M. Weintraub and F. Tip

#### **TESTING STRATEGIES**

# Thanks go to Andreas Zeller for allowing incorporation of his materials

Northeastern University College of Computer and Information Science

440 Huntington Avenue 202 West Village H Boston, MA 02115 T 617.373.2462 ccis.northeastern.edu

#### **TESTING**



+ Testing: a procedure intended to establish the quality, performance, or reliability of something, esp. before it is taken into widespread use

#### **RECALL FROM BEFORE – THESE ARE OUR TECHNIQUES FOR EVALUATING SOFTWARE**



#### THE CURSE OF FUNCTIONAL TESTING



∞ possible runs ,

#### **ITS STRUCTURAL TESTING COROLLARY**



∞ possible runs

#### **COMBINING METHODS**



∞ possible runs

# WHY IS SOFTWARE VERIFICATION HARD?

Many different quality requirements
Evolving (and deteriorating) structure

- Inherent non-linearity
- Uneven distribution of faults



("desemble information of the second I I THE THE WYNER order1 = order>>4; order2 = order & 15; if (!(fpm = fopen("ppmenc.doc", "wb" #ifdef TRACE fprintf(stderr, " \n Error: Car exit(2); } #endif /\* allocate 'order+1' elements

# WHY IS SOFTWARE VERIFICATION HARD?

- Many different quality requirements
- Evolving (and deteriorating) structure
- Inherent non-linearity
- Uneven distribution of faults



If an elevator can safely carry a load of 1000 kg, it can also safely carry any smaller load



# WHY IS SOFTWARE VERIFICATION HARD?

- Many different quality requirements
- Evolving (and deteriorating) structure
- Inherent non-linearity
- Uneven distribution of faults





If an elevator can safely carry a load of 1000 kg, it can also safely carry any smaller load If a procedure correctly sorts a set of 256 elements, it may fail on a set of 255 or 53 elements, as well as on 257 or 1023

allocate is is used

#### **A TESTING PROGRAM INVOLVES TRADE-OFFS**





# Waterfall Model

(1968)



# We built it!





# Waterfall Model (1968)



# Waterfall Model (1968)

Construction

code test

#### Deployment

delivery support feedback



#### Validation

Ensuring that software has been built according to customer requirements

#### Verification

Ensuring that software correctly implements a specific function





### **VALIDATION AND VERIFICATION**



### VALIDATION



"if a user presses a request button at floor i, an available elevator must arrive at floor i soon"

not verifiable, but validatable

### VERIFICATION



"if a user presses a request button at floor i, an available elevator must arrive at floor i within 30 seconds"

verifiable

## **CORE QUESTIONS**

+When does V&V start? When is it done?

Which techniques should be applied?

+How do we know a product is ready?

+How can we control the quality of successive releases?

+How can we improve development?

# Waterfall Model (1968)





#### **V MODEL**



### **UNIT TESTS**

+Aims to uncover errors at module boundaries

Typically written by programmer herself

+Should be completely automatic ( $\rightarrow$  regression testing)



#### **TESTING COMPONENTS: STUBS AND DRIVERS**



✦A driver exercises a module's functions

A stub simulates notyet-ready modules

 Frequently realized as mock objects

#### PUTTING THE PIECES TOGETHER: INTEGRATION TESTS

+General idea: Constructing software while conducting tests

Options: Big Bang or Incremental Construction



## **BIG BANG APPROACH**

All components are combined in advance

The entire program is tested as a whole



### **BIG BANG APPROACH**

All components are combined in advance

The entire program is tested as a whole



For every failure, the entire program must be taken into account



## **TOP-DOWN INTEGRATION**



- Top module is tested with stubs (and then used as driver)
- Stubs are replaced one at a time ("depth first")
- As new modules are integrated, tests are re-run

Allows for early demonstration of capability

## **BOTTOM-UP INTEGRATION**



 Bottom modules implemented first and combined into clusters

- Drivers are replaced one at a time
- Removes the need for complex stubs

Allows for early demonstration of capability

## **SANDWICH INTEGRATION**



 Combines bottom-up and top-down integration

 Top modules tested with stubs, bottom modules with drivers

Combines the best of the two approaches

#### ONE DIFFERENCE FROM UNIT TESTING: EMERGENT BEHAVIOR

#### Some behaviors are only clear when components are put together

Usually this is identified after the fact, and causes test suites/cases to be refactored.



## WHO TESTS THE SOFTWARE?



Developer

understands the system

but will test gently driven by delivery



Independent Tester

must learn about system

will attempt to break it driven by quality

#### WEINBERG'S LAW



# **EVERYONE IS A TESTER!**



#### Experienced Outsiders and Clients

+Good for finding gaps missed by developers, especially domain specific items

#### Inexperienced Users

+Good for illuminating other, perhaps unintended uses/errors

#### Mother Nature

+Crashes tend to happen during an important client/customer demo...

## **SYSTEM TESTING**



## **SPECIAL KINDS OF SYSTEM TESTING**

#### + Recovery testing

forces the software to fail in a variety of ways and verifies that recovery is properly performed

#### + Security testing

verifies that protection mechanisms built into a system will, in fact, protect it from improper penetration

#### + Stress testing

executes a system in a manner that demands resources in abnormal quantity, frequency, or volume

#### + Performance testing

test the run-time performance of software within the context of an integrated system

## **PERFORMANCE TESTING**

Measures a system's capacity to process a specific load over a specific time-span, usually:

- 1. number of concurrent users
- 2. specific number of concurrent transactions

Involves defining and running operational profiles that reflect expected use



#### **TYPES OF PERFORMANCE TESTING**



1. Load

Aims to assess compliance with non-functional requirements

2. Stress

Identifies system capacity limits

3. Spike

Testing involving rapid swings in load

4. Endurance (or Soak)

Continuous operation at a given load

#### MANY OPTIMIZATIONS ARE POSSIBLE

- For Throughput or Concurrency?
   *Getting the most data processed Greatest number of simultaneous*
- + For Server response time?
- + For Service request round-trip time?
- For Server utilization?
- + For End-User Experience?
- ✦ For Cost?



# **SECURITY TESTING**



#### +Confidentiality

+Information protection from unauthorized access or disclosure

#### Integrity

Information protection from unauthorized modification or destruction

#### Availability

+System protection from unauthorized disruption

### **ACCEPTANCE TESTING**



## **ACCEPTANCE TESTING**



+ Acceptance testing checks whether contractual requirements are met

#### May be incremental

+ Alpha / Beta

+ Work is over when acceptance testing is done

#### HOW DO WE KNOW WHEN A PRODUCT IS READY?



+Let the customer test it :-)

+We're out of time...

neoretically sound and experiment  $v_{\mu}^{e}$  validated statistical e done sufficient testing arguing with 95% confidence that  $f = 1,000 \text{ CP}^{\text{II}}$  type of allure-free operation is  $\geq 0.995$ . This is the type of allure-free operation is  $\geq 0.995$ . Relative to a theoretically sound and experimodel, we have done sufficient test: the probability of 1,000 CPI''

# **REGRESSION TESTS**

- set up automated tests
  using, e.g., JUnit
- ideally, run regression tests after each change
- f running the tests takes too long:
   prioritize and run a subset
  - ★apply regression test selection to determine tests that are impacted by a set of changes



#### **COLLECTING DATA**

Mozilla Vulnerabilities					
security	mailnews	content extensions nsprpub			
nss	base imap	base >	slt xul	canvas3d webservice pyt	hon spellch pr li
lib	src util src	src p	rc temp doc	src soap pro xp	co src src tests t
libpkix freebl softoken		xslt	xpath src src	W i wa	lia uniu laska md
pkix_pl_nss mpi ecl				sche wa	wi uni ma
modu pki sy	search		cont	iava	- sic sic
	src 📕	SV.	events xml	xncom me	t ore ins typ
pkix incl ssl util certd smi	addrbook compose import	html cont	nt src d	xforms	misc pthre
ton uti r	src src outl src	content doc sr	TIBES ytf	xmlterm	, aut w s p thr cp include
		src src	xbl	base line   sql	coo s io md p III
	eud oex		src can	b.	
crmf pki pkcs c			directory		JAIII P
CKIW pk11wr	local news exter	io alue	unectory	colito 2	Compiler Utilitie wizard
builtins ca photo	mime src src palm	giue giue	Ldan	squites	Code Front Coner windows libyone
ns pkil	src		librarian alia	SIC	Code Front Gener Windows inDxpne
certhig bas a	sn bs		libidan		Primi zli qa
	ald ma db b	reflect string typel	b moldap exa		
cmd	t Old Ind	xptcal x pu sr xpi	suncsdk	xp	os2
zlib lib m pk si fips pk ce c	modules	src s i i i i i i i i i i i i i i i i i i	c-sdk	b mork tri	Runtim gc Pack setup unix
	oii plugin	ds base tests	Idap	SrC Sr	Syste sr i
manager jss	s tests sr tools sam		libraries cli i	md md	C N C Tools O Src
org	src test s s def	build comp			Ex D
layout		obsolete	intl	editor	toolkit xpfe
generic style yul	C Arr A C base S	c MoreFi thr p	ucony	libeditor txm	components airbag compone bootstra
generic style hase	Ac		ucvlat src uti u	icy html base	place his s airbag sear boo app
STC STC		widget		com	
	libing libront libpron zlib	src	u u		do pa xre m hi
	png jmcge dec s src	mac gtk2	ucren u	text txtsv	appshel
	src ====		unichar Locala		
tables mathml for	ms Elite libre libr libr		unicital locale c	calendar pa	irser tools accessible
base base	softupd src src src	windows os2 beos	sic sic s	ite ilbical ntmipa	rser expa trace- codes re src
src src	src wml s li		Charder	s SFC SFC	p lib li re d f p atk bas nt xu
	sic libjar	xpwi ot ph			le ip ms
base re prin lin ht			netwerk	au test dou	m measdk cek ac
	gfx		base protoco	bace sr	C C expat muc boebm
	src		src http f	tp Dase si	a protocol
JS	ps xlib mac theb xlib		Src	js2 Dus	driver ib
Sic talilarin		embedding		src	
xpconnect invect core	os2 remoting	browser	streamco test	co re	piugin unioader camino ipc
	gtk windo be xp sh	activex gtk pho		bui other-l	icense work work
	y11ch f	src src	cache dns	s 7zst 1	ibart e
fdlib		co co wet		src src	lib mston view mail
pcre code M	cairo thebe	plu pl powerp at	java	7zi	rdf mac src src com
	cairo glitz src	ض لــــــا(ا تلـدَلـدَنا)	webclient plugg	ab browser	atk-1, base chro huil dhm sun hunh
	src src	compon qa tests	src_moz wf	components	g src d profile win s i stu w
shell pl	e public	printin teste mfc w		places migrat	Sr with St Stu W
	libpixma	win fi	xpcom do p	src book	docshell config caps sto gcon mini
jsd	src publ	web	te ja u jni		base s src sr chro

#### **REMEMBER PARETO'S LAW**



## **CORE QUESTIONS**

+When does V&V start? When is it done?

Which techniques should be applied?

+How do we know a product is ready?

+How can we control the quality of successive releases?

+How can we improve development?

## **BEST PRACTICES**

+Specify requirements in a quantifiable manner

State testing objectives explicitly

 Understand the users of the software and develop a profile for each user category

Develop a testing plan that emphasizes "rapid cycle testing"

## **BEST PRACTICES**

- Build "robust" software that is designed to test itself
- Use effective formal technical reviews as a filter prior to testing
- Conduct formal technical reviews to assess the test strategy and test cases themselves
- Develop a continuous improvement approach for the testing process

### **DESIGN FOR TESTING**

OO design principles also improve testing
 Encapsulation leads to good unit tests

Provide diagnostic methods
 Primarily used for debugging, but may also be useful as regular methods

Assertions are great helpers for testing
 Test cases may be derived automatically