Beta Testing

Beta Testing

- System-level, black-box test of the entire product
- Customers define and execute tests
- Often dominated by a functional approach
  - But may also include usability, reliability, stress & performance
- Almost always manual

Beta Testing: Validation

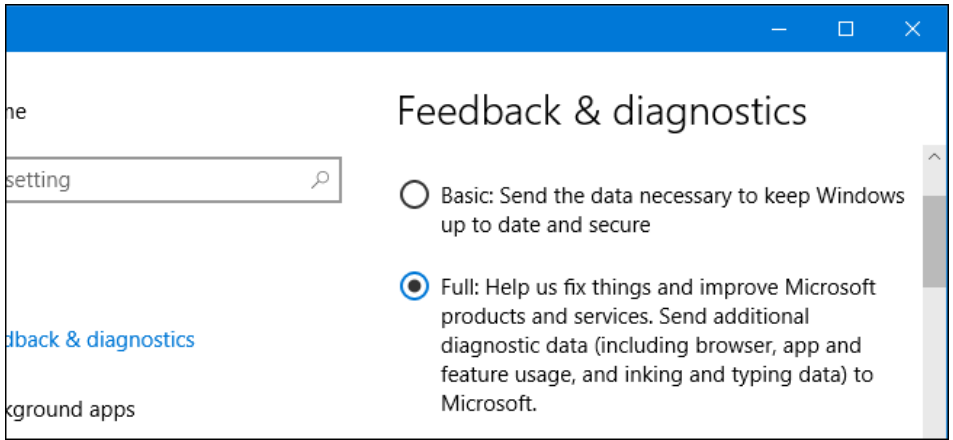
- Other defect removal activities verify that the product meets our specification
  - i.e., it does what the developers expect
- Beta Testing validates the product meets our customers' needs
  - i.e., it does what customers expect
- Beta Testing is invaluable and should be included in CS471/CS481 whenever sponsor is supportive

Beta Testing: Effectiveness [Jones'96]

- Small (<10 users) beta program: 25..40%
- Large (>1000 users) beta program: 60..85%

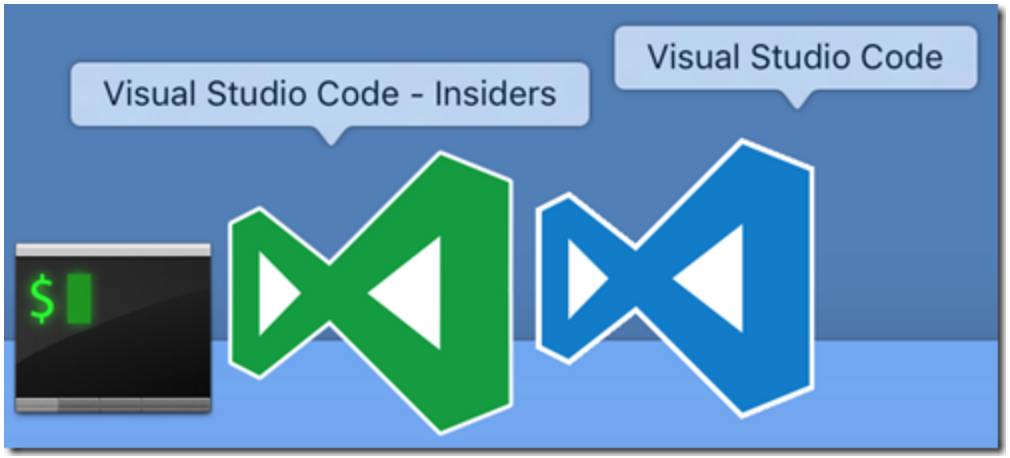
Windows 10

- More than 10,000,000 beta testers



Beta Testing: Large Scale Distribution

- It is not uncommon for companies to make available:
  - beta releases
  - (developer) "previews"
  - "insider" programs
  - "dev" channels



mozilla

Firefox Release Channels

The Mozilla release process delivers new features, performance improvements and security upgrades to users every six weeks. This consists of four development channels producing concurrent releases of Firefox for Windows, Mac, Linux and Android.

Firefox

FOR: Everyone! Released to the more than 400 million Firefox users worldwide

ENJOY: The polished and stable features of Firefox that move the Web forward with great performance and unparalleled customization

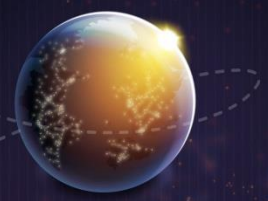
EXPECT: An awesome Web experience that answers to no one but you!

Firefox Beta

FOR: Early adopters and Mozilla fans

ENJOY: Testing the next version of Firefox with stability

EXPECT: Mostly stable builds that need fine tuning and majority add-on compatibility



Firefox Aurora

FOR: Web/platform developers, early adopters and adventure seekers

ENJOY: Access to experimental new features — your feedback helps determine what makes it to Beta

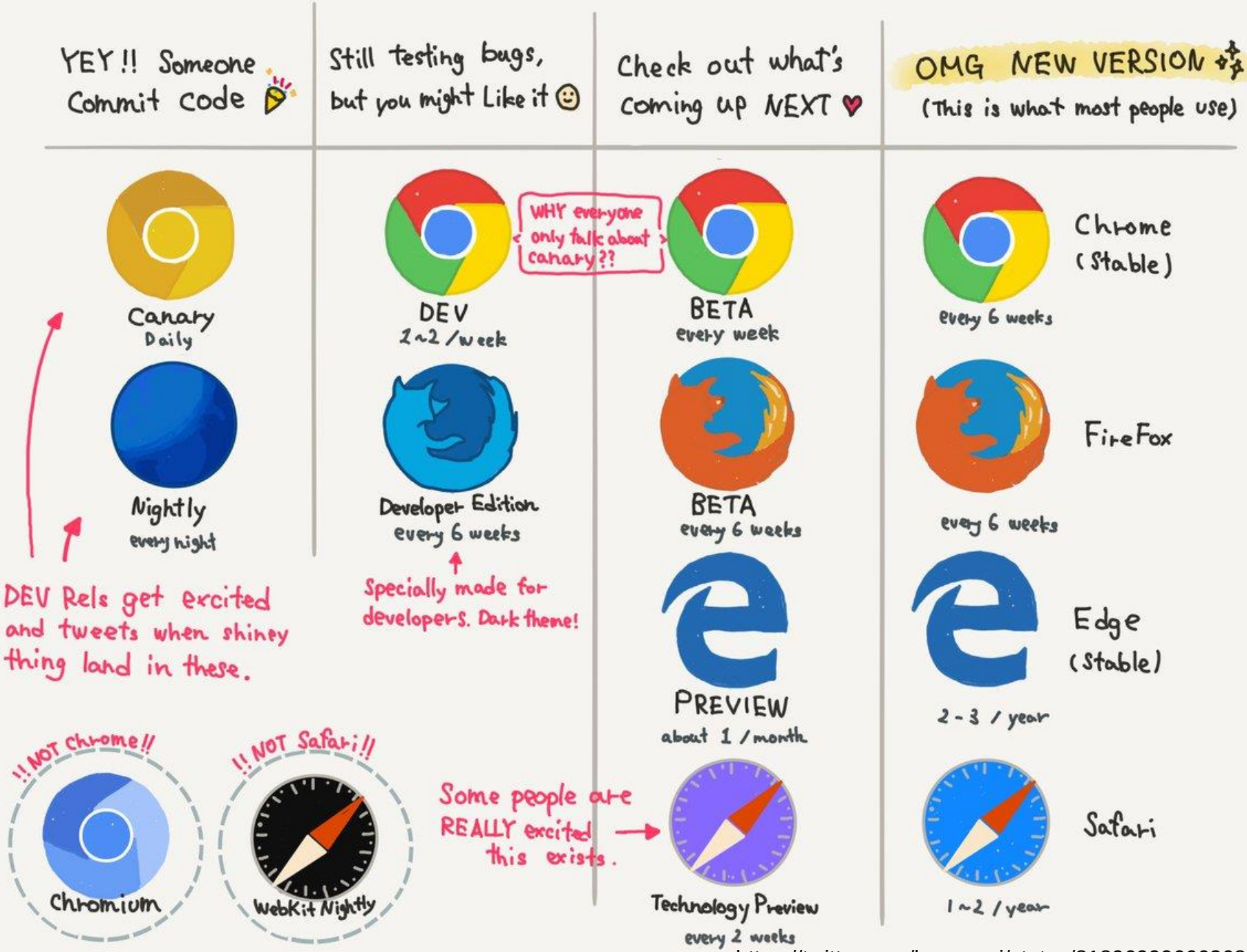
EXPECT: Test builds with bugs and incompatible add-ons

Firefox Nightly

FOR: Platform developers and Mozilla contributors

ENJOY: Access to cutting edge features still under active development

EXPECT: Crashes, unstable test builds with bugs and incompatible add-ons



https://twitter.com/kosamari/status/818969996663681026

Beta Testing: Large Scale Distribution

- staged rollouts: https://www.youtube.com/watch?v=qGoCFoEt_CU

Example Staged Rollout

Google play | Developer Console

Developer Console Demo ▾ developerconsoledemo@gmail.com Sign out Give us feedback

All applications
Financial reports
Settings
Announcements

Statistics
Revenue
Ratings & Reviews
Crashes & ANRs
Optimization Tips

APK
Store Listing
Pricing and Distribution
In-app Products
Services & APIs

FORTUNE TELLER – com.undergroundpowers.fortuneteller

Published ▾

APK Publish now Revert changes

Simple Mode

PRODUCTION
Versions
1

BETA TESTING
Set up Beta testing for your app

ALPHA TESTING
Set up Alpha testing for your app

There is no APK in Beta testing

Upload new APK

OTHER APKS Hide

VERSION	STATUS	ACTIONS
3 (3.0)	Unpublished	Move to Beta
2 (2.0)	Unpublished	Move to Beta
1 (1.0)	in Prod	

APP TRANSLATION SERVICE

Purchase professional translations of your application. Learn more

Example Staged Rollout

Google play | Developer Console

Developer Console Demo developerconsoledemo@gmail.com Sign out Give us feedback

All applications
Financial reports
Settings
Announcements

Statistics
Revenue
Ratings & Reviews
Crashes & ANRs
Optimization Tips

APK
Store Listing
Pricing and Distribution
In-app Products
Services & APIs

Published
Simple Mode

STAGED ROLLOUT FOR PRODUCTION

With a staged rollout, you can publish your app to a certain percentage of the market first to make sure your app works properly before rolling it out to everybody.

Publish this app to

- 0.5%
- 1%
- 5%
- 10%
- 20%
- 50%
- 100%
  full rollout, completely replaces the old configuration

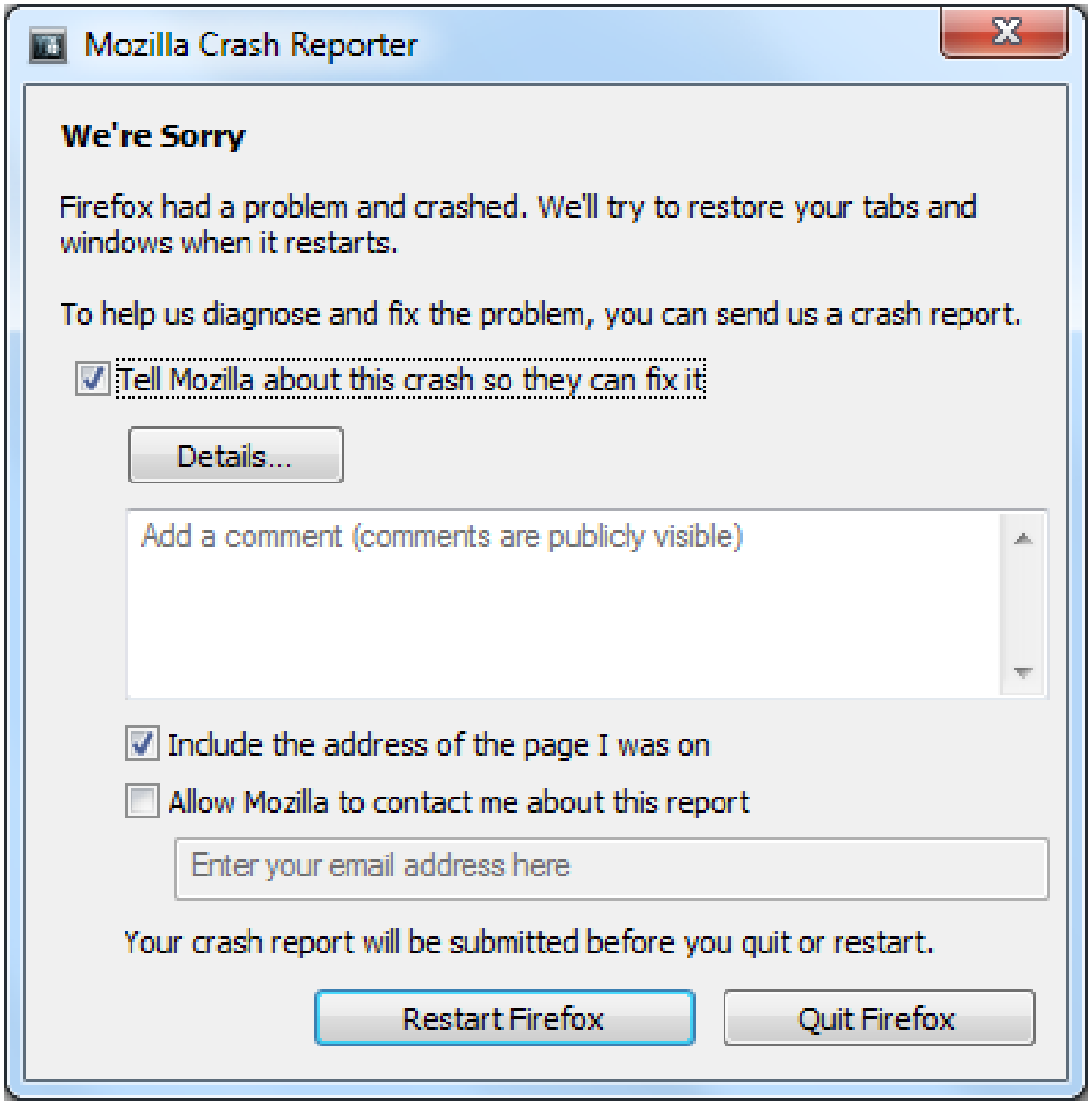
Save and publish | Cancel

APP TRANSLATION SERVICE

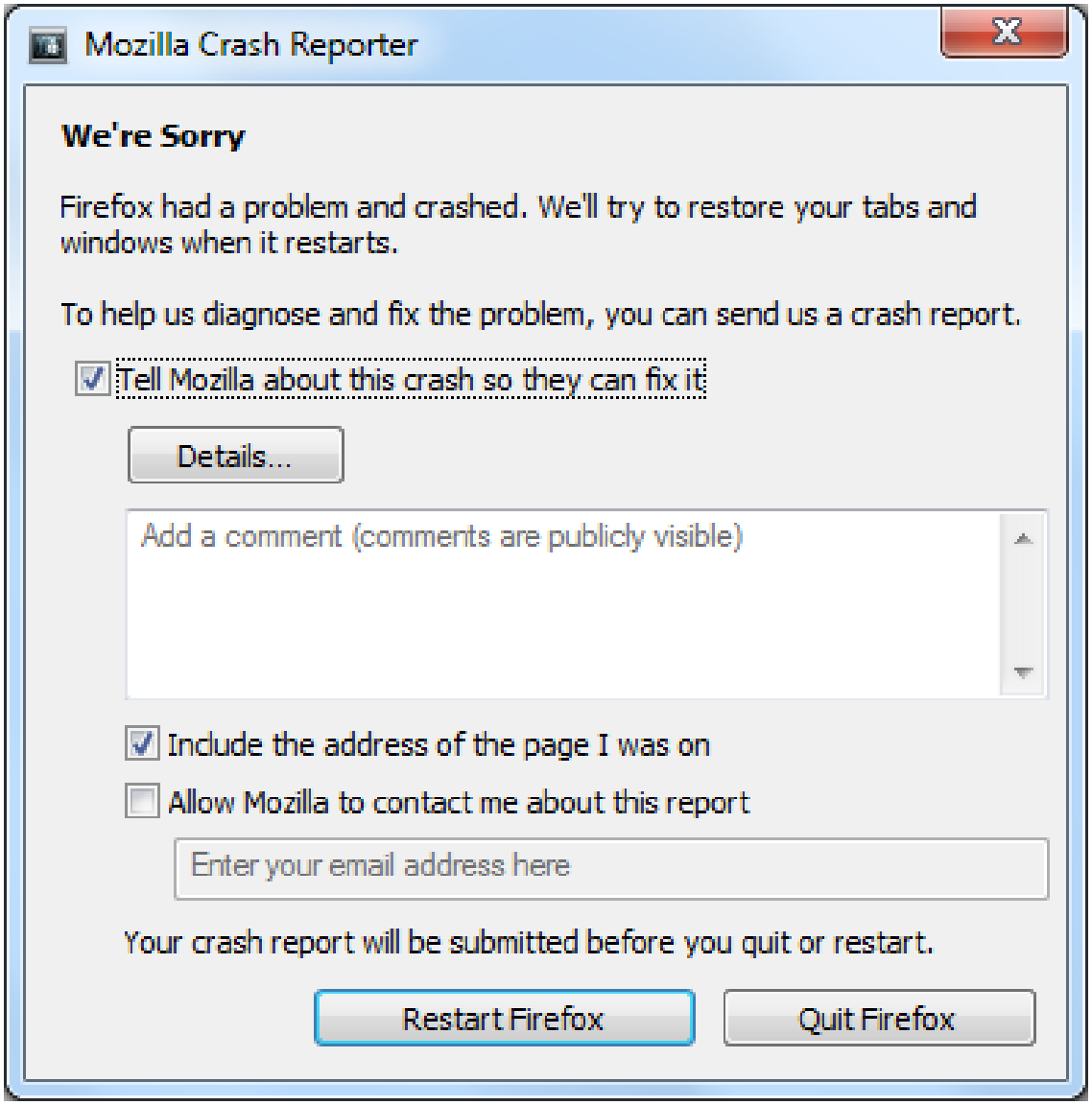
Purchase professional translations of your application. Learn more

How Do Developers Collect “Feedback” from beta testers?

Firefox Nightly

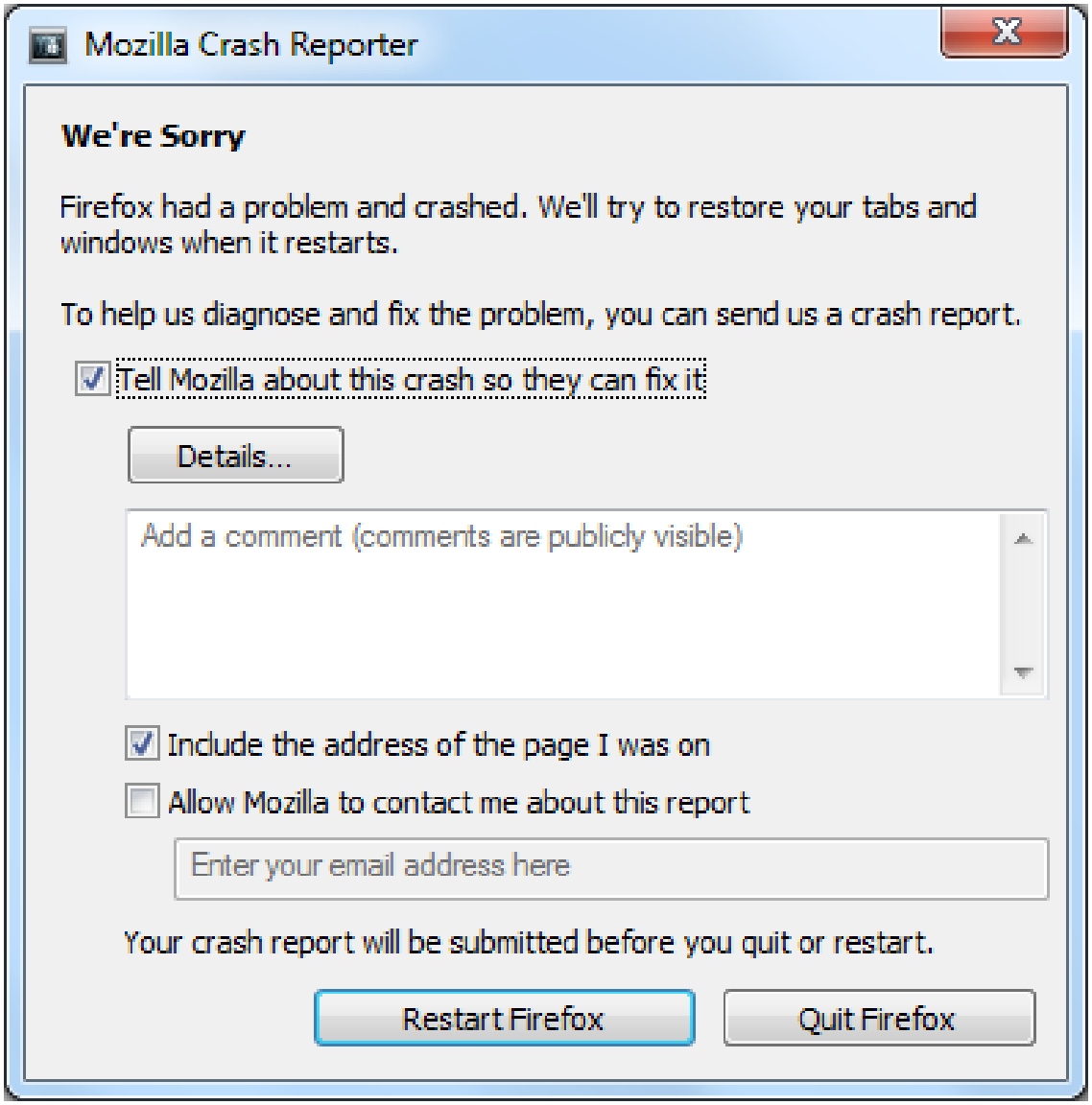


Firefox Nightly

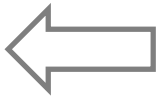


How will developers collect and analyze this data?

Firefox Nightly



Mozilla Crash Reports



How will developers collect and analyze this data?

mozilla crash reports

Find Crash ID or Signature

Product: Firefox | Current Versions | Report: Overview | Super Search

Firefox Crash Data

Crash Reports

Firefox 61.0a1	Firefox 60.0b	Firefox 59.0.2
Top Crashers	Top Crashers	Top Crashers
Top Plugin Crashers	Top Plugin Crashers	Top Plugin Crashers
Top Content Crashers	Top Content Crashers	Top Content Crashers
Top GPU Crashers	Top GPU Crashers	Top GPU Crashers

https://crash-stats.mozilla.com/home/product/Firefox

mozilla crash reports

Find Crash ID or Signature

Product: Firefox | 61.0a1 | Report: Top Crashers | Super Searcl

Top Crashers for Firefox 61.0a1

Top 50 Crashing Signatures. 7 days ago through a few seconds ago.

The report covers 4385 (67.7%) of all 6477 crashes during this period.

Report Date: Most Recent | By Day **Range Type:** By Report Date | By Build Date

Type: Any | Browser | Plugin | Content | GPU **Days:** 1 | 3 | 7 | 14 | 28 **OS:** All | Windows | Mac OS X | Linux

Result Count: 50 | 100 | 200 | 300

Rank	%	Diff	Signature	Count	Win	Mac	Lin	Installs	Is GC	First Appearance	Bugzilla IDs
1	13.4%	new	nsDocShell::AddState	868	868	0	0	440	0	2011-11-15	~~1453572~~ ~~745502~~
2	11.72%	-4.75%	mozilla::ipc::MessageChannel::Close	759	739	17	3	709	0	2015-04-16	~~1453252~~ ~~1446444~~ ~~1446387~~ ~~1433856~~ ~~1424922~~ ~~1405375~~
3	9.08%	-3.53%	EMPTY: no crashing thread identified; ERROR_NO_MINIDUMP_HEADER	588	0	0	0	349	0	2013-11-14	1360392 ~~1279269~~ ~~1255050~~ 1245570 ~~945328~~ ~~837835~~ ~~711568~~ 610551

https://crash-stats.mozilla.com/topcrashers/?product=Firefox&version=61.0a1

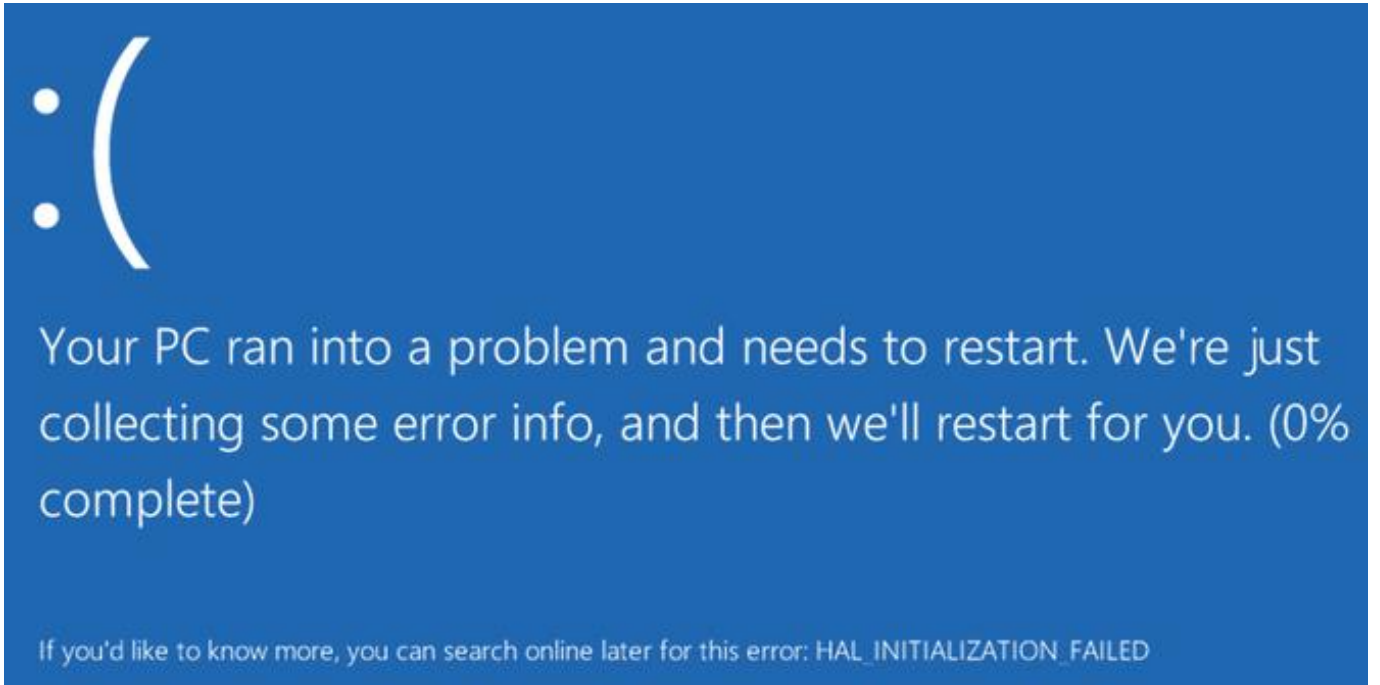
Beta Testing: Overhead

- You will need a formal way to track defect reports from Beta Test users (e.g., Mozilla Crash Reports)
  - Track status of reports
  - Identify duplicate reports
  - Prioritize reports

Beta Testing: Overhead

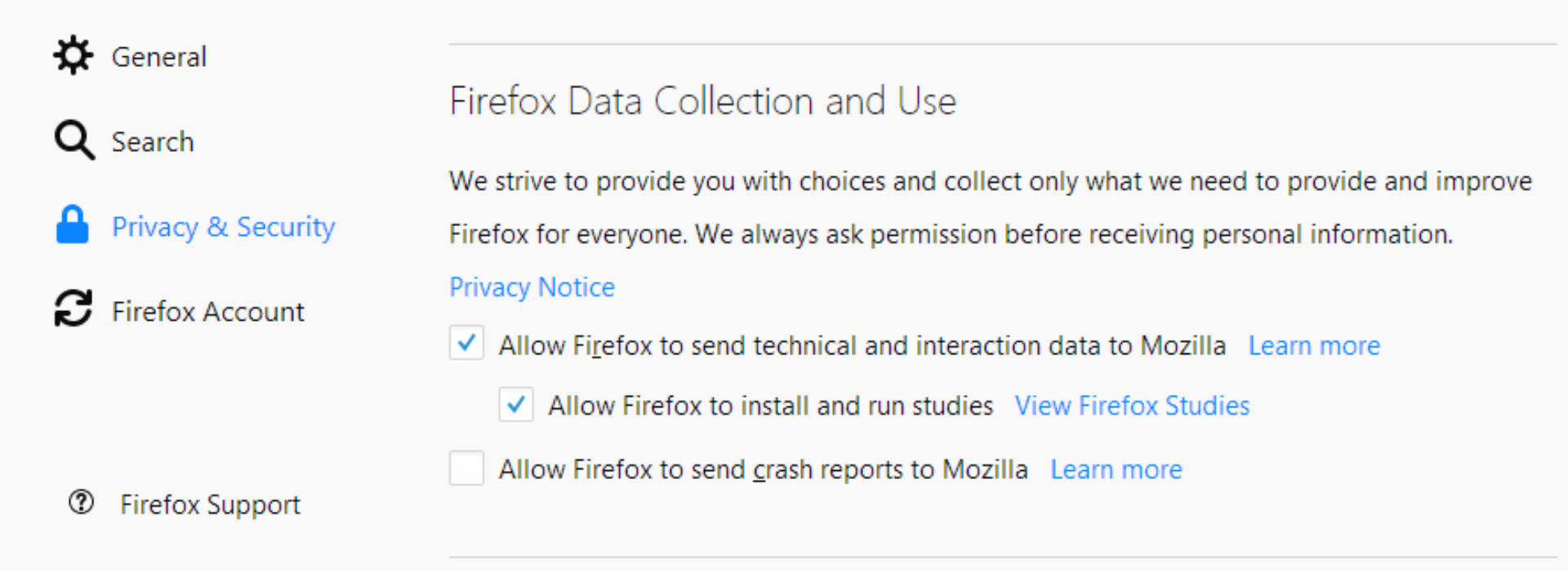
- You will need a formal way to track defect reports from Beta Test users (e.g., Mozilla Crash Reports)
  - Track status of reports
  - Identify duplicate reports
  - Prioritize reports

Microsoft ⇒



Beta Testing: Overhead

- Setup infrastructure to automatically collect & analyze usage statics
  - Can be used to improve software usability

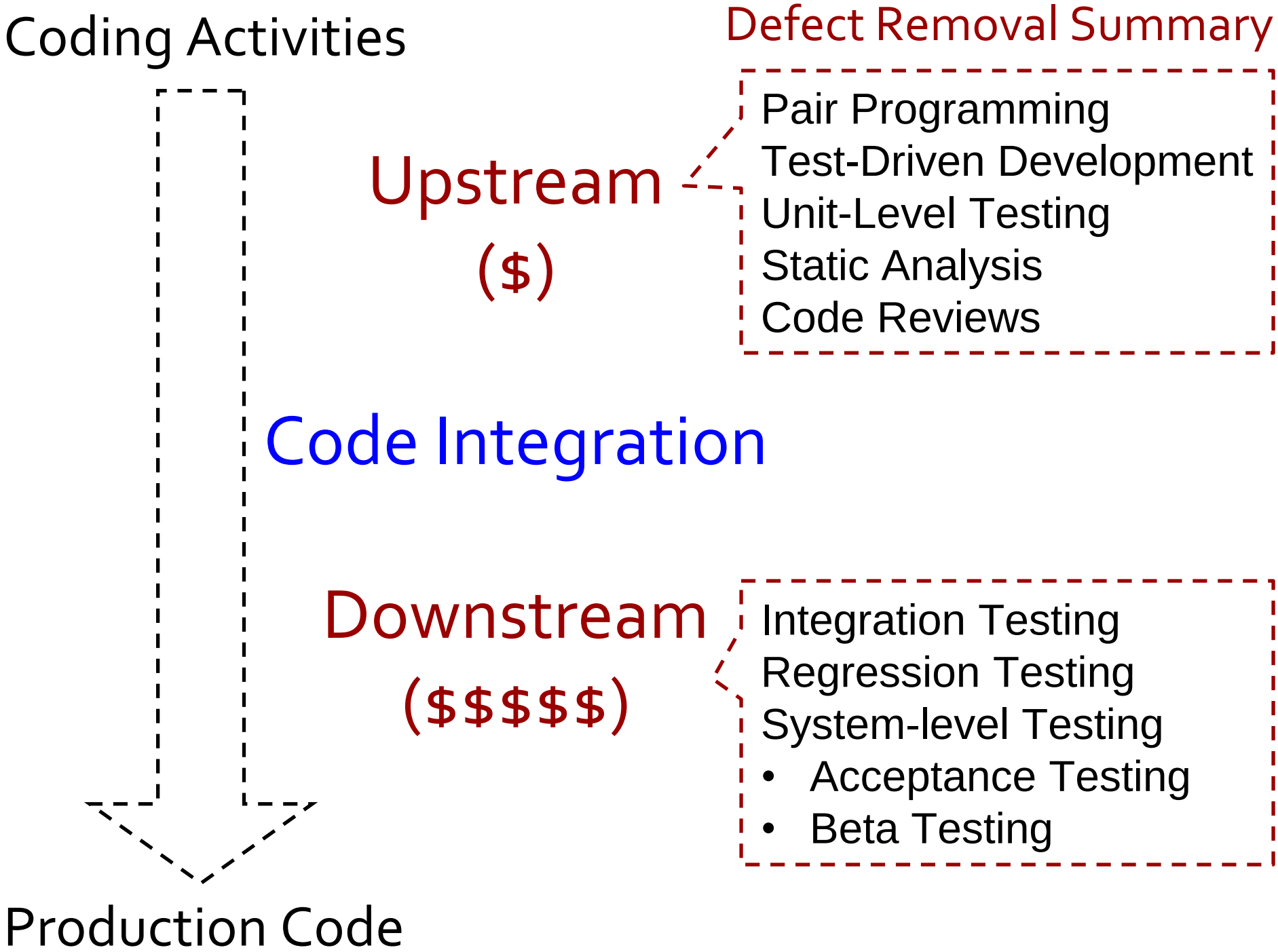


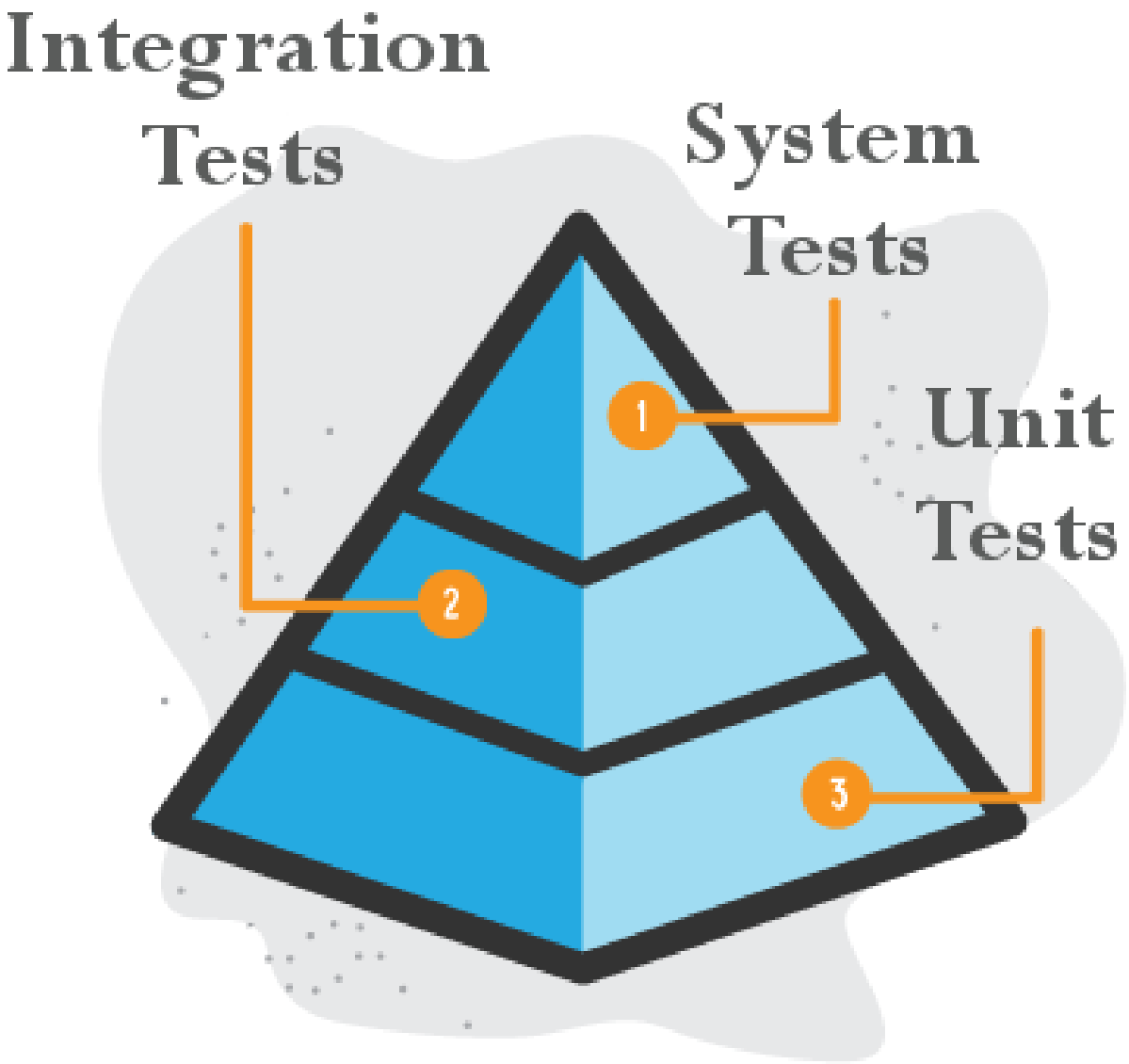
Beta Testing: Warnings

- Very expensive ($$$) to reproduce, debug and repair defects found by Beta Testing

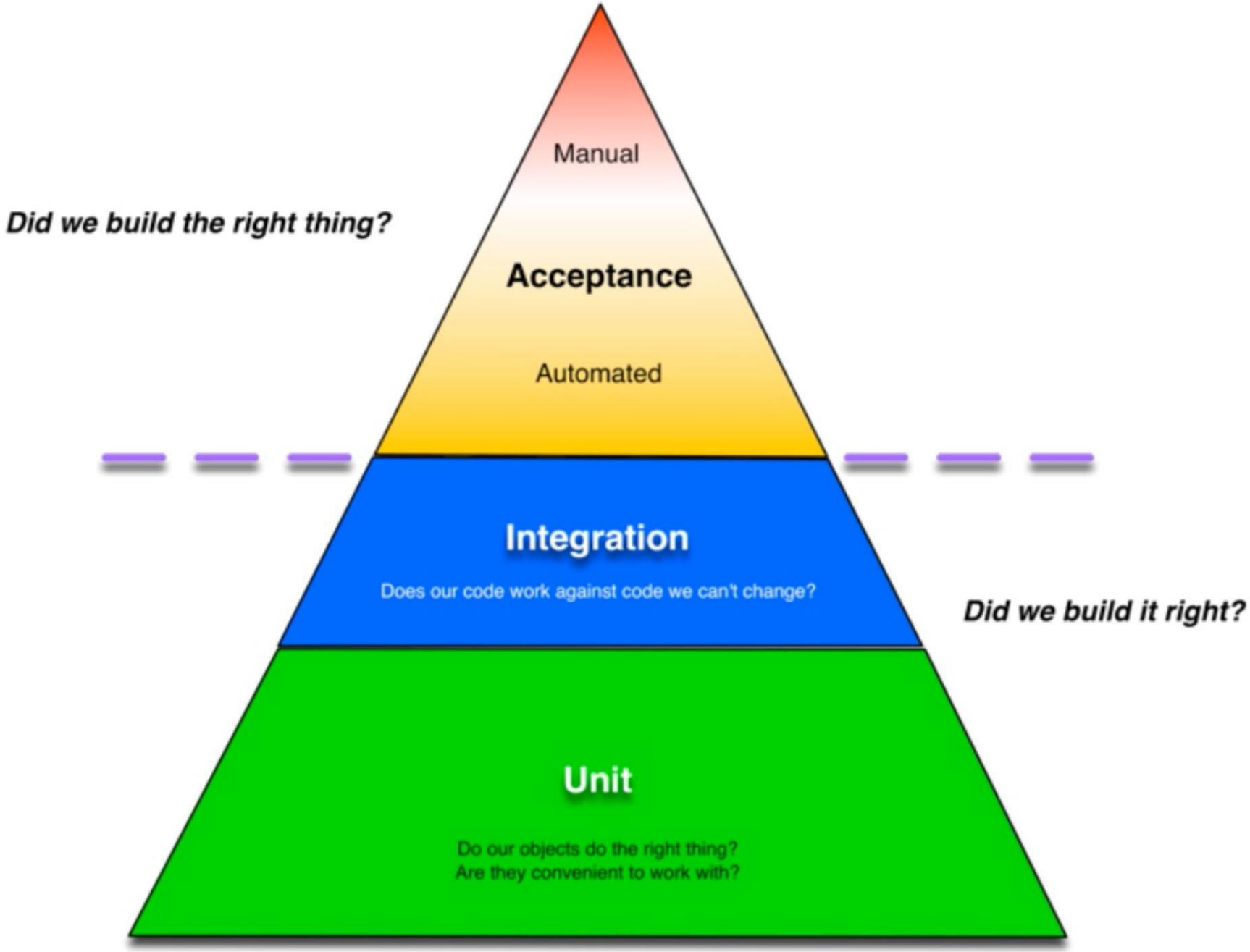
- In CS471/CS481, you may track them as User Stories just like any other customer requests

Defect Removal Summary

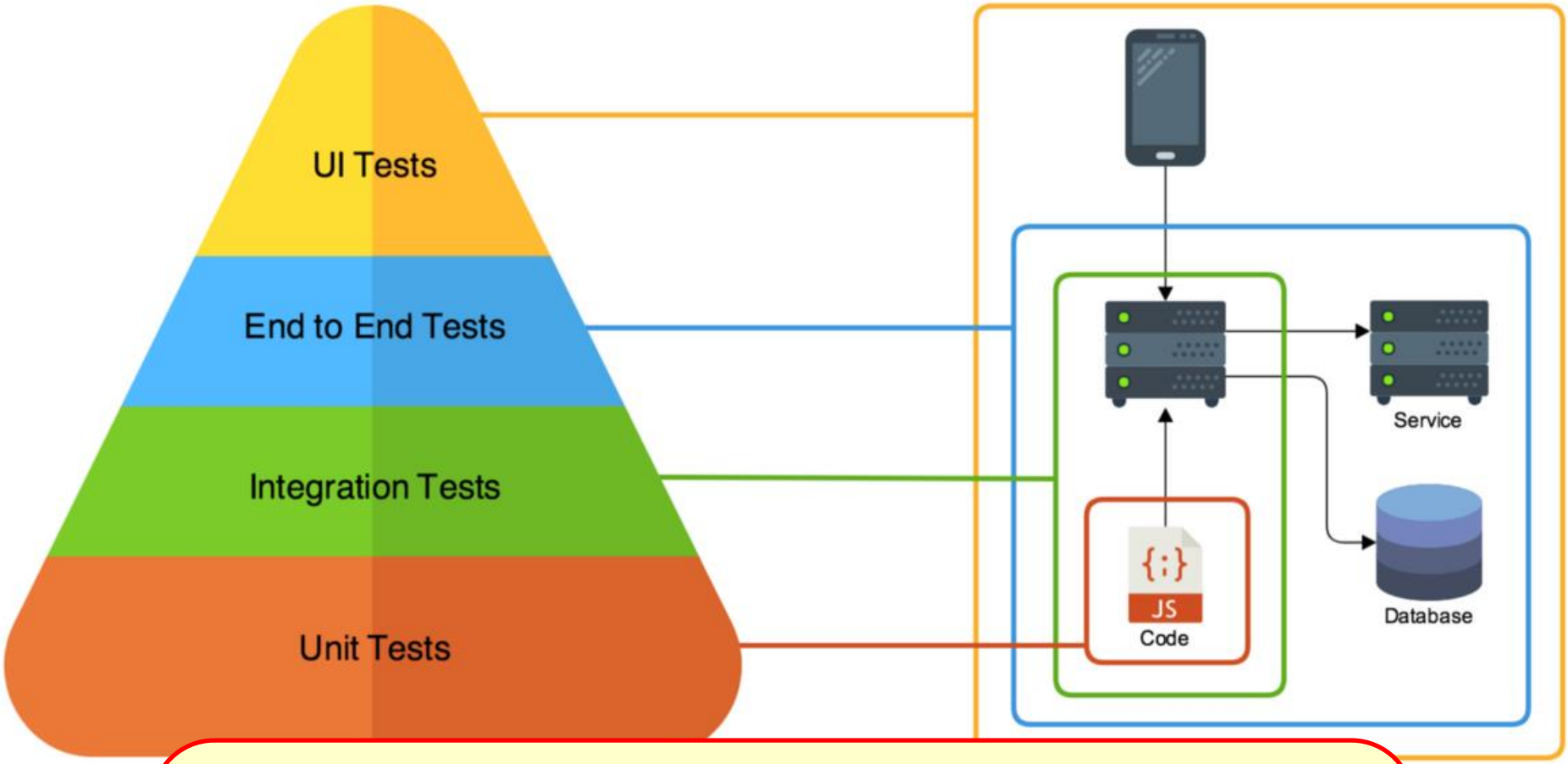

Coding Activities
Defect Removal Summary
Upstream
($)
Pair Programming
Test-Driven Development
Unit-Level Testing
Static Analysis
Code Reviews
Code Integration
Downstream
($$$$$)
Integration Testing
Regression Testing
System-level Testing
• Acceptance Testing
• Beta Testing
Production Code



http://www.labviewcraftsmen.com/blog/do-you-unit-test



https://legacy.gitbook.com/book/gkedge/javascript-acceptance-testing/details



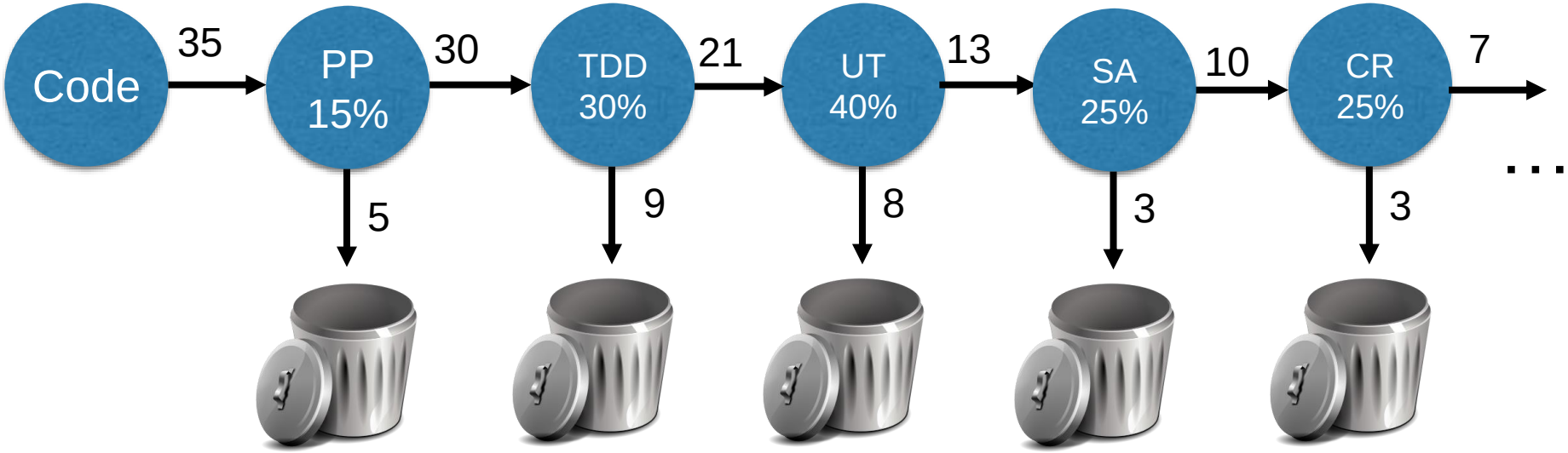
Maximize number of Automated Tests:

- Improves code health
- Allows safe refactoring
- Allows adding features faster

https://medium.com/@nathankpeck/microservice-testing-unit-tests-d795194fe14e

Defect Removal: Summary of Effectiveness

- *Effectiveness* refers to the fraction of defects removed by an activity
- Because no single defect removal activity is 100% effective, we combine them to form a defect removal pipeline



Defect Removal Advice

- NB: Improve overall software quality by choosing complimentary defect removal activities

- Use both White-box and Black-box Approaches
  - White-Box: Pair Programming, TDD, Unit-Level Test, Static Analysis, Code Review, Integration-Level Test
  - Black-Box: Acceptance Test, Beta Test

- Use both Verification and Validation
  - Verification: Almost everything except Beta Testing
  - Validation: Beta Testing

Defect Removal Advice

- Over-optimizing a single activity (e.g., Acceptance Testing) may be more expensive than introducing a new activity (e.g., Pair Programming)

Software Quality at Full Throttle

- Some products, or at least their subsystems, require extreme quality (<1.0 defects/KLOC) achieved through additional state-of-the-art techniques:
  - Requirements, Specification, Design and Test Reviews
  - (Fagan) Formal Code Inspections
  - Proof of Correctness
  - Prototyping
  - Large beta programs
  - Stress/Load/Penetration/Usability Testing
  - Performance/Reliability/Availability Testing
- Quality (in addition to functionality) drives their designs

Software Quality at Full Throttle Example

- The NASA Space Shuttle Primary Avionics Software System:
  - 0.06 defects/KLOC (i.e., around 1 defect every 16,000 LOC)
  - (so low it was difficult to measure definitively)

Quality Plan

What is a *Quality Plan*?

- A *Quality Plan* is an engineering document defining:
  - The project's quality goal (e.g., 4.5 defects/KLOC)
  - Defect removal model
    - contains defect removal activities chosen to achieve the goal

What is a *Quality Plan*?

- A *Quality Plan* is an engineering document defining:
  - The project's quality goal (e.g., 4.5 defects/KLOC)
  - Defect removal model
    - contains defect removal activities chosen to achieve the goal

Activity	Input Defects/KLOC	Effectiveness	Removed Defects/KLOC	Output Defects/KLOC
Programming	0.0	0	0.0	35.0
Pair Programming	35.0	15%	5.3	29.8
TDD	29.8	25%	7.4	22.3
Unit-Level Tests	22.3	30%	6.7	15.6
Static Analysis	15.6	15%	2.3	13.3
Integration Testing	13.3	25%	3.3	10.0
System-Level Testing	10.0	40%	4.0	6.0
Beta Testing	6.0	25%	1.5	4.5

What is a *Quality Plan*?

- A *Quality Plan* is an engineering document defining:
  - The project's quality goal (e.g., 4.5 defects/KLOC)
  - Defect removal model
    - contains defect removal activities chosen to achieve the goal

- Most development teams cannot afford to implement all the defect removal activities we've discussed in class
  - But how will you choose those activities?
  - Why will a project use some activities but not others?

Quality Planning: Two Approaches

- A waterfall-like approach uses conservative up-front planning to ensure success

- An agile-like "Lean Development" approach
  - creates an initial plan,
  - evaluates its performance in each sprint, and
  - corrects it as necessary in the following sprint

- In CS471/CS481, we will focus on a Lean Development approach :)

Applying Lean Development: Plan, Do, Check, Act (PDCA or Deming Cycle)

- Plan: Formulate a plan for the upcoming sprint to deliver the project's quality goal

- Do: Implement the plan in the upcoming sprint

- Check: At the end of the sprint, measure the performance of the plan
  - did it deliver the quality goal or not?

- Act: In the Sprint Retrospective Meeting, adjust the plan to correct anomalies

Planning Assumptions: In Lieu of Specific Data for Our Team

- 2003Kan: Defect injection rate ≈35 defects/KLOC
- 2000Cockburn: Pair Programming Effectiveness ≈15%
- 2011Williams: Test Driven Development ≈25%
- 1996Jones: Unit-Level Testing ≈30%
- 2011Conrad: Static Analysis ≈15% (Note: Customer-visible)
- 2003Kan: Code Review ≈50%
- 1996Jones: Continuous Integration Testing ≈25%
- 1996Jones: Acceptance (System-Level) Testing ≈40%
- 1996Jones: (Low Volume) Beta Test ≈25%

Bibliography

- [1996Jones] Jones, Capers. "Software defect-removal efficiency." IEEE *Computer* 29.4 (1996): 94-95.
- [2000Cockburn] Cockburn, A. & Williams, L. "The costs and benefits of pair programming." *Extreme Programming Examined*. 2000.
- [2003Kan] Kan, Stephen. *Metrics and Models in Software Quality Engineering 2nd Edition*. Addison-Wesley. 2003.
- [2011Conrad] Conrad, J. Technical Report. 2011.
- [2011Williams] Williams, L., Brown, G., Meltzer, A., & Nagappan, N. "Scrum+ engineering practices: Experiences of three microsoft teams." In *Empirical Software Engineering and Measurement (ESEM), 2011 International Symposium* on (pp. 463-471). IEEE. 2011.

Beyond Functional Testing

Beyond Functional Testing

- *Functional tests* exercise a minuscule fraction of the product's overall functionality

- Solution:
  - Code Coverage
  - Equivalence Partitioning / Boundary Value Analysis

Code Coverage

- *Code Coverage* (AKA *Test Coverage*) = fraction of the product's code actually exercised by a test suite

- Code Coverage "guides" Structural/White-box Testing in an attempt to exercise "all" of the code structure:
  - Every statement
  - Every conditional clause
  - Every method

Code Coverage Example (Java, Eclipse, Emma)

```
public boolean addAll(int index, Collection c) {
    if(c.isEmpty()) {
        return false;
    } else if(_size == index || _size == 0) {
        return addAll(c);
    } else {
        Listable succ = getListableAt(index);
        Listable pred = (null == succ) ? null : succ.prev();
        Iterator it = c.iterator();
        while(it.hasNext()) {
            pred = insertListable(pred, succ, it.next());
        }
        return true;
    }
}
```

1 of 2 branches missed.

Press 'F2' for focus

http://www.eclemma.org/userdoc/annotations.html

EclEmma Code Coverage Plugin for Eclipse

Java - CursorableLinkedList.java - Eclipse SDK

File Edit Source Refactor Navigate Search Project Run Window Help

JUnit

Finished after 34,898 seconds

Runs: 13009/13009 Errors: 0 Failures: 0

Failures | Hierarchy

- junit.framework.TestSuite
  - junit.framework.TestSuite
    - TestBagUtils
    - org.apache.commons.collections.TestClos
    - org.apache.commons.collections.TestColle
    - TestBufferUtils
    - TestEnumerationUtils
    - org.apache.commons.collections.TestFact
    - TestListUtils
    - TestMapUtils
    - org.apache.commons.collections.TestPrec
    - TestSetUtils
    - org.apache.commons.collections.TestTrar
    - TestArrayStack
    - TestBeanMap
    - org.apache.commons.collections.TestBina
    - TestBoundedFifoBuffer
    - TestBoundedFifoBuffer2
    - TestCursorableLinkedList
    - TestDoubleOrderedMap
    - org.apache.commons.collections.TestExte
    - TestFastArrayList
    - TestFastArrayList1
    - TestFastHashMap
    - TestFastHashMap1
    - TestFastTreeMap
    - TestFastTreeMap1

Failure Trace

CursorableLinkedList.java

```
public boolean addAll(int index, Collection c) {
    if(c.isEmpty()) {
        return false;
    } else if( size == index ||  size == 0) {
        return addAll(c);
    } else {
        Listable succ = getListableAt(index);
        Listable pred = (null == succ) ? null : succ.prev();
        Iterator it = c.iterator();
        while(it.hasNext()) {
            pred = insertListable(pred,succ,it.next());
        }
        return true;
    }
}
```

Problems | Javadoc | Declaration | Console | Coverage

TestAllPackages (31.10.2006 15:04:14)

Element	Coverage	Covered Lines	Total Lines
java - commons-collections	79,5 %	10927	13738
org.apache.commons.collections	74,1 %	3842	5183
ArrayStack.java	86,5 %	32	37
BagUtils.java	86,7 %	13	15
BeanMap.java	72,4 %	155	214
BinaryHeap.java	87,6 %	127	145
BoundedFifoBuffer.java	93,2 %	82	88
BufferOverflowException.java	55,6 %	5	9
BufferUnderflowException.java	88,9 %	8	9
BufferUtils.java	30,8 %	4	13
ClosureUtils.java	93,9 %	31	33
CollectionUtils.java	92,4 %	293	317
ComparatorUtils.java	8,6 %	3	35
CursorableLinkedList.java	85,4 %	444	520

Writable | Smart Insert | 149 : 28

Code Coverage Example (lisp)

Coverage report: /tmp/test1.lisp

Kind	Covered	All	%
expression	17	29	58.6
branch	6	10	60.0

Key

Not instrumented
Conditionalized out
Executed
Not executed

Both branches taken
One branch taken
Neither branch taken

```
1  (declaim (optimize sb-cover:store-cover
2
3  (defun test (n)
4    (when (zerop n)
5      (if (eql n 0)
6          (print 'zero)
7          (if (eql n 0.0)
8              (print 'single-fp-zero)
9              (print 'double-fp-zero))))
10   (when (minusp n)
11     (print 'negative))
12   (when (plusp n)
13     (tagbody
14        (print 'positive)
15        (go end)
16        (print 'dummy)
17        end)))
18
19 (test 0)
20 (test 1)
21
```

https://www.snellman.net/blog/archive/2007-05-03-code-coverage-tool-for-sbcl.html

Code Coverage Example Report (JavaScript)

JSCover

Summary | Source | About

☐ Show missing statements column

File	Statements	Executed	Branches	Branches Hit	Functions	Functions Hit	Coverage	Branch	Function
Total: 23	1032	350	8436	1023	1781	439	33%	12%	24%
/experiences/healthcare/js/healthcare/config.js	2	2	0	0	0	0	100%	N/A	N/A
/experiences/healthcare/js/healthcare/confighandler.js	24	11	12	5	4	2	45%	41%	50%
/experiences/healthcare/js/healthcare/throttlingService.js	3	3	0	0	1	0	100%	N/A	0%
/experiences/healthcare/js/healthcare/dependencies/ga_custom_code.js	249	144	180	60	41	22	57%	33%	53%
/experiences/healthcare/js/healthcare/test-constructor.js	30	27	26	11	4	3	90%	42%	75%
/experiences/healthcare/js/healthcare/dependencies/detectmobilebrowser.min.js	1	1	6	0	2	0	100%	0%	0%
/experiences/healthcare/js/healthcare/dependencies/ieFixes.js	62	10	66	9	11	0	16%	13%	0%
/experiences/healthcare/js/healthcare/dependencies/select2.min.js	2	1	1282	110	250	52	50%	8%	20%
/experiences/healthcare/js/healthcare/dependencies/underscore.min.js	1	1	496	71	151	32	100%	14%	21%
/experiences/healthcare/js/healthcare/dependencies/handlebars.min.js	2	2	570	154	165	66	100%	27%	40%
/experiences/healthcare/js/healthcare/dependencies/backbone.min.js	1	1	556	117	117	51	100%	21%	43%
/experiences/healthcare/js/healthcare/dependencies/backbone-validation.min.js	1	1	168	4	58	6	100%	2%	10%
/experiences/healthcare/js/healthcare/dependencies/backbone.localStorage-min.js	1	1	70	25	23	11	100%	35%	47%
/experiences/healthcare/js/healthcare/dependencies/backbone.layoutmanager.js	316	32	186	11	63	4	10%	5%	6%
/experiences/healthcare/js/healthcare/dependencies/jquery.autotab.min.js	11	1	256	6	20	1	9%	2%	5%
/experiences/healthcare/js/healthcare/dependencies/placeholders.jquery.min.js	1	1	164	3	31	4	100%	1%	12%
/experiences/healthcare/js/healthcare/dependencies/mediator.js	132	30	102	6	24	4	22%	5%	16%
/experiences/healthcare/js/tthealth.min-63b54fa1b3e21ac0bbfb7fddf56fc844.js	5	2	2494	88	488	46	40%	3%	9%
/experiences/devmode/js/toolbar.js	101	32	42	3	22	5	31%	7%	22%
/experiences/healthcare/js/healthcare/dependencies/ga_invoke.js	84	44	76	26	20	15	52%	34%	75%
/js/widget/webengage-min-v-4.0.js	1	1	1684	314	286	115	100%	18%	40%
/gz.js	1	1	0	0	0	0	100%	N/A	N/A
/webengage-files/webengage/76aa436/v3.js	1	1	0	0	0	0	100%	N/A	N/A

https://innovatorsdelight.wordpress.com/2014/05/05/collecting-javascript-code-coverage-during-a-selenium-automation-test-run/

Code Coverage Example Report + Visualization

#808 | Job: **Default Job** was successful

Job Summary | Tests | Changes | Artifacts | Logs | Clover | Metadata | Agent Performance | Build Times | Issues | Run | Actions

Clover Code Coverage

View history

Clover Coverage
Clover Coverage Report

Dashboard
Coverage Reports
Coverage (Aggregate)
Test Code (Aggregate)
Test Results

Application Packages

com.atlassian.clover *(22%)*
com.atlassian.clover.ant *(42.5%)*
com.atlassian.clover.ant.groovy *(12.7%)*
com.atlassian.clover.api *(50%)*
com.atlassian.clover.api.ci *(80.4%)*
com.atlassian.clover.api.instrumentatior
com.atlassian.clover.api.optimization (5
com.atlassian.clover.api.registry *(100%)*
com.atlassian.clover.ci *(23%)*

Classes | **Tests** | **Results**

Class

AboutAction
AboutDialog
AbstractAntLogger
AbstractCancellableTaskDelegate
AbstractCategory
AbstractClassesScopeAction
AbstractCloverPluginConfig
AbstractCloverPluginConfig.PropertyCha
AbstractCloverTask
AbstractColumnInfo
AbstractCoverageMonitor
AbstractEditorProvider
AbstractExpirableTaskDelegate
AbstractHasMetricsColumnInfo
AbstractHasMetricsNodeComparator
AbstractInlineConfigPanel
AbstractInstrTask
AbstractReportAction
AbstractTestCaseNodeComparator
AbstractTestLayoutAction
AbstractTestViewScopeAction
AbstractToggleInclusionAction
ActionUtils
ActiveEditorListener

Clover Coverage Report - Clover Coverage
Coverage timestamp: Sat Mar 16 2013 06:08:37 CDT

Overview | Package | File

FRAMES | NO FRAMES | SHOW HELP

Statistics for project Clover database Sat Mar 16 2013 06:09:38 CDT:

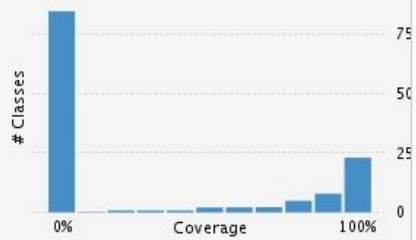
Stmts:	36,891	**LOC:**	114,683	**Total cmp:**	17,933	**Stmts/Method:**	3.76
Branches:	12,318	**NCLOC:**	87,545	**Cmp density:**	0.49	**Methods/Class:**	6.7
Methods:	9,811	**Files:**	1,138	**Avg method cmp:**	1.83	**Classes/Pkg:**	10.69
Classes:	1,464	**Packages:**	137				

0.3% of code in this project is excluded from these metrics. Remove Filter

Coverage 1,464 classes, 19,948 / 59,020 elements
33.8%

Test Results 690 / 699 tests 313.66 secs
98.7%

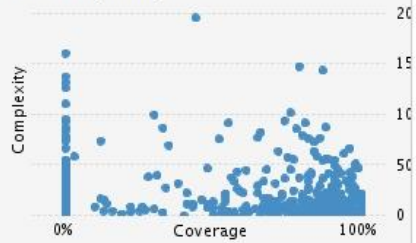
Class Coverage Distribution



Top 20 Project Risks

CoverageEdgeComparator BaseCoverageNodeViewer
OptimizedTestRunListener CloverProperties BrowserLaunch
FixedSourceRegion.RegionStartComparator RegexpValidator
DocMarkupPlugin.HighlightMarkupBuilder ProjToLeafPkgFragRelationship
ProjectTreeCellRenderer ModelUtils CloverExcludableAdapterFactory AggregatingFilter
ProjToTreePkgFragNodeRelationship ProjToPkgFragRootRelationship AboutAction
WizardToCloverReportConfigConverter MergeReportJob.MergeAndReport
AbstractHasMetricsNodeComparator AbstractTestCaseNodeComparator

Class Complexity



Coverage Tree Map

Most Complex Packages

1. 78% com.cenqua.clover.registry (1189)
2. 88.1% com.atlassian.clover.instr.java (657)

Least Tested Methods

1. 0% ColumnSelectionDialog.createDialogArea(Composite) : Control (1)
2. 0% LineCoverageModelImpl.LineCoverageModelImpl(CloverDatabase,FileInfo) (24)
3. 0% PDFReporter.processArgs(String[]) : CloverReportConfig (35)
4. 0% BaseCoverageNodeViewer.setNode(HasMetrics,TestPassInfo) : void (9)

https://confluence.atlassian.com/display/BAMBOO55/Viewing+the+Clover+code-coverage+for+a+build

Code Coverage Measurement

- Coverage is measured by tools, usually with an IDE's support

- Easily measured during automated testing
  - Unit Tests
  - Integration Tests

Coverage Tools

- Java: Clover, EclEmma, JCov, JaCoCo, Cobertura
- C/C++: gcov
- C#: clover.NET, Cover
- Perl: Devel::Cover
- Python: pylid, trace.py
- Swift: Xcode 7

Code Coverage Assumption

Q:What is the relation between Test Effectiveness and Code Coverage?

A: ?

Code Coverage Assumption

Q:What is the relation between Test Effectiveness and Code Coverage?

A: Test Effectiveness $\propto$ Code Coverage

- i.e., higher code coverage $\Rightarrow$ higher test effectiveness

Code Coverage Assumption

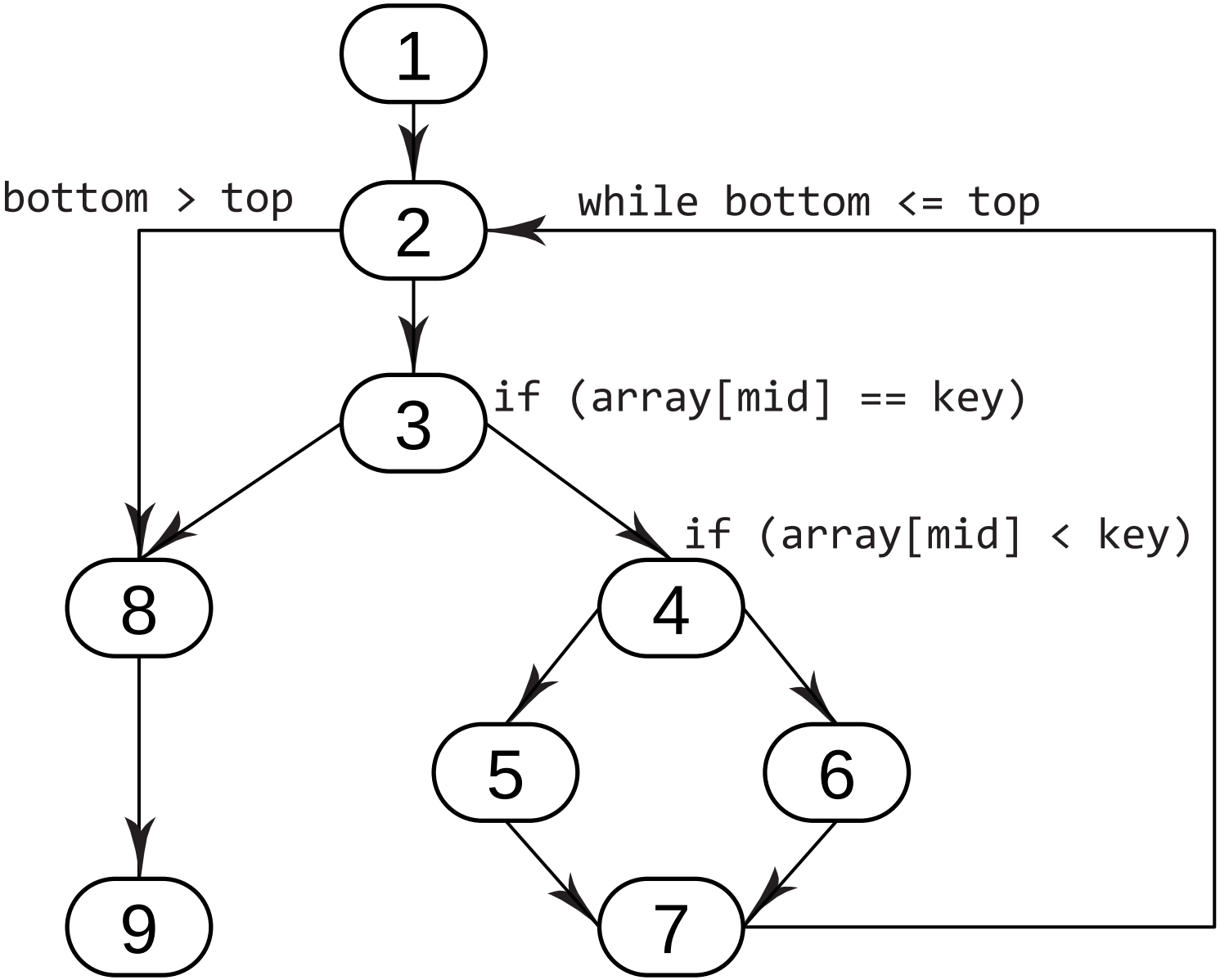
Q:What is the relation between Test Effectiveness and Code Coverage?

A: Test Effectiveness $\propto$ Code Coverage

- i.e., higher code coverage $\Rightarrow$ higher test effectiveness

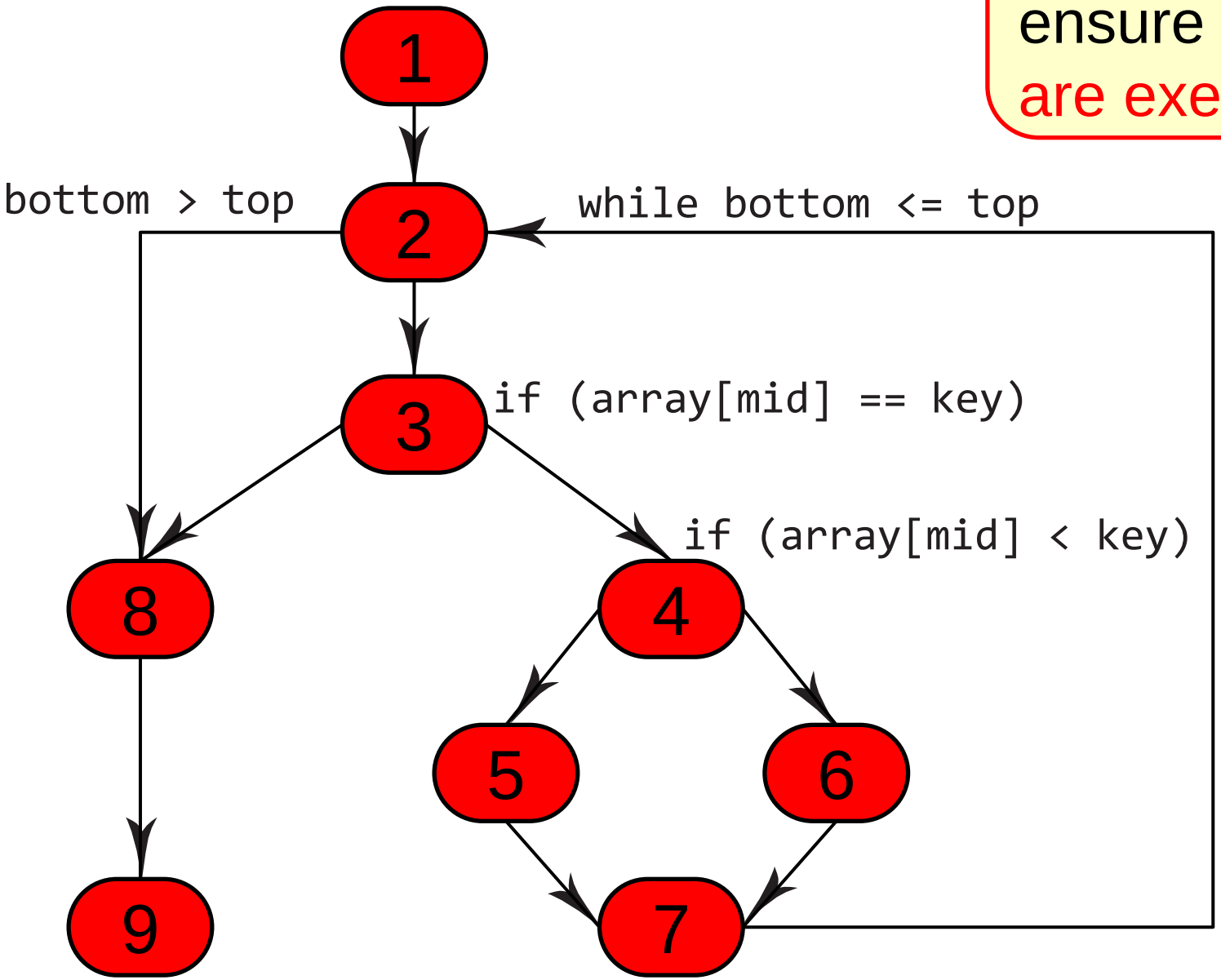
- How do we measure code coverage?

Binary Search Flow Graph

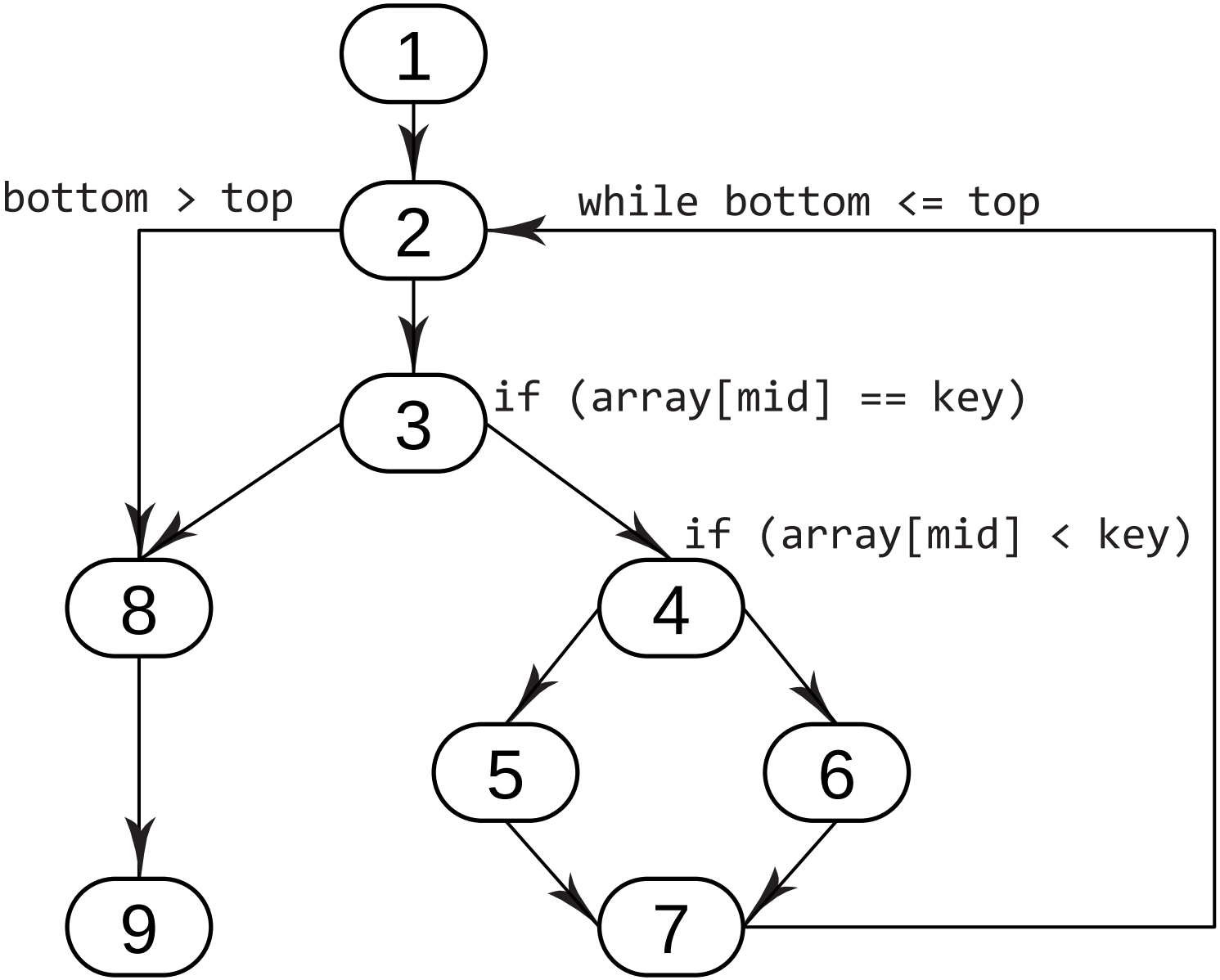


Binary Search Flow Graph

Block coverage (test cases should ensure all blocks are executed)

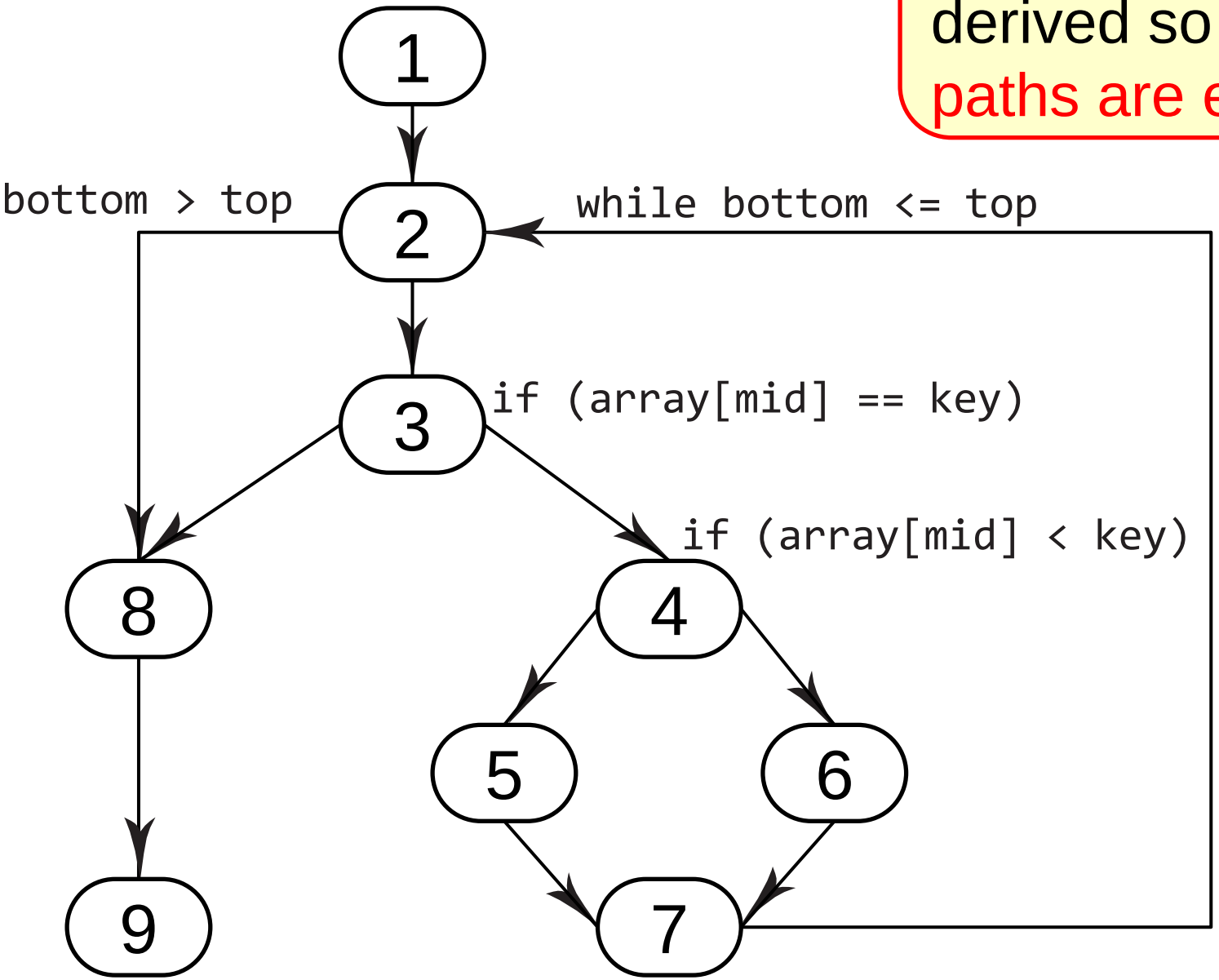


Binary Search Flow Graph



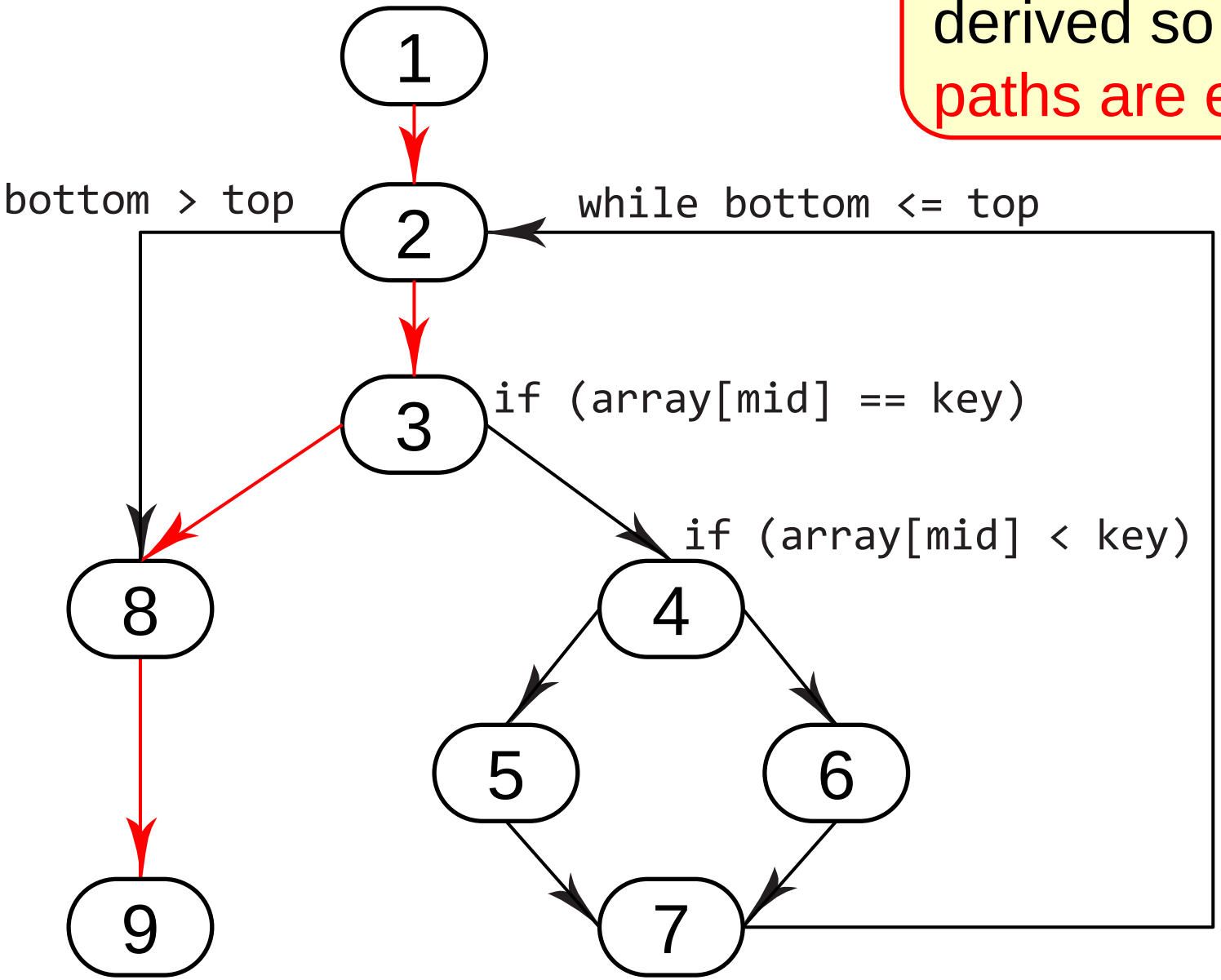
Binary Search Flow Graph

Path coverage (test cases should be derived so that all paths are executed)



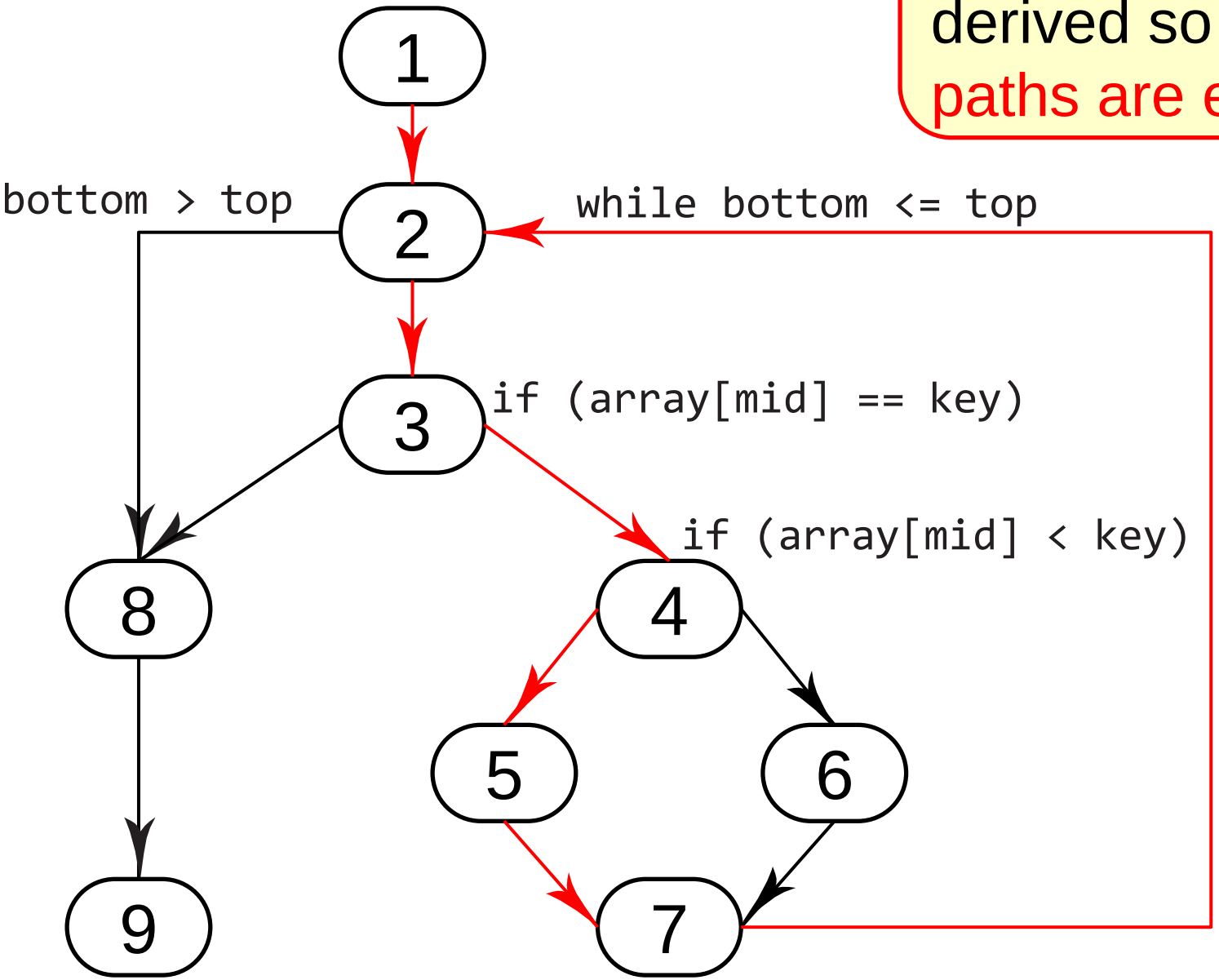
Binary Search Flow Graph

Path coverage (test cases should be derived so that all paths are executed)



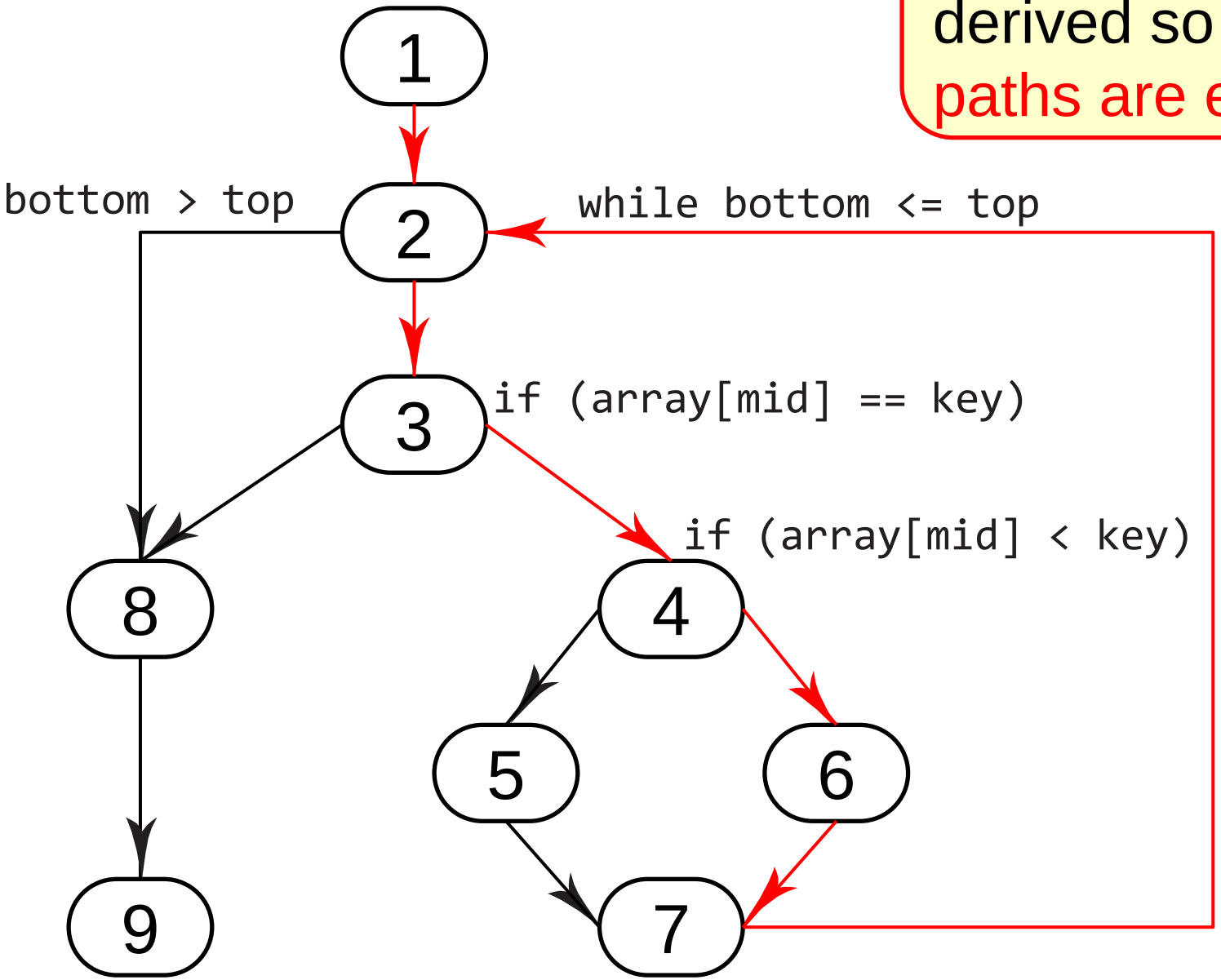
Binary Search Flow Graph

Path coverage (test cases should be derived so that all paths are executed)



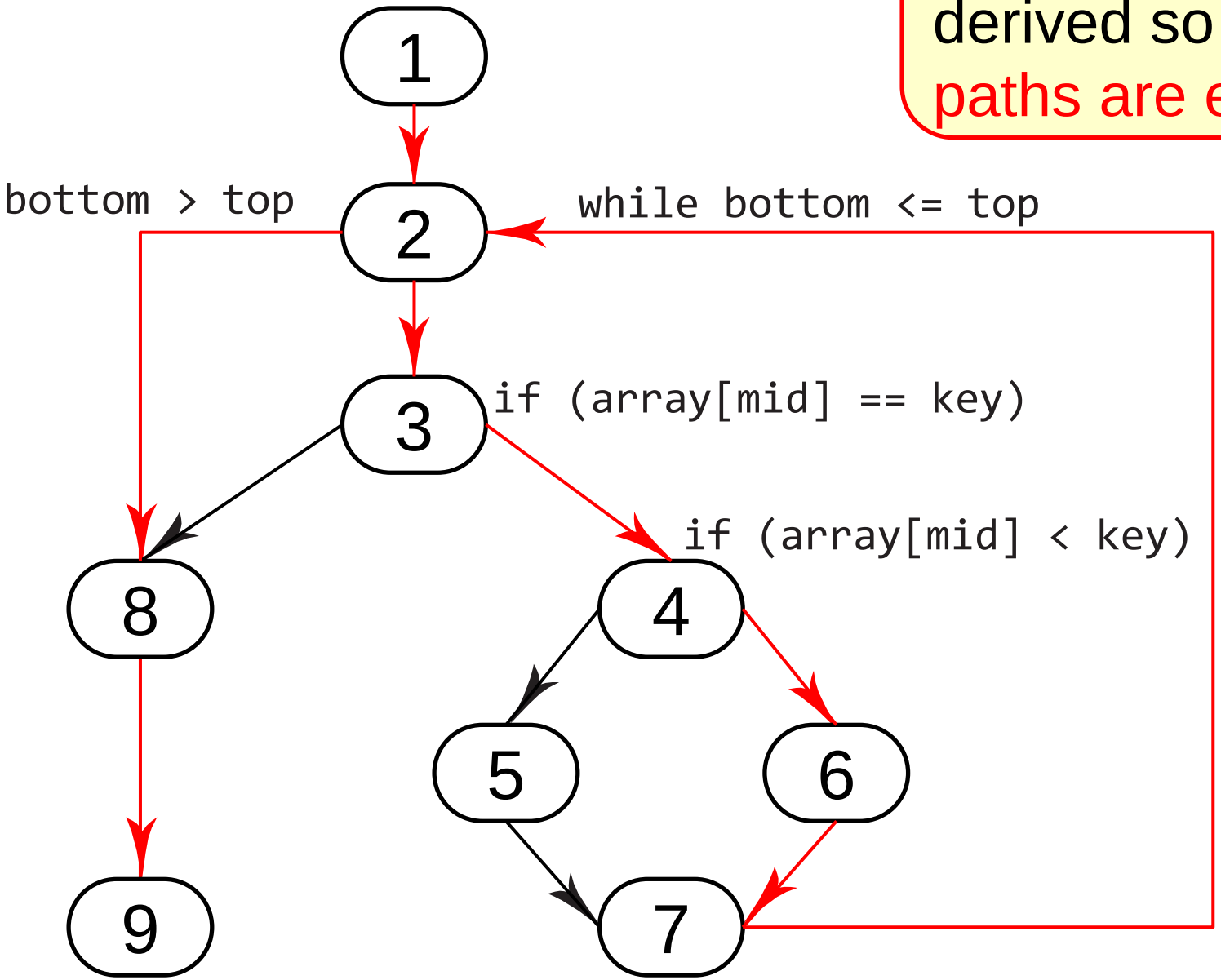
Binary Search Flow Graph

Path coverage (test cases should be derived so that all paths are executed)



Binary Search Flow Graph

Path coverage (test cases should be derived so that all paths are executed)



Code Coverage Measurement: Block vs. Path Coverage

- Block (or statement) coverage (test cases should ensure the following blocks are executed):
  - `1, 2, …, 9`
- Path coverage (test cases should be derived so that all of these paths are executed):
  - `1, 2, 3, 8, 9`
  - `1, 2, 3, 4, 6, 7, 2`
  - `1, 2, 3, 4, 5, 7, 2`
  - `1, 2, 3, 4, 6, 7, 2, 8, 9`
  - A dynamic program analyzer may be used to check that paths have been executed

Statement (Block) Coverage

- Measures the fraction of the source statements (or code blocks) exercised by the test suite
- Most tools provide an overall report
- Many tools provide a report for each class or method

Example Code Coverage Report – EMMA

EMMA Coverage Report (generated Thu Apr 06 14:08:35 MDT 2017)

[all classes]

OVERALL COVERAGE SUMMARY

name	class, %	method, %	block, %
all classes	98% (401/411)	44% (2046/4609)	47% (34552/73201)

OVERALL STATS SUMMARY

```
total packages:          8
total executable files:  278
total classes:           411
total methods:           4609
```

COVERAGE BREAKDOWN BY PACKAGE

name	class, %	method, %	block, %
net.fortuna.ical4j.transform	0% (0/2)	0% (0/4)	0% (0/74)
net.fortuna.ical4j.model.property	100% (127/127)	38% (824/2169)	33% (9945/30161)
net.fortuna.ical4j.model.parameter	94% (47/50)	34% (240/714)	35% (3442/9897)
net.fortuna.ical4j.model.component	100% (58/58)	38% (247/643)	46% (5428/11923)
net.fortuna.ical4j.util	100% (20/20)	82% (78/95)	72% (1351/1881)
net.fortuna.ical4j.filter	67% (4/6)	68% (13/19)	72% (218/301)
net.fortuna.ical4j.data	94% (17/18)	88% (89/101)	74% (2274/3086)
net.fortuna.ical4j.model	98% (128/130)	64% (555/864)	75% (11894/15878)

[all classes]

EMMA 2.1.5320 (stable) (C) Vladimir Roubtsov

Path Coverage

- Measures the fraction of paths through the decisions statements (e.g., switch/case, if/else, etc.) in the source code

Path Coverage

- Measures the fraction of paths through the decisions statements (e.g., switch/case, if/else, etc.) in the source code

- A module having n decisions may have 2^n paths through it!
  - Loop traversals aggravate the problem

Path Coverage

- Measures the fraction of paths through the decisions statements (e.g., switch/case, if/else, etc.) in the source code

- A module having n decisions may have 2^n paths through it!
  - Loop traversals aggravate the problem

- Path Coverage is usually infeasible for overall system
  - may be feasible for small components/units

Code Coverage in Practice

- Required practice in some applications:
  - Certified avionics*
  - Automotive**

- Is so simple that it's almost irresponsible to not measure coverage for at least some of the test suite

* DO-178C, *Software Considerations in Airborne Systems and Equipment Certification*
** ISO 26262, *Road Vehicles — Functional Safety*

How Much Code Coverage is Enough?

- Safety-critical applications may require 100% coverage
- 2012Fowler: "…expect… in the upper 80s or 90s…"
- 1992Grady reports 80% coverage is readily achievable
- Teams who don't measure coverage may achieve <50%

- Very high coverage is expensive and will not find many new defects

Beyond the Overall Coverage Number

- Software quality experts rarely dwell on overall coverage
  - …Unless it's really bad (like… <50%)!!!

- The real value of coverage analysis* is to

"…find which bits of your code aren't being tested"

*Fowler

Beyond Functional Testing

- Code Coverage
- Equivalence Partitioning
- Boundary Value Analysis