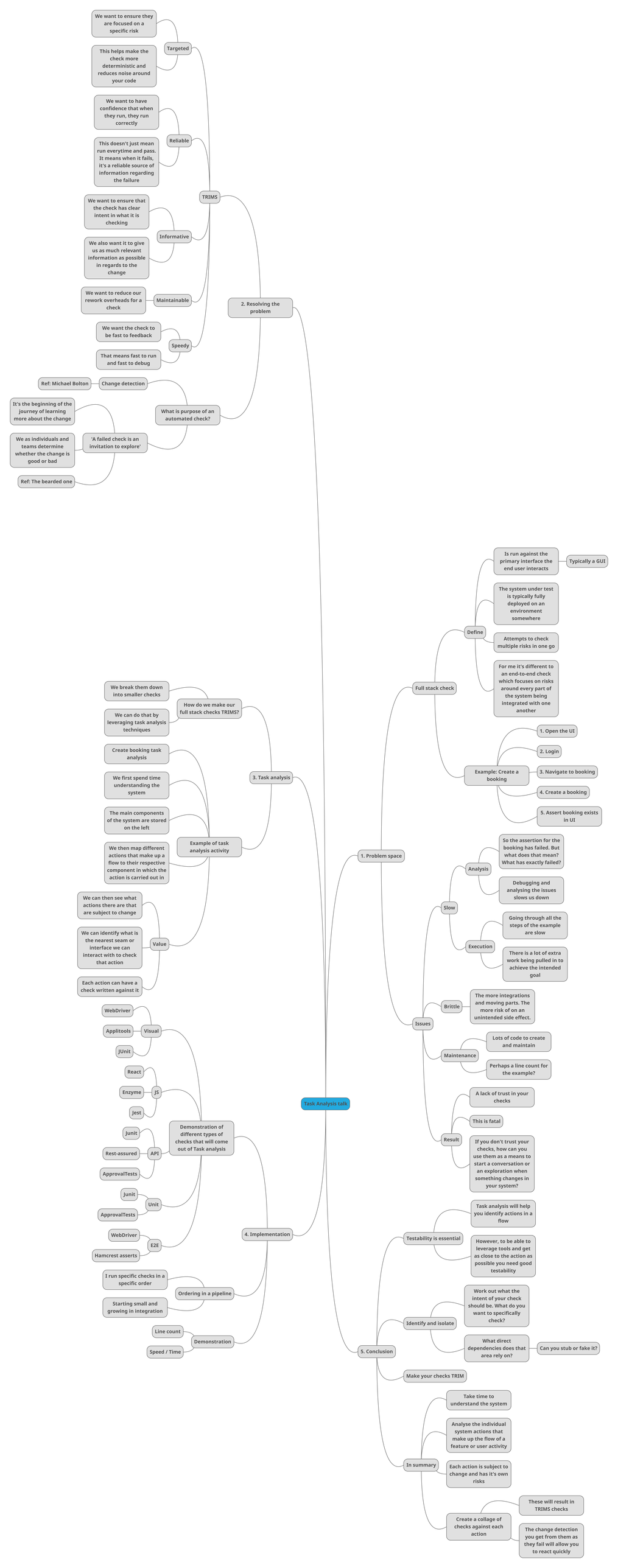
Task Analysis talk
1. Problem space
Full stack check
Define
Is run against the primary interface the end user interacts
Typically a GUI
The system under test is typically fully deployed on an environment somewhere
Attempts to check multiple risks in one go
For me it's different to an end-to-end check which focuses on risks around every part of the system being integrated with one another
Example: Create a booking
1. Open the UI
2. Login
3. Navigate to booking
4. Create a booking
5. Assert booking exists in UI
Issues
Slow
Analysis
So the assertion for the booking has failed. But what does that mean? What has exactly failed?
Debugging and analysing the issues slows us down
Execution
Going through all the steps of the example are slow
There is a lot of extra work being pulled in to achieve the intended goal
Brittle
The more integrations and moving parts. The more risk of on an unintended side effect.
Maintenance
Lots of code to create and maintain
Perhaps a line count for the example?
Result
A lack of trust in your checks
This is fatal
If you don't trust your checks, how can you use them as a means to start a conversation or an exploration when something changes in your system?
2. Resolving the problem
TRIMS
Targeted
We want to ensure they are focused on a specific risk
This helps make the check more deterministic and reduces noise around your code
Reliable
We want to have confidence that when they run, they run correctly
This doesn't just mean run everytime and pass. It means when it fails, it's a reliable source of information regarding the failure
Informative
We want to ensure that the check has clear intent in what it is checking
We also want it to give us as much relevant information as possible in regards to the change
Maintainable
We want to reduce our rework overheads for a check
Speedy
We want the check to be fast to feedback
That means fast to run and fast to debug
What is purpose of an automated check?
Change detection
Ref: Michael Bolton
'A failed check is an invitation to explore'
It's the beginning of the journey of learning more about the change
We as individuals and teams determine whether the change is good or bad
Ref: The bearded one
3. Task analysis
How do we make our full stack checks TRIMS?
We break them down into smaller checks
We can do that by leveraging task analysis techniques
Example of task analysis activity
Create booking task analysis
We first spend time understanding the system
The main components of the system are stored on the left
We then map different actions that make up a flow to their respective component in which the action is carried out in
Value
We can then see what actions there are that are subject to change
We can identify what is the nearest seam or interface we can interact with to check that action
Each action can have a check written against it
4. Implementation
Demonstration of different types of checks that will come out of Task analysis
Visual
WebDriver
Applitools
JUnit
JS
React
Enzyme
Jest
API
Junit
Rest-assured
ApprovalTests
Unit
Junit
ApprovalTests
E2E
WebDriver
Hamcrest asserts
Ordering in a pipeline
I run specific checks in a specific order
Starting small and growing in integration
Demonstration
Line count
Speed / Time
5. Conclusion
Testability is essential
Task analysis will help you identify actions in a flow
However, to be able to leverage tools and get as close to the action as possible you need good testability
Identify and isolate
Work out what the intent of your check should be. What do you want to specifically check?
What direct dependencies does that area rely on?
Can you stub or fake it?
Make your checks TRIM
In summary
Take time to understand the system
Analyse the individual system actions that make up the flow of a feature or user activity
Each action is subject to change and has it's own risks
Create a collage of checks against each action
These will result in TRIMS checks
The change detection you get from them as they fail will allow you to react quickly