|  |
| --- |
| **개발자 역량 측정 매트릭스**참고: http://www.indiangeek.net/programmer-competency-matrix/다음은 개발자의 개발 역량을 종합적으로 평가할 수 있는 역량 측정용 매트릭스 평가표입니다. 각 항목별로 1 ~ 4점까지 부여하면, 140점 (보너스 포함 142점 만점)이 됩니다. 각자 자기의 개발 역량을 점수로 매겨 보시기 바랍니다. (이 자료는 프리랜서와 같이 작업할 때, 프리랜서의 역량 수준을 빠르게 측정하는 데에도 도움이 됩니다.)**전산학 이해** |
|  | 2n (Level 0) | n2 (Level 1) | n (Level 2) | log(n) (Level 3) |
| **1 자료구조** | 어레이와 링크드 리스트의 차이를 모른다. | 실제 프로그래밍 환경에서 어레이와 링크드 리스트와 딕셔너리의 차이를 설명할 수 있다. | 어레이와 링크드 리스트의 시간/공간 타협관계를 설명할 수 있다. 해쉬의 콜리전 처리를 할 수 있고, 우선순위 큐를 만들 수 있다. | B트리, 이진트리, 피보나치힙, AVL트리, Red/Black트리, Splay 트리, Skip 리스트, 트라이 등 고급 데이터 스트럭처에 대해 이해하고 있다. |
| **2 알고리듬** | 어레이에 들어 있는 숫자들의 평균을 낼 줄 모른다. | 소팅, 서칭, 트래버싱 알고리듬을 이해한다. | 트리, 그래프, 단순한 그리디 알고리듬. Divide and Conquer알고리듬을 이해한다. 이 표의 레벨 구분자가 왜 그렇게 쓰였는지 설명할 수 있다. | 다이내믹 프로그래밍 솔루션을 이해한다. 그래프 알고리듬, 수치연산 알고리듬을 이해하고, NP문제를 식별할 수 있다. |
| **3 시스템 프로그래밍** | 컴파일러, 링커, 인터프리터를 구분하지 못한다. | 컴파일러, 링커, 인터프리터를 이해한다. 하드웨어 레벨의 어셈블리 언어 동작을 이해한다. 가상 메모리와 페이징에 대한 이해가 있다. | 커널모드/유저모드 차이를 알고, 멀티스레딩, 동기화를 이해하고 그것들이 어떻게 구현되었는지 안다. 어셈블리 코드를 읽을 수 있다. 네트웍이 어떻게 동작하는지 안다. 프로토콜을 알고, 소켓 수준의 프로그램을 읽을 수 있다. | 전체 프로그래밍 스택을 이해한다.하드웨어 (CPU + 메모리 + 캐쉬 + 인터럽트 + 마이크로코드), 바이너리 코드, 어셈블리, 정적/동적 링킹, 컴파일, 인터프리테이션, JIT 컴파일, 가비지 콜렉션, 힙, 스택, 메모리 어드레싱 등을 구분해서 이해한다. |
| **소프트웨어 엔지니어링** |
|  | 2n (Level 0) | n2 (Level 1) | n (Level 2) | log(n) (Level 3) |
| **4 소스코드****버전컨트버** | 날짜 단위 폴더 백업 | CVS/SVN/VSS등의 사용을 시작했다. | CVS/SVN에 대한 능숙한 사용. 브랜치/머지를 할 수 있고, 팻치를 만들 수 있고, 리파지토리 속성에 맞게 새로 만들 수 있다. | 분산 VCS 시스템을 이해한다. Bzr/Mercurial/Darcs/Git 등을 쓰려고 한다. |
| **5 빌드 자동화** | IDE 에서만 빌드할 수 있다. | 코맨드 라인으로 빌드를 만들 수 있다. | 빌드 스크립트를 직접 짠다. | 문서, 설치 인스톨러, 릴리즈 노트를 포함한 빌드 스크립트를 만든다. |
| **6 테스트 자동화** | 테스트는 테스터의 일이라고 생각한다. | 유닛 테스트를 짜고 새로 짜는 코드에 유닛 테스트를 작성하고 있다. | TDD 방식으로 코드를 짠다. | 기능적, 로드/성능적, GUI 측면의 테스트 자동화를 이해하고 실현한다. |
| **프로그래밍** |
|  | 2n (Level 0) | n2 (Level 1) | n (Level 2) | log(n) (Level 3) |
| **7 프로그램분해** | 길게 라인단위로 복사/붙여넣기로 코드를 짠다. | 문제를 여러 함수로 나눠서 코딩한다. | 재사용 가능한 객체로 코드를 작성한다. | 적절한 알고리듬으로 제네릭/OOP방법을 써서 변경이 있을 법한 부분은 적절히 캡슐화 하면서 코딩한다. |
| **8 시스템 분해** | 파일 1개, 클래스 1개 이상의 범위를 생각하지 못한다. | 같은 플랫폼, 같은 기술 범위 내에서는 문제를 쪼개서 해결책을 설계해 낼 수 있다. | 복수개의 기술과 시스템에 걸쳐 있는 문제에 대한 솔루션을 만들어 낸다. | 복잡한 다수개의 제품들을 가시화하여 설계하고 외부 시스템과 연동을 이끌어 낸다. 모니터링, 리포팅, 장애복구 등의 운영 작업도 설계할 수 있다. |
| **9 의사소통** | 아이디어와 생각을 잘 표현하지 못하고, 스펠링과 문법이 엉망이다. | 동료가 뭔 말하는지 알아 먹도록 말한다. 스펠링과 문법은 좋다. | 효과적으로 의사 소통 할 수 있다. | 모호한 상황에서 생각/설계/아이디어/스펙을 이해하고 소통할 수 있으며, 상황에 맞게 소통할 수 있다. |
| **10 파일 내의 코드 구성** | 코드 파일 내에서 구조화가 안 되어 있다. | 메소드들이 논리적으로든 접근성으로든 어떻게든 구조화 되어 있다. | 코드가 영역별로 그루핑 되어 있고, 코멘트도 잘 되어 있고, 서로 다른 파일간의 참조도 잘 설명되어 있다. | 파일은 라이선스 헤더도 있고, 요약설명도 있고, 코멘트도 잘 되어 있고, 공백 사용은 일관성이 있고, 파일 자체가 보기 좋게 정렬되어 있다. |
|  | 2n (Level 0) | n2 (Level 1) | n (Level 2) | log(n) (Level 3) |
| **11 파일 간의 코드 구성** | 파일간의 구성에 대한 어떤 구조도 없다. | 파일들이 폴더로 나뉘어져 있다. | 파일 별로 고유한 목적이 있다. 예를 들어 클래스 하나 정의, 기능 하나 구현 등. | 코드 구성이 설계와 잘 매치되어 코드 파일명만 보더라도, 설계에 대한 이해가 가능하도록 만든다. |
| **12 소스 트리 구성** | 모든 게 폴더 하나에 다 있다.  | 논리적인 폴더로 나눠져 있다. | 서큘러 의존성이 없고, 바이너리, 라이브러리, 문서, 빌드, 써드파티 코드 등이 폴더로 구분되어 나눠져 있다. | 물리적 코드 구성이 논리적인 계층을 잘 반영한다. 디렉토리명으로 시스템 설계에 대한 이해가 가능하도록 만든다. |
| **13 코드 가독성** | 단음절 변수명. 코드 가독성이 낮게 짠다. | 파일, 변수, 클래스, 메소드 등에 대한 이름을 잘 부여한다. | 긴 함수는 없다. 통상적이지 않은 코드나 버그수정이나 전제조건 등에 대해서 코멘트를 달아 둔다. | 전제조건 등은 assert로 검증한다. 네스팅 단계가 깊지 않고, 자연스럽게 코드가 흐른다. |
| **14 방어적 코딩** | 방어적 코딩이 뭔지 모른다. | 인수를 다 체크하고, 크리티컬한 전제조건에 대해서는 assert를 사용한다. | 리턴값도 틀림없이 체크하고, 익셉션을 항상 체크한다. | 방어적 코딩을 하기 위한 자신만의 라이브러리가 있다. 실패 케이스를 시험하는 유닛 테스트 코드를 작성한다. |
|  | 2n (Level 0) | n2 (Level 1) | n (Level 2) | log(n) (Level 3) |
| **15 에러 핸들링** | 정상적인 코드만 작성한다. | 익셉션/에러가 생성되는 주변에 기본적인 에러 핸들링 코드가 있다. | 에러/익셉션으로 가도 프로그램이 안정적인 상태에 있도록 유지한다. 리소스, 커넥션, 메모리 등이 깨끗하게 해제됨을 보장한다. | 가능한 익셉션 상황을 미리 감지해 내도록 코딩한다. 코드 전체에 대해서 일정한 익셉션 핸들링 정책을 사용한다. 전체 시스템에 대한 익셉션 핸들링 가이드라인을 만든다. |
| **16 IDE** | 대부분 텍스트 에디팅에 IDE를 사용한다. | 인터페이스 이면에 숨어있는 IDE기능을 메뉴를 이용해서 효과적으로 불러내서 쓴다. | 거의 모든 IDE 사용을 단축키로 한다. | IDE에 매크로를 정의해서 사용한다. |
| **17 API** | 자주 문서를 봐야 한다. | 기억 속에 자주 쓰는 API는 들어 있다. | API에 대해서 폭넓고 깊이 있는 이해를 하고 있다. | 자주 호출하는 것에 대해서 단순화 시키기 위해 API위에 라이브러리를 추가로 개발하고, 결과적으로 API의 부족한 점은 직접 채운다. |
| **18 프레임웍** | 코어 플랫폼 밖의 어떤 프레임웍도 쓰지 않는다. | 유명한 프레임웍을 들어는 봤으나, 써보지는 못했다. | 프레임웍을 여러 개 능숙하게 쓰고, 해당 프레임웍을 효과적으로 잘 쓰는 법 전형적인 방법을 알고 있다. | 프레임웍을 직접 개발한다. |
|  | 2n  (Level 0) | n2 (Level 1) | n (Level 2) | log(n) (Level 3) |
| **19 요구사항** | 주어진 요구사항을 코드 스펙으로 바꾼다. | 스펙에서 비어있는 케이스에 대한 질문을 해 낸다. | 전체 그림을 이해하고, 스펙으로 정의할 영역 전체를 도출해 낸다. | 경험에 기반해서 주어진 요구사항에 대해 더 좋은 대안과 플로우를 제시할 수 있다. |
| **20 스크립팅** | 스크립팅 툴을 모른다. | 뱃치파일, 쉘스크립팅을 한다 | Perl/Python/Ruby/VBScript/Powershell 류의 스크립팅을 한다. | 재사용 가능한 코드를 짜고 공개한다. |
| **21 데이터베이스** | 엑셀이 데이터 베이스라고 생각한다. | 데이터 베이스 기본 개념과 정규화, ACID, 트랜잭션을 이해하고, 간단한 SELECT를 할 수 있다. | 실행될 질의문을 염두해 두고, 스키마를 잘 정규화 해서 정의할 수 있으며, 뷰와 스토어드 프로세져, 트리거, 사용자 정의 타입 등을 능숙하게 쓸 수 있다. 클러스터드 인덱스를 이해하며, ORM 툴을 사용한다. | 기본 DB관리, 성능 최적화, 색인 최적화, 고급 SELECT Query를 쓸 수 있고, 커서를 쓸 수 있고, 데이터가 내부적으로 어떻게 저장되는지 이해하며, 색인이 어떻게 저장되는지 알고, 데이터 베이스 미러링, 복제를 이해하고, 2-페이스 커밋을 이해한다. |
| **경험** |
|  | 2n (Level 0) | n2 (Level 1) | n (Level 2) | log(n) (Level 3) |
| **22 전문경험을 가진 언어** | 절차형 언어, OO언어를 안다. | 절차형 언어, OO언어, 선언적(SQL)언어를 알고, 동적/정적 타입과 약한/강한 타이핑을 이해하며, 정적으로 유도된 타입을 이해하면 보너스 추가. | 함수언어를 알고, Lazy Evaluation, Currying, Continuation을 이해하면 보너스 추가. | 병렬언어(Erlang,Oz)과 논리적 언어 (Prolog) 를 이해한다. |
| **23 전문 경험을 가진 플랫폼 수** | 1 | 2-3 | 4-5 | 6+ |
| **24** **전문 경험 기간 (년)** | 1 | 2-5 | 6-9 | 10+ |
| **25** **도메인 지식** | 도메인 지식이 없다 | 해당 도메인의 제품 1개에서 일해 봤다. | 해당 도메인의 여러 제품에서 일해 봤다. | 도메인 전문가.해당 도메인의 여러 제품을 설계하고 구현해 봤다. 표준 도메인 용어 및 프로토콜에 대해 잘 구사할 수 있다. |
| **지식** |
|  | 2n (Level 0) | n2 (Level 1) | n (Level 2) | log(n) (Level 3) |
| **26 도구 지식** | 주사용 IDE에 지식이 국한되어 있다. | 유명하거나 표준적인 다른 도구들을 쓸 줄 안다. | 에디터, 디버거, IDE에 대해 잘 알고 있고, 오픈소스 대체물도 잘 알고 있다. Scott Hanselman의 파워툴은 대부분 알고 있다. ORM 툴을 사용한다. | 도구를 직접 만든다. 그걸 공개했으면 보너스 점수. |
| **27 노출된 언어** | 절차형 언어, OOP | 절차형 언어, OOP, 선언적언어 SQL. 동적/정적 타이이핑, 약한/강한 타이핑, 정적 유도 타이핑을 알면 보너스 추가. | 함수언어를 알고, Lazy Evaluation, Currying, Continuation을 이해하면 보너스 추가. | 병렬언어(Erlang,Oz)과 논리적 언어 (Prolog) 를 이해한다. |
| **28 코드베이스 지식** | 코드베이스를 본 적이 없다. | 코드 레이아웃에 대한 이해가 있고, 시스템 빌드에 대해서 이해한다. | 코드 베이스를 잘 알고, 버그픽스를 코딩했고, 몇몇 기능도 추가했다. | 코드베이스에 여러 개의 주요 기능을 넣었다. 대부분의 기능과 버그 수정에 소요되는 변경내역을 가시화 시킬 수 있다. |
| **29** **최신 기술 이해** | 최신 기술을 모른다 | 해당 분야의 최신 기술을 들어 봤다. | 알파 프리뷰, CTP, 베타를 다운로드 해 봤다. 온라인 매뉴얼 등을 읽어 봤다. | 프리뷰를 시험해 보고, 뭔가 만들어 봤다. 그걸 공개했으면 보너스 추가. |
|  | 2n (Level 0) | n2 (Level 1) | n (Level 2) | log(n) (Level 3) |
| **30** **플랫폼 내부** | 플랫폼 내부에 대해서 전혀 모른다. | 플랫폼 내부가 어떻게 동작하는지 기본 이해를 가지고 있다. | 플랫폼 내부가 어떻게 동작하는지 잘 알고 있고, 가시화 시켜서 플랫폼이 어떻게 코드를 실행시키는 지 설명할 수 있다. | 플랫폼 내부에 대한 정보를 제공하기 위해서 디컴파일, 디버깅, 디스어셈블 등을 하거나 툴을 만든다. |
| **31** **책** | Unleashed 시리즈, 21일 시리즈, 24시간 시리즈, 더미 시리즈 책을 읽는다. | Code Complete, Don’t Make me Think, Matering Regular Expressions과 같은 책을 읽는다. | 디자인 패턴, 피플웨어, Programming Pearls, 알고리듬 디자인 매뉴얼, 실용적 프로그래머, Mythical Man Month같은 책을 읽는다. | 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 등을 읽는다. |
| **32****블로그** | 들어 봤지만, 별로 친하지 않다. | 주기적으로 programming, software engineering 블로그를 읽고, Podcast를 듣는다. | 펌글 블로그를 운영하고, 유용한 팁과 기사를 모아서 관리하고 있다. | 프로그램에 대한 개인적인 통찰이나 생각을 적는 블로그를 개설해서 다른사람과 공유한다. |  |

|  |
| --- |
| **외국어** |
|  | 2n (Level 0) | n2 (Level 1) | n (Level 2) | log(n) (Level 3) |
| **33 외국어** | 한국어 아닌 자료로는 기술 습득을 못한다. | 영어로 된 기술 교재를 읽고 기술을 파악한다. | 유럽인 / 미국인 으로부터 개발 업무 지시를 받고 그에 따라 업무수행 후에 결과를 보고할 수 있다.  | 영어로 유럽인 / 미국인 에게서 S/W발주 받아서, 인도인 중국인에게 S/W 개발 업무를 아웃소싱 시키고 국내 팀과 연계하여 국제적인 프로젝트를 진행시킬 수 있다. |

|  |
| --- |
| **문서화** |
|  | 2n (Level 0) | n2 (Level 1) | n (Level 2) | log(n) (Level 3) |
| **34 문서 포맷** | 문서 작성이 서툴고 뭘 작성해야 하는지 자발적으로 판단하지 못한다. | 프로젝트에 필요한 문서 포맷이 어떤 것인지 알고 있고 적어도 1개 이상의 프로젝트에서 단계별 문서를 작성해 본 적이 있다. | 여러 S/W 프로젝트에서 S/W 사용된 다양한 문서 포맷을 여러 세트 작성해 본 적이 있고, 여러 세트를 확보하고 있다. | 필요하면 문서 세트를 종류 및 포맷을 직접 정의할 수 있고, 베스트 프랙티스를 알고 있다. |
| 35오피스툴사용 | 오피스 문서도구 사용법을 완전히 숙지하지 않은 상태이다. | 오피스를 이용해서 각종 문서를 작성하지만, 문서 포맷이 주어지지 않으면 작성에 애를 먹는다. | 오피스 사용법을 완전히 숙지하고 있으며, 단축키도 사용하고 효율적인 파일 처리를 할 수 있다. | 상당한 작업을 단축키로 수행하며, 매크로도 직접 정의해서 사용하고, 오피스 툴 외의 공개판 도구들도 적극적으로 사용해서 효과적인 문서를 생산한다. |