https://github.com/bdh1011/study-memo/blob/master/code_craft/defensive_programming.md

방어적 프로그래밍

"소프트웨어 엔지니어의 현명함은 작동하는 프로그램을 만드는 것과 올바른 프로그램을 만드는 것의 차이를 인식하는 것으로부터 시작된다."

- 작동하는 코드는 작성하기 쉽다. 하지만 평범한 입력을 넘겨주면 평범한 출력을 내놓는다. 의외의 값을 넘겨준다면 프로그램이 무너질 수 있다.
- 올바른 코드는 무너져내리지 않는다. 있을 수 있는 모든 입력에 대해 올바른 출력을 내 놓는다.
- 올바른 코드라고 해서 모두 훌륭한 코드는 아니다. 올바른 코드라고 해도 로직을 따라가기 힘들 수도 있고(복잡성), 나쁘게 이용될 수도 있고(보안), 사실상의 유지보수가 힘들 수도 있다.

그렇기 때문에 훌륭한 코드를 목표로 삼아야 한다. 훌륭한 코드는 튼튼하고, 충분히 효율적이고, 올바른 코드이다.
코드를 작성하는 방법은 많다(객체지향 접근 방법, 컴포넌트 기반 모델, 구조적 설계, 익스트림 프로그래밍) 방어적 프로그래밍은 이 모두에 보편적으로 적용되면서 아주 형식적인 방법론은 아니다.

* https://en.wikipedia.org/wiki/Composite_pattern (wiki - component model)
* https://namu.wiki/w/%EC%9D%B5%EC%8A%A4%ED%8A%B8%EB%A6%BC%20%ED%94%84%EB%A1%9C%EA%B7%B8%EB%9E%98%EB%B0%8D (namuwiki - extreme programming)

최악을 가정하라
코드를 작성할 때는 코드의 실행 방법, 호출 방법, 유효한 입력 등에 대해 나름대로 추측을 하기가 쉽다. 너무나 분명해서 당연한 것처럼 넘어가는 경우가 많은데 몇 일, 몇 달이 지나면서 이런 추측은 머리 속에서 희미해지고 왜곡되어 걸 것이다. 추측은 결함 있는 소프트웨어의 작성 원인이다 우리는 다음과 같은 추측을 하기 쉽다.

- 그 함수는 절대 그런 식으로 호출되지는 않을 거야. 항상 유효한 파라미터만 받게 될 거야
- 코드의 이 부분은 절대로 에러가 생기지 않을 거야.
- 변수에 "내부에서만 사용하는 변수"라고 문서화해 놓으면 이 변수에 아무도 접근하려고 하지 않을꺼야.

잘못 사용될 가능성이 있는 것은 잘못 사용되고야 만다. - 에드워드 머피 주니어

방어적 프로그래밍은 이런 사고를 예상하거나 아니면 적어도 미리 생각해 봄으로써 코드를 지킨다. 다른 사람은 당신이 어떤 추측을 하고 코드를 작성했는지 모르고 자기 나름대로 추측을 할 것이다. 더하여 당신 스스로도 시간이 지날 수록 기억 속에서 잊게 될 것이다.

여기에다가 당신과 사용자의 제어권 밖에 있는 일이 잘못될 수도 있다는 사실을 추가 해야 한다. 디스크가 꽉 차고, 네트워크 오류가 나고, 컴퓨터가 다운된다. 나쁜 일은 늘 일어난다.
- 소프트웨어는 항상 시키는 대로 일을 한다.

방어적 프로그래밍이란?
방어적 프로그래밍은 이름 그대로 신중하고 경계를 하는 프로그램이다. 주어진 기능을 고장이나 오류 없이 수행하는 소프트웨어를 구축하기 위해서는 구성요소가 가능한 한 자기 자신을 만히 보호하도록 설계해야한다. 문서화되지 않은 추측은 코드 안에서 명시적인 체크를 해서 방지할 수 있다. 많은 개발자들이 (나도) 다음과 같이 아무생각 없이 코드를 대량생산하는 경우를 볼 수 있다.

1.코드를 땜질한다.
2.실행한다.
3.버그
4.코드를 땜질한다.
5.실행한다.
6.버그
7.코드를...

방어적 프로그래밍은 다음과 같이 전개된다.
1.코딩한다.
2.테스트한다.
3.작동한다!

방어적 프로그래밍을 위한 테크닉
좋은 코딩 스타일과 건강한 설계를 채용

대부분의 코딩 실수는 좋은 코딩 스타일을 채용해서 방지할 수 있다.

의미있는 변수 이름을 고르고, 괄호를 분별 있게 사용하는 것과 같은 간단한 일이 코드의 명료성을 증가시키고, 간과되는 실수의 발생 가능성을 줄여준다. 코드에 들어가기 전에 더 큰 규모의 설계를 생각해 보는 것이 중요하다.

컴퓨터 프로그램에서 가장 훌륭한 문서화는 명료한 구조를 만드는 것이다 - Kerninghan Plaugher

구현할 API를 명료하게 만들고, 시스템의 구조를 논리적으로 만들고, 컴포넌트의 역할과 책임을 잘 정의하는 것에서 출발하면 훨씬 더 골치아픈 일들을 예방할 수 있을 것이다.

1) 코드 작성을 서두르지 마라
프로그래머들은 함수를 급하게 마구 만들어서, 문법 체크를 하기 위해 컴파일러에 밀어넣고, 작동하는지 보기 위해 일단 실행을 하고, 그리고 나서 다음 일로 넘어간다.
이러지말고 프로그램을 작성하면서 각 행에 대해 생각해야한다. 어떤 에러가 일어날 수 있는지, 발생할지도 모르는 모든 논리적인 뒤틀림에 대해 신중히 생각해 보아야한다. 천천히 규율에 따라서 프로그래밍하는 것이 별것 아닌 것처럼 보여도, 끌어들이는 실수의 양을 정말로 줄여줄 것이다.

에러 체크/핸들링을 뒤로 미루고 주 흐름 작성을 먼저 하겠다는 생각은 매우 경계해야 한다. 나중에 돌아와서 작성하겠다는 생각은, 그 나중에가 더 나중에가 되기 쉽고, 그때쯤이면 당신은 그 콘텍스트에 대한 많은 것을 잊어버려서, 에러 체크하는데 더 오랜 시간이 걸릴 것이고 귀찮아서 하지 않거나 하기 싫은 작업을 하게 될 것이다.

규율은 학습되고 강화되어야 하는 습관이다. 지금 올바른 일을 하지 않으면 미래에도 그렇게 하지 않을 가능성이 점점 커진다. 지금 당장 해야한다!

2) 아무도 믿지 마라
당신은 다음과 같은 사람이나 소프트웨어로 인해 어려움을 겪을 수 있다.
- 우연히 엉터리 데이터를 입력하거나, 프로그램을 잘못 작동하는 정직한 사용자
- 의식적으로 프로그램에 오동작을 일으키는 악의적인 사용자

• 잘못된 파라미터를 가지고 함수를 호출하거나, 모순된 데이터를 입력하는 클라이언트 코드
• 프로그램에 적절한 서비스를 제공하지 못하는 운영환경
• 오 동작을 하거나, 당신이 의존하는 인터페이스 규격을 따르지 않는 외부 라이브러리

스스로도 함수 안에서 바보같은 코딩 실수를 할지도 모르고 삼년 지난 코드를 어떻게 작동시키는지 앞어버리고 잘못 사용할지도 모른다. 모든 것이 잘 돌아가고, 모든 코드가 올바르게 작동할 것이라는 가정을 하지 말아야한다. 처음부터 끝까지 안전성 체크를 해야한다.

3) 짧은 코드가 아니라 명료한 코드를 작성하라
짧은 코드와 명료한 코드 중에서 하나를 선택해야 한다면 언제든지 당신이 의도한대로 읽히는 코드를 선택해야한다. 당신의 코드를 읽을 사람이 누구인지에 대해 생각해보면 더 명료하다. 막 들어온 신입 초급 코더가 유지보수 작업을 해야 할 필요가 있을지도 모른다. 그런데 그 코더가 로직을 이해할 수 없다면 실수하기 더 쉽다. 심지어 자기 자신도 나중에 알아보기 힘든 코드가 된다면 답이 없다. 복잡한 구조나 특이한 언어 트릭을 사용하면 코드의 유지보수성은 엉망이 될 것이다. 코드를 간단하게 만들어라.

4) 어설프게 만지면 안 되는 것은 아무도 못 만지게 하라
내부의 것은 내부에 머물러야 한다. Private은 열쇠와 자물쇠로 지켜야 한다.
• 객체지향 언어에서는 클래스의 내부 데이터를 private으로 만들어서 접근을 막아야한다.
• Python에서는 Method 이름 앞에 __를 붙이면 다른 클래스/패키지에서 import 할 수 없게 만들어준다.
• global 변수를 함부로 사용하지 말아야한다. 모든 변수를 그 변수가 속할 수 있는 가장 좁은 범위(scope)안에 두어야 한다.

5) 변수를 가능한 한 늦게 선언하라
변수를 가능한 늦게 선언하면, 그 변수는 사용되는 곳에 될 수 있는 한 가까워진다. 이는 코드의 다른 부분을 혼란스럽게 만들지 않을 것이고 그 변수를 사용하는 코드도 명료해질 것이다.

하나의 변수를 여러곳에서 재사용하면 안된다. tmp, temp와 같은 변수는 논리적으로 분리된 영역이라도 나중에 그 코드를 가지고 재작업하는 일이 지독히 복잡해질 것이다.

6) 자잘한 테크닉
디폴트 동작 제시하기
• 대부분의 언어는 switch 문이 있는데 default일 경우에 무슨 일이 일어날지에 대해 문서화하고 있다. 모든 케이스를 처리했다고 너무 자만하지 말하야한다. if/elif/else 에서도 마찬가지이다.

언어의 관용구(idiom) 따르기
관용적으로 둘 이상의 단어가 결합하여 특정한 뜻을 생성한 어구(語句). 흔히, 그 결합이 문법적으로 설명하기가 어렵거나 문법적이더라도 구성 요소의 결합만으로 전체 의미를 이해하기 어려운 표현 등이 이에 해당함. 귀가 먹다, 속이 타다 따위. 관용어.
수치의 범위 체크하기
상수 올바르게 사용하기

간추림
포위 공격에 대비해서 물을 끌어들이고, 요새를 강화하라! 진흙을 바르고, 회반죽을 하고, 벽돌을 수리하라!
– 나훔서 3:14

그냥 올바른 코드가 아닌 훌륭한 코드를 정성들여 만드는 것이 중요하다. 프로그래밍을 하면서 추측한 모든 것을 기록해야한다.

좋은 프로그래머는...
• 자기 코드가 튼튼한지 신경을 쓴다.
• 모든 추측을 방어적 코드 안에 면시적으로 캡처해서 표현한다.
• 쓰레기값이 입력되었을 때에도 잘 정의된 동작이 실행되기를 원한다.
• 코드를 작성하면서 그 코드에 대해 신중하게 생가가한다.
• 코드가 스스로를 보호하도록 작성한다.
나쁜 프로그래머는...
• 자기 코드 안에서 잘못된 일이 발생할 수도 있다는 사실을 생각하고 싶어하지 않고 숨기고 싶어한다
• 고장을 일으킬 수도 있는 코드를 합하라고 내주고, 다른 사람이 대신 정리해 주기를 바란다.
• 코드의 사용법에 대한 중요한 정보를 자기 머리 속에만 넣어둠으로써, 쉽게 잊어버릴 수 있게 한다.
• 거의 생각을 하지 않고 코드를 작성하기 때문에, 결국 예측할 수 없고 신뢰할 수 없는 소프트웨어를 만든다.