Testing - Foundations

Mindmap Summaries on TestingEducation.Org – Testing Foundations
Course by: Cem Kaner, James Bach & Rebecca L. Fiedler

the
test
tribe

Rahul Parwal

Foreword by James Marcus Bach

Acknowledgement

This book is published based on the material on Foundations available at testingeducation.org which is under the restrictions of the Creative Commons Attribution - Share Alike License.

We would like to explicitly acknowledge the authors and copyright holders, i.e. Dr. Cem Kaner and James Marcus Bach for the remarkable work that they have done and made publicly available for study, reference, and self-learning.

Mind Map Summary E-Book on Testing - Foundations

I came across TestingEducation.org Course after watching a keynote talk by Ajay Balamurugadas at CAST 2015. If you are also interested in the future of testing and the learning opportunities for testers, then I would recommend this talk to you too. It's available at bit.ly/ajkeynote.

I started the Testing Foundations Course using the self-paced video(s) available at http://www.testingeducation.org/
Having spent almost 4 years in the software industry, I was confident that I would be able to cover this 2.5 hours (157 mins) course on testing basics (foundations) within 2-3 days. However, when I started with this course, I realized that each chapter is filled with so much and would require a lot of notetaking, processing, & challenging the existing understanding of things. I started making mind map summaries for each lecture and started sharing them on LinkedIn as my daily learning capsule.

The response that was received from the Testing community was overwhelmingly positive. I would like to mention the name of Ajay Balamurugadas and Shailesh Gohel, who saw the seed of this book in me. Thanks to everyone for helping me with your positive feedback on mind maps/summaries.

This e-book is useful for anyone who wants to **understand, revise, study, or learn about software testing** and its foundational concepts.

Happy Reading! Happy Learning!

Rahul Parwal
Student of Software Testing
Member of The Test Tribe Community
rahul-parwal parwalrahul

Dedicated to my father and mother,
who taught me how to test, explore & share in life

Foreword

My name is on the BBST class, but I've never taught it. Cem Kaner put my name on it because he used so much of my material and ideas in the design. But, in fact, the class is a monumental curriculum development effort by Cem, himself. It's his vision and his philosophy of teaching, plus a couple of thousand hours of his meticulous labor. The closest I ever got to teaching it was when I was a "beta tester" student during the first-ever attempt to teach BBST. But I never finished it. I was expelled! Well, more accurately, Michael Bolton and I were kindly asked by Cem to drop out, because he was worried that we were too obsessive about the exercises. We were staying up all night competing with each other to give the most elaborate and deep answers to even simple questions. Cem thought we might be intimidating the other students.

I was very happy to stop. I needed to sleep. Taking BBST is a lot like climbing a mountain. I have my disagreements with the class, but in general, I would say that I admire anyone who passes it; and even people who didn't pass it but worked hard.

Back when he created BBST, Cem and I were collaborating on changing the world of testing. Each of us pursued this in his own ways. I am a high school dropout who distrusts formal schooling; Cem has two doctorates (a Ph.D. in psychophysics and a J.D.) and was a professor at the Florida Institute of Technology. I enjoy personally coaching and teaching, but that limits the impact I can have; Cem wanted something easier to scale.

BBST was originally developed as an undergraduate course at FIT, which explains its emphasis on grading. Cem was also hoping to create a compelling alternative to the shallow and poorly researched ISTQB certification.

In hindsight, Cem's vision didn't work out. Why? The ISTQB is popular BECAUSE it's shallow and poorly researched! That's why.

BBST is hard because developing COMPETENCE is hard.

ISTQB is easy because recycling popular myths on the internet about testing is easy.

In this booklet, Rahul has put together a tantalizing glimpse of some of its content.

If you are a serious student of testing, then I strongly suggest that you dive in.

James Marcus Bach

Creator of Rapid Software Testing methodology

[james-bach](#) [jamesmarcusbach](#)

TABLE OF CONTENTS

Introduction

1 Overview & Basic Definitions

2 Strategy

3 Oracles

4 Programming Fundamentals & Coverage

5 The Impossibility of Complete Testing

6 Introduction to Measurement

RECOMMENDED READINGS

Introduction

The Testing Foundations course is one of the most eye-opening and in-depth online course on the fundamental concepts in software testing and its critical challenges. I have tried to compile this e-book for anyone who wants to **understand, revise, study, or learn about software testing** and its foundational concepts.

NOTE: This e-book is in not a substitute for the TestingEducation.Org - Testing Foundations course but is an extension to it. It will help you to revisit the testing concepts and could be used as a cheat sheet for foundational testing knowledge on Software Testing.

This e-book consists of the topics ranging from the scope of testing, to software testing metrics.

It presents basic terminology in the field of software testing and considers:

- The Mission of Testing
- The Oracle Problem
- The Measurement Problem
- The Impossibility of Complete Testing

How to read mind maps:

- Start at 12 o'clock and go clockwise.
- Colors and Images have been added to the mind maps to give strength to the summary and make it easier to read.
- Different colored lines have been used to separate the different areas of the mind map.
- Symbols have been used to add extra strength to the associations and it can have a meaning of its own (not always).

Chapter One

Overview & Basic Definitions

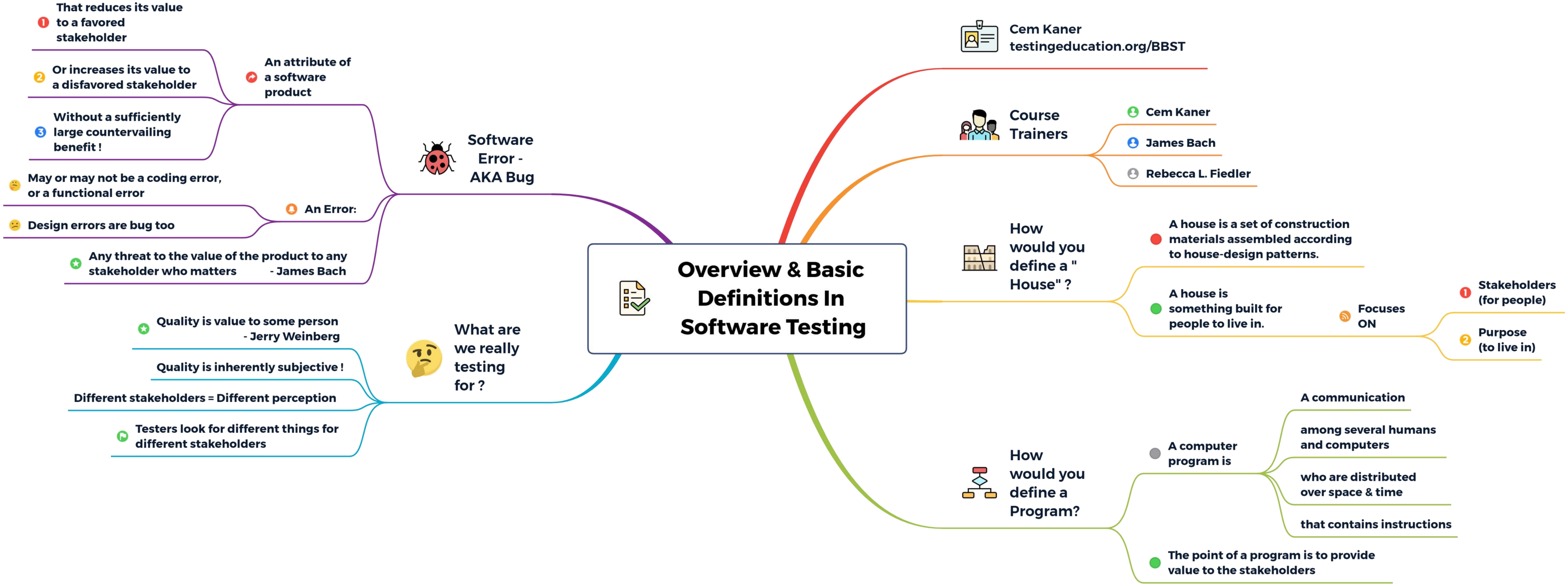
Overview & Basic Definitions

This section provides an overview of the online Testing Foundations course and introduces some definitions commonly used in the testing field.

Topics Covered:

- Definitions
- What are we really testing for?
- Software Error – AKA Bug
- Software Testing
- Testing Approaches
- Levels of Testing
- Functional & Parafunctional Testing
- Acceptance Testing
- Independent Testing

@parwalrahul



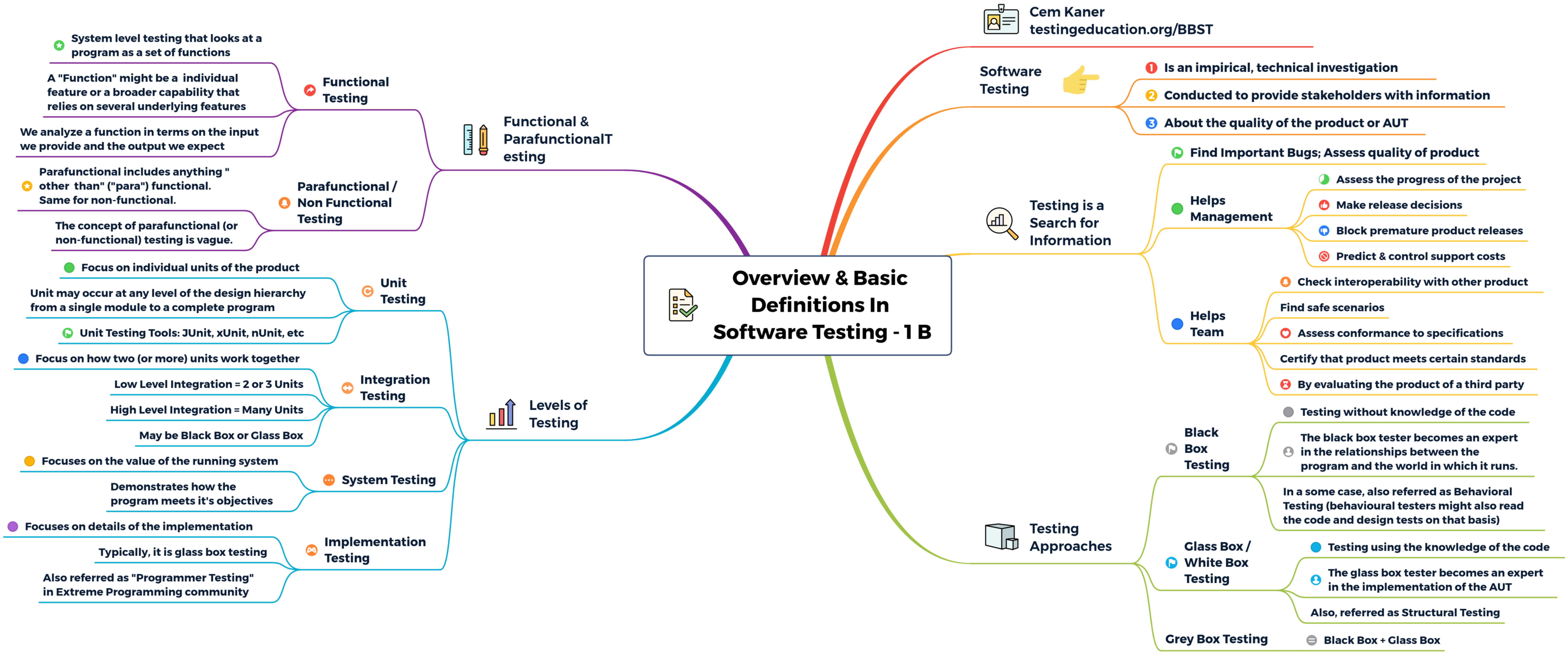
Overview & Basic Definitions In Software Testing

- Cem Kaner testingeducation.org/BBST
- Course Trainers
  - Cem Kaner
  - James Bach
  - Rebecca L. Fiedler
- How would you define a " House" ?
  - A house is a set of construction materials assembled according to house-design patterns.
  - A house is something built for people to live in.
    - Focuses ON
      1. Stakeholders (for people)
      2. Purpose (to live in)
- How would you define a Program?
  - A computer program is
    - A communication
    - among several humans and computers
    - who are distributed over space & time
    - that contains instructions
  - The point of a program is to provide value to the stakeholders
- Software Error - AKA Bug
  - An attribute of a software product
    1. That reduces its value to a favored stakeholder
    2. Or increases its value to a disfavored stakeholder
    3. Without a sufficiently large countervailing benefit !
  - An Error:
    - May or may not be a coding error, or a functional error
    - Design errors are bug too
  - Any threat to the value of the product to any stakeholder who matters - James Bach
- What are we really testing for ?
  - Quality is value to some person - Jerry Weinberg
  - Quality is inherently subjective !
  - Different stakeholders = Different perception
  - Testers look for different things for different stakeholders

Foundations – 1A, Overview & Basic Definitions in Software Testing

Click Here For Interactive Mindmap

@parwalrahul

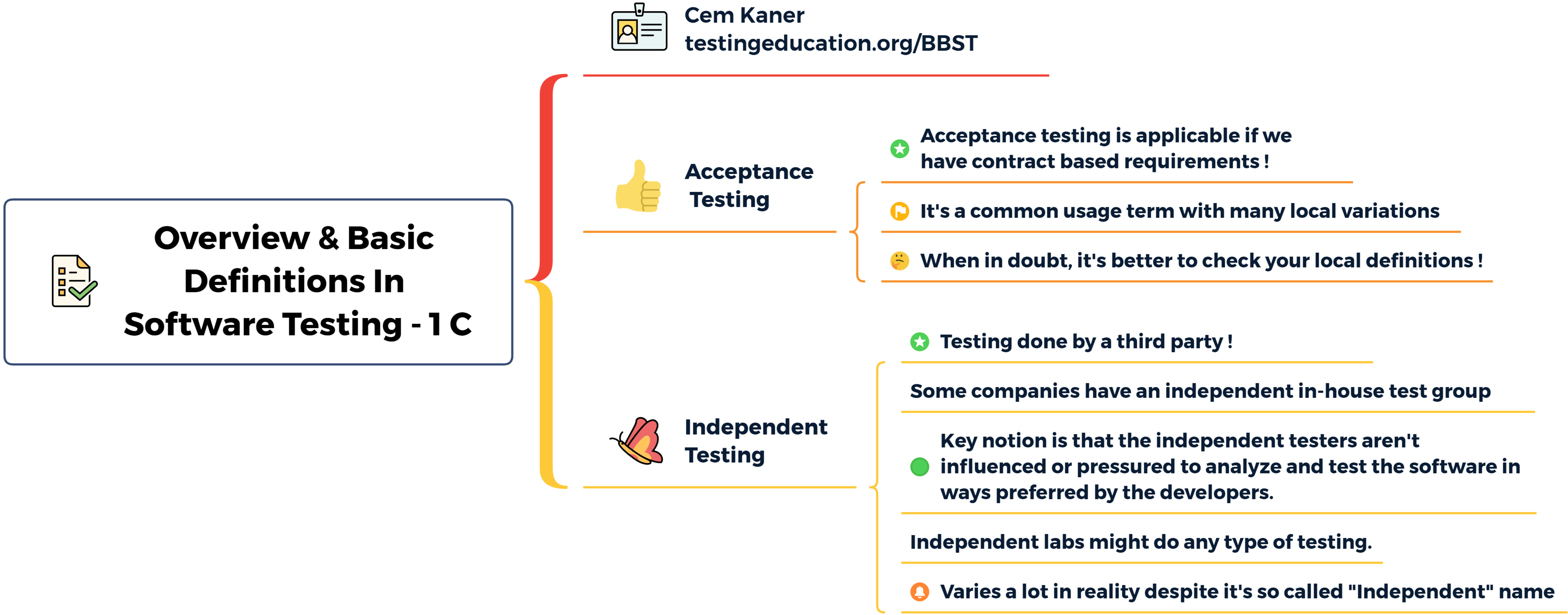


Overview & Basic Definitions In Software Testing - 1 B

- Cem Kaner testingeducation.org/BBST
- Software Testing
  - Is an impirical, technical investigation
  - Conducted to provide stakeholders with information
  - About the quality of the product or AUT
- Testing is a Search for Information
  - Find Important Bugs; Assess quality of product
  - Helps Management
    - Assess the progress of the project
    - Make release decisions
    - Block premature product releases
    - Predict & control support costs
  - Helps Team
    - Check interoperability with other product
    - Find safe scenarios
    - Assess conformance to specifications
    - Certify that product meets certain standards
    - By evaluating the product of a third party
- Testing Approaches
  - Black Box Testing
    - Testing without knowledge of the code
    - The black box tester becomes an expert in the relationships between the program and the world in which it runs.
    - In a some case, also referred as Behavioral Testing (behavioural testers might also read the code and design tests on that basis)
  - Glass Box / White Box Testing
    - Testing using the knowledge of the code
    - The glass box tester becomes an expert in the implementation of the AUT
    - Also, referred as Structural Testing
  - Grey Box Testing
    - Black Box + Glass Box
- Functional & ParafunctionalTesting
  - Functional Testing
    - System level testing that looks at a program as a set of functions
    - A "Function" might be a individual feature or a broader capability that relies on several underlying features
    - We analyze a function in terms on the input we provide and the output we expect
  - Parafunctional / Non Functional Testing
    - Parafunctional includes anything " other than" ("para") functional. Same for non-functional.
    - The concept of parafunctional (or non-functional) testing is vague.
- Levels of Testing
  - Unit Testing
    - Focus on individual units of the product
    - Unit may occur at any level of the design hierarchy from a single module to a complete program
    - Unit Testing Tools: JUnit, xUnit, nUnit, etc
  - Integration Testing
    - Focus on how two (or more) units work together
    - Low Level Integration = 2 or 3 Units
    - High Level Integration = Many Units
    - May be Black Box or Glass Box
  - System Testing
    - Focuses on the value of the running system
    - Demonstrates how the program meets it's objectives
  - Implementation Testing
    - Focuses on details of the implementation
    - Typically, it is glass box testing
    - Also referred as "Programmer Testing" in Extreme Programming community

Foundations – 1B, Overview & Basic Definitions in Software Testing

Click Here For Interactive Mindmap



Overview & Basic Definitions In Software Testing - 1 C

Cem Kaner
testingeducation.org/BBST

Acceptance Testing

- Acceptance testing is applicable if we have contract based requirements !
- It's a common usage term with many local variations
- When in doubt, it's better to check your local definitions !

Independent Testing

- Testing done by a third party !
- Some companies have an independent in-house test group
- Key notion is that the independent testers aren't influenced or pressured to analyze and test the software in ways preferred by the developers.
- Independent labs might do any type of testing.
- Varies a lot in reality despite it's so called "Independent" name

Foundations – 1C, Overview & Basic Definitions in Software Testing

Click Here For Interactive Mindmap

Chapter Two

Strategy

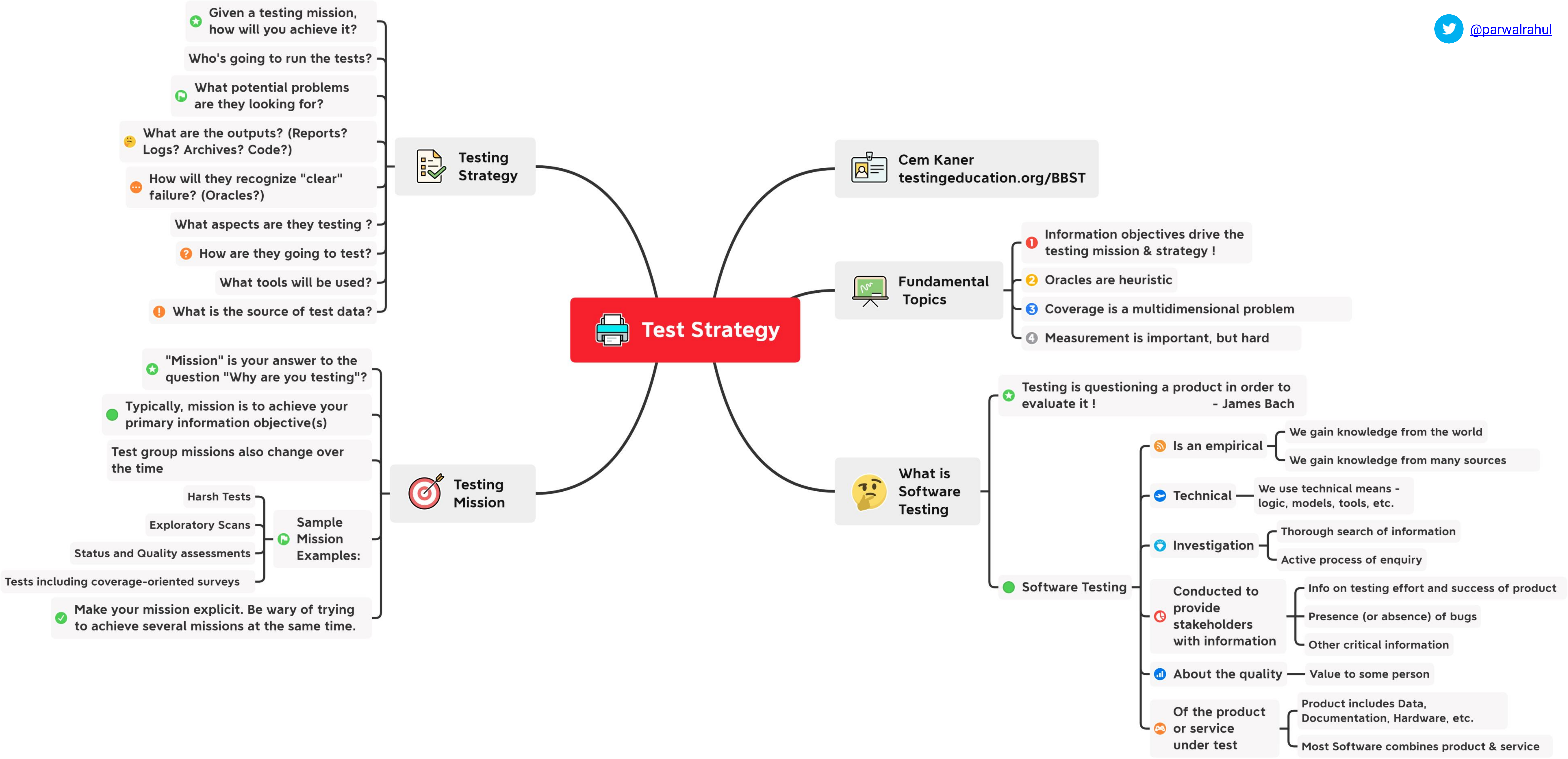
Strategy

This chapter considers why testers test, what they are trying to learn, and how they can organize their work to achieve their mission.

Topics Covered:

- What is Software Testing?
- Testing Mission
- Testing Strategy
- Test Techniques
- Typical Testing Tasks
- External Test Lab vs In-House Test Lab
- Verification vs Validation

@parwalrahul



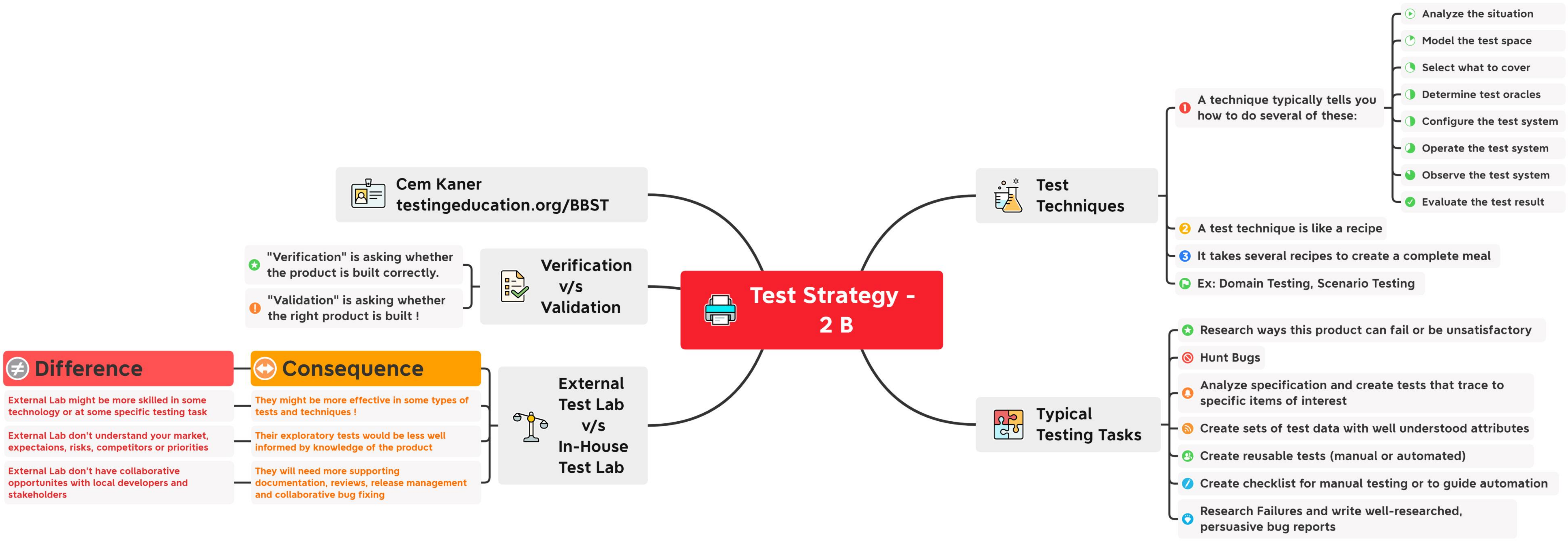
Test Strategy

- Testing Strategy
  - Given a testing mission, how will you achieve it?
  - Who's going to run the tests?
  - What potential problems are they looking for?
  - What are the outputs? (Reports? Logs? Archives? Code?)
  - How will they recognize "clear" failure? (Oracles?)
  - What aspects are they testing ?
  - How are they going to test?
  - What tools will be used?
  - What is the source of test data?
- Testing Mission
  - "Mission" is your answer to the question "Why are you testing"?
  - Typically, mission is to achieve your primary information objective(s)
  - Test group missions also change over the time
  - Sample Mission Examples:
    - Harsh Tests
    - Exploratory Scans
    - Status and Quality assessments
    - Tests including coverage-oriented surveys
  - Make your mission explicit. Be wary of trying to achieve several missions at the same time.
- Cem Kaner testingeducation.org/BBST
- Fundamental Topics
  1. Information objectives drive the testing mission & strategy !
  2. Oracles are heuristic
  3. Coverage is a multidimensional problem
  4. Measurement is important, but hard
- What is Software Testing
  - Testing is questioning a product in order to evaluate it ! - James Bach
  - Software Testing
    - Is an empirical
      - We gain knowledge from the world
      - We gain knowledge from many sources
    - Technical
      - We use technical means - logic, models, tools, etc.
    - Investigation
      - Thorough search of information
      - Active process of enquiry
    - Conducted to provide stakeholders with information
      - Info on testing effort and success of product
      - Presence (or absence) of bugs
      - Other critical information
    - About the quality
      - Value to some person
    - Of the product or service under test
      - Product includes Data, Documentation, Hardware, etc.
      - Most Software combines product & service

Foundations – 2A, Strategy

Click Here For Interactive Mindmap

@parwalrahul



- Test Strategy - 2 B
  - Cem Kaner testingeducation.org/BBST
  - Verification v/s Validation
    - "Verification" is asking whether the product is built correctly.
    - "Validation" is asking whether the right product is built !
  - External Test Lab v/s In-House Test Lab
    - Consequence
      - They might be more effective in some types of tests and techniques !
        - Difference: External Lab might be more skilled in some technology or at some specific testing task
      - Their exploratory tests would be less well informed by knowledge of the product
        - Difference: External Lab don't understand your market, expectaions, risks, competitors or priorities
      - They will need more supporting documentation, reviews, release management and collaborative bug fixing
        - Difference: External Lab don't have collaborative opportunites with local developers and stakeholders
  - Test Techniques
    - 1 A technique typically tells you how to do several of these:
      - Analyze the situation
      - Model the test space
      - Select what to cover
      - Determine test oracles
      - Configure the test system
      - Operate the test system
      - Observe the test system
      - Evaluate the test result
    - 2 A test technique is like a recipe
    - 3 It takes several recipes to create a complete meal
    - Ex: Domain Testing, Scenario Testing
  - Typical Testing Tasks
    - Research ways this product can fail or be unsatisfactory
    - Hunt Bugs
    - Analyze specification and create tests that trace to specific items of interest
    - Create sets of test data with well understood attributes
    - Create reusable tests (manual or automated)
    - Create checklist for manual testing or to guide automation
    - Research Failures and write well-researched, persuasive bug reports

Foundations – 2B, Strategy

Click Here For Interactive Mindmap

Chapter Three

Oracles

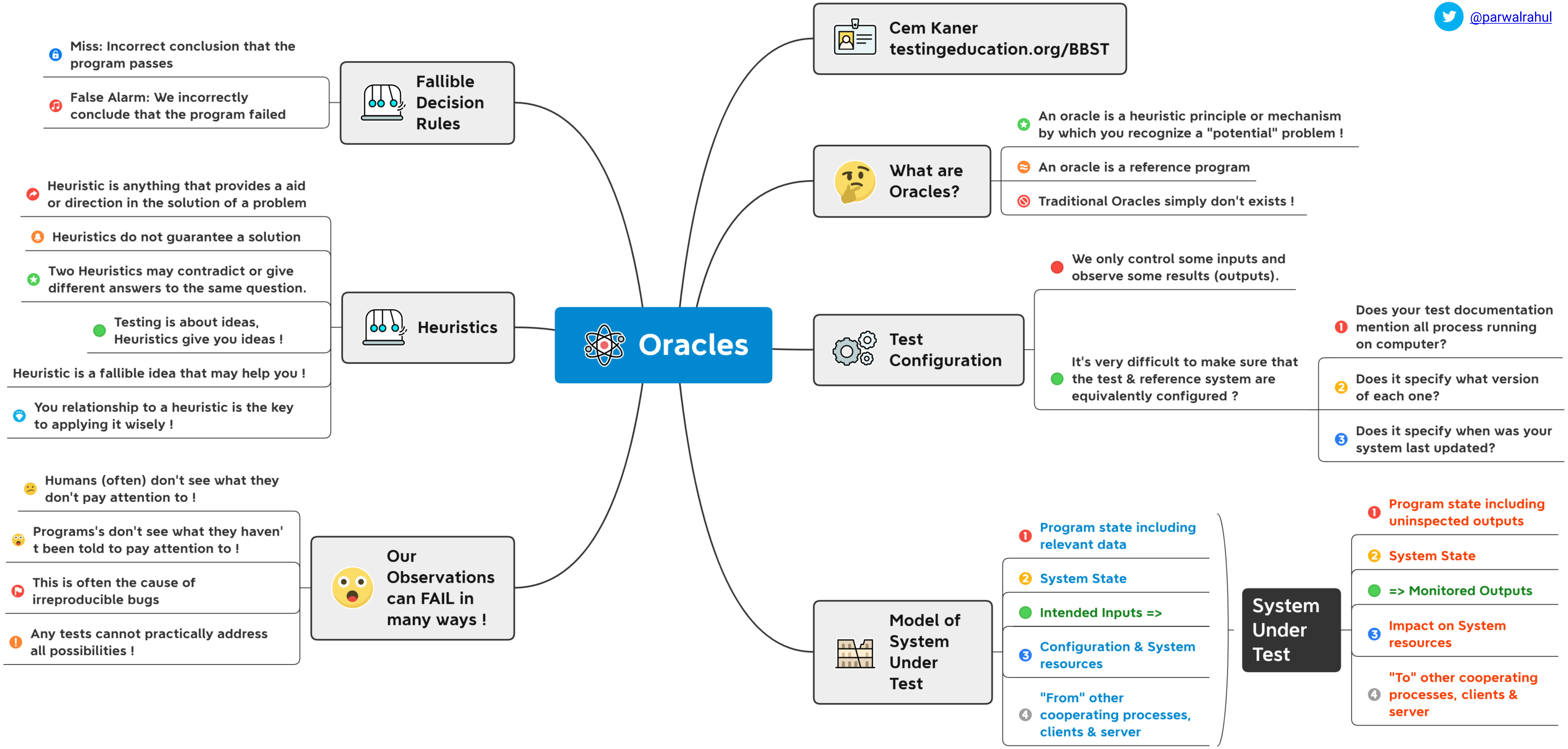
Oracles

This chapter presents software oracles as heuristics that help testers make a judgment whether or not software passes the tests that are run.

Topics Covered:

- What are Oracles
- Test Configuration
- Model of System Under Test
- How Observations FAIL?
- Heuristics
- Fallible Decision Rules
- Oracles
- Risk Based Testing
- Consistence Oracles
- Various Types of Oracles

@parwalrahul



Oracles

Cem Kaner testingeducation.org/BBST

What are Oracles?
- An oracle is a heuristic principle or mechanism by which you recognize a "potential" problem !
- An oracle is a reference program
- Traditional Oracles simply don't exists !

Test Configuration
- We only control some inputs and observe some results (outputs).
- It's very difficult to make sure that the test & reference system are equivalently configured ?
  1. Does your test documentation mention all process running on computer?
  2. Does it specify what version of each one?
  3. Does it specify when was your system last updated?

Model of System Under Test
1. Program state including relevant data
2. System State
- Intended Inputs =>
3. Configuration & System resources
4. "From" other cooperating processes, clients & server

System Under Test
1. Program state including uninspected outputs
2. System State
- => Monitored Outputs
3. Impact on System resources
4. "To" other cooperating processes, clients & server

Fallible Decision Rules
- Miss: Incorrect conclusion that the program passes
- False Alarm: We incorrectly conclude that the program failed

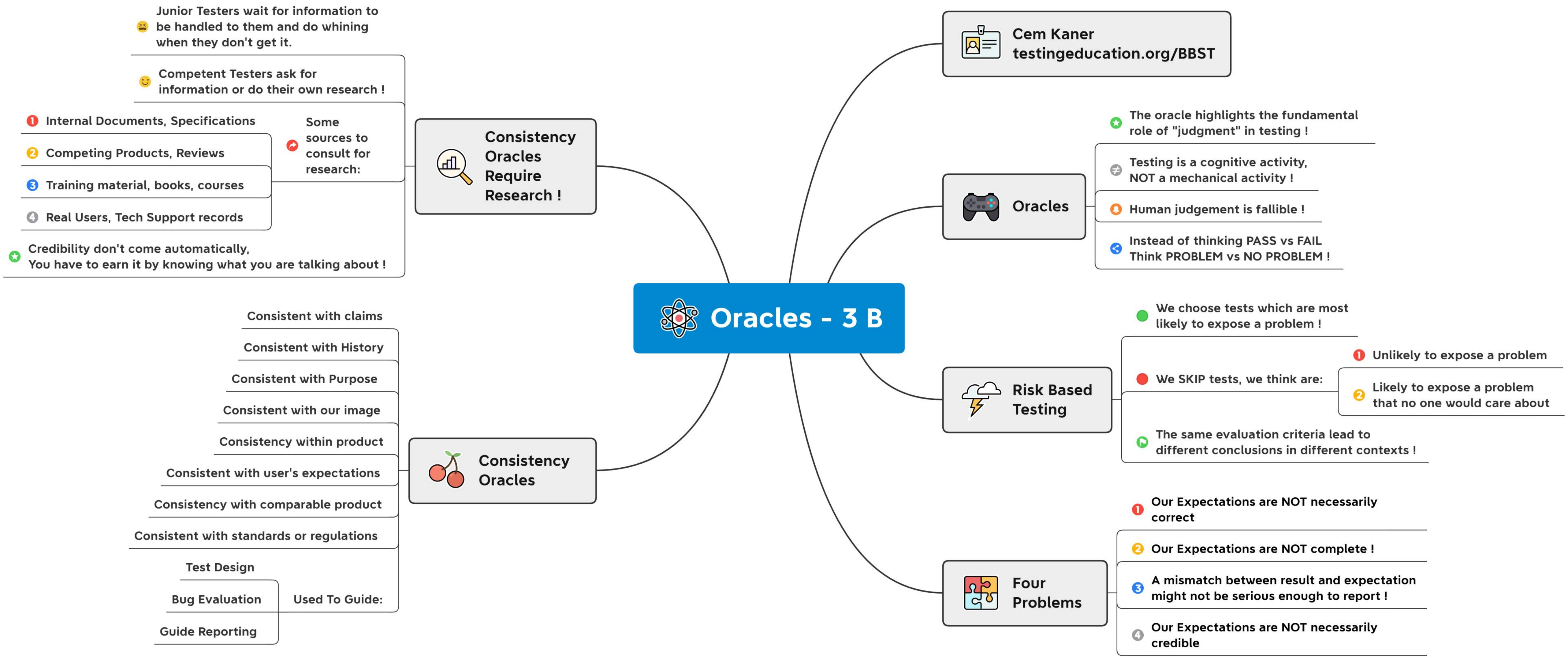
Heuristics
- Heuristic is anything that provides a aid or direction in the solution of a problem
- Heuristics do not guarantee a solution
- Two Heuristics may contradict or give different answers to the same question.
- Testing is about ideas, Heuristics give you ideas !
- Heuristic is a fallible idea that may help you !
- You relationship to a heuristic is the key to applying it wisely !

Our Observations can FAIL in many ways !
- Humans (often) don't see what they don't pay attention to !
- Programs's don't see what they haven' t been told to pay attention to !
- This is often the cause of irreproducible bugs
- Any tests cannot practically address all possibilities !

Foundations – 3A, Oracles

Click Here For Interactive Mindmap

@parwalrahul

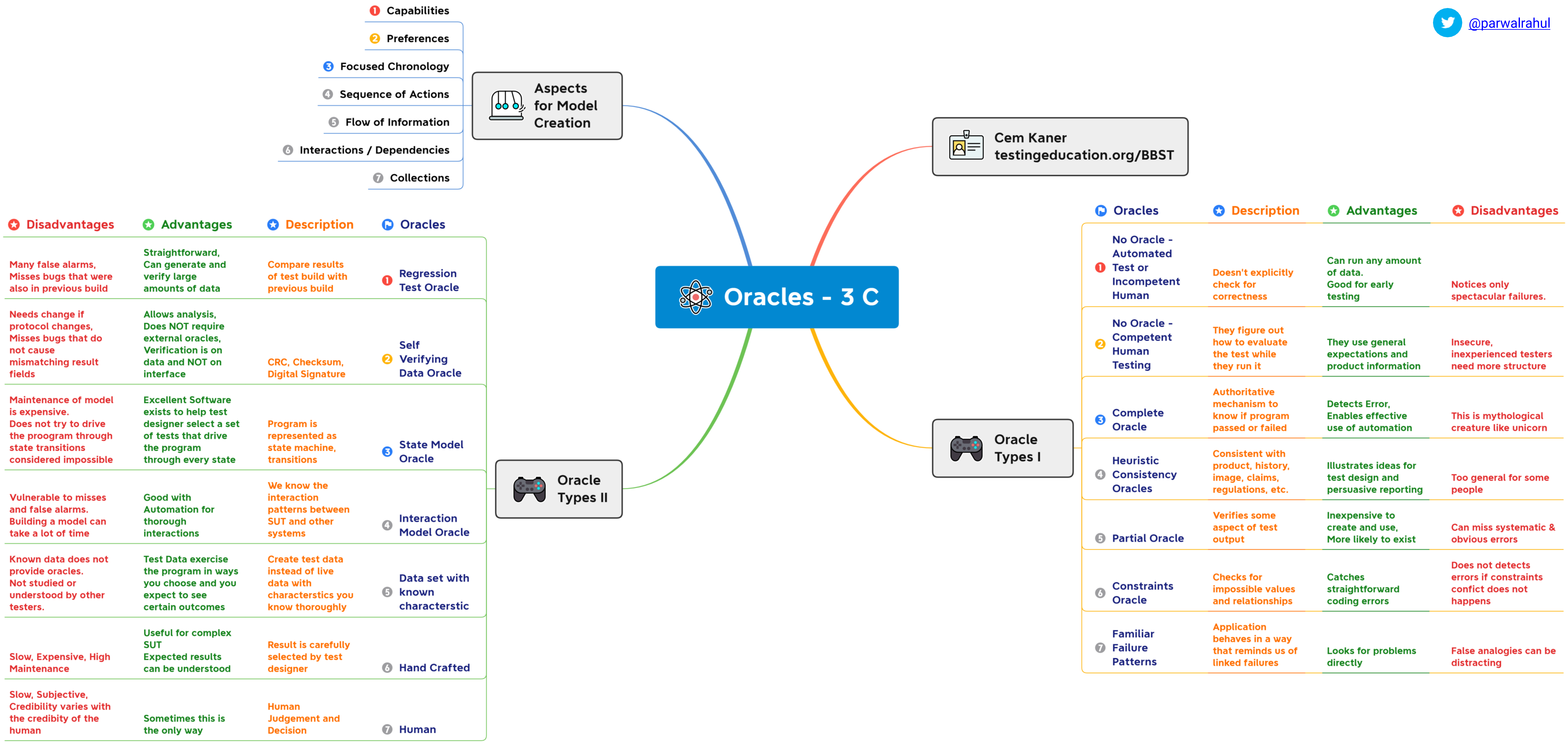


Oracles - 3 B

- Cem Kaner testingeducation.org/BBST
- Oracles
  - The oracle highlights the fundamental role of "judgment" in testing !
  - Testing is a cognitive activity, NOT a mechanical activity !
  - Human judgement is fallible !
  - Instead of thinking PASS vs FAIL Think PROBLEM vs NO PROBLEM !
- Risk Based Testing
  - We choose tests which are most likely to expose a problem !
  - We SKIP tests, we think are:
    1. Unlikely to expose a problem
    2. Likely to expose a problem that no one would care about
  - The same evaluation criteria lead to different conclusions in different contexts !
- Four Problems
  1. Our Expectations are NOT necessarily correct
  2. Our Expectations are NOT complete !
  3. A mismatch between result and expectation might not be serious enough to report !
  4. Our Expectations are NOT necessarily credible
- Consistency Oracles Require Research !
  - Junior Testers wait for information to be handled to them and do whining when they don't get it.
  - Competent Testers ask for information or do their own research !
  - Some sources to consult for research:
    1. Internal Documents, Specifications
    2. Competing Products, Reviews
    3. Training material, books, courses
    4. Real Users, Tech Support records
  - Credibility don't come automatically, You have to earn it by knowing what you are talking about !
- Consistency Oracles
  - Consistent with claims
  - Consistent with History
  - Consistent with Purpose
  - Consistent with our image
  - Consistency within product
  - Consistent with user's expectations
  - Consistency with comparable product
  - Consistent with standards or regulations
  - Used To Guide:
    - Test Design
    - Bug Evaluation
    - Guide Reporting

Foundations – 3B, Oracles

Click Here For Interactive Mindmap



@parwalrahul

Oracles - 3 C

Aspects for Model Creation

1. Capabilities
2. Preferences
3. Focused Chronology
4. Sequence of Actions
5. Flow of Information
6. Interactions / Dependencies
7. Collections

Cem Kaner testingeducation.org/BBST

Oracle Types I

Oracles	Description	Advantages	Disadvantages
1 No Oracle - Automated Test or Incompetent Human	Doesn't explicitly check for correctness	Can run any amount of data. Good for early testing	Notices only spectacular failures.
2 No Oracle - Competent Human Testing	They figure out how to evaluate the test while they run it	They use general expectations and product information	Insecure, inexperienced testers need more structure
3 Complete Oracle	Authoritative mechanism to know if program passed or failed	Detects Error, Enables effective use of automation	This is mythological creature like unicorn
4 Heuristic Consistency Oracles	Consistent with product, history, image, claims, regulations, etc.	Illustrates ideas for test design and persuasive reporting	Too general for some people
5 Partial Oracle	Verifies some aspect of test output	Inexpensive to create and use, More likely to exist	Can miss systematic & obvious errors
6 Constraints Oracle	Checks for impossible values and relationships	Catches straightforward coding errors	Does not detects errors if constraints conflict does not happens
7 Familiar Failure Patterns	Application behaves in a way that reminds us of linked failures	Looks for problems directly	False analogies can be distracting

Oracle Types II

Disadvantages	Advantages	Description	Oracles
Many false alarms, Misses bugs that were also in previous build	Straightforward, Can generate and verify large amounts of data	Compare results of test build with previous build	1 Regression Test Oracle
Needs change if protocol changes, Misses bugs that do not cause mismatching result fields	Allows analysis, Does NOT require external oracles, Verification is on data and NOT on interface	CRC, Checksum, Digital Signature	2 Self Verifying Data Oracle
Maintenance of model is expensive. Does not try to drive the progam through state transitions considered impossible	Excellent Software exists to help test designer select a set of tests that drive the program through every state	Program is represented as state machine, transitions	3 State Model Oracle
Vulnerable to misses and false alarms. Building a model can take a lot of time	Good with Automation for thorough interactions	We know the interaction patterns between SUT and other systems	4 Interaction Model Oracle
Known data does not provide oracles. Not studied or understood by other testers.	Test Data exercise the program in ways you choose and you expect to see certain outcomes	Create test data instead of live data with charactersitics you know thoroughly	5 Data set with known characterstic
Slow, Expensive, High Maintenance	Useful for complex SUT Expected results can be understood	Result is carefully selected by test designer	6 Hand Crafted
Slow, Subjective, Credibility varies with the credibity of the human	Sometimes this is the only way	Human Judgement and Decision	7 Human

Foundations – 3C, Oracles

Click Here For Interactive Mindmap

Chapter Four

Programming Fundamentals & Coverage

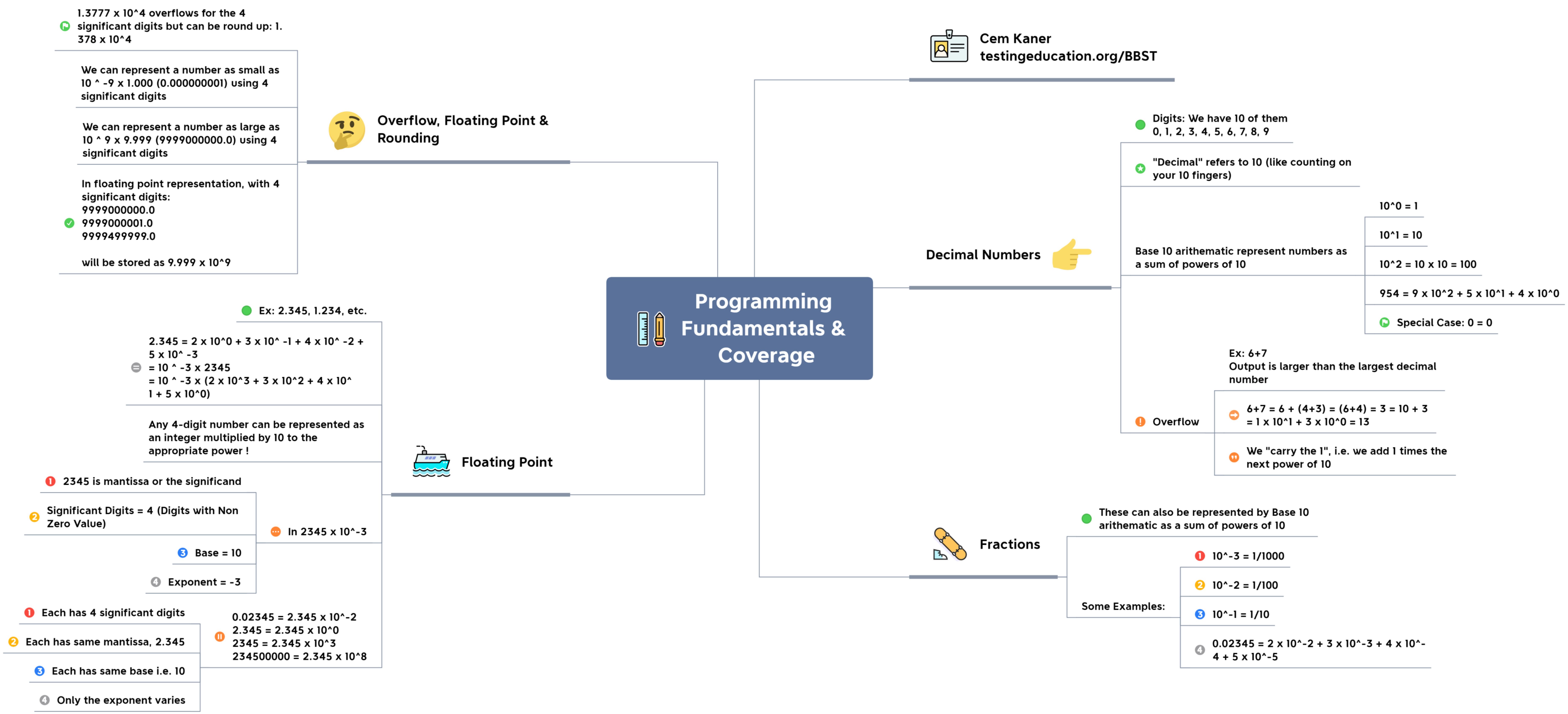
Programming Fundamentals & Coverage

This chapter presents information about basic data handling and storage to help testers think about the multi-dimensional problem of test coverage in more sophisticated ways.

Topics Covered:

- Decimal Numbers
- Fractions
- Floating Point
- Binary Numbers
- 8, 16, 32, 64 Bit Words
- Integer, Float, Double, ASCII
- Data Structures
- Control Structures
- Coverage
- Coverage as a Measurement

@parwalrahul



Programming Fundamentals & Coverage

Cem Kaner testingeducation.org/BBST

Decimal Numbers

- Digits: We have 10 of them 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
- "Decimal" refers to 10 (like counting on your 10 fingers)
- Base 10 arithematic represent numbers as a sum of powers of 10
  - 10^0 = 1
  - 10^1 = 10
  - 10^2 = 10 x 10 = 100
  - 954 = 9 x 10^2 + 5 x 10^1 + 4 x 10^0
  - Special Case: 0 = 0
- Overflow
  - Ex: 6+7
Output is larger than the largest decimal number
  - 6+7 = 6 + (4+3) = (6+4) = 3 = 10 + 3 = 1 x 10^1 + 3 x 10^0 = 13
  - We "carry the 1", i.e. we add 1 times the next power of 10

Fractions

- These can also be represented by Base 10 arithematic as a sum of powers of 10
- Some Examples:
  1. 10^-3 = 1/1000
  2. 10^-2 = 1/100
  3. 10^-1 = 1/10
  4. 0.02345 = 2 x 10^-2 + 3 x 10^-3 + 4 x 10^-4 + 5 x 10^-5

Overflow, Floating Point & Rounding

- 1.3777 x 10^4 overflows for the 4 significant digits but can be round up: 1.378 x 10^4
- We can represent a number as small as 10 ^ -9 x 1.000 (0.000000001) using 4 significant digits
- We can represent a number as large as 10 ^ 9 x 9.999 (9999000000.0) using 4 significant digits
- In floating point representation, with 4 significant digits:
9999000000.0
9999000001.0
9999499999.0

will be stored as 9.999 x 10^9

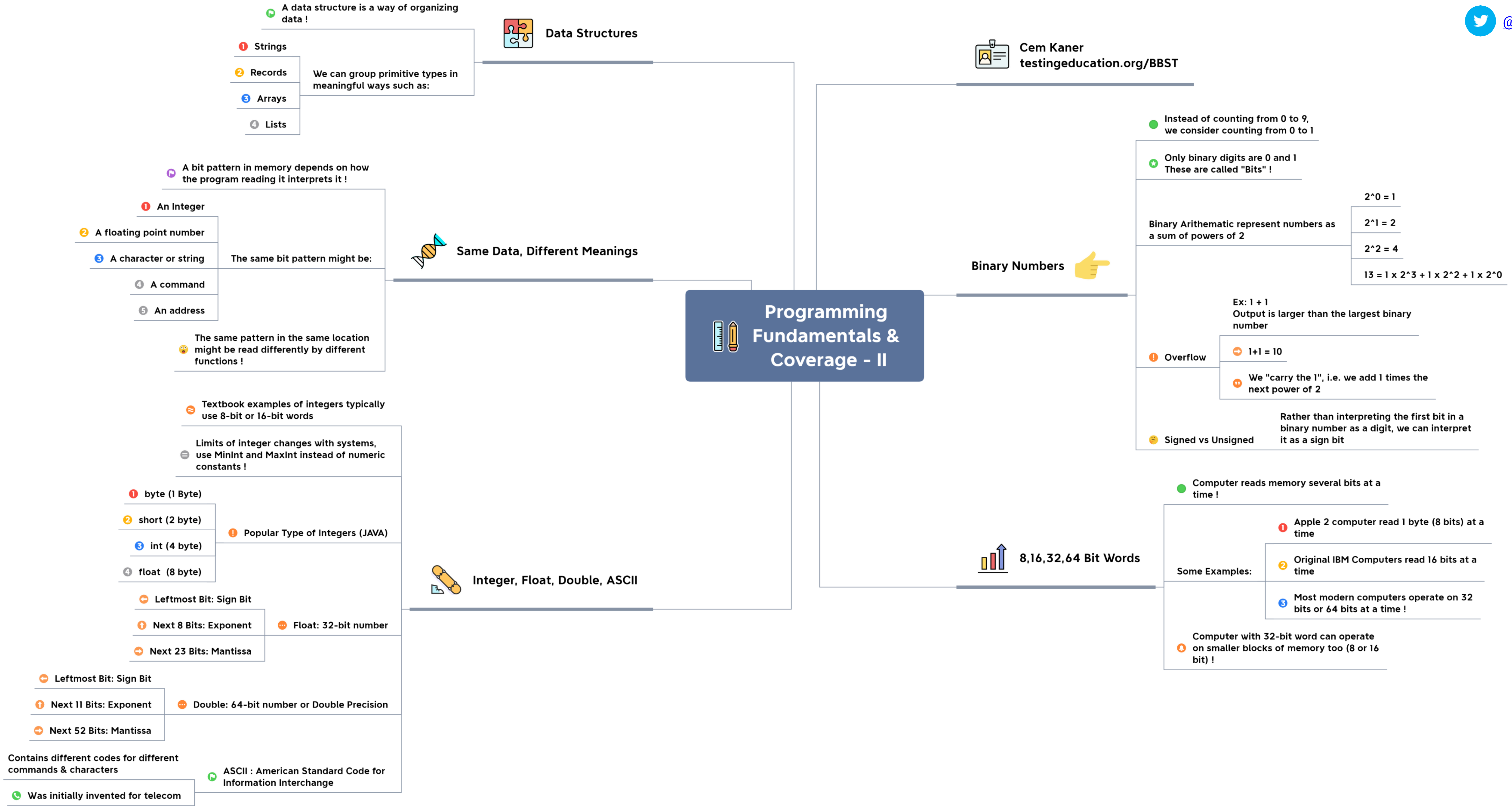
Floating Point

- Ex: 2.345, 1.234, etc.
- 2.345 = 2 x 10^0 + 3 x 10^ -1 + 4 x 10^ -2 + 5 x 10^ -3
= 10 ^ -3 x 2345
= 10 ^ -3 x (2 x 10^3 + 3 x 10^2 + 4 x 10^1 + 5 x 10^0)
- Any 4-digit number can be represented as an integer multiplied by 10 to the appropriate power !
- In 2345 x 10^-3
  1. 2345 is mantissa or the significand
  2. Significant Digits = 4 (Digits with Non Zero Value)
  3. Base = 10
  4. Exponent = -3
- 0.02345 = 2.345 x 10^-2
2.345 = 2.345 x 10^0
2345 = 2.345 x 10^3
234500000 = 2.345 x 10^8
  1. Each has 4 significant digits
  2. Each has same mantissa, 2.345
  3. Each has same base i.e. 10
  4. Only the exponent varies

Foundations – 4A, Programming Fundamentals & Coverage

Click Here For Interactive Mindmap

@parwalrahul



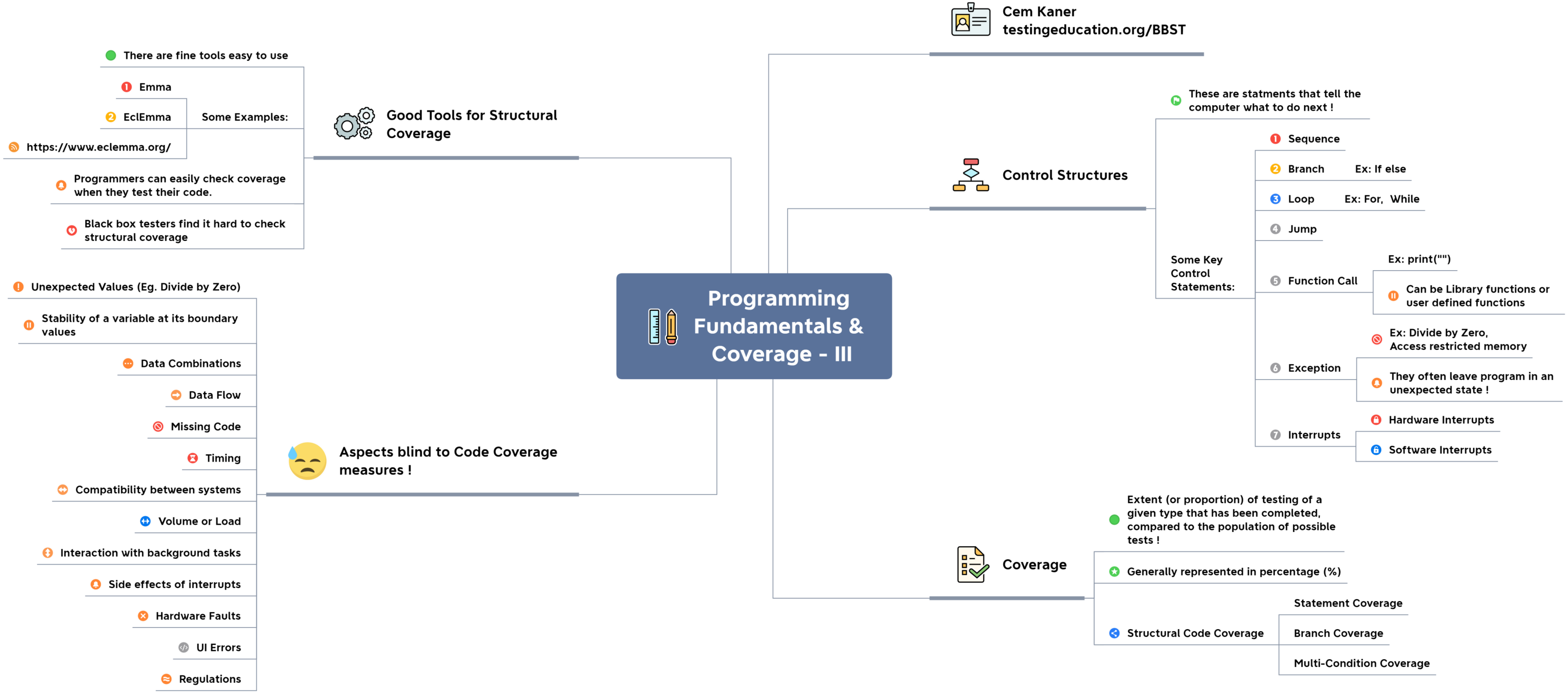
Programming Fundamentals & Coverage - II

- Data Structures
  - A data structure is a way of organizing data !
  - We can group primitive types in meaningful ways such as:
    1. Strings
    2. Records
    3. Arrays
    4. Lists
- Same Data, Different Meanings
  - A bit pattern in memory depends on how the program reading it interprets it !
  - The same bit pattern might be:
    1. An Integer
    2. A floating point number
    3. A character or string
    4. A command
    5. An address
  - The same pattern in the same location might be read differently by different functions !
- Integer, Float, Double, ASCII
  - Textbook examples of integers typically use 8-bit or 16-bit words
  - Limits of integer changes with systems, use MinInt and MaxInt instead of numeric constants !
  - Popular Type of Integers (JAVA)
    1. byte (1 Byte)
    2. short (2 byte)
    3. int (4 byte)
    4. float (8 byte)
  - Float: 32-bit number
    - Leftmost Bit: Sign Bit
    - Next 8 Bits: Exponent
    - Next 23 Bits: Mantissa
  - Double: 64-bit number or Double Precision
    - Leftmost Bit: Sign Bit
    - Next 11 Bits: Exponent
    - Next 52 Bits: Mantissa
  - ASCII : American Standard Code for Information Interchange
    - Contains different codes for different commands & characters
    - Was initially invented for telecom
- Cem Kaner testingeducation.org/BBST
- Binary Numbers
  - Instead of counting from 0 to 9, we consider counting from 0 to 1
  - Only binary digits are 0 and 1 These are called "Bits" !
  - Binary Arithematic represent numbers as a sum of powers of 2
    - 2^0 = 1
    - 2^1 = 2
    - 2^2 = 4
    - 13 = 1 x 2^3 + 1 x 2^2 + 1 x 2^0
  - Overflow
    - Ex: 1 + 1 Output is larger than the largest binary number
    - 1+1 = 10
    - We "carry the 1", i.e. we add 1 times the next power of 2
  - Signed vs Unsigned
    - Rather than interpreting the first bit in a binary number as a digit, we can interpret it as a sign bit
- 8,16,32,64 Bit Words
  - Computer reads memory several bits at a time !
  - Some Examples:
    1. Apple 2 computer read 1 byte (8 bits) at a time
    2. Original IBM Computers read 16 bits at a time
    3. Most modern computers operate on 32 bits or 64 bits at a time !
  - Computer with 32-bit word can operate on smaller blocks of memory too (8 or 16 bit) !

Foundations – 4B, Programming Fundamentals & Coverage

Click Here For Interactive Mindmap

@parwalrahul



Programming Fundamentals & Coverage - III

Cem Kaner
testingeducation.org/BBST

Control Structures

These are statments that tell the computer what to do next !

Some Key Control Statements:

- Sequence
- Branch
  - Ex: If else
- Loop
  - Ex: For, While
- Jump
- Function Call
  - Ex: print("")
  - Can be Library functions or user defined functions
- Exception
  - Ex: Divide by Zero, Access restricted memory
  - They often leave program in an unexpected state !
- Interrupts
  - Hardware Interrupts
  - Software Interrupts

Coverage

- Extent (or proportion) of testing of a given type that has been completed, compared to the population of possible tests !
- Generally represented in percentage (%)
- Structural Code Coverage
  - Statement Coverage
  - Branch Coverage
  - Multi-Condition Coverage

Good Tools for Structural Coverage

- There are fine tools easy to use
- Some Examples:
  - Emma
  - EclEmma
  - https://www.eclemma.org/
- Programmers can easily check coverage when they test their code.
- Black box testers find it hard to check structural coverage

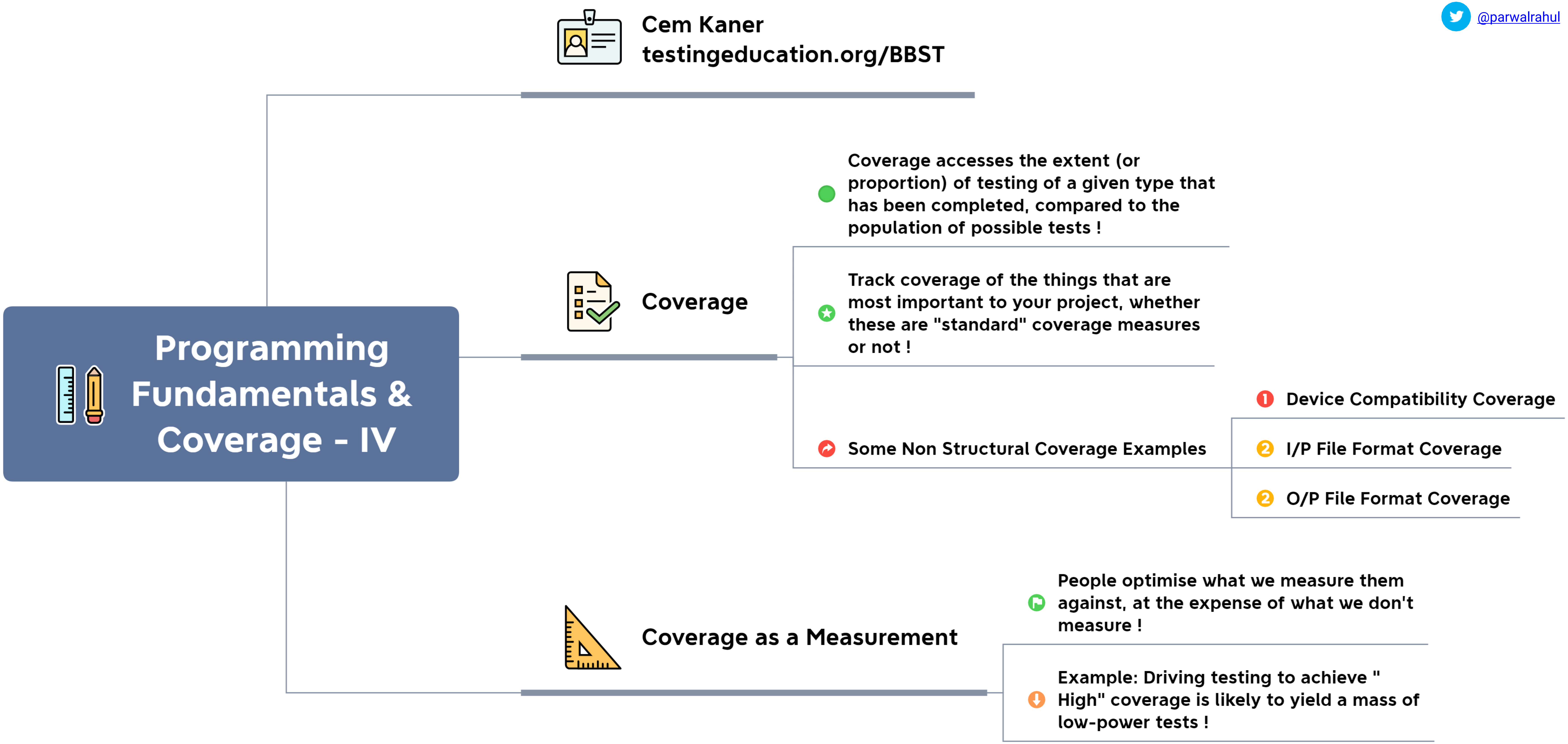
Aspects blind to Code Coverage measures !

- Unexpected Values (Eg. Divide by Zero)
- Stability of a variable at its boundary values
- Data Combinations
- Data Flow
- Missing Code
- Timing
- Compatibility between systems
- Volume or Load
- Interaction with background tasks
- Side effects of interrupts
- Hardware Faults
- UI Errors
- Regulations

Foundations – 4C, Programming Fundamentals & Coverage

Click Here For Interactive Mindmap

@parwalrahul



Programming Fundamentals & Coverage - IV

Cem Kaner testingeducation.org/BBST

Coverage

- Coverage accesses the extent (or proportion) of testing of a given type that has been completed, compared to the population of possible tests !
- Track coverage of the things that are most important to your project, whether these are "standard" coverage measures or not !
- Some Non Structural Coverage Examples
  1. Device Compatibility Coverage
  2. I/P File Format Coverage
  2. O/P File Format Coverage

Coverage as a Measurement

- People optimise what we measure them against, at the expense of what we don't measure !
- Example: Driving testing to achieve " High" coverage is likely to yield a mass of low-power tests !

Foundations – 4D, Programming Fundamentals & Coverage

Click Here For Interactive Mindmap

Chapter Five

The Impossibility of Complete Testing

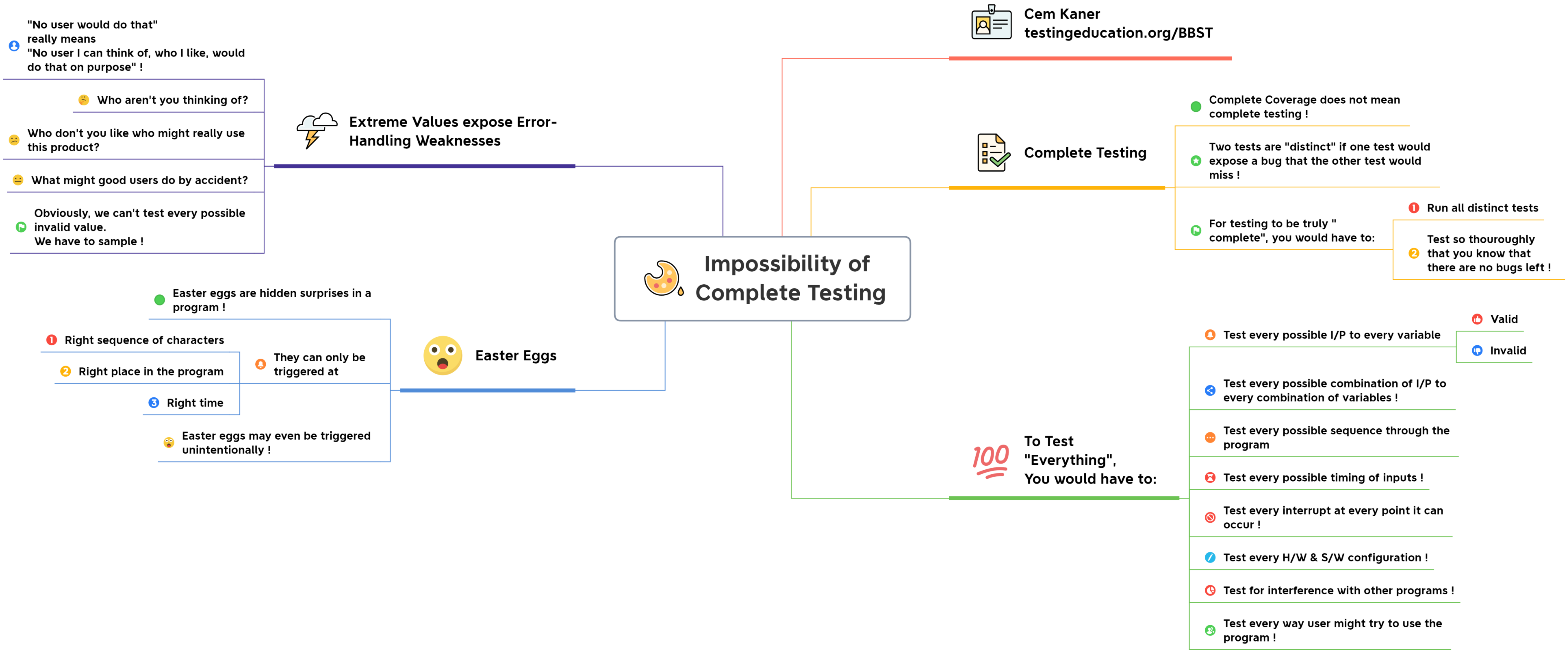
The Impossibility of Complete Testing

This chapter explores the complexity of determining when testing is finished and how the goal of complete testing is unattainable.

Topics Covered:

- Complete Testing
- Easter Eggs
- Error Handling Weakness
- Combination Testing
- Paths & Subpaths
- Data Flows Caution

@parwalrahul



Impossibility of Complete Testing

Cem Kaner
testingeducation.org/BBST

Complete Testing

- Complete Coverage does not mean complete testing !
- Two tests are "distinct" if one test would expose a bug that the other test would miss !
- For testing to be truly " complete", you would have to:
  1. Run all distinct tests
  2. Test so thouroughly that you know that there are no bugs left !

To Test "Everything", You would have to:

- Test every possible I/P to every variable
  - Valid
  - Invalid
- Test every possible combination of I/P to every combination of variables !
- Test every possible sequence through the program
- Test every possible timing of inputs !
- Test every interrupt at every point it can occur !
- Test every H/W & S/W configuration !
- Test for interference with other programs !
- Test every way user might try to use the program !

Extreme Values expose Error-Handling Weaknesses

- "No user would do that" really means "No user I can think of, who I like, would do that on purpose" !
- Who aren't you thinking of?
- Who don't you like who might really use this product?
- What might good users do by accident?
- Obviously, we can't test every possible invalid value. We have to sample !

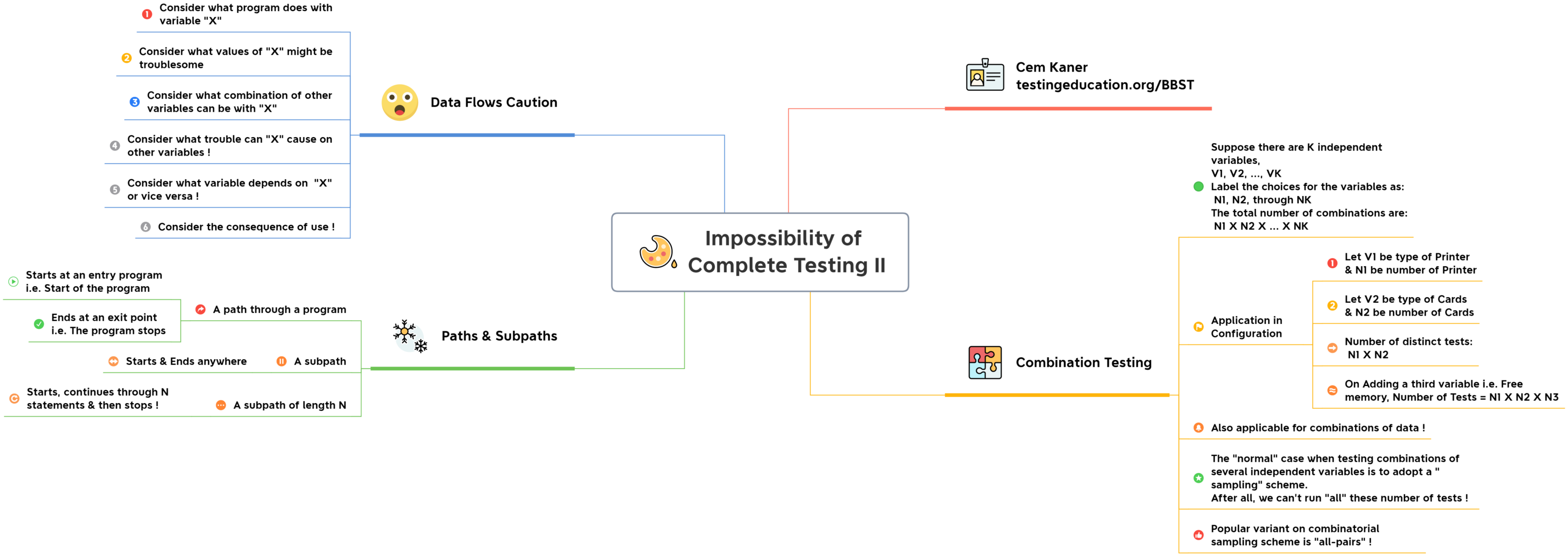
Easter Eggs

- Easter eggs are hidden surprises in a program !
- They can only be triggered at
  1. Right sequence of characters
  2. Right place in the program
  3. Right time
- Easter eggs may even be triggered unintentionally !

Foundations – 5A, The Impossibility of Complete Testing

Click Here For Interactive Mindmap

@parwalrahul



Impossibility of Complete Testing II

Data Flows Caution

1. Consider what program does with variable "X"
2. Consider what values of "X" might be troublesome
3. Consider what combination of other variables can be with "X"
4. Consider what trouble can "X" cause on other variables !
5. Consider what variable depends on "X" or vice versa !
6. Consider the consequence of use !

Paths & Subpaths

- A path through a program
  - Starts at an entry program i.e. Start of the program
  - Ends at an exit point i.e. The program stops
- A subpath
  - Starts & Ends anywhere
- A subpath of length N
  - Starts, continues through N statements & then stops !

Cem Kaner testingeducation.org/BBST

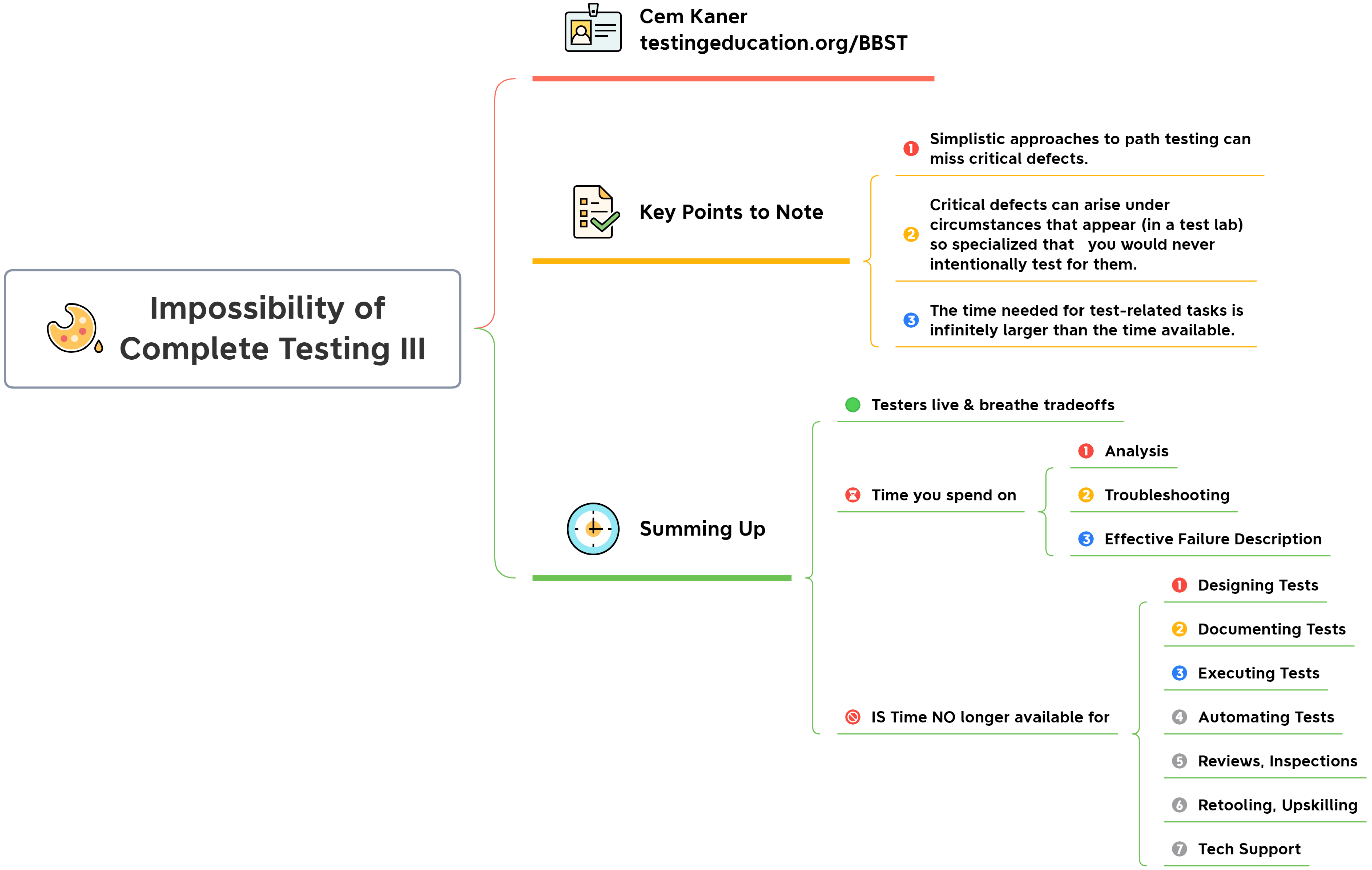
Combination Testing

- Suppose there are K independent variables, V1, V2, ..., VK Label the choices for the variables as: N1, N2, through NK The total number of combinations are: N1 X N2 X ... X NK
- Application in Configuration
  1. Let V1 be type of Printer & N1 be number of Printer
  2. Let V2 be type of Cards & N2 be number of Cards
  - Number of distinct tests: N1 X N2
  - On Adding a third variable i.e. Free memory, Number of Tests = N1 X N2 X N3
- Also applicable for combinations of data !
- The "normal" case when testing combinations of several independent variables is to adopt a " sampling" scheme. After all, we can't run "all" these number of tests !
- Popular variant on combinatorial sampling scheme is "all-pairs" !

Foundations – 5B, The Impossibility of Complete Testing

Click Here For Interactive Mindmap

@parwalrahul



Impossibility of Complete Testing III

- Cem Kaner
  testingeducation.org/BBST
- Key Points to Note
  1. Simplistic approaches to path testing can miss critical defects.
  2. Critical defects can arise under circumstances that appear (in a test lab) so specialized that you would never intentionally test for them.
  3. The time needed for test-related tasks is infinitely larger than the time available.
- Summing Up
  - Testers live & breathe tradeoffs
  - Time you spend on
    1. Analysis
    2. Troubleshooting
    3. Effective Failure Description
  - IS Time NO longer available for
    1. Designing Tests
    2. Documenting Tests
    3. Executing Tests
    4. Automating Tests
    5. Reviews, Inspections
    6. Retooling, Upskilling
    7. Tech Support

Foundations – 5C, The Impossibility of Complete Testing

Click Here For Interactive Mindmap

Chapter Six

Introduction to Measurement

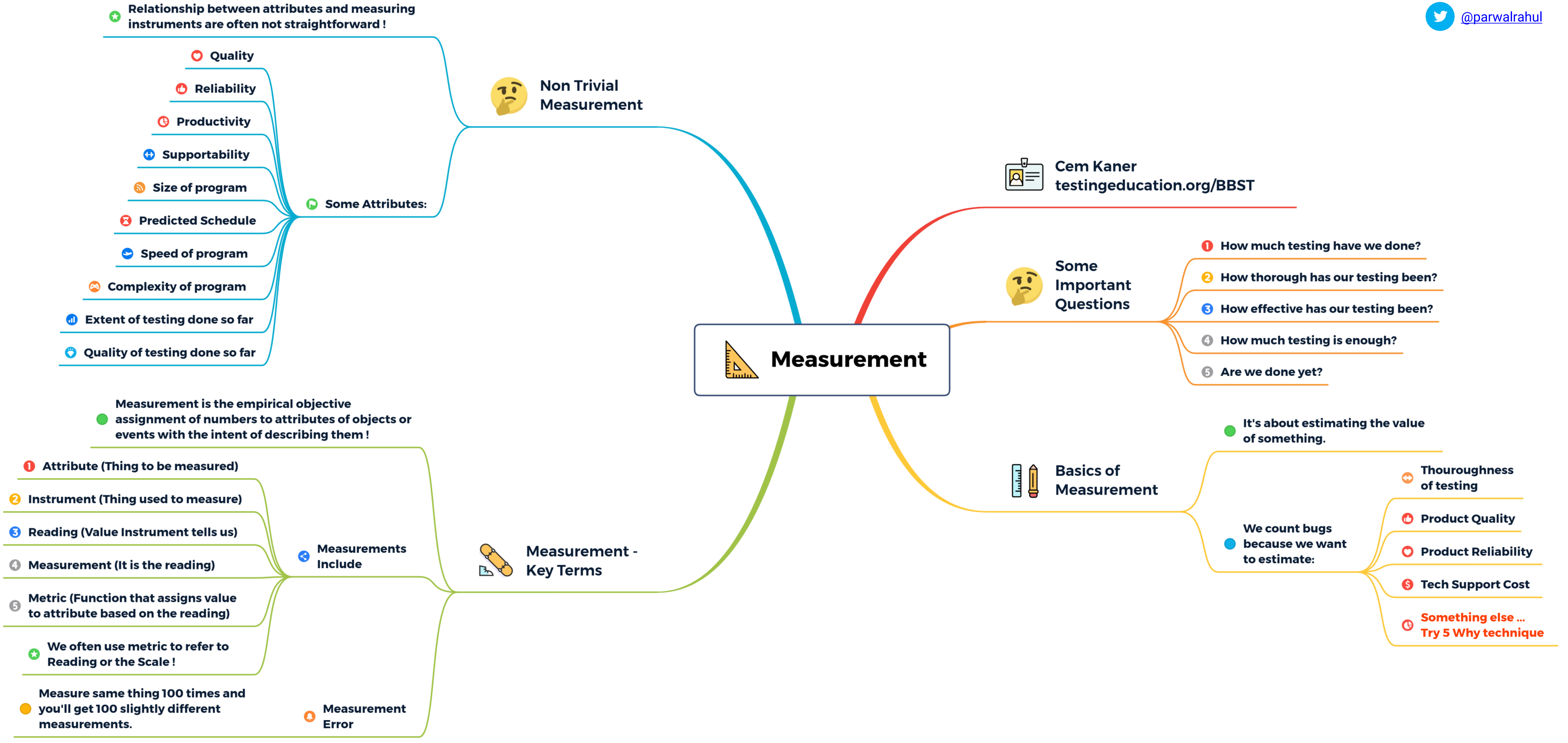
Introduction to Measurement

This chapter addresses the challenges of measurement in software testing.

Topics Covered:

- Basics of Measurement
- Measurement – Key Terms
- Non-Trivial Measurement
- Surrogate or Proxy Measures
- Early Testing
- Distortion & Dysfunction
- Recap

@parwalrahul



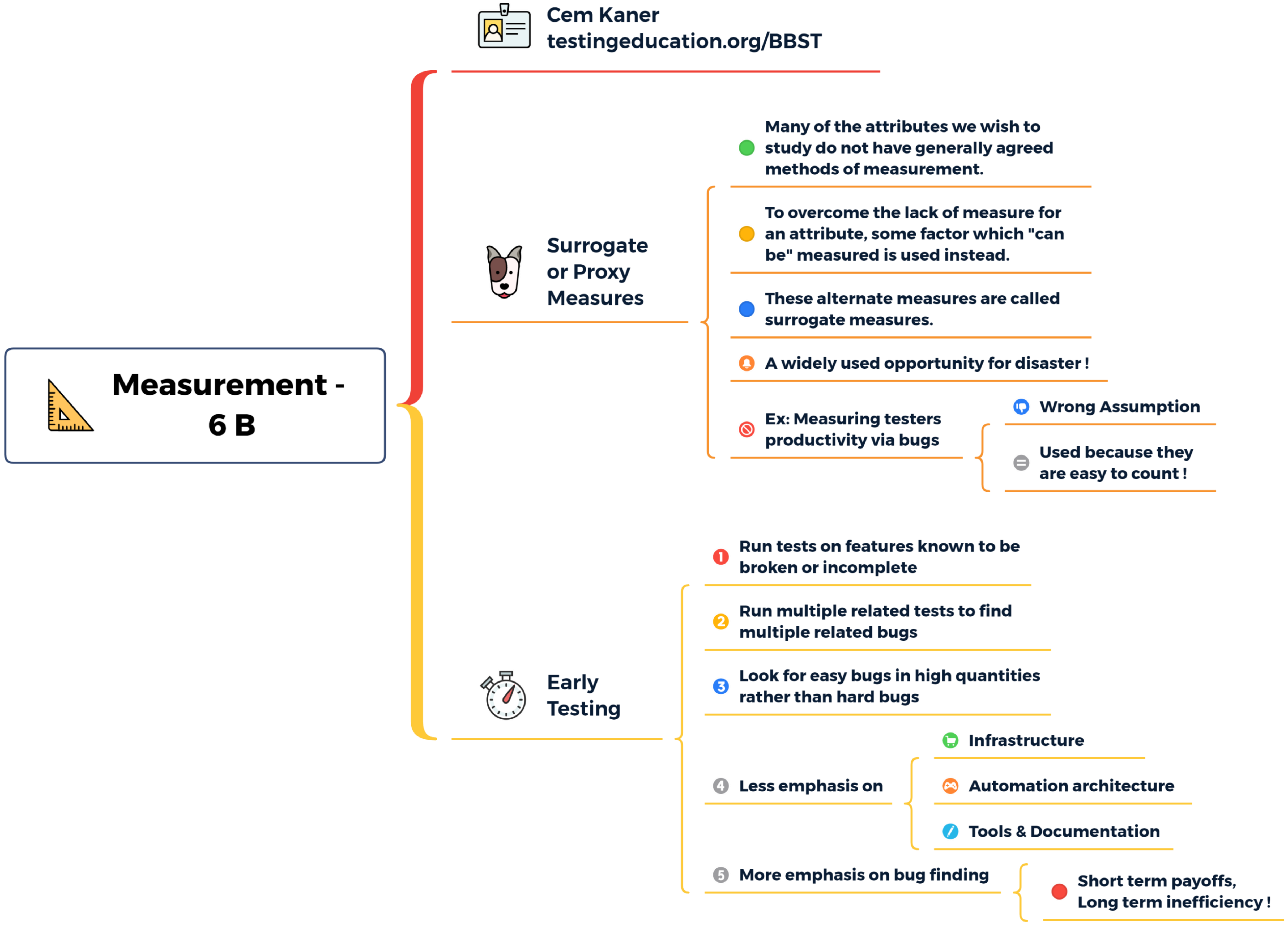
Measurement

- Non Trivial Measurement
  - Relationship between attributes and measuring instruments are often not straightforward !
  - Some Attributes:
    - Quality
    - Reliability
    - Productivity
    - Supportability
    - Size of program
    - Predicted Schedule
    - Speed of program
    - Complexity of program
    - Extent of testing done so far
    - Quality of testing done so far
- Cem Kaner testingeducation.org/BBST
- Some Important Questions
  1. How much testing have we done?
  2. How thorough has our testing been?
  3. How effective has our testing been?
  4. How much testing is enough?
  5. Are we done yet?
- Basics of Measurement
  - It's about estimating the value of something.
  - We count bugs because we want to estimate:
    - Thouroughness of testing
    - Product Quality
    - Product Reliability
    - Tech Support Cost
    - Something else ... Try 5 Why technique
- Measurement - Key Terms
  - Measurement is the empirical objective assignment of numbers to attributes of objects or events with the intent of describing them !
  - Measurements Include
    1. Attribute (Thing to be measured)
    2. Instrument (Thing used to measure)
    3. Reading (Value Instrument tells us)
    4. Measurement (It is the reading)
    5. Metric (Function that assigns value to attribute based on the reading)
    - We often use metric to refer to Reading or the Scale !
  - Measurement Error
    - Measure same thing 100 times and you'll get 100 slightly different measurements.

Foundations – 6A, Introduction to Measurement

Click Here For Interactive Mindmap

@parwalrahul



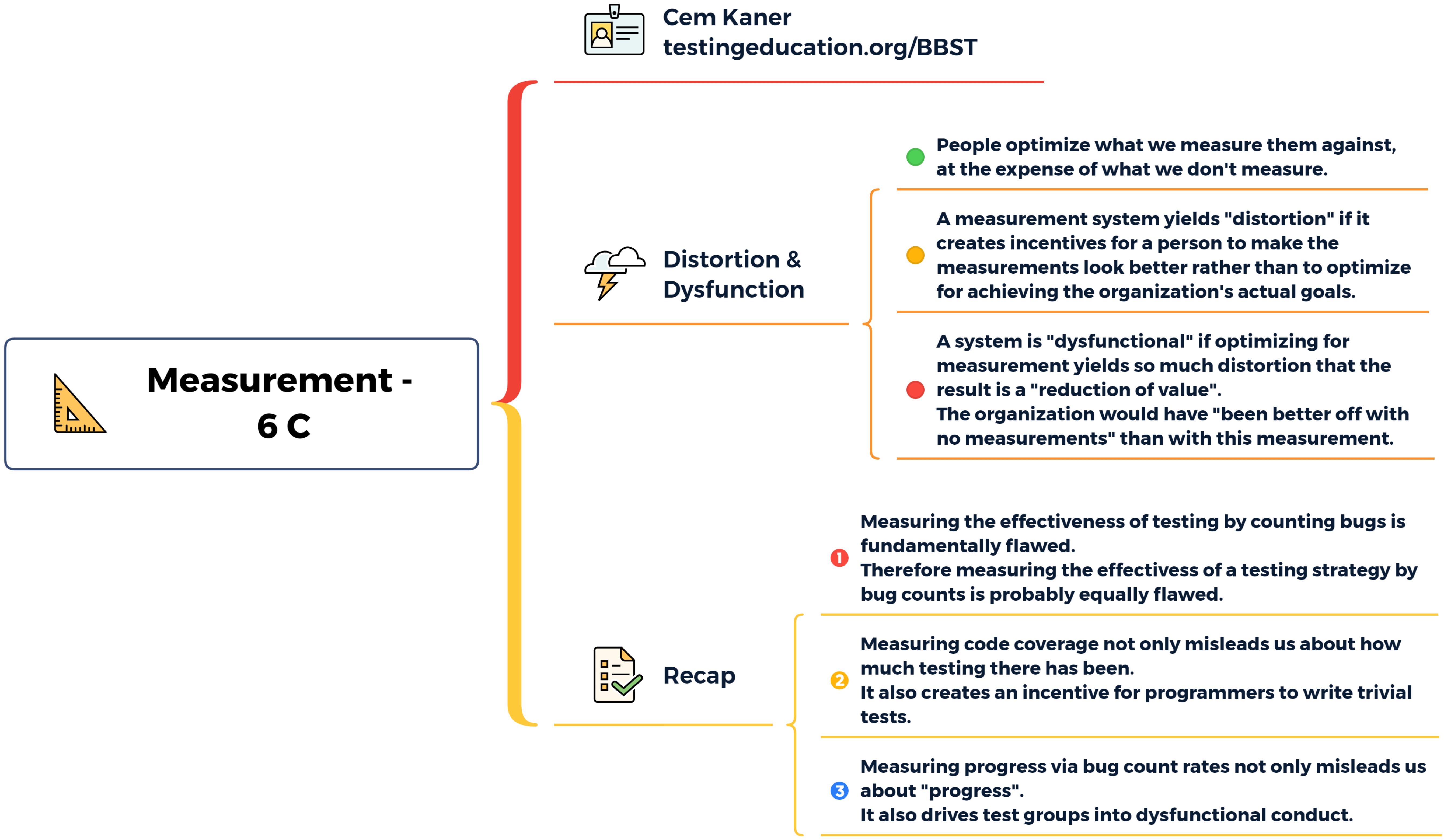
Measurement - 6 B

- Cem Kaner testingeducation.org/BBST
- Surrogate or Proxy Measures
  - Many of the attributes we wish to study do not have generally agreed methods of measurement.
  - To overcome the lack of measure for an attribute, some factor which "can be" measured is used instead.
  - These alternate measures are called surrogate measures.
  - A widely used opportunity for disaster !
  - Ex: Measuring testers productivity via bugs
    - Wrong Assumption
    - Used because they are easy to count !
- Early Testing
  1. Run tests on features known to be broken or incomplete
  2. Run multiple related tests to find multiple related bugs
  3. Look for easy bugs in high quantities rather than hard bugs
  4. Less emphasis on
     - Infrastructure
     - Automation architecture
     - Tools & Documentation
  5. More emphasis on bug finding
     - Short term payoffs, Long term inefficiency !

Foundations – 6B, Introduction to Measurement

Click Here For Interactive Mindmap

@parwalrahul



Measurement - 6 C

Cem Kaner
testingeducation.org/BBST

Distortion & Dysfunction

- People optimize what we measure them against, at the expense of what we don't measure.
- A measurement system yields "distortion" if it creates incentives for a person to make the measurements look better rather than to optimize for achieving the organization's actual goals.
- A system is "dysfunctional" if optimizing for measurement yields so much distortion that the result is a "reduction of value".
  The organization would have "been better off with no measurements" than with this measurement.

Recap

1. Measuring the effectiveness of testing by counting bugs is fundamentally flawed.
   Therefore measuring the effectivess of a testing strategy by bug counts is probably equally flawed.
2. Measuring code coverage not only misleads us about how much testing there has been.
   It also creates an incentive for programmers to write trivial tests.
3. Measuring progress via bug count rates not only misleads us about "progress".
   It also drives test groups into dysfunctional conduct.

Foundations – 6C, Introduction to Measurement

Click Here For Interactive Mindmap

Required Readings

- Michael Bolton: Testing Without a Map (PDF)
- Douglas Hoffman: Exhausting your test options (PDF)
- Cem Kaner: The impossibility of complete testing (PDF)
- Cem Kaner: Software negligence and testing coverage (PDF)
- Cem Kaner, Elisabeth Hendrickson & Jennifer Smith-Brock: Managing the proportion of testers to (other) developers (PDF)
- Brian Marick: How to misuse code coverage (PDF)

Recommended Readings - I

- Austin, Robert. (1996), Measuring and Managing Performance in Organizations (BOOK)
- James Bach: Heuristic Test Strategy Model (PDF)
- Rex Black: Factors that influence test estimation (WEBSITE)
- Michael Bolton: Meaningful metrics (PDF)
- David Goldberg: What every computer scientist should know about floating-point arithmetic (PDF)
- Douglas Hoffman: The darker side of software metrics (PDF)
- Cem Kaner and Walter P. Bond: Software engineering metrics: What do they measure and how do we know? (PDF)
- Cem Kaner: Negotiating testing resources: A collaborative approach (PDF)
- Cem Kaner: Recruiting software testers (PDF)
- Michael Kelly: Using heuristic test oracles (PDF)
- Michael Kelly: Estimating testing using spreadsheets (PDF)

Recommended Readings - II

- Billy V. Koen: The engineering method and the heuristic: A personal history ("This was the beginning of a 37 year quest to find one thing that was not a heuristic.") (PDF)
- Koen, Billy V. Definition of the Engineering Method, American Society for Engineering Education (ASEE). (A later version that is more thorough but maybe less approachable is Discussion of the Method, Oxford University Press, 2003) (BOOK)
- Jonathan Kohl: How do I Create Value with my Testing? (PDF)
- Brian Marick: Experience with the cost of different coverage goals for testing (PDF)
- Petzold, Charles. (1993), Code: The Hidden Language of Computer Hardware and Software. Microsoft Press (BOOK)
- Popper, Karl (2002, 3rd Ed.) , Conjectures and Refutations: The Growth of Scientific Knowledge (RoutledgeClassics). (BOOK)
- Erik Simmons: When will we be done testing? Software defect arrival modeling using the Weibull distribution (PDF)
- Elaine J. Weyuker: On testing nontestable programs (PDF)

Join the Community

the test tribe

Click to Join