

# Impact of Sequence-Based Specification on Statistical Software Testing

Stacy Prowell

**Q-Labs<sup>®</sup>**

5516 Lonas Rd, Suite 110, Knoxville, TN 37909, USA  
<http://www.q-labs.com/>  
Email: [Stacy.Prowell@Q-Labs.com](mailto:Stacy.Prowell@Q-Labs.com)

## Primary Benefits

Sequence-based specification contributes to the understanding of:

- Precise test boundary
- Complete input domain
- Valid input sequencing
- Intended software function

Sequence-based specification also helps enforce an external, user-centered view of software function.

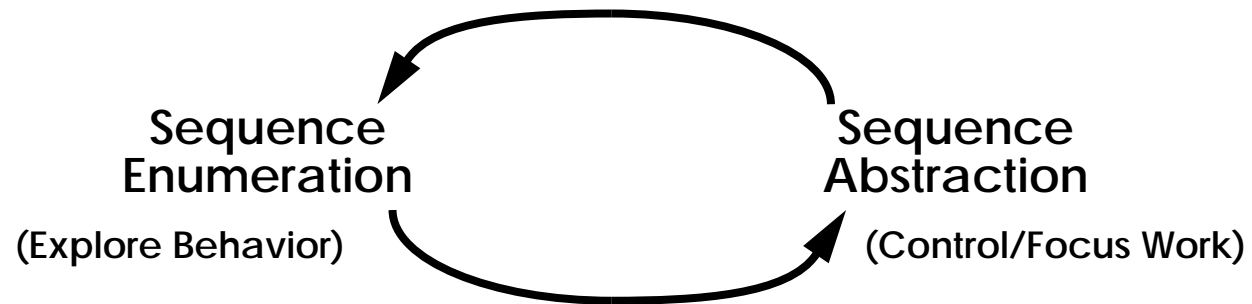
## Sequence-Based Specification

Sequence-based specification is a black box specification method.

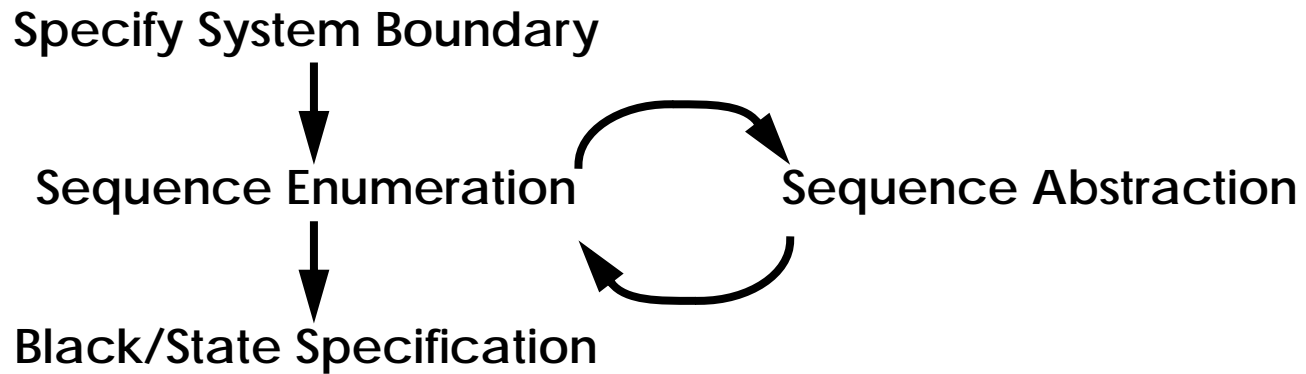


The specification gives an external, implementation-independent "user's view" of software function.

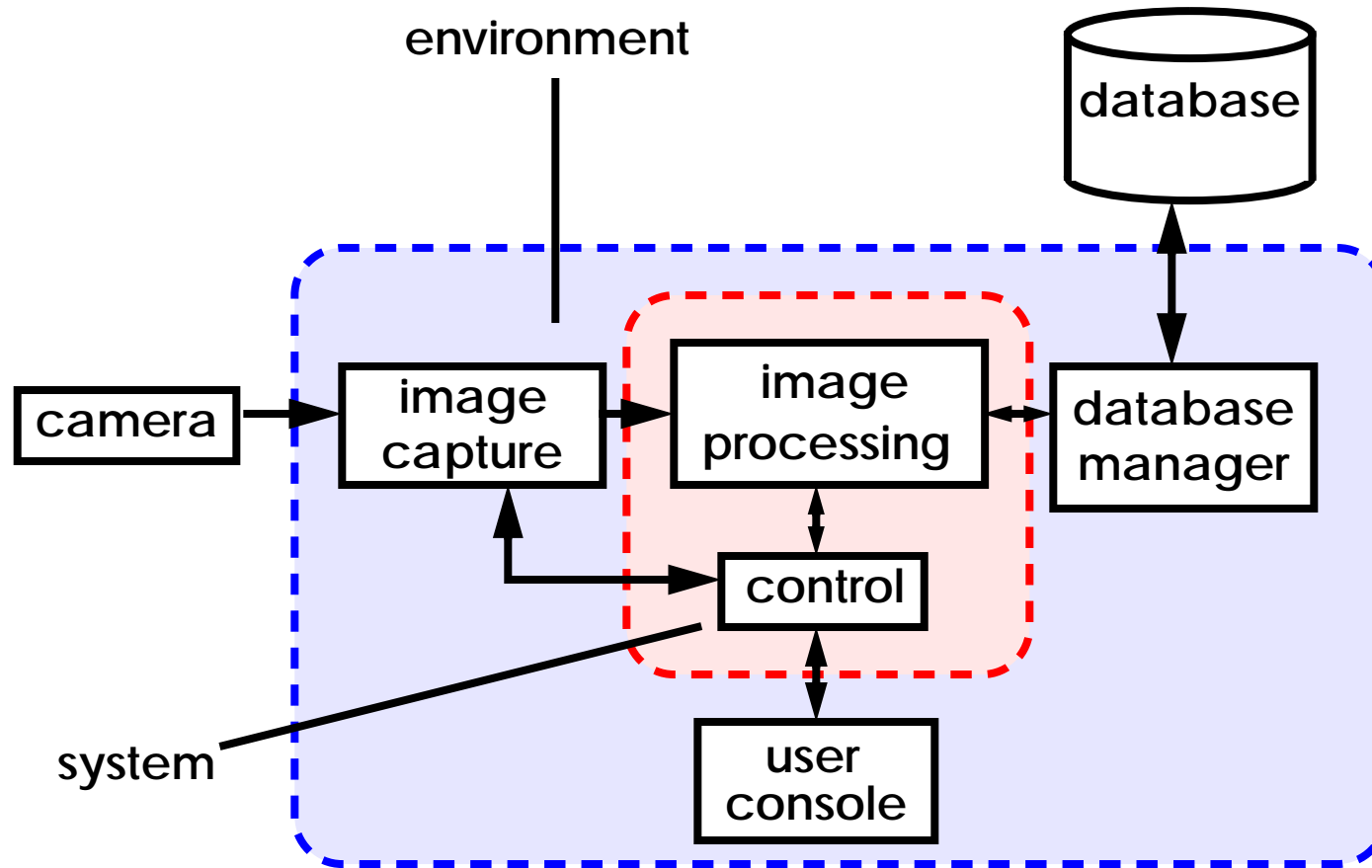
## Primary Techniques



## Sequence-Based Specification Process



## Specify Software Interfaces



## Black Box Specification

### Issues

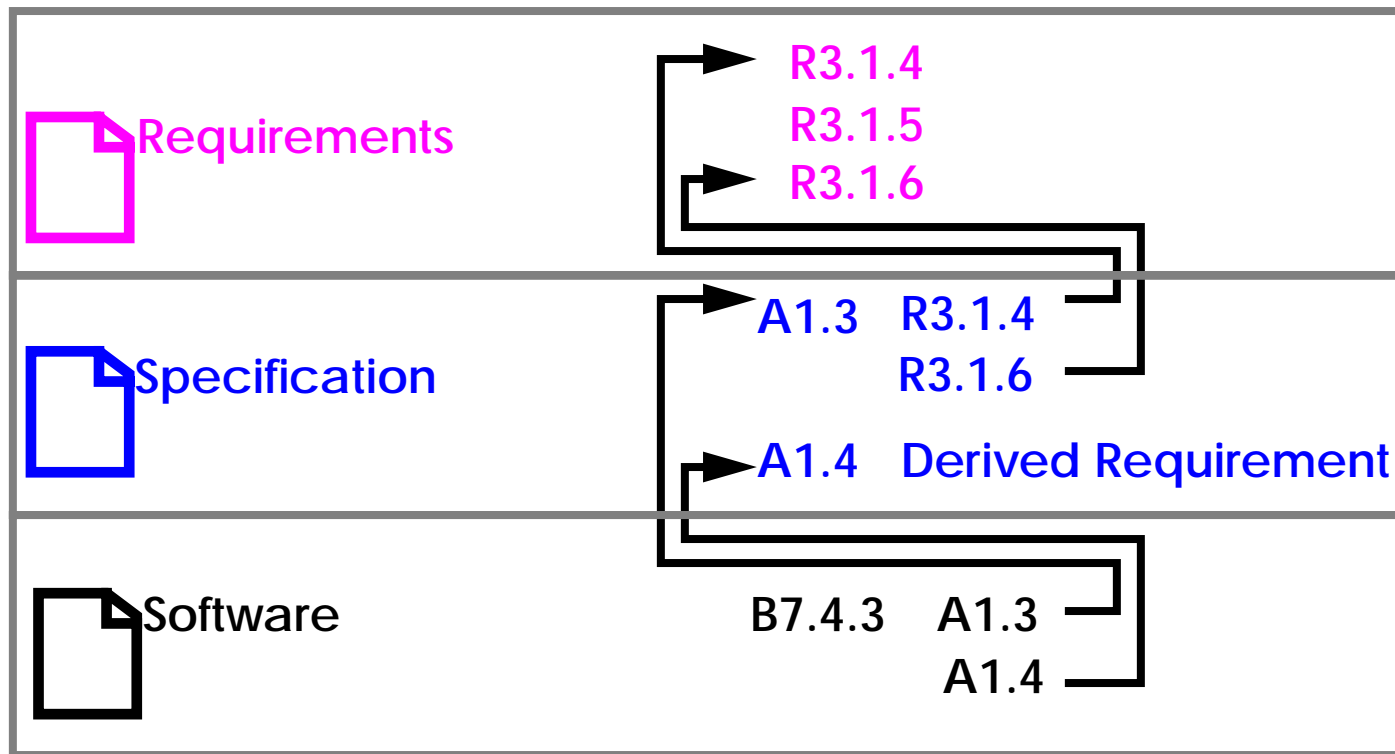
- 1 Software need not generate a response for every stimulus.  
clock pulses
- 2 Some sequences may violate the definition of the system or environment, or may violate known design assumptions.  
stimuli before invocation  
events from deactivated hardware

### Solution

$R$  contains two special values: 0 and  $\omega$ , the **null response** and **illegal**, respectively. These two values allow the black box to be a total function.

## Requirements Traceability

The **correctness** of the black box function must be shown through **requirements traceability**.



## Sequence Enumeration

**Basic Idea:** Enumerate stimulus histories in order by length. For each history, give the response.

$\lambda$	0
T	t
F	$\omega$
L	$\omega$
TT	$\omega$
TF	f
TL	r
FT	$\omega$
FF	$\omega$
FL	$\omega$
LT	$\omega$
LF	$\omega$
LL	$\omega$

## Illegal Prefix

Whenever a sequence has an illegal prefix, the sequence is illegal. Do not expand illegal sequences in the enumeration.

$\lambda$	0
T	t
F	$\omega$
L	$\omega$
TT	$\omega$
TF	f
TL	r
TFT	$\omega$
TFF	f
TFL	a
TLT	t
TLF	$\omega$
TLL	$\omega$

## Equivalent Sequences

Note that the sequences  $\lambda$  and TL both leave the plane in the initial condition. Thus the future behavior of the system is independent of which of these sequences occurred.

We say two sequences  $u$  and  $v$  are **equivalent** if and only if, when extended by non-empty histories, the responses always match.

$u \equiv v$  if and only if  $\forall (w \in S^*), w \neq \lambda \Rightarrow \text{BB}(uw) = \text{BB}(vw)$

Basic Idea: When enumerating, note equivalences to previous sequences in the enumeration, as with  $\lambda \equiv \text{TL}$ . We say TL is **reduced** to  $\lambda$ .

Since the sequences agree on future behavior, only extend one. Do not extend sequences which have been reduced.

## Complete Enumeration

Noting reductions in the enumeration, we obtain the following.

$\lambda$	0
T	t
F	$\omega$
L	$\omega$
TT	$\omega$
TF	f
TL	$r / \equiv \lambda$
TFT	$\omega$
TFF	$f / \equiv TF$
TFL	$a / \equiv \lambda$

There are no sequences left to extend; the enumeration is **complete**, and gives a response for any sequence in  $S^*$ .

## Example Enumeration

Sequence	Response	Equivalence	Trace	Notes
IN MG HR IN	INA	IN	3.5	
IN MG HR HR	ERR	IN MG HR	4.3.5, 6.1, 6.4	4.3.5 If the SOS receives a command (other than IN) from the GCS during processing of a previous command, a protocol error shall be generated and processing of the previous command shall continue.
IN MG HR OTE	HF	IN MG HR ASN	4.3.3, 4.3.7	4.3.7 The outcome of an HT shall not affect subsequent functionality.

## Canonical Sequences

One is only allowed to reduce to previous histories in the enumeration. Every sequence in an enumeration will be equivalent to an unreduced sequence.

There will be one unreduced sequence, called the **canonical sequence**, for each equivalence class of the equivalence relation discovered on  $S^*$ .

## Abstraction

Basic Idea: **Abstractions** may be used to omit or hide details about histories, resulting in shorter histories.

Formally, a **sequence abstraction** is a mapping  $\phi$  from  $X^*$  (**atomic** sequences) to  $Y^*$  (**abstract** sequences), such that

- sequences get no longer ( $|\phi(u)| \leq |u|$ )
- the stimulus ordering is preserved ( $\phi(u)$  is a prefix of  $\phi(uv)$ )

## Example

Two stimuli:

$R(n)$  = User requests a connection

$A(n)$  = Connection request accepted

A system  $n$  is “connected” if there has been a request followed by an accept.

## Example

Define an abstract stimulus  $C(n)$  formally as follows.

$$p_C(\lambda, n) = \text{false}$$

$$p_C(ux, n) = \begin{array}{l} \text{true if } x = A(n) \text{ and } p_C(u, n) = \text{false and} \\ \text{\ } u \text{ contains an } R(n) \end{array}$$

$$p_C(ux, n) = \text{false otherwise}$$

The atomic sequence

$R(7) R(5) A(3) A(5) R(4) R(9) A(4) A(7)$

is mapped to abstract sequence

$C(5) C(4) C(7)$ .

## Working with Abstractions

**Basic Idea: Enumerate with the abstract stimuli.**

**One enumerates to discover system behavior. Thus, one may not know enough about an abstraction to specify it formally.**

**In this case, give an initial, informal definition of the abstraction and enumerate using the abstract stimuli.**

**Based on the behavior discovered, create a formal definition of the abstraction.**

## Writing the Specification and Transformation to State Machine

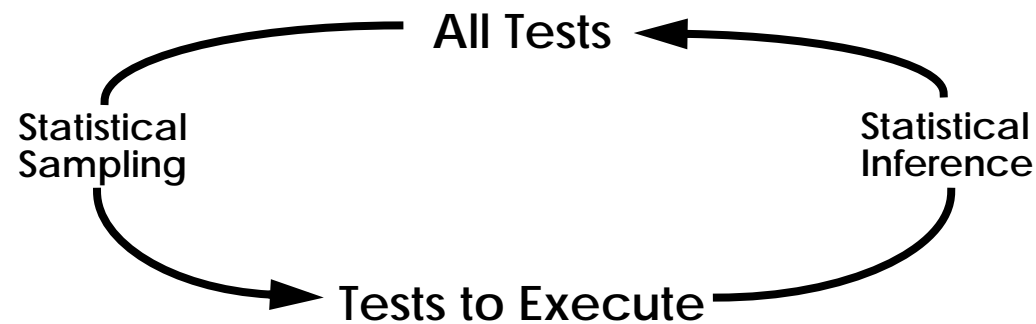
A state machine can be generated from the enumerations, specification functions, and abstraction definitions—the **enumeration state machine**.

The enumeration state machine gives intended software response for a known state and input.

States are distinguished by future behavior (responses).

## Statistical Software Testing

Statistical software testing is the application of statistical science to software testing problems.



Statistical testing must be a well-defined procedure, performed under specified operating conditions.

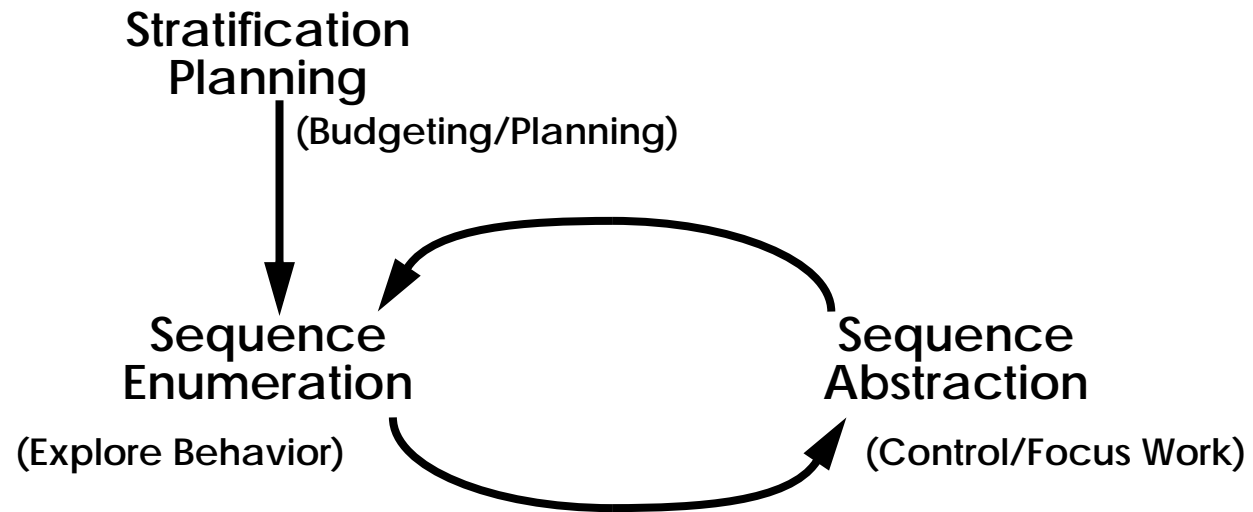
## Statistical Software Testing

Statistical software testing is primarily a black box testing method.

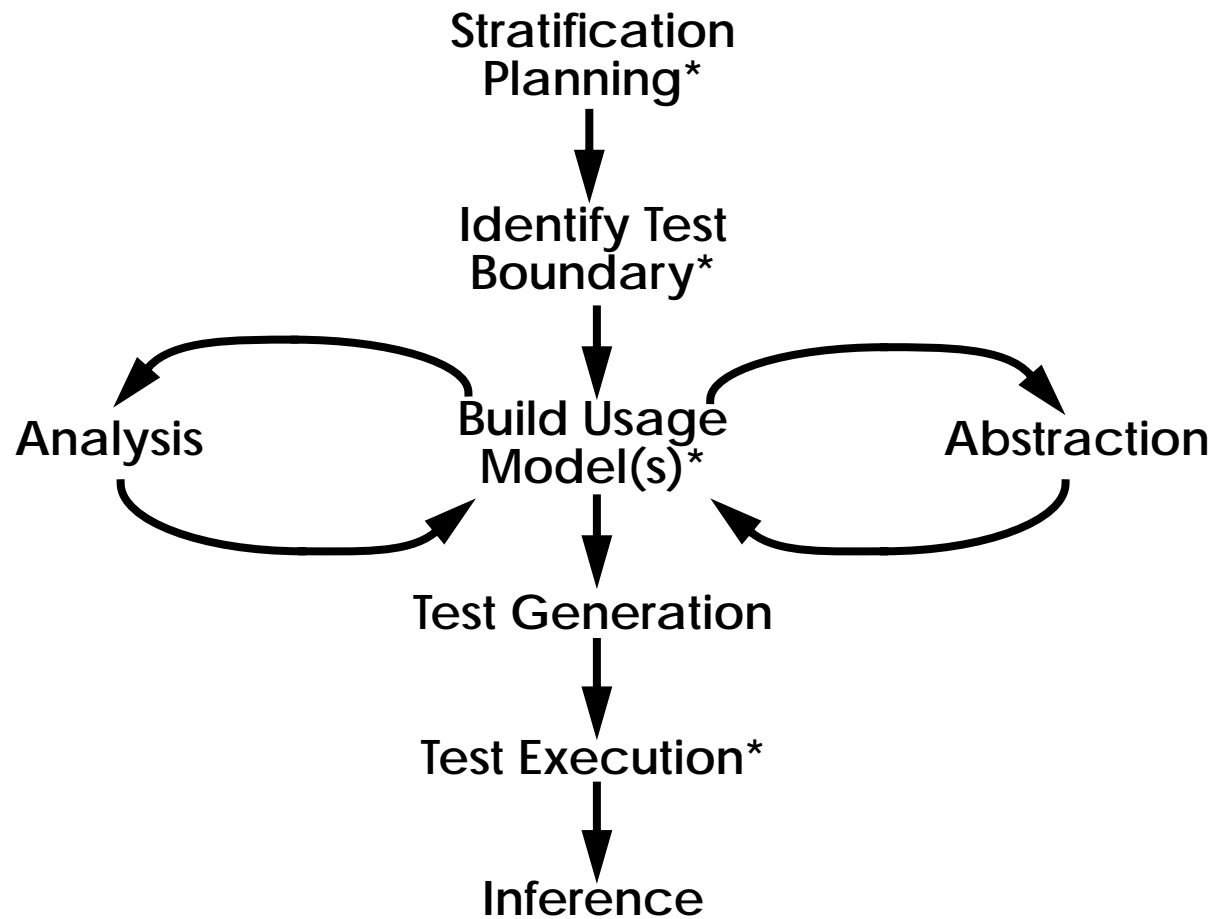


A usage specification gives an external, "user's view" of software use.

## Primary Techniques



## Statistical Software Testing Process



## Statistical Software Testing

Implementation knowledge may lead to bias in testing.

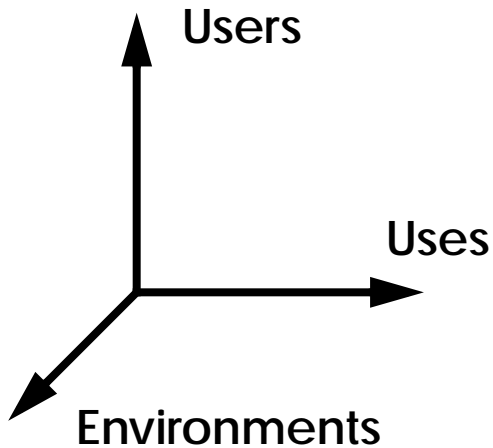
Focus on **states of use**, not software states.

External view in testing is aided by external specification.

Requirements traceability of specification supports

- Evaluating requirements coverage for testing
- Evaluating testability of requirements
- Generating non-random tests for particular requirements

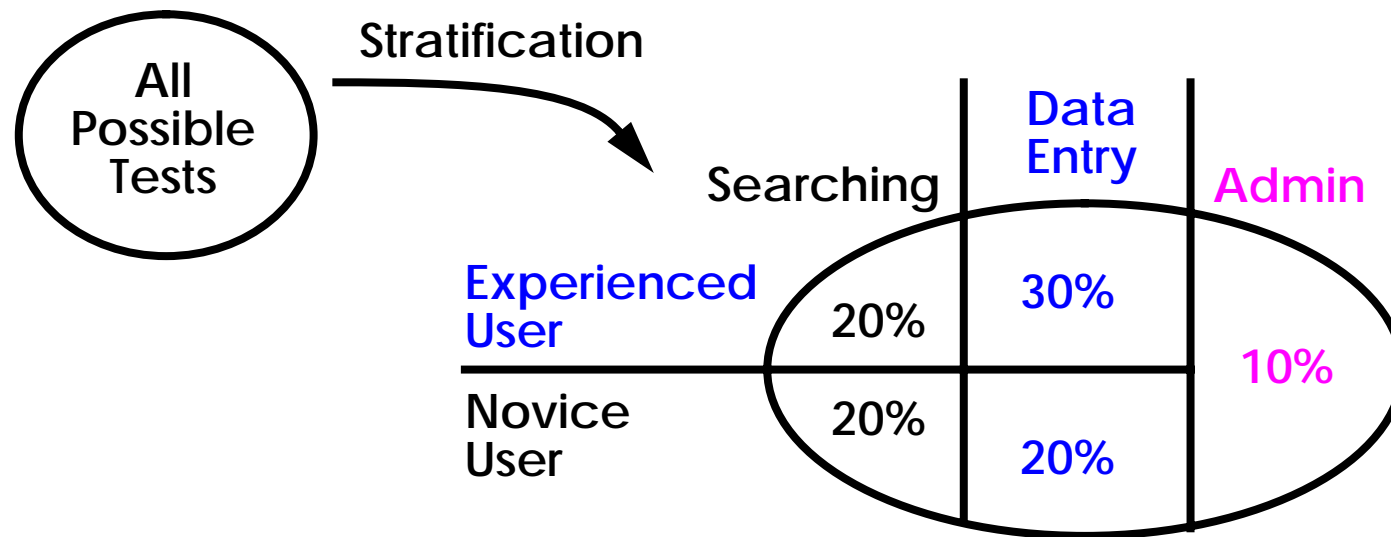
## Stratification Planning



A **stratum** is a combination of user, use, and environment.

- Definition of use must include initial and final states.
- Final states must be verifiable.

## Stratification Planning



Initial state: software uninvoked  
Final state: software terminated

What are all the conditions under which the software should terminate? These conditions are given by the specification.

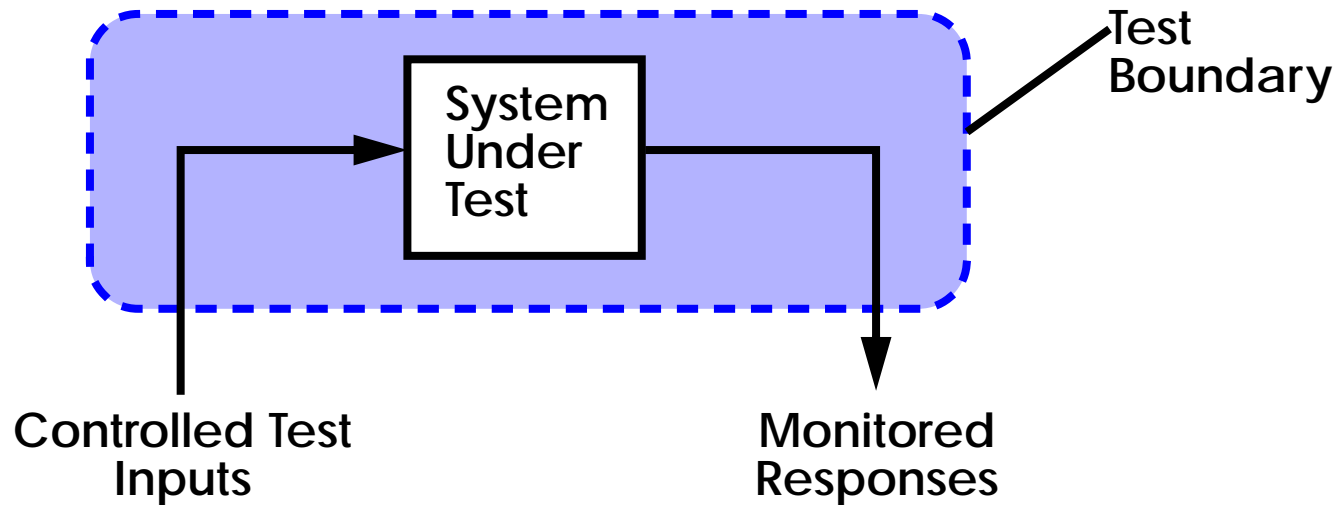
## Stratification Planning

The complete and consistent nature of the specification reveals classes of use which might otherwise be missed or insufficiently tested:

- Error conditions
- Critical use
- Unexpected use

This analysis is based on user-perceived function and risk.

## Identify Test Boundary



Identify and understand all interfaces to be cut during testing.

- Drive stimuli
- Monitor responses

## Identify Test Boundary

Sequence-based specification's deterministic model helps avoid:

- Undocumented inputs
- Misunderstood interfaces

while providing:

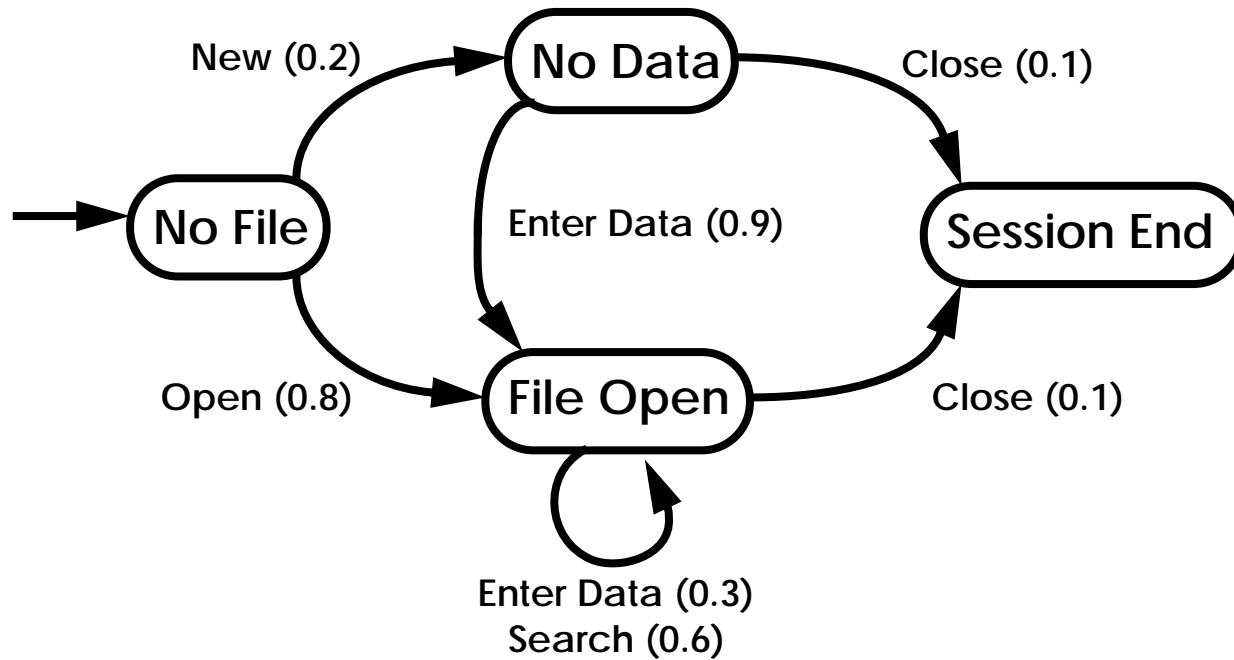
- Planning for test drivers / monitors
- Evaluation of interface testability
- Evaluation of requirements testability

## Usage Modeling

Expected use can be modeled as a finite-state, discrete parameter Markov chain.

- Whittaker and Thomason (1993) proposed the Markov chain as the model for software use.
- Walton (1995) explored optimization of model probabilities given constraints.
- Gutjahr (1997) explored acceleration of testing by modifying the chain probabilities and weighting the results.

## Usage Modeling



## Usage Modeling

A usage model is essentially a state machine whose transitions have an associated probability distribution.

States are distinguishable by future use (stimuli)

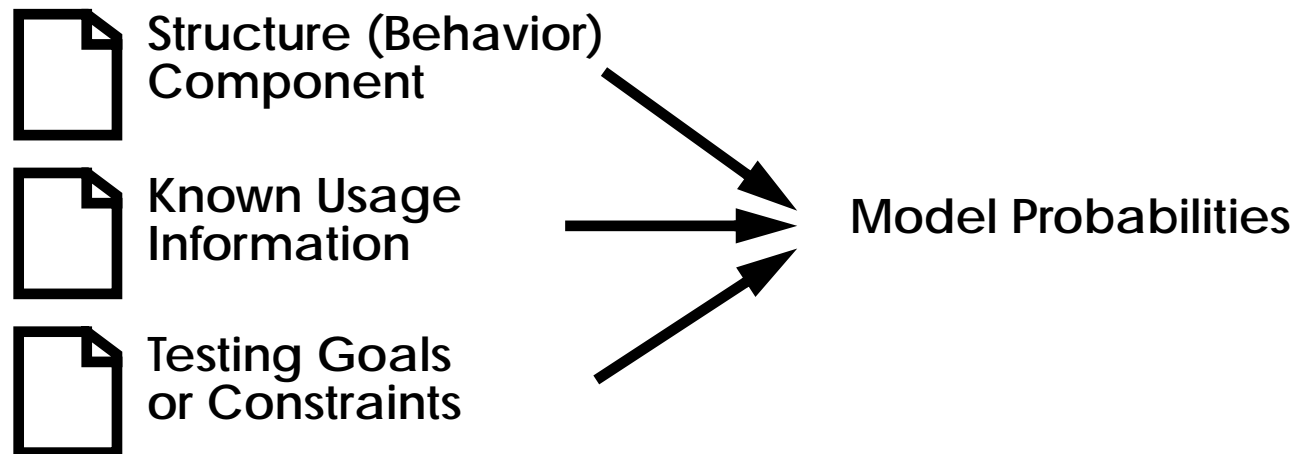
Different states from the specification state machine typically correspond to different states of use.

The specification state machine may be used as the “first cut” of a usage model. Testers proceed by:

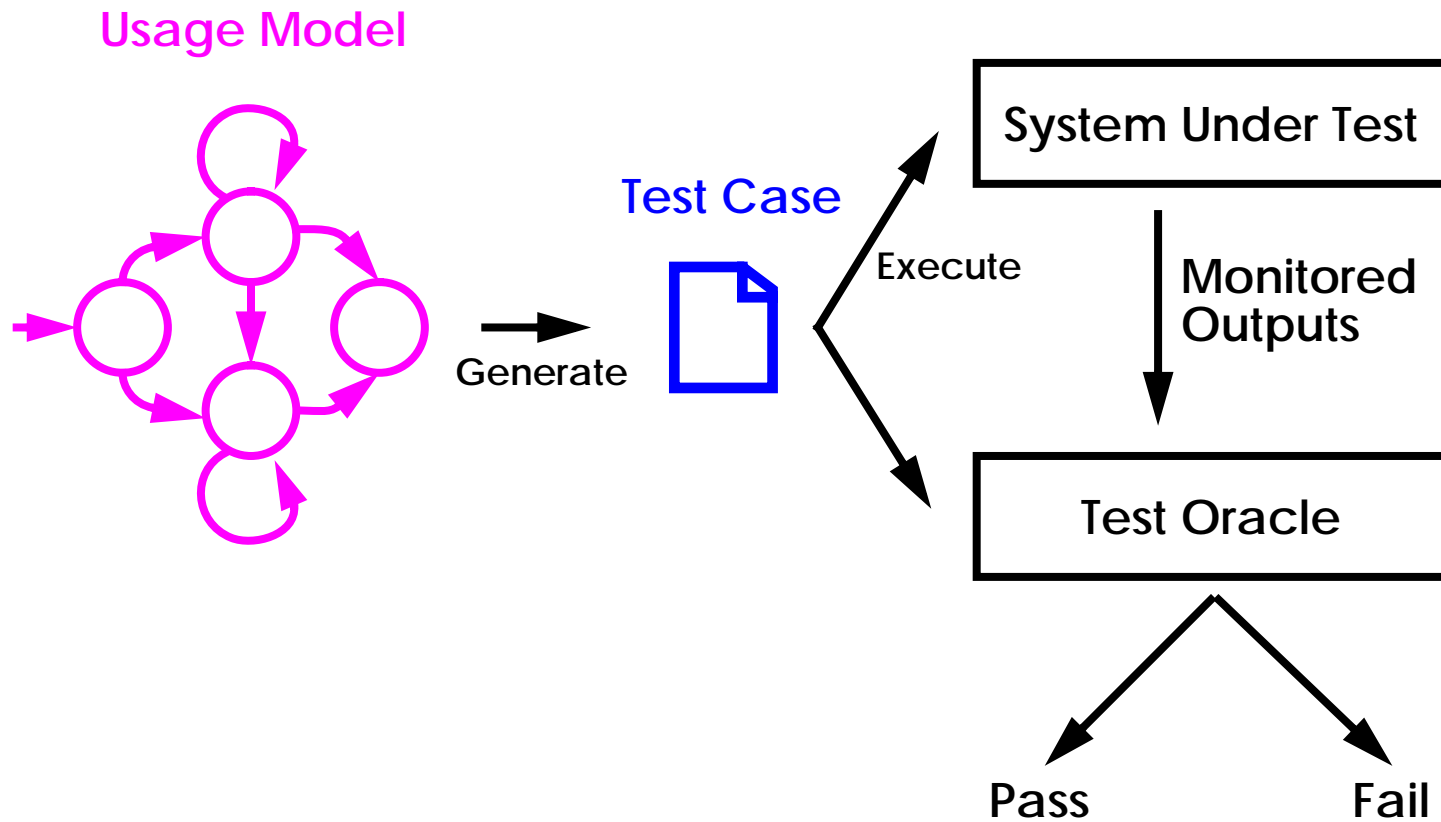
- Combining states to reduce model size
- Splitting states to reflect pure usage factors

This changes the focus from constructing the model to refining the model.

## Usage Modeling



## Test Execution



## Test Oracle

The specification state machine can be used as the test oracle.

The derivation of the usage model from the specification state machine enforces the correspondence.

Two issues:

- **Abstraction:** The oracle may not compute all abstractions.
- **Boundary:** The test boundary may differ from the system boundary; some unavoidable non-determinism may be introduced.

## Summary

The use of sequence-based specification has several impacts on a statistical software testing phase.

Direct benefits:

- Communication reduced by precise specification
- Requirements traceability simplifies test case crafting
- Precise system boundary reduces start-up time
- Reuse of specification state machine reduces model construction effort
- Reuse of specification state machine as test oracle reduces cost of automated testing

## Summary

### Indirect benefits:

- External focus reduces implementation bias
- Strict definition of interfaces improves testability analysis
- System understanding is improved for all practitioners