
Architecture & Design of Automated Software Tests

PNSQC Workshop, Spring 2000

Douglas Hoffman
Software Quality Methods, LLC.
24646 Heather Heights Place
Saratoga, California 95070-9710
Phone 408-741-4830
Fax 408-867-4550
doug.hoffman@acm.org

Copyright © 2000, Software Quality Methods, LLC. No part of these graphic overhead slides may be reproduced, or used in any form by any electronic or mechanical duplication, or stored in a computer system, without written permission of the author.

Acknowledgment

- Many of the ideas in this presentation were presented and refined in Los Altos Workshops on Software Testing.
- LAWST 1-3 focused on several aspects of automated testing. Participants were Chris Agruss, Tom Arnold, Richard Bender, James Bach, Jim Brooks, Karla Fisher, Chip Groder, Elizabeth Hendrickson, Doug Hoffman, Keith W. Hooper, III, Bob Johnson, Cem Kaner, Brian Lawrence, Tom Lindemuth, Brian Marick, Thanga Meenakshi, Noel Nyman, Jeffery E. Payne, Bret Pettichord, Drew Pritsker, Johanna Rothman, Jane Stepak, Melora Svoboda, Jeremy White, and Rodney Wilson.
- LAWST 5 focused on oracles. Participants were Chris Agruss, James Bach, Jack Falk, David Gelperin, Elisabeth Hendrickson, Doug Hoffman, Bob Johnson, Cem Kaner, Brian Lawrence, Noel Nyman, Jeff Payne, Johanna Rothman, Melora Svoboda, Loretta Suzuki, and Ned Young.

Workshop Topics

- Why Automate Software Tests?
- Automated Test Design
- Test Automation Strategies
- Automated Test Oracles
- Automation Architectures

Workshop Objectives

- Describe the major factors in software test automation
- Identify strategies for automated software testing
- Explore automated testing techniques
- Show how to design an automation architecture

What Is Software Architecture?

“As the size and complexity of software systems increase, the design and specification overall system structure become more significant issues than the choice of algorithms and data structures of computation. Structural issues include the organization of a system as a composition of components; global control structures; the protocols for communication, synchronization, and data access; the assignment of functionality to design elements; the composition of design elements; physical distribution; scaling and performance; dimensions of evolution; and selection among design alternatives. This is the *software architecture* level of design.”

“Abstractly, software architecture involves the description of elements from which systems are built, interactions among those elements, patterns that guide their composition, and constraints on these patterns. In general, a particular system is defined in terms of a collection of components and interactions among those components. Such a system may in turn be used as a (composite) element in a larger design system.”

—*Software Architecture*, M. Shaw & D. Garlan, 1996, p.1.

What Is Software Architecture?

“The quality of the architecture determines the conceptual integrity of the system. That in turn determines the ultimate quality of the system. Good architecture makes construction easy. Bad architecture makes construction almost impossible.”

— Steve McConnell, *Code Complete*, p 35; see 35-45

“We’ve already covered some of the most important principles associated with the design of good architectures: coupling, cohesion, and complexity. But what really goes into making an architecture good? The essential activity of architectural design . . . is the partitioning of work into identifiable components. . . Suppose you are asked to build a software system for an airline to perform flight scheduling, route management, and reservations. What kind of architecture might be appropriate? The most important architectural decision is to separate the business domain objects from all other portions of the system. Quite specifically, a business object should not know (or care) how it will be visually (or otherwise) represented . . .”

— Luke Hohmann, *Journey of the Software Professional: A Sociology of Software Development*, 1997, p. 313. See 312-349

What is Automation Design?

- Selecting automation components
- Setting relationships between components
- Identifying locations of components and events
- Sequencing test events

What Are Automated Software Tests?

- No intervention needed after launching tests
- Automatically sets-up and/or records relevant test environment
- Runs test exercise
- Captures relevant results
- Evaluates actual against expected results
- Reports analysis of pass/fail

Capabilities of Automation Tools

- Automated test tools combine a variety of capabilities. For example, GUI test tools provide:
 - capture/replay for easy manual creation of tests
 - execution of test scripts
 - recording of test events
 - compare the test results with expected results
 - report test results
- Some GUI tools provide additional capabilities, but no tool does everything well.

Capabilities of Automation Tools

- | | |
|---|---|
| • Analyze source code | • Select tests to be run |
| • Design test cases | • Execute test scripts |
| • Create test cases (from requirements or code) | • Record test events |
| • Generate test data | • Measure software responses to tests (Discovery Functions) |
| • Easy manual creation of test cases | • Determine expected results of tests (Reference Functions) |
| • Easy creation/management of traceability matrix | • Evaluate test results (Evaluation Functions) |
| • Manage testware environment | • Report and analyze results |

Workshop Topics

- Why Automate Software Tests?
- Automated Test Design
- Test Automation Strategies
- Automated Test Oracles
- Automation Architectures

Why Automate Software Tests?

- Short term ROI
- Tests that require automation
- Long term ROI



Manual Software Tests

- Person initiates each test case
- Person interacts with the test, SUT, or the environment during the test run
- Person is required to act to analyze test results
- Person summarizes and reports results

Levels of Automation

- Fully automated software testing
- Semi-automated software testing
- Manual software testing

Key Automation Factors

- Components of SUT
- Important features and capabilities
- SUT environments
- Testware elements
- Access to inputs and results
- Form of inputs and results

A Typical Automated Test

- What is being tested
- How is the test set up
- Where are the inputs coming from
- What is being checked
- Where are the expected results
- How do you know pass or fail

Exercise

Why are you automating tests?

Workshop Topics

- Why Automate Software Tests?
- **Automated Test Design**
- Test Automation Strategies
- Automated Test Oracles
- Automation Architectures

An Excellent Test Case

- Reasonable probability of catching an error
- Not redundant with other tests
- Exercise to stress the area of interest
- Minimal use of other areas
- Neither too simple nor too complex
- Makes failures obvious
- Allows isolation and identification of errors

Designing Automated Tests

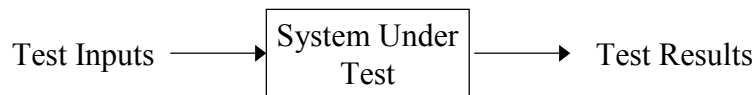
- Start with known states
- Design variation into the tests
- Check for errors
- Put your analysis into the tests
- Capture information when an error is found
- Don't encourage error cascades or masking

Black Box Testing Paradigms*

- Domain driven
- Stress driven
- Specification driven
- Risk driven
- Random / statistical
- Function
- Regression
- Scenario / use case / transaction flow
- User testing
- Exploratory

* James Bach and Cem Kaner, *Black Box Testing Paradigms*, USPDI and STAR 1999.

Simple Testing Model (Black Box)



Implications of the Simple Model

- We control the inputs
- We can verify results
- We aren't dealing with all the factors
 - Memory and data
 - Program state
 - System environment

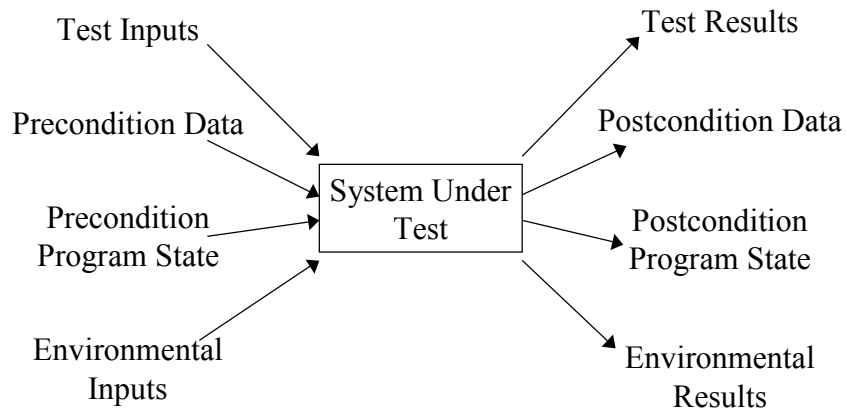
Size Of The Testing Problem

- Input one value in a 10 character field
- 26 UC, 26 LC, 10 Numbers
- Gives 62^{10} combinations
- How long at 1,000,000 per second?

What is your domain size?

*We can only run a
vanishingly small portion
of the possible tests*

Expanded Testing Model (Black Box)



Implications of the Expanded Model

- We don't control all inputs
- We don't verify everything
- Multiple domains are involved
- Test exercise may be the easy part
- We can't verify everything
- We don't know all the factors

Size Of The Testing Problem

- Input one value in a 10 character field
- How many data values in the program?
- How many program states?
- What system environment characteristics?

What is your input/result size?

Choosing The Subset

- High value tests
- Exploratory testing
- Automated tests
- More powerful exercises

Six Successful [Test] Architectures*

- Quick & dirty
- Equivalence testing
- Framework
- Data-driven
- Application-independent data-driven
- Real-time simulator with event logs

*Kaner, Cem; *Avoiding Shelfware: A Manager's View of Automated GUI Testing*, 1998. See www.kaner.com

Quick & Dirty

- Smoke tests
- Configuration tests
- Variations on a theme
- Stress, load, or life testing

Equivalence Testing

- A/B comparison
- Random tests using an oracle
- Regression testing is the weakest form

Framework

- Code libraries
- Components are reused
- Separation of test case from interface details

Data-Driven Tests

- Variables are data
- Commands are data
- UI is data
- Program's state is data

Application-Independent Data-Driven

- Generic tables of repetitive types
- Rows for instances
- Automation of exercises

Real-time Simulator

- Test embodies rules for activities
- Random walk approach
- Possible monitors
 - Code assertions
 - Event logs
 - Oracles

Exercise

Describe your typical automated test

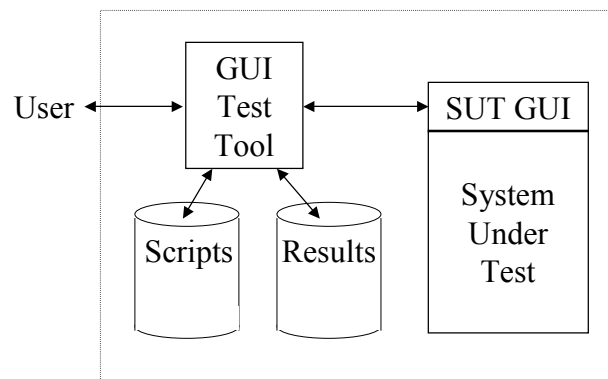
Workshop Topics

- Automated Software Testing
- Automated Test Design
- **Test Automation Strategies**
- Automated Test Oracles
- Automation Architectures

Regression Strategy

- Write the test exercise
- Run it
- Capture the results
- “Verify” the results
- Run it again as a regression test
- Compare current with captured results

A Regression Test Model



Regression Advantages

- Straightforward
- Same approach for all tests
- Fast implementation
- Variations easy
- Repeatable tests

Regression Disadvantages

- “20 Questions”
- Master validation
- Analysis of failures
- Limited scope

There's A Universe Beyond Regression Automation

Douglas HoffmanCopyright © 2000, SQM, LLC.43

Strategies for Automation

<p>Characteristics</p> <ul style="list-style-type: none">- goal of testing- software under test- environment- test generator- reference function- evaluation function- access to internals- users- risks	<p>That support, counter-indicate, or drive you toward a strategy</p> <ul style="list-style-type: none">- consistency evaluation- small sample, pre-specified values- exhaustive sample- random (aka statistical)- heuristic analysis of a large set
---	---

Douglas HoffmanCopyright © 2000, SQM, LLC.44

Strategies for Automation

- Consistency evaluation (regression)
- Small sample
- Exhaustive sample
- Random (aka statistical)
- Heuristic analysis of a large set

Evaluation: Consistency

- Advantages
 - The program can serve as its own oracle
 - Easily used at an API
 - Effective when test cases are very expensive or when the software design is very stable
- Disadvantages
 - Every time the software changes, tests change
 - High maintenance cost likely
 - Common mode of failure errors won't be detected
 - Legacy errors won't be detected

Small Sample

- Limited number of tests
- Pre-specified values
- With an oracle
- Computed as needed

Examples: Small Sample

- Equivalence and boundary analysis
- Scenario tests
- Exploratory test or a one-use scenario test for computer language

Evaluation: Small Sample

- Advantages
 - Can be fast
 - Automation can be customized
- Disadvantages
 - It only checks a few values (we don't know anything about the rest)
 - Doesn't necessarily consider specific, key data values
 - False security if domains not correctly analyzed

Exhaustive Sample

- Using all values within a given domain, such as:
 - all valid inputs to a function
 - compatibility tests across all relevant equipment configurations
- Oracle based
- Consistency based

Example: Exhaustive Testing

- MASPAR functions, square root tests

Evaluation: Exhaustive

- Advantages
 - Complete management of certain risks
 - Discover special case failures that are not visible at boundaries
- Disadvantages
 - Expensive
 - Often impossible

Random (aka Statistical) Testing

- Function Equivalence Testing
- Complex simulations with long series of events or combinations
- Random transition from state to state
 - Check whether the program has actually reached the expected state
 - Check for assertion fails or other debug warning messages
- Statistical testing
- Generate many tests (e.g. clean room)

Example: Random Testing

- Testing the math functions in excel
- Random walks
- Life, load, and stress tests
- Statistical characteristics

Evaluation: Random

- Advantages
 - Can run a huge number of test cases
 - Few or no evaluation errors
- Disadvantages
 - Doesn't consider specific, key data values
- Risks
 - People sometimes underestimate the need for an oracle
 - Risk of false negatives and common mode errors
 - Risk of overestimating coverage

Heuristic Analysis

- Heuristics are rules of thumb that support but do not mandate a given conclusion
- We have partial information that will support a probabilistic evaluation
- It won't tell you that the program works correctly but it can tell you that the program is broken
- This can be a cheap way to spot errors early in testing

Heuristics

- Possible false alarms and misses
- Check (apparently) insufficiently complete attributes
- Check probabilistic attributes
- Compare incidental but correlated attributes
- Consistent relationships

Examples: Heuristic

- Possible false alarms and misses
- Check (apparently) insufficient attributes
 - ZIP Code entries are 5 or 9 digits
- Check probabilistic attributes
 - X is usually greater than Y
- Sine function
 - sawtooth wave
 - $\sin(x)^2 + \cos(x)^2 = 1$

Examples: Heuristic

- Data communications
 - value patterns (start, increment, number of values)
 - CRC
- Data base
 - selected records using specific criteria
 - selected characteristics for known records
 - standard characteristics for new records
 - correlated field values (time, order number)
 - timing of functions

Heuristic Oracle Relationships

- Nice
 - follows heuristic rule for some range of values
 - ranges are knowable
 - few or no gaps
- Predictable
 - identifiable patterns
- Simple
 - easy to compute or identify
 - requires little information as input

Choosing A Heuristic

- Rules of thumb
 - similar results that don't always work
 - low expected number of false errors, misses
- Simplify
 - use subsets
 - break down into ranges
 - step back (20,000 or 100,000 feet)
 - look for harmonic patterns
- Other relationships not explicit in SUT
 - date/transaction
 - one home address
 - employee start date

Evaluation: Heuristic

Advantages

- May allow exhaustive testing of values over domain
- Allows discovery of problems early in testing
- Heuristic oracles are often reusable
- Handy, powerful for early detection

Heuristic Oracle Disadvantages

- Inexact
 - will miss specific classes of errors
 - may miss gross systematic errors
 - might not cover entire input/result domains
- May generate false errors
- Can become too complex
 - exception handling
 - too many ranges
 - require too much precision
- Application may need better verification

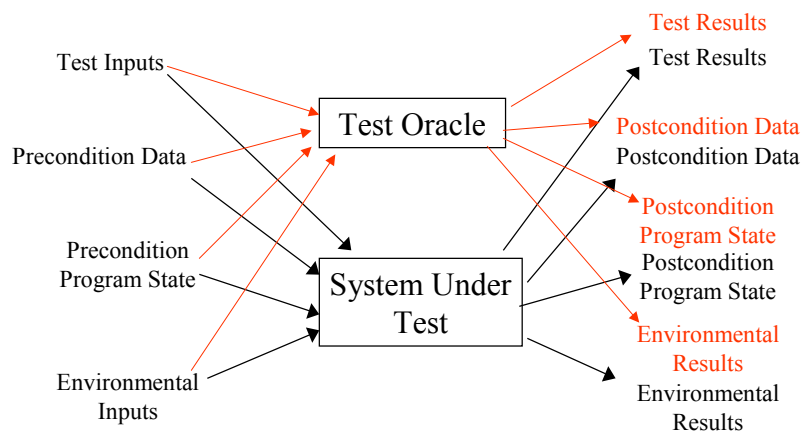
Workshop Topics

- Automated Software Testing
- Automated Test Design
- Test Automation Strategies
- Automated Test Oracles
- Automation Architectures

Where Do We Fit In The Oracle?

- Identify what to verify
- How do we know the “right answer”
- How close to “right” do we need
- Decide when to generate the expected results
- Decide how and where to verify results
- Get or build an oracle

Testing With An Oracle



Oracle Characteristics

- Completeness of information from an oracle
- Accuracy of information from an oracle
- Usability of results
- Temporal relationships
- Supportability

Completeness

- Sufficiency
- Input Coverage
- Result Coverage
- SUT environments
- Types of errors possible

Accuracy

- How similar
 - Arithmetic accuracy
 - Statistically similar
- How independent
- Types of possible errors

Usability

- Form of information
- Location of information
- Availability of comparators
- Support in SUT environments
- Cost

Temporal Relationships

- How fast to generate results
- How fast to compare
- When is it run
- When are results compared

Supportability

- COTS or custom
- Correspondence through SUT changes
 - Test exercises
 - Tools
- Ancillary support activities required

Oracle Approaches

	True Oracle	Heuristic Strategy	Consistency	Self Referential	No Oracle
Definition	-Independent generation of all expected results	-Verifies some values, as well as consistency of remaining values	-Verifies current run results with a previous run (Regression Test)	-Embeds answer within data in the messages	-Doesn't check correctness of results, (only that some results were produced)
Advantages	-No encountered errors go undetected	-Faster and easier than True Oracle -Much less expensive to create and use	-Fastest method using an oracle -Verification is straightforward -Can generate and verify large amounts of data	-Allows extensive post-test analysis -Verification is based on message contents -Can generate and verify large amounts of complex data	-Can run any amount of data (limited only by the time the SUT takes)
Disadvantages	-Expensive to implement -Complex and often time-consuming when run	-Can miss systematic errors (as in <i>sine</i> wave example)	-Original run may include undetected errors	-Must define answers and generate messages to contain them	-Only spectacular failures are noticed.

True Oracle

- Independent implementation
- Complete coverage over domains
 - Input ranges
 - Result ranges
- “Correct” results
- Usually expensive

Heuristic Strategy

- Rules of thumb
 - Estimates
 - Approximations
 - Trends
- Levels of abstraction
 - General characteristics
 - Statistical properties

Consistency Strategy

- A / B compare
- Check for changes
- Regression checks
 - Validated
 - Unvalidated
- Alternate versions or platforms
- Foreign implementations

Self-Referential Strategy

- Embed results in the data
- Cyclic algorithms
- Shared keys with algorithms

Self-Referential Oracle Examples

- Data base
 - embedded linkages
- Data communications
 - value patterns (start, increment, number of values)
- Noel Nyman's "Self Verifying Data"*

* "Self Verifying Data - Validating Test Results
Without An Oracle," STAR East 1999

'No Oracle' Strategy

- Easy to implement
- Tests run fast
- Only spectacular errors are noticed
- False sense of accomplishment

Choosing Which Strategy

- Decide how the oracle fits in
- Identify the oracle characteristics
- Prioritize testing risks
- Choose a combination of approaches

More Powerful Exercises

- Increasing the number of combinations
- Self-verifying tests and diagnostics
- More frequency, intensity, duration
- Increasing the variety in exercises

Pseudo Random Numbers

- Used for selection or construction of inputs
 - With and without weighting factors
 - Selection with and without replacement
- Statistically “random” sequence
- Randomly generated “seed” value
- Requires oracles to be useful

Random Selection Among Alternatives

- Pseudo random numbers
- Partial domain coverage
- Small number of combinations
- Use an oracle for verification

Mutating Automated Tests

- Closely tied to instrumentation and oracles
- Using pseudo random numbers
- Positive and negative cases possible
- Diagnostic drill down on error

Mutating Tests Examples

- Data base contents (Embedded)
- Processor instruction sets (Consistency)
- Compiler language syntax (True)
- Stacking of data objects (None)

Exercise

Which types of oracles should you use?

Workshop Topics

- Automated Software Testing
- Automated Test Design
- Test Automation Strategies
- Automated Test Oracles
- **Automation Architectures**

What is Automation Design?

- Automation components
- Relationships between components
- Locations of components and events
- Sequences of events

Key Automation Factors

- Components of SUT
- Important features and capabilities
- SUT environments
- Testware elements
- Access to inputs and results
- Form of inputs and results

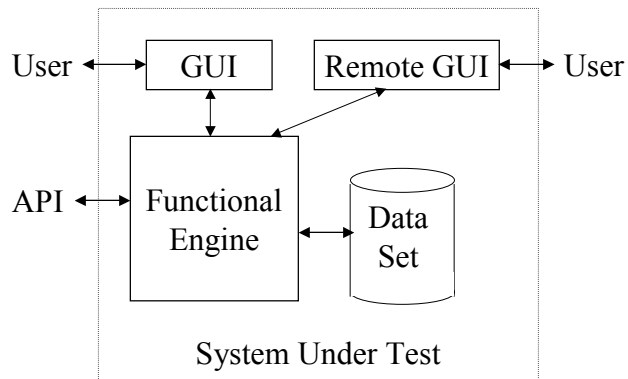
Automation Process

- List the sequence of automated events
- Identify elements of each event
- Decide on location(s) of events
- Determine flow control mechanisms
- Design automation mechanisms

Automation Architecture

- Model for SUT and environment
- Break down software testing problem
- Decide on location(s) of automation
- Decide on level(s) of automation
- Describe automation architecture

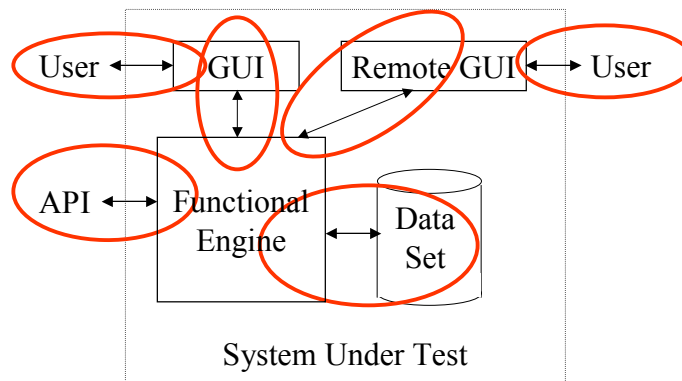
A Model For SUT



Location and Level for Automated Testing

- Availability of inputs and results
- Ease of automation
- Stability of SUT
- Practicality of oracle creation and use
- Priorities for testing

Break Down The Testing Problem



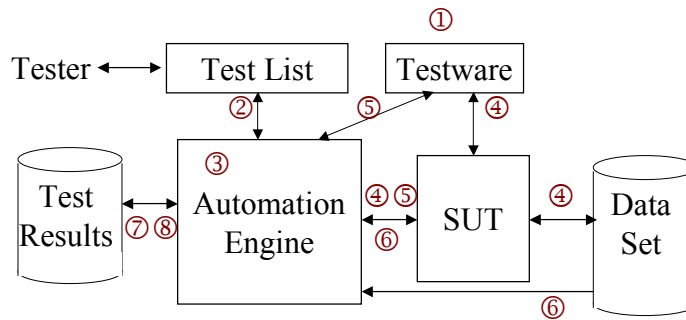
Identify Where To Intervene

- Natural break points
- Ease of automation
- Availability of oracles
- Leverage of tools and libraries
- Expertise

Automated Test Sequencing

1. Testware creation, version control, and configuration management
2. Selecting the subset of test cases to run
3. Set-up and/or record environmental variables
4. Run the test exercises
5. Monitor test activities
6. Capture relevant results
7. Compare actual with expected results
8. Report analysis of pass/fail

An Automated Software Testing Process Model



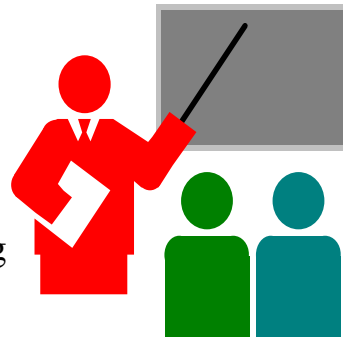
Douglas Hoffman

Copyright © 2000, SQM, LLC.

97

Summary

- Covered major factors in software test automation
- Identified strategies for automated software testing
- Explored automated testing techniques
- Showed how to design an automation architecture



Douglas Hoffman

Copyright © 2000, SQM, LLC.

98