Risk Analysis Heuristics *(for Digital Products)*

By James Bach and Michael Bolton

v. 2.0

This is a set of guideword heuristics for use in analyzing product risks for digital products, mainly software. "Guidewords" are words or phrases that help focus your attention on potentially important factors. Guidewords are not mutually exclusive—they interact and overlap to some degree. But that's okay. In heuristic risk analysis we do not use mathematics to calculate risk, however if many of these guidewords seems to apply to a particular component of your product, you will probably consider that part more likely to harbor serious bugs, and more worth testing.

Project Factors

Things going on in projects, among people, may lead to bugs.

Learning Curve: When developers are new to a tool, technology, or solution domain, they are likely to make mistakes. Many of those mistakes they will be unable to detect.

Poor Control: Code and other artifacts may not be under sufficient scrutiny or change control, allowing mistakes to be made and to persist. Also people may try to subvert weak controls when they perceive themselves to be under time pressure.

Rushed Work: The amount of work exceeds the time available to do it comfortably. Corners are likely to be cut; details are likely to be forgotten.

Fatigue: Programmers and other members of the development team are more likely to make mistakes when they're physically tired or even just bored.

Overfamiliarity: When people are immersed in a project or a community for an extended time, they may become blind to patterns of risks or problems that are easy for an outsider to see.

Distributed Team: When people are working remotely from each other, communication may become strained and difficult, simple collaborations become expensive, the conditions for the exchange of tacit knowledge are inhibited.

Third-party Contributions: Any part of a product contributed by a third-party vendor may contain hidden features and bugs, and the developers may otherwise not fully understand it.

Bad Tools: The project team may be saddled with tools that interfere with or constrain their work; or that may introduce bugs directly into their work.

Expense of Fixes: Some components or type of bugs may be especially expensive to fix, or take a long time to fix (platform bugs are typically like this). In that case, you may need to focus on finding those bugs especially soon.

Not Yet Tested: Any part of the product that hasn't yet been tested is obviously likely to have fresh bugs in it, compared to things that *have* been tested. Therefore, for instance, it may be better to focus on parts of the product that have not been unit tested.

Technology Factors

The structure and dynamics of technology itself may give rise to bugs.

New Technology: Over time, the risks associated with any new kind of technology will become apparent, so if your product uses the latest whizzy concept, it is more likely to have important and unknown bugs in it.

New Code: The newer the code you are testing, the more likely it is to have unknown problems.

Old Code: A product that has been around for a while may contain code that is unsuited to its current context, difficult to understand, or hard to modify.

Changed Code: Any recently changed code is more likely to have unknown problems.

Brittle Code: Some code may be written in a way that makes it difficult to change without introducing new problems. Even if this code never changes, it may be brittle in the sense that it tends to break when anything around it changes.

Complexity: The more different interacting elements a product has, the more ways it can fail; the more states or state transitions it has, the more states can be wrong.

Failure History: The more that a product or part of a product has failed in the past, the more you might expect it to fail in the future. Also, if a particular product has failed in a particularly embarrassing way, it perhaps should not be allowed to fail in that way again without bring the project team into disrepute.

Dependencies Upstream: One part of a system or one feature of a product may depend on data or conditions that are controlled by other components that come before it. The more upstream processing that must occur correctly, the more likely that any bugs in those processes may cause failure in the downstream component.

Dependencies Downstream: Any particular component that has many other components that rely on it will involve more risk, because the upstream bugs will propagate trouble downstream.

Distributed Components: A product may be comprised of things that spread out over a large area, connected by tenuous network links that introduce uncertainty, noise, or lag time into the system.

Open-Ended Input: The greater freedom there is in data, the more likely that a particular configuration of data could trigger a bug. Lack of filtering and bounding are especially a problem for security.

Hard to Test: When something is hard to test, perhaps because it is hard to observe or hard to control, there will be greater risk that bugs will go undetected, and it will require extra effort to find the important bugs.

Hardware: Hardware components can’t be changed easily. Hardware related problems must be found early because of the long lead time for fixing.

Specification Factors

Aspects of specifications may indicate or promote the presence of bugs.

Ambiguity: Words and diagrams are always interpreted by people, and different people will often have different interpretations of things. More ambiguity means more likelihood that a bug can be introduced through honest misunderstanding.

Very High Precision: Sometimes a document will specify a higher level of precision than is necessary or achievable. Sometimes the product should behave in a way that is more precise than the specification suggests. In any case, higher the precision required, the more likely it is that the product will not meet that requirement.

Mysterious Silence: Sometimes a specification will leave out things that a tester might think are essential or important. This "mysterious" silence might indicate that the designers are not thinking enough about those aspects of the design, and therefore there are perhaps more bugs in it. This is commonly seen with error handling.

Undecided Requirements: The designers might have intentionally left parts of the product unspecified because they don't yet know how it should work. Postponing the design of a system is a normal part of Agile development, for instance, but wherever that happens there is a possibility that a big problem will be hiding in those unknown details.

Evolving Requirements: Requirements are not static, they are changed and developed and extended. Any document is a representation of what some person believed at some time in the past; and when when a requirement is updated, it's possible that other requirements which SHOULD have changed, didn't. Fast evolving requirements often develop inconsistencies and contradictions that lead to bugs.

Imported Requirements: Sometimes requirement statements are "borrowed"— cut and pasted from other documents or even from other projects. These may include elements not appropriate to the current project.

Hard to Read: If the document is large, poorly formatted, repetitive, or otherwise hard to read, it is less likely to have been carefully written or properly reviewed.

Non-Native Writers: When the person writing the specification is not fluent in the specification's language, misunderstanding and error are likely.

Non-Native Readers: When the people reading and interpreting the specification are not fluent in the specification's language, misinterpretation is likely.

Operational Factors

The circumstances and patterns of use affect the probability and impact of bugs.

Critical Feature: The more important a feature is, the more important its bugs will be.

Popular Feature: The more people use a feature, the more likely any bugs in it will be found by users.

Strategic Feature: A feature might be key to differentiating your product from a competitor; or might have a special notoriety that would make its bugs especially important.

Disconnection: Different parts of a product that must work together may fall into incompatible states, leading to a failure of the system as a whole.

VIP Opinion: A particular important person might be paying attention to a particular feature or configuration or type of use, making bugs in that area more important. Or the important person's fascination with one aspect of the product may divert needed attention from other parts of the product.

Misusable: A feature might be easily misused, such that it might misbehave in a way that while not technically a flaw in the design, is still effectively a bug.

Glaring Failure: A problem or its consequences may be obvious to anyone who encounters it.

Insidious Failure: The causes or symptoms of a problem may be invisible or difficult to see for some time before they are noticed, allowing more trouble to build.