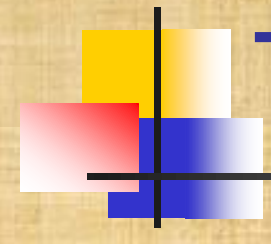
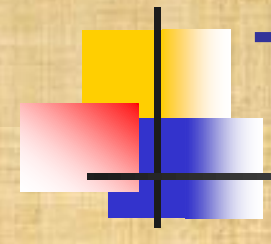


The Role of Testing

- Not a separate phase but an intrinsic component of every phase
- Must accompany every s.e. activity
- At end of every phase, additional testing by team of software engineers
  - Ex. Check sketch (flowchart? Class diagram, sequence diagram) of module for correct logic

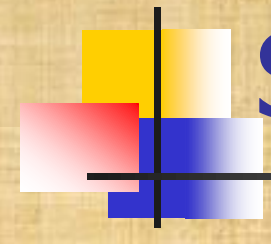


Testing

- Check completed (compiled and linked) module against various test cases
- Finally, rigorous testing by software quality assurance grout (SQA)
- Components of the software development process
  1. Plan
  2. Do
  3. Check
  4. Act

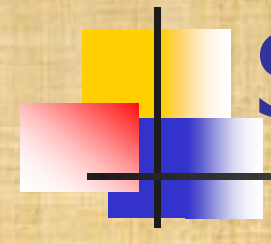
The Four Components

1. Plan: Define your objective and the strategy and supporting methods to achieve objective
2. Do: Execute the plan. Create the conditions and perform the necessary training to execute the plan
3. Check: Check the results. Determine whether work is progressing according to the plan and whether expected results are obtained
4. Act: Take the necessary action based on checkup results



Software Testing

- Who's associated with Testing?
  - Software customer
  - Software user
  - Software developer

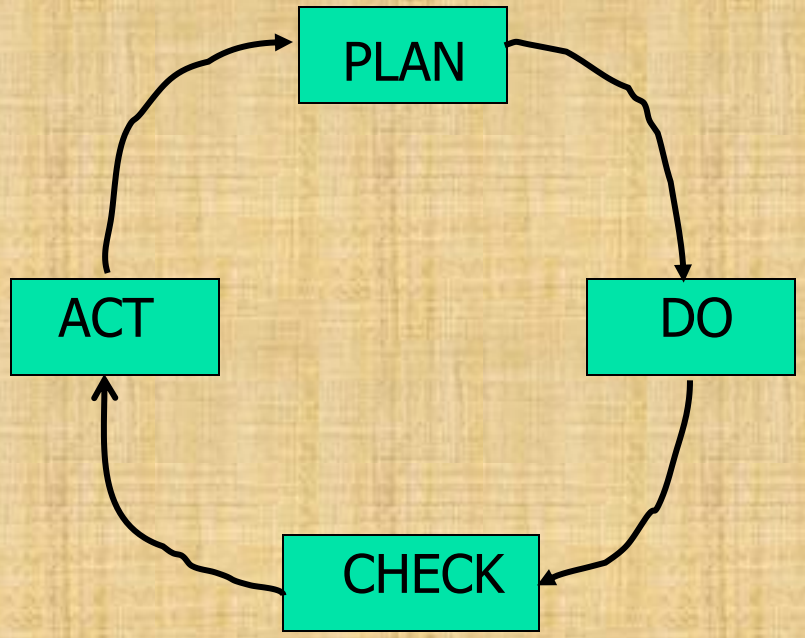


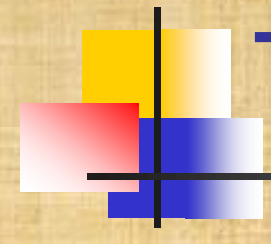
Software Testing – Who contd

- Software tester
- IT management
  - Individual or group with responsibility for fulfilling the IT mission

Software Testing

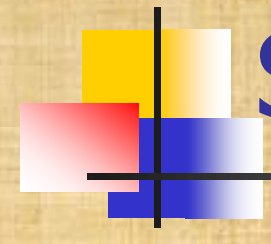
- The four Components of Software Development Process





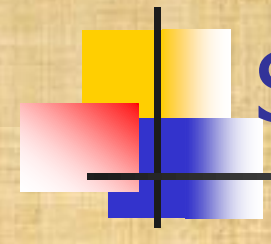
The Role of Testing

- Check completed (compiled and linked) module against various test cases
- Finally, rigorous testing by software quality assurance team (SQA)
- Types of Testing:
  - Test by engineers involved in that phase
  - Rigorous testing by independent team



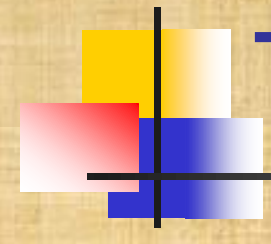
Software Testing Concepts

- Typically very little time spent learning about software testing
- Questions:
  - What is software testing?
  - What's our objective as we conduct testing?
  - Why can't we guarantee that we'll find all errors before we deliver the software to the user?



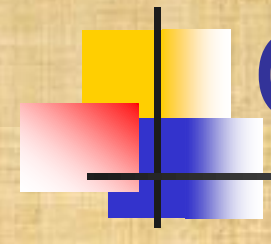
Software Testing – The Big Picture

- Although software testing is the most time and resource-consuming of all s.e. activities, many software developers do not understand it
- The need for testing is recognized but managers and practitioners often miss the "big picture"



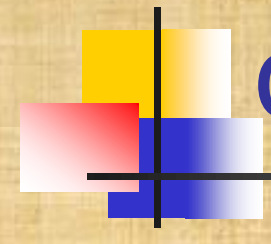
The Objectives of Testing

- Testing is a process of executing a program with the intent of finding an error
- A good test case is one that has a high probability of finding an as yet undiscovered error
- A successful test is one that uncovers an as yet undiscovered error
- The above are counter to the common held view that a successful test is one in which no errors are found



Our Objective!!!

- Design tests that systematically uncover different classes of errors and to do so with a minimum amount of time and effort
- Testing conducted successfully will uncover errors in the software
- As a secondary benefit, testing demonstrates that:
  - Software functions appear to work according to specification
  - Performance requirements appear to have been met
  - Data collected during testing provide a good indication of software reliability and some indication of software quality

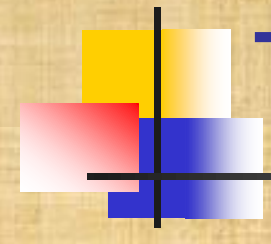


objective

- NOTE: Testing cannot show the absence of defects; it can only show that software defects are present
- Software testing is one element of a broader topic: V and V
- Verification: the set of activities that ensures that the software correctly implements a specification
- Validation: a different set of activities that ensures that the software that has been built is traceable to customer requirements

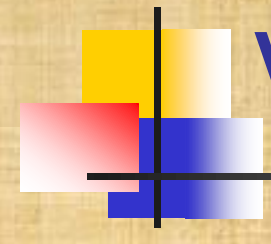
Verification & Validation

	When Process Takes Place	Description of Process
Verification	At end of each phase	Testing whether current phase has been correctly carried out
Validation	When complete product has been developed	Testing whether product as a whole satisfies its specification document



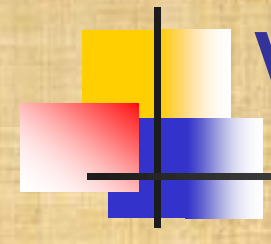
Testing

- Verification:
  - Testing at end of each phase
- Validation:
  - After the complete product has been developed and before acceptance testing
  - Does the product as a whole satisfies its specification document?



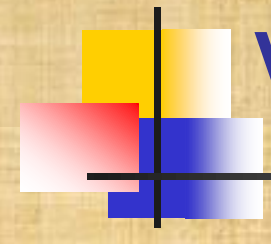
Validation and Verification

- Two major problems of software development:
  1. Make sure the software meets users' needs
  2. Produce error-free software
- Verification: Are we building the product right?
- Validation: Are we building the right product?



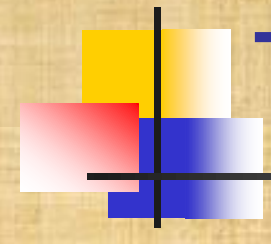
Validation and Verification

- Validation example: Acceptance testing
  - Client sees the system put through it paces and confirm it does what was expected of it
- Verification example: Testing a subroutine (method) to ensure it conforms to its specification
- The difference????
  - Validation – client's view of the system
  - Verification – internal consistencies in the system



Validation and Verification

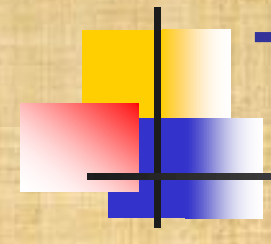
- Current dominant verification technique is testing
- Testing consumes much (~50%) of the effort in software development.
  - Hence verification is a major problem and requires improved techniques
- Many systems fail to meet users' needs, hence validation is another problem



The Problem of Testing

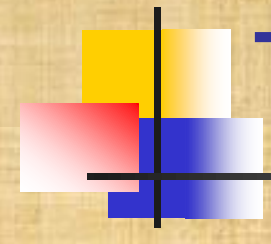
- Possible approaches:
- Devise selection of input data values and compare actual with expected outcome
  - To avoid conscious/unconscious bias have someone else devise data
- Ex. read (x, y)

```
product = x * y;
if prod = 42 then prod := 0 end if
   write (prod)
```



The Problem of Testing

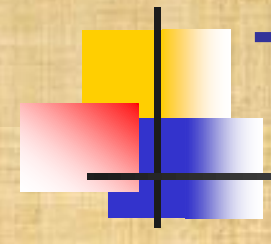
- Method 2: Use all possible input values, and check outcome. Problem!!!!!!!!
  - Ex To multiply two 32 bit integers take about 50 billion years. So exhaustive testing is almost always impractical
- Method 3: From (1) we cannot realistically test system by considering it merely as a 'black box'.
  - Adequate testing require knowledge of its internal structure



The Problem of Testing

- So, use test data that causes every program path to be executed in all possible combination.
- Testing using knowledge of internal workings of a system is "**white box**" *(Glass box!!!)* testing
- Conclusion then?? The best we can do is devise test data that cause execution of every system path at least once in testing

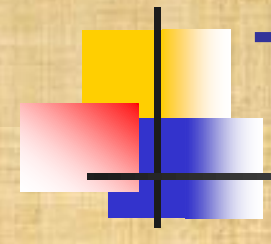
The Problem of Testing



- That's not fool proof
- Consider:

```
read (x)
while not end of file do
        total = total + x
        read (x)
endwhile
write (total)
```

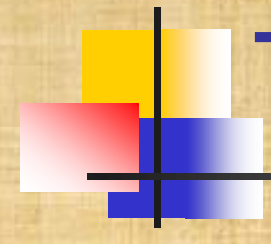
- Will sometimes give the expected answer



The Problem of Testing

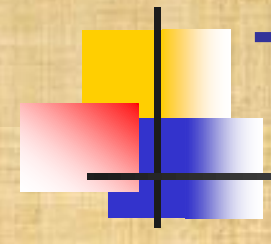
- Another view:
  - Be sure to test actions that a system takes in special cases
  - However much we test, we can never be sure all faults are eradicated.
    - According to Dijkstra, then:

      "**Testing can only show the presence of bugs, never their absence**"



The Problem of Testing

- It's tempting to view a test that reveals no bugs as a successful test. Instead, view such tests as unsuccessful!!!
- The trouble??!!!!
  - Bugs surface at the worst possible time – For example, in demo to client.
    - ***"if a system can fail, it will; and at the worst possible moment".***

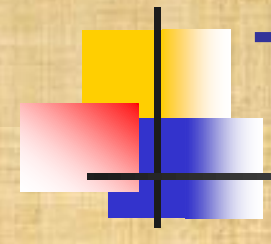


The Problem of Testing

- In the real world, evidence of limitations of testing is everywhere
- More scientifically: In experiment in 1978:

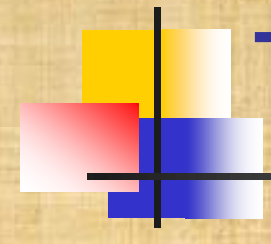
  59 people were asked to test a 63 line PL/I program.

  All worked in the computer industry and most were programmers, with an average of 11 yrs computing experience. All were told of the suspicion that the program was not perfect and asked to find all errors (if any)



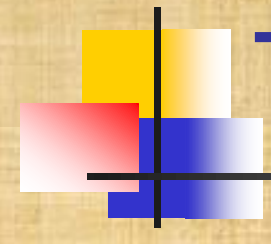
The Problem of Testing

- Error → Discrepancy between program and the specification
- People given the program spec, program listing, computer to run program, and told to take all the time needed
- Findings:
  - Mean number of bugs found? …5.7
  - Most errors found by an individual? …9
  - Least number found by anyone? …3
  - The actual number of bugs? … 15
  - **There were 4 bugs no one found**



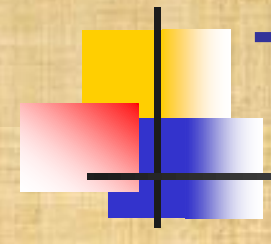
The Problem of Testing

- Additional findings:
  - The people made no careful comparison between the program's actual output and the expected outcome
  - Too much time spent on testing normal conditions the program had to encounter, rather than on testing special cases and invalid input situations



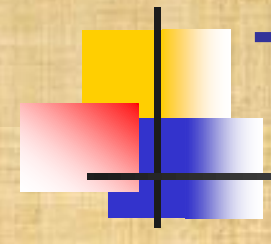
The problem of Testing

- You can't find every design error:
  - If the system does exactly what the spec says it should, and does nothing else, it meets the specs
  - Some people declare a system correct if it meets the spec, but is this reasonable?
    - What if the spec says 2 + 2 should be 8?
  - Is it a bug if the system meets a spec that has a typo, or is it a bug if the system deviates from the spec?



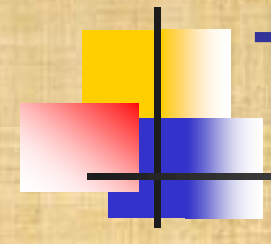
The Problem of Testing

- Specs often contain errors
  - Some are accidents (2 + 2 = 8)
  - Some are deliberate – the designer thought he/she had a better idea, but didn't
- Many user interface failings are design errors
  - Being in the spec doesn't make them right
  - If a system follows a bad spec it's wrong
- Can't find all the errors in the user interface, so
  - You can't completely test a system if you can't find all its design errors



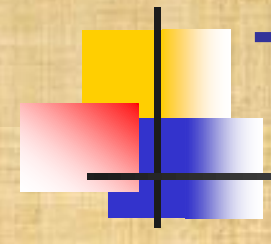
The problem with Testing

- You can't Prove System Correct using Logic
  - If program is well organized you should be able to
    - make assertions about the state of the system under various conditions
    - Prove by tracking through the logic of the system that the assertions are correct
  - Ignoring time and the number of conditions, this method can only validate the internal consistency of the system
    - It might prove that the system performs according to the spec, **but is the spec good**?



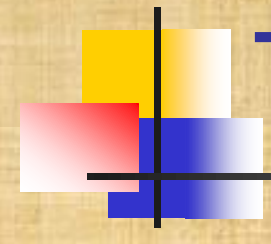
The Problem of Testing

- The Tester's Objective: System Verification
  - Testing is often described as a process of verifying that the system works correctly
    - This doesn't really make sense: You can't test the program thoroughly enough to verify that it works correctly
    - It's also mistaken: the system doesn't work correctly, so you can't verify that it does
  - It sets testers up for failure: if your goal is to show that the program works correctly, you fail every time you find an error



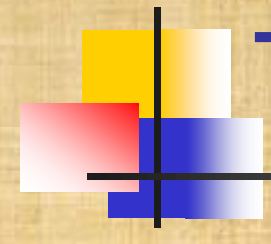
The Tester's objective contd.

- It fosters an ineffective attitude:
  - If you set your mind to showing that the system works correctly, you'll be more likely to miss problems than if you want and expect the system to fail



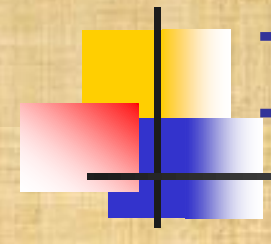
The Problem of Testing

- The System Doesn't Work Correctly;
  - It's easy to spend $100,000 testing a system – if you have the money, spending a million is only a little harder
  - Common estimates of the cost of finding and fixing errors in a system range from 40% to 80% of the total development cost
    - Companies don't spend this kind of money to "verify that a system works".
    - They spend it because the system doesn't work



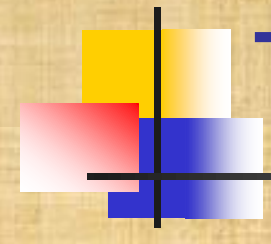
The Problem of Testing

- Is testing a failure if the system doesn't work correctly?
  - If the purpose of testing is to verify that the system works correctly then the tester fails to achieve his/her purpose every time he/she proves that the system is full of bugs



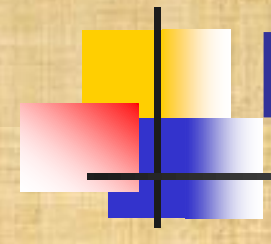
Is Testing a Failure contd.

- There are project managers who:
  - Berate testers for continuing to find errors in a system that's behind schedule
  - Blame testers for the bugs
  - Just complain (jokingly!!!) that the testers are too tough on the system:



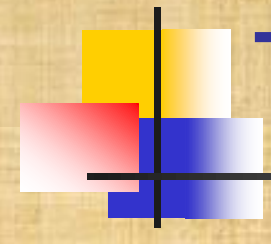
Testing

- If you think your task is to find problems, you'll look harder for them than if you think your task is to verify that the system has none
  - Psychological research suggests people tend to see what they expect to see
- If you expect to find many bugs and you're praised or rewarded for finding them, you'll find a lot



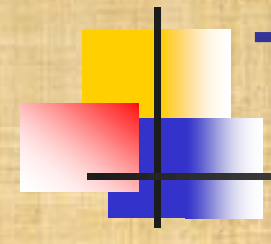
Problem with Testing

- If you want and expect a system to work, you will be more likely to see a working system
- If you expect it to fail, you will be more likely to see the problems
- If you are punished for reporting failures, you will miss failures



Testing

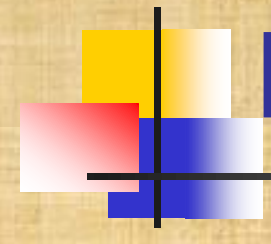
- Advise:
  - Adopt a thoroughly destructive attitude towards the system
  - You should want it to fail, expect it to fail, and
  - Concentrate on finding test cases that show its failures
- This is a harsh attitude, but it's essential



Testing

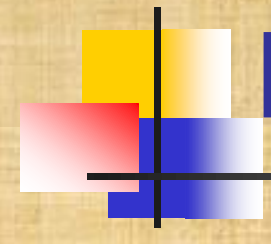
- So Why Test?
  - You can’t find all the bugs
  - You can’t prove the system correct, and you don’t want to
  - It’s expensive, frustrating, and it won’t win you any popularity contests ….

    … So why bother testing?



Purpose of Testing

- Recall, The purpose of testing a system is to find problems in it
- Finding problems should be the core of your work
- You should want to find as many faults as possible, the more serious the better
- ….Physician’s analogy!!!

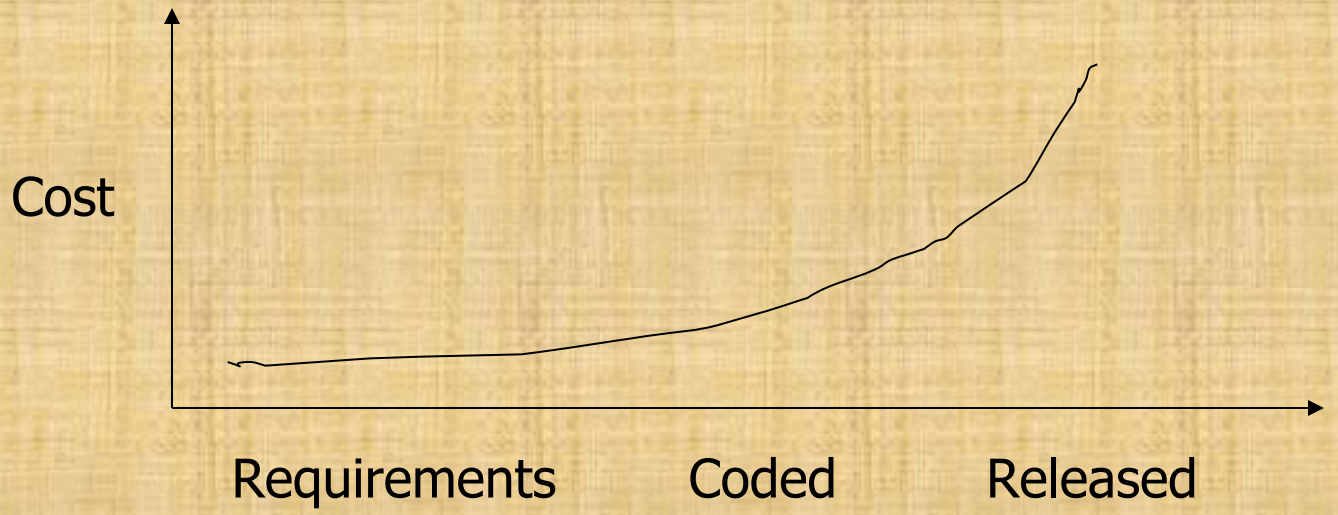


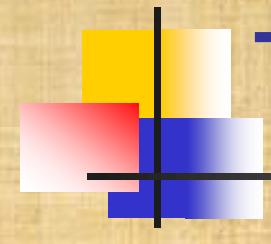
Purpose of Finding Problems

- The purpose of finding problems is to fix them
- The prime benefit of testing is that it results in improved quality
- Bugs get fixed
- You take a destructive attitude toward the system when you test, but in the long run, your work is constructive

Testing a System

- Testing and fixing can be done at any stage in the life cycle, however..
- The cost of finding and fixing errors increases dramatically as development progresses

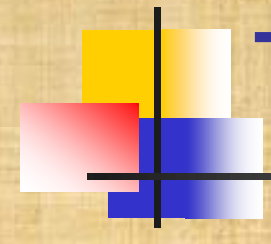




Testing Strategies

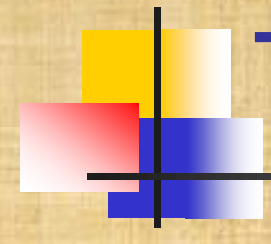
- Recall, Testing is a poor technique ..But until program proving becomes widely applicable it's a vital technique
- Different test techniques discover different errors.

- Both black box and white box testing have a role to play…



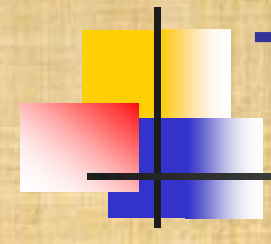
Testing Strategies

- White box testing should select test data that:
  - Exercise every program path (though not every combination of paths)
  - Explore the decisions that control the choice of path
  - Take care to test actions that are taken in special cases
- Studies show different people tend to uncover different errors, hence, this suggests using team techniques



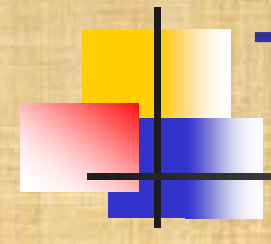
Testing the Test Data

- Difficult to ensure adequacy of test data in large systems
- Use profiler to try test if data indeed cause all statements to be executed
  - Profiler $\rightarrow$ software package that monitors the testing by inserting probes into the software under test
- Alternative method in investigating test data is **mutation testing**



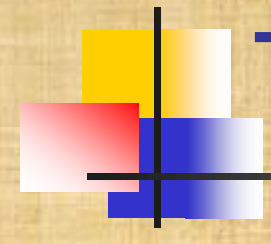
Testing the Test Data

- Mutation Testing:
  - Artificial bugs are inserted into the program
    - Example: change a “*” into “-”. If bugs are not found during testing then test data are inadequate …Modify data until artificial bugs are exposed



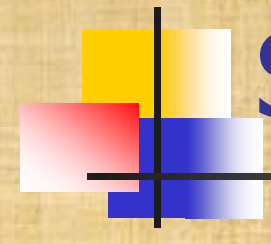
Team Techniques

- Who should carry out testing?
  - Individual or team that developed software is most familiar with it and will (supposedly) be able to draw up best test
  - Separate test individual or team will not be biased or influenced by involvement in project development
  - Developer may be blind to his/her errors; may be under deadline pressure & so skimpy on testing



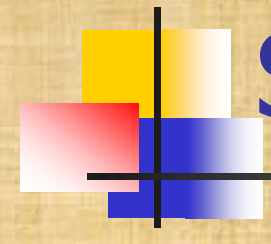
Team Techniques

- Many organizations have separate testing teams (quality assurance (QA) team)
- Fruitful grounds for conflict between teams
- To exploit conflict:
  - Set up adversary teams to do testing…
    - Harness their malice to effective discovery of bugs
- Another approach : set up bounty hunters.



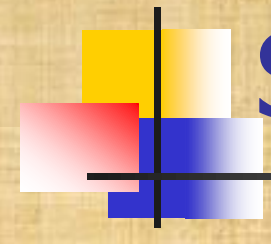
System Testing

- Follows a phase of unit or module testing, when components are tested in isolation
- Common approach to system testing is **monolithic** testing – modules are linked and tested together – This is **terrrrrrible**.. How do you know which module is the cause of the fault?
- Alternative – use **incremental** testing. i.e. Test one component of the system. Now link a second module with this and test. Any error is likely in the new module or in the interface between the two



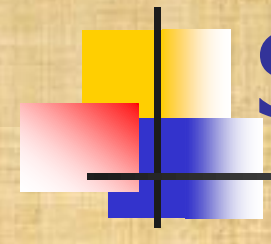
System Testing

- Continue adding one module at a time. Any fault is likely caused by new module or its interface to the rest of the system
- Integration Testing – systematic technique for conducting tests to uncover errors associated with interfacing
- The objective? Use Unit-tested modules to build a program structure that's been dictated by design



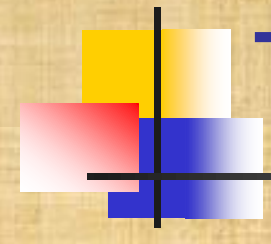
Software Testing

- Non-incremental integration (the "*big-bang*")- chaotic!!!
  - Error correction is difficult as the vast expanse of entire system complicates isolation of cause
- Integration testing – antithesis of the "big bang" approach:
  - System is constructed and tested in small segments
  - Errors easier to isolate and correct
  - software more likely to be tested completely



System Testing

- Approaches to incremental testing
  - Bottom-up
  - Top-down
- Top-down integration
  - Integrate modules by moving downward through control hierarchy, beginning with main module
  - Incorporate modules subordinate to main control structure in depth-first or breadth-first manner



Top-Down Integration

- Depth-first: integrate all modules on a major control path of the structure
- Major path is arbitrary and depends on application-specific characteristics
- Ex, M_1, M_2, M_5, first
  Next, M_8 or M_6, then
  central and right-hand
  path

Path