

Application Development

IS240 – DBMS

Lectures # 13a & 13b – 2010-04-19 & 21

M. E. Kabay, PhD, CISSP-ISSMP

Assoc. Prof. Information Assurance

Division of Business & Cyber Studies, Norwich University

<mailto:mkabay@norwich.edu>

V: 802.479.7937

1

Copyright © 2010 M. E. Kabay. All rights reserved.

Schedule

- Because of the importance of this material, these slides will be presented in two sessions
 - ❑ Monday 19 April 2010: SLIDES 1-31
 - ❑ Wednesday 21 April 2010: SLIDES 33-66

BOTH LECTURES WILL BE IN DEWEY 108

2

Copyright © 2010 M. E. Kabay. All rights reserved.

Notes on the Subject Matter

- This lecture does *not* follow Dr Post's Chapter 8 – which you should nonetheless *read*.
 - Quiz and exam questions will be based on this lecture, not on Chapter 8.
 - ❑ That material repeats much of what you have already studied in previous chapters, although it puts it in the context of writing programs that use databases.
 - However, I have opted to review more general principles of system development and quality assurance in a DBMS context.
 - ❑ These concepts are more appropriate to the level of this second-year course.
- M. E. Kabay

3

Copyright © 2010 M. E. Kabay. All rights reserved.

Source Materials

- Some slides are derived from the work of Prof Ian Sommerville, author of *Software Engineering, 6th Edition*. Pearson Education Limited (Harlow, Essex UK), ISBN 0-201-39817-X. xx + 693. Index.
 - ❑ These are marked **IS**
 - Others are from material supplied by Dr Jerry Post in his materials for this course
 - ❑ Those are marked **JP**
 - Yet others are from Prof David Kroenke's materials for Kroenke, D. M. (2001). *Database Processing, Eighth Edition*. Prentice Hall (Upper Saddle River, NJ). ISBN 0-13-064839-6.xv + 671.
 - ❑ They are marked **DK**
 - The rest are my own slides.
- M. E. Kabay

4

Copyright © 2010 M. E. Kabay. All rights reserved.

Topics



- Applications in DBMS Work
- Security in DBMS Application Development
- The Software Process
- Quality Assurance

Applications in DBMS Work



- Application Importance
- Application Organization
- User Orientation
- Where to Enforce Constraints?

Application Importance



- User interface
 - ❑ Make users' jobs easier.
 - ❑ Tie input forms and reports.
 - ❑ Automate basic tasks
 - ❑ Tie to external data collection devices.
 - ❑ Help system.
- Ensure data integrity
 - ❑ Validate data.
 - ❑ Perform computations.
 - ❑ Verify totals.
 - ❑ Control user access.
 - ❑ Maintain related transactions.
 - ❑ Backup and recovery.
- Decision Support
 - ❑ Monitoring of events.
 - ❑ Analysis, Graphs, Reports.
 - ❑ Statistical analysis and optimization.
 - ❑ Forecasts and simulation.
 - ❑ Linking to other software.
- Expert Systems & Intelligence
 - ❑ Logic and forward chaining.
 - ❑ Analysis and decisions in code.
 - ❑ Databases of cases, situations and solutions.

Application Organization



- Organized by user needs.
 - ❑ Identify user.
 - ❑ Outline tasks.
 - ❑ Organize forms and reports.
- Direct users to tasks.
- Potential drawbacks
 - ❑ Too many layers makes it difficult for users to find anything.
 - ❑ Poor organization confuses users and requires additional support and training.
- Build forms and reports.
- Start with a core concept.
 - ❑ Identify most important features. Get them correct.
 - ❑ Add features, forms and reports. Issue application updates--number and date!
- Use menu stubs for incomplete and future work.
 - ❑ Make them invisible to the user with the *Visible* property.
 - ❑ Be sure they are disabled.

User Orientation

- Database application is a model of the organization.
 - ❑ Applications based on user jobs.
 - ❑ Flexibility and user control.
- Application organization
 - ❑ User tasks.
 - ❑ User control over sequence.
- Forms
 - ❑ Minimize user entry.
 - ❑ Anticipation.
- Reports
 - ❑ Easy access from forms.
 - ❑ User selection of scope and conditions or filters.

Where to Enforce Constraints (1)?

- DBMS / pro
 - ❑ Central control over data constraints:
 - ✓ Define once, apply consistently to all applications
 - ✓ Change for all applications by modifying one rule
 - ❑ May need to examine all records -- faster
- DBMS / con
 - ❑ Not all DBMS engines handle constraints fully or well

Where to Enforce Constraints (2)?

- Application / pro
 - ❑ May be easier to implement constraints in apps
 - ❑ Some constraints are application-dependent
 - ✓ Specific business rules
 - ✓ Some domain constraints such as assigning an input to a specific field (e.g., PATIENT.Name vs DOCTOR.Name)
- General principle: try to enforce constraints in DBMS first, then move to application.

Security in DBMS Application Development

- Integrate Security into Design and Implementation
- Security Basics
- Confidentiality
- Possession
- Integrity
- Authenticity
- Availability
- Utility
- Identification, Authentication & Authorization
- Security in DBMS

Integrate Security into Design and Implementation

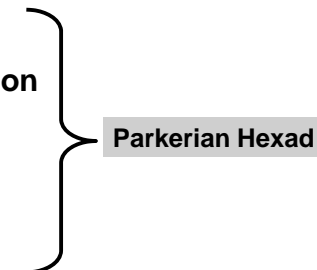


- Growing in importance as more PII exposed to Internet
 - ❑ PII = Personally Identifiable Information
 - ❑ E.g., Social Security Numbers, names, addresses, birthdates....
 - ❑ Increasing rate of identity theft: fastest-growing crime in USA
 - ❑ Catastrophic consequences for individual victims
 - ❑ Legal liability for corporations
- Don't try to add security as afterthought
- Build in security from ground up

Security Basics



- Basic concepts
 - ❑ Confidentiality
 - ❑ Control or possession
 - ❑ Integrity
 - ❑ Authenticity
 - ❑ Availability
 - ❑ Utility
- Mechanisms
 - ❑ Identification, Authentication & Authorization
 - ❑ User-specific views



Confidentiality



Restricting access to data

- Protecting against unauthorized disclosure of *existence* of data
 - ❑ E.g., allowing industrial spy to deduce nature of clientele by looking at directory names
- Protecting against unauthorized disclosure of *details* of data
 - ❑ E.g., allowing 13-yr old girl to examine HIV+ records in Florida clinic

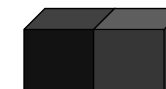


Possession



Control over information

- Preventing physical contact with data
 - ❑ E.g., case of thief who recorded ATM PINs by radio (but never looked at them)
- Preventing copying or unauthorized use of intellectual property
 - ❑ E.g., violations by software pirates

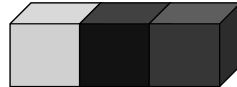


Integrity



Internal consistency, validity, fitness for use

- Avoiding physical corruption
 - ☐ E.g., database pointers trashed or data garbled
- Avoiding logical corruption
 - ☐ E.g., inconsistencies between order header total sale & sum of costs of details



17

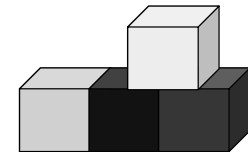
Copyright © 2010 M. E. Kabay. All rights reserved.

Authenticity



Correspondence to intended meaning

- Avoiding nonsense
 - ☐ E.g., part number field actually contains cost
- Avoiding fraud
 - ☐ E.g., sender's name on e-mail is changed to someone else's



18

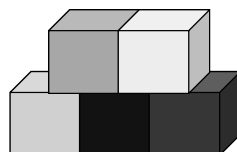
Copyright © 2010 M. E. Kabay. All rights reserved.

Availability



Timely access to data

- Avoid delays
 - ☐ E.g., prevent system crashes & arrange for recovery plans
- Avoid inconvenience
 - ☐ E.g., prevent mislabeling of files



19

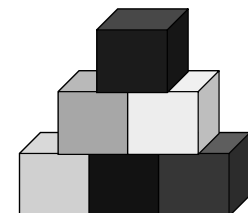
Copyright © 2010 M. E. Kabay. All rights reserved.

Utility



Usefulness for specific purposes

- Avoid conversion to less useful form
 - ☐ E.g., replacing dollar amounts by foreign currency equivalent
- Prevent impenetrable coding
 - ☐ E.g., employee encrypts source code and "forgets" decryption key



20

Copyright © 2010 M. E. Kabay. All rights reserved.

Identification, Authentication & Authorization



- Identification
 - ❑ Assigning specific, unique *identifier* to each user
- Authentication
 - ❑ Linkage between an identifier and the real-world entity (person, device) attempting to use the identifier
 - ❑ Four principles
 - ✓ What you know
 - ✓ What you have
 - ✓ What you are
 - ✓ What you do
- Authorization
 - ❑ Granting privileges to user depending on context
 - ❑ E.g., read-access to some records, write-access to fewer records

21

Copyright © 2010 M. E. Kabay. All rights reserved.

Security in DBMS



- User-specific views
 - ❑ *Vertical security*: limit *columns (fields)*
 - ❑ *Horizontal security*: limit *rows (records)*
- Implementations
 - ❑ Incorporate user-name or role in records
 - ❑ Use different forms, reports for different people
- *Class Discussion*
 - ❑ *How might a hospital application apply security restrictions to its DB?*

22

Copyright © 2010 M. E. Kabay. All rights reserved.

The Software Process



- Generic Software Process Models
- The System Development Life Cycle (SDLC)
- Evolutionary Development
- Reuse-oriented Development
- Incremental Development
- Incremental Development Advantages
- Extreme Programming

23

Copyright © 2010 M. E. Kabay. All rights reserved.

The Software Process



- A structured set of activities required to develop a software system
 - ❑ Specification;
 - ❑ Design;
 - ❑ Validation;
 - ❑ Evolution.

24

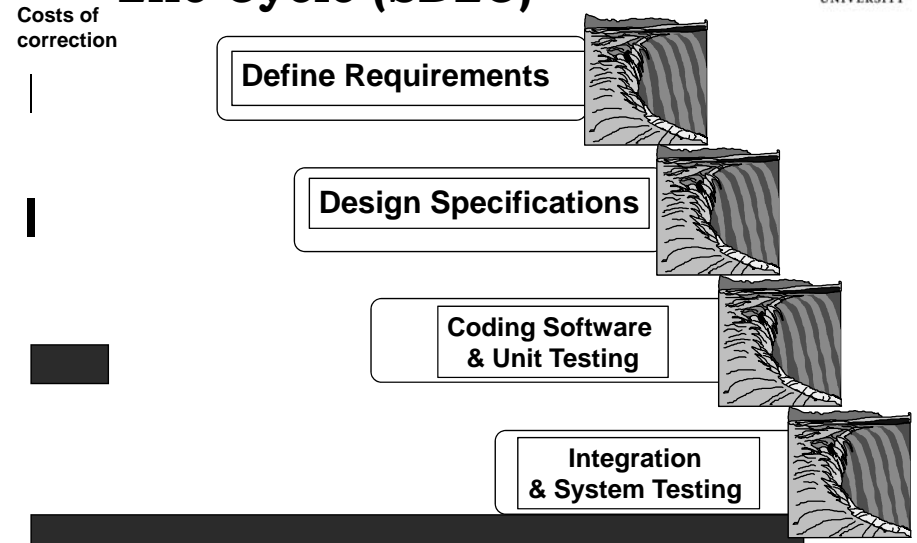
Copyright © 2010 M. E. Kabay. All rights reserved.

IS

Generic Software Process Models

- The waterfall model
 - ❑ Separate and distinct phases of specification and development.
- Evolutionary development
 - ❑ Specification, development and validation are interleaved.
- Component-based software engineering
 - ❑ The system is assembled from existing components.
- There are many variants of these models e.g. formal development where a waterfall-like process is used but the specification is a formal specification that is refined through several stages to an implementable design.

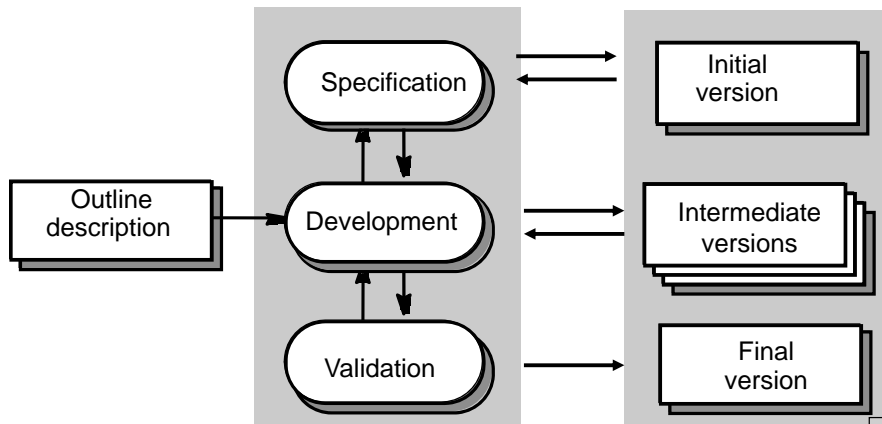
The System Development Life Cycle (SDLC)



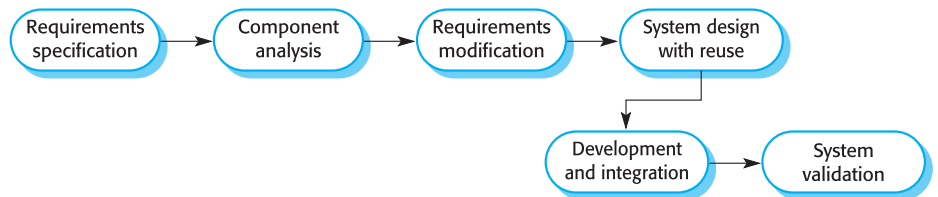
Evolutionary Development

RAD: Rapid Application Dev't
 JAD: Joint Application Dev't

Concurrent Activities

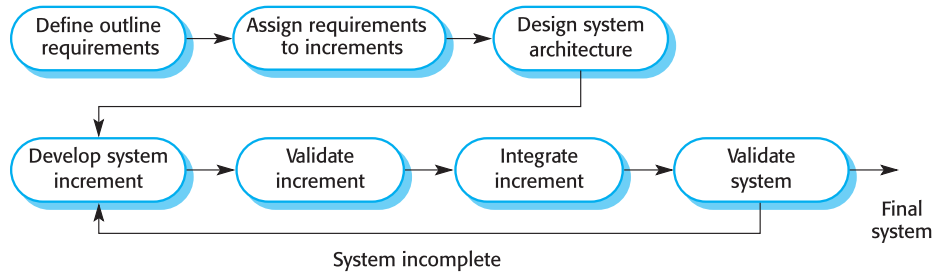


Reuse-oriented Development



Kindly overlook the odd formatting in the text boxes.

Incremental Development



Kindly overlook the odd formatting in the text boxes.

Incremental Development Advantages



- Customer value can be delivered with each increment so system functionality is available earlier.
- Early increments act as a prototype to help elicit requirements for later increments.
- Lower risk of overall project failure.
- The highest priority system services tend to receive the most testing.

Extreme Programming



- An approach to development based on the development and delivery of very small increments of functionality.
- Relies on constant code improvement, user involvement in the development team and pairwise programming.

End of Lecture 13a

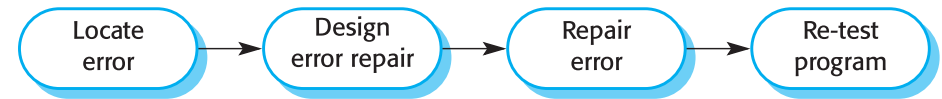


Quality Assurance



- The Debugging Process
- Software Validation
- Definitions & Orientation
- The Testing Process

The Debugging Process



Kindly overlook the odd formatting in the text boxes.

Software Validation



- Verification and validation (V & V) are intended to show that a system conforms to its specification and meets the requirements of the system customer.
- Involves checking and review processes and system testing.
- System testing involves executing the system with test cases that are derived from the specification of the real data to be processed by the system.

Definitions & Orientation

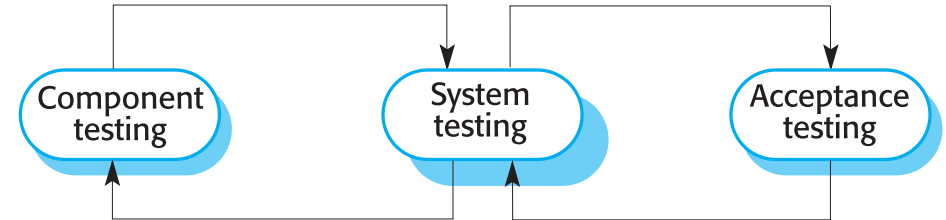


- Trying to *find* errors
- NOT trying to show there are *no* errors
- *Successful* test finds errors
- Problems of language and psychology

The Testing Process

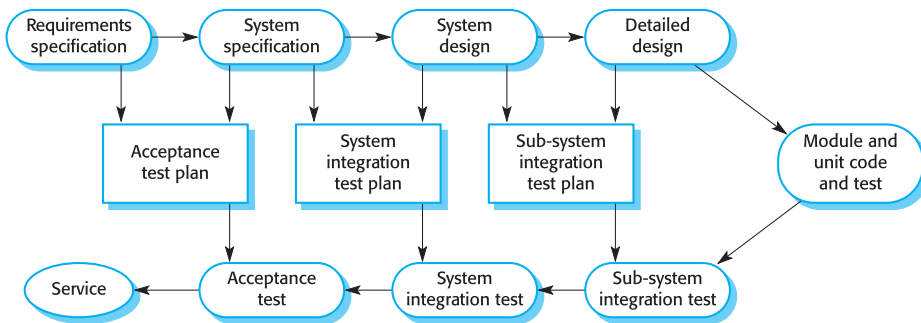
- Testing Phases
- Economics of Testing
- Some Principles of Testing
- Inspections / Walkthroughs
- Types of Testing
- The Case for Automated Testing

The Testing Process



We are trying to FIND ERRORS.
We are sorry there ARE errors but we are HAPPY to FIND errors.
We do not try to AVOID finding errors!

Testing Phases



Kindly overlook the odd formatting in the text boxes.

Economics of Testing

- Costs of errors
 - ❑ Roughly 10x increase at every level of development
 - ❑ Analysis, design, coding, implementation
- Costs of finding errors
 - ❑ Must balance cost of error vs cost of finding error
 - ❑ Possible test cases usually infinite
 - ❑ Impossible to locate all errors
 - ❑ Unnecessary to locate all errors: just significant ones

Economics of Testing



- **Black-Box Testing**
 - ❑ Derive test data from *specifications* only
 - ❑ Use exhaustive input testing
 - ❑ But include all possible wrong inputs too
 - ❑ Time and money constraints make it impossible to test everything
- **White-Box Testing**
 - ❑ Know how the code is *designed*
 - ❑ Try to execute all possible execution paths
 - ❑ But astronomically high # paths
 - ❑ And have to multiply by # of inputs

41

Copyright © 2010 M. E. Kabay. All rights reserved.

Some Principles of Testing



- Define expected values
- Use independent testers
- Pay attention to every result
- Include invalid and unusual inputs
- Look for forbidden results
- Record test cases for re-use
- Errors bespeak more errors
- 80/20 rule (Pareto Principle)

42

Copyright © 2010 M. E. Kabay. All rights reserved.

Inspections / Walkthroughs



- Human testing can be highly effective
 - ❑ Apply after analysis/design
 - ❑ Before coding
 - ❑ Catch errors early = cheaper & better correction
- BUT desk-checking by *individual* simply doesn't work well enough to catch enough errors
- Inspections – team approach
 - ❑ Finds 30%-70% errors
 - ❑ Programmer explains every line of code (~150 lines 3GL/hour)
- Walkthroughs – play computer (think about every instruction) – increase number of catches

43

Copyright © 2010 M. E. Kabay. All rights reserved.

Types of Testing



- Module / Unit
- Integration Testing
- Function Testing
- System Testing
- Acceptance Testing
- Installation Testing

44

Copyright © 2010 M. E. Kabay. All rights reserved.

Testing Modules Alone



How can we execute a subroutine by itself?

- A *driver* program
 - ❑ Calls a module and
 - ❑ Passes parameters to it
- A *stub* program
 - ❑ Represents an as-yet missing module
 - ❑ Not simply a place-holder
 - ❑ Must receive data from calling module
 - ❑ Must return valid values to calling module

45

Copyright © 2010 M. E. Kabay. All rights reserved.

System Testing



- Facility Testing
- Stress Testing
- Volume Testing
- Usability Testing
- Security Testing
- Performance Testing
- Storage Testing
- Configuration Testing
- Compatibility / Conversion Testing
- Installability Testing
- Reliability Testing
- Recovery Testing
- Serviceability Testing
- Documentation Testing
- Procedure Testing

46

Copyright © 2010 M. E. Kabay. All rights reserved.

Types of Errors



- User I/F
- Error handling
- Boundary-related errors
- Calculation errors
- Initial and later states
- Control-flow errors
- Errors in handling or interpreting data
- Race conditions
- Load conditions
- Hardware
- Source, version and I/D control
- Errors in the testing process

47

Copyright © 2010 M. E. Kabay. All rights reserved.

Designing Good Tests



- Design Philosophy
- Boundary Analysis
- Testing State Transitions
- Testing Race Conditions And Other Time Dependencies
- Function-equivalence Testing
- Regression Testing

48

Copyright © 2010 M. E. Kabay. All rights reserved.

Test-Case Design Philosophy



- Always work to FIND errors
- Complete testing is impossible
- Intuitive approach is “random-input testing”
 - ❑ Sit at terminal
 - ❑ Invent test data at random
 - ❑ See what happens
 - ❑ *Worst possible approach*
- Therefore define subset of test cases likely to detect most (or at least many) errors
 - ❑ Use well-established principles for designing test cases

49

Copyright © 2010 M. E. Kabay. All rights reserved.

Boundary-Value Analysis



- Cases at boundaries have high value for testing
- Select cases just below, at and just above limits of each equivalency class
- Some testers include mid-range value as well just for additional power of test

50

Copyright © 2010 M. E. Kabay. All rights reserved.

Testing State Transitions



- Every change in output is a state transition
- Test every option in every menu
- If possible, test every pathway to every option in every menu
- Interactions among paths
 - ❑ Draw menu maps
 - ❑ Identify multiple ways of reaching every state
 - ❑ Keep careful records of what you test (can get confusing)

51

Copyright © 2010 M. E. Kabay. All rights reserved.

Testing Race Conditions and Other Time Dependencies



- Check different speeds of input
- Try to disrupt state transitions (e.g, press keys while program switches menus)
- Challenge program just before and just after time-out periods
- Apply heavy load to cause failures (not just poor performance)

52

Copyright © 2010 M. E. Kabay. All rights reserved.

Function-Equivalence Testing



- Use a program that produces known-good output
- Feed same inputs to both the standard program and the program under test
- Compare the outputs
- Automated testing techniques can help
 - ❑ For numerical and alphanumeric output
 - ❑ For real-time process-control applications

53

Copyright © 2010 M. E. Kabay. All rights reserved.

Regression Testing



- Did the bug get fixed?
 - ❑ Some programmers patch symptom
 - ❑ Few test effectively
- Check that you can produce bug at will in bad version of code
- Use same tests on revised code
 - ❑ Stop if bug reappears
 - ❑ Push the testing if bug seems to have been fixed

54

Copyright © 2010 M. E. Kabay. All rights reserved.

The Case for Automated Testing



- Current Testing Methods are Inadequate
- Consequences of Manual Testing Methods
- Automated Testing
- Benefits of Structured Automated Testing
- Case Study: COGNOS / Ottawa, Canada

55

Copyright © 2010 M. E. Kabay. All rights reserved.

Current Testing Methods are Inadequate



- Manual input
- Unstructured
- Slow
- Depend on testers' awareness and attention
- Leave no audit trail
- Poor or no statistics
- Manual demonstration of errors

56

Copyright © 2010 M. E. Kabay. All rights reserved.

Consequences of Manual Testing Methods



- Quality is not emphasized during SDLC
- Time pressures always squeeze testing
- Testing never catches all the bugs

57

Copyright © 2010 M. E. Kabay. All rights reserved.

Automated Testing



- Capture/Playback
 - ❑ record macros showing mouse movements and alphanumeric input
 - ❑ typically no editing language
- Structured Automated Testing
 - ❑ tool creates structured, editable script
 - ❑ can use databases as source of input
 - ❑ intelligent handling of errors

58

Copyright © 2010 M. E. Kabay. All rights reserved.

Benefits of Structured Automated Testing



- Consistent, reproducible testing
- Increased test coverage
- Easier maintenance
- Fully documented testing
- Higher-quality software

59

Copyright © 2010 M. E. Kabay. All rights reserved.

Case Study: COGNOS / Ottawa, Canada



Using manual testing:

- 6 people
- 3 test phases per product release
- 3,000 manual tests per phase
- 12.5% test coverage
- 15 days for testing
- \$81,000 per release @ 12.5% coverage
- (\$648,000 per release @ 100%)

Using automated testing:

- 6 people
- 3 test phases
- 24,000 tests/phase
- 100% test coverage
- 5 days elapsed time
- \$81,000 per release @ ~100% coverage

60

Copyright © 2010 M. E. Kabay. All rights reserved.

Review Questions for Quizzes and Exams



1. Why do we need to write DBMS applications at all? Why not just use the query systems and report writers packaged with the DBMS?
2. Why would one choose to enforce constraints on a database system by (a) incorporating them into the metadata or (b) writing them into program code?
3. Why is security important in developing database applications?
4. What are the six fundamental attributes of information that we protect with information security?
5. How do we use user-specific views to enforce security in DBMS applications?
6. In software development, what is the SDLC?

61

Copyright © 2010 M. E. Kabay. All rights reserved.

Review Questions (2)



7. What is the Waterfall Model?
8. Contrast the SDLC with evolutionary development models.
9. What do the acronyms RAD and JAD mean in the context of software development?
10. What distinguishes reuse-oriented development from other models of software development?
11. What are the advantages of incremental development methodologies over the SDLC?
12. What is extreme programming?
13. What are the three fundamental elements of the testing process?
14. Why is it economically important to catch errors early in system development?

62

Copyright © 2010 M. E. Kabay. All rights reserved.

Review Questions (3)



15. Distinguish between black-box testing and white-box testing.
16. What are some fundamental principles of software testing?
17. Describe the differences between a code inspection and a walkthrough.
18. Distinguish among the types of testing (module/unit vs integration etc).
19. What is a driver? What is a stub? How are they used in quality assurance?
20. How do we use boundary-value analysis in software testing?
21. How do we use state transitions in software testing?

63

Copyright © 2010 M. E. Kabay. All rights reserved.

Review Questions (4)



22. What are *race conditions* and how do we test for them?
23. What is function-equivalence testing?
24. What is regression testing and why is it important?
25. Why is automated testing important in quality assurance?

64

Copyright © 2010 M. E. Kabay. All rights reserved.