

# Software Quality

SE401: Software Quality Assurance and Testing

# Outline

- Factors in Project Success & Failure
- A Case Study, The Initial Launch of HealthCare.gov
- Software Reliability
- The Spectrum of Software Quality
- Cost of Software Defects
- Software Verification and Validation (V&V)
- Software Testing in Development Life Cycle

# Factors in Project Success & Failure

# Software Crisis

- Many software-related failures: auto-pilot systems, air traffic control systems, banking systems, IRS.
  - On January 15, 1990, the AT&T long-distance telephone network broke down, interrupting long-distance telephone services in US for over 8 hours. [Missing **break** in a **switch** statement.]
  - On June 4, 1996, the maiden flight of the new and improved Ariane 5 rocket exploded 37 seconds after lift-off.
  - On June 8, 2001, a software problem caused the NYSE to shut down the entire trading floor for over an hour.
  - On May 27, 2017, a software problem caused a disruption for 75,000 passengers in British Airways
  - February 2020: Heathrow disruption, More than 100 flights were disrupted after it was hit by technical issues
  - Many, many, many more.

# What is the problem?

## Software Projects have a terrible track record

A 1995 Standish Group study (CHAOS) [see notes] found that only 16.2% of IT projects were successful in meeting scope, time, and cost goals (on-time & on-budget) [Things have improved a bit since.]

Over 31% of IT projects were canceled [never seeing completion], costing over \$81 billion in the U.S. alone

- They never worked

- Too late for the market window

Most projects are

- Late in delivery

- Missing functionality

- Have major defects (bugs)

- Did not do what the customer wanted

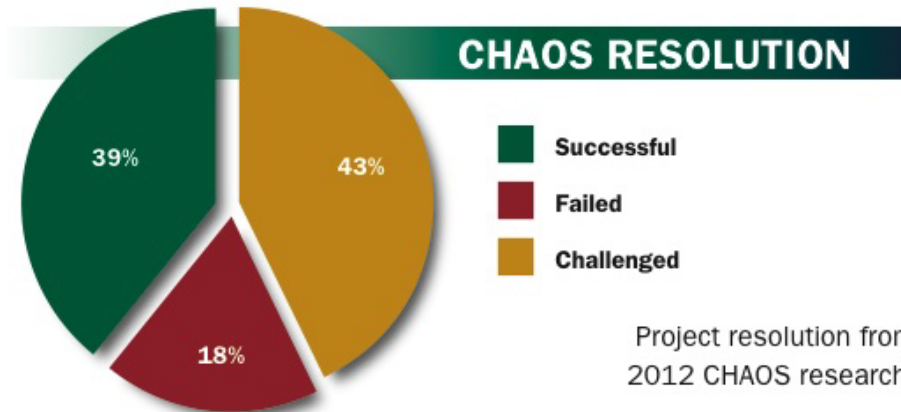
- Hard to maintain and support

# Chaos Report 2012

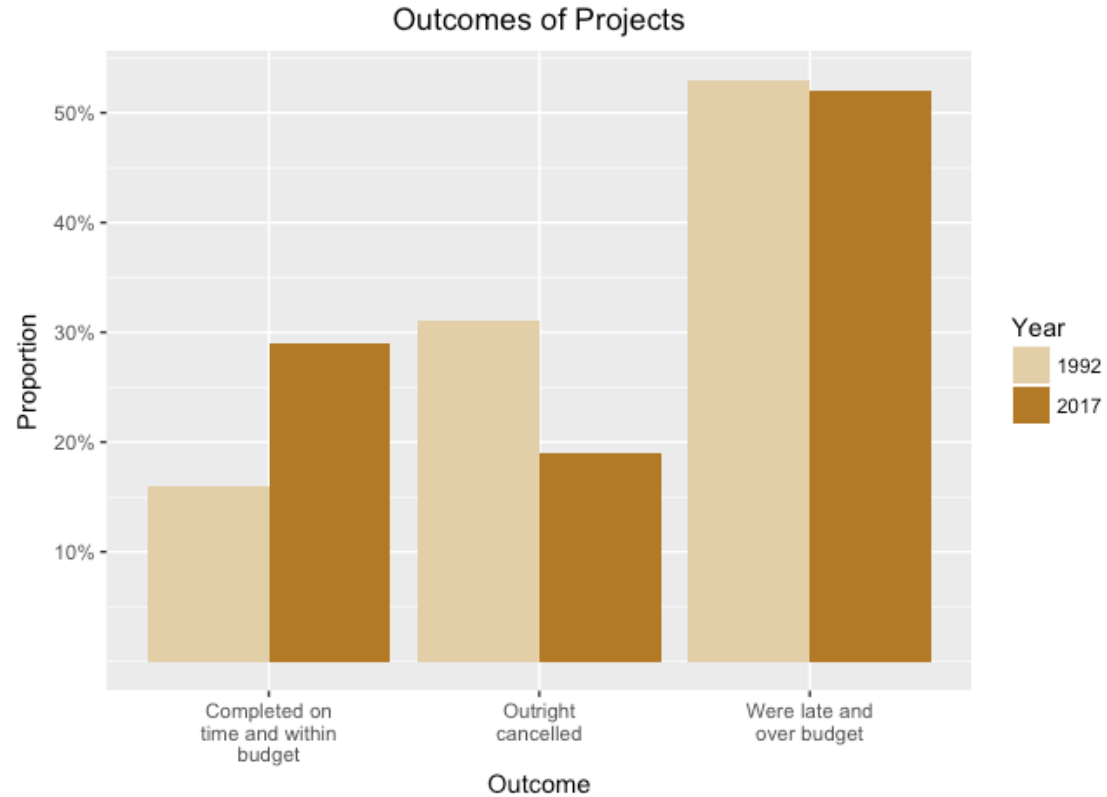
**Project Success:** Type 1. The project is completed on-time and on-budget, with all features and functions as initially specified. **(2012: 39%)**

**Project Challenged:** Type 2. The project is completed and operational but over-budget, over the time estimate, and offers fewer features and functions than originally specified. **(2012: 43%)**

**Project Impaired:** Type 3.  
The project is canceled at some point during the development cycle. **(2012: 18%)** (Are ALL impaired projects failures???)



# 1992-2017





# A Case Study

## The Initial Launch of HealthCare.gov (2013)



# ACA – HealthCare.gov

- ACA signed into law on March 23, 2010
- HealthCare.gov is a healthcare exchange website.
  - “One-stop shopping sites for health insurance”
  - CBO forecast: 7 million users during the first year
- Development contracts awarded in September 2011
  - No-bid, cost-plus contracts
  - Pre-certified private contractors
- HealthCare.gov launched on October 1, 2013
  - Serious technological problems

# HealthCare.gov – The Launch Problems

- Performance: response time (landing page) > 8s
  - “Maddeningly long wait times”
- Navigation: broken UI
- Stability: intermittent crashes, availability  $\approx$  43%
- Functionality: incorrect and incomplete data
- Error rate (per page)  $\approx$  6%
- Scalability: < 1,100 concurrent users
- Enrollment completion rate < 30%

# HealthCare.gov – The Contractors & The Cost

- The lead contractor: CGI Group
  - At least 47 private companies involved
  - Including QSSI, Equifax, Serco
- Coordinated by the Centers for Medicare and Medicaid Services (CMS)
- Total budget: \$293 million
  - CGI: \$196 million (2013). \$112 million paid Oct. 2013
  - QSSI: \$85 million
- Estimated actual cost: > \$500 million by Oct. 2013

# HealthCare.gov – The Failures – Software Eng.

- **Inadequate Testing**

- “This system just wasn't tested enough.” – CMS
- Full test began T -2 weeks (time before launch).
- Final “pre-flight checklist” T -1 week: 41 of 91 functions fail.
- No “end-to-end” test as late as T -4 days
- Stress tests T -1 day: performance degradation with only 1,100 concurrent users. (50,000-60,000 expected)
- Final top-to-bottom security tests not finished.
- No integration test. No beta test.

# HealthCare.gov – The Failures – Software Eng.

- **Evolving, Rolling Requirements**

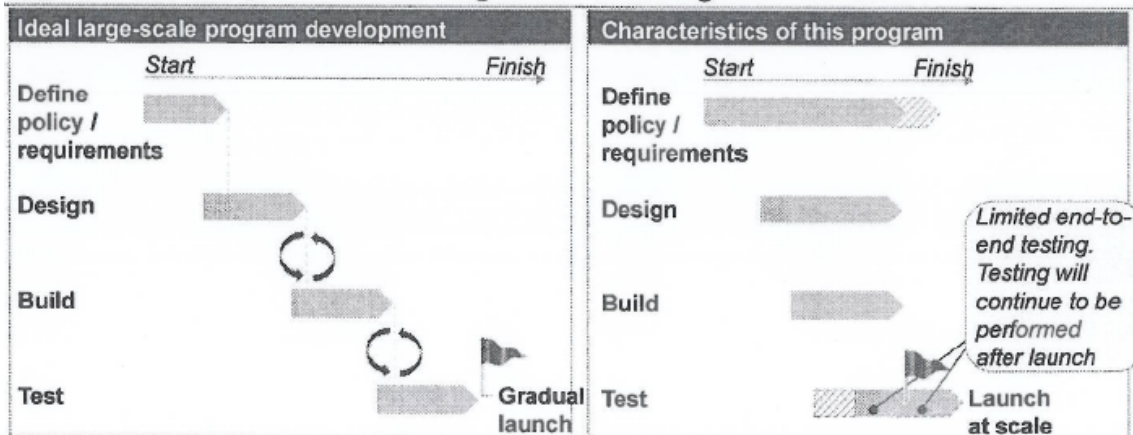
- Regulations and policies were still in flux when contracts awarded in 2011.
- The specifications for the project were delayed repeatedly.
- The regulations and policies were modified repeatedly until summer 2013.
- Repeated changes result in design changes.
- CGI did not start coding until Spring 2013

- **Failure to Effectively Manage Changes**

- “Write-down-all-the-requirements-then-build-to-those-requirements”
- Did not adopt an agile development approach.
- Committed to an all-or-nothing launch date.

# Case Study: HealthCare.gov– McKinsey “Red Team” Assessment

Programs of this type ideally have a sequential planning, design, and implementation process with significant testing and revision



#### Description of ideal situation:

- Clear articulation of requirements & success metrics
- Minimized dependency on third parties
- Sequential requirements, design, build, and testing
- Iteration and revision between phases
- End-to-end integrated operations and IT testing
- Limited initial launch

#### Current situation:

- Evolving requirements
- Multiple definitions of success
- Significant dependency on external parties/contractors
- Parallel “stacking” of all phases
- Insufficient time and scope of end-to-end testing
- Launch at full volume

CMS has been working to mitigate challenges resulting from program characteristics



# Case Study: HealthCare.gov: The Failures – Management

- Management expertise is in getting contracts not delivering projects.
- Project quality is sacrificed for the sake of appearances.
- Seriously substandard staffing and under staffing
  - CGI. Three months before launch, only 10 developers were working on a crucial part of the site , and of those, only one was "at a high enough skill level."
- Lack of coordination among contractors
  - Unclear responsibilities. Fragmented authority.

# Case Study: HealthCare.gov: The Failures – Gov. & Policies

- Government IT projects
  - Most are over budget and/or behind schedule.
  - “Write-down-all-the-requirements-then-build-to-those-requirements” is outdated.
- IT procurement policies
  - Cost-plus contract, no-bid contracts
  - “The firms that typically get contracts are the firms that are good at getting contracts, not typically good at executing on them.”

# Case Study: HealthCare.Gov – Dec. 2013

- 400+ bug fixes, by the end of Nov. 2013
  - “Operate smoothly for most users.” – W.H.
- Availability > 90%
- Response time (landing page) < 1s
- Error rate (per page) < 1%
- Completion rate  $\approx$  80%
- System capacity  $\approx$  50,000 concurrent users
- Sign-ups
  - 27,000 in Oct, 110,000 in Nov, 975,000 in Dec

# Case Study: HealthCare.Gov – April 2014

- Issues on data accuracy and completeness
  - Estimated 10-15% of sign-ups missing
  - Other inaccuracies have been reported
- CGI work during repair continue to be substandard
  - Half of the software fixes failed. – CMS
- CGI contract has been terminated. (Jan. 10, 2014)
- March 31, 2014 (last day to sign up), site went down
- April 1, 2014. 7.1 million signed up – W.H.

# Case Study: HealthCare.Gov – Aftermath

- Accenture took over in Jan. 2014 as the lead contractor for development and maintenance: \$175M
- Cost of building the system – GAO, July 2014
  - \$834M through Feb. 2014
    - Total estimated cost: > \$2B
  - “CMS undertook the development of HealthCare.gov without effective planning or oversight practices”
  - Also found “increased and unnecessary risk of unauthorized access, use, disclosure, modification or loss” of information
  - CMS only withheld \$267,000 in requested fees, 2% of the contract, from CGI.

# Case Study: HealthCare.Gov – 2015 Enrollment Cycle

- Enrollment for 2015 (37 states)
  - Nov 15, 2014 – February 15, 2015.
  - Outages on the first day
  - More smooth operation thereafter
- February, 2015
  - ~ 11.4 million sign-ups (~8.6 million re-enrollments)
    - Open enrollment extensions: March 15 – April 30
  - ~800,000 received incorrect tax information,
    - Incorrect amount on 1095-A Form

# Case Study: HealthCare.gov: The Lessons Learned

- Adopt software engineering best practices
  - Agile software development. Testing early.
  - Software quality assurance and testing. Testing throughout.
- Adopt management best practices
  - Clear responsibility and accountability
  - Performance metrics and progress tracking
- Revamp government IT procurement policies
  - Current system is antiquated, and has failed.
  - Bring government IT to the 21<sup>st</sup> century.

# Software Reliability

# Metrics of Software Quality – Performance & Scalability

- Performance
  - The ability to complete requested functions or services within the expected time span by the users.
  - e.g., average response time for a given task
- Scalability
  - The capacity of a system to handle increasing load or demand.
  - e.g., # of concurrent users, # of transactions per second, # of requests per second

# Product Quality Metrics

- Two key metrics for intrinsic product quality are Mean Time To Failure (MTTF) and availability
- **MTTF** is most often used with safety critical systems such as air traffic control systems, avionics, and weapons
- **Availability** is the probability that a system will work as required when required during the period of a mission.
- Both are correlated, but different in the same way that failures and defects are different

# Metrics of Software Quality – Mean Time Between Failures

- Mean time between failures (MTBF)
  - Average of intervals between consecutive failures.
- Mean time to failures (MTTF)
  - Average amount of time a system operates before it fails
- Mean time to repair (MTTR)
  - Average time to repair/restart the system and get it back to running
- MTBF is a simple measure of reliability

$$\text{MTBF} = \text{MTTF} + \text{MTTR}$$



Memorize  
You'll Solve

# Metrics of Software Quality – Availability & Reliability

- Availability

- The probability of a system to be available.
- The fraction of time the system is available.

$$\frac{\text{available time ("up time")}}{\text{total time}}$$

*understand the differences.*

- Reliability

- The probability of a system to operate without failures.
- The fraction of all attempted operations that complete *successfully.*

$$\frac{\text{\# of successful operations}}{\text{\# of total operations attempted}}$$

# Software Availability

- Software availability is the probability that a program is operating according to requirements at a given point in time and is defined as

$$\text{Availability} = [\text{MTTF}/(\text{MTTF} + \text{MTTR})] \times 100\%$$

- Consider 5 nines availability (99.999%); what does this mean?
  - 5 minutes of down time per year

[See Availability (system) – [https://en.wikipedia.org/wiki/Availability\\_\(system\)](https://en.wikipedia.org/wiki/Availability_(system))]

# Metrics of Software Quality – Error Rate & Completion Rate

- Reliability depends on the *unit* of operation
  - An operation may consists of multiple steps
  - Reliability  $\neq$  Completion rate
- Error rate (per page)
  - The fraction of pages (unit of operation) that time out or fail
- Completion rate
  - The fraction of all attempted operations that *eventually* complete the operation
  - Completion  $\neq$  Success

# Integration & System Testing

- Integration testing
  - To expose defects in the interfaces and the interactions between integrated sub-systems.
- System (“end-to-end”) testing
  - Test of an integrated system to determine whether it meets the specification.

# Acceptance & Beta Testing

- Acceptance testing
  - To determine whether or not a system satisfies the user needs and requirements.
  - To enable the user, customers, or other authorized entity to determine whether or not to accept the system.
- Beta testing
  - One form of acceptance testing
  - Performed by *real* users in their own environment
  - Perform actual tasks without interference.

# The Spectrum of Software Quality

# What is Quality?

Some possible definitions:

- Quality = zero defects (Crosby)
- The totality of features and characteristics of a product or service that bear on its ability to **satisfy specified or implied needs.** (ISO) ✎
- Quality = fitness for purpose (Juran)
- Quality n., the degree of excellence (OED)

# Software System Qualities

- Correctness
- Availability
- Reliability
- Performance
- Scalability
- Efficiency
- Safety
- Usability
- Security
- Robustness
- Maintainability
- Reusability
- Portability
- Interoperability

# On Expected Behavior – Correctness vs. Reliability

- Correctness
    - Whether a system is consistent with its specification.
  - Reliability
    - The probability of a system to operate without failures.
    - Relative to its specification and a usage profile.
    - Statistical approximation to correctness
- 100% reliable  $\approx$  correct

# On Exceptional Behavior – Safety vs. Robustness

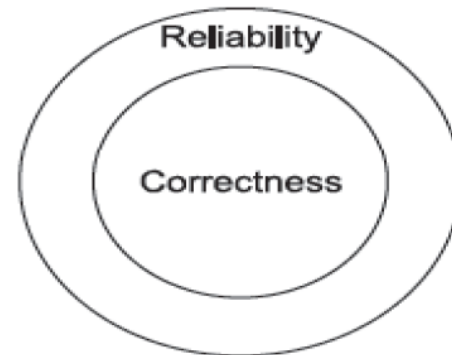
- Safety
  - The ability of a software system to prevent certain undesirable behaviors, i.e., *hazards*.
- Robustness
  - The ability of a software system to *fail or degrade gracefully* outside its normal operating parameters.
  - Acceptable (degraded) behavior under extreme conditions.

# Correctness

- Correctness is an all-or-nothing proposition.
- A program cannot be mostly correct or somewhat correct or 30% correct, it is absolutely correct on all possible behaviors or else it is not correct.
- It is very easy to achieve correctness, since every program is correct with respect to some (very bad) specification.
- Correctness is a goal to aim for, but is rarely provably achieved.

# Reliability

- Informally, software is reliable if the user can depend on it. The specialized literature on software reliability defines reliability in terms of statistical behavior—the probability that the software will operate as expected over a specified time interval.



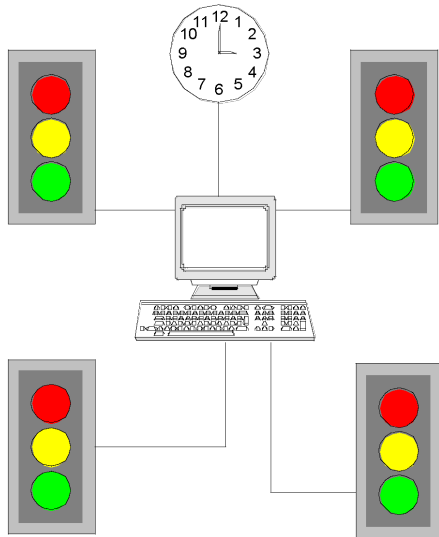
# Robustness

- A program is robust if it behaves “reasonably,” even in circumstances that were not anticipated in the requirements specification
  - for example, when it encounters incorrect input data or some hardware malfunction (say, disk crash).
- Obviously, robustness is a difficult-to-define quality

# Correctness and reliability

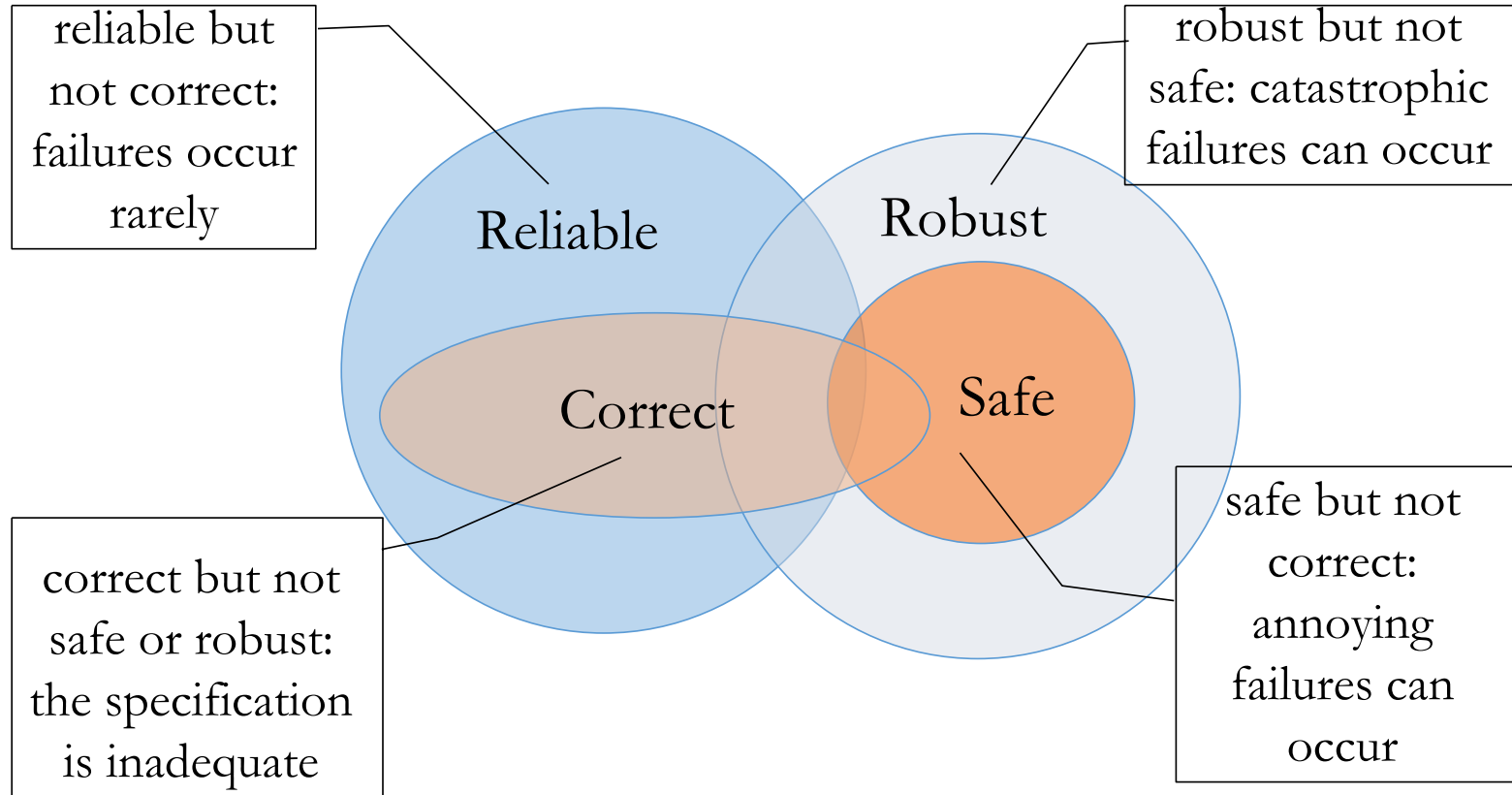
- Correctness and reliability are contingent on normal operating conditions.
- Software that is “correct” may still fail when the assumptions of its design are violated. How it fails matters.
- Software that “gracefully” fails is robust.
  - Consider events that could cause system failure.
  - Decide on an appropriate counter-measure to ensure graceful degradation of services.

# Focusing on – Different Facets of the System

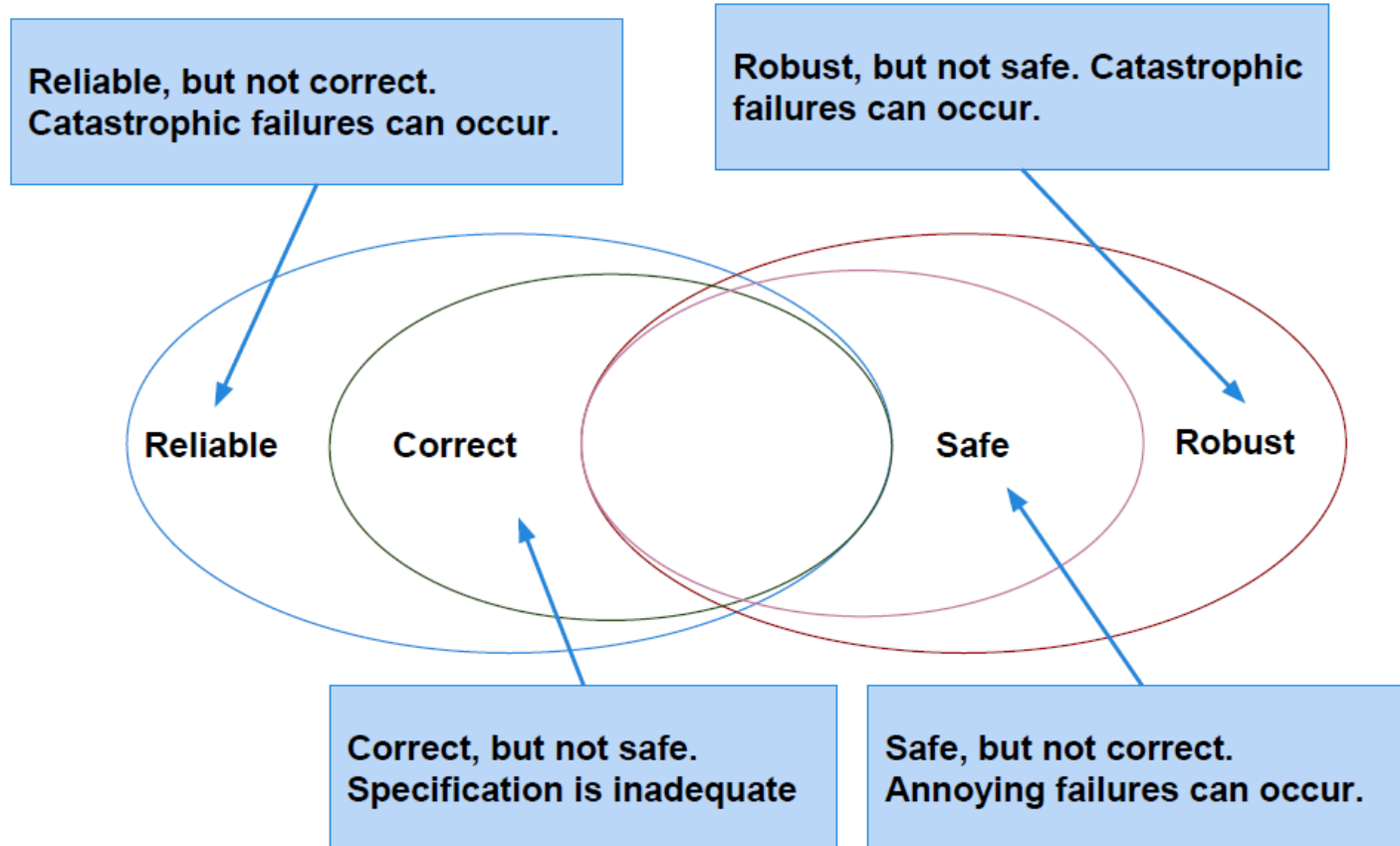


- Correctness, reliability:
  - Let traffic pass according to correct pattern and central scheduling
- Robustness, safety:
  - Provide degraded function when possible
    - never signal conflicting greens.
    - blinking red / blinking yellow is better than no lights
    - no lights is better than conflicting greens

# Relationship Among the Qualities



# Relationship Among the Qualities



# Performance Related Qualities

- Performance
  - The ability to complete requested functions or services within the expected time span by the users.
- Scalability
  - The capacity of a system to handle increasing load or demand.
- Efficiency
  - The ability to make maximum and efficient use of system resources.

# Usability & Security

- Usability
  - The ability for the users to use all the features of the system without special efforts.
- Security
  - The ability to maintain integrity of the system operation and the data.

# Internal Qualities

- Maintainability
  - The ability to make changes, enhance, adapt, and evolve a software system over a long period of time.
- Reusability
  - The ability to use parts of the system in different project without special effort on the part of the developers
- Portability
  - The ability to port a software system to a different platform or operating environment

# Software Quality

Conformance to customers' requirements

# Quality

- For software, two kinds of quality may be encountered:
  - **Quality of design** encompasses requirements, specifications, and the design of the system.
  - **Quality of conformance** is an issue focused primarily on implementation.
  - user satisfaction = compliant product + good quality + delivery within budget and schedule

# Cost of Quality

- Prevention costs include
  - Quality planning
  - Formal technical reviews
  - Test equipment
  - Training
- Internal failure costs include
  - Rework
  - Repair
  - Failure mode analysis
- External failure costs are
  - Complaint resolution
  - Product return and replacement
  - Help line support
  - Warranty work

# Customers' Expectations

- What's wrong with “performance to customers' expectations” rather than requirements?
- Often hear people say “We must exceed the customers' expectations!”
- What's the basic problem with this?
- The result is?

# Software Quality



explain

- Conformance to explicitly stated functional and performance requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software.
- Quality must be defined and measured if improvements are to be achieved
- In the narrowest sense, it is commonly recognized as the lack of “bugs” in the product
- Also, the most basic meaning of conformance to requirements because if the software contains too many functional defects, the basic requirement of providing the desired function is not met
- How is this usually expressed?

# Application to Software

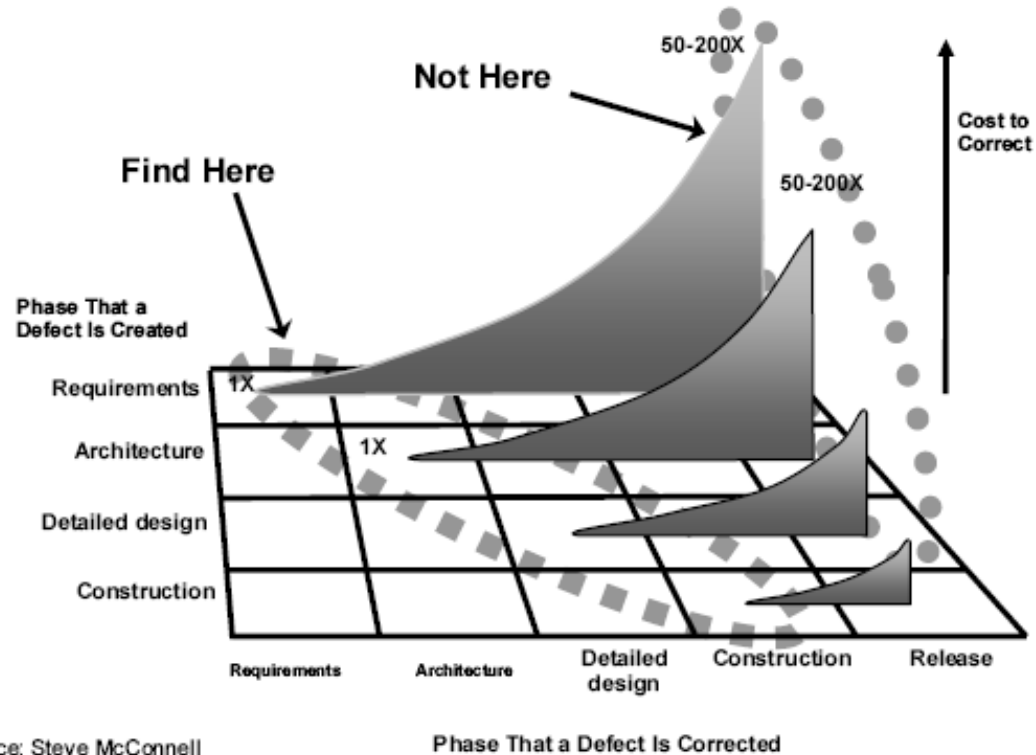
- Simplistically, software product quality is lack of “bugs” in the product
- Why is this problematical for software systems?
  - Correct operation is not sufficient – performance?
  - Usability by the end-user
  - Software specifications

# Cost of Software Defects

# Saving Time and Money

- Even experienced software engineers inject a defect about every ten lines of code
- The cost of finding and fixing defects increases at every step in the development process
- The defect find & fix times range from 3 minutes in code reviews to 25 minutes in inspections and 1400 minutes in system testing
- For accurate plans and reliable commitments, you must insist on what?

# Cost of Software Defects



Source: Steve McConnell

# Estimated Cost of Fixing Defects

		Time detected				
		spec	design	code	test	post-release
Time introduced	spec	1×	3×	5–10×	15×	30–100×
	design	-	1×	10×	20×	30–100×
	code	-	-	1×	10×	20–50×

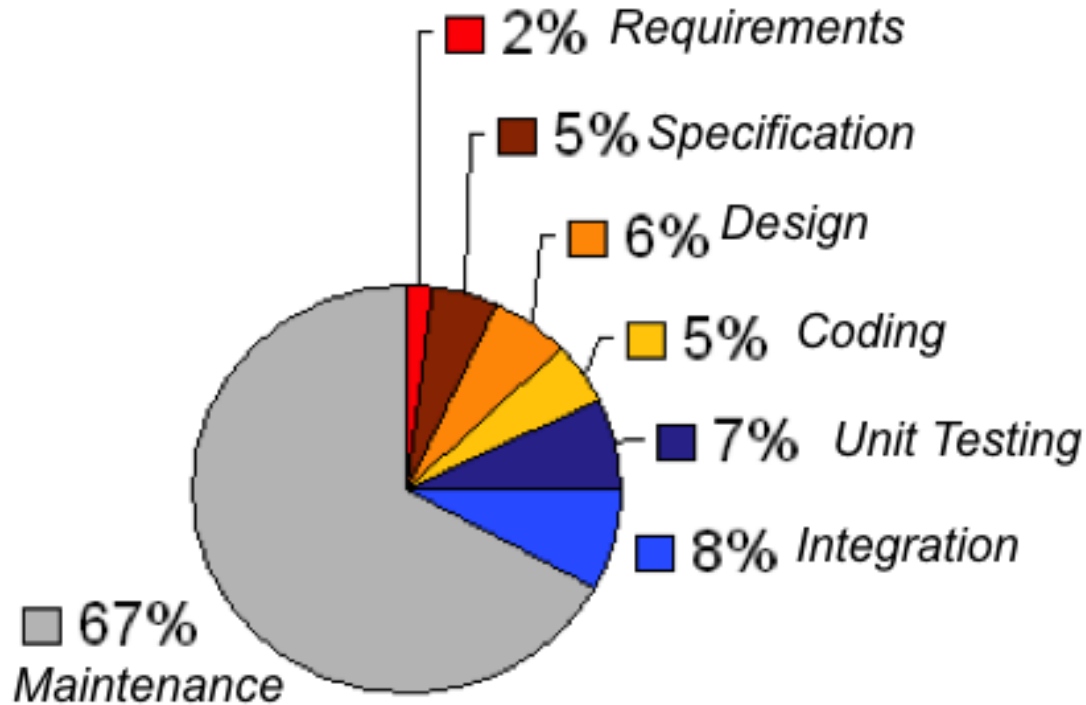
The earlier a defect is discovered, the lower the cost of fixing the defect.

# Distribution of Defects – Time Introduced and Fixed

Time introduced (%)	Time detected (%)					Total
	spec design	code unit test	integration system test	beta test	post-release	
specification design	3.5	10.5	35	6	15	70
code unit test	-	6	9	2	3	20
integration system test	-	-	6.5	1	2.5	10
Total	3.5	16.5	50.5	9	20.5	100%

- Majority of defects are introduced early
- Majority of defects are discovered late.

# Cost by Development Phases



# Software Verification and Validation (V&V)

# Verification and Validation

*What's the difference?*

- **Verification**

Does the software system meet the requirements specifications?

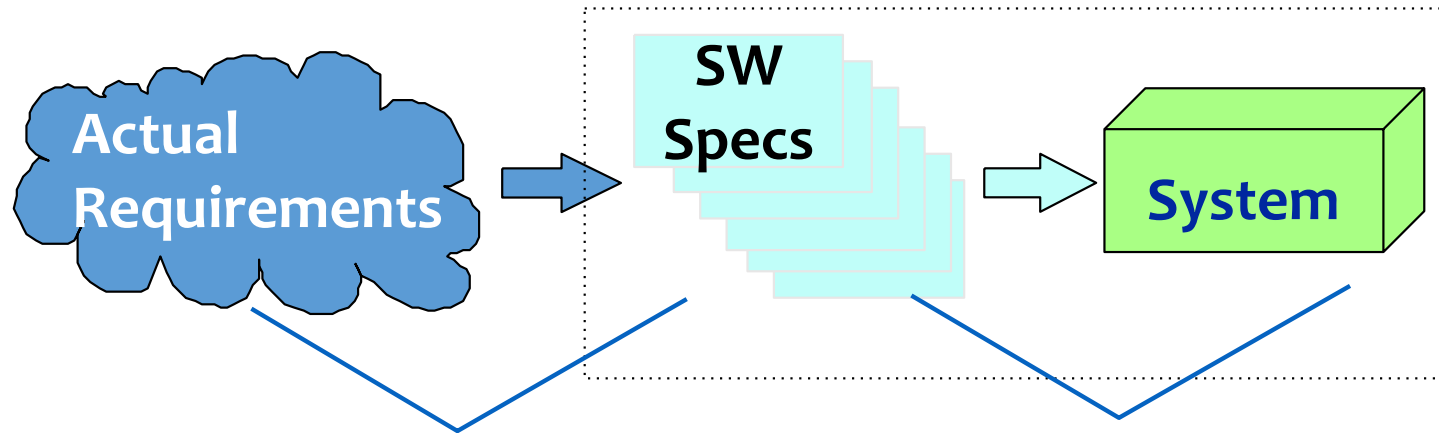
*\* Are we building the software right?*

- **Validation**

Does the software system meet the user's real needs?

*Are we building the right software?*

# Validation vs. Verification



## Validation

Includes

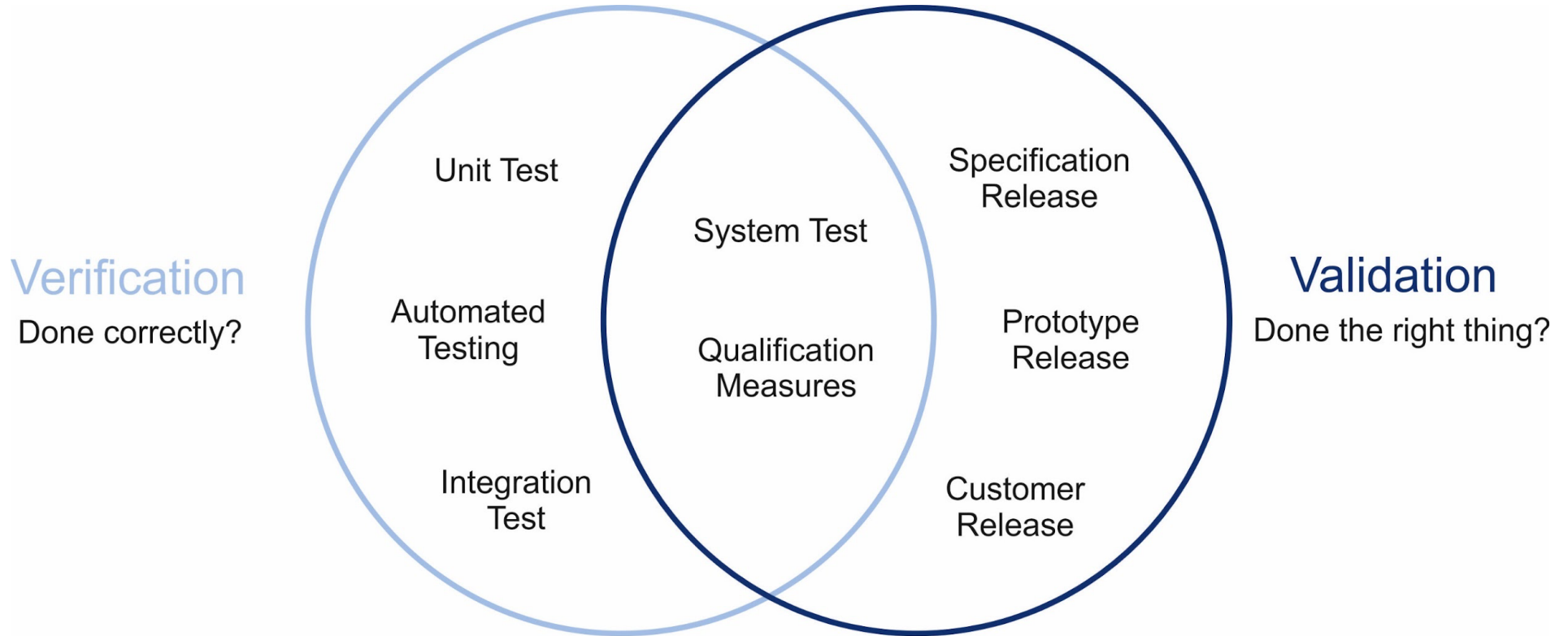
- usability testing
- user feedback

## Verification

Includes

- testing (mostly)
- inspections
- static analysis

# Validation vs. Verification



# Software Testing in V&V

- Testing can be done for verification and validation
- Verification:
  - To find defects by executing a program in a test or simulated environment
  - e.g., functional test, integration test
- Validation:
  - To find defects by executing a program in a real environment or with real users
  - e.g., usability test, beta test

# Software Testing in Development Life Cycle

# Software Qualities and Process

- Qualities cannot be added after development
  - Quality results from a set of inter-dependent activities
  - Analysis and testing are crucial but far from sufficient.
- Testing is not a phase, but a lifestyle
  - Testing and analysis activities occur from early in requirements engineering through delivery and subsequent evolution.
  - Quality depends on every part of the software process
- An essential feature of software processes is that software test and analysis is thoroughly integrated and not an afterthought

# The Quality Process

- Quality process: set of activities and responsibilities
  - focused primarily on ensuring adequate dependability
  - concerned with project schedule or with product usability
- The quality process provides a framework for
  - selecting and arranging activities
  - considering interactions and trade-offs with other important goals.

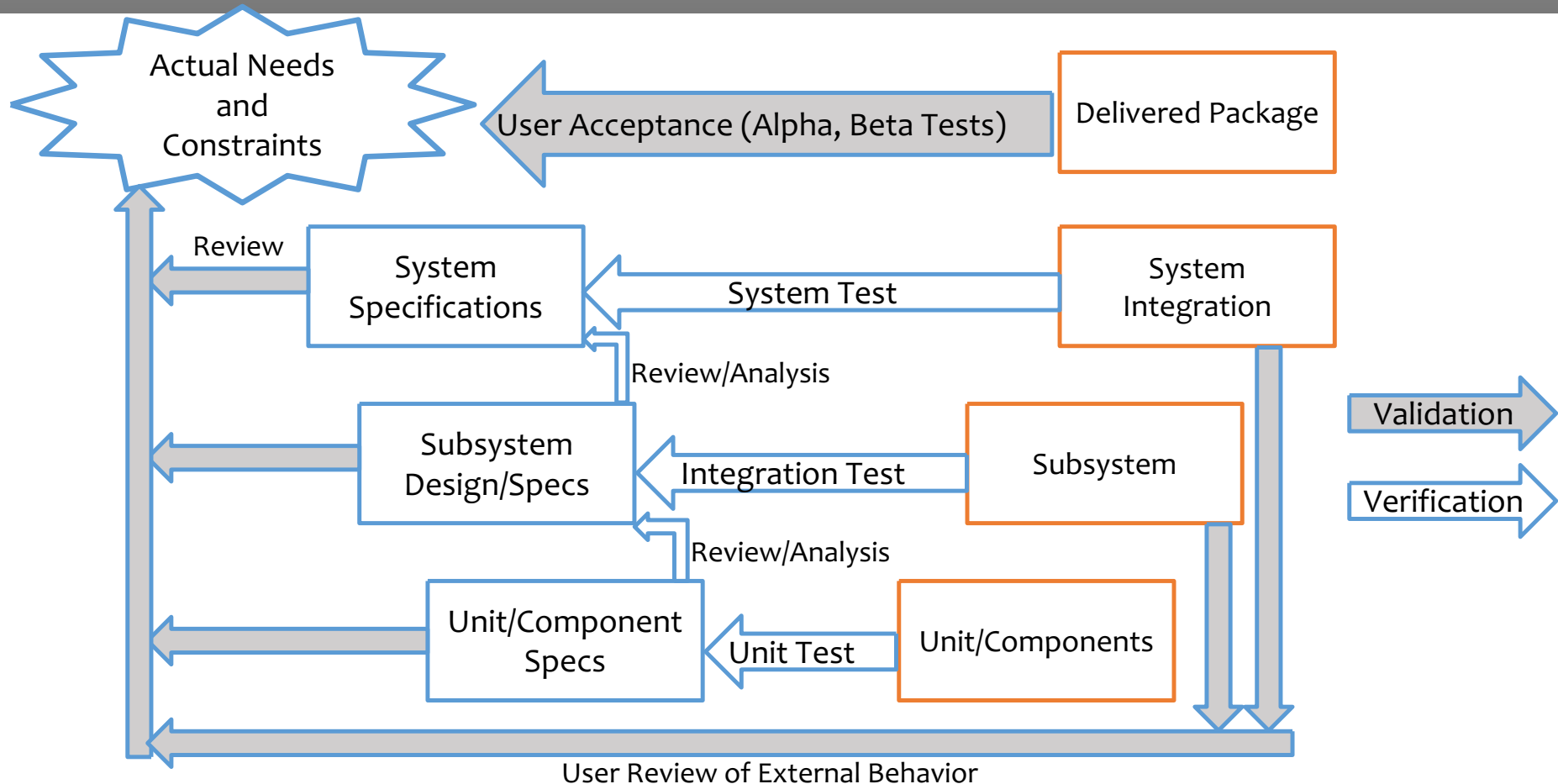
# Testing Activities in Life Cycle

- For every development activity there is a corresponding testing activity
  - Development phases, development levels
- Each test level has objectives specific to that level
- Test design should start as early as possible
  - as soon as relevant documents are available
  
- Applicable to waterfall and agile development model

# Levels of Granularity of Testing

- Unit (component, module) testing
- Integration testing
- System testing
- Acceptance testing

# The V-Model of – Validation & Verification



# Unit Testing

- Testing of individual software unit/module/components
  - Synonymous to *module testing*, *component testing*
- Focus on the functions of the unit
  - functionality, correctness, accuracy
- Usually carried out by the developers of the unit
- Basis for unit testing
  - component specifications
  - detailed design and code

# Integration Testing

- Testing performed to expose defects in the *interfaces* and in the *interactions between* integrated components or sub-systems.
- Focus on the interactions between modules
- Usually carried out by the developers of the sub-systems involved
- Basis for integration testing
  - system design and architecture
  - subsystem and interface specification

# Regression Testing

- Used when a large amount of testing is needed and the changes, while small, can affect many parts of the system.
- Best example is in compiler development:
  - Collect selected examples of code that exercise each part of the compiler
  - Add new examples when a bug is detected
  - Run the compiler over the entire collection and capture the output
  - After any change of the code within the compiler, repeat the run
  - Compare with the baseline results

# System Testing

- Testing of an integrated system to verify that it meets the specification.
  - A.k.a. the *end-to-end test*
- Verify functional and *non-functional* requirements
- Carried out by the developers and *independent testers*
- Basis for system testing
  - software requirement *specification*
  - functional *specification*

# Acceptance Testing

- Test the whole system to ensure that it meets the requirements
- Focus on customer acceptance
- Carried out by independent testers and the customers
- Basis for acceptance testing
  - system and user requirements
  - use cases, business processes, risk analysis

# Acceptance Testing & Criteria

- Acceptance testing

Formal testing with respect to user needs, requirements, and business processes conducted to determine whether or not a system satisfies the acceptance criteria and to enable the user, customers or other authorized entity to determine whether or not to accept the system.

- Acceptance criteria

The exit criteria that a component or system must satisfy in order to be accepted by a user, customer, or other authorized entity.

# Acceptance Testing Techniques

- Random (statistical) testing
- Alpha testing
- Beta testing

# Acceptance Testing – Random Test

- Random test (statistical test)
  - Test cases are selected randomly, possibly using a pseudo-random number generation algorithm, to match an *operation profile*, or *usage profile*.
- Not the same as *ad hoc* testing

# Acceptance Testing – Alpha Test

- Simulated operational testing.
- Performed by personnel *acting as* potential users/customers.
- Carried out in a *controlled* environment.
- Observed by the development organization.

# Acceptance Testing – Beta Test

- Operational testing to determine whether or not a component or system satisfies the user/customer needs and fits within the business processes.
- Performed by real users in their own environment.
- Perform actual tasks without interference or close monitoring

# Summary: Key Concepts

- Spectrum of software qualities
- Metrics of quality attributes
- Cost of software defects
- V-model of validation and verification
- Levels of granularity of testing
  - Unit, integration, system, acceptance test
  - Regression test