An Introduction to Software Reliability Engineering

Laurie Williams
williams@csc.ncsu.edu

About the tutorial author

John D. Musa

SREH9H 2

Tutorial Goal

You will be introduced to techniques that will help you to engineer reliability …

… develop more reliable software faster and cheaper;
… making it more competitive in the marketplace;
… enhancing your organization's market share and profitability;

… and increasing your value as a professional!

This prepares you for further learning, either a 2-day course [1] or self study using the book *Software Reliability Engineering: More Reliable Software Faster and Cheaper – Second Edition* [3].

Tutorial Objectives

Upon completing this tutorial, you will be able to:

1. Define a software-based product you plan to develop in SRE terms
2. Express relative use of a product's principal functions by developing operational profiles
3. Employ operational profiles and criticality information to:
   A. Greatly increase efficiency of development and test by optimally distributing people resources, test cases, and test time over operations
   B. Invoke test so as to much more accurately represent field use
   C. Plan feature release dates to better match customer needs

Tutorial Objectives

4. Determine the reliability / availability your customers need for a product, making optimal tradeoffs with cost and time of delivery
5. Engineer software reliability strategies to meet reliability / availability objectives more efficiently
6. Identify failures during system test and process failure data to track reliability growth of systems, guiding product release
7. Discuss how these practices can be used in your environment

Software Reliability Engineering – Developed to Address the Problem

1. SRE is primarily quantitative.
2. You add and integrate software reliability engineering (SRE) with other good processes and practices; you do not replace them.
   A. Development process is not externally imposed.
   B. You use quantitative information to choose the most cost-effective software reliability strategies for your situation.

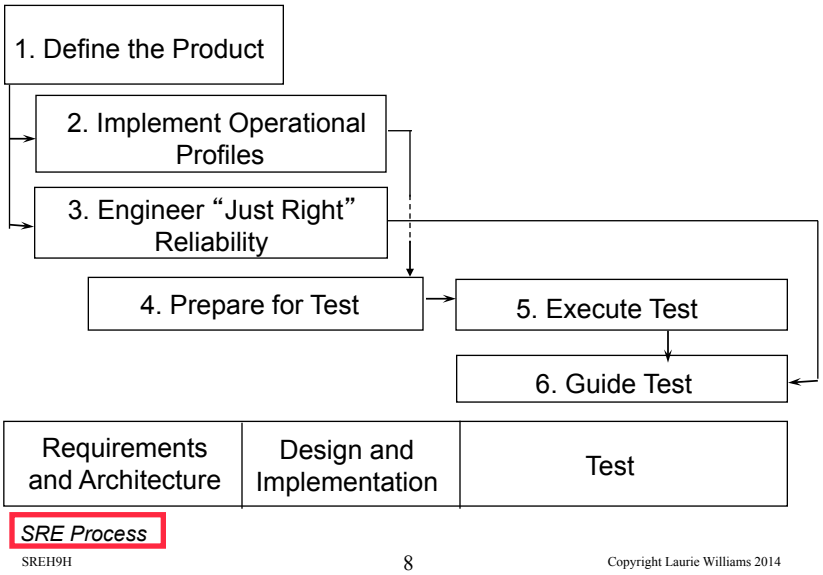
Overall … some simple ideas that will make you change the way you think about things that will improve your reliability … and some more complicated techniques for even more benefit.

6

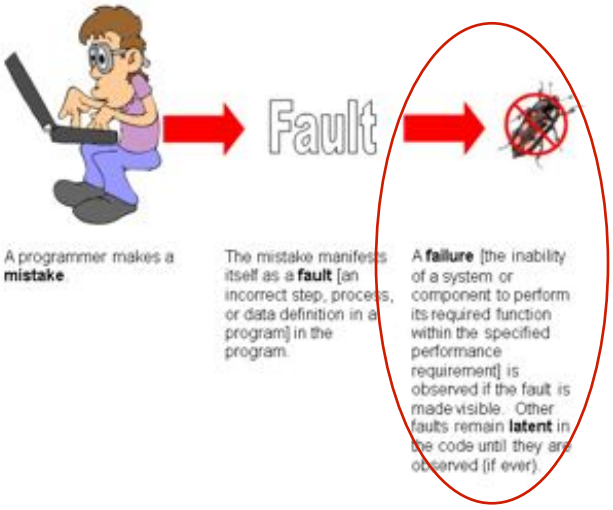
Outline

1. Introduction
2. SRE Process
3. Define the Product
4. Implement Operational Profiles
5. Engineer "Just Right" Reliability
6. Prepare for Test
7. Execute Test
8. Guide Test
9. Conclusion & Deploy SRE

Activities of SRE Process and Relation to Software Development Process



Faults vs. Failures



Mindset

- All faults should not be considered equally.
- Some faults are likely to surface as failures in normal use and will affect the reliability of the product in the eyes of the customer.
- Other faults can easily remain latent forever and will, therefore, never affect the reliability of the product product in the eyes of the customer.

- SRE is a system to get out the faults likely to affect product reliability.

Reliability

- Reliability: the probability that a system will continue to function without failure for a specified number of natural units or a specified time
  - Correctness, safety, operational aspects of usability and user-friendliness
  - "time" may be in natural or time units
    - Examples of "natural" units – runs, pages of output, transactions, telephone calls, jobs, semiconductor wafers, queries, API calls
  - Failure intensity = failures per natural or time unit

Availability

- Availability: average (over time) probability that a system is currently functional in a specified environment OR ratio of uptime to the sum of uptime plus downtime
  - Downtime for a given interval is the product of the length of the interval, the failure intensity, and the mean time to repair (MTTR)
    - 10 hours * .1 failures/hour * .5 hours/failure = .5 hours
  - MTTR is average time required to restore the data for a program, reload the program, and resume execution
  - Availability = 9.5/10 = .95 = 95%

Running Example - FONE FOLLOWER (FF) - Product Description

1. Subscriber calls FF, enters planned phone numbers (forwardees) to which calls are to be forwarded vs time.
2. FF forwards incoming calls (voice or fax) from network to subscriber as per program. Incomplete voice calls go to pager (if subscriber has one) and then voice mail.
3. Subscribers view service as the combination of standard telephone service with call forwarding.

SRE Process

13

Define the product

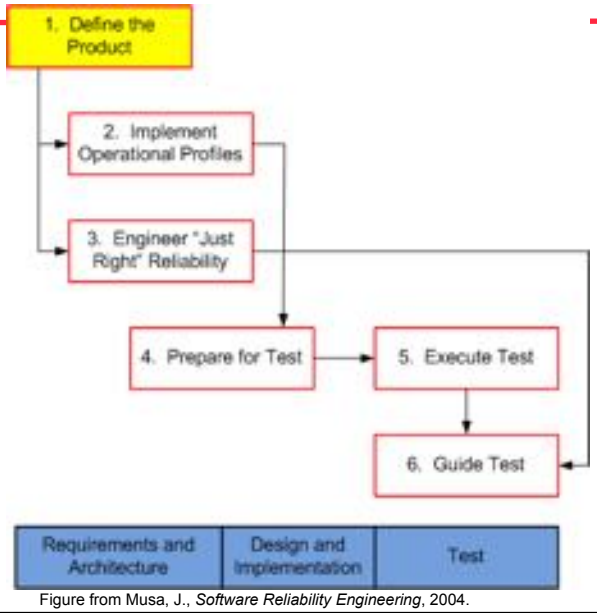


Figure from Musa, J., *Software Reliability Engineering*, 2004.

Define the Product

1. Who is supplier?
2. Who are customers and users?
3. List associated systems

   associated system: base product or system specially related to it that is tested separately

   A. Base product

   B. Major variations of base product (for substantially different environments, platforms, or configurations)
4. Consider frequently used supersystems (whole context) of base product or variations

Remember: User can't separate out new system from whole system when a problem occurs

Define the Product

SREH9H 15 Copyright Laurie Williams 2014

FF Product Description

A subscriber calls FF and enters the phone numbers to which calls are to be forwarded as a function of time. Incoming calls (voice or fax) from the network to the subscriber are forwarded as per the program. Incomplete voice calls go to a pager (if the subscriber has paging service) and then voice mail. FF uses a vendor-supplied operating system of unknown reliability. Subscribers view the service as the combination of standard telephone service with call forwarding.

The supplier is a major telecommunications systems developer. The customers are telecommunications operating companies, and they sell FF service to a wide range of businesses and individuals.

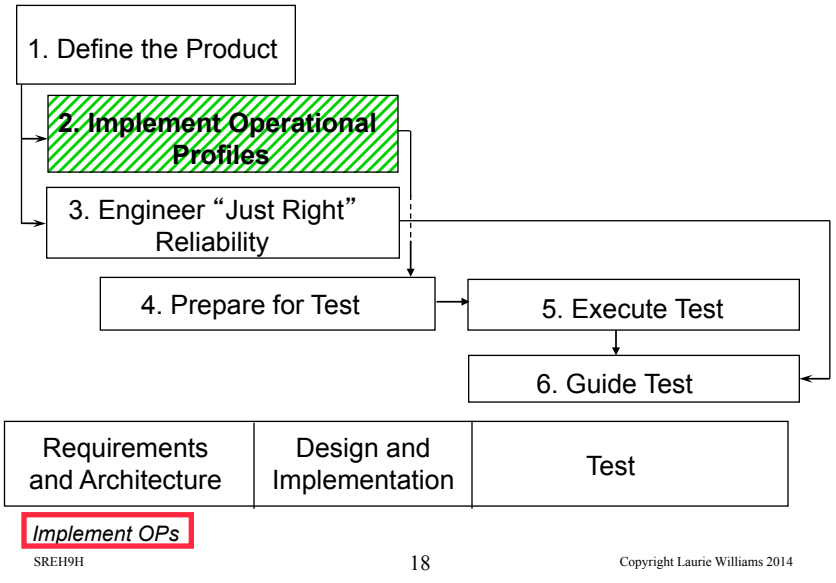
SREH9H 16 Copyright John D. Musa 1996-2006

FF "Define the Product"

- base product: FF
  - variation: FF Japan
- supersystem of base product: US telephone network and FF
  - supersystem of variation: Japanese telephone network and FF Japan

SREH9H 17

Activities of SRE Process and Relation to Software Development Process



SREH9H 18

Operations - 1

- Operation: a major system task performed for an initiator with control returned to the system when it is complete [so a new operation can start].
  - "major" implies it is related to a functional requirement or feature, not a subtask in the design
  - Use cases (and user stories in agile software development) are very operation-oriented.

Operations - 2

Illustrations - FF:

Process fax call, Enter forwardees, Audit section of phone number database

Other Illustrations:

Process transactions (purchases, sales, service deliveries, reservations)

Respond to events (alarms, mechanical movements, changes in state)

Produce reports

Operational Profiles

- An operational profile is a complete set of the operations (major system logical tasks) of a system with their probability of occurrence [e.g. the requirements and the probability of how often the users will use each requirement/scenario].
- You can use the operational profile to prioritize all aspects of development and to allocate resources accordingly.

Simple Operational Profile Example

Suppose you have a system with operations A and B. Operation A executes 90% of the time and Operation B, 10%. Assume each operation has 10 faults and you have 10 hours of test and debugging effort available. Finding and fixing each faults requires 1 hour of effort.

- How many "operational" faults if you spend 5 hours on each?
  - Spend equal amount of time on each
  - 5 faults remain in each
  - .9(5) + .1(5) = 5 "operational" faults (faults likely to be encountered by customer)
- How many "operational" faults if you spend your time relative to the operational profile?
  - Spend 9 hours on A, 1 hour on B
  - 1 fault remains in A, 9 faults remain in B
  - .9(1) + .1(9) = 1.8 "operational" faults

Developing an Operational Profile

Often done by systems engineers/marketing and product personnel, but system testers should be involved too.

Five principle steps in developing an operational profile:

1. Identify initiators of operations
2. Create operations list
3. Review operations list
4. Determine occurrence rates
5. Determine occurrence probabilities

All started in the requirements phase and refined iteratively in future phases.

Generally takes 1-2 weeks for small products, longer with larger products, but decreases after first release.

1. Identify the Initiators of Operations

Customer type: set of customers (organizations or individuals who acquire but may not directly employ your product) who have similar businesses and hence tend to have the same user types.

User types: set of users (individuals who directly employ the product) who tend to employ the product in the same way … list developed by considering customer types (above).

- User is anyone who can initiate an operation on the system
- Look at product business case, marketing data to obtain
- Consider job roles
- Don't forget maintainers and administrators
- E.g. FF: subscriber, system administrator

1. Identify the Initiators of Operations (cont'd)

- External systems that initiate operations on the system
  - e.g. FF: the network
- System under study if it initiates operations itself
  - e.g. FF: FF

- … think "actor" from a use case perspective 

2. Create the Operations List

- Generate an operations list for each initiator-type
- Consult system requirements, work process flow diagrams, user manuals, prototypes, and information on previous releases
- Meet with systems engineers, human factors engineers, marketing personnel and expected users
- Be sure to include "housekeeping operations" that (re)initialize or clean up data
- Rough guideline: each operation should have more than 100 deliverable source lines different from another operation.
  - High probability each test case would reveal unique faults.
- We should execute each operation at least once in test
  - unless it has a very low occurrence probability **and** is non-critical

Create Operations List: Illustration - FF

Initiator	Operations List
Subscriber	Enter forwardees
System admin.	Add subscriber
	Delete subscriber
Network	Proc. voice call, no pager, ans.
	Proc. voice call, no pager, no ans.
	Proc. voice call. pager, ans.
	Proc. voice call, pager, ans. on page
	Proc. voice call, pager, no ans. on page
	Proc. fax call
FF	Audit section of phone number database
	Recover from hardware failure

3. Review the Operations List

- Identify at least one expert for each initiator-type for the operations list to be as complete as possible
- Check to make sure:
  - Operations are of short duration in execution time (want to run lots of tests rather than a few long tests)
  - Each operation has substantially different processing from the others.
  - Operations are well-formed (sending messages and displaying data are PART of the operation, not the operation itself)
  - The list is complete with high probability
  - The total number of operations is reasonable (taking into account the budget)
    - 20 to several hundred operations, typically, depending on size
    - Cost to develop operational profile: roughly half a staff hour per operation
- The list will evolve over time and as the system is developed (need to adjust profile)

4. Obtaining Occurrence Rates

- Occurrence rate: number of occurrences of the operation divided by the time the total set of operations running
- Where to find data
  - Look for existing field data from previous release/similar system
  - Look at system logs
  - Search for existing business data; product business case
  - Ask a marketer (engineers should network with marketers!)
  - Record field operations from current product (data for old operations)
  - No recourse . . . Make estimates
    - Group low probability operations (or all of them) and assign equal probability to each
    - Apply the Delphi method.
- Instrument your code so that it identifies the operations that were executed . . . for future occurrence data.

Determine Occurrence Rates: Illustration - FF

Operation	Occ. Rate (per hr)
Proc. voice call, no pager, ans.	21,000
Proc. voice call, pager, ans.	19,000
Proc. fax call	17,000
Proc. voice call, pager, ans. on page	13,000
Proc. voice call, no pager, no ans.	10,000
Proc. voice call, pager, no ans. on page	10,000
Enter forwardees	9,000
Audit sect. - phone number database	900
Add subscriber	50
Delete subscriber	50
Recover from hardware failure	0.1
Total	100,000

5. Determine Occurrence Probabilities

- Divide occurrence rate of each operation by the total operation occurrence rate.
- Sort in order of descending probabilities.

Determine Occurrence Probabilities: Illustration - FF

Operation	Occ. Rate	Occ. Pr.
Proc. voice call, no pager, ans.	21,000	0.21
Proc. voice call, pager, ans.	19,000	0.19
Proc. fax call	17,000	0.17
Proc. voice call, pager, ans. on page	13,000	0.13
Proc. voice call, no pager, no ans.	10,000	0.10
Proc. voice call, pager, no ans. on page	10,000	0.10
Enter forwardees	9,000	0.09
Audit sect. - phone number data base	900	0.009
Add subscriber	50	0.0005
Delete subscriber	50	0.0005
Recover from hardware failure	0.1	0.000001
Total	100,000	1.0

Applying Operational Profiles

Proportionally distribute the number of new test cases and other validation and verification (V&V) activities according to the operational profile

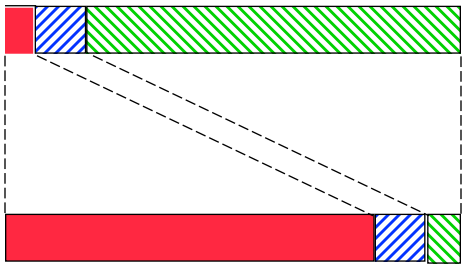
- Maximize chances the most important faults for reliability considerations are found

But, this information can be used in other ways as well:

- Aids in determining a competitive release strategy
  - Implement the most critical and/or most used in early releases
- Allocate development resources to best serve the needs of the customer as quickly as possible
  - Pareto principle (a small number of things occur most of the time): in a typical software system, 20% of the software operations may provide 80% of the functionality the customer wants

Operational Development - Illustration

Proportion of operations developed



Proportion of use/value represented

Release 1

Release 2

Release 3

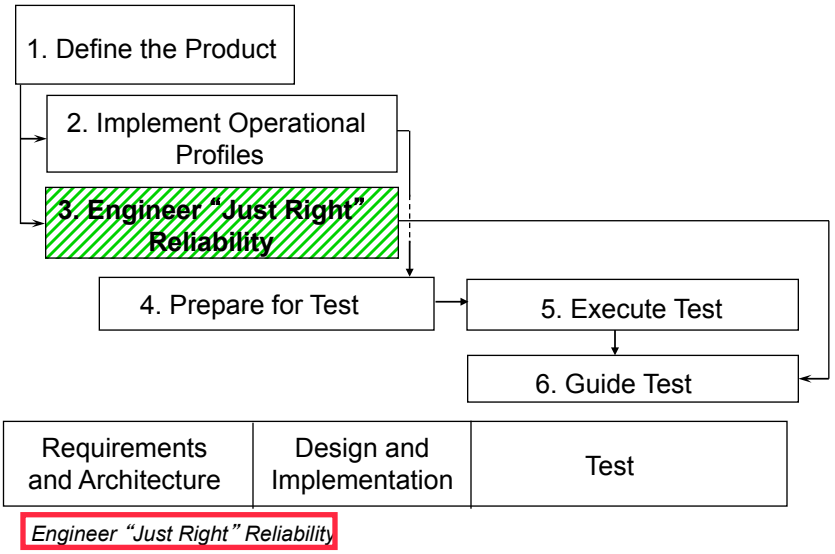
Finish highly used operations earlier (Release 1); delay less-used operations (Releases 2,3)

Summary

The operational profile is developed to systematically determine how to proportion effort.

- Operation initiators are enumerated
- The operations they users want to perform are enumerated
- The proportion of time each type wants to perform each operation are estimated

Activities of SRE Process and Relation to Software Development Process



SREH9H 36 Copyright Laurie Williams 2014

Steps to engineering "just right" reliability for your product

1. Set a system failure intensity objective … relative to the importance of quality for your product
2. From the system failure intensity objective, determine the FIO for the software under development
3. Choose software reliability strategies to optimally meet the FIO for the software under development

System failure intensity guidelines

Table 3.4 System failure intensity objective guidelines

Failure impact	Typical failure intensity objective (failure / hr)	Time between failures
Hundreds of deaths, more than $10^9 cost	10^{-9}	114,000 years
One death, around $10^6 cost	10^{-6}	114 years
Around $1000 cost	10^{-3}	6 weeks
Around $100 cost	10^{-2}	100 hours
Around $10 cost	10^{-1}	10 hours
Around $1 cost	1	1 hour

Table from Musa, J., *Software Reliability Engineering*, 2004.

Steps to engineering "just right" reliability for your product

1. Set a system failure intensity objective … relative to the importance of quality for your product
2. From the system failure intensity objective, determine the FIO for the software under development
3. Choose software reliability strategies to optimally meet the FIO for the software under development

Base product FIO

- Customer don't care where the failures come from … the hardware, the operating system, your base product, or the new software you are developing
- You can only find the FIO of your product by seeing how much is left for you after you take all "their" FIOs out.
- Example (Fone Follower):

– System FIO	200 failures/M calls
– US Telephone network FIO	- 95 failures /M calls
– Hardware	-1 failure /M calls
– Operating system	- 4 failures/M calls
– Base product FIO	100 failures/M calls

Steps to engineering "just right" reliability for your product

1. Set a system failure intensity objective … relative to the importance of quality for your product
2. From the system failure intensity objective, determine the FIO for the software under development
3. Choose software reliability strategies to optimally meet the FIO for the software under development

Software reliability strategies

- A software reliability strategy is a development activity that reduces failure intensity, incurring development cost and perhaps development time
- Plan software reliability strategy in the requirements phase, focusing on new operations of release.
- A software reliability strategy may be selectable (requirements, design, or code reviews) or controllable (amount of system test, amount of fault tolerance).

Software reliability strategies (cont'd)

- Basic failure intensity is the failure intensity that would exist at the start of system test for a project without reviews or fault tolerance.
  - Similar across many products given that no software reliability strategies have been applied
- FIRO is the ratio of the basic failure intensity to the software under development failure intensity objective.
- Choice of strategy based on predicting the required failure intensity reduction objective (FIRO)
  - FIRO is the failure intensity reduction that must be obtained through software reliability strategies

Software reliability strategies (cont'd)

Possible reliability strategies:

- Use of requirements review
- Use of design review
- Use of code review
- Use of unit testing
- Degree of fault tolerance designed into system
  - Fault tolerance is the ability of a system or component to continue normal operation despite the presence of hardware or software fault. [IEEE]
- Amount of system test

Illustration - Ultrareliable System

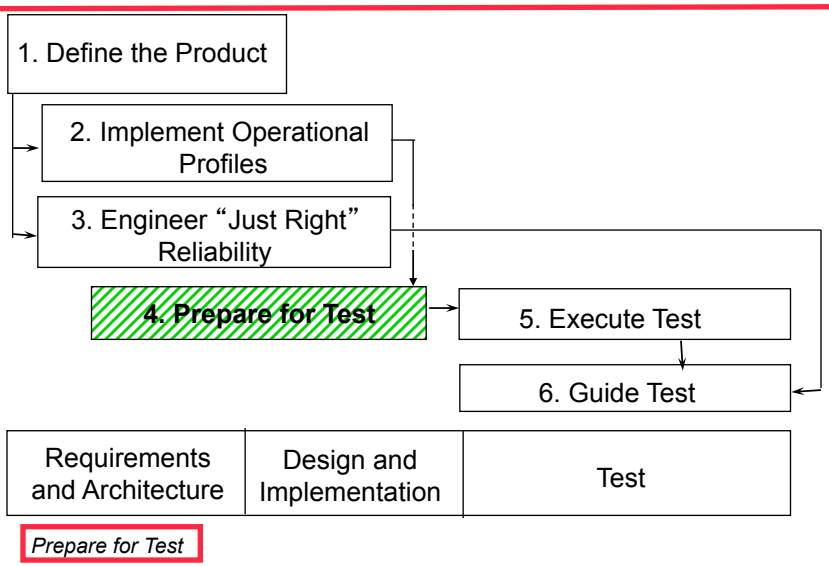
Allocate FIRO among reliability strategies

FIRO = 29000

Step	Reliability Strategy	FIRO Alloc.	Remaining FIRO
1	Early system test	8	3625
2	Requirements reviews	2	1812
3	Design reviews	2	906
4	Code reviews	2	453

Engineer "Just Right" Reliability - Choose SR Strategies – Ultrareliable System

Activities of SRE Process and Relation to Software Development Process



SREH9H

46

Prepare for Test

1. For each base product and variation:
   A. Specify new test cases for new operations for current release, based on the operational profile
   B. Specify test procedure, based on the test operational profile and traffic level
2. Provide for:
   A. Setup and invocation of test cases
   B. Recording of run executed (operation, test case, indirect input variables) and outputs
   C. Cleanup
   D. Documentation of expected behavior of test cases

Step 1: Estimate the number of new test cases for current release

- Estimate the number of test cases you need
  - History of your project's test cases/line of executable code
  - Industry data (depends on failure intensity reduction objective):
    - 2-3 test cases/thousand lines of code for moderate reliability
    - 20-33 test cases/thousand lines of code for high reliability
  - FF example:
    - 8 new test cases/KLOC* x 80 KLOC = 640 test cases

*KLOC = thousand lines of code

Step 2: Estimate the number of new test cases you can prepare, staff hours

- Estimate the number you have the capacity to prepare
  - Industry data: 0.4-16 hours/test cases (preparation)
  - Your knowledge of staff hours available for preparation
- Example:
  - 18 weeks, 720 hours
  - Available staff 2.5
    - 1800 staff hours
  - 3 hours/test case
  - 600 new test cases

Step 3: Estimate the number of new test cases you can prepare, budget

- Estimate the number you have the capacity to prepare based upon the test case budget (% of software development budget)
- Example
  - Software development budget $2M
  - Test case budget 10% of budget
  - Test case preparation cost $250/test case
  - Test case budget $200K
  - $200K / $250 per test case = 800 new test cases

Step 4: Decide how many test cases you will prepare

- Take minimum of Steps 2 and 3
- Example
  - Minimum based upon Step 2: 600 new test cases
  - Not that far away from "needed" (Step 1)
    - So we will feel OK
    - If the difference is large, we must consider implications to quality and/or resource

Step 5: Distributing test cases among new operations

Based upon occurrence proportion for new operation

- Proportion of occurrences of new operation with respect to occurrences of all new operations for a release.
- First release: occurrence proportion = occurrence probability
- Future releases: occurrence probability of operation/sum of occurrence probabilities of all new operations

Table 4.4 Occurrence proportions for Fone Follower base product

Operation	Occurrence probability	Occurrence proportion
Process voice call, no pager, answer	0.21	0.21
Process voice call, pager, answer	0.19	0.19
Process fax call	0.17	0.17
Process voice call, pager, answer on page	0.13	0.13
Process voice call, no pager, no answer	0.10	0.10
Process voice call, pager, no answer on page	0.10	0.10
Enter forwardees	0.09	0.09
Audit section - phone number database	0.009	0.009
Add subscriber	0.0005	0.0005
Delete subscriber	0.0005	0.0005
Recover from hardware failure	0.000001	0.000001
Total	1	1

Table 4.5 Occurrence proportions for Fone Follower 2 base product

Operation	Occurrence probability	Occurrence proportion
Process voice call, no pager, answer	0.168	0
Process voice call, pager, answer	0.152	0
Process fax call	0.136	0
Process voice call, pager, answer on page	0.104	0
Y	0.1	0.5
Z	0.1	0.5
Process voice call, no pager, no answer	0.08	0
Process voice call, pager, no answer on page	0.08	0
Enter forwardees	0.072	0
Audit section - phone number database	0.0072	0
Add subscriber	0.0004	0
Delete subscriber	0.0004	0
Recover from hardware failure	0.0000008	0
Total	1	1

Tables from Musa, J., *Software Reliability Engineering*, 2004.

Step 6: Distribute New Test Cases Among New Operations

Illustration - FF - Base product:

Operation	Occ. Prop.	Init. New TC
Proc. voice call, no pager, ans.	0.21	105
Proc. voice call, pager, ans.	0.19	95
Proc. fax call	0.17	85
Proc. voice call, pager, ans. on page	0.13	65
Proc. voice call, no pager, no ans.	0.10	50
Proc. voice call, pager, no ans. on page	0.10	50
Enter forwardees	0.09	45
Audit section – phone number data base	0.009	5
Add subscriber	0.0005	0
Delete subscriber	0.0005	0
Recover from hardware failure	0.000001	0
Total	1	500

Step 7: More considerations

Must have at least one test case per operation

Table 4.7 Setting test case minimums for Fone Follower base product

Operation	Occurrence proportion	Adjusted new test cases
Process voice call, no pager, answer	0.21	105
Process voice call, pager, answer	0.19	95
Process fax call	0.17	85
Process voice call, pager, answer on page	0.13	65
Process voice call, no pager, no answer	0.10	50
Process voice call, pager, no answer on page	0.10	50
Enter forwardees	0.09	45
Audit section – phone number database	0.009	5
Add subscriber	0.0005	1
Delete subscriber	0.0005	1
Recover from hardware failure	0.000001	1
Total	1	503

Table from Musa, J., *Software Reliability Engineering*, 2004.

Step 8: And more considerations

Critical operations – one for which successful execution adds a great deal of extra value and failure causes a great deal of impact with respect to human life, cost o system capability

Acceleration factor (A):
- FIO (system)/FIO (operation)

Example:
- FIO system = 100 failures/Mca
- FIO "recover from hardware failures" = .025 failures/Mcalls
- Acceleration factor (A)=4000
- Test cases=
  - (500)(0.000001)(4000) = 2

Table 4.8 Distributing test cases for critical operation of Fone Follower

Operation	Occurrence proportion	Modified new test cases
Process voice call, no pager, answer	0.21	105
Process voice call, pager, answer	0.19	95
Process fax call	0.17	85
Process voice call, pager, answer on page	0.13	65
Process voice call, no pager, no answer	0.10	50
Process voice call, pager, no answer on page	0.10	50
Enter forwardees	0.09	45
Audit section – phone number database	0.009	5
Add subscriber	0.0005	1
Delete subscriber	0.0005	1
Recover from hardware failure	0.000001	2
Total	1	504

Table from Musa, J., *Software Reliability Engineering*, 2004.

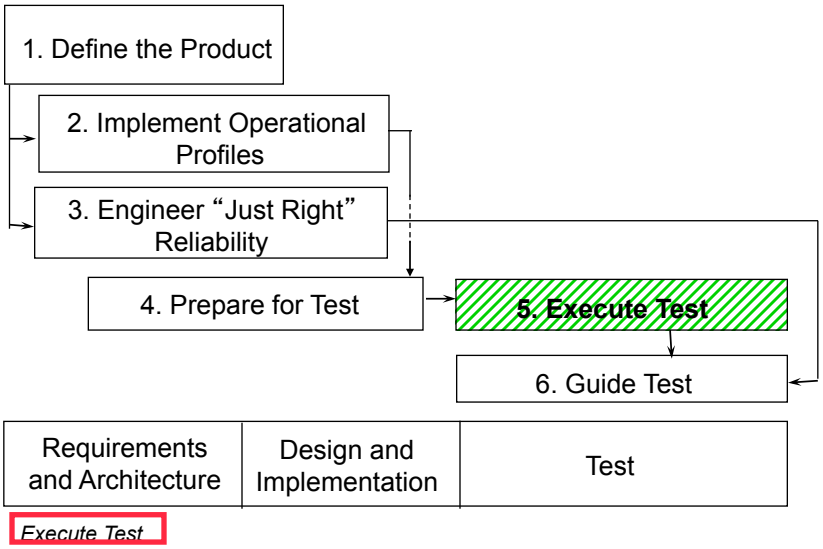
Step 9: Using judgment

- Adjust number of test cases based upon other judgment
  - Number of equivalence classes < number of test cases
  - Number of equivalence classes > number of test cases
  - Other …

Step 10: What if there are too many test cases now?

- Try to do them all
- Redistribute by ratio:
  - Original number of test cases/new number of test cases
  - Make sure you don't go below one test case per operation
  - Combine operations if you need to ☹
    - Multiple equivalence classes

Activities of SRE Process and Relation to Software Development Process



SREH9H 58

Testing

- Feature Test
  - All new test cases for new operations
  - Independent of each other (need set up and clean up for each operation)
  - Test cases not replaced in group for possible future re-selection
- Load Test
  - All valid tests for all releases including acceptance tests and performance tests
  - Full interaction with other test cases in different environments, no setup before test
  - Test cases are replaced in group for possible future re-selection
- Regression Test
  - All critical test cases + subset of all valid test cases from all releases
  - Independent of each other [for each build during the load test period]
  - Test cases not replaced in group for possible future re-selection

Step 1: Determine test time

- System period multiplied by the number of test units
  - e.g. 8 weeks, 40 hours/week = 320 hours

High Level: Planning and Allocating test time

1. Allocate among associated system to be tested (base, variations, supersystem of base product and variations)
2. Allocate rest among feature, regression, and load test for reliability growth

Step 2: Expected fraction of field use

Associated system	*F*	*D*	*R*	*N*	*A*	Test time (hr)
Fone Follower (base product)	0.6	1	1	0.6	0.521	167
Fone Follower Japan (variation)	0.4	0.3	1	0.12	0.104	33
Supersystem FF	0.6	1	0.6	0.36	0.313	100
Supersystem FF Japan	0.4	0.3	0.6	0.072	0.062	20
Totals				1.152		320

•F = expected fraction of field use

• Note: sum of the Fs of the supersystems related to a base product or variation must equal the F of that base product or variation

Table from Musa, J., *Software Reliability Engineering*, 2004.

Step 3: Assign Difference

Associated system	*F*	*D*	*R*	*N*	*A*	Test time (hr)
Fone Follower (base product)	0.6	1	1	0.6	0.521	167
Fone Follower Japan (variation)	0.4	0.3	1	0.12	0.104	33
Supersystem FF	0.6	1	0.6	0.36	0.313	100
Supersystem FF Japan	0.4	0.3	0.6	0.072	0.062	20
Totals				1.152		320

•D=difference between base product and variations (so base product = 1)

• If variations result from functional differences, D=S where S is the sum of the occurrence probabilities of new operations that are functionally different from those of the base product or previously-considered variations.

•If differences occur from implementation differences, estimate the fraction of lines of developed code that is not in the base product.

Table from Musa, J., *Software Reliability Engineering*, 2004.

Step 4: Estimate relative reliability risk

Associated system	*F*	*D*	*R*	*N*	*A*	Test time (hr)
Fone Follower (base product)	0.6	1	1	0.6	0.521	167
Fone Follower Japan (variation)	0.4	0.3	1	0.12	0.104	33
Supersystem FF	0.6	1	0.6	0.36	0.313	100
Supersystem FF Japan	0.4	0.3	0.6	0.072	0.062	20
Totals				1.152		320

• R=relative reliability risk increase prior to test caused by independent systems of subsystem (base = 1) based upon how much we know about them when the current release of the base product or variations was designed.

Table from Musa, J., *Software Reliability Engineering*, 2004.

Step 5: Determine Allocation Number

Associated system	*F*	*D*	*R*	*N*	*A*	Test time (hr)
Fone Follower (base product)	0.6	1	1	0.6	0.521	167
Fone Follower Japan (variation)	0.4	0.3	1	0.12	0.104	33
Supersystem FF	0.6	1	0.6	0.36	0.313	100
Supersystem FF Japan	0.4	0.3	0.6	0.072	0.062	20
Totals				1.152		320

- N = F x D x R

Table from Musa, J., *Software Reliability Engineering*, 2004.

Step 6: Compute test time allocation fraction

Associated system	*F*	*D*	*R*	*N*	*A*	Test time (hr)
Fone Follower (base product)	0.6	1	1	0.6	0.521	167
Fone Follower Japan (variation)	0.4	0.3	1	0.12	0.104	33
Supersystem FF	0.6	1	0.6	0.36	0.313	100
Supersystem FF Japan	0.4	0.3	0.6	0.072	0.062	20
Totals				1.152		320

- Add N's, normalize, and allocate test time proportionally

Table from Musa, J., *Software Reliability Engineering*, 2004.

Step 7: Calculate test time

Associated system	*F*	*D*	*R*	*N*	*A*	Test time (hr)
Fone Follower (base product)	0.6	1	1	0.6	0.521	167
Fone Follower Japan (variation)	0.4	0.3	1	0.12	0.104	33
Supersystem FF	0.6	1	0.6	0.36	0.313	100
Supersystem FF Japan	0.4	0.3	0.6	0.072	0.062	20
Totals				1.152		320

- Test time = A * available test time (from Step 1)

Table from Musa, J., *Software Reliability Engineering*, 2004.

Step 8: Allocate test time

- For each system
  - Execute to completion all new tests for the new releas
  - Allocate regression test time
    - Expected number of builds for the new release multiplied by the average time required to execute the test cases
  - Remaining time to load test
    - Usually at least several times the length of feature test + regression test (combined)
    - If remaining time is too low or negative, negotiate immediately

Invoke Test Reprisal

Feature
- Select randomly from set of new test cases for release
- Run one to completion before running the next
- Provide setup and cleanup
- Do not replace test case after execution

Load
- Choose from set of all valid test cases according to test operational profile
- Replace test case after execution

Regression
- Choose subset, including all critical test cases and specified number of randomly-chosen non-critical test cases
- Do not replace test case after execution
-

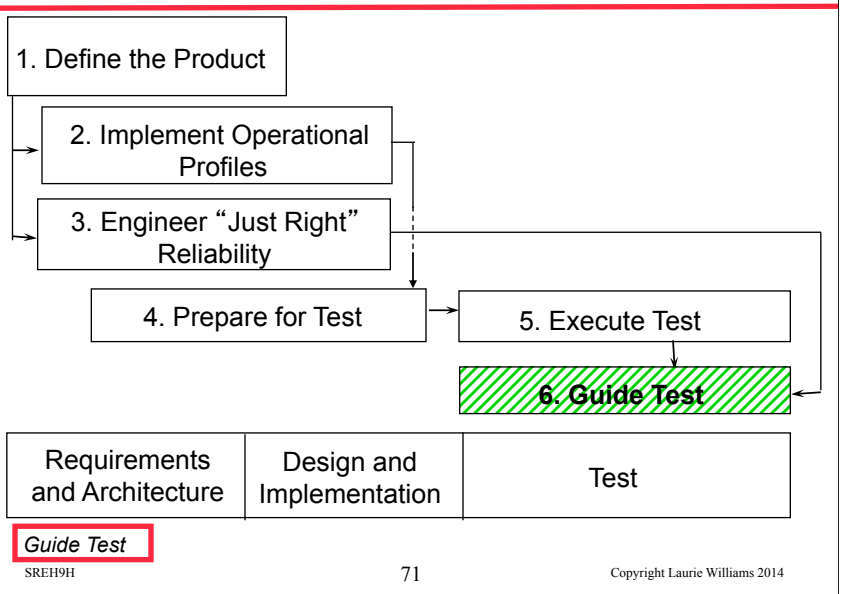
Identify System Failures

1. Analyze test output promptly for deviations
   deviation: departure of system behavior in execution from expected behavior
2. Determine which deviations are failures
3. Establish when failures occurred, using common reliability unit chosen for failure intensities, with units accumulated in sequence they occur

Execute Test - Identify System Failures

SREH9H 70

Activities of SRE Process and Relation to Software Development Process



SREH9H 71 Copyright Laurie Williams 2014

Estimating failure intensity

Make periodic estimates of FI/FIO based on failure data using a software reliability estimation program, such as CASRE.

Software reliability estimation programs are based on software reliability estimation models (such as the Musa-Basic model) and statistical inference.

Guideline:

- Estimate weekly if more than 3 months until release
- Estimate semi-weekly if 1-3 months until release
- Estimate daily if less than one month until release

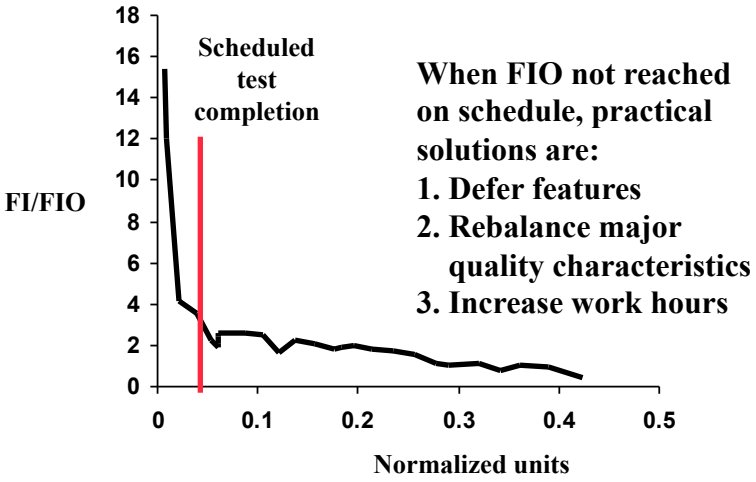
.

Sharp drop is typical when you drive test with the operational profile – invoke most frequently-used operations first and remove these faults efficiently.

Dotted line more typical of system test when all operations are tested with equal probability.

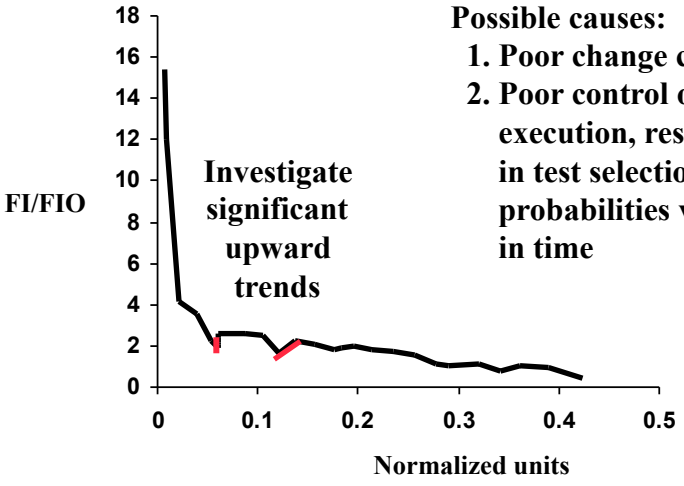
Figure from Musa, J., *Software Reliability Engineering*, 2004.

Interpret Plot : Illustration - FF



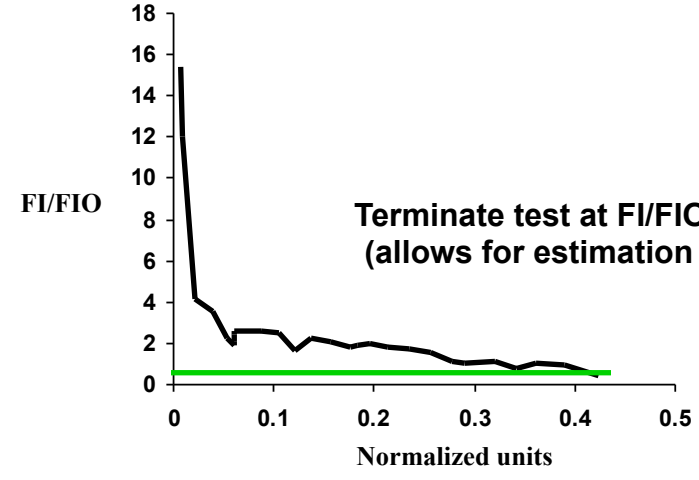
When FIO not reached on schedule, practical solutions are:
1. Defer features
2. Rebalance major quality characteristics
3. Increase work hours

Interpret Plot : Illustration - FF



Possible causes:
1. **Poor change control**
2. **Poor control of test execution, resulting in test selection probabilities varying in time**

Interpret Plot : Illustration - FF



Selecting software reliability estimation model

- Model simplicity
  - Should be understandable by software engineer without extensive mathematical background
- Model maturity
  - Well developed model that has been applied broadly with real data and given reasonable results
- Based on execution time
  - Evidence in support of execution time models rather than calendar (ordinary) time models
  - Best characterizes the failure-inducing stress placed on software

Point Estimates and Confidence Bounds

- Models compute point estimates for reliability, or the "most likely" or "best value."
- Most also compute confidence bounds around the point estimate to see how much one can rely upon the point estimate
  - Probable error (variance) for the model parameters and collected data
- "With 90% confidence, we expect to get between 17 and 25 failures."

Simplest model: Nelson Model

Point estimate

- (point estimate) R = lim (n→ ∞) 1-(n_f/n)
- n is the number of runs and n_f is the number of failures in n runs

Confidence bounds

- Assume: normal distribution, n>30

- p=the proportion of successes in a random sample of n runs,
- q = 1-p
- $Z_{\alpha/2}$ is the value of the standard normal curve leaving an area of α/2 to the right. For α=0.95, $z_{0.025}$ = 1.96.

Musa models

- Musa Basic
  - Assumes finite failures in infinite time
  - Tends to be optimistic (low) in estimating FI/FIO
  - Use if: product is stable (not changing or evolving as test proceeds), has a very low FIO, and a long test period [so you can expect that all but a very small number of failures can be removed by the end of system test]
- Musa-Okumoto logarithmic
  - Assumes infinite failures
  - Tends to be pessimistic (high) in estimating FI/FIO
  - Use if: characteristics not as specified above

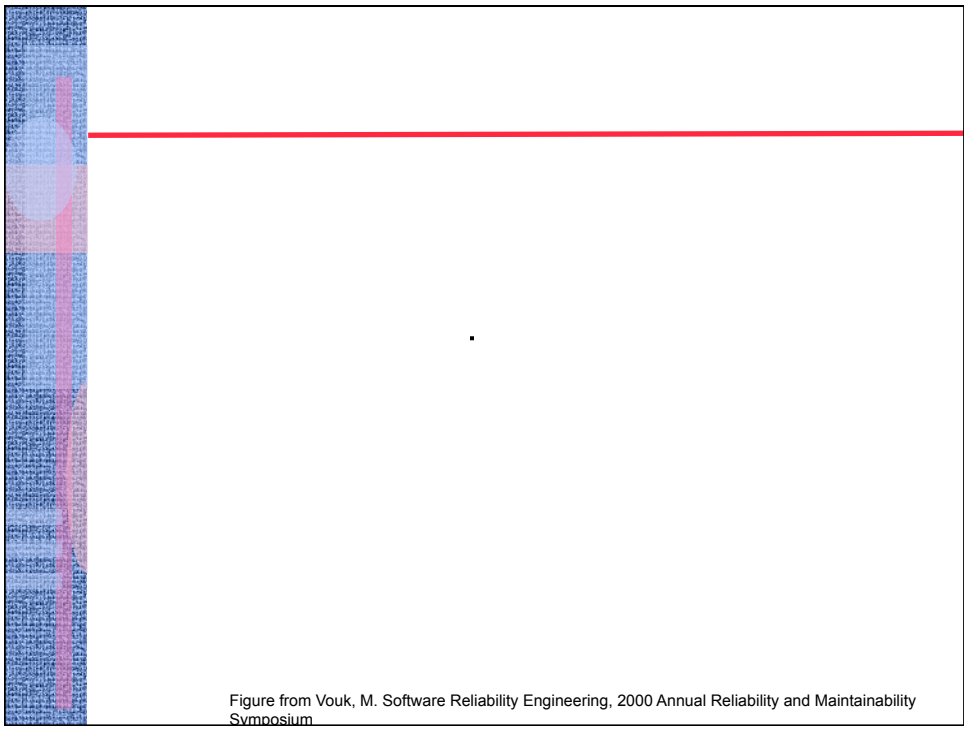


Figure from Vouk, M. Software Reliability Engineering, 2000 Annual Reliability and Maintainability Symposium

When to release the product

1. Terminated test satisfactorily for the base product with the failure intensity to failure intensity objective (FI/FIO) ratio at 0.5 or less
2. Terminated test satisfactorily for all the product variations, with their FI/FIO ratios not appreciably exceeding 0.5
3. Accepted the product and its variations in any acceptance test rehearsals planned for them.
4. Accepted all supersystems.
5. Resolved all outstanding (usually Sev 1 and 2) failures.

During Post Delivery and Maintenance

- Determine the actual reliability achieved
- Determine the actual operational profile experienced
  - Affects the engineering of "just right" reliability for the next release.
  - Build recording and reporting mechanisms into the product.
- Collect data on failure intensity and customer satisfaction.

SRE and You

1. SRE gives you a powerful way to engineer software-based products so you can be confident in the availability and reliability of the product you deliver as you deliver it in minimum time with maximum efficiency.
2. With SRE you control the process; it doesn't control you.
3. Discussion:
   - How much of all of this can be fit into your current (and more modern) software development methodologies?
   - Brainstorm how you can fit it in. We will share.

SREH9H 84

To Explore Further

1. *More Reliable Software Faster and Cheaper,* classroom or distance learning course: http://members.aol.com/JohnDMusa/
2. SRE website: The essential guide to software reliability: http://members.aol.com/JohnDMusa/
   A. SRE Orientation (overviews of different lengths)
   B. Courses (classroom and distance learning)
   C. Consulting information
   D. Practitioners' Corner (extensive user experiences with SRE and important application examples, advice on deploying SRE, comprehensive standards information)

Conclusion & Deploy SRE - Explore

SREH9H 85

To Explore Further

E. Resources for Everyone (download free failure intensity estimation program CASRE, join free SRE professional organization, access SRE Network, view conference information, learn from Question of the Month, use glossary)
F. Researchers' Corner (access to failure interval data and enormous debugging history archive, access to comprehensive lists of open source projects likely to have free access to all kinds of data)
G. Professors' Corner (how to teach SRE, slides and material for SRE courses, network to other professors teaching SRE)

Conclusion & Deploy SRE - Explore

SREH9H 86

To Explore Further

3. Musa, J. D., *Software Reliability Engineering: More Reliable Software Faster and Cheaper – Second Edition.* Detailed, extensive treatment of practice. Browse & order at SRE Website [2].
4. Musa, Iannino, Okumoto; *Software Reliability: Measurement, Prediction, Application,* ISBN 0-07-044093-X, McGraw-Hill, 1987. Extensive theoretical background.
5. Musa, J.D., *More Reliable Software Faster and Cheaper.* Overview of SRE for managers and anyone wanting a fast, broad understanding of the topic. Download from SRE Website [2]. (Click on "Overview")

Conclusion & Deploy SRE - Explore
SREH9H 87

To Explore Further

6. Lyu, M. (Editor), *Handbook of Software Reliability Engineering* , ISBN 0-07-039400-8, McGraw-Hill, 1996.
7. SRE professional organization: IEEE Computer Society Technical Committee on Software Reliability Engineering. Publishes newsletter, sponsors ISSRE annual international conference. Join through SRE Website [2].
8. SRE Network: Communicate by email with hundreds of people interested in field. See SRE Website [2].

Conclusion & Deploy SRE - Explore
SREH9H 88