Name: Architecture Reverse Engineering

Elisabeth Hendrickson

Objective: Create a map of the architecture to guide a testing effort.

Problem: How do you know what the bug hunting territory looks like without a map?

Context: You are a tester. There is no existing picture of the architecture, or existing pictures are not useful for your purposes. You need a picture of the entire system to help guide your testing and suggest useful tests to perform.

Forces:

- Getting information about the architecture or expected results may be difficult.
- The goals of the architecture may not have been articulated or may be unknown.

Solution:

Start by investigating the program yourself:

1. Run the program and make lists of the functions and objects you see. For example, in running a Word Processor, you might notice functions such as editing text, inserting pictures, and formatting. You might notice objects such as pictures, text boxes, bullets, table of contents.
2. Use the User Documentation, release notes, specifications, requirements documents, or any other available artifacts to learn more about the intention and implementation of the system.
3. Use the Task Manager or another utility to see what processes are running.
4. Use Dependency Walker or another utility to see what libraries are loaded.
5. Use a program like InCtrl to determine what changes on disk and in the registry (for Windows systems) while the program is operating.

Produce a usable, if inaccurate, picture of the system as you see it based on your investigations. Now you're ready to work with other people on the team to correct your picture and fill in the details.

Approach other team members with your picture:

1. Determine which part(s) of the system the person works on or knows about.
2. Ask the person about your picture—if it's accurate, what's missing.
3. Make the corrections the person indicates before showing your picture to the next person.

You can have this conversation with two other people at once. However, when more than a few people try to discuss the picture at one time, you don't get the quality of information you really need. Keep the discussions to three or four people total.

If two people tell you to make conflicting corrections ("The authentication code lives on the client." "It lives on the server."), bring the two people together and allow them to hash out their differences. Don't play middleman in the discussion.

Rationale:
Many organizations do not have a big picture of the system—or if they have one it isn't useful for testing. Testers feel this lack keenly. Fortunately we can build the picture we need ourselves and use our initial guesses to ask intelligent questions of other team members.

Resulting Context:
You now have a picture of the system that you can use to guide your testing, to perform an *Architecture Achilles Heels Analysis*, or to plan a test effort. The picture may not be entirely accurate. As long as it is not sufficiently misleading as to cause worse testing rather than better, inaccuracies are acceptable. This is a living document. You can always update it as you learn more.

Additional Benefits:
- Increases communication between different groups within the project.
- Finds potential architectural flaws earlier.
- Aligns architecture with its goals.
- Provides additional fodder for test planning for current and future releases.

Liabilities:
- You may need to spend a large amount of time understanding the system to map it.
- Other members of the team or organization may need to invest more time answering questions to contribute to the test effort than anticipated.
- A grossly inaccurate picture—one that implies that there are no dependencies where there really are dependencies, for example—could result in testers ignoring important tests or configurations. It's best to use the diagram to suggest tests rather than to suggest areas that are safe to ignore.

Note that the last two liabilities may actually be good for the project or software in the long run.

- Some members of the team may feel threatened during this process.
- Information uncovered during this process may throw the project into chaos, at least temporarily.

Related Patterns:
- Architecture Achilles Heels Analysis
- Explicit Architectural Goal Articulation