# 개발자의 격, SW 장인 마인드

- 이 세미나는 `소프트웨어 장인`, `코드 작성 가이드`, `프로그래밍의 규칙`, `소프트웨어 장인 정신 이야기`, `개발자로 살아남기` 등 다양한 베스트셀러를 위니브의 색에 맞춰 각색한 것입니다.
- 이 세미나는 오늘만을 위한 강의로 외부 공유는 하지 않습니다.

먼저, 너무나 반갑습니다. 위니브 이호준입니다. 돌아가면서 잠깐 자기소개를 부탁드립니다.

대 AI 시대, 우리는 어떤 마음을 가지고, 어떤 방향성으로 공부해야 할까요?

ChtGPT, Copilot, Claude, Cursor.
이전과 같은 마음과 방법으로 나아가도 되는 것일까요?

이미 수 많은 변화를 겪어온 다른 책에서 지혜를 얻어보면 좋을 것 같습니다.

*소프트웨어 개발자가 소프트웨어 개발 업무만 하면 되던 시절은 지나갔다. - 소프트웨어 장인*

이전에도 소프트웨어 개발자의 롤은 점차 확장되어 왔습니다. 그러나 이제 AI의 도움으로 개발자의 범위는 점차 확장되고 있습니다. 8살짜리 꼬마 아이가 Cursor로 프로그램을 만드는 영상을 보셨나요? 개발자가 아닌 사람이 코드를 짤 수 있다면 개발자는 어떤 역할을 해야 할까요?

*절차와 도구보다는 개성과 화합을*
*방대한 문서 작업보다는 동작하는 소프트웨어를*
*계약 조건에 대한 협상보다는 고객과의 협력을*
*계획을 따르는 것을 넘어서서 변화에 대처하는 것을*
*더 가치있게 여긴다.*
*좌측도 가치 있음을 인정하지만 우측 사항에 더 높은 가치를 둔다는 것이다.*
*- 애자일 매니페스토*

장기적으로 SW 직업군이 줄어들거나 사라질 것이라 보고 있진 않아요. 우리가 생각하는 것보다 사람과 사람이 일하는 것은 아직 사람과 사람이 중요하니까요. 다만 동작하는 소프트웨어의 품질이 높아지고, 기대하는 소프트웨어의 품질도 높으며, 문서 작업 같은 것도 꼼꼼하게 작성이 되어 있는 품질 좋은 소프트웨어를 만들 수 있는 시대가 되었습니다. 도구는 이제 더 이상 도구일 뿐이라는 생각은 없어져야 합니다. 누군가에게는 동료이고, 멘토입니다.

*소스 코드는 예측가능하고 유지보수될 수 있는 상태여야 한다. - 소프트웨어 장인*

그 사람이 와야만 코드를 이해를 하고 수정할 수 있는 대체 불가한 코드나 기술 스택을 사용하는 것은 회사 입장에서 리스크입니다. 평생 한 직장에 다니지 않기 때문이죠. 항상 동료 누구나 대체 가능하도록 가독성 좋은 코드를 작성하세요. AI 도움을 받아 가독성을 위해 좀 더 상세한 주석을 달아보세요.

*소프트웨어 프로페셔널로 대우받기를 원한다면 프로처럼 행동해야 한다. - 소프트웨어 장인*

프로가 공부하는데, 연습하는데 돈을 아낄까요? 모든 경기에 참여할까요? 만나길 원하는 모든 사람을 만나고 다닐까요? 참여한 경기인데 최선을 다하지 않을까요?

모르는 것을 용기있게 인정하고 더 다가서세요. 목록화 하고, 정복하세요. 부끄러운 것이 아닙니다. 모르는 것을 모른다 인정하지 않는 것은 프로다운 모습이 아닙니다. 회사에서 사용하고 있는 기술스택을 나열해보세요. 가능하다면 AI 함께 하나씩 그 의미를 풀어보세요.

장인은 꾸준히 코드 베이스를 돌보고 두려움 없이 빠르게 리펙토링 한다. 장인에게 시간적인 제약이나 요구사항 변경이 나쁜 코드를 위한 변명이 될 수는 없다. 좋은 디자인 패턴을 전체 애플리케이션 생애에 걸쳐 적용했기 때문이다. 코드의 품질이 성공을 보증하지는 못하더라도 실패의 핵심 요인이 될 수는 있다. - 소프트웨어 장인

많은 책에서 다음과 같은 것을 언급합니다. 두려움, 용기, 보이 스카우트 규칙, 리펙토링, 디자인 패턴, 품질, 배포, 테스트 주도 개발, 설계, 문서화, 생산성 등이요. 이런 용어들에 대해 생각해보거나 공부해보신 적이 있을까요? 관심사가 아니라는 이유로, 사용하지 않는다는 이유로 들여다보고 싶지 않으신가요? 한계를 넓혀가세요.

*여러분이 오늘 사용 중인 언어와 5년 후의 사용하는 언어는 아마 다를 것이다. 오늘 사용 중인 프레임워크는 아마 내년에 사용하는 프레임워크와 다를 것이다. 여러분 주위의 무엇이 바뀌는지 전방위적으로 파악하여 이런 변화에 대비하라. 프로그래머에게 매년 새로운 언어를 배우라는 조언을 많이들 한다. - 소프트웨어 장인 정신 이야기*

하나의 언어에, 하나에 프레임워크에 너무 집착하고 있진 않으신가요? 아니, 집착도 안하고 있진 않으세요?

*컨벤션 조합으로 인해 생기는 인지 부하는 과소평가되고 있다. - 프로그래밍의 규칙*

컨벤션을 설계하고 정립하고 시스템화하는 것은 실제로 더 많은 시간을 들여야 하는 일입니다.

*대부분의 경우 사소한 성능 향상을 잊어야 한다. 섣부른 최적화는 만악의 근원이다. - 도널드 커누스*

여러분 프로젝트에 최적화를 하고 있으신가요? 접속자는 몇 명이죠?

기억하세요. 100배의 성능 향상이 필요할 정도로 심각한 실수라면 조기에 알아차릴 수 있습니다. 그렇게 심각한 실수는 수풀 속에 숨지 못하거든요. 심각한 실수는 처음부터 명확해서 대부분 빨리 발견됩니다. 다시 한번 말하지만, 걱정하지 마세요. 최적화의 두 가지 교훈을 믿으세요. 단순하고 명확한 코드를 작성하고, 자신이 마주하게 될 어떤 성능 문제도 해결할 수 있다고 믿으세요. - 프로그래밍의 규칙

*누군가 보고 있다는 것을 안다면 모두가 더 좋은 코드를 작성한다. - 프로그래밍의 규칙*

코드를 누군가에게 보여주고 피드백 받으세요. AI 도움으로 빠르게 코드 리뷰를 즉각 받을 수 있게 되었습니다.