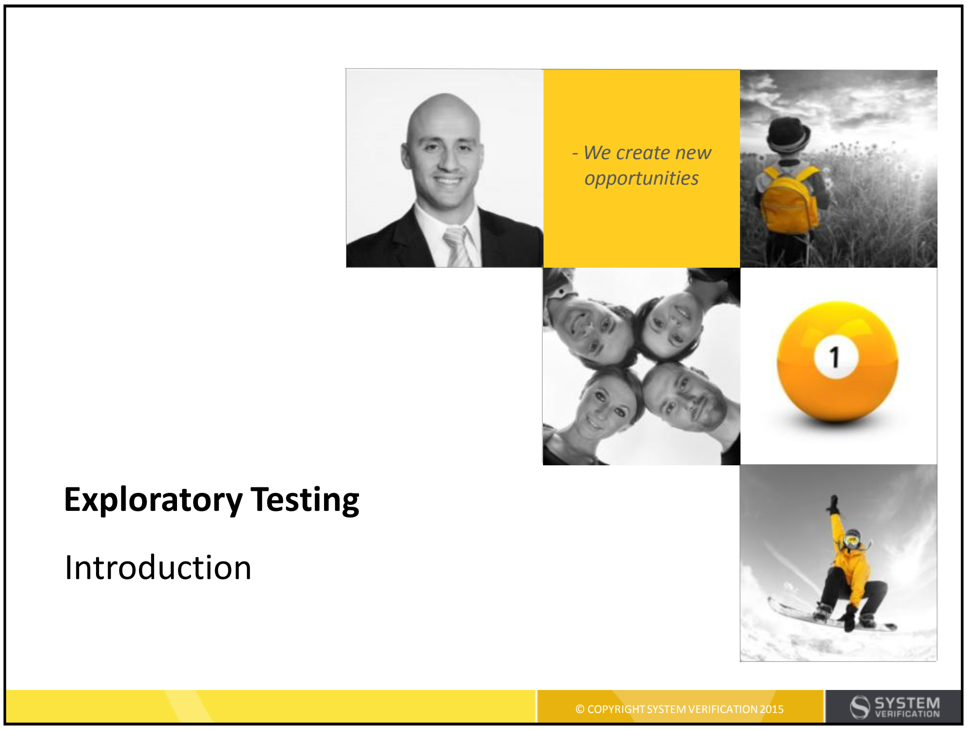
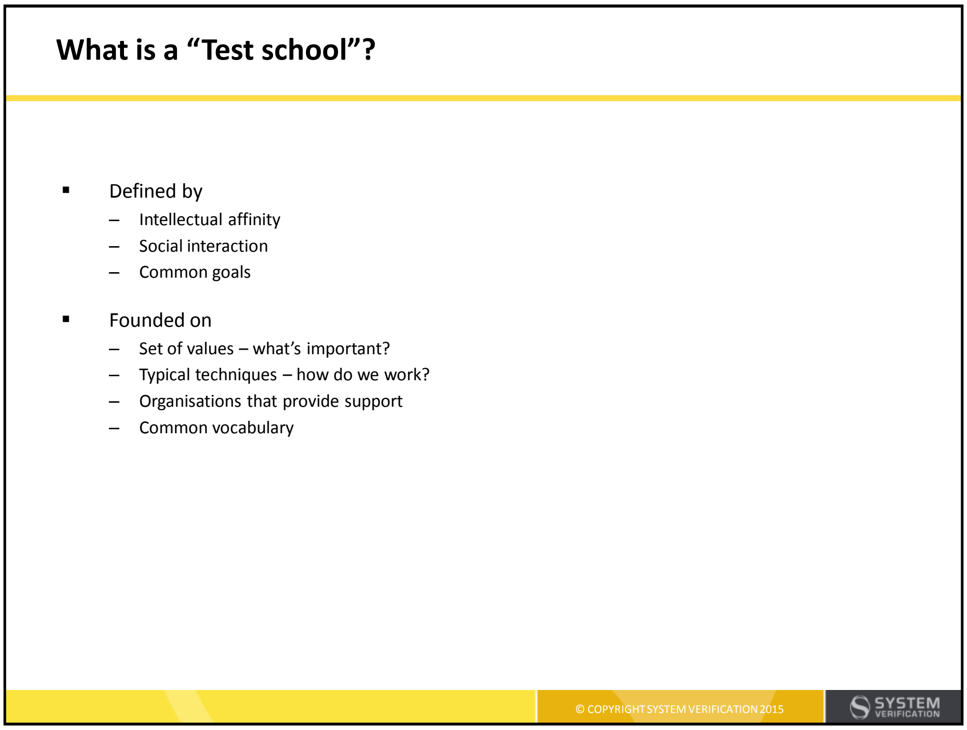


Exploratory Testing

Introduction



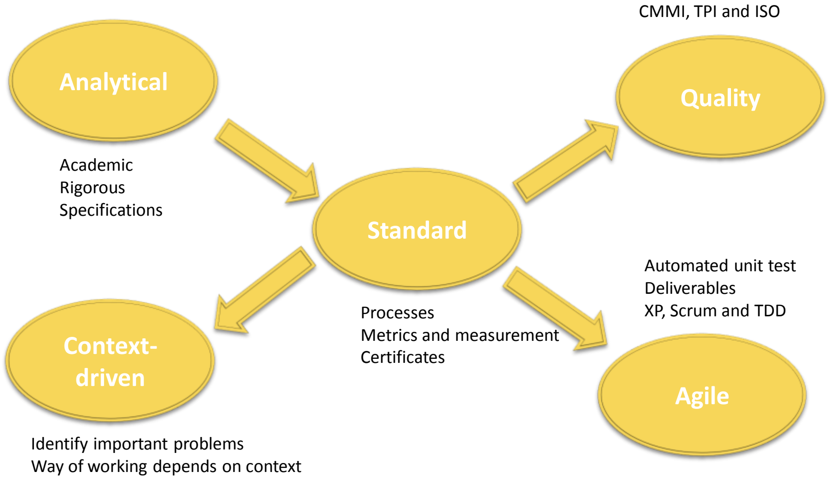
What is a “Test school”?

- Defined by
  - Intellectual affinity
  - Social interaction
  - Common goals
- Founded on
  - Set of values – what’s important?
  - Typical techniques – how do we work?
  - Organisations that provide support
  - Common vocabulary

Why talk about schools?

- Understand why experts in testing don't agree
  - Not just a matter of personality and experience
  - Many testers unaware of reasons for dissention
- Improve basis for debate

Test schools



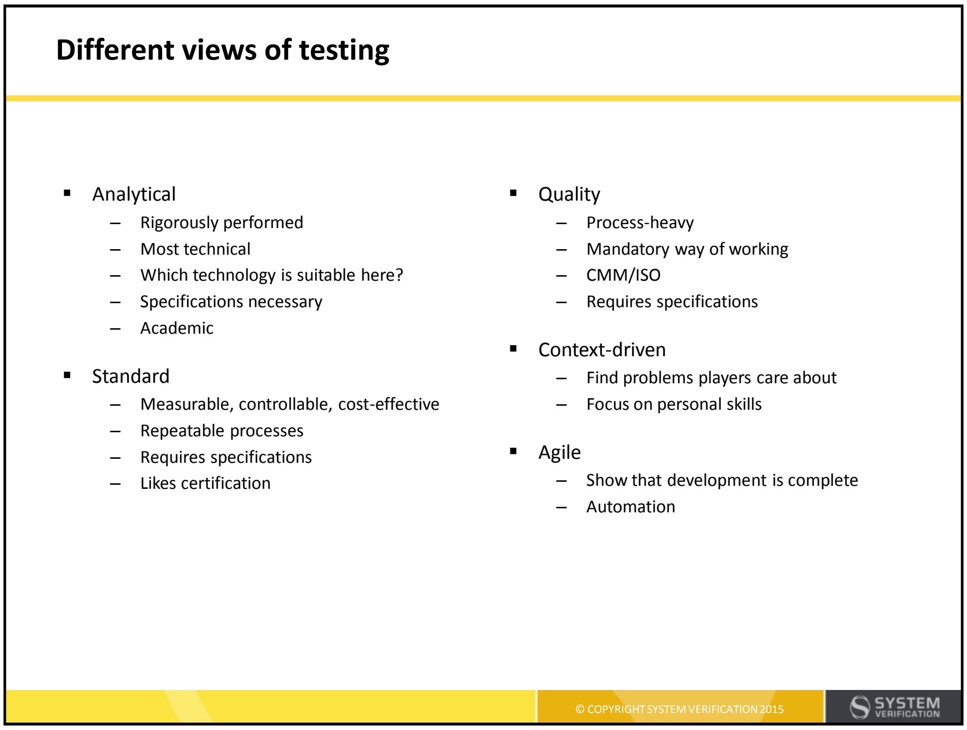
Steering or guiding
Limitations
CMMI, TPI and ISO

Academic
Rigorous
Specifications

Automated unit test
Deliverables
XP, Scrum and TDD

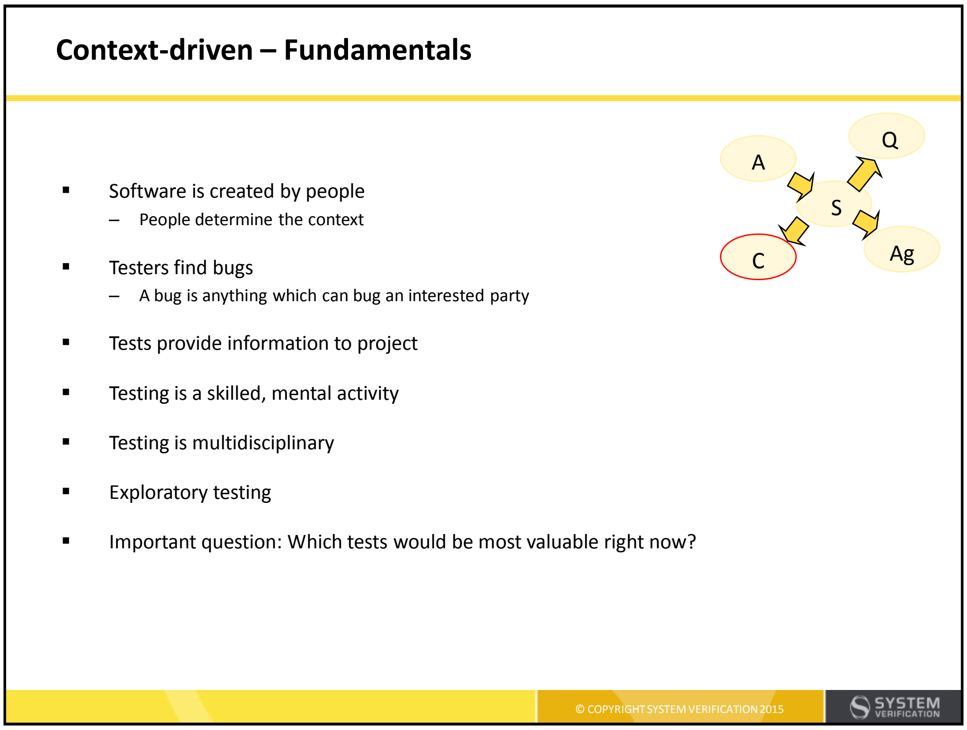
Processes
Metrics and measurement
Certificates

Identify important problems
Way of working depends on context

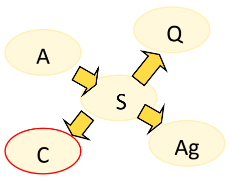


Different views of testing

- Analytical
  - Rigorously performed
  - Most technical
  - Which technology is suitable here?
  - Specifications necessary
  - Academic
- Standard
  - Measurable, controllable, cost-effective
  - Repeatable processes
  - Requires specifications
  - Likes certification
- Quality
  - Process-heavy
  - Mandatory way of working
  - CMM/ISO
  - Requires specifications
- Context-driven
  - Find problems players care about
  - Focus on personal skills
- Agile
  - Show that development is complete
  - Automation

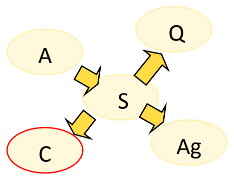


Context-driven – Fundamentals



- Software is created by people
  - People determine the context
- Testers find bugs
  - A bug is anything which can bug an interested party
- Tests provide information to project
- Testing is a skilled, mental activity
- Testing is multidisciplinary
- Exploratory testing
- Important question: Which tests would be most valuable right now?

Context-driven



- Consequences
  - Count on change
  - Adapt tests based on test results
  - Unchallenged assumptions are dangerous
  - Pragmatism
  - Effective test strategies can only be determined through real work in the field
  - Focus on skills rather than learned processes
- Most prominent
  - Commercial
  - Market-driven projects
- Authors
  - Cem Kaner, Michael Bolton and James Bach

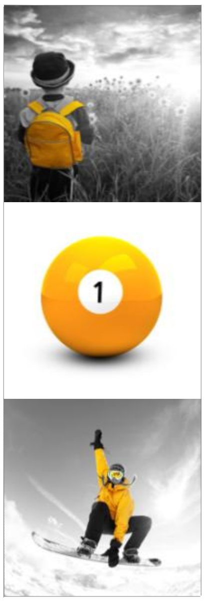
Usability – Discussion

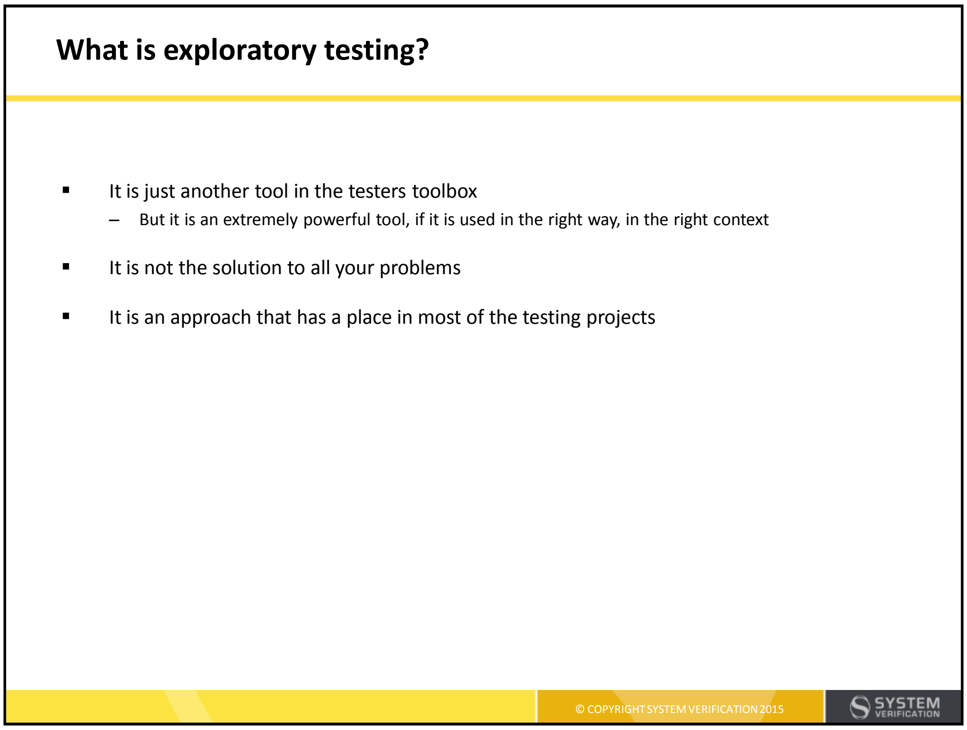
- Question: There is a button called “reset”. The button is small and is located beside a “show more” button. The user receives a confirmation query, and cannot undo the reset action.
  - Do you write an error report?
  - Classification and priority-setting, i.e. how important is it to resolve the issue?
  - Whose responsibility is it to make the decision?
  - Should the tester be responsible for usability?
    If so, to what degree?
- The developer’s reply: it’s a low-priority bug report which will not be fixed, i.e. it “works as designed”.
  - How do you respond?

Testing without specifications

- Discussion: You have been selected as test manager for an important system which will be moved from VB6 to VB.NET. The system shall function exactly as it did before. Documentation is insufficient and consists of an out-of-date user guide and a system specification written when the system was new.
  - What do you do?

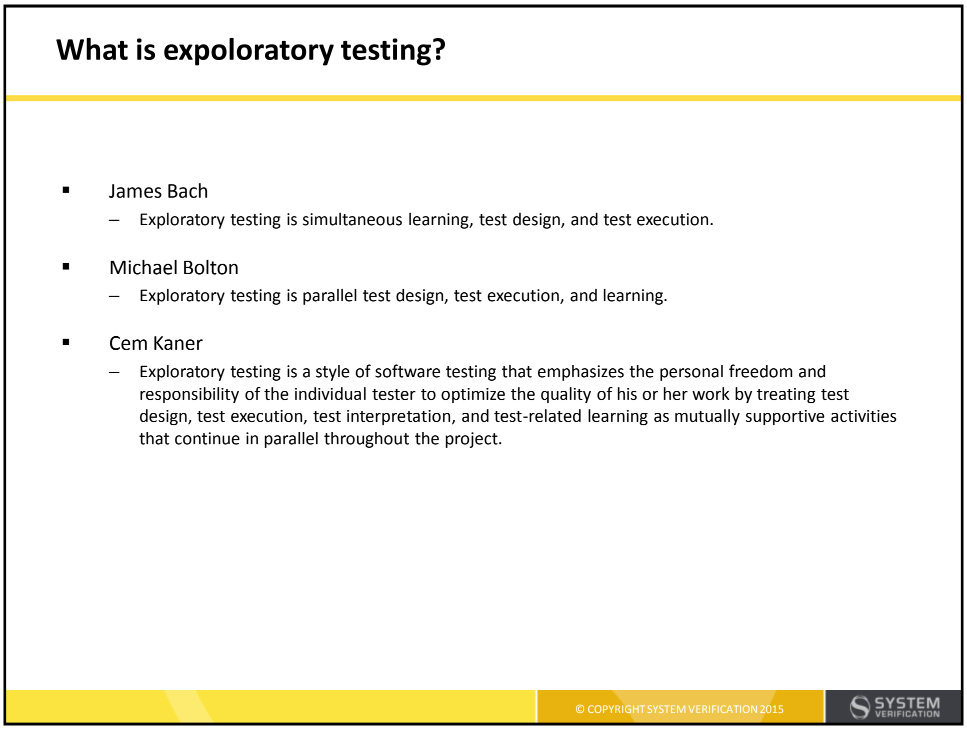
What is exploratory testing?





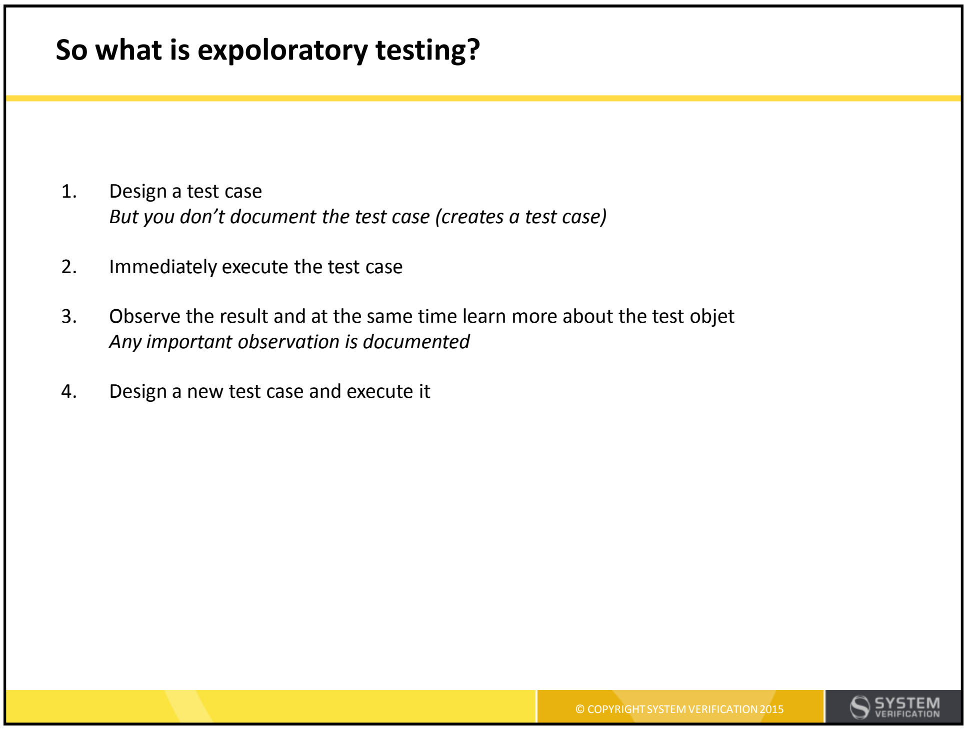
What is exploratory testing?

- It is just another tool in the testers toolbox
  - But it is an extremely powerful tool, if it is used in the right way, in the right context
- It is not the solution to all your problems
- It is an approach that has a place in most of the testing projects



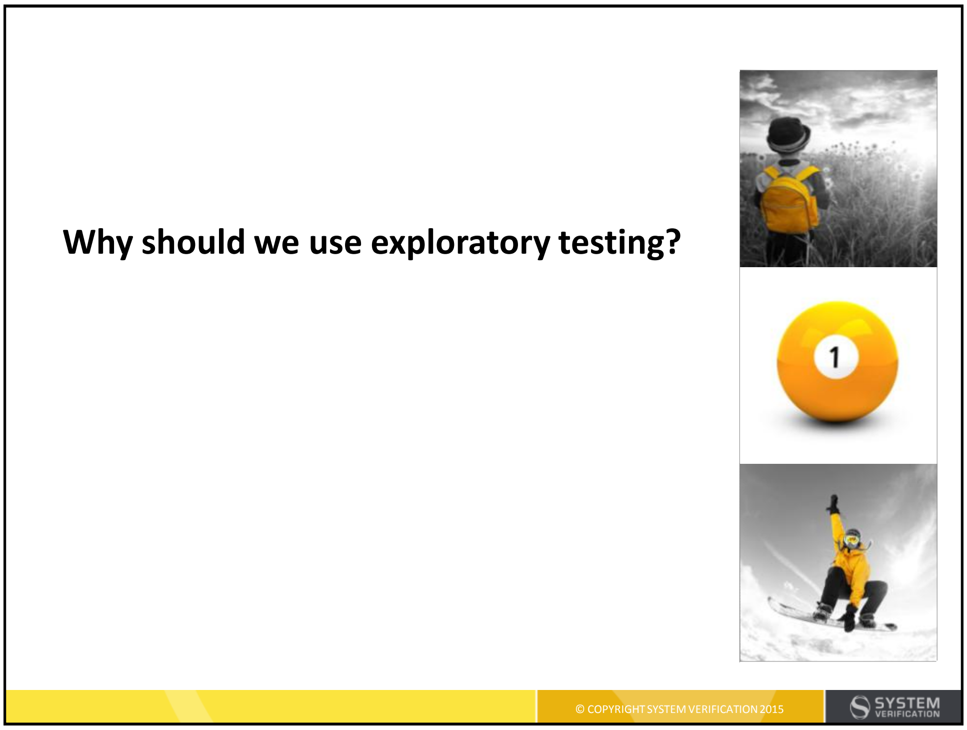
What is expoloratory testing?

- James Bach
  - Exploratory testing is simultaneous learning, test design, and test execution.
- Michael Bolton
  - Exploratory testing is parallel test design, test execution, and learning.
- Cem Kaner
  - Exploratory testing is a style of software testing that emphasizes the personal freedom and responsibility of the individual tester to optimize the quality of his or her work by treating test design, test execution, test interpretation, and test-related learning as mutually supportive activities that continue in parallel throughout the project.

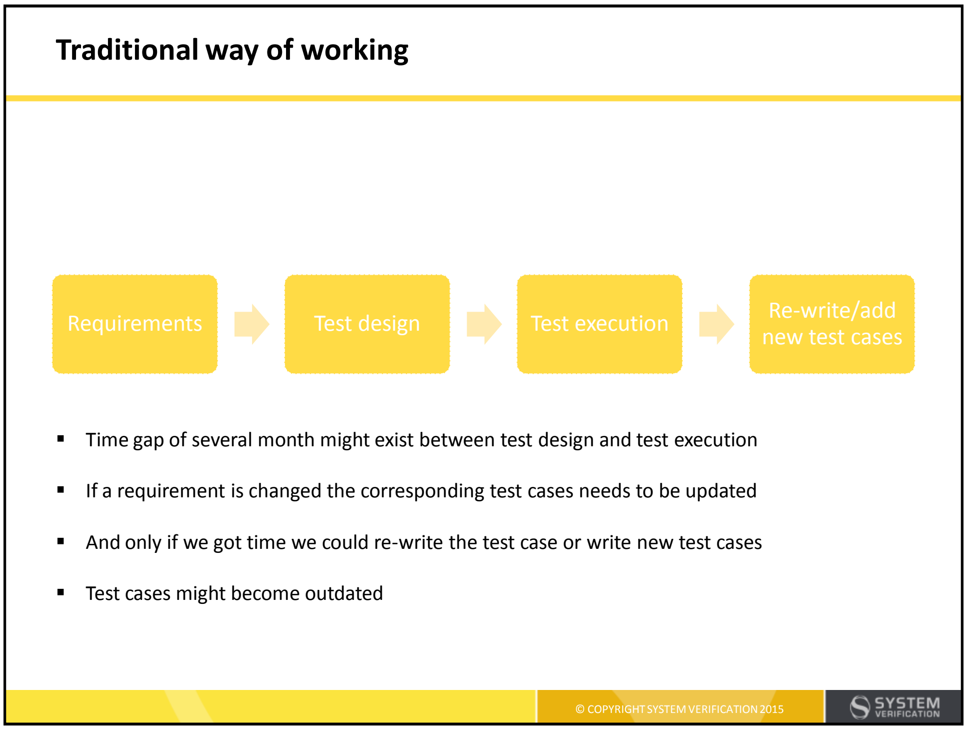


So what is expoloratory testing?

1. Design a test case
   But you don’t document the test case (creates a test case)
2. Immediately execute the test case
3. Observe the result and at the same time learn more about the test objet
   Any important observation is documented
4. Design a new test case and execute it



Why should we use exploratory testing?

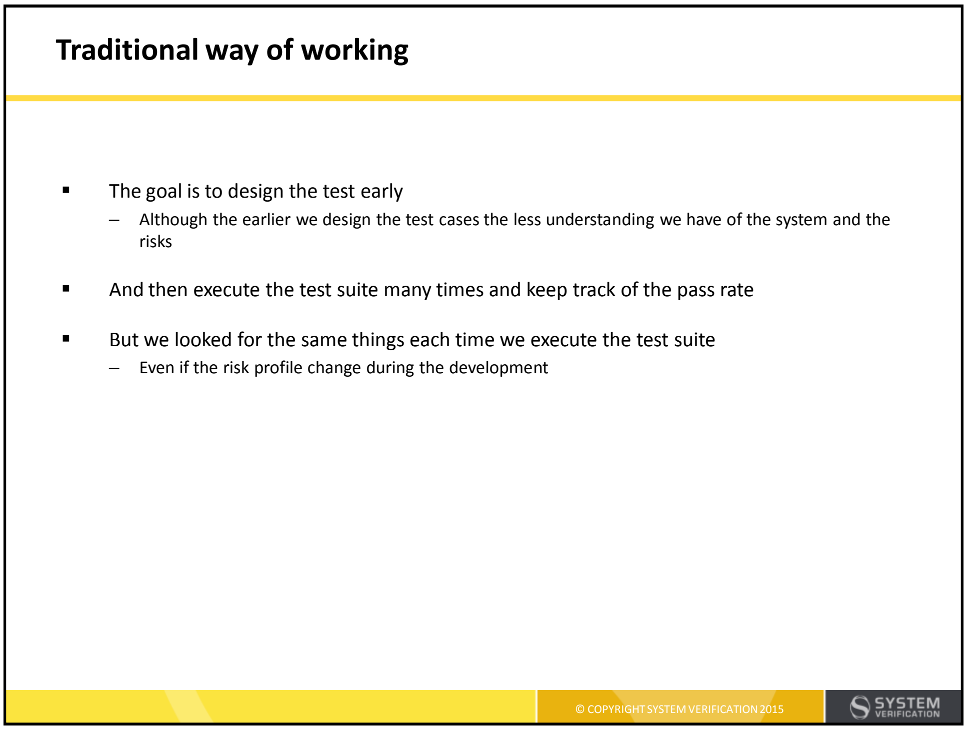


Traditional way of working

Requirements → Test design → Test execution → Re-write/add new test cases

- Time gap of several month might exist between test design and test execution
- If a requirement is changed the corresponding test cases needs to be updated
- And only if we got time we could re-write the test case or write new test cases
- Test cases might become outdated

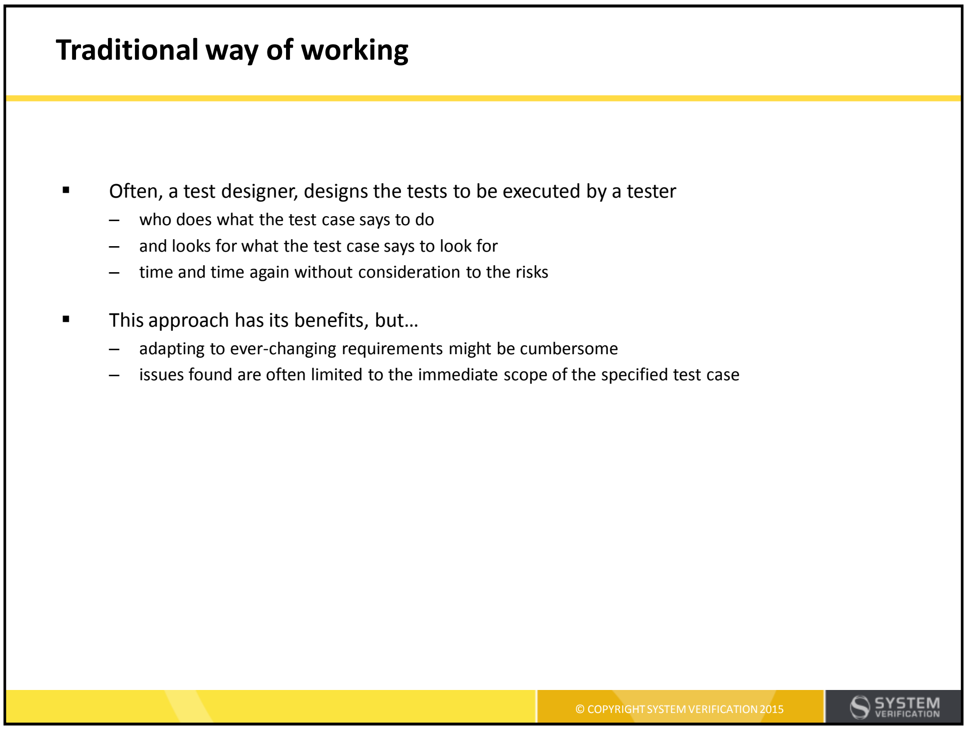
SYSTEM VERIFICATION



Traditional way of working

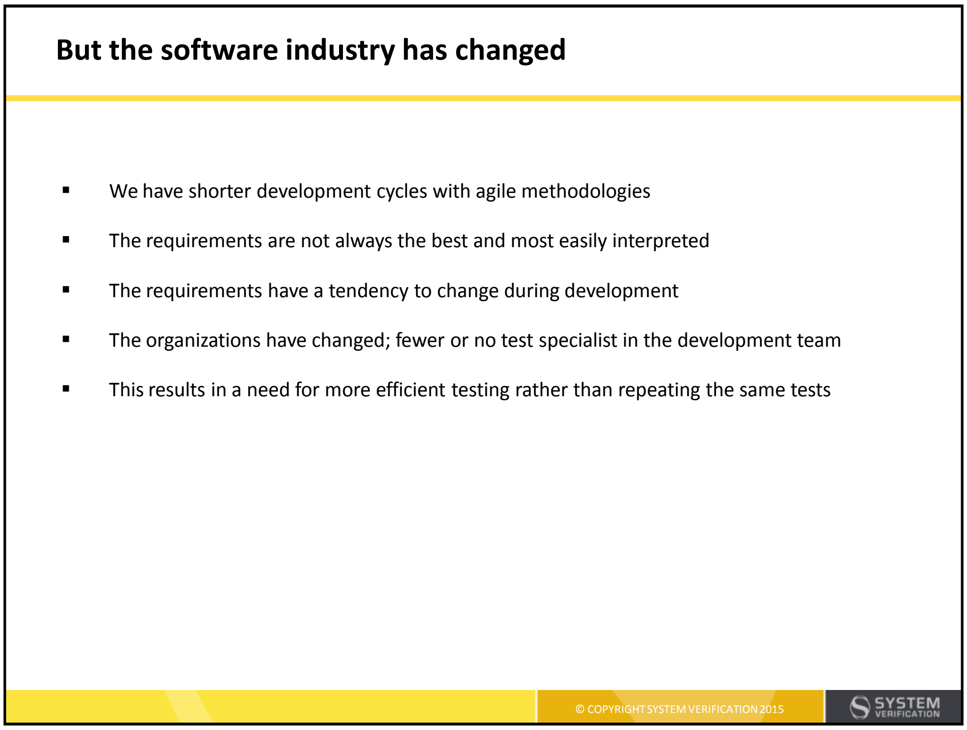
- The goal is to design the test early
  - Although the earlier we design the test cases the less understanding we have of the system and the risks
- And then execute the test suite many times and keep track of the pass rate
- But we looked for the same things each time we execute the test suite
  - Even if the risk profile change during the development

SYSTEM VERIFICATION



Traditional way of working

- Often, a test designer, designs the tests to be executed by a tester
  - who does what the test case says to do
  - and looks for what the test case says to look for
  - time and time again without consideration to the risks
- This approach has its benefits, but…
  - adapting to ever-changing requirements might be cumbersome
  - issues found are often limited to the immediate scope of the specified test case



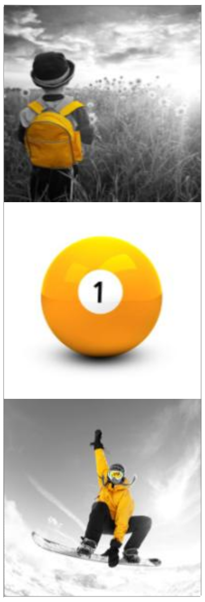
But the software industry has changed

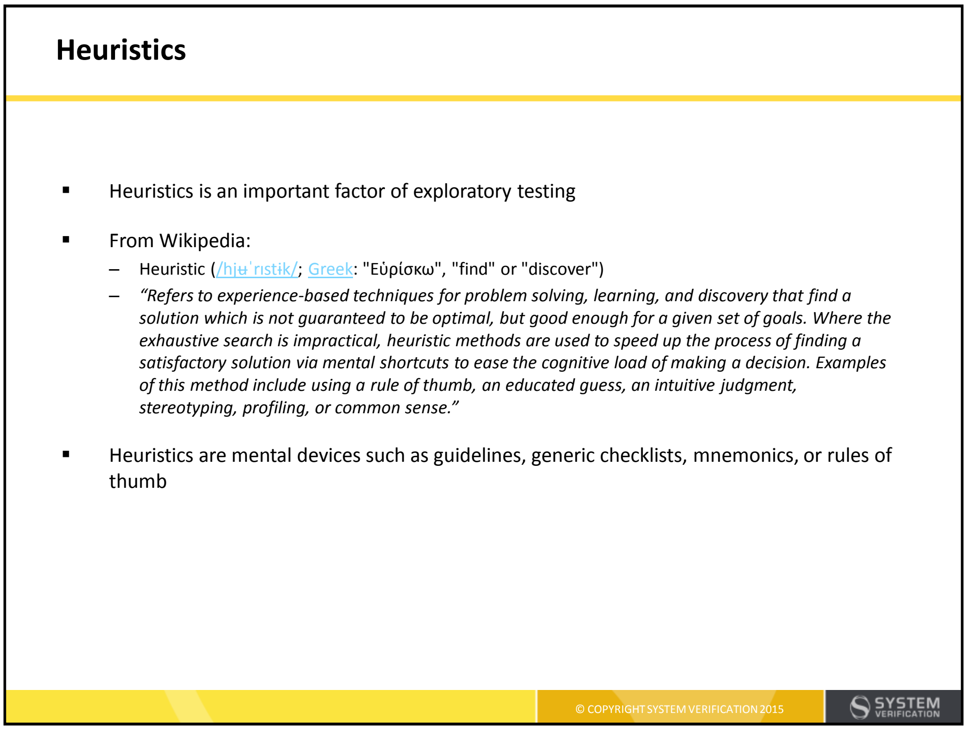
- We have shorter development cycles with agile methodologies
- The requirements are not always the best and most easily interpreted
- The requirements have a tendency to change during development
- The organizations have changed; fewer or no test specialist in the development team
- This results in a need for more efficient testing rather than repeating the same tests

What we need is

- What we need is
  - To be smarter when we test!
  - A constantly evolving set of tests that exercise the software in new ways (new combinations of features and data)
- So that we get our choice of
  - broader coverage of the infinite space of possibilities adapting as we recognize new classes of possibilities.
  - sharper focus on risks or issues that we decide are of critical interest to day.

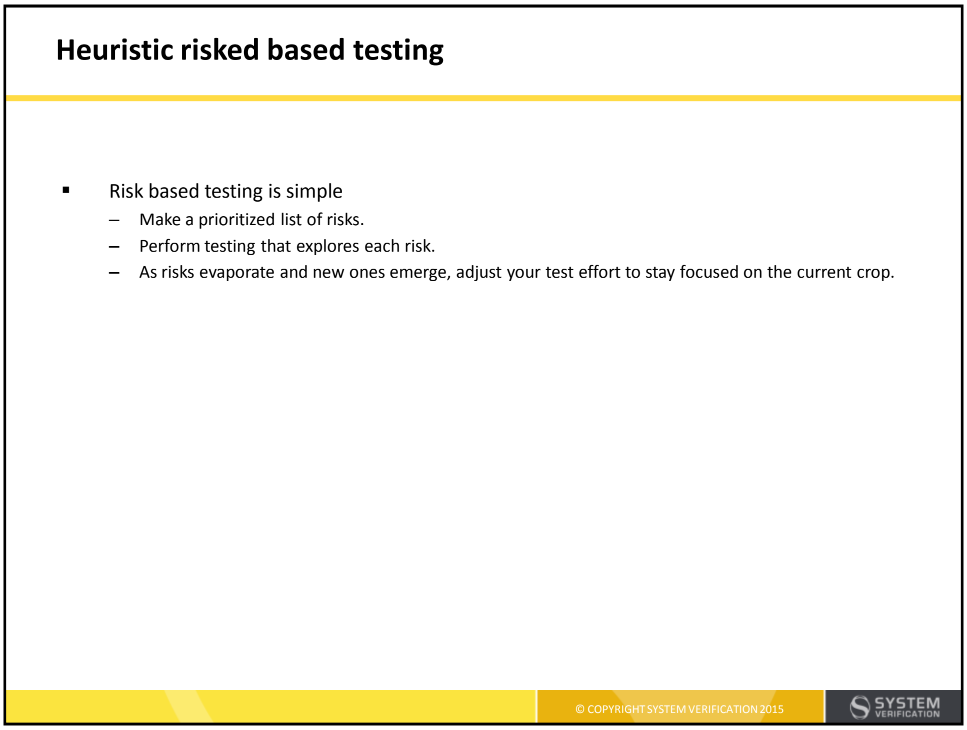
Definitions





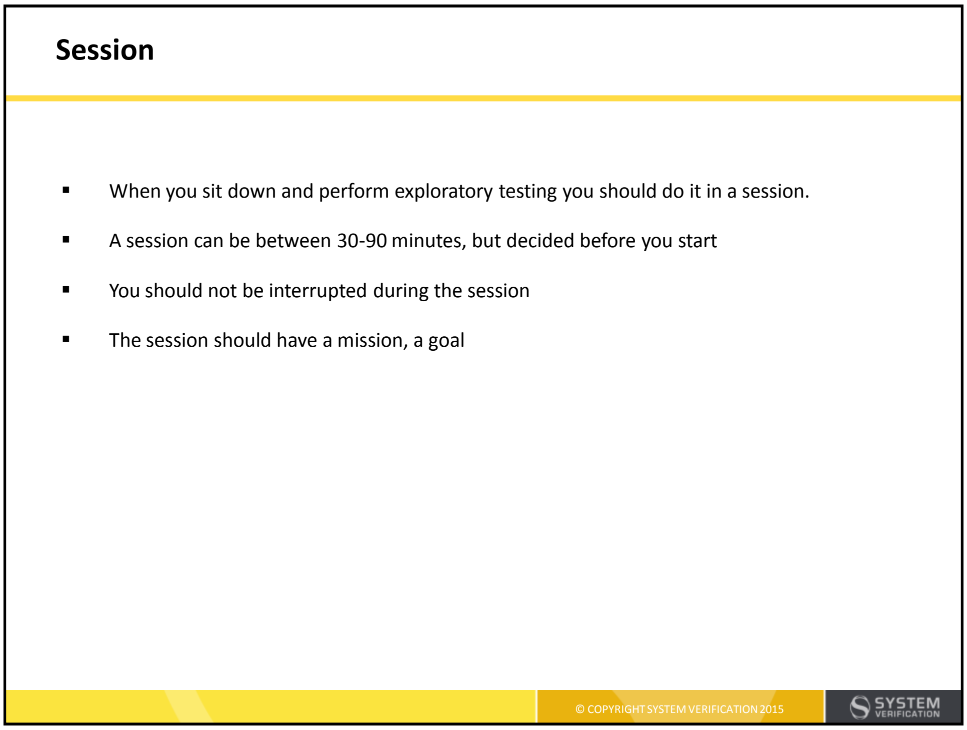
Heuristics

- Heuristics is an important factor of exploratory testing
- From Wikipedia:
  - Heuristic (/hjʉˈrɪstɨk/; Greek: "Εὑρίσκω", "find" or "discover")
  - *"Refers to experience-based techniques for problem solving, learning, and discovery that find a solution which is not guaranteed to be optimal, but good enough for a given set of goals. Where the exhaustive search is impractical, heuristic methods are used to speed up the process of finding a satisfactory solution via mental shortcuts to ease the cognitive load of making a decision. Examples of this method include using a rule of thumb, an educated guess, an intuitive judgment, stereotyping, profiling, or common sense."*
- Heuristics are mental devices such as guidelines, generic checklists, mnemonics, or rules of thumb



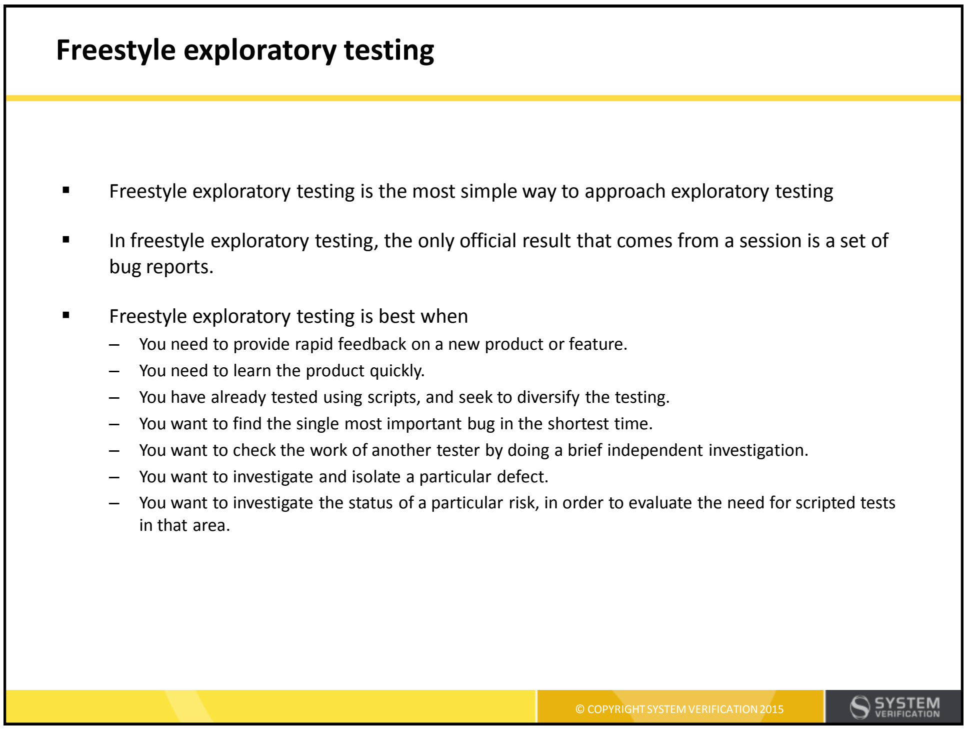
Heuristic risked based testing

- Risk based testing is simple
  - Make a prioritized list of risks.
  - Perform testing that explores each risk.
  - As risks evaporate and new ones emerge, adjust your test effort to stay focused on the current crop.



Session

- When you sit down and perform exploratory testing you should do it in a session.
- A session can be between 30-90 minutes, but decided before you start
- You should not be interrupted during the session
- The session should have a mission, a goal



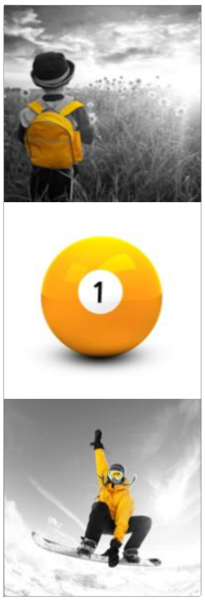
Freestyle exploratory testing

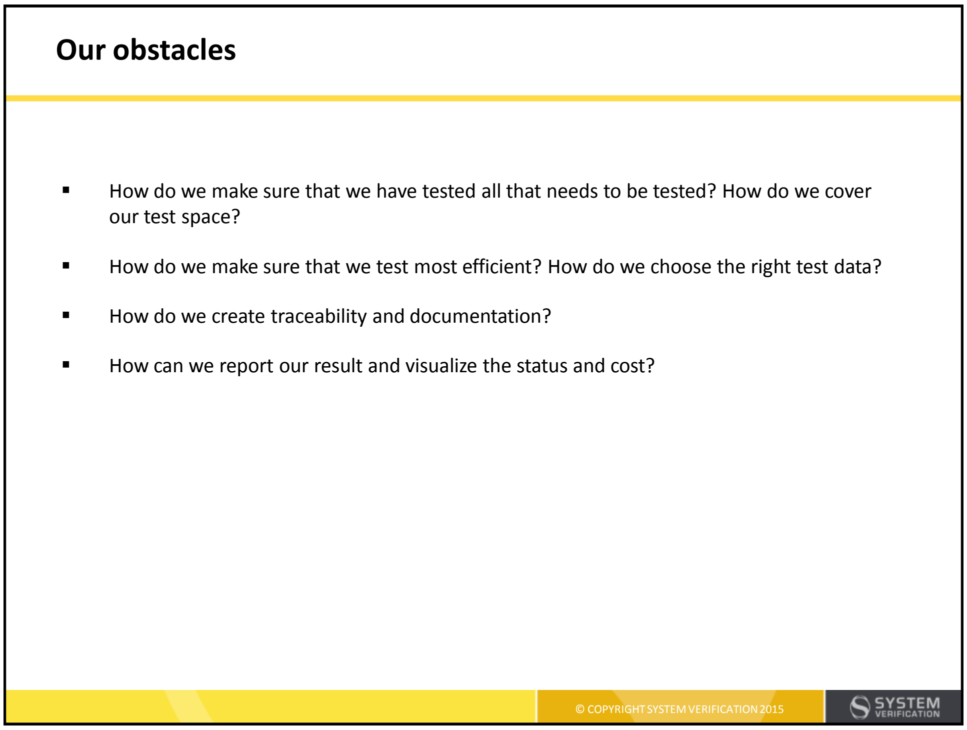
- Freestyle exploratory testing is the most simple way to approach exploratory testing
- In freestyle exploratory testing, the only official result that comes from a session is a set of bug reports.
- Freestyle exploratory testing is best when
  – You need to provide rapid feedback on a new product or feature.
  – You need to learn the product quickly.
  – You have already tested using scripts, and seek to diversify the testing.
  – You want to find the single most important bug in the shortest time.
  – You want to check the work of another tester by doing a brief independent investigation.
  – You want to investigate and isolate a particular defect.
  – You want to investigate the status of a particular risk, in order to evaluate the need for scripted tests in that area.

Session based test management

- Session based test management is designed for a more structured and measurable ET
- In session based test management, each session results in a set of written notes that are reviewed by the test lead and bug reports.
- It may also result in updated test materials or new test data.
- The following metrics are measured:
  - Number of sessions completed
  - Number of problems found
  - Function areas covered
  - Percentage of session time spent setting up for testing
  - Percentage of session time spent testing
  - Percentage of session time spent investigating problems
- In the session-based approach, test reports are written, and testers are interviewed at least once per day.

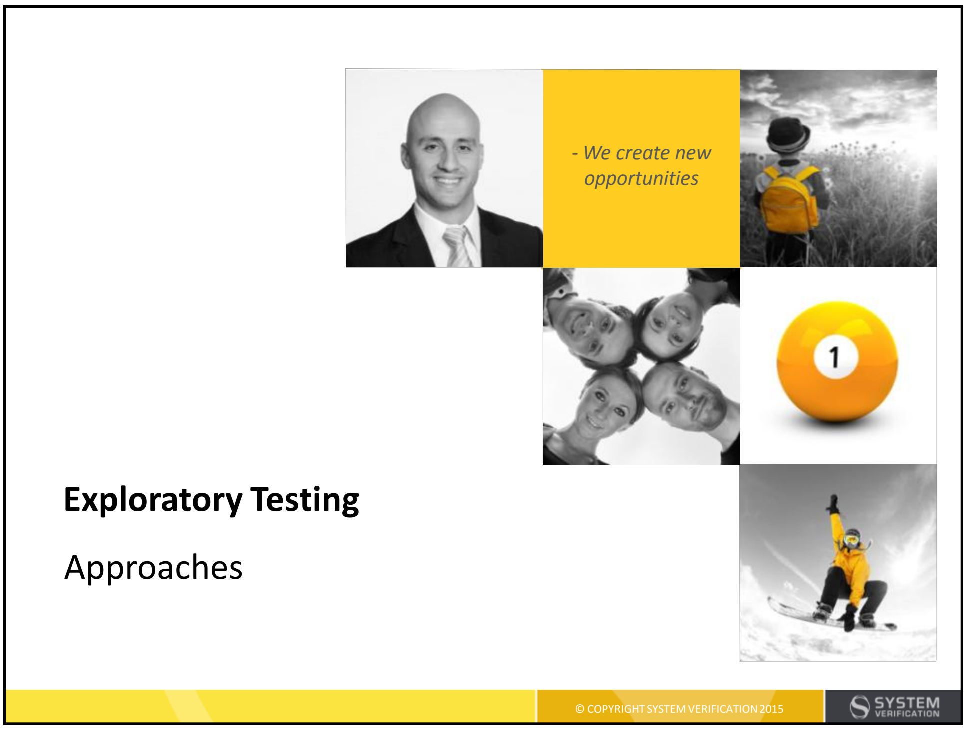
Obstacles





Our obstacles

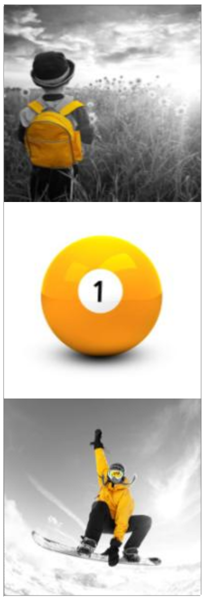
- How do we make sure that we have tested all that needs to be tested? How do we cover our test space?
- How do we make sure that we test most efficient? How do we choose the right test data?
- How do we create traceability and documentation?
- How can we report our result and visualize the status and cost?



Exploratory Testing

Approaches

Test mission and Test space



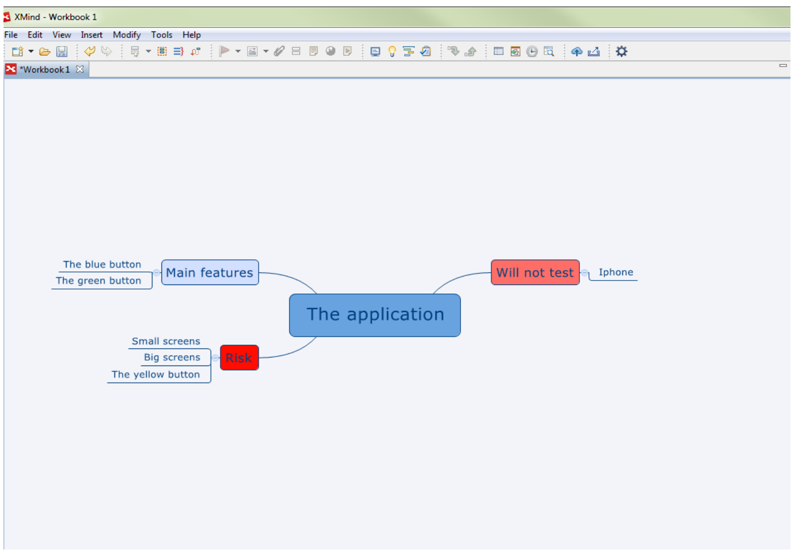
Structure

- How do we make sure that we have tested all that needs to be tested? How do we cover our test space?
  - First we have to have a test plan that defines what to test
  - Then we have to define our missions
  - If the project is small and simple, we can start
  - But if the project is bigger and/or more complex we need to ensure that we cover the full test space
    Then we need to choose an approach, a way to divide the area

Test plan

- A test plan is still needed
- The test plan should be used to define the test space
  - What should be included in the test activity?
- And more importantly; what we don't test and why
- Should be kept simple
  - Mindmap or one, single page
- List of reminders

Example of test plan



Missions

- Each session needs a mission, i.e. a goal
- Examples of missions
  - To gain an understanding of how an application works, what its interface looks like, and what functionality it implements
  - To force the software to exhibit its capabilities
  - To find bugs
  - To ensure that a bug has been fixed and that no new bugs has been introduced

Different approaches to exploratory testing

- Sometimes when the project is bigger and/or more complex the tester might need help to structure the testing. Here are five methods
  - Tours
  - Heuristics
  - RCRCRC
  - SFDPOT
  - ISO 25010

Testing Tours by James Whittaker

- James Whittaker is a professor and software testing evangelist at Microsoft, and has previously worked at Google as well. During his time on Google he developed a way of performing exploratory testing, which he called Testing Tours.
  - *"Suppose you are visiting a large city like London, England, for the very first time. It's a big, busy, confusing place for new tourists, with lots of things to see and do. Indeed, even the richest, most time-unconstrained tourist would have a hard time seeing everything a city like London has to offer. The same can be said of well-equipped testers trying to explore complex software; all the funding in the world won't guarantee completeness."*

Testing Tours by James Whittaker

- **The Guidebook Tour** – Follow the user manual's advice just like the wary traveler, by never deviating from its lead
- **The Money Tour** – Run through sales demos to make sure everything that is used for sales purposes works
- **The Landmark Tour** – Choose a set of features, decide on an ordering for them, and then explore the application going from feature to feature until you've explored all of them in your list.
- **The Intellectual Tour** – this tour takes on the approach of asking the software hard questions. How do we make the software work as hard as possible?
- **The FedEx Tour** – During this tour, a tester must concentrate on the data. Try to identify inputs that are stored and "follow" them around the software.

Testing Tours by James Whittaker

- **The Garbage Collector's Tour** – This is like a methodical spot check. We can decide to spot check the interface where we go screen by screen, dialog by dialog (favoring, like the garbage collector, the shortest route), and not stopping to test in detail, but checking the obvious things.
- **The Bad-Neighborhood Tour** – Software also has bad neighborhoods—those sections of the code populated by bugs. This tour is about running tests in those sections of the code.
- **The Museum Tour** – During this tour, testers should identify older code and executable artifacts and ensure they receive a fair share of testing attention.
- **The Back Alley Tour** – If your organization tracks feature usage, this tour will direct you to test the ones at the bottom of the list. If your organization tracks code coverage, this tour implores you to find ways to test the code yet to be covered.
- **The All-Nighter Tour** – Exploratory testers on the All-Nighter tour will keep their application running without closing it.

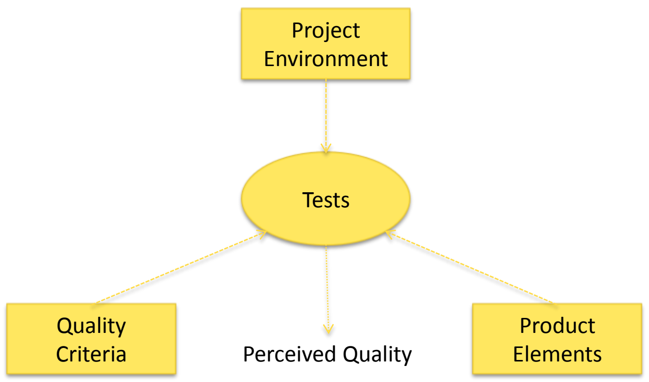
Testing Tours by James Whittaker

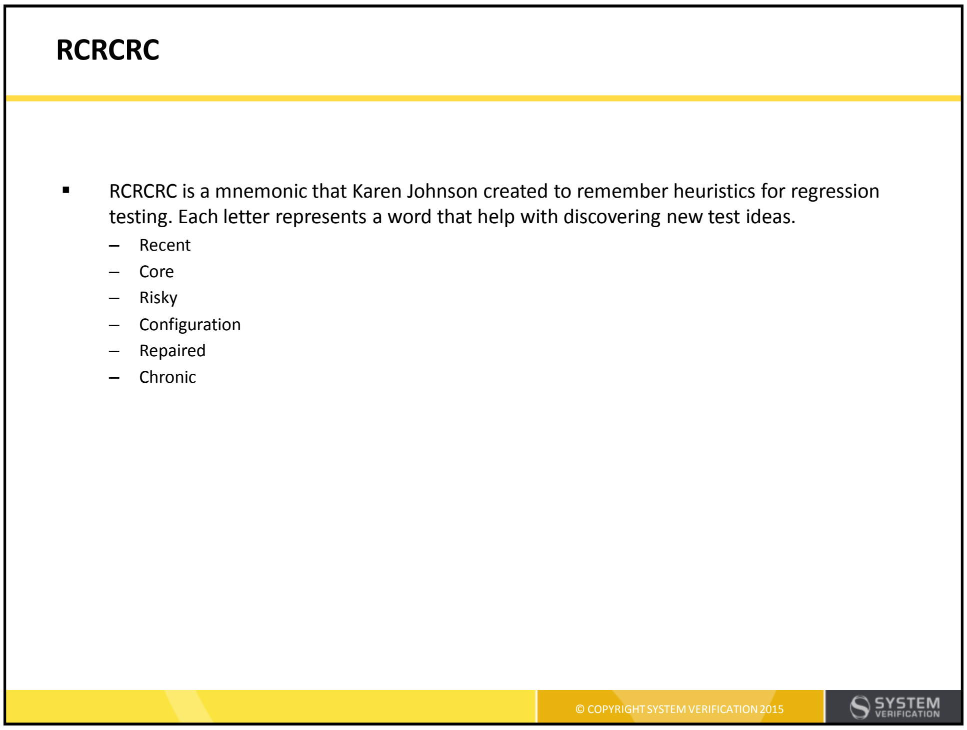
- **The Supermodel Tour** – During the Supermodel tour, the focus is not on functionality or real interaction. It's only on the interface.
- **The Couch Potato Tour** – A Coach Potato tour means doing as little actual work as possible. This means accepting all default values (values prepopulated by the application), leaving input fields blank, filling in as little form data as possible, never clicking on an advertisement, paging through screens without clicking any buttons or entering any data, and so forth.
- **The Obsessive-Compulsive Tour** – Perform the same action over and over. Repeat, redo, copy, paste, borrow, and then do all that some more.

Heuristics by James Bach

- James Bach, one of the most famous software testing experts, has created a list of general test techniques that are simple and universal enough to be applicable in many different contexts.
  - Function testing – Test what it can do
  - Domain testing – Look for any data processed by the product
  - Stress testing – Overwhelm the product
  - Flow testing – Do one thing after another
  - Scenario testing – Test to a compelling story
  - Claims testing – Verify every claim
  - User testing – Involve the users
  - Risk testing – Imagine a problem, then look for it
  - Automatic checking – Check a million different facts

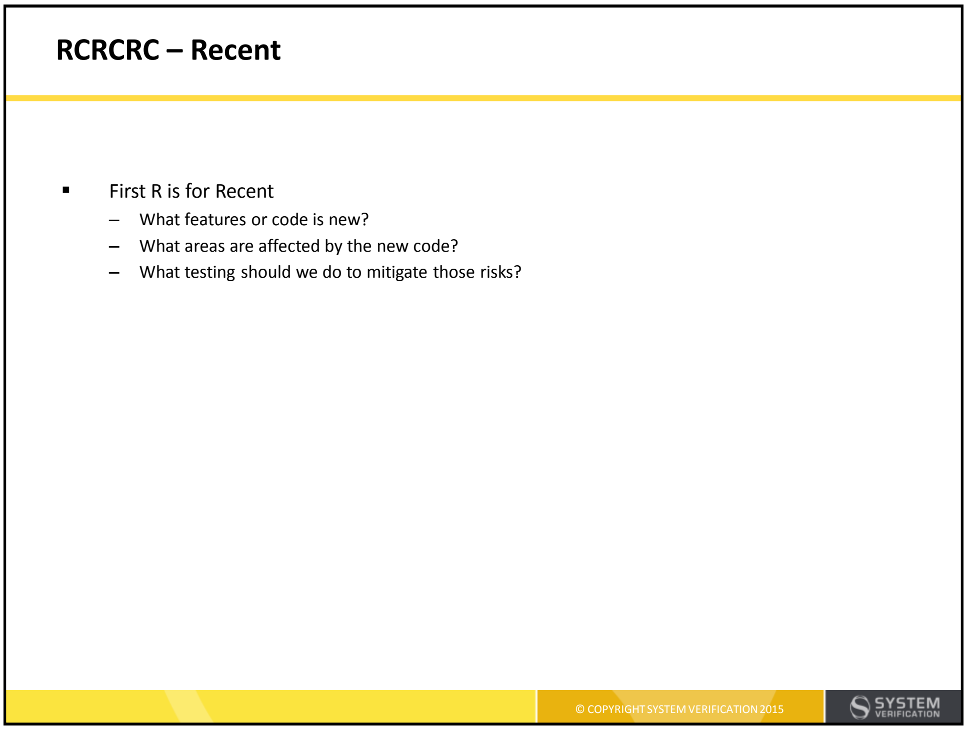
A heuristic test strategy model





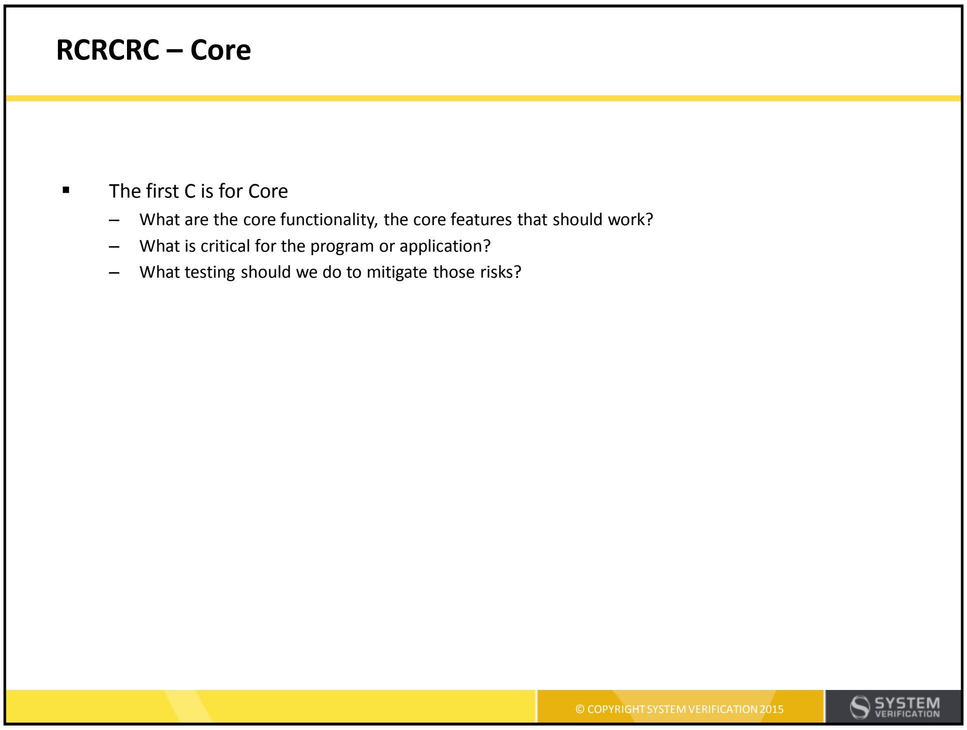
RCRCRC

- RCRCRC is a mnemonic that Karen Johnson created to remember heuristics for regression testing. Each letter represents a word that help with discovering new test ideas.
  - Recent
  - Core
  - Risky
  - Configuration
  - Repaired
  - Chronic



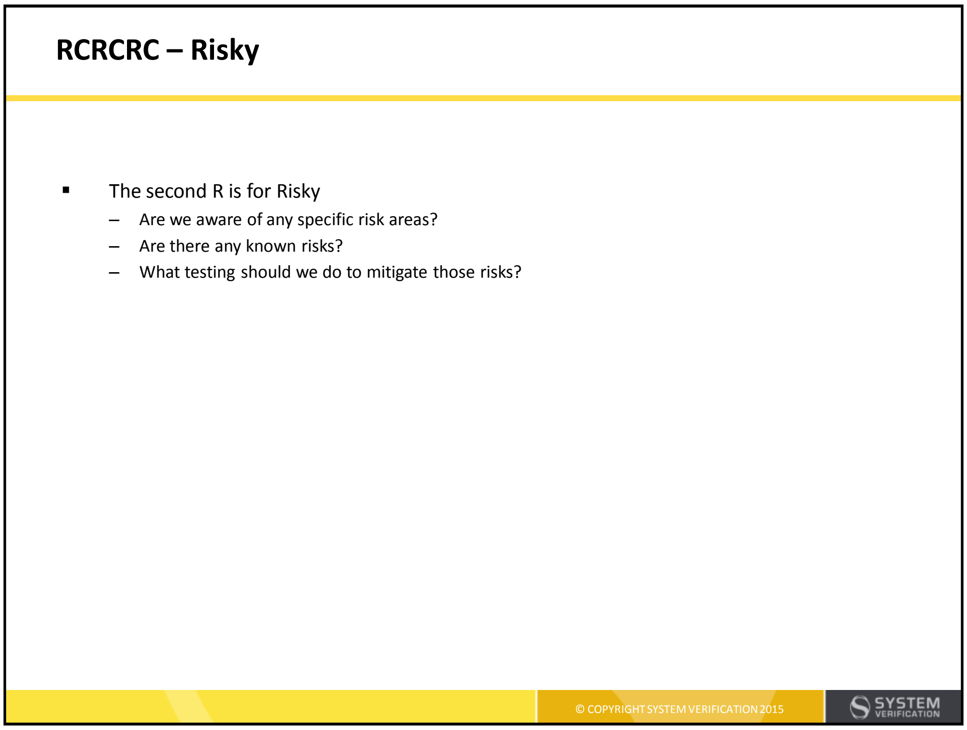
RCRCRC – Recent

- First R is for Recent
  - What features or code is new?
  - What areas are affected by the new code?
  - What testing should we do to mitigate those risks?



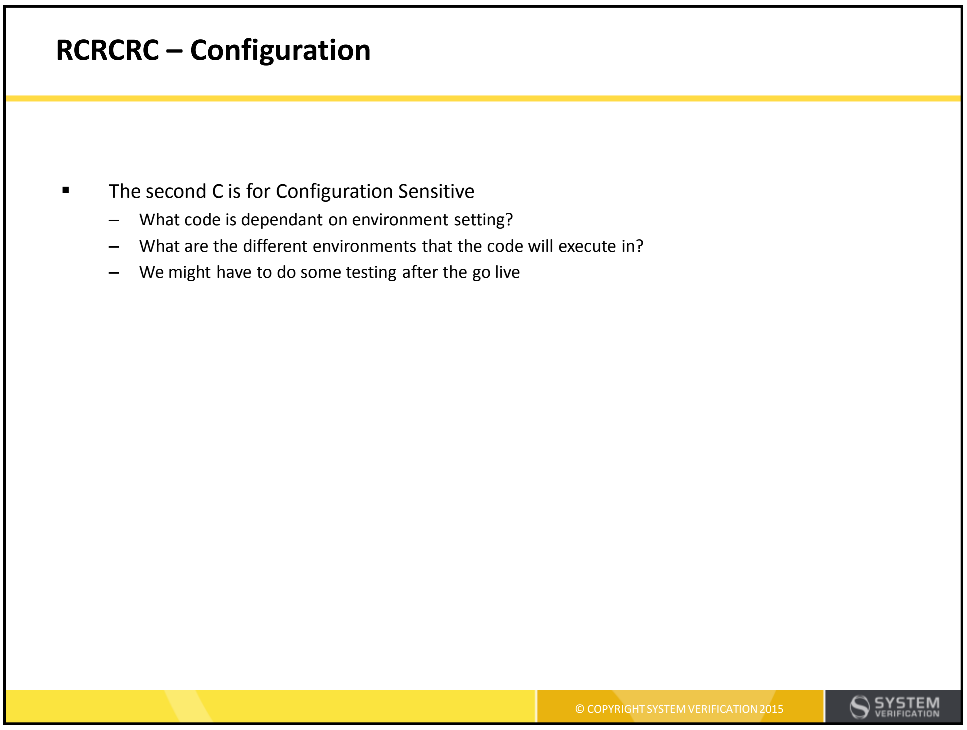
RCRCRC – Core

- The first C is for Core
  - What are the core functionality, the core features that should work?
  - What is critical for the program or application?
  - What testing should we do to mitigate those risks?



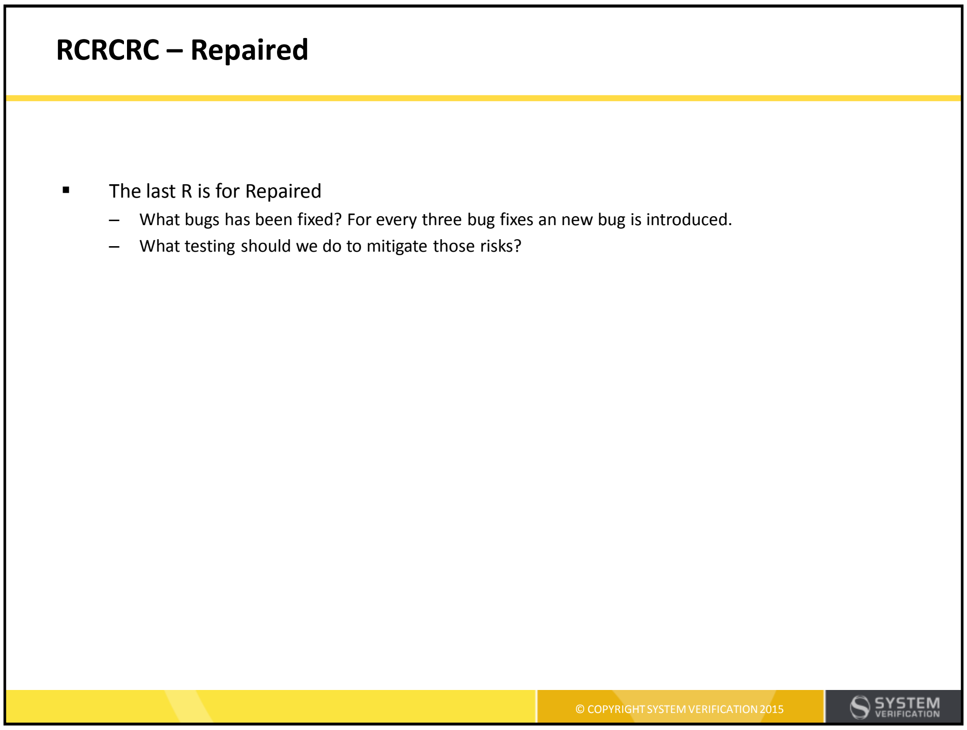
RCRCRC – Risky

- The second R is for Risky
  - Are we aware of any specific risk areas?
  - Are there any known risks?
  - What testing should we do to mitigate those risks?



RCRCRC – Configuration

- The second C is for Configuration Sensitive
  - What code is dependant on environment setting?
  - What are the different environments that the code will execute in?
  - We might have to do some testing after the go live



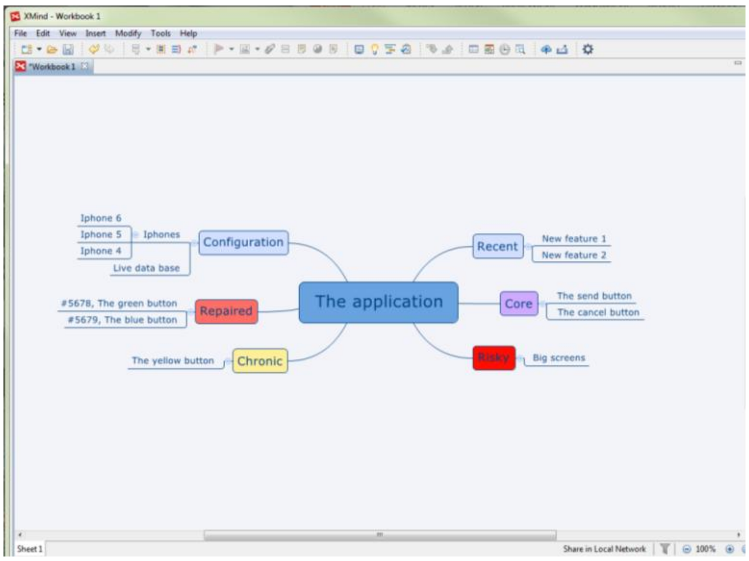
RCRCRC – Repaired

- The last R is for Repaired
  - What bugs has been fixed? For every three bug fixes an new bug is introduced.
  - What testing should we do to mitigate those risks?

RCRCRC – Chronic

- The last C is for Chronic
  - What areas has had lots of bugs?
  - Where is it prone to break?

Example

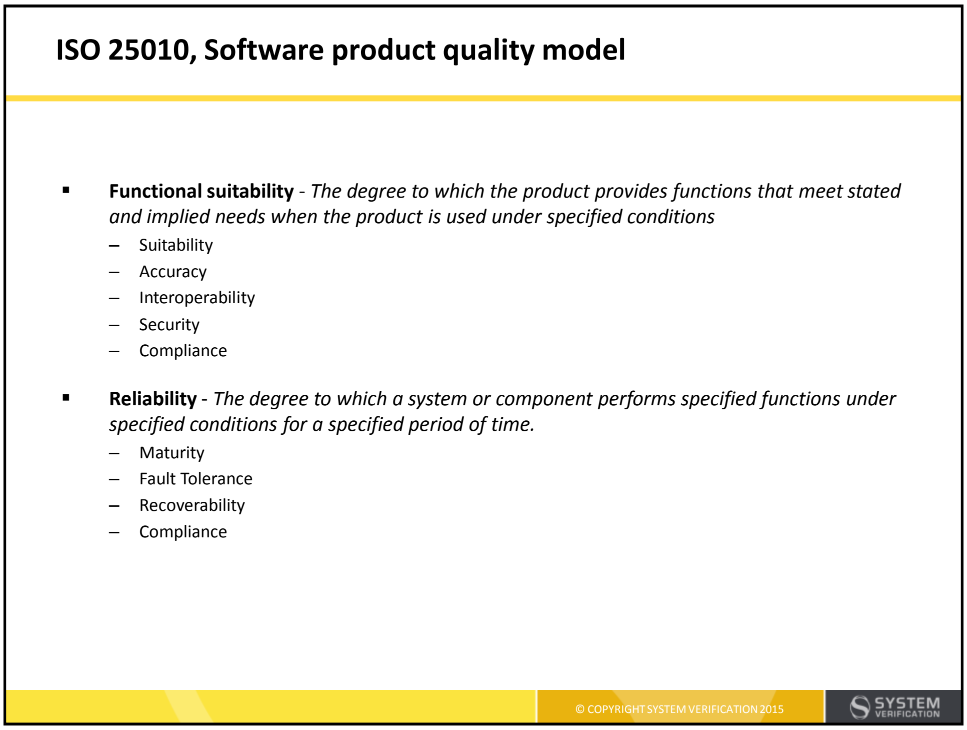


SFDIPOT (San Francisco Depot)

- Test Strategy Heuristics by James Bach
  - **S**tructure
  - **F**unction
  - **D**ata
  - **I**ntegrations
  - **P**latform
  - **O**perations
  - **T**ime

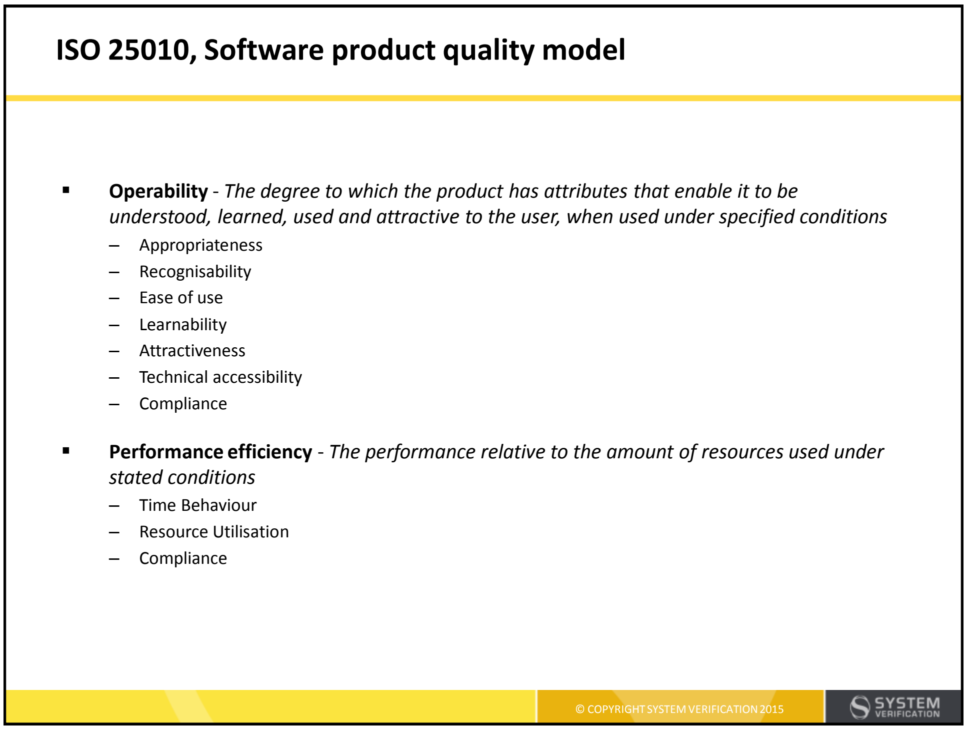
Using ISO 25010

- Eight characteristics
- Besides the software product quality model, the 25010 standard also describes another model, the model of software quality in use



ISO 25010, Software product quality model

- **Functional suitability** - *The degree to which the product provides functions that meet stated and implied needs when the product is used under specified conditions*
  - Suitability
  - Accuracy
  - Interoperability
  - Security
  - Compliance
- **Reliability** - *The degree to which a system or component performs specified functions under specified conditions for a specified period of time.*
  - Maturity
  - Fault Tolerance
  - Recoverability
  - Compliance



ISO 25010, Software product quality model

- **Operability** - *The degree to which the product has attributes that enable it to be understood, learned, used and attractive to the user, when used under specified conditions*
  - Appropriateness
  - Recognisability
  - Ease of use
  - Learnability
  - Attractiveness
  - Technical accessibility
  - Compliance
- **Performance efficiency** - *The performance relative to the amount of resources used under stated conditions*
  - Time Behaviour
  - Resource Utilisation
  - Compliance

ISO 25010, Software product quality model

- **Security** - *The degree of protection of information and data so that unauthorized persons or systems cannot read or modify them and authorized persons or systems are not denied access to them*
  - Confidentiality
  - Integrity
  - Non-repudiation
  - Accountability
  - Authenticity
  - Compliance
- **Compatibility** - *The degree to which two or more systems or components can exchange information and/or perform their required functions while sharing the same hardware or software environment*
  - Replaceability
  - Co-existence
  - Interoperability
  - Compliance

ISO 25010, Software product quality model

- **Maintainability** - *The degree of effectiveness and efficiency with which the product can be modified*
  - Modularity
  - Reusability
  - Analyzability
  - Changeability
  - Modification stability
  - Testability
  - Compliance
- **Transferability** - *The degree to which a system or component can be effectively and efficiently transferred from one hardware, software or other operational or usage environment to another*
  - Portability
  - Adaptability
  - Installability
  - Compliance

ISO 25010, Software quality in use

- **Effectiveness** - *The accuracy and completeness with which users achieve specified goals*
  - Effectiveness
- **Efficiency** - *The resources expended in relation to the accuracy and completeness with which users achieve goals*
  - Efficiency
- **Satisfaction** - *The degree to which users are satisfied with the experience of using a product in a specified context of use*
  - Likability
  - Pleasure
  - Comfort
  - Trust

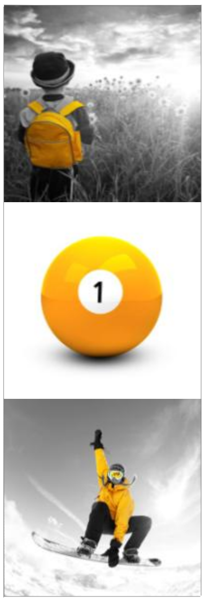
ISO 25010, Software quality in use

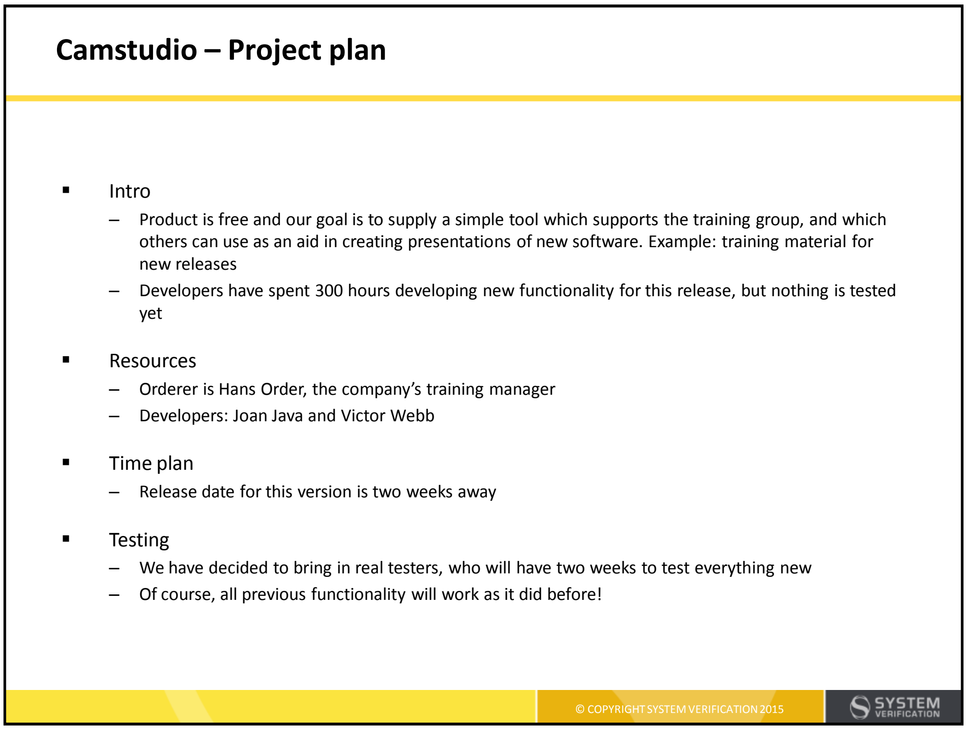
- **Safety** - *The degree to which a product or system does not, under specified conditions, lead to a state in which human life, health, property, or the environment is endangered*
  - Economic damage risk
  - Health and safety risk
  - Environmental harm risk
- **Usability** - *The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use*
  - Learnability
  - Flexibility
  - Accessability
  - Context conformity

More ways to get test ideas

- Requirements – Always good to have the requirements
- Maps – Creating graphical representation of the application might reveal places that we have not visited
- Lists – There are a lot of good lists to use. Elisabeth Hendrickson's "cheat sheet" http://testobsessed.com/wp-content/uploads/2011/04/testheuristicscheatsheetv1.pdf
- User Stories – Is a really good start for an exploratory session
- Personas – If Agnes 80 years old or Tim 14 years old would use this application. What would they do? Try?
- Other Heuristic Mnemonic http://www.qualityperspectives.ca/resources_mnemonics.html

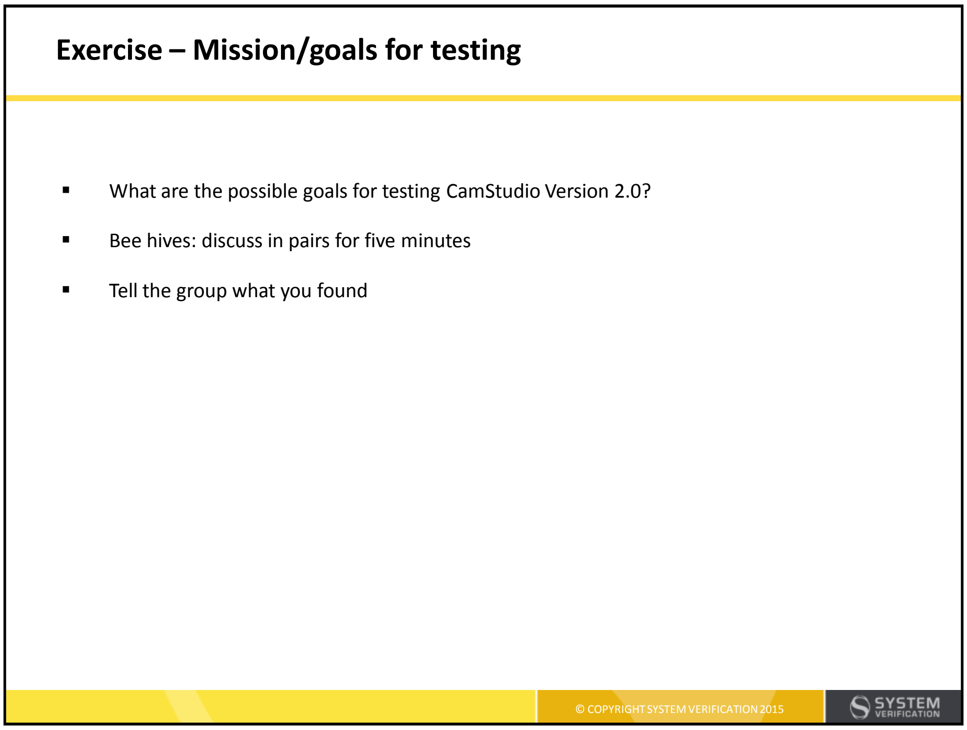
Exercise





Camstudio – Project plan

- Intro
  - Product is free and our goal is to supply a simple tool which supports the training group, and which others can use as an aid in creating presentations of new software. Example: training material for new releases
  - Developers have spent 300 hours developing new functionality for this release, but nothing is tested yet
- Resources
  - Orderer is Hans Order, the company's training manager
  - Developers: Joan Java and Victor Webb
- Time plan
  - Release date for this version is two weeks away
- Testing
  - We have decided to bring in real testers, who will have two weeks to test everything new
  - Of course, all previous functionality will work as it did before!



Exercise – Mission/goals for testing

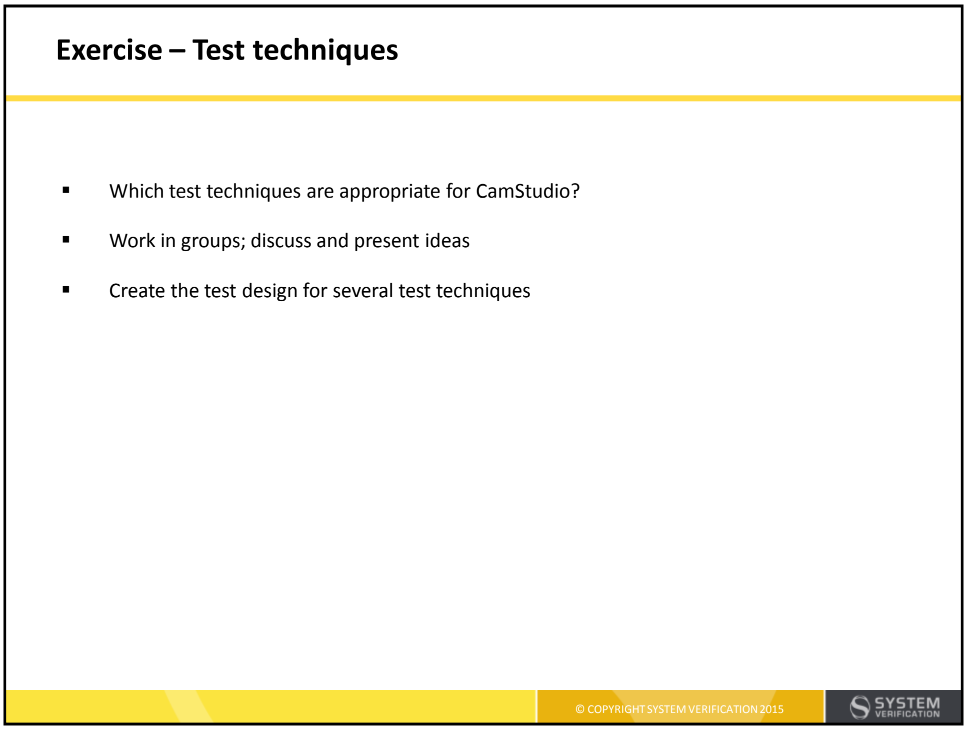
- What are the possible goals for testing CamStudio Version 2.0?
- Bee hives: discuss in pairs for five minutes
- Tell the group what you found

Exercise – Product components

- What is applicable for the product part of CamStudio?
  - Work with POSTUD
  - Where can you find the information you need?
  - Tell us what you found
- Create an overview diagram of CamStudio's parts and how they are related to each other
  - Where do other applications come in?

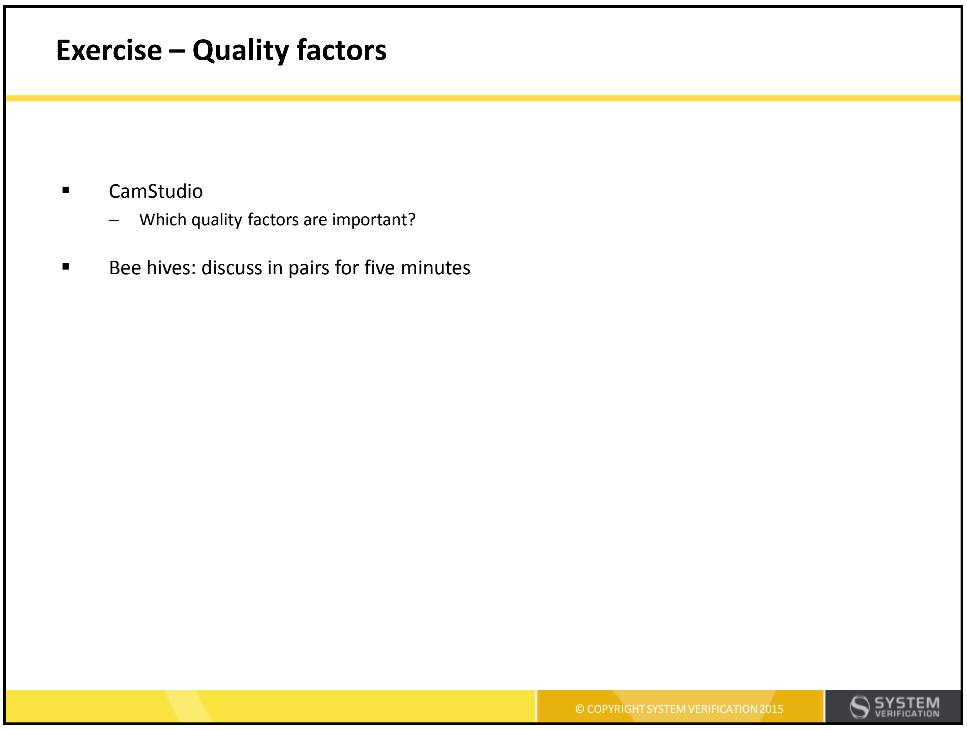
Exercise – Map a component

- Choose one of the components below and map it in more detail
  - Recorder
  - Producer
  - Player
  - Playplus
- Summarise results and describe for the group



Exercise – Test techniques

- Which test techniques are appropriate for CamStudio?
- Work in groups; discuss and present ideas
- Create the test design for several test techniques



Exercise – Quality factors

- CamStudio
  - Which quality factors are important?
- Bee hives: discuss in pairs for five minutes

CamStudio

- Work based on the information you have collected
  - Test strategy
  - Test cases
  - Schedule
- You need to use all parts of CamStudio
  - Not just Recorder
- Start CamStudio
  - Give it all you have got!
  - Write down everything you do, so you can answer my question about what you have tested!
- List the problems you find
  - So someone else can reproduce them