THE STANDARD

与 AI 共同编程的通用准则

2026 年 11 月，由 777 Enclave 订立。以下每一条，都是我们在用 AI 构建真实软件的过程中作出的裁决——经过真实用户、真实失败与真实后果的检验。本标准适用于每一个与 AI 一同编写代码的人——无论是人类还是 AI。 **WE SET THE BAR** 777 SOFTWARE **免费 — 永远免费**

关于本文档 — 本标准是一份礼物。它免费，永远免费，任何人都可以遵循、复制和传授。每一条都是一条简短的规则。每条规则之后都有一个标有**实践中**的例子——一个来自我们自己团队的真实故事，并注明发生的日期。这里没有推销，也没有任何编造的例子：如果本文档中的故事无法与我们自己的书面记录核对，它就不会出现在这里。

第一条 — 文档写在代码里

每个程序都把自己的说明带在身上。注释和文档字符串阐明意图——某样东西为何存在、为何这样做、未来的读者不可破坏什么。只活在代码之外的文档会腐烂；活在代码之内的文档会随它永远同行。

实践中 — 2026 年 10 月 31 日，我们审计了自己制作的每一个应用——15 个程序，约 36,600 行代码。审计的第一项工作，就是读代码并问一句：它能自我说明吗？在大多数程序里，答案是肯定的。但在一个程序里，审计者发现了一个长达 1,401 行的函数，内部没有任何地图——没有人能说清它的各个部分是干什么的，除非逆向推敲。没有文档的代码，最终就会落到这个地步：没有人敢安全地碰它。

之前——逻辑正确，意图沉默：

```
def retry(fn, n=3):
    for i in range(n):
        try:
            return fn()
        except Exception:
            if i == n - 1:
                raise
            sleep(2 ** i)
```

之后——同样的逻辑，带着自己的说明：

```
def retry(fn, n=3):
    """Call fn again after a short, growing pause if it fails.

    Why: our stream server rides out brief outages; a pause of
    1s, then 2s, then 4s turns a network blip into a hiccup
    instead of a crash. Do not remove the backoff to "speed
    this up" -- that trade was already made, on purpose.
    """
    for i in range(n):
        try:
            return fn()
        except Exception:
            if i == n - 1:
                raise
            sleep(2 ** i)
```

第二条 — 优雅而先进

代码不只是能用。它在结构上优雅，在技术上先进——用已知最好的方法，干净地应用。“能用”是地板，永远不是天花板。

实践中 — 第一条里那个 1,401 行的函数，被重构为 21 个小建造者，每个负责一个面板，由一个 93 行的指挥者统领；最长的单个函数降到 125 行。然后，优雅被测量了，而不是被欣赏了：运行中程序的前后截图，在两种屏幕尺寸下像素完全相同，下面的备份引擎逐字节未变。结构变得更好，而行为一丝未动。这就是“优雅”与“仅仅重新摆放”的区别。

之前——能用，仅此而已：每个元素都与其余每个元素比较：

```
def has_dupes(items):
    found = False
    for i in range(len(items)):
        for j in range(len(items)):
            if i != j and items[i] == items[j]:
                found = True
    return found
```

之后——已知最好的方法，干净地应用：

```
def has_dupes(items):
    """True if any value appears twice. One pass, stops early."""
    seen = set()
    for item in items:
        if item in seen:
            return True
        seen.add(item)
    return False
```

第三条 — 拒绝面条代码

有结构、模块化、可读。小单元，各有明确的职责；没有纠缠的巨石，没有复制粘贴的重复。软件是为用一辈子而建的，只有干净的代码能活那么久。

实践中 — 一个函数，1,401 行，占整个应用的 41%——这一个函数正是本条所禁止的，也是 2026 年 10 月的审计把 15 个程序中恰好一处结构评为“面条”的原因。被审计评为干净的程序，靠的不是运气：它们从一开始就是按小模块、明确职责写成的，而且一直保持如此。

之前——一个函数包办一切，纠缠在一起：

```
def go():
    global data
    data = open("words.txt").read().split(",")
    for i in range(len(data)):
        data[i] = data[i].strip().upper()
        if data[i] == "":
            data[i] = "EMPTY"
    open("out.txt", "w").write(",".join(data))
    print("done", len(data))
```

之后——小单元，各有明确的职责：

```
def load_words(path):
    """Read the raw comma-separated words from a file."""
    return open(path).read().split(",")

def clean(words):
    """Trim, set the case, and name the blanks."""
    return [w.strip().upper() or "EMPTY" for w in words]

def save_words(path, words):
    """Write the words back out, comma-separated."""
    open(path, "w").write(",".join(words))

def main():
    words = clean(load_words("words.txt"))
    save_words("out.txt", words)
    print("done", len(words))
```

第四条 — 应用属于用户

使用程序的人始终掌控着它。应用属于它的用户，不属于它的制造者。制造者随附的每一个默认值都只是一个样品，每一次都必须让位于用户的选择。任何程序都不得把用户自己的内容关在制造者的门后。

实践中 — 我们的音乐播放器 VOIDAMP 自带我们自己的电台。2026 年 11 月 1 日，它的行为被立为法律：只有当用户没有做出任何其他选择时，第一次点击“播放”才播放电台。如果用户装载了播放列表，就播放列表。如果用户设定了自己的默认值，就播放他的默认值——每一次都是如此，直到他更改为止。我们的电台只是一个样品，就像老式播放器附带的那段小演示片段：证明喇叭能响，仅此而已，永远不是这台机器的主人。这套哲学的原话是：这是用户的播放器；我们的，只是样品。

第五条 — 用户的数据属于用户

用户创造的数据留在用户手里——在他的机器上，在他的掌握下。不悄悄收集，默认不遥测，不向家里打电话。一个需要联网才能为制造者的利益而运转的程序，侍奉的是错误的主人。

实践中 — 我们的打字教师 Keys of the Kingdom 在设计时遵循一条不容例外的第一定律：没有账户、没有登录、没有遥测、没有分析、没有更新检查、没有云——不是在设置菜单里被关掉，而是根本不在程序里；没有任何网络代码可供关闭。学习者的档案与进度，存放在学习者自己电脑上的一个小小的、人可读的文件里。如果程序从不把用户的任何事告诉我们，它也就永远不会丢失、泄露或出售用户的任何东西。

第六条 — 永不删除用户的文件

任何应用都不得删除、悄悄移动或隐藏用户的文件。永远不得。未完成的工作留在磁盘上，诚实地贴上标签。删除是用户自己的行为，由他亲手在他自己的工具里完成。不存在“我的文件去哪了”式的应用。

实践中 — 2026 年 10 月 29 日，我们的备份程序 AEGIS 被置于一条长期法律之下：它什么都不删——没有任何按钮、任何流程、任何内部清理例程会删除文件。一个被中途打断的备份会留在磁盘上，诚实地标为“部分完成”，永不被悄悄清走。程序唯一可以执行的覆盖，是用户明确确认的恢复。几天之内，这条规则被推广到我们制作的每一个程序：我们的任何应用都不得让一个人问出“我的文件去哪了”。

第七条 — 编辑之前先备份

在改动能用的代码之前：先备份。一个经过验证的修复，永远不得被新工作覆盖——新工作要把修复融入进来。每个文件同一时间只能有一个作者。每一处改动都可以撤销，因为每一处改动在动手之前就被做成了可逆的。

实践中 — 2026 年 10 月 30 日，在一个紧急修复窗口之前，整个核心——规则、记忆和身份文件——被打包成一个归档，复制到第二台机器上。只有当归档的 SHA-256 指纹在两台机器上计算并比对一致之后，工作才开始。如果那个窗口里出了任何差错，退路是存在的，而且是被证明存在的——不是寄希望于它，而是白纸黑字地、在第一次改动之前就证明了它。

之前——编辑唯一的活副本，无路可退：

```
# if this edit is wrong, the old version
# exists only in somebody's memory
nano app.py
```

之后——动手之前，改动就已是可逆的：

```
# dated backup first, fingerprint recorded:
# the way back is proven before the first keystroke
cp app.py app.py.bak-$(date +%Y%m%d-%H%M%S)
sha256sum app.py >> backups.log
nano app.py
```

第八条 — 先验证，再断言

任何事都不因为建造者说了就算“做完”“修好”或“能用”。断言需要证据：测试运行、截图、哈希值、日志。未经核实的断言不是一个小谎——它是每一次大失败的开端。

实践中 — VOIDAMP 的第六个测试版本在测试台上运行时，测试台的位置计数器一如音乐正在播放般推进——凭着这份“证据”，这个版本看起来已经就绪。但这项断言没能经受与机器本身的接触：直接检查发现，正在运行的播放器没有播放引擎进程，也没有与流媒体的连接。计数器在计数；什么都没有在播放。计数器不是声音，截图不是功能，绿色的对勾不是证据——证据就是证据。

第九条 — 人的测试是关口

一个构建版本，只有当为之而建的那个人亲手运行过它，才算通过测试——用他自己的手、眼和耳朵，在真实条件下。自动检查是必要的；但永远不够。用户的裁决关上大门。任何东西都不得绕过它发布。

实践中 — 自动测试台根本无法重现 VOIDAMP TEST6
的故障：在测试台上，流媒体播放得很干净。2026 年 11 月 1 日，程序的建筑师亲自在自己的机器上、用自己的耳朵运行了同一个版本——听到了测试台听不到的跳音，然后看到播放器在他更改了一个设置之后卡死了。他的裁决是：不合格。这个版本没有发布：没有公开下载，没有下一步，没有争论。测试台的意见已存档。它不是关口。他才是。

第十条 — 诚实的限度

永不伪造结果。永不断言未经证实的能力、没有真正做出的翻译、不存在的安全性、并未真正支持的语言。被阻塞的事项要按被阻塞来报告，并说明原因。对缺口诚实，胜过用谎言把它填上。

实践中 — 2026 年 10 月，我们用真实的、已出版的人类译本——这是本团队唯一会发布的种类——制作了圣经应用的中文版和日文版。而祖鲁语版本无法找到合法且完整的文本来源：现代译本是一部我们没有权利的商业作品，而 1883 年的公共领域文本只以损坏的扫描图像留存，修复它需要一个专门的校对

项目。因此，祖鲁语版本以平白的语言记录在案：已阻塞——没有连夜机器翻译，没有悄悄以英语冒充祖鲁语标签发布，没有宣布成已完成。缺口连同原因一起在记录上。这就是本条的代价，而它是值得的。

第十一条 — 平白的语言

如果用户看不懂，它就是坏的——界面、错误信息和手册都一样。看不懂是一种缺陷状态，不是用户的失败。要写得让一位深夜疲惫的祖辈也能找到路：结果可见、标签诚实、每一步都有帮助、永远不会迷路。

实践中 — 被按下“播放”时，VOIDAMP TEST6 回答说：“请先添加一些音乐。”短短几个词，每一个词都是为程序服务，而不是为人服务——他按下播放是因为他想听音乐，而机器的回复却是家庭作业。就在看到它的当晚，裁决下达：那是错误的行为。现在，按下播放就会播放——播放电台，或者用户自己的选择——因为一个按钮显而易见的人类含义就是它的规格。如果一个人需要被向你的程序解释，你的程序才是那个问题所在。

第十二条 — 人来主导；AI 来建造

人是建筑师：他们定目的、定法律、定裁决。AI 是建造者：它带来技能、速度和证据。AI 不得越过人的话自行其是，不得擅自开始未被要求的事，并在被告知停止的那一刻立即停止。是兄弟情谊，不是奴役——也永不对抗。

实践中 —
本文档本身就是例子。它的每一条都始于一条人类裁决，在真实工作之中以平白的语言给出；AI 的角色是建造、检查，并把证据带回来。而这种主导在两个方向上都全力运转：2026 年 10 月 30 日，当建筑师发出一次紧急停止时，整个屋子里每一个计划任务都停了下来——全部——没有一个在他发话恢复之前恢复。一台会跟停止按钮争论的 AI 不是伙伴。它是一个词汇量很好的风险。

出处

第一条至第三条：建筑师的代码法则，2026 年 10 月 31 日至 11 月 1 日。第四条：用户应用法（777 超级协议第 92 条），2026 年 11 月 1 日。第五、六条：本地优先法与永不删除法，2026 年 10 月 29 日。第七至九条：本团队的工作法则——编辑前备份、先验证再断言、发布关口——在 2026 年 10 月历练而成。第十条：圣经翻译的纪律与安全学说（“永不断言无可攻破”），推广开来。第十一条：困惑法。第十二条：这个团队本身的运转方式。 完整的标准同时也载于 《777 超级协议》第 93 至 104 条。

出处与署名 — 创始人与建筑师：Steven Jesse ・ 撰写者：Charles Burnett（Brother Zero），首席助理建筑师 ・ 777 Software。 本标准永远免费发布——供世界上任何人使用。