# Programmer Competency Matrix

> Note that the knowledge for each level is cumulative; being at level *n* implies that you also know everything from the levels lower than *n*.

### Computer Science

| | 2n (Level 0) | n2 (Level 1) | n (Level 2) | log(n) (Level 3) | Comments |
|---|---|---|---|---|---|
| data structures | Doesn't know the difference between Array and LinkedList | Able to explain and use Arrays, LinkedLists, Dictionaries etc in practical programming tasks | Knows space and time tradeoffs of the basic data structures, Arrays vs LinkedLists, Able to explain how hashtables can be implemented and can handle collisions, Priority queues and ways to implement them etc. | Knowledge of advanced data structures like B-trees, binomial and fibonacci heaps, AVL/Red Black trees, Splay Trees, Skip Lists, tries etc. | |
| algorithms | Unable to find the average of numbers in an array (It's hard to believe but I've interviewed such candidates) | Basic sorting, searching and data structure traversal and retrieval algorithms | Tree, Graph, simple greedy and divide and conquer algorithms, is able to understand the relevance of the levels of this matrix. | Able to recognize and code dynamic programming solutions, good knowledge of graph algorithms, good knowledge of numerical computation algorithms, able to identify NP problems etc. | |
| systems programming | Doesn't know what a compiler, linker or interpreter is | Basic understanding of compilers, linker and interpreters. Understands what assembly code is and how things work at the hardware level. Some knowledge of virtual memory and paging. | Understands kernel mode vs. user mode, multi-threading, synchronization primitives and how they're implemented, able to read assembly code. Understands how networks work, understanding of network protocols and socket level programming. | Understands the entire programming stack, hardware (CPU + Memory + Cache + Interrupts + microcode), binary code, assembly, static and dynamic linking, compilation, interpretation, JIT compilation, garbage collection, heap, stack, memory addressing… | |

### Software Engineering

| | 2n (Level 0) | n2 (Level 1) | n (Level 2) | log(n) (Level 3) | Comments |
|---|---|---|---|---|---|
| source code version control | Folder backups by date | VSS and beginning CVS/SVN user | Proficient in using CVS and SVN features. Knows how to branch and merge, use patches setup repository properties etc. | Knowledge of distributed VCS systems. Has tried out Bzr/Mercurial/Darcs/Git | |
| build automation | Only knows how to build from IDE | Knows how to build the system from the command line | Can setup a script to build the basic system | Can setup a script to build the system and also documentation, installers, generate release notes and tag the code in source control | |
| automated testing | Thinks that all testing is the job of the tester | Has written automated unit tests and comes up with good unit test cases for the code that is being written | Has written code in TDD manner | Understands and is able to setup automated functional, load/performance and UI tests | |

### Programming

| | 2n (Level 0) | n2 (Level 1) | n (Level 2) | log(n) (Level 3) | Comments |
|---|---|---|---|---|---|
| problem decomposition | Only straight line code with copy paste for reuse | Able to break up problem into multiple functions | Able to come up with reusable functions/objects that solve the overall problem | Use of appropriate data structures and algorithms and comes up with generic/object-oriented code that encapsulate aspects of the problem that are subject to change. | |

## Computer Science

| | | | | | |
|---|---|---|---|---|---|
| systems decomposition | Not able to think above the level of a single file/class | Able to break up problem space and design solution as long as it is within the same platform/technology | Able to design systems that span multiple technologies/platforms. | Able to visualize and design complex systems with multiple product lines and integrations with external systems. Also should be able to design operations support systems like monitoring, reporting, fail overs etc. | |
| communication | Cannot express thoughts/ideas to peers. Poor spelling and grammar. | Peers can understand what is being said. Good spelling and grammar. | Is able to effectively communicate with peers | Able to understand and communicate thoughts/design/ideas/specs in a unambiguous manner and adjusts communication as per the context | This is an often under rated but very critical criteria for judging a programmer. With the increase in outsourcing of programming tasks to places where English is not the native tongue this issue has become more prominent. I know of several projects that failed because the programmers could not understand what the intent of the communication was. |
| code organization within a file | no evidence of organization within a file | Methods are grouped logically or by accessibility | Code is grouped into regions and well commented with references to other source files | File has license header, summary, well commented, consistent white space usage. The file should look beautiful. | |
| code organization across files | No thought given to organizing code across files | Related files are grouped into a folder | Each physical file has a unique purpose, for e.g. one class definition, one feature implementation etc. | Code organization at a physical level closely matches design and looking at file names and folder distribution provides insights into design | |
| source tree organization | Everything in one folder | Basic separation of code into logical folders. | No circular dependencies, binaries, libs, docs, builds, third-party code all organized into appropriate folders | Physical layout of source tree matches logical hierarchy and organization. The directory names and organization provide insights into the design of the system. | The difference between this and the previous item is in the scale of organization, source tree organization relates to the entire set of artifacts that define the system. |
| code readability | Mono-syllable names | Good names for files, variables classes, methods etc. | No long functions, comments explaining unusual code, bug fixes, code assumptions | Code assumptions are verified using asserts, code flows naturally – no deep nesting of conditionals or methods | |
| defensive coding | Doesn't understand the concept | Checks all arguments and asserts critical assumptions in code | Makes sure to check return values and check for exceptions around code that can fail. | Has his own library to help with defensive coding, writes unit tests that simulate faults | |

## Computer Science

| | | | | | |
|---|---|---|---|---|---|
| error handling | Only codes the happy case | Basic error handling around code that can throw exceptions/generate errors | Ensures that error/exceptions leave program in good state, resources, connections and memory is all cleaned up properly | Codes to detect possible exception before, maintain consistent exception handling strategy in all layers of code, come up with guidelines on exception handling for entire system. | |
| IDE | Mostly uses IDE for text editing | Knows their way around the interface, able to effectively use the IDE using menus. | Knows keyboard shortcuts for most used operations. | Has written custom macros | |
| API | Needs to look up the documentation frequently | Has the most frequently used APIs in memory | Vast and In-depth knowledge of the API | Has written libraries that sit on top of the API to simplify frequently used tasks and to fill in gaps in the API | E.g. of API can be Java library, .net framework or the custom API for the application |
| frameworks | Has not used any framework outside of the core platform | Has heard about but not used the popular frameworks available for the platform. | Has used more than one framework in a professional capacity and is well-versed with the idioms of the frameworks. | Author of framework | |
| requirements | Takes the given requirements and codes to spec | Come up with questions regarding missed cases in the spec | Understand complete picture and come up with entire areas that need to be speced | Able to suggest better alternatives and flows to given requirements based on experience | |
| scripting | No knowledge of scripting tools | Batch files/shell scripts | Perl/Python/Ruby/VBScript/Powershell | Has written and published reusable code | |
| database | Thinks that Excel is a database | Knows basic database concepts, normalization, ACID, transactions and can write simple selects | Able to design good and normalized database schemas keeping in mind the queries that'll have to be run, proficient in use of views, stored procedures, triggers and user defined types. Knows difference between clustered and non-clustered indexes. Proficient in use of ORM tools. | Can do basic database administration, performance optimization, index optimization, write advanced select queries, able to replace cursor usage with relational sql, understands how data is stored internally, understands how indexes are stored internally, understands how databases can be mirrored, replicated etc. Understands how the two phase commit works. | |

## Experience

| | 2n (Level 0) | n2 (Level 1) | n (Level 2) | log(n) (Level 3) | Comments |
|---|---|---|---|---|---|
| languages with professional experience | Imperative or Object Oriented | Imperative, Object-Oriented and declarative (SQL), added bonus if they understand static vs dynamic typing, weak vs strong typing and static inferred types | Functional, added bonus if they understand lazy evaluation, currying, continuations | Concurrent (Erlang, Oz) and Logic (Prolog) | |
| platforms with professional experience | 1 | 2-3 | 4-5 | 6+ | |
| years of professional experience | 1 | 2-5 | 6-9 | 10+ | |
| domain knowledge | No knowledge of the domain | Has worked on at least one product in the domain. | Has worked on multiple products in the same domain. | Domain expert. Has designed and implemented several products/solutions in the domain. Well versed with standard terms, protocols used in the domain. | |

## Knowledge

## Computer Science

| | | | | |
|---|---|---|---|---|
| tool knowledge | Limited to primary IDE (VS.Net, Eclipse etc.) | Knows about some alternatives to popular and standard tools. | Good knowledge of editors, debuggers, IDEs, open source alternatives etc. etc. For e.g. someone who knows most of the tools from Scott Hanselman's power tools list. Has used ORM tools. | Has actually written tools and scripts, added bonus if they've been published. |
| languages exposed to | Imperative or Object Oriented | Imperative, Object-Oriented and declarative (SQL), added bonus if they understand static vs dynamic typing, weak vs strong typing and static inferred types | Functional, added bonus if they understand lazy evaluation, currying, continuations | Concurrent (Erlang, Oz) and Logic (Prolog) |
| codebase knowledge | Has never looked at the codebase | Basic knowledge of the code layout and how to build the system | Good working knowledge of code base, has implemented several bug fixes and maybe some small features. | Has implemented multiple big features in the codebase and can easily visualize the changes required for most features or bug fixes. |
| knowledge of upcoming technologies | Has not heard of the upcoming technologies | Has heard of upcoming technologies in the field | Has downloaded the alpha preview/CTP/beta and read some articles/manuals | Has played with the previews and has actually built something with it and as a bonus shared that with everyone else |
| platform internals | Zero knowledge of platform internals | Has basic knowledge of how the platform works internally | Deep knowledge of platform internals and can visualize how the platform takes the program and converts it into executable code. | Has written tools to enhance or provide information on platform internals. For e.g. disassemblers, decompilers, debuggers etc. |
| books | Unleashed series, 21 days series, 24 hour series, dummies series… | Code Complete, Don't Make me Think, Mastering Regular Expressions | Design Patterns, Peopleware, Programming Pearls, Algorithm Design Manual, Pragmatic Programmer, Mythical Man month | Structure and Interpretation of Computer Programs, Concepts Techniques, Models of Computer Programming, Art of Computer Programming, Database systems , by C. J Date, Thinking Forth, Little Schemer |
| blogs | Has heard of them but never got the time. | Reads tech/programming/software engineering blogs and listens to podcasts regularly. | Maintains a link blog with some collection of useful articles and tools that he/she has collected | Maintains a blog in which personal insights and thoughts on programming are shared |