CSE 219

COMPUTER SCIENCE III

PROPERTIES OF HIGH QUALITY SOFTWARE

This semester I intend to brainwash you

- Important Principles for creating a Software Solution:

- First Define the Problem
- Design, then Code
- Always Provide Feedback

- First Define the Problem
- Design, then Code
- Always Provide Feedback

- First Define the Problem
- Design, then Code
- Always Provide Feedback

- First Define the Problem
- Design, then Code
- Always Provide Feedback

- First Define the Problem
- Design, then Code
- Always Provide Feedback

- First Define the Problem
- Design, then Code
- Always Provide Feedback

Do you remember why we're here?

- To learn a *methodology* for constructing software systems of high quality

- **Do you remember:**
  - what makes software high quality?
  - the Software Development Life Cycle?

What properties make a software system *high quality*?

- Correctness
- Efficiency
- Ease of use
  - for the user
  - for other programmers using your framework
- Reliability/robustness
- Reusability
- Extensibility
- Scalability
- Maintainability, Readability, Modifiability, Testability, etc.

Correctness

- Does the program perform its intended function?
  - And does it produce the *correct* results?

- This is not just an implementation (coding) issue
  - *Correctness* is a function of the problem definition

- A flawed Requirements Analysis results in a flawed Design

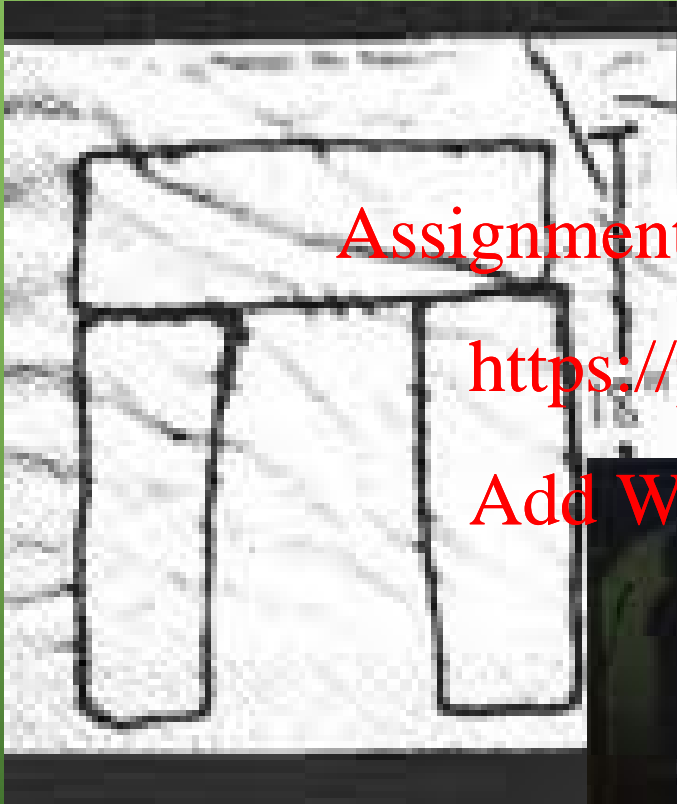- A flawed Design results in a flawed program

Garbage In – Garbage Out

- Ever see *This is Spinal Tap*?

Efficiency

- Plan for efficiency
  - wisely choose your data structures & algorthms, O(N), in the *design phase*
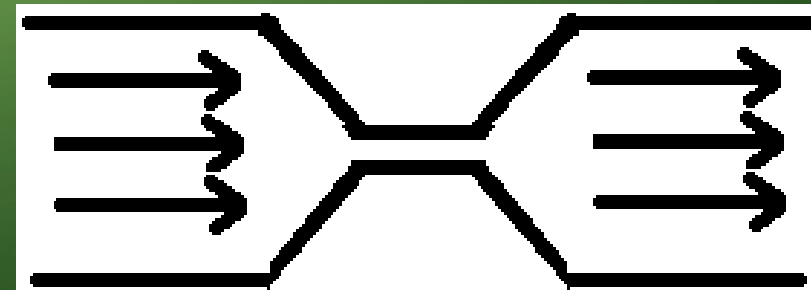 - tools & technologies too.

- Does the program meet user performance expectations?

- If not, find the bottlenecks
  - done after implementation
  - called *profiling*

Ease of Use for End User

- Is the GUI easy to *learn* to use?
  - a gently sloped learning curve

- What makes a GUI easy to use?
  - familiar GUI structures
  - familiar icons when possible instead of text
  - components logically organized & grouped
  - appealing to look at
    - colors, alignment, balance, etc.
  - forgiving of user mistakes
  - help, tooltips, and other cues available
  - etc.

Ease of Use for other Programmers

- In particular for frameworks
  - think of the Java API developers

- Should you even build a framework?

- What makes a framework easy to use?
  - logical structure (models of the world)
  - naming choices (classes, methods, etc.)
  - flexibility (usable for many purposes)
  - feedback (exceptions for improper use)
  - documentation (APIs & tutorials)
  - etc.

Reliability/Robustness

- Does your program:
  - anticipate erroneous input?
  - anticipate all potential program conditions?
  - handle erroneous input intelligently?
    - again, think about this in the design stage
  - provide *graceful degradation?*

- What's *graceful degradation*?

Graceful Degradation

- An error condition occurs in your program

- Should your program:
  - crash? Assignment Project Exam Help
  - exit? https://powcoder.com
  - notify the user and exit? Add WeChat powcoder
  - provide an approximated service?
    - not always possible

- Web Browsers
  - What does they do with poorly formatted HTML?

Feedback, Feedback, Feedback, Feedback

- Feedback

- Feedback

- Feedback

- Feedback

- Feedback

- Feedback

- Feedback

- Feedback

- Feedback

- Feedback

- Feedback

Assignment Project Exam Help

- Feedback

https://powcoder.com

Add WeChat powcoder

- Feedback

- Feedback

Feedback to whom?

- *End user* due to:
  - bad input, equipment failure, missing files, etc.
- How?

  - popup dialogs, highlighting (red x in Web form), etc.

- *Other programmers* using your framework due to:
  - passing bad data, incorrect initialization, etc.
- How?
  - exception throwing, error value returning, etc.

Flexibility in a Framework

- Programmers need to know:
  - when and why things in a framework might go wrong
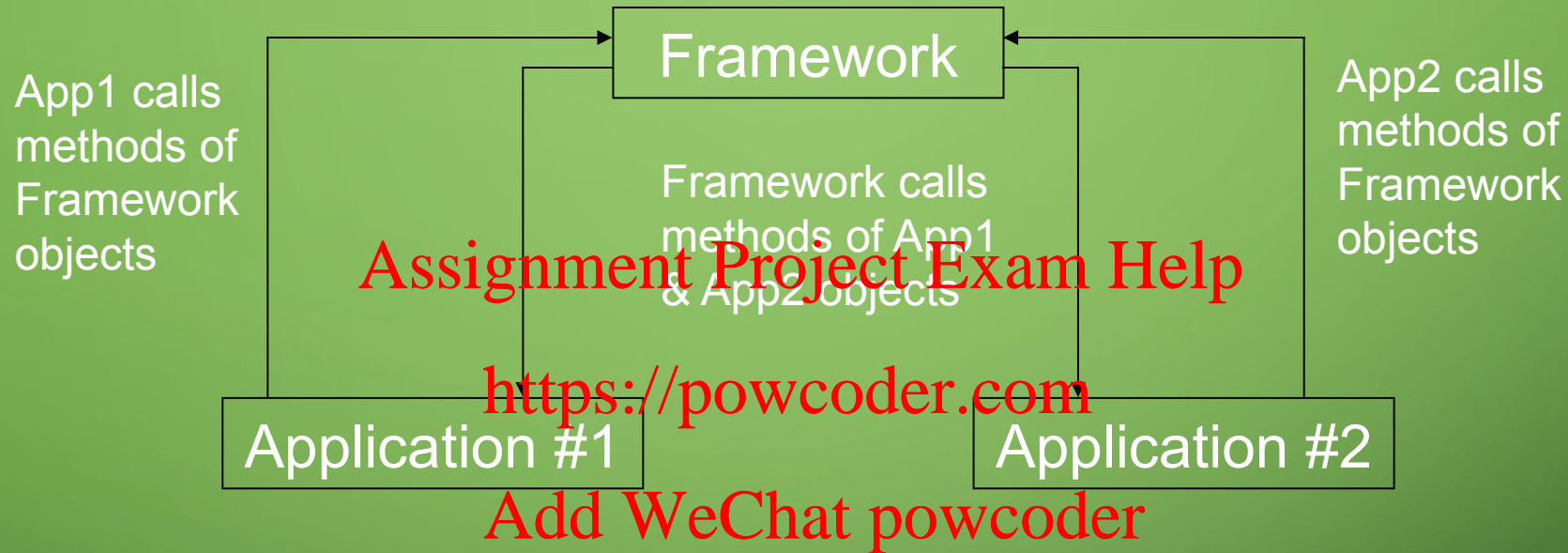
  AND

  - When and why things in a framework do go wrong

- Why?
  - customized response
  - Ex:
    - `System.out.println` notification
    - `JOptionPane` notification
    - Web page generated and sent via **Servle**t notification
    - use alternative services
    - etc.

Applications Using Frameworks

App1 calls methods of Framework objects

Framework

Framework calls methods of App1 & App2 objects

App2 calls methods of Framework objects

Application #1

Application #2

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder

- Note, making a framework is much more difficult than making an application. Why?

Reusability

- Code serving multiple purposes

- Who cares?
  - management does
    - avoid duplication of work (save $)
  - software engineering does
    - avoid duplication of work (save time & avoid mistakes)

- How can we achieve this?
  - careful *program decomposition*
  - separate technology-dependent components

Extensibility

- Can the software easily be extended?

- Huh?
  - can it be used for other purposes

- Ex:
  - plug-ins
  - exporters
  - add-ons
  - etc

Extensibility Example

- In NetBeans, try Tools → Plugins

- Anyone can make a plugin

- Download, install, and share

Scalability

- How will the program perform when we increase:
  - # of users/connections
  - amount of data processed
  - # of geographic locations users are from

- A function of design as well as technology

And More

- Maintainability

- Readability

- Modifiability

- Testability
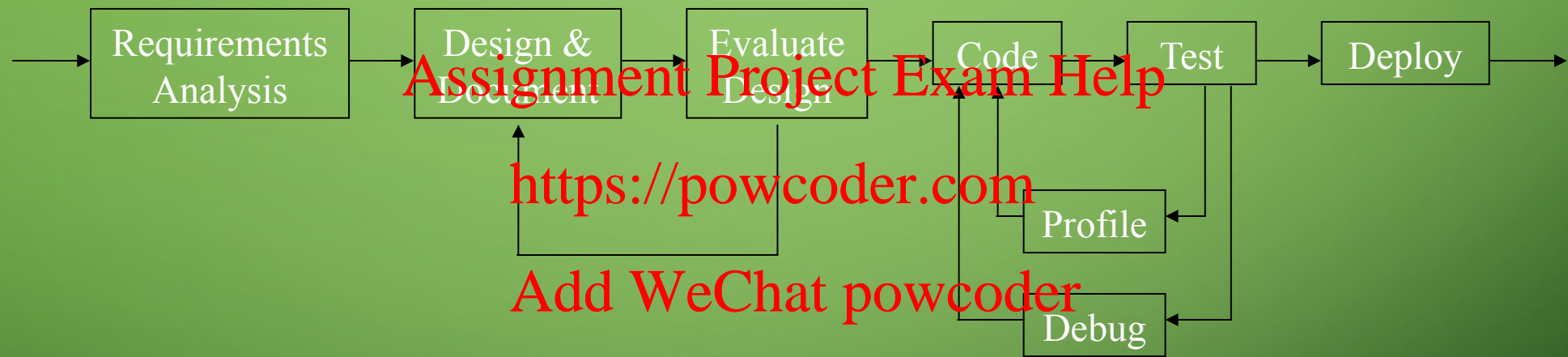
- etc.

- All of these, as with the others, must be considered early in design

How can these properties be achieved?

- By using well proven, established processes
  - preferably while taking advantage of good tools

| Requirements Analysis | Design & Document | Evaluate Design | Code | Test | Deploy |

Profile

Debug

- Software Development Life Cycle

Hopefully, the point of no return

- Correctness, Efficiency, Ease of use, Reliability/robustness, Reusability, Maintainability, Modifiability, Testability, Extensibility, Scalability

- When should we consider these properties?
  - the requirements analysis & design stages

- How about the implementation stages?
  - too late to make a big impact