# Review Session

Jamal Madni

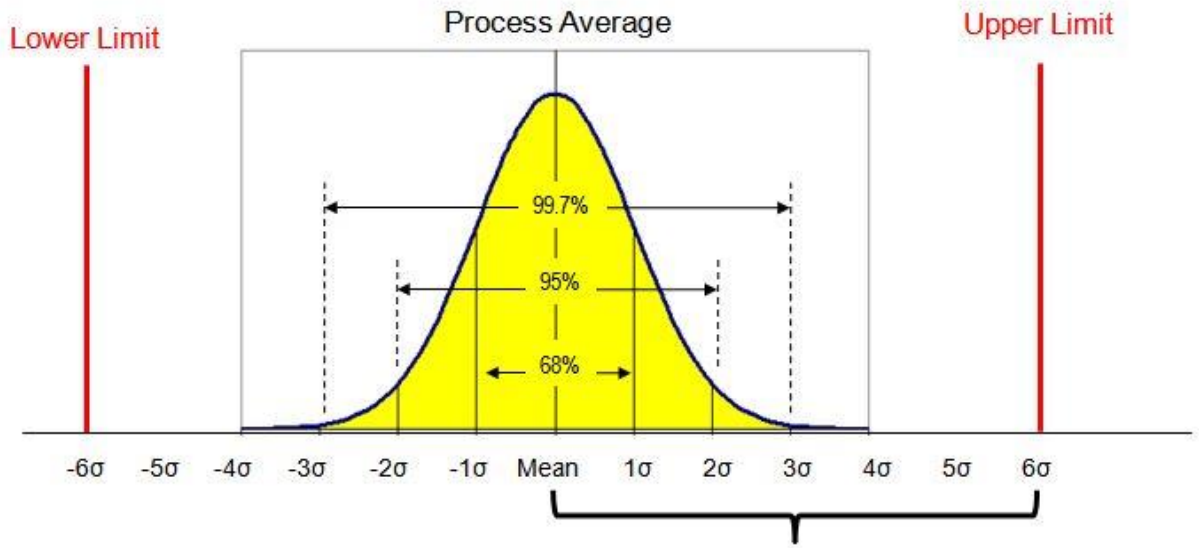
CECS 445

Lecture 15: April 6th, 2021

# Six Sigma


- *Define* customer requirements and deliverables and project goals via well-defined methods of customer communication.
- *Measure* the existing process and its output to determine current quality performance (collect defect metrics).
- *Analyze* defect metrics and determine the vital few causes.

- *Improve* the process by eliminating the root causes of defects.
- *Control* the process to ensure that future work does not reintroduce the causes of defects.



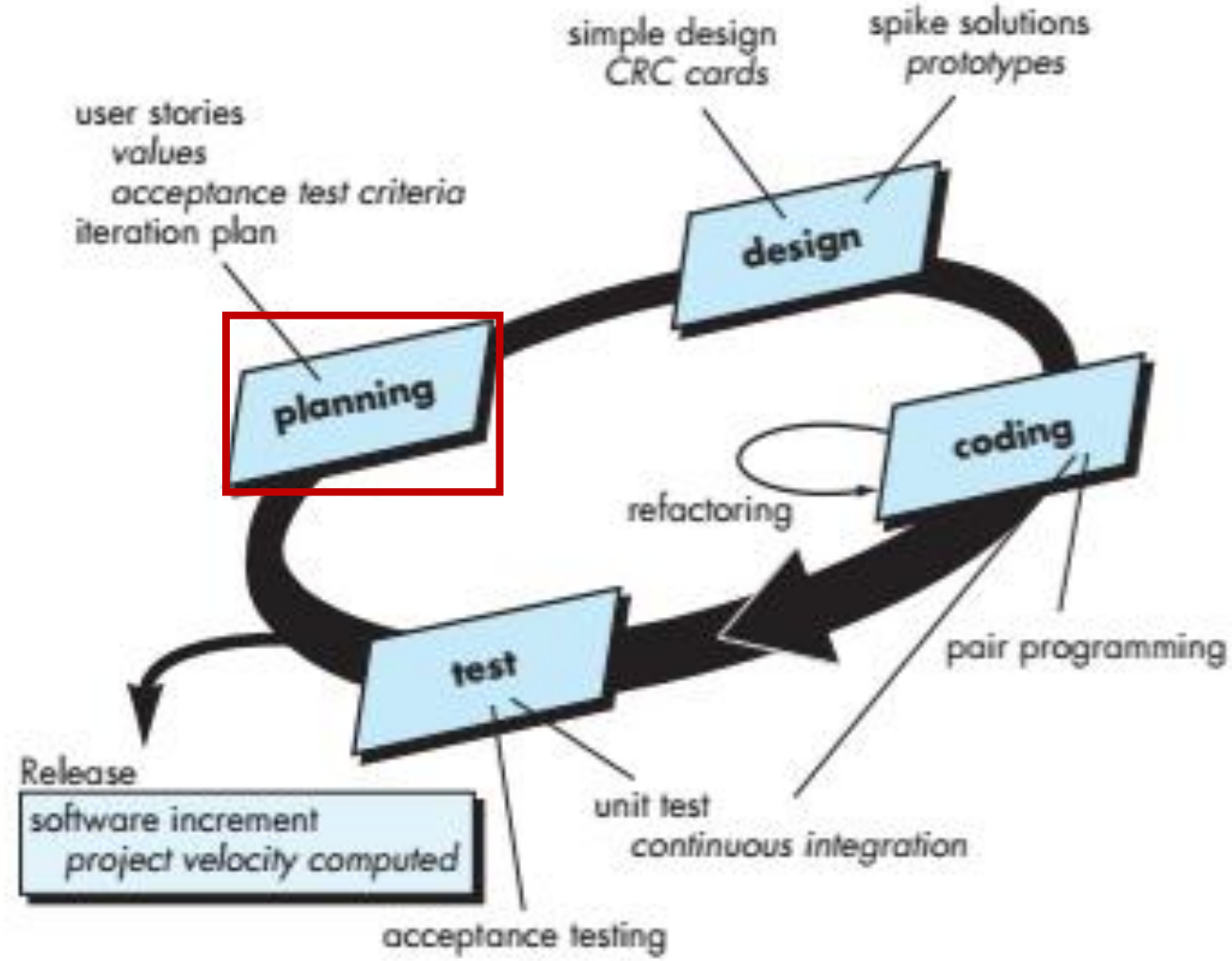

**Define** = User Stories & Requirements Tables, Architecture Diagrams, Schedules

**Measure** = CPI, Risk, Defect Amplification Model, Error Density, Availability, Software Maturity Index

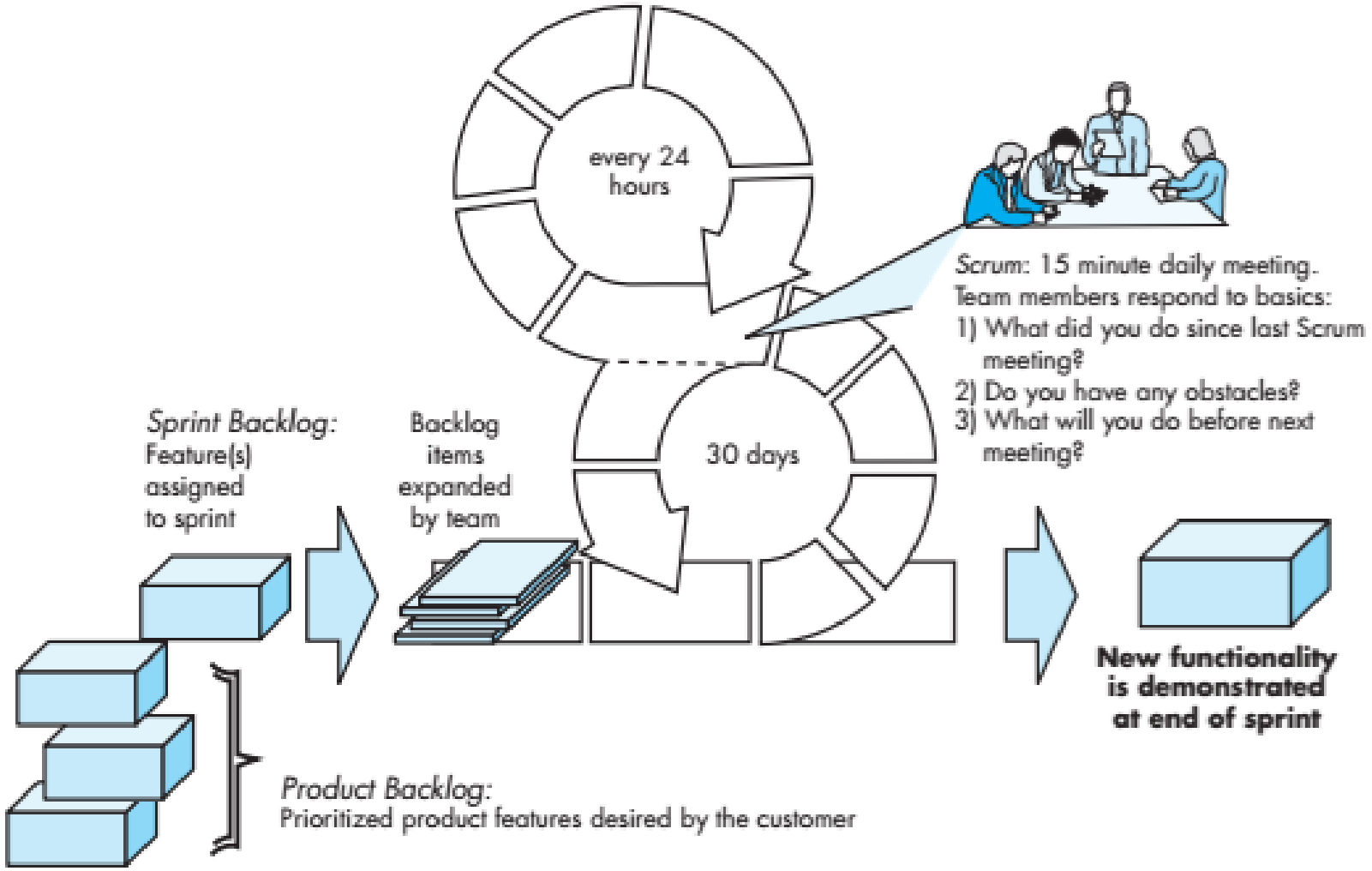
**Analyze** = What do these metrics mean? The "Why" in Scrum & Status Tag-Ups

**Improve & Control** = Kaizen

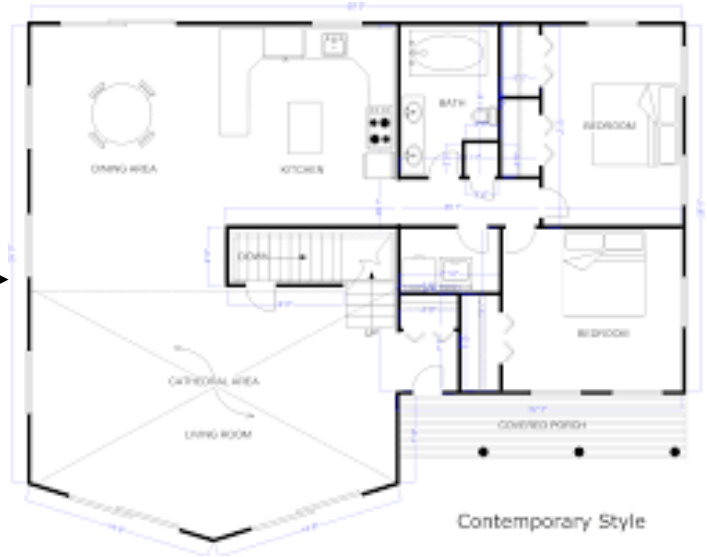
# Extreme Programming (XP)



# Scrum Process Flow



# Scrum Master's Ultimate Job: Translate!




**Stories**

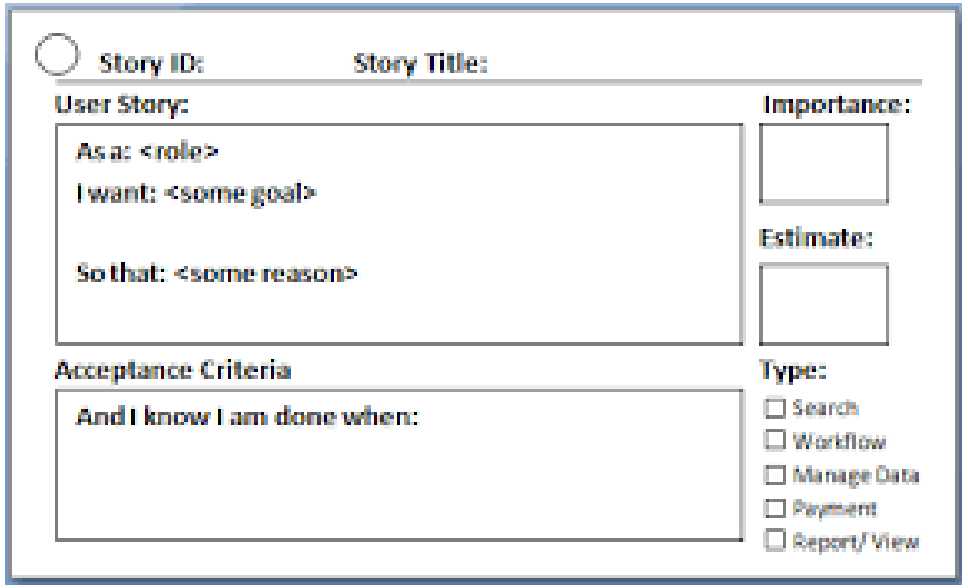
*"The What"*

**Requirements**

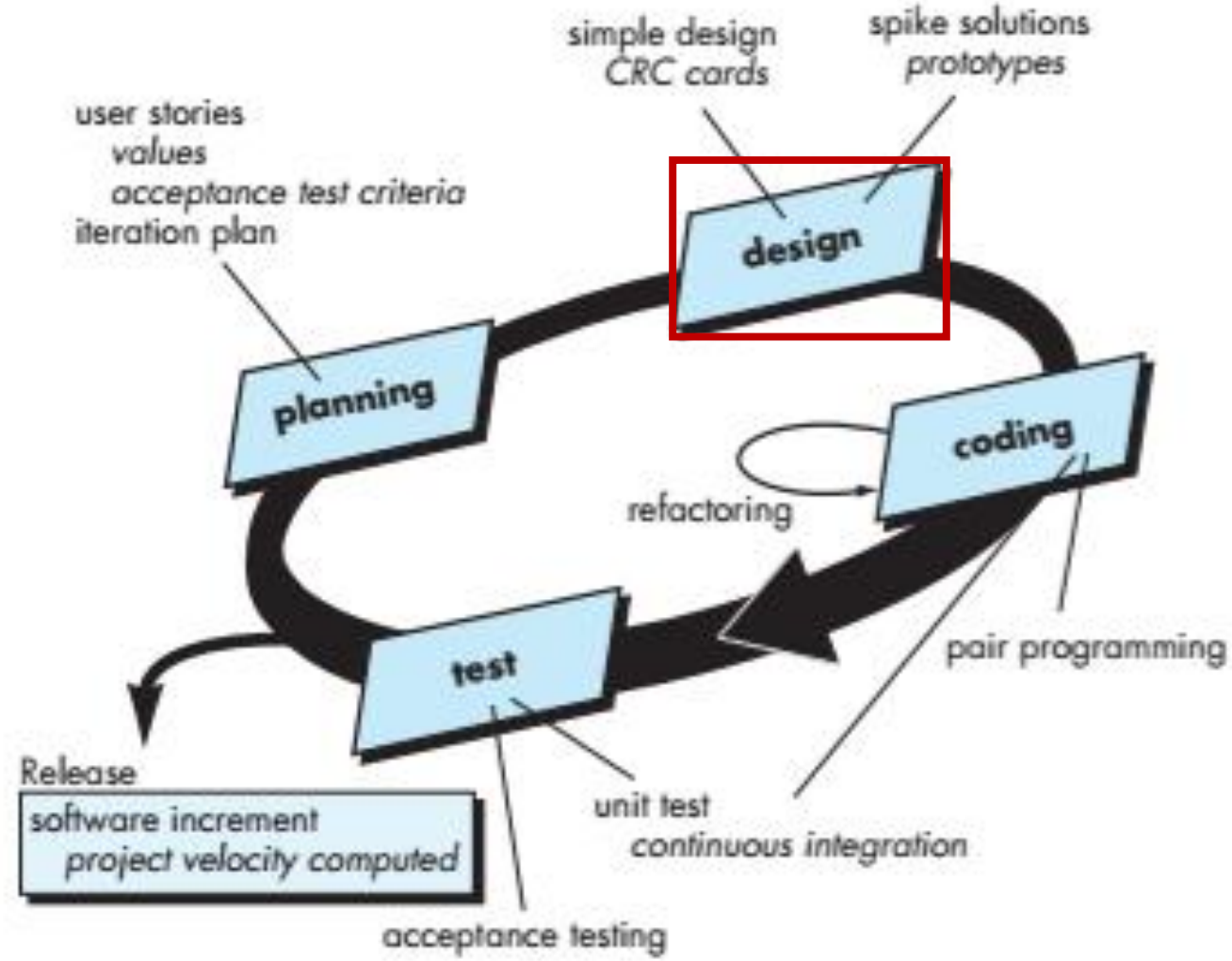
*"The How"*

# User Story Templates

| Title: | Priority: | Estimate: |
| --- | --- | --- |
| **User Story:**<br><br>As a [description of user],<br>I want [functionality]<br>so that [benefit]. | | |
| **Acceptance Criteria:**<br><br>Given [how things begin]<br>When [action taken]<br>Then [outcome of taking action] | | |

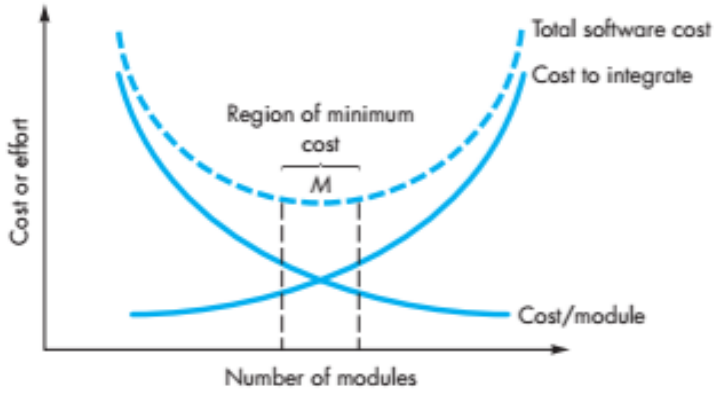


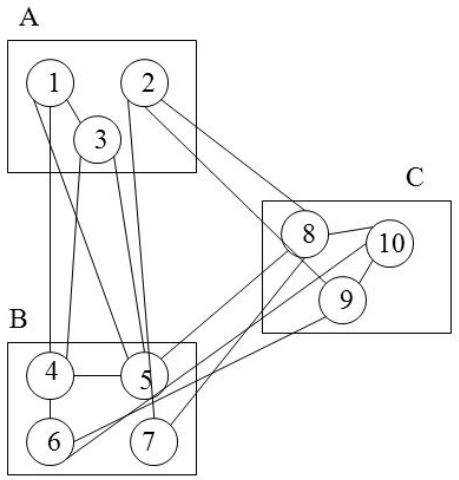
# Extreme Programming (XP)



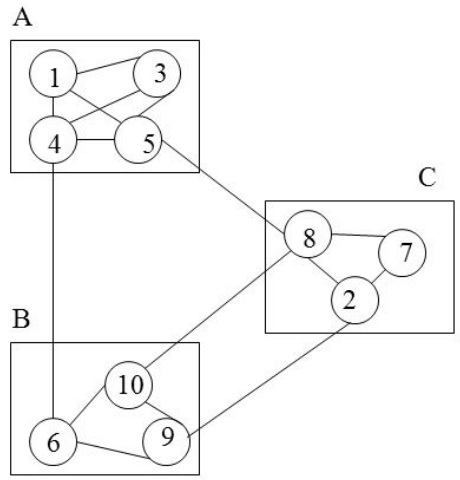
# Design Principles

- Abstraction
- Modularity
- Functional Independence
  *(Cohesion vs. Coupling)*
- Patterns
- Information Hiding

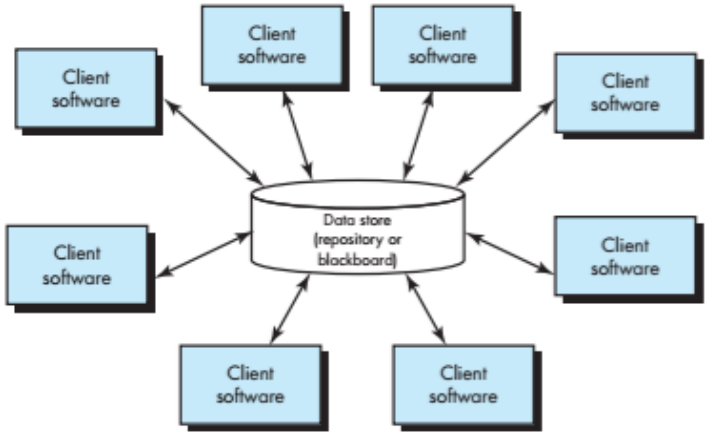




Bad modularization:
low cohesion, high coupling

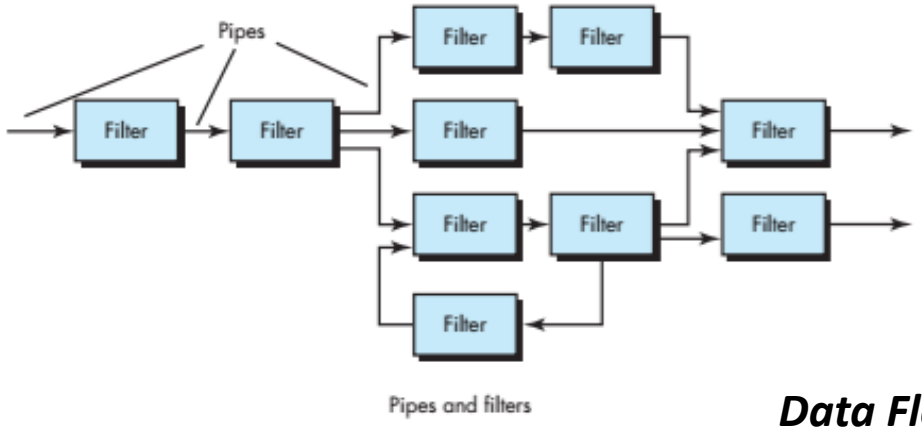



Good modularization:
high cohesion, low coupling

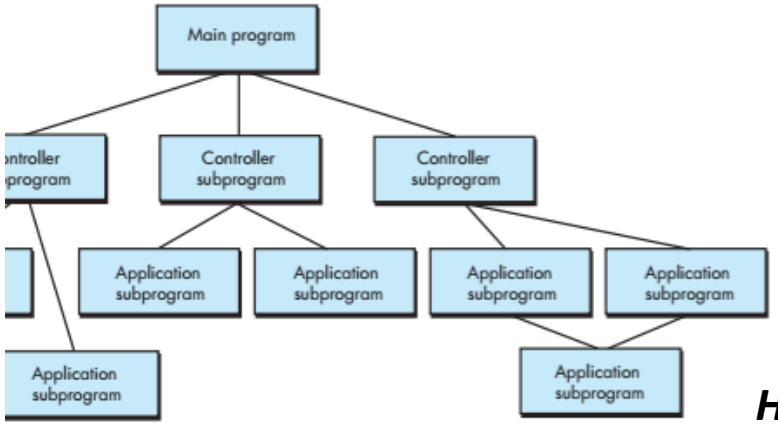
# Architecture Types



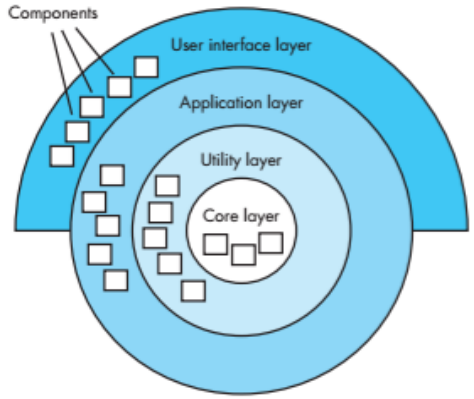

***Data-Centric***




***Data Flow***

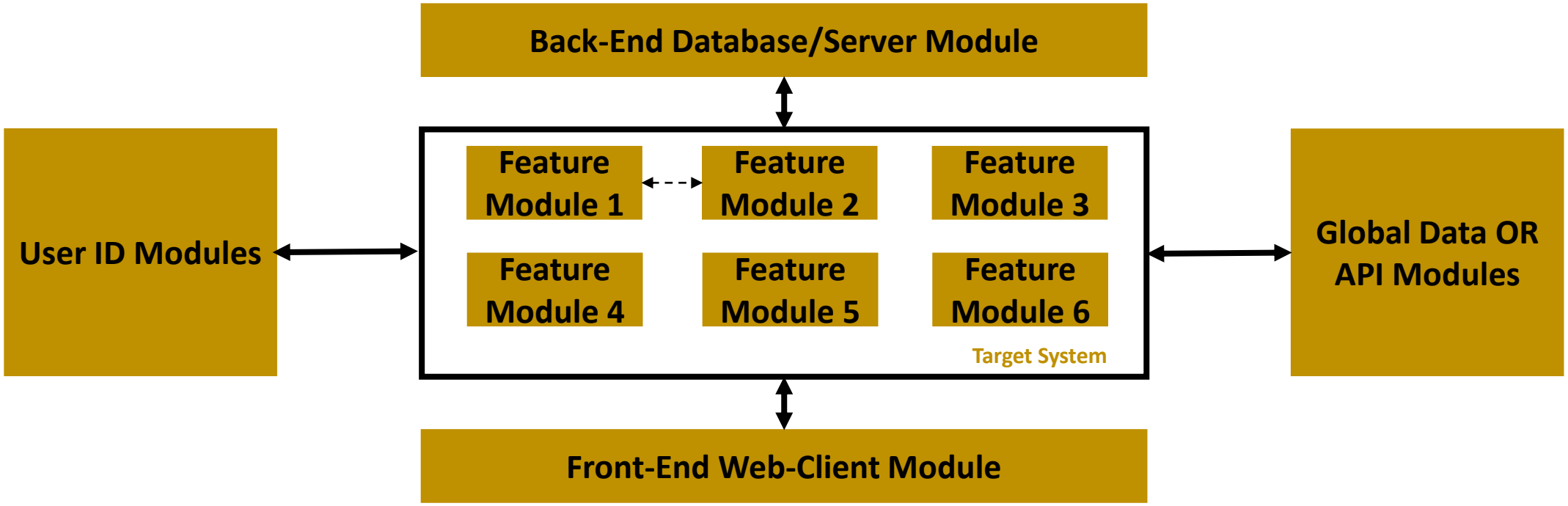



***Hierarchical***




***Layered***

# Architecture Diagram



# How Do I Measure Complexity of My Architecture?

| Information Domain Value | Count | | Weighting factor: Simple | Average | Complex | |
|---|---|---|---|---|---|---|
| External Inputs (EIs) | | 3 | 3 | 4 | 6 | = |
| External Outputs (EOs) | | 3 | 4 | 5 | 7 | = |
| External Inquiries (EQs) | | 3 | 3 | 4 | 6 | = |
| Internal Logical Files (ILFs) | | 3 | 7 | 10 | 15 | = |
| External Interface Files (EIFs) | | 3 | 5 | 7 | 10 | = |
| Count total → | | | | | | |

$$FP = \text{count total} \times [0.65 + 0.01 \times \Sigma(F_i)]$$

1. Does the system require reliable backup and recovery?
2. Are specialized data communications required to transfer information to or from the application?
3. Are there distributed processing functions?
4. Is performance critical?
5. Will the system run in an existing, heavily utilized operational environment?
6. Does the system require online data entry?
7. Does the online data entry require the input transaction to be built over multiple screens or operations?
8. Are the ILFs updated online?
9. Are the inputs, outputs, files, or inquiries complex?
10. Is the internal processing complex?
11. Is the code designed to be reusable?
12. Are conversion and installation included in the design?
13. Is the system designed for multiple installations in different organizations?
14. Is the application designed to facilitate change and ease of use by the user?

| | |
|---|---|
| **External Inputs** | *Parameters* |
| **External Outputs** | *Return Types, Error Messages, UI* |
| **External Inquiries** | *Function Calls By Others* |
| **Internal Logical Files** | *Local Data Dependent On Global Variables / Files* |
| **External Interface Files** | *Global States Used By Function* |

# How Do I Measure Effort? Step 1 of 2.

- LOC Cost Estimate = (LOC / pm) * Cost Per Programmer Per Month
- **LOC Effort Estimate** = (LOC / group size) * Hours Per Student Per Month

- FP Cost Estimate = (FP / pm) * Cost Per Programmer Per Month
- **FP Effort Estimate** = (FP / group size) * Hours Per Student Per Month

| Software Requirement | Acceptance Criteria | Module | FP Requirement Score | LOC Module Score | Effort | Priority | Lead | Time |
|---|---|---|---|---|---|---|---|---|
| "X shall perform Y" | "How do I know it's completed successfully?" | Specific Target System Module, Database Module, Front End Web-Client Module | | | | H/M/L | | Number between 0 -1 |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |

# Effort Equation For Each Requirement

**L** : LOC or FP

$$E = \frac{L^3}{P^3 t^4}$$

**P** (for LOC) : 1200 = Expert in Tool/Language of Requirement, 700 = Proficient, 200 = Beginner *(i.e., Learning Curve)*

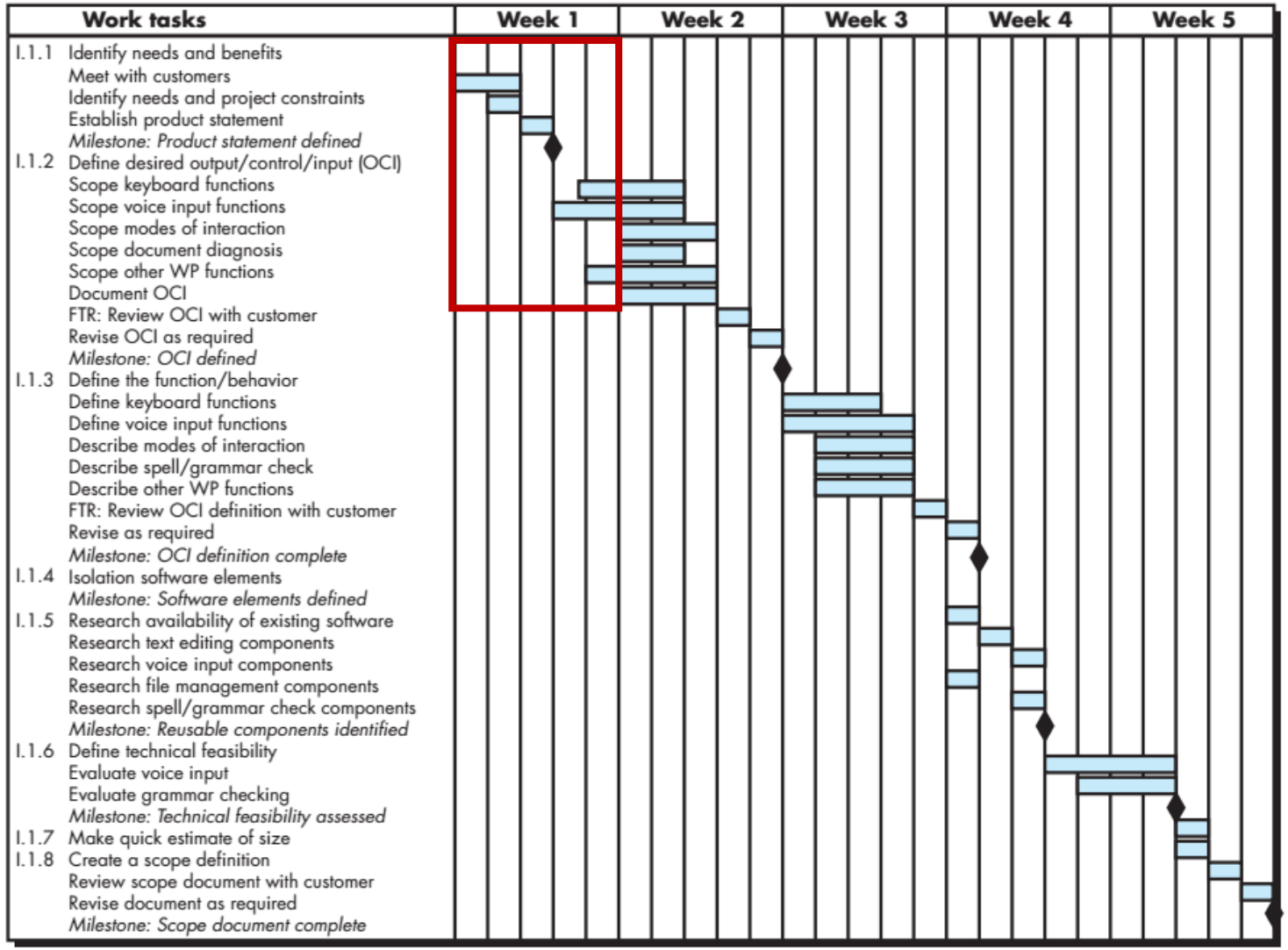
**P** (for FP) : 30 = Expert in Tool/Language of Requirement, 20 = Proficient, 10 = Beginner *(i.e., Learning Curve)*

**t :** Lead Programmer Hours = (3 * Lead Programmer Hours Per Month Dedicated to CECS 445) / 100

| Information domain value | Opt. | Likely | Pess. | Est. count | Weight | FP count |
|---|---|---|---|---|---|---|
| Number of external inputs | 20 | 24 | 30 | 24 | 4 | 97 |
| Number of external outputs | 12 | 15 | 22 | 16 | 5 | 78 |
| Number of external inquiries | 16 | 22 | 28 | 22 | 5 | 88 |
| Number of internal logical files | 4 | 4 | 5 | 4 | 10 | 42 |
| Number of external interface files | 2 | 2 | 3 | 2 | 7 | 15 |
| *Count total* | | | | | | *320* |

| Function | Estimated LOC |
|---|---|
| User interface and control facilities (UICF) | 2,300 |
| Two-dimensional geometric analysis (2DGA) | 5,300 |
| Three-dimensional geometric analysis (3DGA) | 6,800 |
| Database management (DBM) | 3,350 |
| Computer graphics display facilities (CGDF) | 4,950 |
| Peripheral control function (PCF) | 2,100 |
| Design analysis modules (DAM) | 8,400 |
| *Estimated lines of code* | *33,200* |

# Mapping Requirements to Schedule



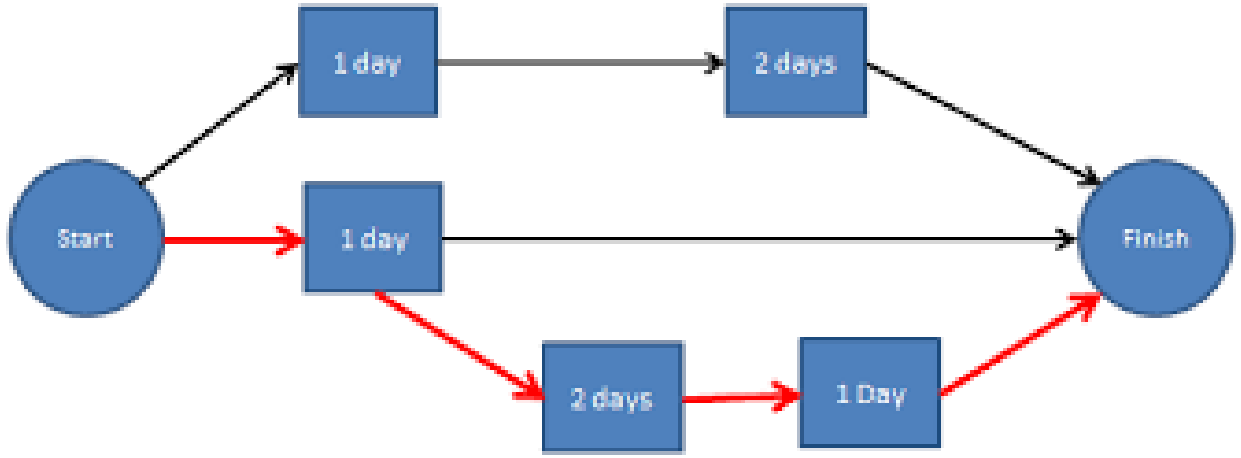

- Tasks at Requirement Level or Module Level
- Map each programmer to his/her tasks (i.e., first column should be names)
- Weeks should be allocated by Scrum Cycles (i.e., Cycle 1, 2, 3, etc.)
- Dependencies (i.e., 40 – 20 – 40 Rule)
- Integration Buffer

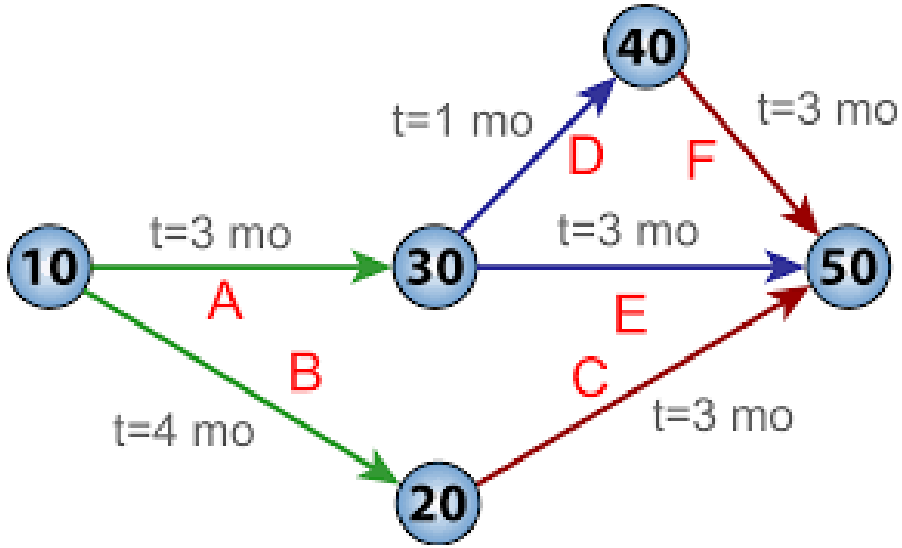
# What Requirements Should I Start With?

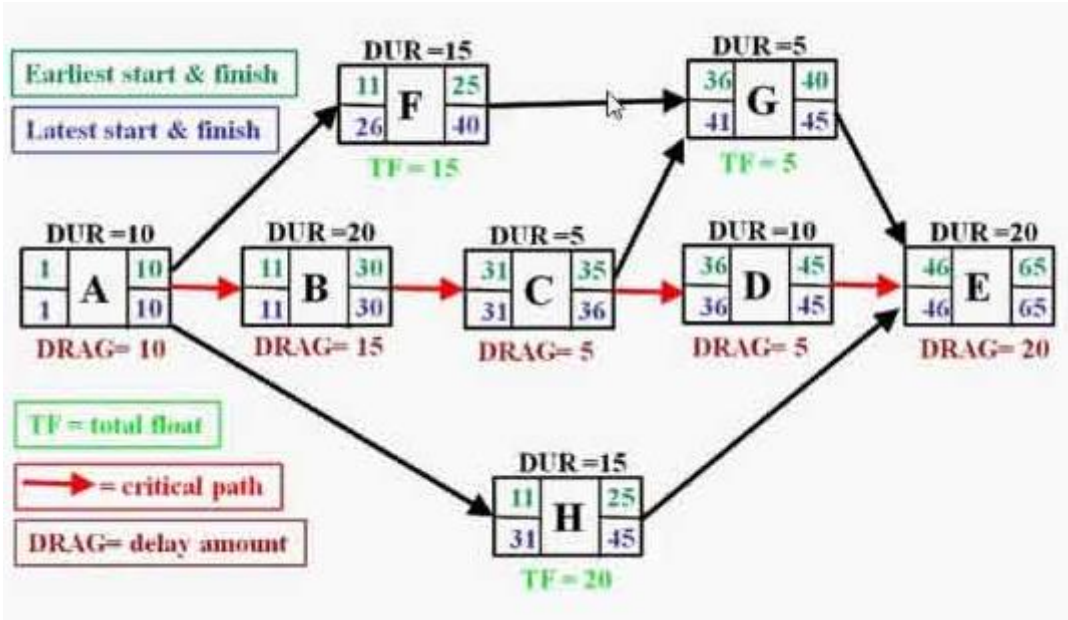
- Effort & Priority **HIGH**
- **Effort =** $\frac{L^3}{P^3 t^4}$
- **Priority = ?**

| | |
|---|---|
| **Effort LOW**<br>**Dependencies HIGH**<br><br>*Easy First Modules*<br>*"Quick Wins"* | **Effort HIGH**<br>**Dependencies HIGH**<br><br>*Critical Path Modules*<br>*"Must Begin Now"* |
| **Effort LOW**<br>**Dependencies LOW**<br><br>*Easy Last Modules*<br>*"Do I Even Need This?"* | **Effort HIGH**<br>**Dependencies LOW**<br><br>*Parallel Path Modules*<br>*"Must Begin But Isolated"* |

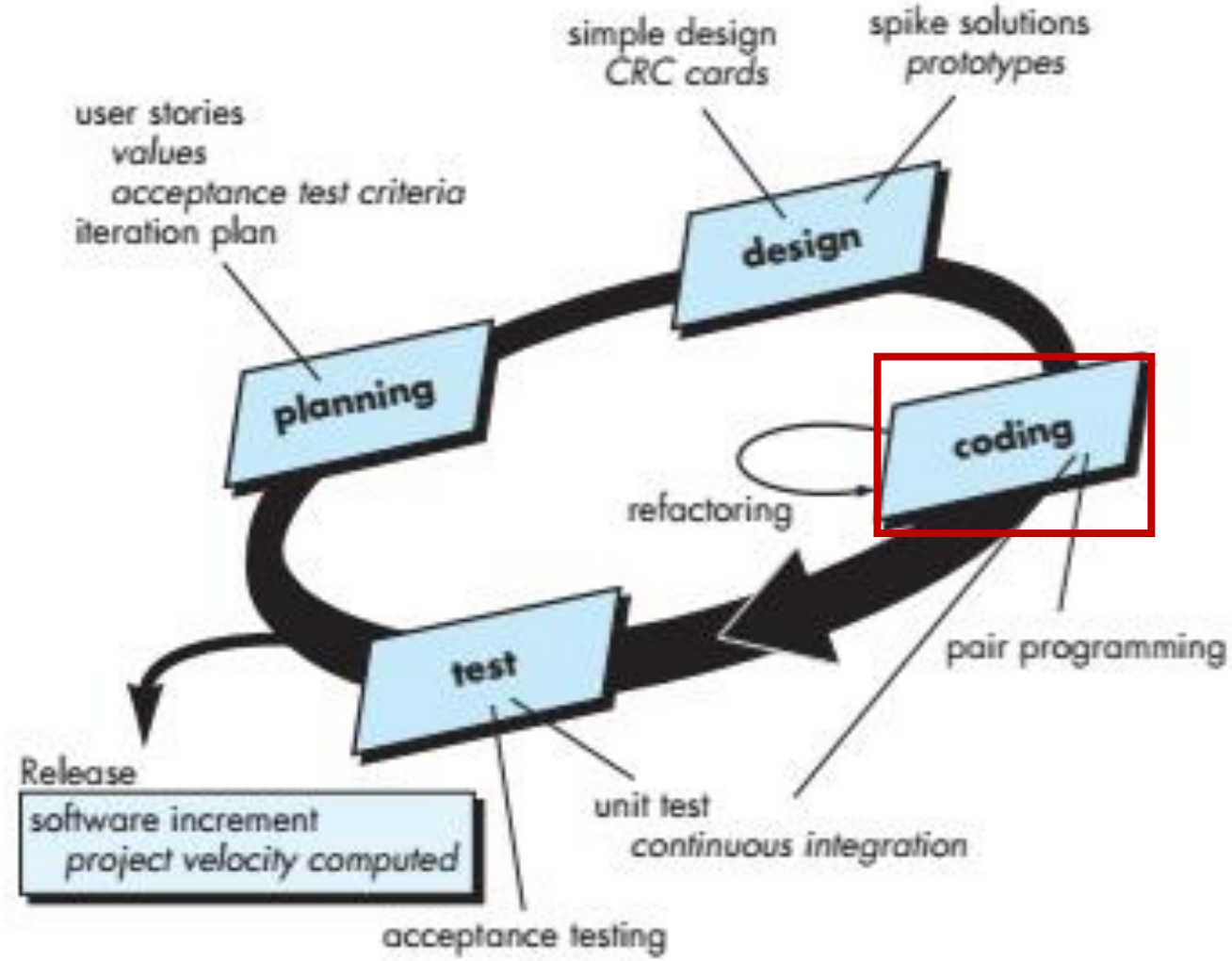
# Critical Path Method (CPM)



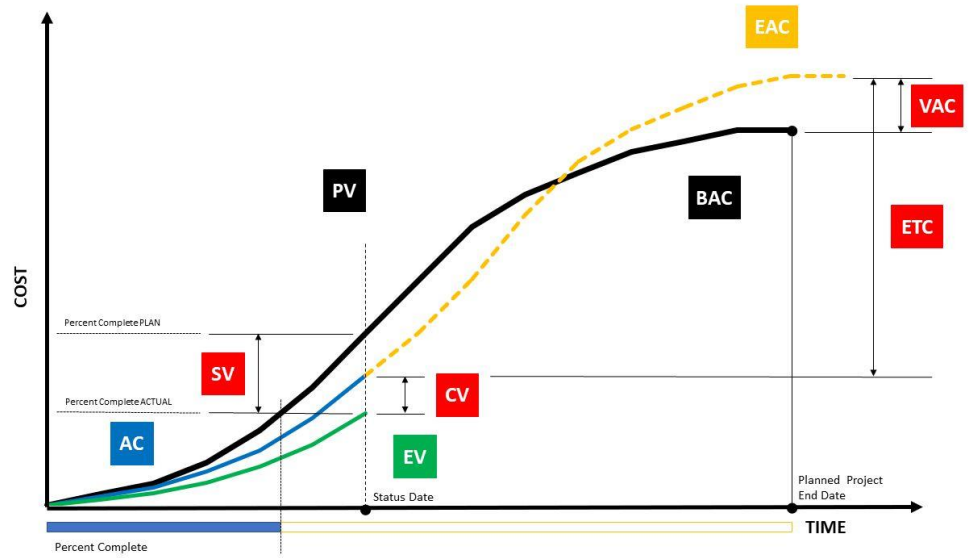



# Extreme Programming (XP)



# Earned Value Management

- **BAC** = Budget at Completion
- **BCWP** = Budget Cost of Work Performed
- **BCWS** = Budget Cost of Work Scheduled
- **ACWP** = Actual Cost of Work Performed




$$BAC = \Sigma\,(BCWS_k) \text{ for all tasks } k$$

$$\text{Schedule performance index, } SPI = \frac{BCWP}{BCWS}$$

$$\text{Schedule variance, } SV = BCWP - BCWS$$

$$\text{Percent scheduled for completion} = \frac{BCWS}{BAC}$$

$$\text{Percent complete} = \frac{BCWP}{BAC}$$

$$\text{Cost performance index, } CPI = \frac{BCWP}{ACWP}$$

$$\text{Cost variance, } CV = BCWP - ACWP$$

# Earned Value Reviewed

- BCWP = Requirement Length * Budgeted Proportion of Time for Lead on Requirement * (Lead Hours Per Month)

- ACWP = Actual Hours Spent on Requirement

- Calculate BCWP & ACWP of **each requirement** attempted/completed to present

- Ex: Budgeted R1 for 0.2 time for 2 weeks at 40 Hours Per Month; Really Spent 22 Hours
- Ex: Budgeted R2 for 0.25 time for 1 week at 20 Hours Per Month; Really Spent 5 Hours

- R1_BCWP = 0.2 * (2/4) * 40 = 4 R1_ACWP = 22
- R2_BCWP = 0.25 * (1/4) * 20 = 1.25 R2_ACWP = 5

- CPI = (R1_BCWP + R2_BCWP + …. + RN_BCWP) / (R1_ACWP + R2_ACWP + … + RN_ACWP)

# Earned Value Reviewed

- CPI = (R1_BCWP / R1_ACWP) + (R2_BCWP / R2_ACWP) + ….
- Why not this?
- What does this equation mean?
- How should I interpret this?

# Earned Value Reviewed

- BCWP = Requirement Length * Budgeted Proportion of Time for Lead on Requirement * (Lead Hours Per Month)

- If two developers are working on a requirement together?
  - Weighted average of proportion of time
  - Sum lead hours per month

  - Ex: Developer 1 is working on requirement 1 at 0.2 time at 40 Hours Per Month
  - Ex: Developer 2 is working on requirement 1 at 0.1 time at 30 Hours Per Month

  - Budgeted Proportion of Time for Lead(s) = (0.2*(40/70)) + (0.1*(30/70)) = 0.16
  - Lead Hours Per Month = 70

# Metric For Maintenance

$M_T$ = number of modules in the current release
$F_c$ = number of modules in the current release that have been changed
$F_a$ = number of modules in the current release that have been added
$F_d$ = number of modules from the preceding release that were deleted in the current release

$$\text{SMI} = \frac{[M_T - (F_a + F_c + F_d)]}{M_T}$$

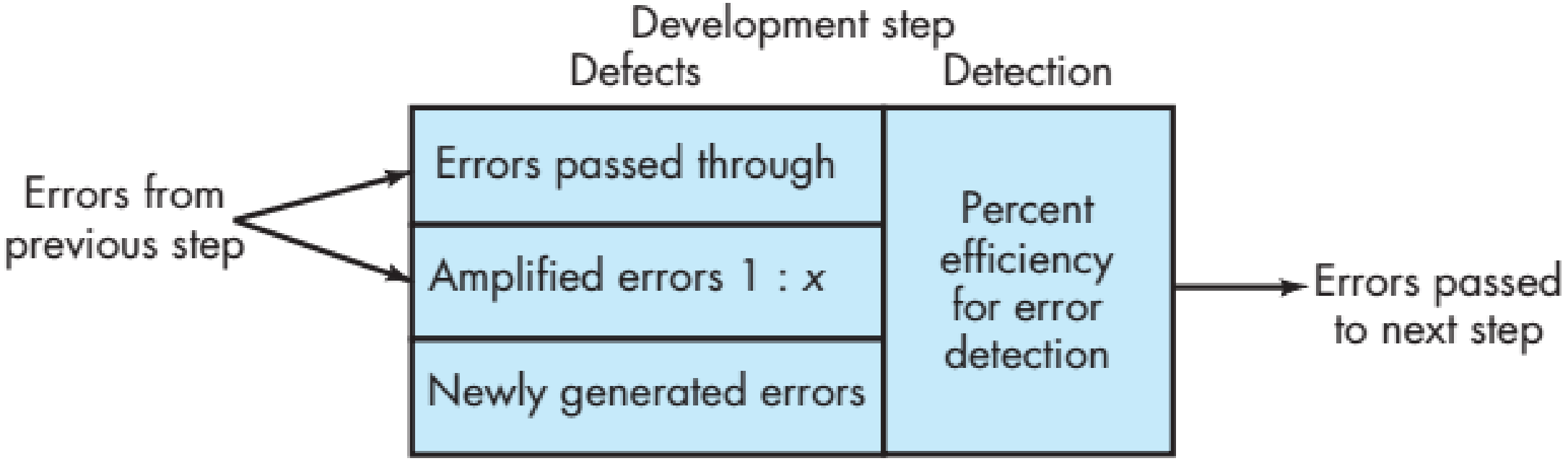
SMI = Software Maturity Index

# Risk Matrix Reviewed

| Category \ Components | | Performance | Support | Cost | Schedule |
|---|---|---|---|---|---|
| Catastrophic | 1 | Failure to meet the requirement would result in mission failure | | Failure results in increased costs and schedule delays with expected values in excess of $500K | |
| | 2 | Significant degradation to nonachievement of technical performance | Nonresponsive or unsupportable software | Significant financial shortages, budget overrun likely | Unachievable IOC |
| Critical | 1 | Failure to meet the requirement would degrade system performance to a point where mission success is questionable | | Failure results in operational delays and/or increased costs with expected value of $100K to $500K | |
| | 2 | Some reduction in technical performance | Minor delays in software modifications | Some shortage of financial resources, possible overruns | Possible slippage in IOC |
| Marginal | 1 | Failure to meet the requirement would result in degradation of secondary mission | | Costs, impacts, and/or recoverable schedule slips with expected value of $1K to $100K | |
| | 2 | Minimal to small reduction in technical performance | Responsive software support | Sufficient financial resources | Realistic, achievable schedule |
| Negligible | 1 | Failure to meet the requirement would create inconvenience or nonoperational impact | | Error results in minor cost and/or schedule impact with expected value of less than $1K | |
| | 2 | No reduction in technical performance | Easily supportable software | Possible budget underrun | Early achievable IOC |

- Each entry is requirement or phenomena (collection of requirements)
- Ignore “Negligible”
- Not meant to be have entries everywhere
- Begin each week with analysis of risk & CPI
- Actual monitoring tool (not projection, busy work, hypothesis)

# Measure Defects – *Defect Amplification Model*




$$\text{Error density} = \frac{\text{Err}_{\text{tot}}}{\text{WPS}}$$ , WPS = # of requirements for that cycle

# Measure Defects – *Error Density, Availability & MTBF*

$$\text{Error density} = \frac{\text{Err}_{\text{tot}}}{\text{WPS}}$$ , WPS = # of requirements for that cycle

$$\text{MTBF} = \text{MTTF} + \text{MTTR}$$

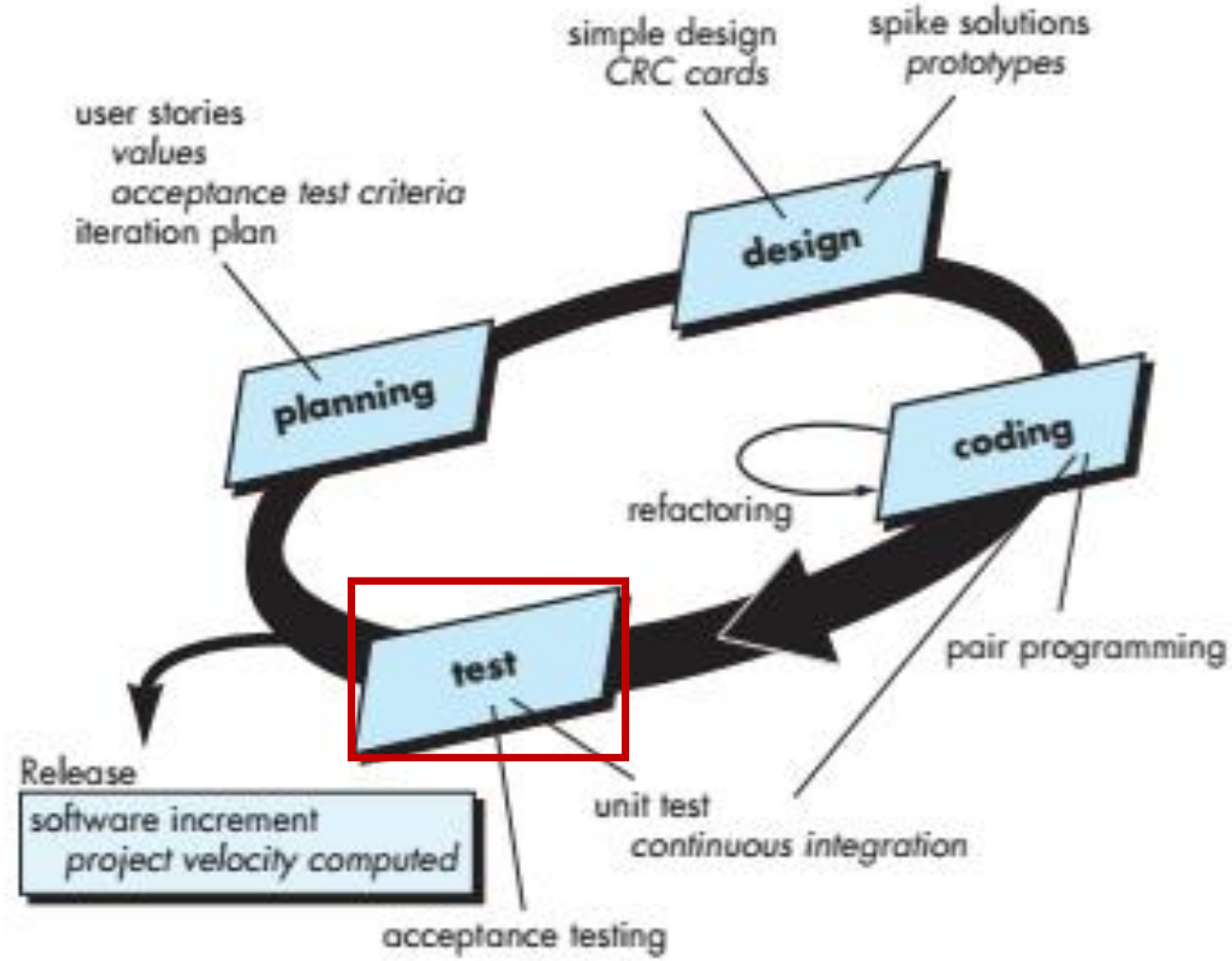
Mean-Time-Between-Failure (MTBF)

Mean-Time-To-Failure (MTTF)

Mean-Time-To-Repair (MTTR)

$$\text{Availability} = \frac{\text{MTTF}}{(\text{MTTF} + \text{MTTR})} \times 100\%$$

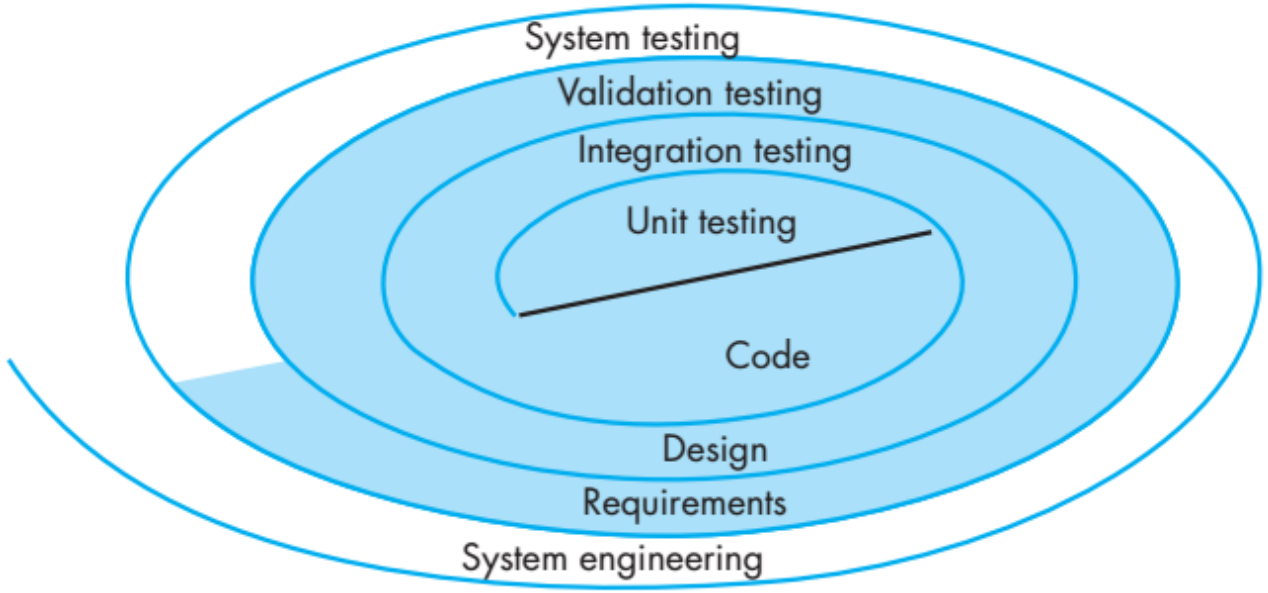
# Extreme Programming (XP)



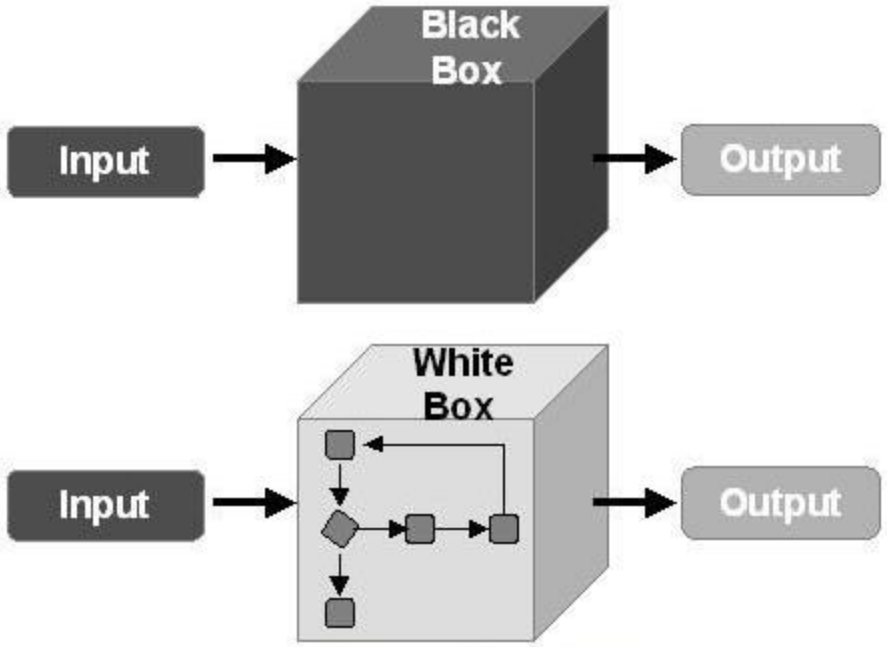
# Intro to Software Testing

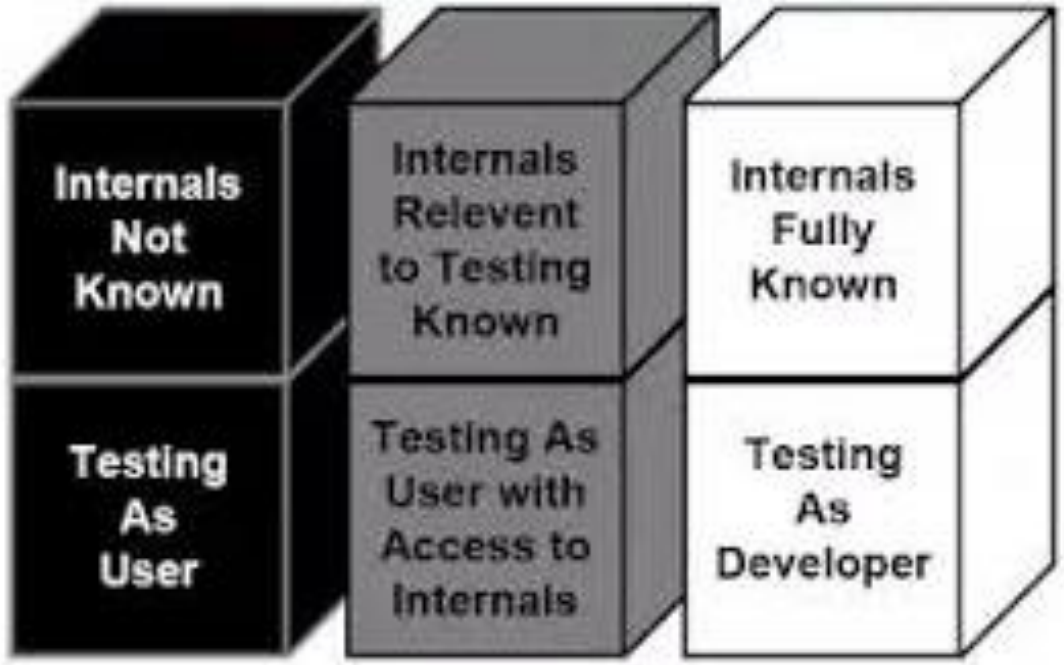
Verification: "Are we building the product right?"

Validation: "Are we building the right product?"

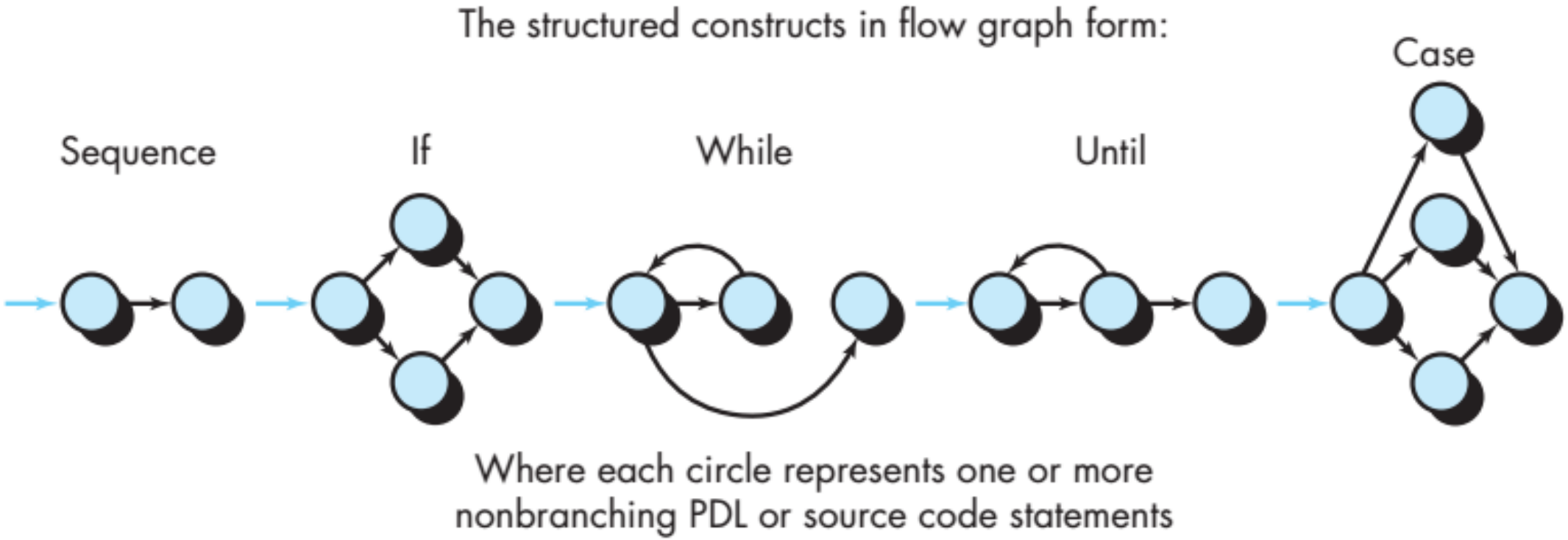


# Black Box vs. White Box Testing

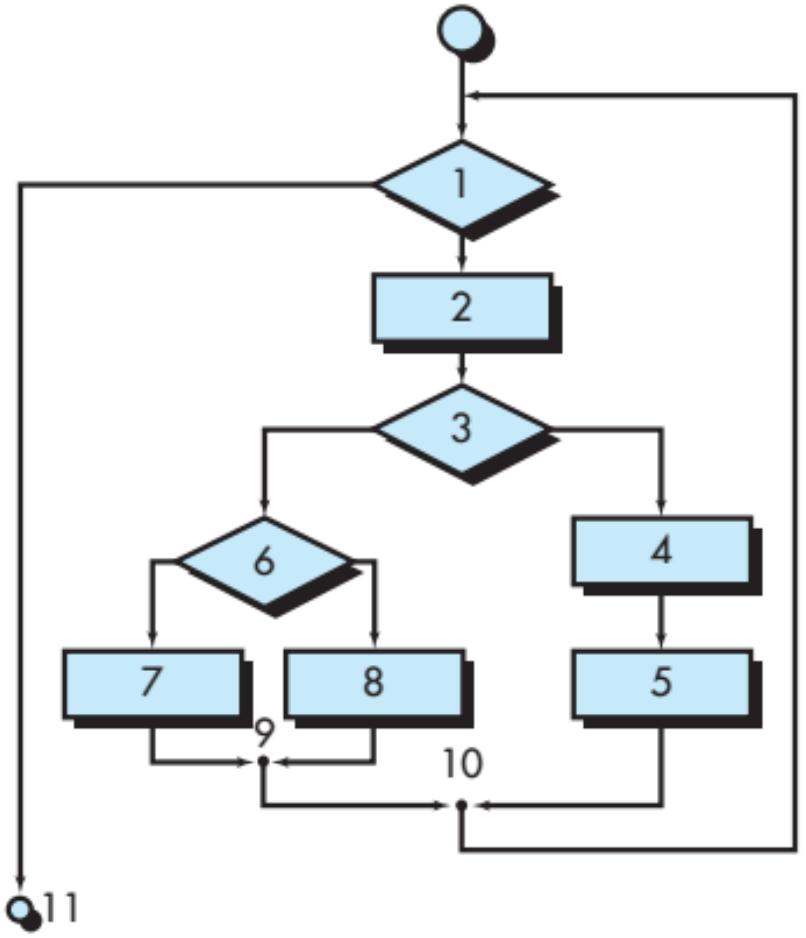


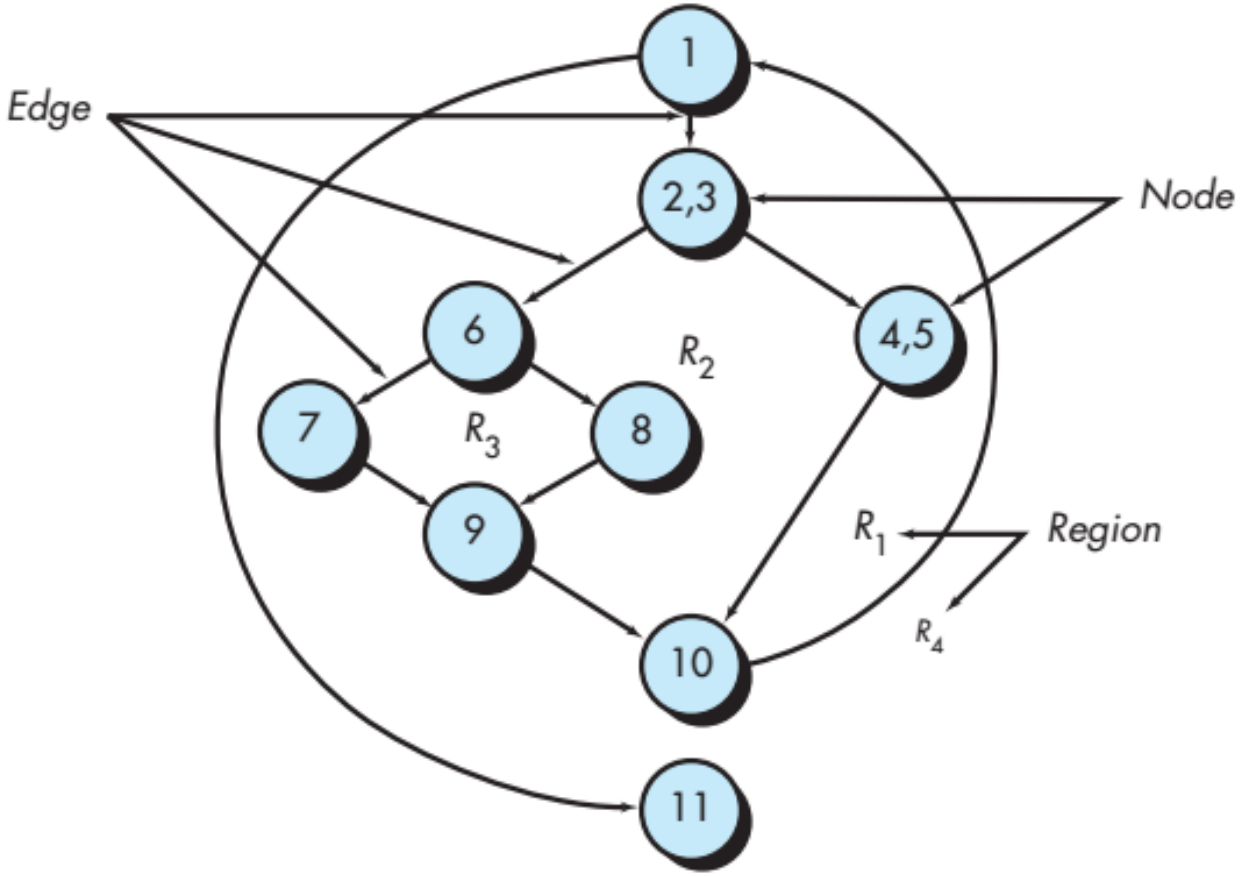
# Basis-Path Testing & Flow Graph Notation



# Flow Chart vs. Flow Graph




(a)

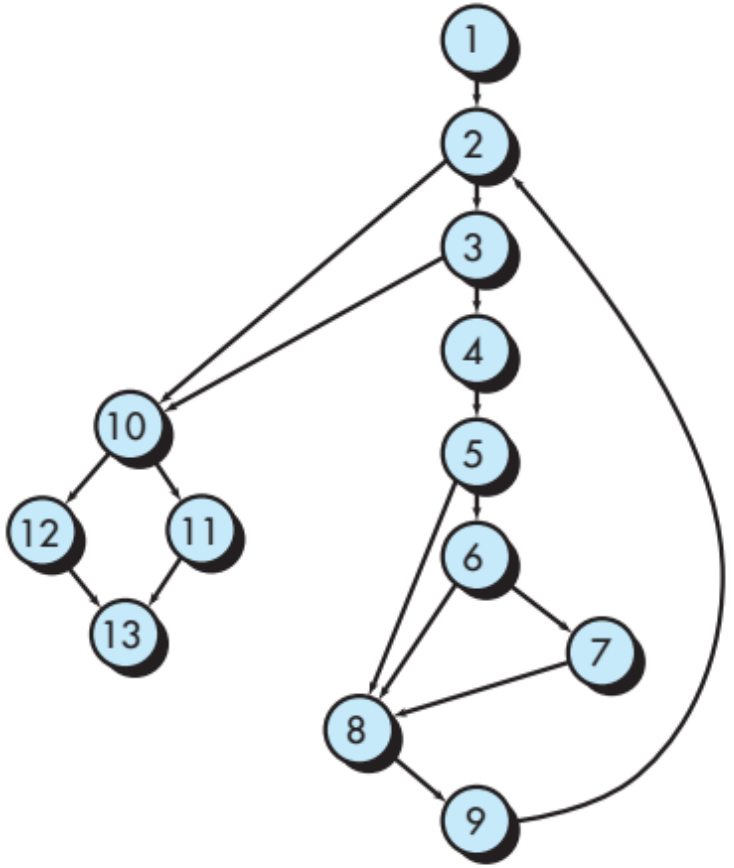



(b)

# Deriving Test Cases

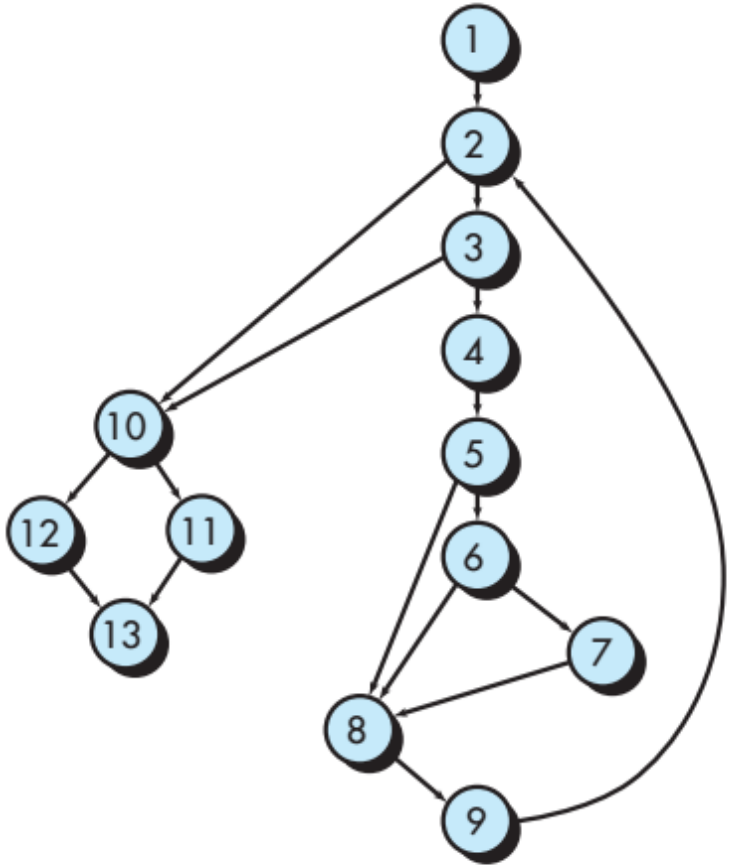
- Step 1: convert code or design to flow path

```
PROCEDURE average;

    *  This procedure computes the average of 100 or fewer
       numbers that lie between bounding values; it also computes the
       sum and the total number valid.

    INTERFACE RETURNS average, total.input, total.valid;
    INTERFACE ACCEPTS value, minimum, maximum;

    TYPE value[1:100] IS SCALAR ARRAY;
    TYPE average, total.input, total.valid;
         minimum, maximum, sum IS SCALAR;
    TYPE i IS INTEGER;
1   i = 1;
    total.input = total.valid = 0;
    sum = 0;
2   DO WHILE value[i] <> –999 AND total.input < 100
3       increment total.input by 1;
4       IF value[i] > = minimum AND value[i] < = maximum
5           THEN increment total.valid by 1;
7                sum = s sum + value[i]
            ELSE skip
6       ENDIF
8       increment i by 1;
9   ENDDO
    IF total.valid > 0
10      THEN average = sum / total.valid;
11      ELSE average = –999;
12  ENDIF
13  END average
```



# Deriving Test Cases

- Step 2: determine cyclomatic complexity of the resultant flow graph

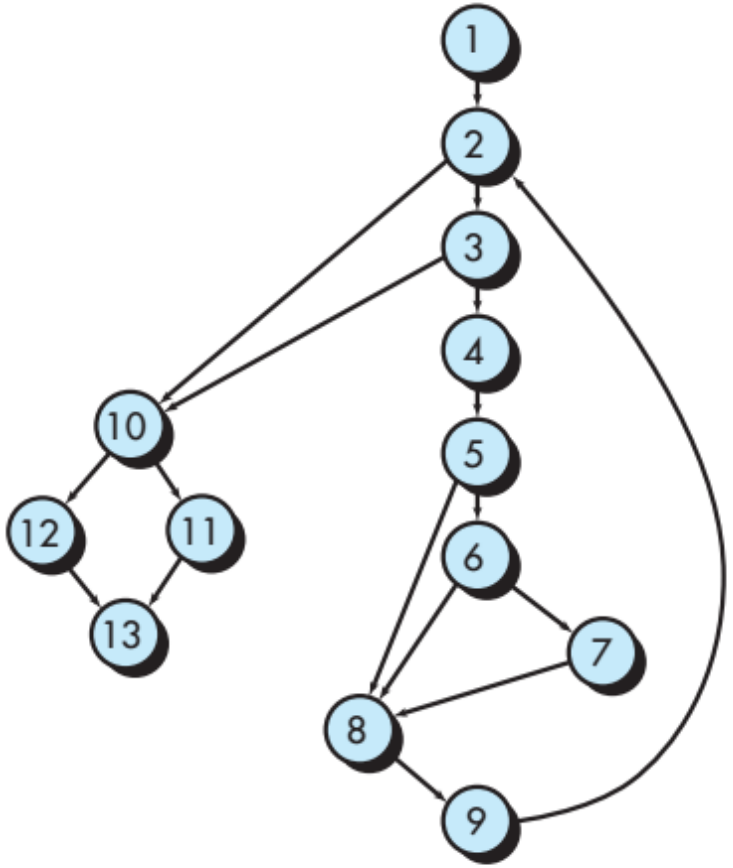



$$V(G) = 6 \text{ regions}$$

$$V(G) = 17 \text{ edges} - 13 \text{ nodes} + 2 = 6$$

# Deriving Test Cases

- Step 3: determine a basis set of linearly independent paths




Path 1: 1-2-10-11-13

Path 2: 1-2-10-12-13

Path 3: 1-2-3-10-11-13

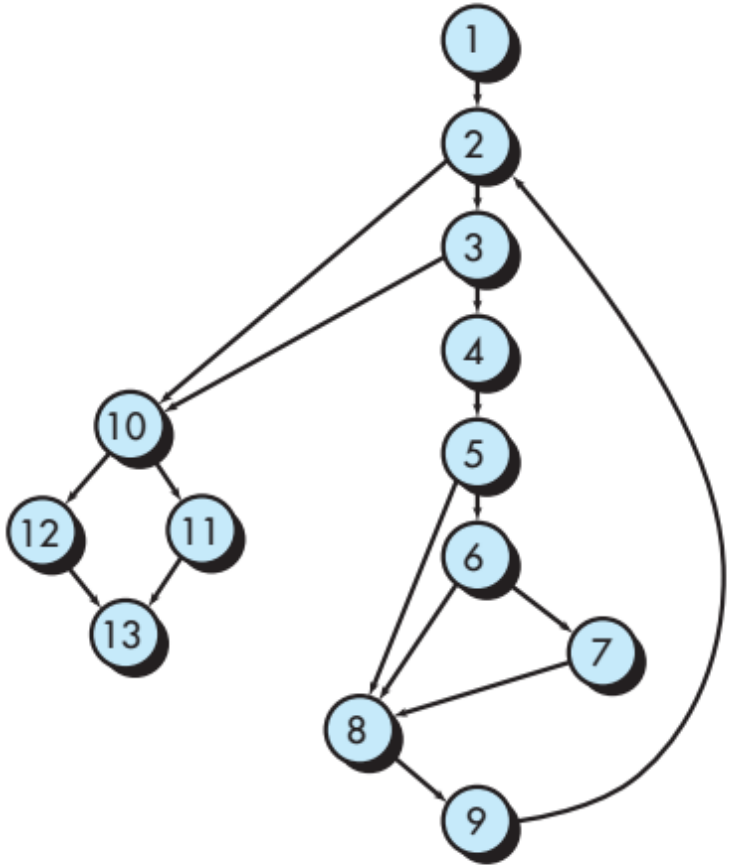
Path 4: 1-2-3-4-5-8-9-2-. . .

Path 5: 1-2-3-4-5-6-8-9-2-. . .

Path 6: 1-2-3-4-5-6-7-8-9-2-. . .

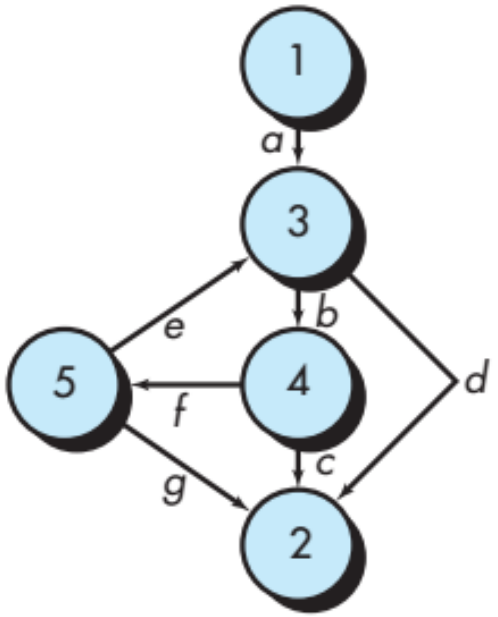
# Deriving Test Cases

- Step 4: prepare test cases that will force execution of each independent path




| Test case | Test parameters | | | |
|---|---|---|---|---|
| | P1 | P2 | P3 | P4 |
| 1 | 1 | 1 | 1 | 1 |
| 2 | 1 | 2 | 2 | 2 |
| 3 | 1 | 3 | 3 | 3 |
| 4 | 2 | 1 | 2 | 3 |
| 5 | 2 | 2 | 3 | 1 |
| 6 | 2 | 3 | 1 | 2 |
| 7 | 3 | 1 | 3 | 2 |
| 8 | 3 | 2 | 1 | 3 |
| 9 | 3 | 3 | 2 | 1 |

# Graph Matrix



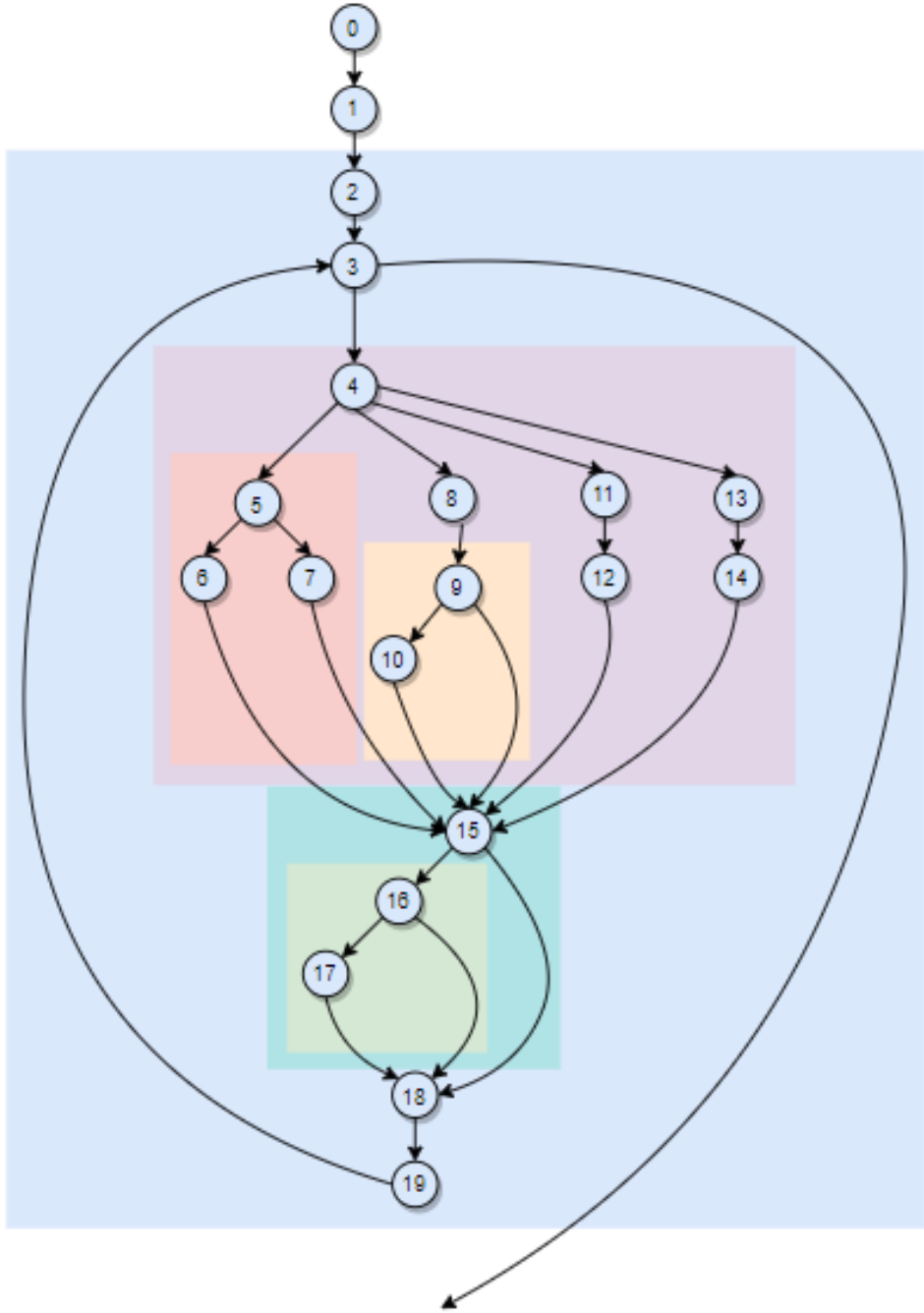
Flow graph

| Node \ Connected to node | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | | | a | | |
| 2 | | | | | |
| 3 | | d | | b | |
| 4 | | c | | | f |
| 5 | | g | e | | |

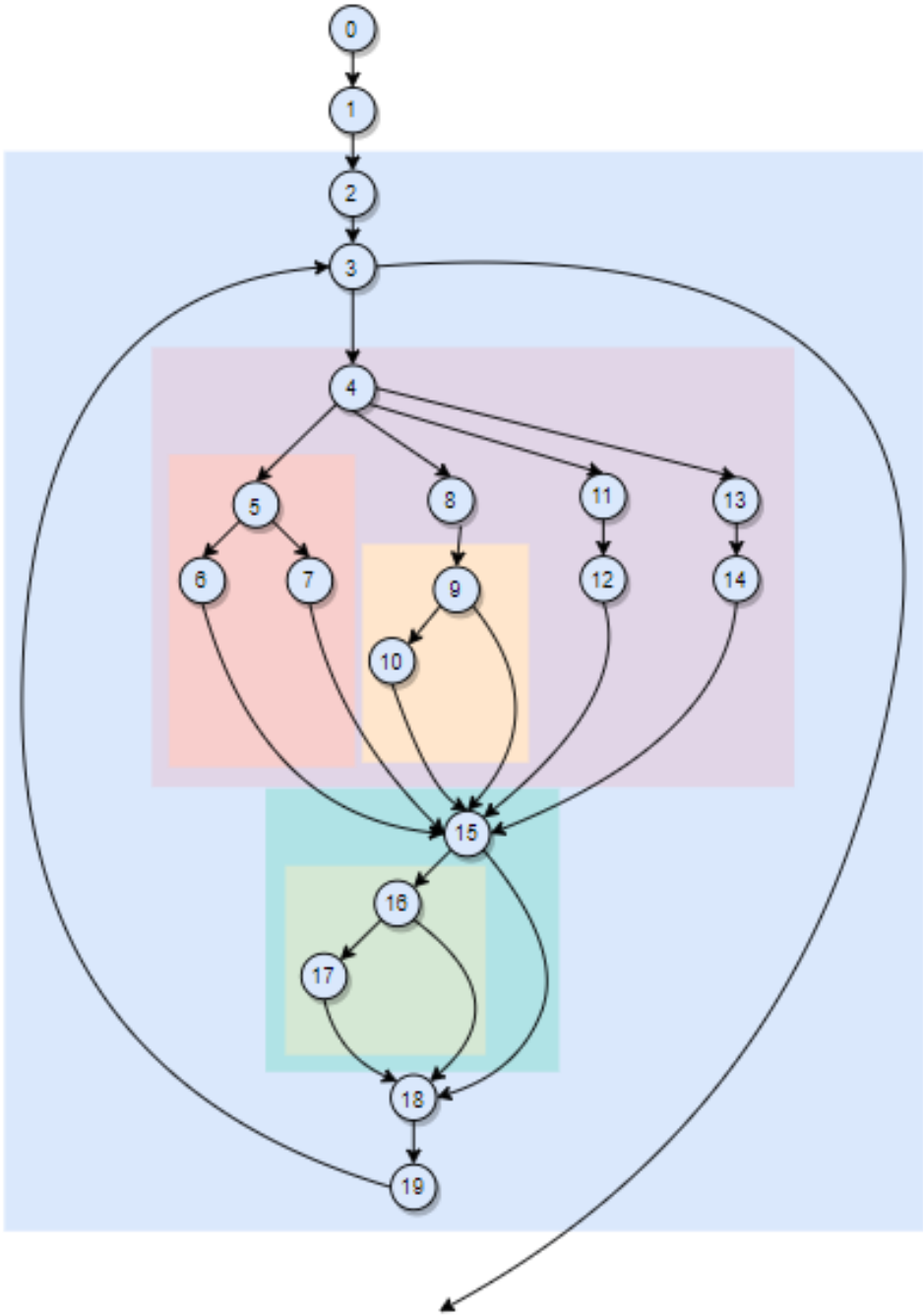
Graph matrix

- The probability that a link (edge) will be executed.
- The processing time expended during traversal of a link
- The memory required during traversal of a link
- The resources required during traversal of a link.

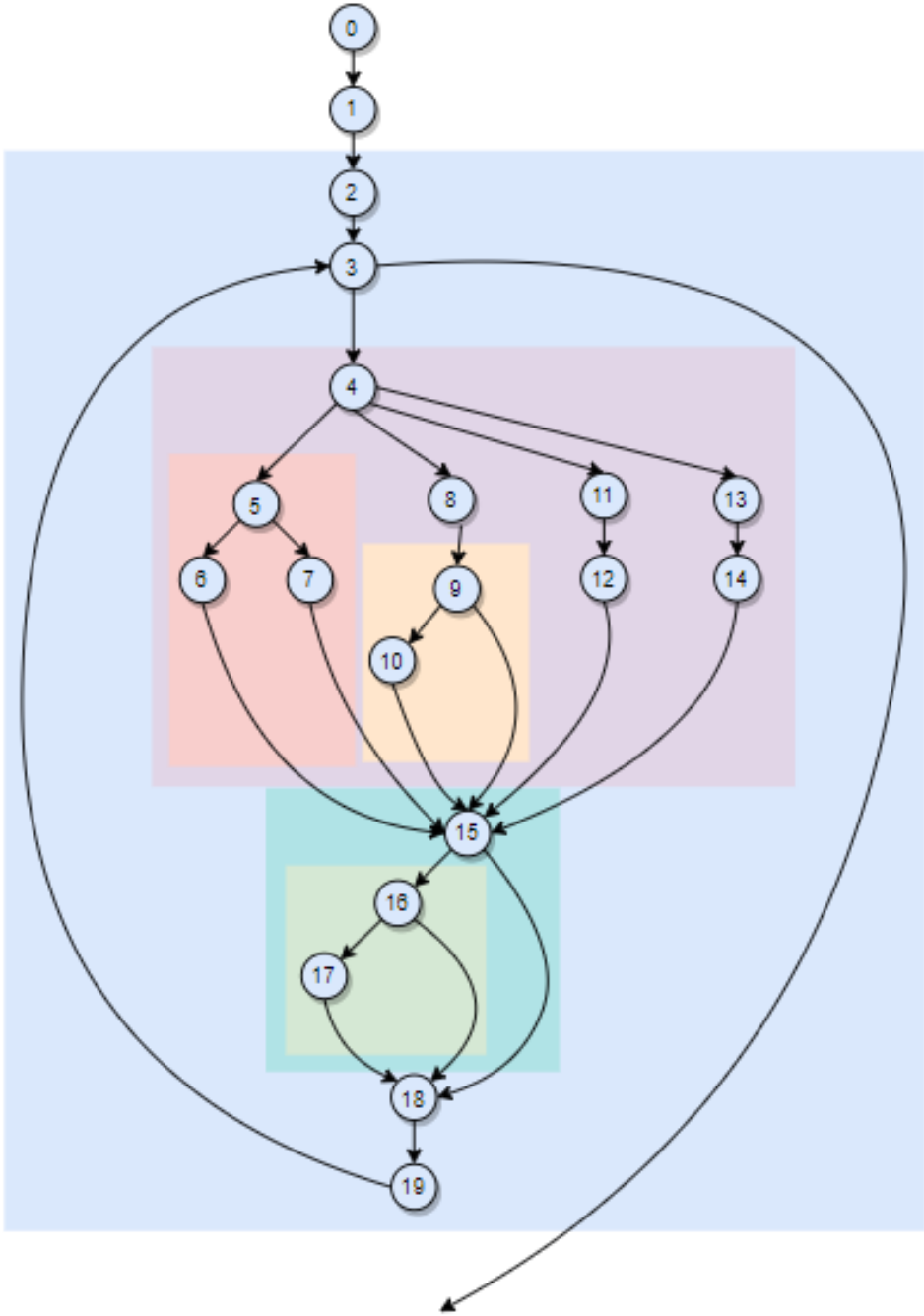
```
items_list = []  # list of dictionary for item types
data = request.POST.dict()  # Get request.POST as a regular dictionary
i = 0
next_key = 'id_form-' + str(i) + '-type'  # a.k.a.: 'id_form-' + str(i) + '-type'
while next_key in data:
    """
    This loop condition should work for all items in the donation since all
    items will have the key 'id_form-' + str(i) + '-type'.
    """

    item_dict = {}
    item_dict['quantity'] = data['id_form-' + str(i) + '-quantity']

    # Get the Item subclass
    item_dict["subclass"] = data['id_form-' + str(i) + '-type']
    if item_dict['subclass'] == 'giftcard':
        item_dict['subclass'] = GiftCard
        item_dict['amount'] = data['id_form-' + str(i) + '-amount']  # TODO: fix once the form
        # get the Giftcard enumerated value
        if 'id_form-' + str(i) + '-sub_type_business' in data:
            item_dict['businessName'] = data['id_form-' + str(i) + '-sub_type_business']
        else:
            item_dict['businessName'] = ""

    elif item_dict['subclass'] == 'clothing':
        item_dict["subclass"] = Clothing
        # get the Clothing enumerated value
        if 'id_form-' + str(i-1) + '-sub_type_clothing' in data:
            item_dict['clothingTypeName'] = data['id_form-' + str(i-1) + '-sub_type_clothing']
        # else:
        #     item_dict['clothingTypeName'] = "men"

    elif item_dict['subclass'] == 'food':
        item_dict["subclass"] = Food

    elif item_dict['subclass'] == 'misc':
        item_dict["subclass"] = Miscellaneous

    if item_dict['subclass'] == Food or item_dict['subclass'] == Miscellaneous:
        # get the name of the Food/Misc
        if 'id_form-' + str(i) + '-sub_type_name' in data:
            item_dict['name'] = data['id_form-' + str(i) + '-sub_type_name']
        else:
            item_dict['name'] = ""

    # Add the item to the list
    items_list.append(item_dict)

    # Set up the next iteration
    i += 1
    next_key = 'id_form-' + str(i) + '-type'
```



- V(G) = E – N + 2 = 28 - 19 + 2 = 11
- Linearly Independent Paths = {F} + {M} + {E}
- Test-Cases = *Each of 11 LI Paths tested with any other piece*
- Front Paths (F):
  - 0-1-2-3-20
  - 0-1-2-3-4
- Middle Paths (M):
  - 4-5-6-15
  - 4-5-7-15
  - 4-8-9-10-15
  - 4-8-9-15
  - 4-11-12-15
  - 4-13-14-15
- End Paths (E):
  - 15-16-17-18-19-3
  - 15-16-18-19-3
  - 15-18-19-3

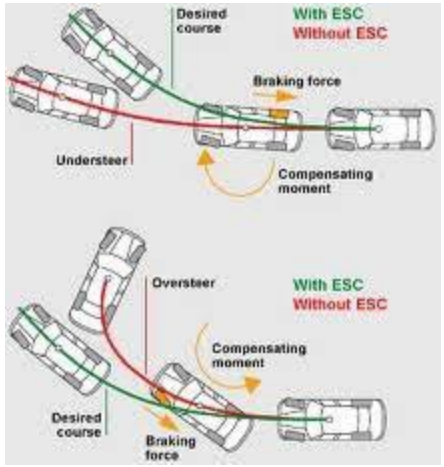


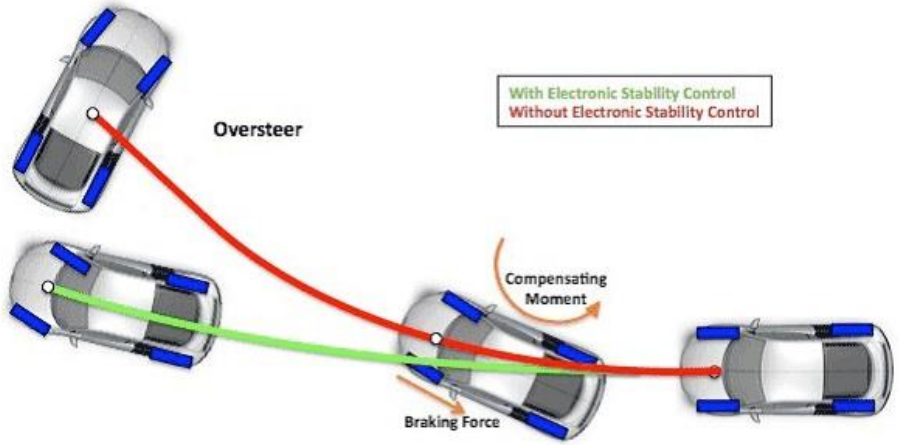
| Test Case | Parameter Values |
|---|---|
| F1 | next_key = X<br>X not in data |
| F2 | … |
| M1 | |
| M2 | |
| M3 | |
| M4 | |
| M5 | |
| M6 | |
| E1 | |
| E2 | |
| E3 | |

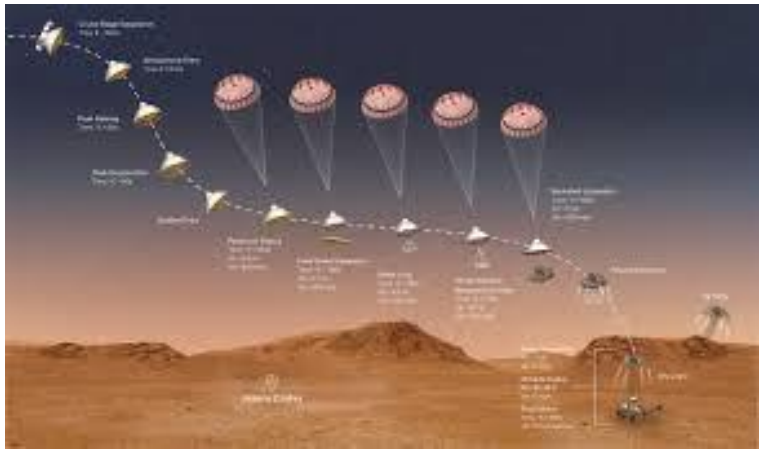


# Security Engineering

- Dependability (operates under hostile conditions)

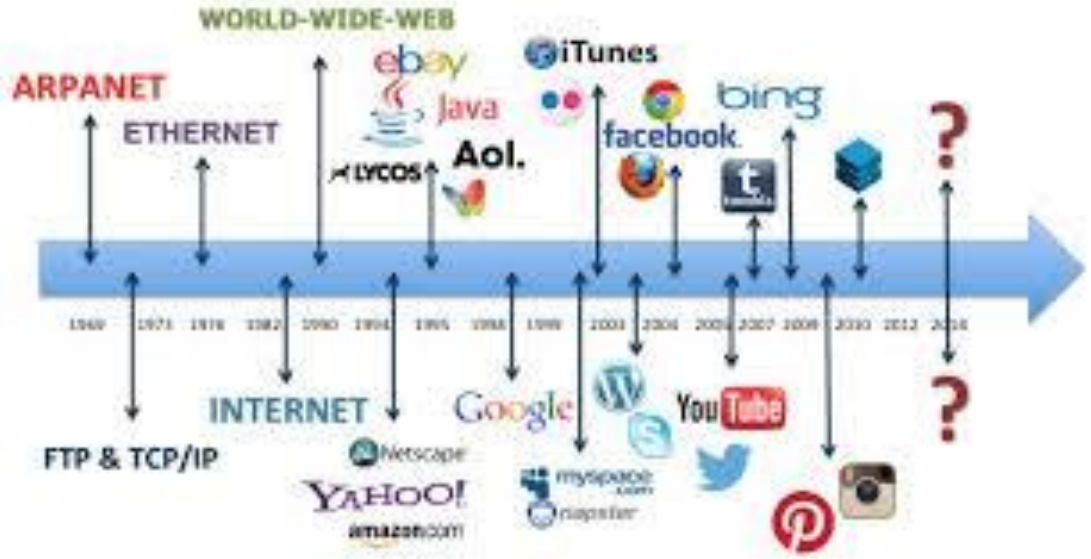


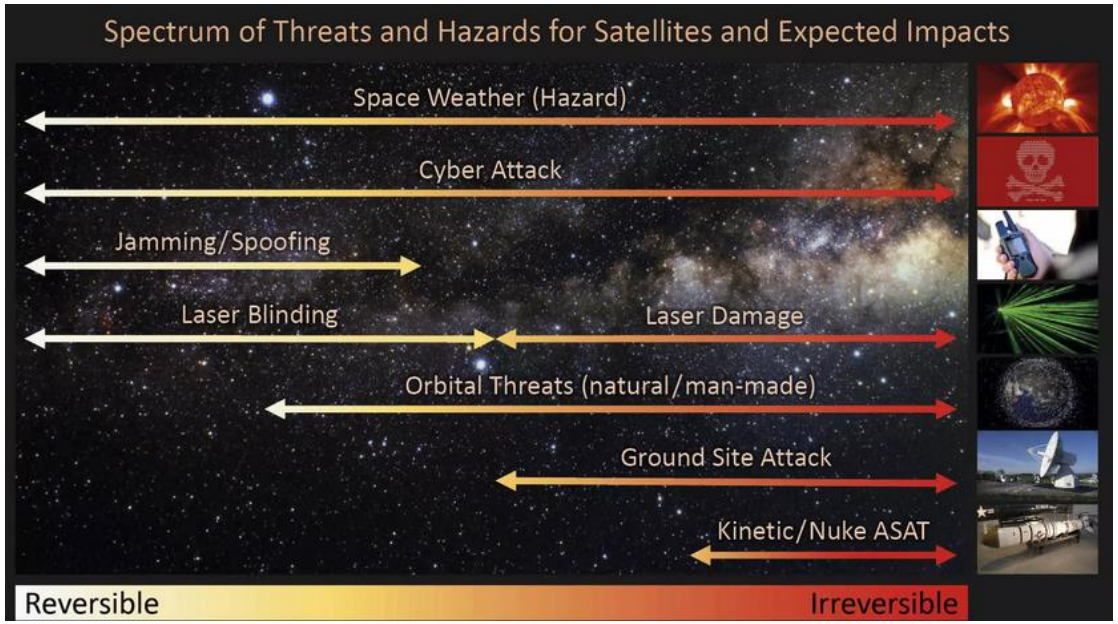
# Security Engineering

- Trustworthiness (system will not behave in malicious manner)

# Security Engineering

- Survivability (continues to operate when compromised)



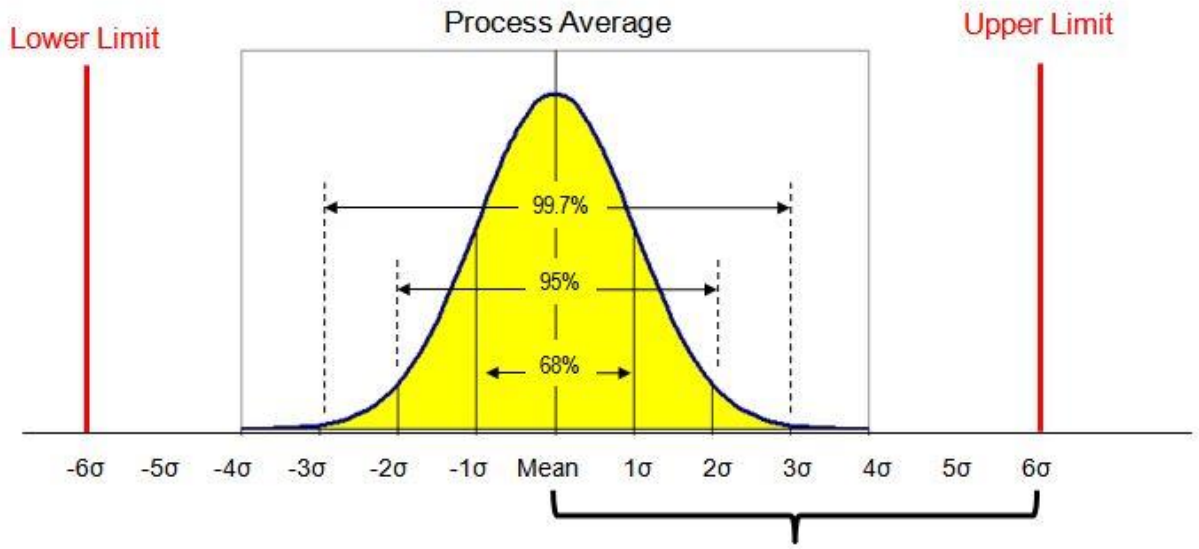
# Project Deliverable Expectations

| Folder | Deliverables |
|---|---|
| Business | Executive Summary on Purpose, Value Proposition & Customer |
| Model | Architecture Diagram & Sub-Diagrams |
| Code | Source Code,<br>Instructions to Deploy (i.e., ReadMe) |
| Documents | User Stories<br>Requirements Table |
| V&V | Test Cases, Test Code, Test Results |
| Project Management | Week-Over-Week Schedule,<br>Week-Over-Week Metrics |

# Six Sigma

- *Define* customer requirements and deliverables and project goals via well-defined methods of customer communication.
- *Measure* the existing process and its output to determine current quality performance (collect defect metrics).
- *Analyze* defect metrics and determine the vital few causes.

- *Improve* the process by eliminating the root causes of defects.
- *Control* the process to ensure that future work does not reintroduce the causes of defects.





**Define** = User Stories & Requirements Tables, Architecture Diagrams, Schedules

**Measure** = CPI, Risk, Defect Amplification Model, Error Density, Availability, Software Maturity Index

**Analyze** = What do these metrics mean? The "Why" in Scrum & Status Tag-Ups

**Analytics & ML**

**Improve & Control** = Kaizen