Implement
Plan next phases
Develop next level product

- Adds risk analysis, and 4gl RAD prototyping to the waterfall model
- Each cycle involves the same sequence of steps as the waterfall process model

College of Engineering and Applied Science
UNIVERSITY OF COLORADO
DENVER | ANSCHUTZ MEDICAL CAMPUS

Spiral Quadrant

Determine objectives, alternatives and constraints

- Objectives: functionality, performance, hardware/software interface, critical success factors, etc.
- Alternatives: build, reuse, buy, sub-contract, etc.
- Constraints: cost, schedule, interface, etc.

Spiral Quadrant

Evaluate alternatives, identify and resolve risks

- Study alternatives relative to objectives and constraints
- Identify risks (lack of experience, new technology, tight schedules, poor process, etc.)
- Resolve risks (evaluate if money could be lost by continuing system development)

Spiral Quadrant

Develop next-level product

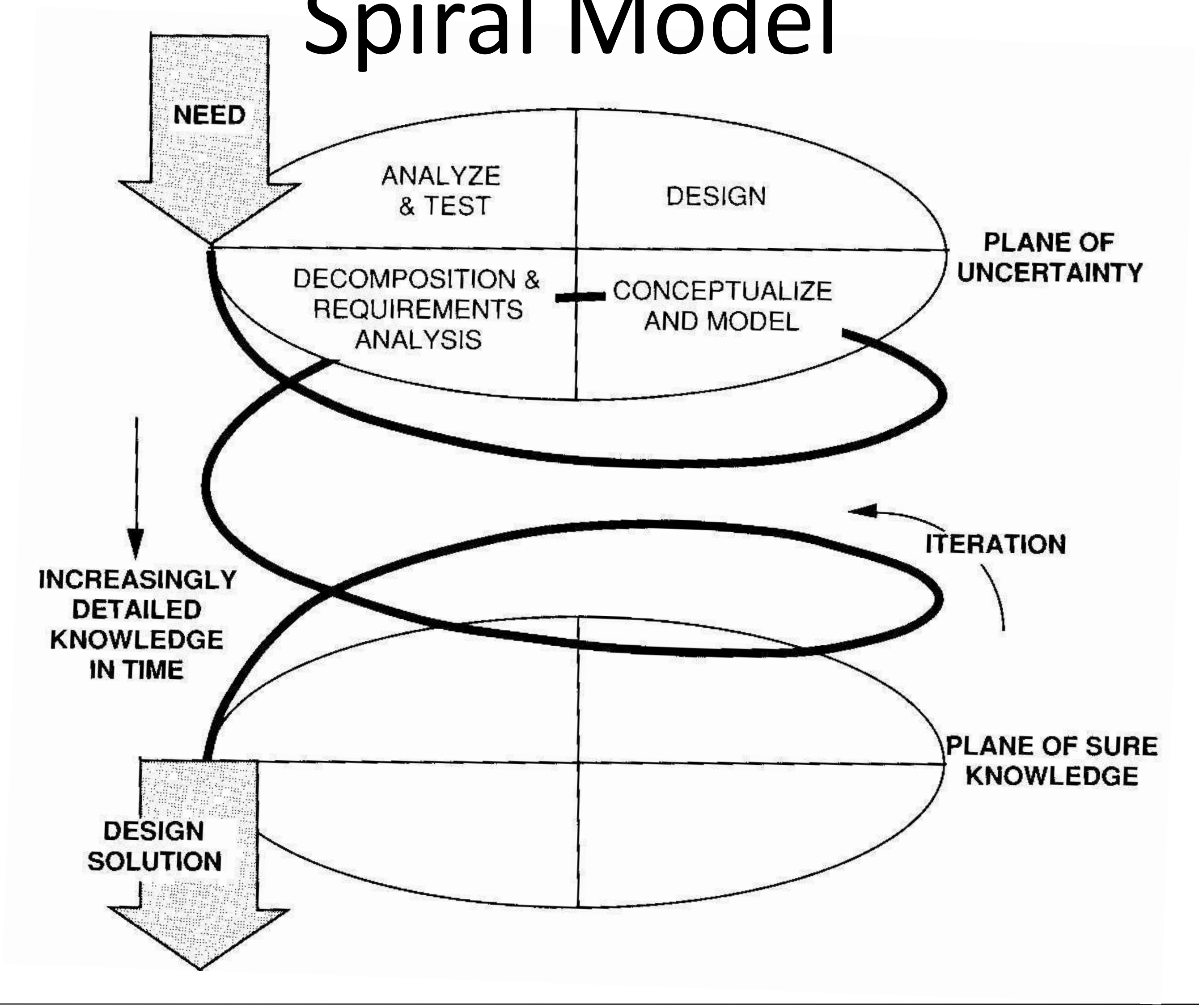
- Typical activities:
  - Create a design
  - Review design
  - Develop code or build parts
  - Inspect code or assemble hardware
  - Test product

Spiral Quadrant

Plan next phase

- Typical activities:
  - Develop project plan
  - Develop configuration management plan
  - Develop a test plan
  - Develop an installation plan

Spiral Model



Spiral Model Strengths

- Provides early indication of insurmountable risks, without much cost
- Users see the system early because of rapid prototyping tools
- Critical high-risk functions are developed first
- The design does not have to be perfect
- Users can be closely tied to all lifecycle steps
- Early and frequent feedback from users
- Cumulative costs assessed frequently

Spiral Model Weaknesses

- Time spent for evaluating risks too large for small or low-risk projects
- Time spent planning, resetting objectives, doing risk analysis and prototyping may be excessive
- The model is complex
- Risk assessment expertise is required
- Spiral may continue indefinitely
- Developers must be reassigned during non-development phase activities
- May be hard to define objective, verifiable milestones that indicate readiness to proceed through the next iteration

When to Use Spiral Model

- When creation of a prototype is appropriate
- When costs and risk evaluation is important
- For medium to high-risk projects
- Long-term project commitment unwise because of potential changes to economic priorities
- Users are unsure of their needs
- Requirements are complex
- New product line
- Significant changes are expected (research and exploration)

The “General” Design Process

1. Identify the problem
2. Define the working criteria/goals
3. Research and gather data
4. Brainstorm ideas
5. Analyze potential solutions
6. Develop and test models
7. Make decision
8. Communicate decision
9. Implement and commercialize decision
10. Perform post-implementation review

EE 4309

Top-Down Design

- If we look at a problem as a whole, it may seem impossible to solve because it is so complex. Examples:
  - writing a tax computation program
  - writing a word processor
- Complex problems can be solved using **top-down design**, also known as **stepwise refinement**, where
  - We break the problem into parts
  - Then break the parts into parts
  - Soon, each of the parts will be easy to do

Advantages of Top-Down Design

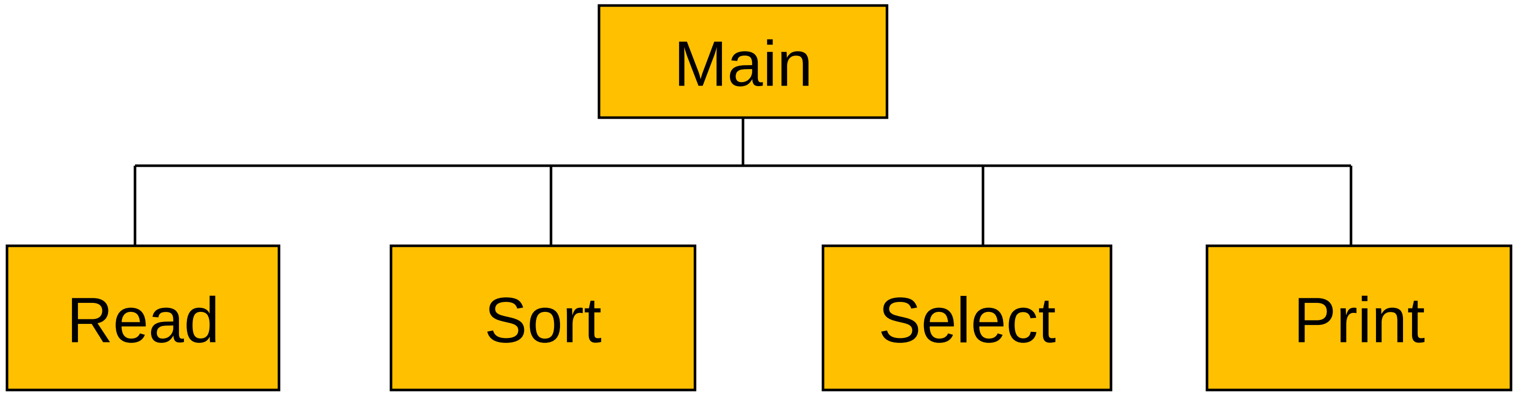
- Breaking the problem into parts helps us to clarify what needs to be done.
- At each step of refinement, the new parts become less complicated and, therefore, easier to figure out.
- Parts of the solution may turn out to be reusable.
- Breaking the problem into parts allows more than one person to work on the solution.

An Example of Top-Down Design

- Problem:
  - We own a home improvement company.
  - We do painting, roofing, and basement waterproofing.
  - A section of town has recently flooded (zip code 21222).
  - We want to send out pamphlets to our customers in that area.

The Top Level

- Get the customer list from a file.
- Sort the list according to zip code.
- Make a new file of only the customers with the zip code 21222 from the sorted customer list.
- Print an envelope for each of these customers.



Another Level?

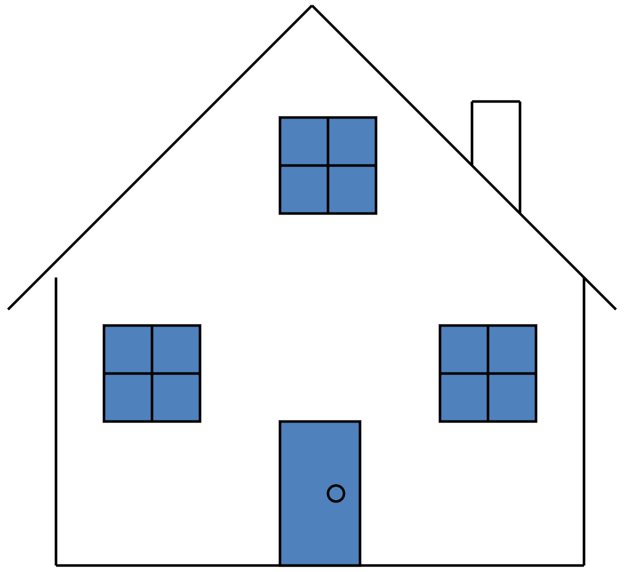
- Should any of these steps be broken down further? Possibly.
- How do I know? Ask yourself whether or not you could easily write the algorithm for the step. If not, break it down again.
- When you are comfortable with the breakdown, write the pseudocode for each of the steps (**modules**) in the **hierarchy**.
- Typically, each module will be coded as a separate function.

Structured Programs

- We will use top-down design for design projects and software programming projects.
- This is the standard way of writing software programs.
- Programs produced using this method and using only the three kinds of control structures, sequential, selection and repetition, are called **structured programs**.
- Structured programs are easier to test, modify, and are also easier for other programmers to understand.

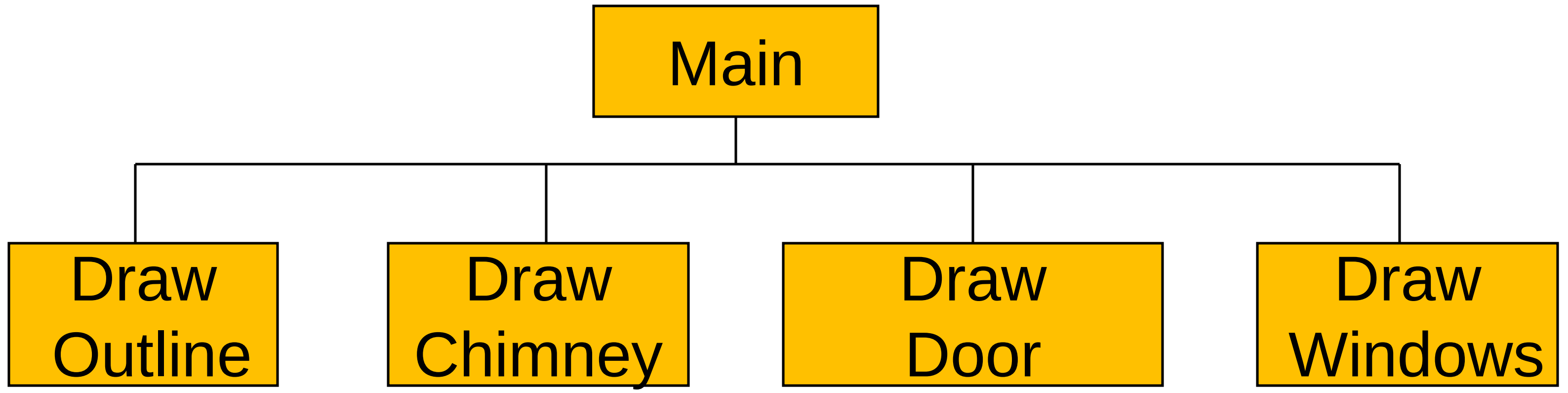
Another Example

- Problem: Write a program that draws this picture of a house.



The Top Level

- Draw the outline of the house
- Draw the chimney
- Draw the door
- Draw the windows

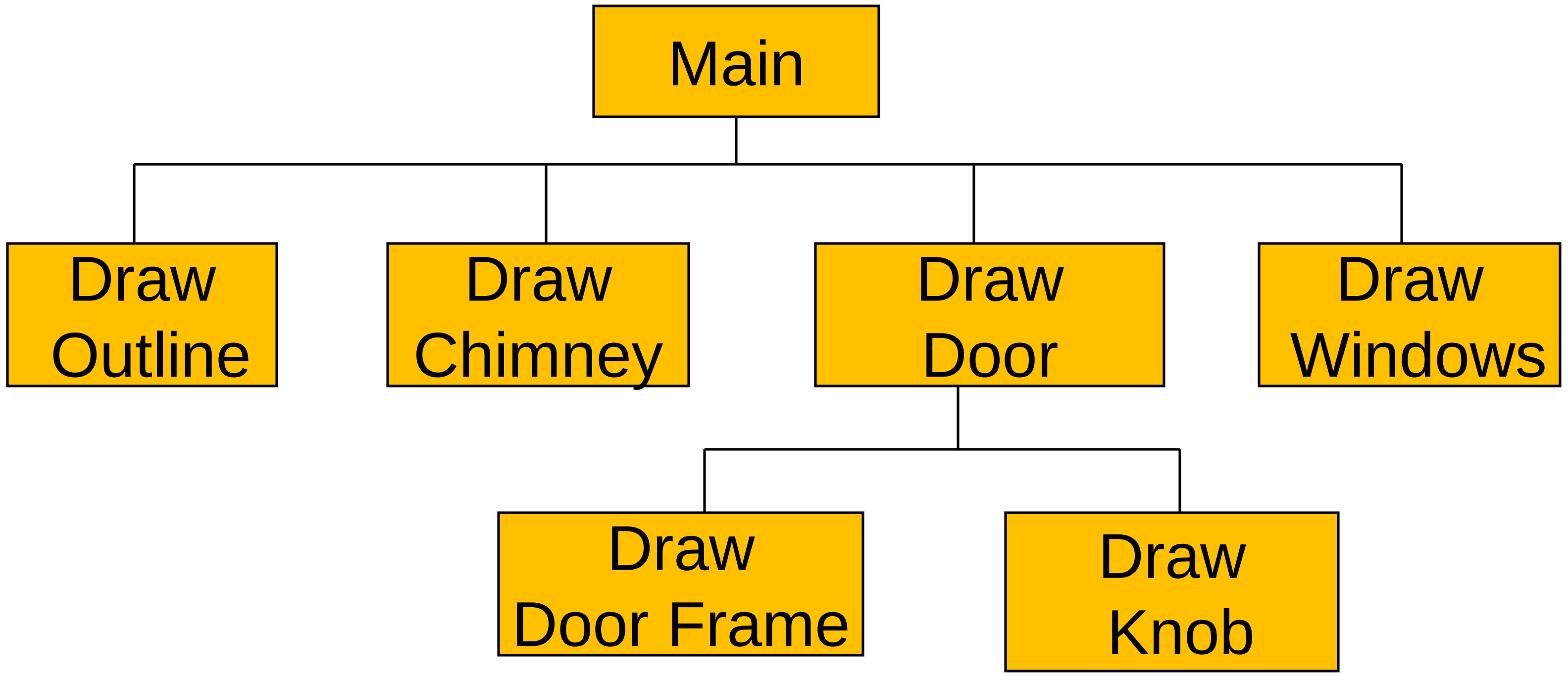


Pseudocode for Main

```
Call Draw Outline
Call Draw Chimney
Call Draw Door
Call Draw Windows
```

Observation

- The door has both a frame and knob. We could break this into two steps.

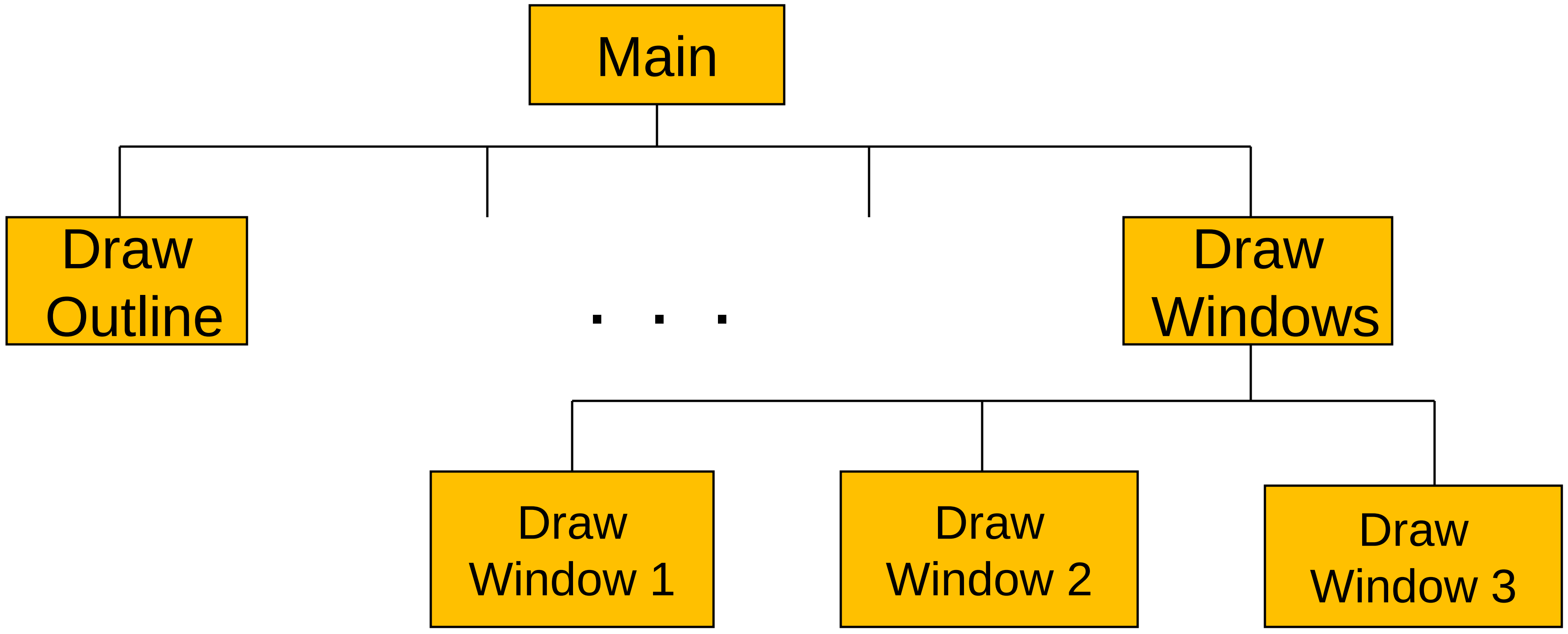


Pseudocode for Draw Door

```
Call Draw Door Frame
Call Draw Knob
```

Another Observation

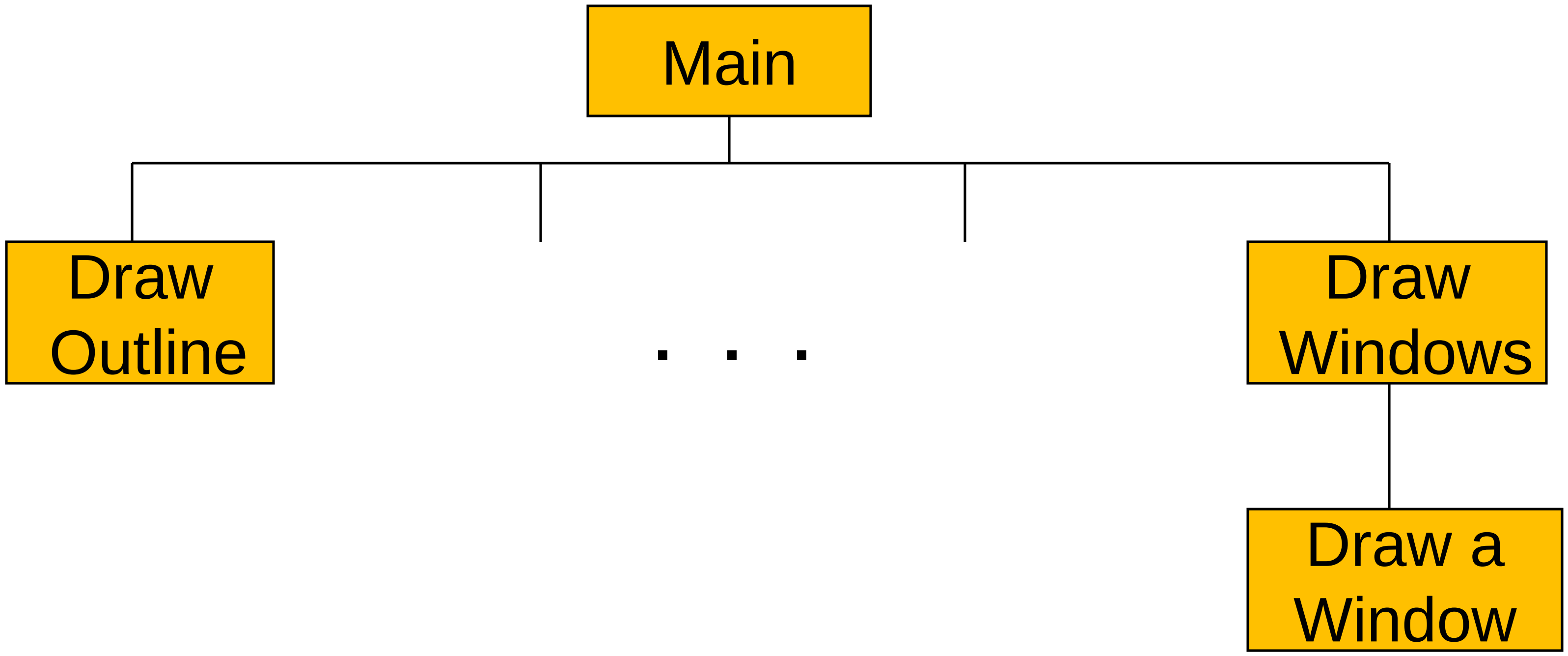
- There are three windows to be drawn.



One Last Observation

- But don’t the windows look the same? They just have different locations.
- So, we can reuse the code that draws a window.
  - Simply copy the code three times and edit it to place the window in the correct location, or
  - Use the code three times, “sending it” the correct location each time (we will see how to do this later).
- This is an example of **code reuse**.

Reusing the Window Code



Pseudocode for Draw Windows

```
Call Draw a Window, sending in Location 1
Call Draw a Window, sending in Location 2
Call Draw a Window, sending in Location 3
```

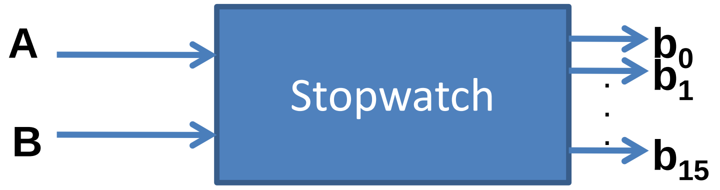
An Electrical Application Example: Digital Design

SIMPLE DIGITAL STOPWATCH

Engineering requirements:

- No more than two control buttons
- Implement Run, Stop and Reset
- Output a 16-bit binary number for seconds

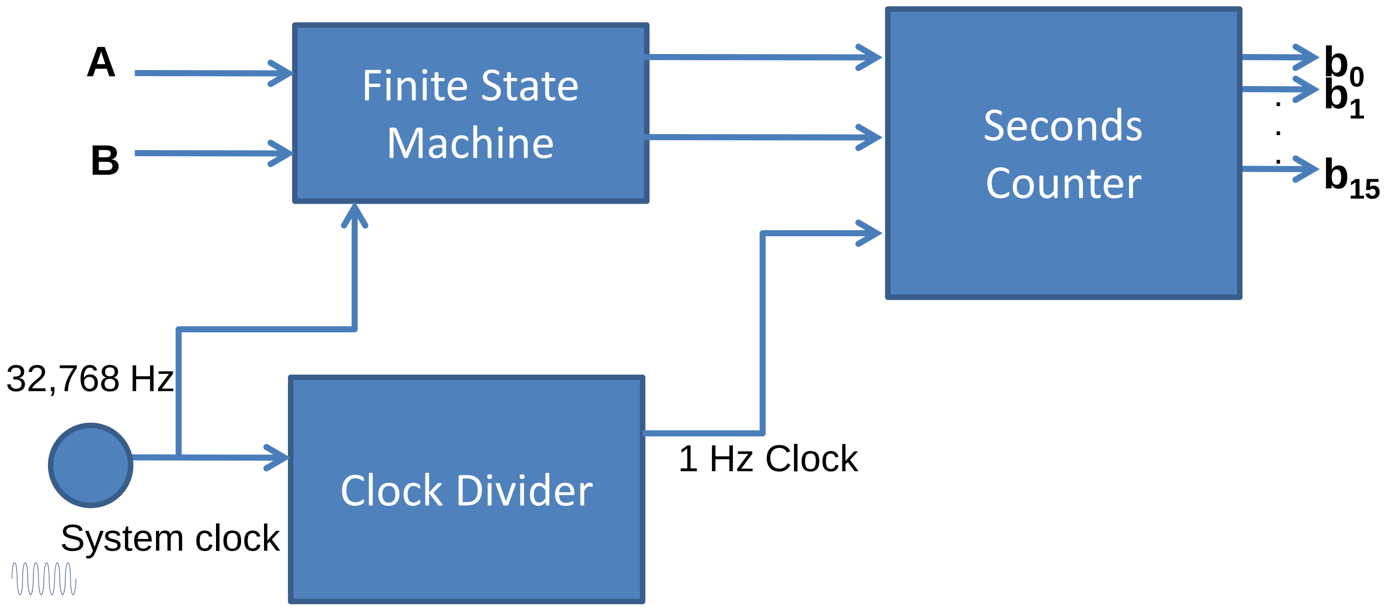
Top-Down Design: Level 0



Level O digital stopwatch functionality

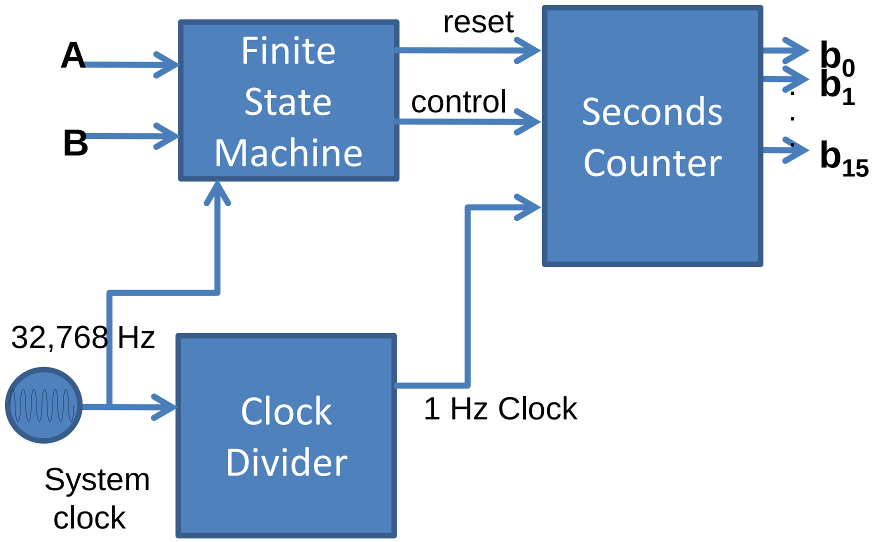
Module	Stopwatch
Inputs	• A: Reset button signal. When the button is pushed, it provides a logic level high signal (5V) that resets the counter to zero. • B: Run/stop toggle signal. When the button is pushed, it provides a logic level high signal (5V) that toggles between run and stop modes.
Outputs	• b_{15}-b_0: 16-bit binary number that represents the number of seconds elapsed.
Functionality	The stopwatch counts the number of seconds after B is pushed when the system is in the Reset or Stop mode. When in Run mode and B is pushed, the stopwatch stops counting. A reset button push (A) will reset the output value of the counter to zero only when in Stop mode.

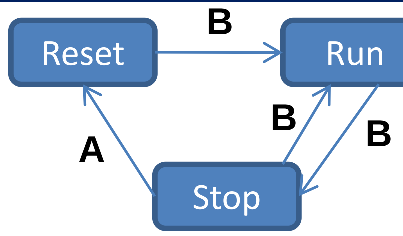
Top-Down Design: Level 1



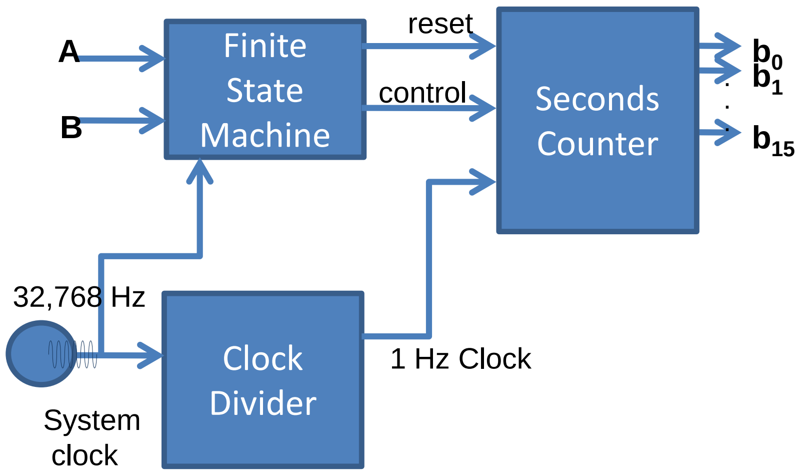
Level 1 design for the digital stopwatch.

Top-Down Design: Level 1



Module	Finite State Machine
Inputs	• A: Signal to reset the counter. • B: Signal to toggle the stopwatch between run and stop modes • Clock: 1 Hz clock signal
Outputs	• Reset: Signal to reset the counter to zero • Control: Signal that enables or disables the counter
Functionality	

Top-Down Design: Level 1



Level 1 design for the digital stopwatch.

Module	Clock Divider
Inputs	• System Clock: 32,768 Hz
Outputs	• Internal clock: 1 Hz clock for seconds elapsed
Functionality	Divide the system clock by 32,768 to produce a 1 Hz clock

Module	Seconds Counter
Inputs	• Reset: Reset the counter to zero • Control: Enable/disable the counter • Clock: Increment the counter
Outputs	• b_{15}-b_0: 16-bit binary representation of number of seconds elapsed.
Functionality	Count the seconds when enabled and resets to zero when reset signal enabled.

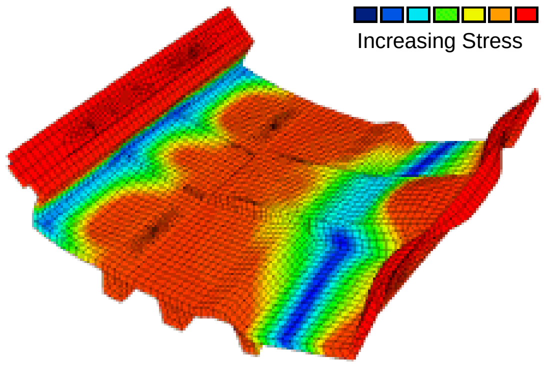
Machine Design

Machine Design…

…is an iterative process that has as its primary objective the synthesis of machines in which the critical problems are based upon material sciences and engineering mechanics sciences.

This synthesis involves the creative conception of mechanisms, and optimization with respect to performance, reliability and cost.

- Machine design does not encompass the entire field of mechanical engineering. Design where the critical problems involve the thermal/fluid sciences fall under the broader category of "mechanical engineering design."
- The primary objective of machine design is synthesis, or creation, not analysis. Analysis is a tool that serves as a means toward an end.

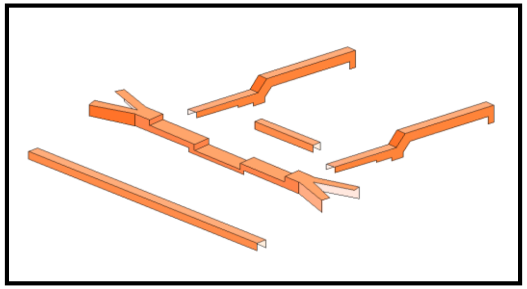


Finite element model of a pickup truck floor pan assembly

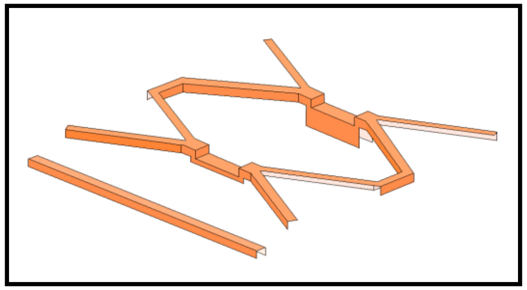
Preliminary Design Phase

Often the first step in which a designer becomes involved, and may not involve intense iteration. In this phase, we deal with the entire machine:

- Define function
- Identify constraints involving cost, size, etc.
- Develop alternative conceptions of mechanism/process combinations that can satisfy the constraints
- Perform supporting analyses (thermodynamic, heat transfer, fluid mechanics, kinematics, force, stress, life, cost, compatibility with special constraints)
- Select the best mechanism
- Document the design



Concept 1 Two longitudinal members, one transverse split-end cross member, small transverse member in transmission tunnel, rear transverse member similar to original, gauge reduction.



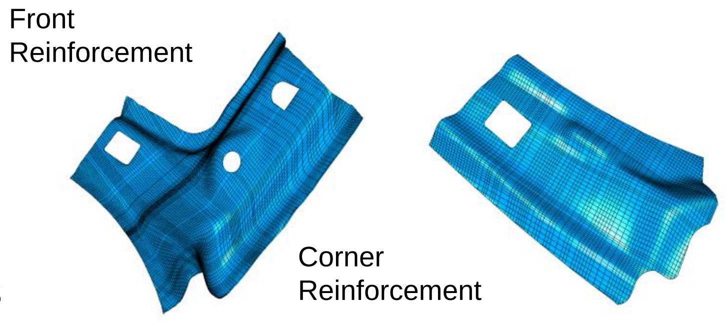
Concept 6 Two integrated, split transverse cross members, rear transverse member similar to original, reduced sheet thickness in cross members.

Alternative design concepts for cross members in a light-duty truck floor pan assembly

Intermediate Design Phase

Generally occurs after preliminary design, but the two phases may overlap. Intermediate design always involves iterations. In this phase, we deal with individual components of the machine:

- Identify components
- Define component functions
- Identify constraints involving cost, size, etc.
- Develop tentative conceptions of the components mechanism/process combinations using good *form synthesis* principles
- Perform supporting analyses (including analyses at each *critical point* in each component)
- Select the best component designs
- Document component designs; prepare a layout drawing

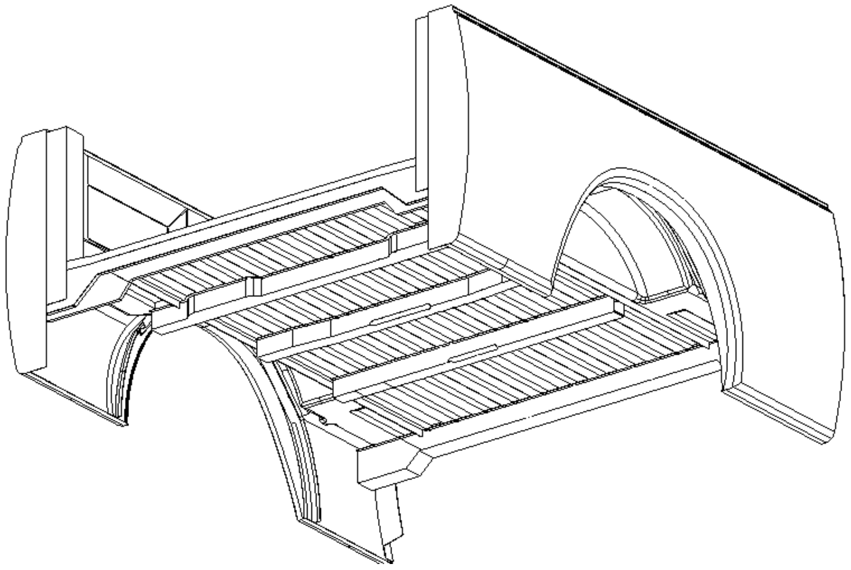


A-pillar component geometries

Detail Design Phase

Subsequent to intermediate and in this phase, we deal with individual components of the machine and the machine as a whole:

- Select manufacturing and assembly processes
- Specify dimensions and tolerances
- Prepare component detail drawings
- Prepare assembly drawings



Line rendering of a pickup box assembly showing geometric details such as wheel well openings, cross members, and bed corrugation

Engineering is Interdisciplinary!

Engineering is a great way to reinforce content taught in subject areas including:

- Science
- Math
- Technology
- Language Arts
- Social Studies/History
- Engineering
- Mechatronics