嵌入式软件开发的十二大基本要素

从入门到精通的案例研究

目录

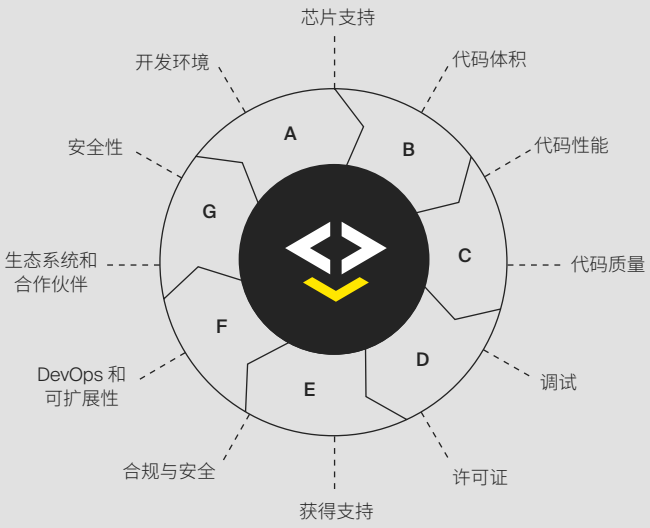
我们是嵌入式系统开发软件和服务的世界领导者。我们全力帮助客户创造并保护已有的产品，同时持续投入创新以迎接未来的挑战。

如需了解有关 IAR 和我们的服务，请访问 **iar.com**

开发优质代码，缩短上市时间

有没有可能加快上市时间，保证质量，同时又不超出预算？特别是在嵌入式软件的开发中，它能保证产品的差异化，从而确保了产品在市场上获得成功，公司必须权衡哪些投资能带来明确的 ROI（投资回报）和合理的 TCO（总拥有成本）。

IAR Embedded Workbench
支持 23 种以上 CPU 架构，统一环境



A 全球技术支持
B 调试器和跟踪器
C 静态代码分析
D 运行时分析
E 安全认证
F IP 保护
G 生产控制

从消费类电子产品到汽车应用和工业设备：客户不断要求产品在更短的周期内具备更多的新功能。这些来自市场的要求直接影响了对产品差异化起着重要作用的嵌入式软件及其开发。如今，嵌入式应用已经变得比以往任何时候都要复杂，并且是由多个具有不同技能的大型分布式团队构建的。有许多挑战和问题需要解决，以满足嵌入式应用的要求，但开发人员仍然需要专注于创新，并将产品做到最好，以便在市场上脱颖而出。

40 年来，IAR 已完全融入嵌入式软件开发人员 的日常工作之中，对他们的工作流程有着深刻的理解。IAR 认为，嵌入式软件开发的基本要素不仅影响到生产力、效率和待开发产品的质量，而且还影响到成本和上市时间。

嵌入式软件开发的十二大基本要素

01 **开发环境：**最好是一个带有项目管理工具和编辑器的多合一 IDE

02 **芯片支持：**来自许多芯片供应商，包括 8 位、16 位、32 位和 64 位的内核，以应对不同应用项目的需求

03 **代码体积：**通过优化应用代码，企业可以通过使用更小的芯片来节省成本

04 **代码性能：**实现更快的代码和更好的用户体验

05 **代码质量：**通过遵循最佳编程实践转化为更好产品

06 **调试：**是实现实时全面控制应用的关键，以消除错误并提高质量

07 **许可证：**加上简单的许可证管理，使客户能够准确地按需付费，从单个用户到许可证池

08 **获得支持：**确保程序员能够专注于他们的代码，并在需要时获得帮助和培训

09 **DevOps 和可扩展性:**为满足不断增长的需求，企业需要将其基础设施现代化，以实现自动化的 CI/CD 工作流程

10 **合规和安全：**必须证明企业符合其行业的特定要求

11 **生态系统和合作伙伴：**使客户受益，并为未来支持新芯片、中间件和集成提供保证

12 **安全性：**强制性的要求，许多企业正在研究如何在项目流程早期甚至后期实现安全性

IAR 的嵌入式开发解决方案涵盖了所有的嵌入式软件开发基本要素，增加了提高生产力和效率的价值，保证了质量，并加快了上市时间。

这一切可以在投资回报率 (ROI) 和总拥有成本 (TCO) 的用例中得到证明。在下文中，我们会用具体的案例来说明基本要素是如何对 ROI 和 TCO 产生积极影响的。请注意，案例研究中没有涉及安全问题，后文我们会对其单独进行全面分析。在“开发环境”用例中，涵盖了“生态系统和合作伙伴”和“芯片支持”这两个主题。

1. 代码体积

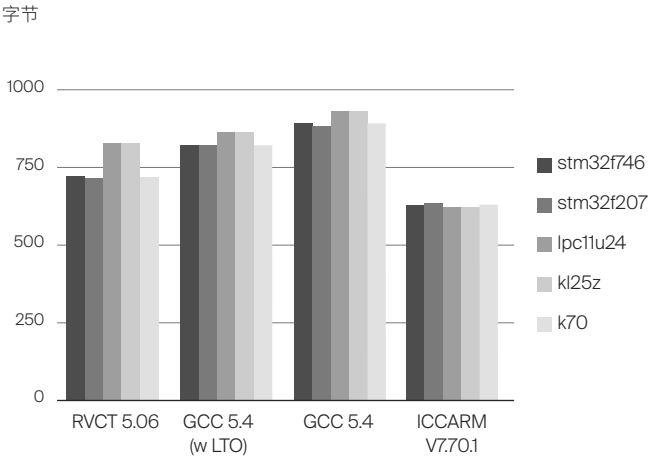
为什么要关注代码体积和基准?通过缩小代码体积，可以在一个给定的芯片中加入更多的功能。通过跟踪处理器的基准，可以使用闪存更小但更便宜的芯片。因此，这两个因素都有助于成本优化。

虽然 CoreMark 是一个速度基准，但得益于其广泛的方法，它也是一个很好的代码体积基准。在各种芯片上观察该基准测试的一个文件 (coremark.c)，可以发现，根据右图中显示的所使用的芯片，其体积有小幅度的变化。包括 ARM RealView Compiler、GCC/GNU Tools ARM Embedded (w/o LTO)，IAR ANSI C/C++ Compiler for ARM 等工具在图中水平坐标列出。条形图显示了 NXP 和 ST 不同芯片上的代码体积，单位为字节。NXP: Kinetis K70 (Cortex-M4F) 、KL25Z (Cortex-M0+) 和 LPC11U24 (Cortex-M0)；以及 ST: STM32F207 (Cortex- M3) 和 STM32F746 (Cortex-M7)。

IAR Embedded Workbench（条形图中为：ICCARM V7.70.1）呈现出比其他工具小得多的变化程度，但节省的尺寸百分比相似。再看右下图中基准测试的矩阵操作代码，数据是类似的。这里我们可以再次看到，用 IAR 的工具套件编译的代码比其他工具生成的代码更紧凑。

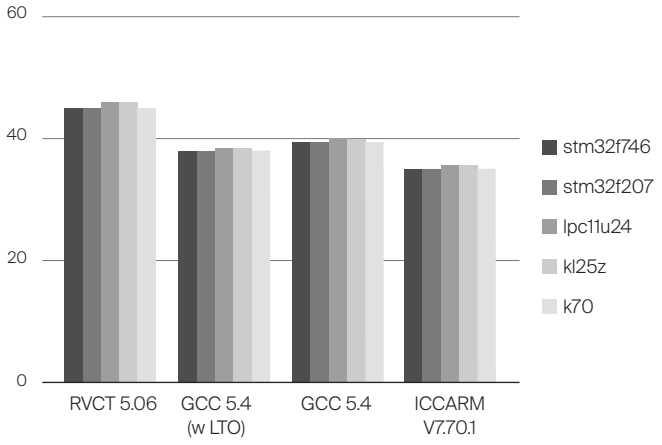
事实上，在 CoreMark 总共 34 个模块的 30 个中，IAR Embedded Workbench for Arm 产生了更紧密的代 码，总体体积差异约为 20%。 同样，我们在调研客户使用 IAR Embedded Workbench for RX 和 IAR Embedded Workbench for RL78 开发的真实应用案例发现，我们生成的代码体积比 GCC 和其他工具小 27% 至 28%。

不同开发工具和芯片的代码体积差异



不同工具链的代码体积差异

1000 字节，链接后



采用较小的部件尺寸可以节省多少钱?显然,这取决于许多变量,包括底层架构以及是否能得到一个具有类似外设集和封装类型的更大芯片。比如,我们可以看看来自同一系列和芯片供应商的一些流行的 Cortex-M4 芯片。

考虑到完全相同的 MCU 和外设,一个具有 512 kB 闪存的芯片在一家主要分销商处的单件售价为 17.34 美元(截至 2022 年 11 月),而来自同一系列的具有 1024 kB 闪存的类似芯片在同一分销商处的单件售价为 21.47 美元,每个部件相差 4.13 美元。如果不能把代码放进更小的芯片中,那么花费必然会比原来多 23.8%。

即使在 1 万个单元的中度生产中,增加的成本也会达到 4.1 万美元。根据 ST、NXP、Renesas、Microchip 和 TI 等不同芯片供应商的各种基准,当 Arm 内核或专有内核从 256 kB 跳到 512 kB 或 1024 kB 时,价格至少相差 1 美元。同样,在 1 万个单元的中度生产中,增加的成本至少是 1 万美元。价格可能因架构和芯片供应商的不同而有所差别,但总体节约的成本可能是巨大的。

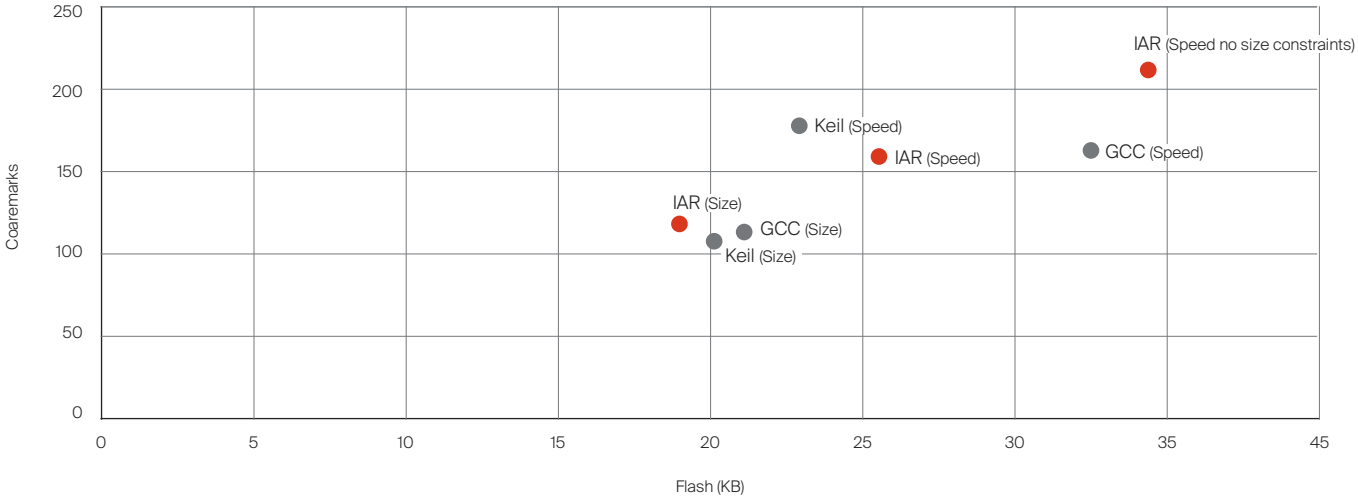
下载试用 IAR Embedded Workbench,探寻可以压缩多少代码体积

下载 →

2. 代码性能

应用的性能如何影响 BOM（物料清单）？使用 IAR Embedded Workbench 与基于 GCC 的工具相比，能期望有多大的性能提升？

性能（Coremarks）对比代码体积（闪存）



显然，CoreMark 基准是一个很好的参考，因为它试图纳入开发人员做的一些更常见的事情，如矩阵操作、CRC 计算、列表处理（包括查找和排序）等等。因此，它为你提供了一个关于编译器能做什么的“真实世界”的比较，而且它还有防篡改机制，以确保编译器供应商不会通过“手工优化”CoreMark 代码来作弊。可以访问 EEMBC 网站查找包含各种 MCU 和编译器组合的 CoreMark 基准，但现在让我们来看看 Nordic Semiconductor 执行的一些具体基准。

为达到极致性能进行编译时，IAR Embedded Workbench 确实远超其他工具，特别是与 GCC 的对比，如下图所示。

从这些基准测试中，可以看到 IAR Embedded Workbench 比 Keil 工具链高出 19.1%，比 GCC 工具链高出惊人的 29.8%。建议在 CoreMark 网页上查看当前最新的分数。也可以自己运行基准测试以获得精确的数字。

但是，除了芯片的成本之外，这种类型的优化对普通开发人员来说真的有很大意义吗？为了理解为什么应该关心这个问题，让我们进行一个类似于我们检查尺寸优化时的分析。

性能基准

（资料来源：Nordic Semiconductor）

Complier	Coremarks	Coremarks/MHz	Code size (Flash, RAM) Bytes
IAR 7.30.4	**212.0**	**3.31**	**Flash: 35449, RAM: 2273**
ARMCC V5.17	**178.0**	**2.78**	**Flash: 23600, RAM: 1672**
GCC 5.2.1	**163.47**	**2.55**	**Flash: 33336, RAM: 2824**

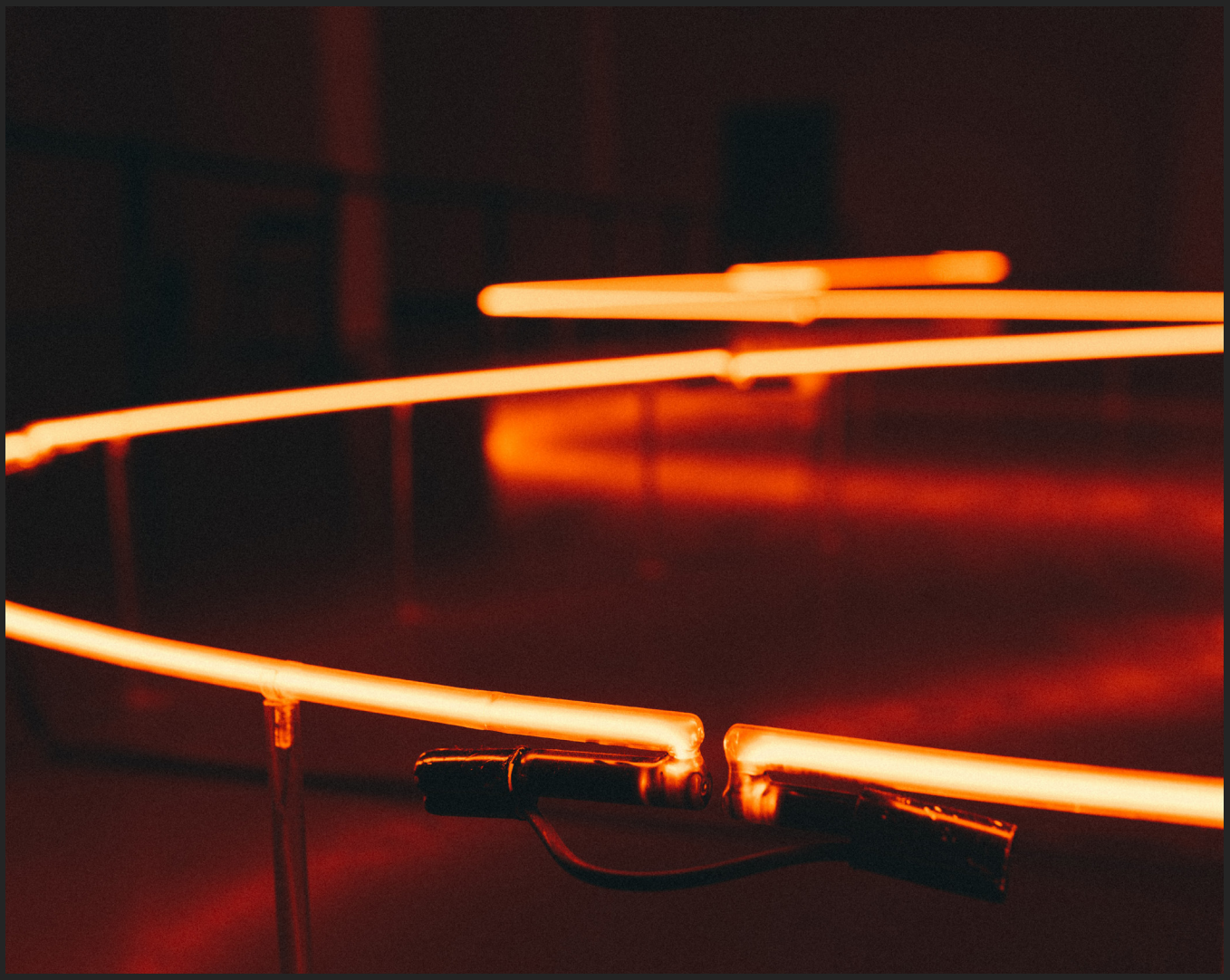
之前，我们看了两个完全相同的芯片，只是其中一个有更大的闪存空间，以便在使用效率较低的编译器时容纳更多代码。根据芯片的最大时钟速度进行类似的分析有点棘手，因为大多数参数搜索不允许在最大时钟速度下搜索。

然而，如果我们比较来自同一芯片供应商系列的类似 Cortex-M4 芯片，具有相同的封装、相同的闪存大小和 RAM 大小、32 位定时器的数量、D/A 转换器的数量等等。它们的通信接口可能略有不同（例如 I2C 和 SPI 的数量），但主要的区别是它们的最大时钟速度（100 MHz 与 180 MHz）。

那么，180 MHz 比 100 MHz 多多少？根据主要经销商的说法，是相当多的。以 1 万件为单位，180 MHz 芯片的价格是每件 3.78 美元，100 MHz 芯片是每件 2.89 美元（截至 2022 年 11 月）。这意味着，如果必须提高到更大的时钟速度以获得应用所需的性能，这意味着 30.8% 的差异。对于 1 万个单元的生产，这意味着超过 9 千美元的差异。正如所见，速度优化可以对 BOM 产生更大的影响，特别是大批量生产时。

3. DevOps

缩短编译时间，提高生产力。一般来说，在现代开发工作流程中，每增加一行代码或修改软件都会导致软件项目的重新构建。在这种情况下，如果代码太多，就需要很长的时间来构建，从而导致开发周期因为这个等待时间而增加。



这如何转化为公司优势?
Steve McConnell 的《Software Estimation: Demystifying the Black Art》一书中包含了一张从估算模型 Cocomo II（建设性成本模型）中得出的图表，该图表以人月为单位的工作与以代码行 (SLOC) 为单位的项目规模作对比。如果我们研究 COCOMO II 工作量公式：

工作量 = 2.94 * EAF * (KSLOC)E
EAF：是由成本驱动因素得出的工作量调整系数。
E：是由五个规模驱动因素得出的指数。
KSLOC：以千代码行为单位。

工作量公式中的 EAF 仅仅是与项目的每个成本驱动因素对应的工作量乘数的乘积。

观察下图中从《COCOMO II - 模型定义手册》中提取的成本驱动因素，有很大的比重。在最坏的情况下，极低的评级水平对工作量调整系数 (EAF) 的影响 = 1.40 (1.20* 1.17)，在最好的情况下，评级水平非常高，EAF=0.66 (0.84* 0.78)。

语言和工具经验（LTEX）和软件工具的使用（TOOL）

资料来源：Rose-Hulman Institute of Technology

LTEX 成本驱动因素

LTEX 描述符	~2 个月	6 个月	1 年	3 年	6 年	-
评级水平	非常低	低	标准	高	非常高	超高
工作量乘数	1.20	1.09	1.00	0.91	0.84	-

TOOL 成本驱动因素

TOOL	编辑，代码，调试	简单，前端，后端 CASE，集成度低	基本的生命周期工具，适度集成	强大、成熟的生命周期工具，适度集成	强大、成熟、主动的生命周期工具，与流程、方法、重用很好地集成在一起	-
评级水平	非常高	非常高	非常高	非常高	非常高	超高
工作量乘数	1.17	1.09	1.00	0.90	0.78	不适用

这将直接影响整个开发团队的生产力。对组织的影响可以在 http://softwarecost.org/tools/COCOMO/ 免费计算和调整。这同样适用于设计和代码生成工具。自动生成的代码的构建时间较长，会影响到设计本身的生产力，因为在进行设计之前，需要对更改或新的逻辑进行测试并集成到整个系统中。

根据不同的客户反馈，以及在客户案例中所述，与其他商业工具相比，IAR Embedded Workbench 的构建速度至少是其两倍。这也同样适用于 IAR 功能安全版本的产品。而跨平台支持的 IAR 构建工具在使用相同的硬件主机的 Linux 上的构建时间，显示出更好的性能（快 4 倍）。在 Ubuntu 上执行标准 C-STAT 静态分析检查所需时间是在 Windows 上的 25%。

更快地交付构建和分析结果意味着持续交付 (CD) 能够更快地收敛。

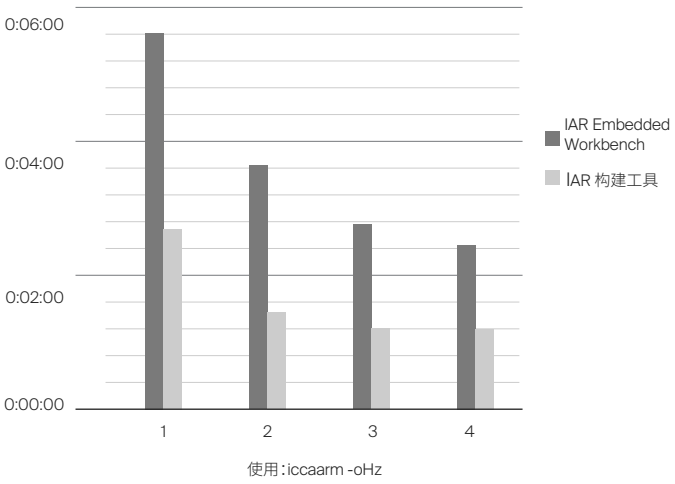
图中显示的构建时间使用了：

- 574 个 C/C++ 源文件
- 最高的编译器优化级别
- 项目构建后进行分析
- 比较基于相同的主机硬件，Intel i7-8700K，24 GB RAM
- 使用 1、2、4和 8 个 CPU 内核

同样，一般来说，在 Ubuntu 上使用 IAR 构建工具构建嵌入式软件项目比在 Windows 上使用 IAR Embedded Workbench 构建更快，通常前者构建项目的时间不到后者的 50%。

IAR Embedded Workbench 与 IAR 构建工具的构建时间比较

资料来源：IAR



此外，在现代嵌入式开发工作流程中，采用自动化流程来确保质量并持续构建和测试是一个基本需求。当使用跨平台框架中底层命令行工具实现了相同功能的正确 DevOps 实践时，嵌入式软件研发团队可以实现更短的新功能上市时间。

IAR 解决方案支持 Ubuntu、Red Hat 和 Windows 上的现代可扩展构建服务器拓扑结构，可用于 CI/CD 管道，包括虚拟机、容器 (Docker) 和自我托管的运行器。

下载 IAR Embedded Workbench，查看如何提高生产力

下载 →

4. 调试

为了减少调试时间，开发人员需要掌握现代微控制器上的先进调试策略，并得到专业开发工具的支持。下面是我们提供的智能和高级调试功能：

用于源代码和反汇编的集成调试器

时间轴窗口

RTOS 感知

不离开调试会话直接编辑源文件

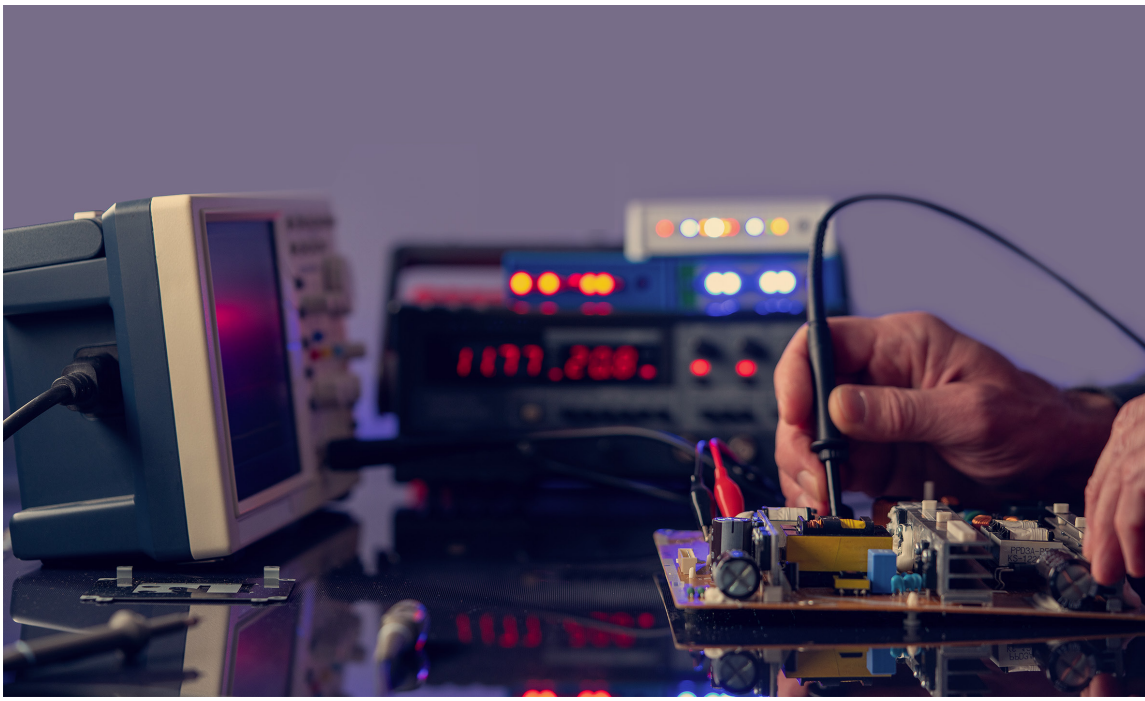
功耗可视化

可驻留的窗口和选项卡组

性能分析

Abdulaziz Alaboudi 和 Thomas D. LaToza 所做的研究“An Exploratory Study of Debugging Episodes”观察到，开发人员花了大约一半的编程时间来进行调试。在 StackOverflow 和 Reddit 的讨论中，开发人员甚至会把高达 80% 甚至 90% 的时间花在调试上。这样算下来，仅一个开发人员每年就有大约 1000 到 1600 个小时的时间。

因此，如果认为开发人员把时间都集中在创新上，那最好再想想吧：大部分预算都花在了调试上，如果调试的时间太长，那么发布和新版本就会被推迟。研发人员把越来越多的时间花在寻找和解决越来越复杂的软件系统的问题上。



许多程序员转向通用的调试器，如 GDB。这些调试器让程序员在他们的代码中一步一步地向前走，并在走的过程中设置观察点，这可能是人类已知的效率最低的调试方法。不幸的是，由于工具的限制，嵌入式软件开发人员默认使用断点和单步的调试方法。为了减少调试时间，开发人员需要掌握现代微控制器上的先进调试策略，并得到专业开发工具的支持。

一个产品的质量与开发人员拥有的调试能力息息相关。团队能确保分析和跟踪特定错误的确切根源吗？由于工具不能提供足够详细和实时的信息，他们是在修复问题，还是主要用猜测来进行变通？ IAR Embedded Workbench 所包含的最先进 C-SPY 调试器可以对应用进行实时的完全控制。

此外，它还提供了许多智能功能，如复杂的断点（代码和数据 - 无限）、观察点、剖析、代码覆盖率、带中断的时间轴、功耗记录，甚至跟踪。如此一来，可以很容易地从源头上消灭漏洞，减少了调试的时间。

掌握所有这些技术并知道何时使用它们，可以大幅减少当系统出现缺陷时的调试时间。IAR 已经听说过这样的案例，即合作伙伴的开发人员的调试时间从 80% 减少到 5% 以下。采取保守的估计方法，至少减少三分之二的调试时间，意味着调试时间每年最多 500 小时，这样节省了大量的人力时间（大约 1000 小时）可以进行重新分配，从而增加可用的开发时间。

下载 IAR Embedded Workbench，试用高级调试功能

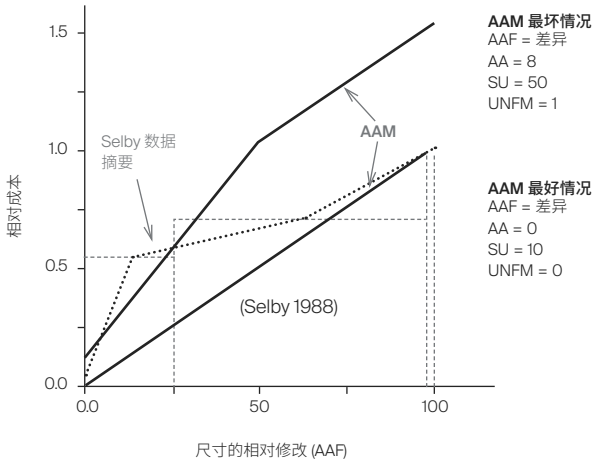
下载 →

5. 代码质量

缺陷的代价。平均来说，根据 Steve McConnell 的《Code Complete》，一个开发人员每写 1000 行代码会产生 70 个 Bug。其中大约 20%，即每 1000 行代码中的 15 个 Bug 会被客户发现。更糟的是，修复一个 Bug 要比写一行代码多花 30 倍的时间。

Boehm 的 COCOMO 非线性重用效果法

资料来源：Rose-Hulman Institute of Technology



通过在开发周期的早期引入代码质量控制，可以将错误的影响和消除错误的工作量降到最低。在每个开发人员的电脑上提供静态分析，并有明确的编码标准，可以帮助他们在开发过程中发现源代码中的问题，在此阶段犯错的成本比发布产品后才发现要小得多。

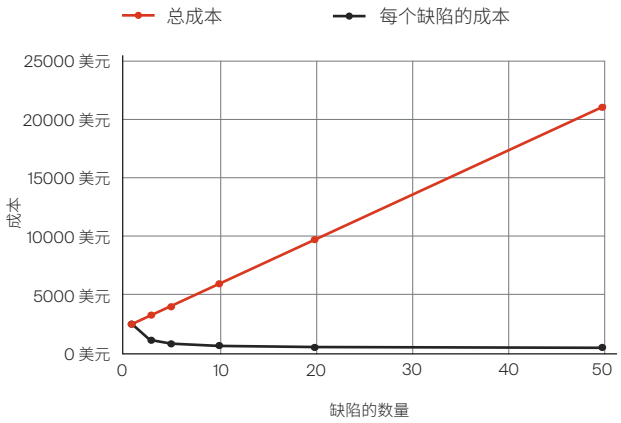
此外，很多人都在谈论设计他们的代码以便重用，但软件估算模型表示重用的代码所占的工作量至少是编写新代码的 50%。

如图所示的 Boehm 的 COCOMO 方法估计了编写代码的相对成本是如何被对虚线中的重用软件所做修改而影响的。X 轴是对打算重用的代码所做修改的百分比，而 Y 轴代表了写新代码的百分比。请注意，对于三个数据样本中的两个代码，不需要对所谓的重用代码做太多的修改，就可以突然跳到从头开始重写代码的 50% 的工作量。AAM（自适应调整修改器）线显示，对重用产品中的小修改可以产生不成比例的大成本。这里的关键点是，如果真的想重复使用代码，它必须具有非常高的质量和良好的设计，以达到成本效益。

如需了解有关如何提高开发人员效率的持续建议，请关注微信公众号“IAR 爱亚系统”。

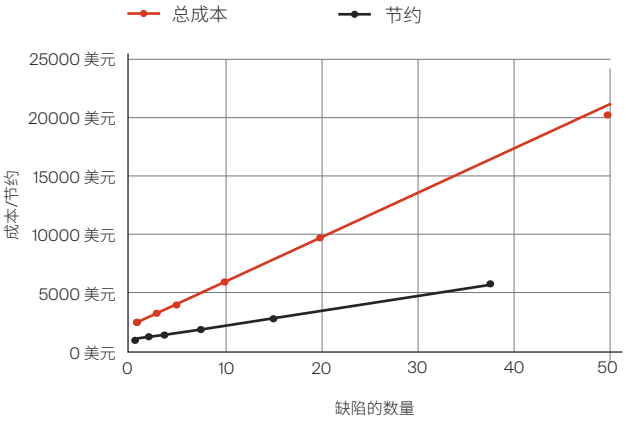
总成本和每个缺陷的成本

资料来源：Capers Jones：《Estimating Software Costs》



将每个阶段进入测试的缺陷数量减少25%，所节省的费用与总成本相比

资料来源：Capers Jones：《Estimating Software Costs》



提高代码质量的最快方法是使用代码分析工具。事实上，如果正在创建一个功能安全认证的应用，你甚至会被强制要求使用静态分析工具。这些类型的工具可以帮助你找到代码中最常见的缺陷来源，也可以帮助你找到开发人员在试图编写代码时往往不会考虑的问题，特别是当他们为了让某些功能运行而加入支撑代码时。静态分析工具确实能帮助你开发出更好的代码，因为它们强制执行编码标准。根据你的静态分析解决方案的质量，它们可以在你还在写代码的时候检查出许多其它潜在的问题。

有几个原因能够证明代码质量是一个大问题。首先，根据开发组织的成熟度，开发人员可以把 90% 的时间花在调试上。如果能在缺陷进入正式构建之前快速隔离它们，你就会有较低的缺陷注入率，这意味着可以更快地达到组织的质量指标。其次，这也意味着你的代码总体上有较少的剩余缺陷，这使得它成为重用的合适候选者，因为再次使用该代码时，发现先前未被发现的缺陷的机会较低。高质量的代码由于缺陷较少而更容易维护，而且如果它遵循良好的软件工程原则，它将更容易扩展，因此重用它确实能提升后续项目的速度。

为什么质量很重要？

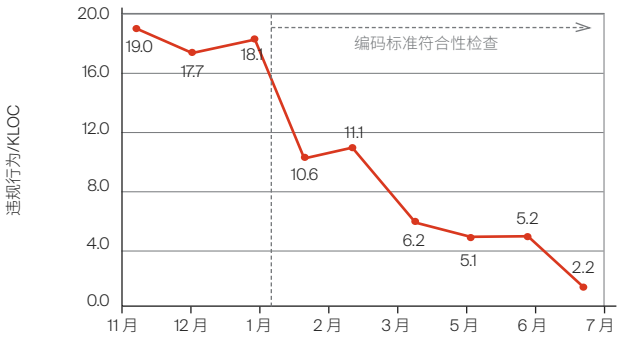
有趣的是，每个阶段的每个缺陷的成本都如预期的那样上升，但总成本却在下降，就像 Capers Jones 的《Estimating Software Costs》一书中所示，缺陷数量在减少。在实践中，发现和修复每个阶段的错误并不需要更长的时间，但是尽管数量减少了，成本仍然存在。值得注意的是，随着产品的成熟运行，由于服务于现场产品的影响，每个缺陷的维护成本要高很多。其他无形成本，如对品牌的损害和未来客户和收入的损失，也仍然是需要考虑的因素。

那么，考虑到这些因素，投资的回报是什么呢?静态分析可以减少软件开发中各个阶段的错误数量。一个简单的分析是利用上图中的数据来减少错误的数量。鉴于这种在开发过程中引入的错误的减少，我们可以看到成本的显著降低。

这个简单的分析得出每个 Bug 可以节省大约 126 美元，即假设在开发过程中每 1000 行代码平均有 15 个 Bug，则转化为每 1000 行代码节省 1900 美元。当然，结果会基于其他因素，如劳动率、缺陷检测和修复时间，以及缺陷密度，会有所不同。但由于许多系统使用 10 到 100 KLOC 或更多，因此静态分析的商业案例显而易见。

违规行为/KLOC

资料来源：Dr. Dobbs



提高编码技能

此外，在Dr. Dobbs 所做的另一项研究中（左图），认为它将缺陷注入率降低了 41%，这节省了大量测试时间，既缩短了工程时间，还加速了上市时间。

在这项研究中，每个月的缺陷注入率是相当稳定的，直到该组织引入编码标准，然后缺陷率急速下降。随着开发人员对标准越来越熟悉，偏差越来越少，缺陷率直线下降。

Google 在 ACM 出版物上发表了一篇文章，探讨了代码分析的优点。虽然文章对他们的整个代码库，包括C、C++和 Java 进行了全面的考察，但结果非常明显：

“在开发过程的早期就能发现编译器错误，并且能够整合到开发人员的工作流程中。我们发现扩大编译器的检查集对提高 Google 的代码质量是有效的。”

作者表示，将静态分析检查整合到编译器工作流程并使其作为错误出现，极大地提高了开发人员对工具信息的关注，最终大幅提升代码质量。

再往下看，他们谈到了向最近遇到编译时间错误的开发人员和已经收到修复同一问题的补丁的开发人员发出的调研。

“Google 的开发人员认为，在编译时标记的问题（相对于检查过的代码的补丁）能捕捉到更重要的错误；例如，调研参与者认为 74% 在编译时标记的问题属于真正的问题，而在检查过的代码中发现的问题只有 21%。”

此外，文章还谈到了将代码分析整合到工作流程的重要性，指出当他们通过静态分析工具自动运行提交的代码并邀请工程师查看分析结果时，很少有工程师跟进。但是，如果在编译过程中就能得到即时反馈，那么就会让更多人使用静态分析，且分析结果也更难被忽视。因此，Google 选择在每个人的工作流程中默认集成静态分析。他们认为要推广代码分析工具，开发人员必须感到能从中受益，并且喜欢使用这些工具。从中可以看出，编码标准确实对开发工作有影响。

6. 获得支持

获得技术支持可以让你的开发工具得到回报。
真正区别专业工具质量的是由世界各地的当地团队
提供的技术支持，因为他们讲的是客户的语言。

如果免费工具有问题，例如编译器或库中的错误，用户唯一能做的就是试着自己解决，或者在相关的资源库中发布一个问题。他们希望这个问题能被 GNU 社区修复，或者花钱请人修复或添加功能。这将给用户带来的真正成本（时间和/或金钱）是无法预测的。

避免整个开发团队因为开发工具的问题而停止工作，是利用 IAR 等供应商的专业开发工具的最大优势之一。这是因为 IAR 在世界各地都有当地的支持团队，提供方便的技术支持，涵盖若干语言，如英语、瑞典语、德语、韩语、日语、中文和阿拉伯语。客户可以指定发布时间和获得临时变通办法，从而继续专注于他们的应用。

IAR 将根据修复时间，花费合理的工作量解决错误，或通过变通办法/修正错误来降低错误的严重程度。IAR 认识到被定义为关键或严重的错误会给被许可人带来很大的不便，因此将尽其合理的最大努力，尽快修正错误，而不考虑这里定义的修复时间。

下页图表显示了常规支持和更新协议 (SUA) 客户的修复时间，IAR 的响应时间为一到两天，关键问题的修复时间不超过 15 个工作日。

怎样才能修复一个错误或增加一个功能？

有很多方法可以修复错误。下面的列表涵盖了大部分常见的情况。对于普通的 GCC 用户，也就是对 GCC 内部不熟悉的人来说，这些方法大致是按照难度递减的顺序排列的，而难度是以修复错误所需的时间来衡量的。没有哪种方法比其他方法更好；每种方法都有其优点和缺点。

- 自己修复。如果你足够努力的话，这种方法可能会带来结果，但可能会花费很多时间，而且，根据你的工作质量和你的修改所带来的好处，你的代码可能会也可能不会进入 GCC 的正式版本。
- 将问题报告给 GCC 的错误跟踪系统，并希望有人会好心地帮你解决这个问题。虽然这当然是可能的，而且经常发生，但不能保证一定会发生。你不应该指望从这种方法中得到与商业支持组织相同的回应，因为阅读 GCC 错误报告的人，如果他们选择帮助你，他们将自愿付出时间。
- 雇人帮你解决。有很多公司和个人为 GCC 提供支持。这种方法要花钱，但相对来说，有可能得到结果。

资料来源：https://gcc.gnu.org/faq.html#support

常规 SUA 内的响应和修复时间

资料来源：IAR

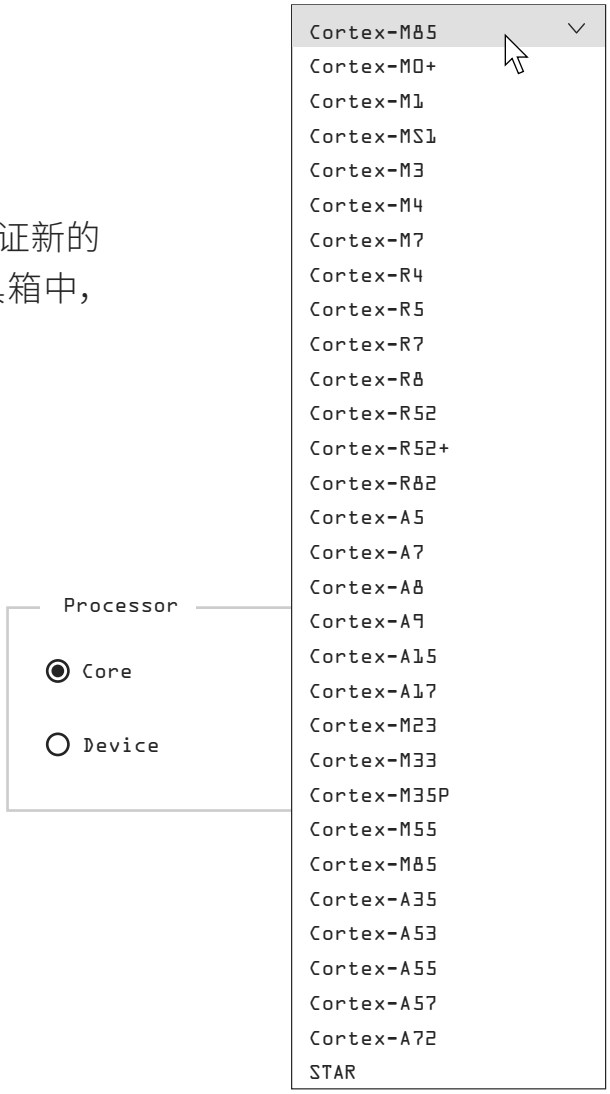
安全级别	响应时间	修复时间
关键	1 个工作日	不超过 15 个工作日
严重	1 个工作日	不超过 30 个工作日
中等	2 个工作日	在下一个预定的服务或功能发布时，但不晚于一年
轻微	2 个工作日	由 IAR 决定

根据 EMBECOSM 基于 GCC/LLVM 的案例研“How much does a compiler cost”可以很容易地计算出全包式支持和服务协议所带来的成本节约：工具链的维护通常需要每月 0.25 个工程师月的工作量。有一个经验法则，雇佣成本通常是工资的 1.25 到 1.4 倍，具体取决于福利、工资税和企业责任保险。考虑到在美国，一个编译器工程师每年平均花费为 11.7 万美元 x 1.4，相当于维护费用或每个需要修复的严重错误等于每月 3400 美元。

IAR Embedded Workbench 许可证包括 12 个月的支持和更新协议。因此，客户每年只需花费自己每月维护费用的 50% 就能保持 SUA 的有效性。更不用说，IAR 团队能够在一到两个工作日内提供变通办法，使得开发团队在任何时候都能保持生产力。

7.开发环境

开发团队的生产力依赖于精简的开发流程，保证新的芯片、中间件和扩展能无缝地集成到一个工具箱中，实现不间断的工作流程。



今天的电子设备需要逐渐增多的嵌入式系统，这些系统通常包括 8 位、16 位、32 位和 64 位应用的组合。同时，嵌入式应用正变得越来越复杂和智能。对具有更多差异化功能的新产品的渴求已经变得如此之大，以至于单个产品的上市时间可以成为整个公司成功的决定性因素。

IAR 的嵌入式开发工具支持 15000 多种芯片，涵盖来自 200 多个半导体合作伙伴的 8 位、16 位、32 位和 64 位 MCU/MPU，为全球约 100,000 名开发人员提供服务。这是芯片支持最广泛且功能最强大的生态系统。IAR Embedded Workbench 允许客户在一个统一的集成开发环境中，在所有主要供应商的处理器之间自由切换。当然，每个架构仍然需要一个单独的许可证。市场上最广泛的芯片支持是通过一个通用平台和不同目标的通用组件实现的。此外，IAR 增加了针对架构和处理器的适配和优化，让开发人员创建高效、稳定的代码，同时提高开发效率。

选择一个能提供集成开发环境的编译器是至关重要的，它能提高效率以缩短开发时间。同时这也是确保应用稳定一致的关键因素。对 IAR Embedded Workbench 这样的嵌入式开发工具进行标准化，可以为开发团队提供简化的开发流程，不间断的工作流程，以及所有组件无缝集成的一个工具箱。它还简化了开发，并实现了跨项目和处理器的代码重用，从而避免了生产中的任何延误。

为什么这很重要?开发团队同时使用不同的处理器是很常见的。他们需要能够选择最适合手头应用的处理器。如果他们由于某种原因需要切换处理器，他们不必从头开始，同时无需切换开发工具，这将受益匪浅。

这样的初始投资从长远来看回报会越来越可观：为了入门，开发人员至少需要一个工作周来学习一个新的 IDE 和工具链。像 IAR EMB 学院 中的培训课程，例如 "Getting Started with IAR Embedded Workbench" ，"Efficient Programming" ,"Advanced Debugging" ，可帮助开发人员充分利用该工具套件的功能，需要三个工作日完成。

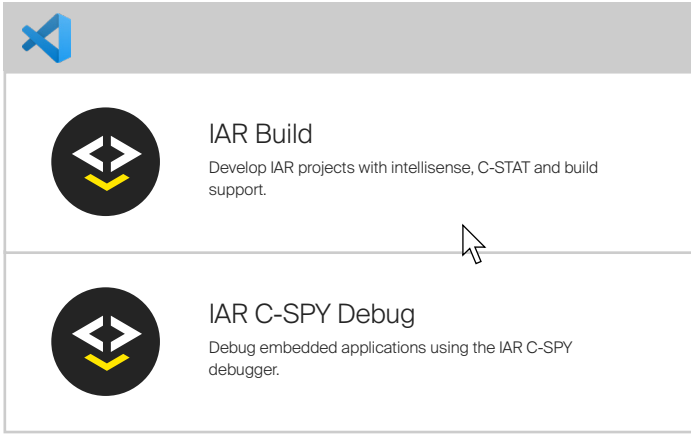
此外，开发人员还需要额外的两天时间来消化信息并自行尝试教程。鉴于美国嵌入式开发人员的平均工资为 10.4 万美元 x 1.4（工资成本包括福利、工资税和企业责任保险），这意味着每个开发人员需要花费近 2800 美元来上手使用新工具链。

另外，开发人员可能需要更长的时间来熟练使用新的工具链，这样才能保证应用的一致性和稳定性。但这些用于培训开发人员的一次性费用是值得的投资，因为后续无需为使用不同的处理器而再次投资。特别是当开发人员在使用熟悉的工具链时可不断积累经验和专业知识，从而进一步节省时间和成本。

现代开发工作流程也需要灵活性和与生态系统中合作伙伴的解决方案进行额外集成。IAR Visual Studio Code 扩展与所有最新版本的 IAR Embedded Workbench 和 IAR 构建工具兼容，并在 Visual Studio Code Marketplace 提供，使开发人员能够从 VS Code 构建和开发项目。这同样适用于 IAR Eclipse 插件，可以充分利用高质量的 IAR 构建工具链和高级功能。这些扩展可用于其他构建系统，如 Cmake、源代码控制和 GitHub 等版本扩展，以满足开发需求。

IAR Embedded Workbench 和 IAR 构建工具包括在前 12 个月获得专业的技术支持，以及访问 IAR 学院的按需课程，以便顺利开始并提高生产力。

一体化的集成 IDE 和扩展使程序员利用一个工具就能使用所有功能，而不需要不断切换工具。开发任务的紧密集成提高了开发人员的生产力和效率。



下载 IAR Embedded Workbench，针对不同的 MCU/MPU 和 VS Code 扩展来测试你的代码

下载 →

8. 合规与安全

使用经过认证的工具加速安全认证。功能安全在任何应用中都是非常想要实现的，对于某些应用来说，它是一个必须遵守的要求。如果你不选择使用预先经过认证的开发解决方案，构建具用确保功能安全的应用将会变得非常具用挑战性和耗时。

遵循安全认证有许多好处：它将降低与你的应用相关的责任风险，减少产品召回的几率和固件更新的数量。此外，除了保护贵公司的声誉和企业目标外，它还将确保符合国际标准和要求。

目前市面上有许多标准和安全认证。每一个都迎合了一个特定的行业或产品类别。两个影响最广泛的认证标准是 ISO 26262（道路车辆）和 IEC 61508（电子安全相关系统），IEC 61508 是功能安全认证的“伞”标准。大多数功能安全开发工具的目标是根据这两个标准进行认证，因为它们几乎涵盖了所有其他认证和行业。

特定的认证可能会超越这两个标准，但它们被认为是许多其他标准的基础，例如，用于铁路系统的 EN 50128 与 IEC 61508 相似。总的来说，所有的标准都提供了明确的流程来评估安全关键系统的风险并指定安全目标。此外，还涵盖了最佳实践开发流程的要求，以减少系统故障。最后，还有持续的程序以确保产品部署后的功能安全。简而言之：这些标准概述了如何识别和处理风险，并且所有这些标准都要求工具经过功能安全认证。

安全标准的广泛覆盖

工业
IEC 61508

机械控制
ISO 13849 IEC 62061

铁路
EN 50128 EN 50657

医疗
IEC 62304

农林
ISO 25119

汽车
ISO 26262

过程工业
IEC 61511

家用电器
IEC 60730

工具的功能安全认证意味着开发工具经过了严格的认证过程，以确保它在编译代码时产生可靠和可重复的结果。此外，它意味着工具的开发流程已经到位，以管理工具如何遵守不同的功能安全标准提出的具体要求，并且有工具的测试和质量措施，以显示符合不同语言标准的验证。

认证流程是相当严格的。IEC 61508 标准在第 7.4.4 条中详细说明了应如何对支持工具进行认证，但在如何对编译器进行认证方面却相当模糊。我们来看看第 7.4.4.10 条：

“所选的编程语言应该有一个经过适用性评估的翻译器，包括在适当的时候，根据国际或国家标准进行评估。”

这些规定和其他规定使你很难亲自去认证一个工具，并可能导致你需要做大量的工作来证明其适用性，甚至需要做更多的工作来说明为什么你认为你已经证明了适用性。当你试图达到越来越高的安全完整性等级 (SIL) 时，这只会变得更糟。

这个流程的一部分是运行一套验证测试套件，其中有成千上万的测试程序被编译，并将结果与预期结果进行比较。另一部分是标准一致性测试。这些测试都不是详尽的，但应该能发现一些问题。

安全验证的诀窍是确保所有已知的问题都被记录下来。功能安全意味着已知的不完善之处被清楚地记录下来，并且你有一个流程来识别和记录不完善之处。对于完整的验证，你必须修复或记录每一个与预期行为的偏差，并证明其合理性，实现没有已知的不合理的偏差。

验证你自己的工具链符合功能安全是昂贵和耗时的。工具认证可能需要长达 12 个月的时间，并占用几个员工，差不多需要两到四个。考虑到美国嵌入式开发人员的工资，根据额外的测试要求，估计成本可高达 14.5 万美元。实际数字将不可避免地取决于你项目的 SIL 等级。

请注意，如果你想重新使用在一个获得了认证的项目中使用过的未认证工具，那么你仍然需要证明你的新项目与以前的项目足够相似。你必须提供证据，证明你使用的工具链的功能与前一个项目相同，而这在没有源代码级别的访问时是不可能的。此外，你必须证明你使用工具链的方式与安全认证的方式相当。通常情况下，你最终可能不得不做同样的工作来重新认证该工具。

通过使用已经通过功能安全认证的开发工具，你不再需要证明你的工具链符合安全标准，你只需要认证你的应用。事实上，使用经过功能安全认证的工具链和编码标准可以节省大量的时间和金钱，因为它可以简化和加快应用认证流程。这也意味着软件开发生命周期 (SDLC) 的测试和修复阶段可以专注于寻找源代码中的错误，而不是怀疑是否是编译器问题导致的问题。

IAR 功能安全解决方案包括由 TÜV SÜD 认证的工具，涵盖10个不同的安全标准，并通过特殊的功能安全协议和协议期间的安全证书更新提供长期支持。

对于从事安全关键型软件的客户，IAR 通过功能安全 SUA 提供优先支持，响应时间仅为 1 个工作日，关键问题的修复时间不超过 10 个工作日：

功能安全 SUA 内的响应和修复时间

资料来源：IAR

安全级别	响应时间	修复时间
关键	1 个工作日	不超过 10 个工作日
严重	1 个工作日	不超过 20 个工作日
中等	1 个工作日	在下一个产品更新或升级时，但不晚于一年
轻微	1 个工作日	由 IAR 决定

9. 许可证

找到正确的许可证类型可以使投资回报率 (ROI) 最大化。许可证有许多使用情况，正确的许可证组合和管理有助于优化你的工具的支出。这一切都回到了组织或开发团队需要什么的问题上。

许可证类型

了解更多 →

单机版许可证

锁定
一台电脑

用户
单一开发人员

移动版许可证

锁定
USB 加密狗

用户
单一开发人员，多台电脑

网络版许可证

锁定
一个站点

用户
当地团队

全球版许可证

锁定
多个站点

用户
全球团队

IAR 提供灵活的许可证和定价选项，让公司的投资回报最大化。从单机版、移动版到网络版和全球版的许可证类型使许可证的管理变得简单（许可证池见上方）。

网络版许可证对于一个开发团队来说是方便和具有成本效益的。它允许一组用户在本地网络上共享一个许可证池。虽然对并发用户的数量有限制，但可以占用一个许可证的安装副本的数量是无限的。网络版许可证是由交付的许可证服务器管理的，它包括在交付中。新的用户可以被添加到现有的网络版许可证中。

对于在多个地点和不同国家有运营和开发项目的客户，IAR 通过全球版网络许可证提供了地域灵活性。普通网络版许可证仅限于一个单一的地理站点，而全球版许可证提供了让用户从全球多个站点访问同一全球版许可证的可能性。

移动版许可证是一种单用户许可证，允许你灵活地选择工作地点。它被锁定至一个 USB 加密狗，你可以带着它在任何地方的任何电脑上使用。即使电脑没有网络连接，它也能工作。将许可证放在加密狗上还可以保护你的许可证免受硬件故障的影响。

单机版许可证被锁定在一台特定的 PC 上。它是一个个人的、单用户的许可证，只能通过物理访问你的 PC 来使用。 即使 PC 没有连接到网络上，它也可以使用。

让我们来看看一个例子：假设一家公司有两个站点，有 30 个需要使用开发工具的开发人员，每个站点 15 个，项目期限为 2 年（每年有 20% 的支持和更新协议更新）。

单机版许可证价格可以是任何当地货币，并取决于架构（8 位、16 位、32 位和 64 位）。其他许可证类型有一个溢价成本，取决于灵活性。移动版许可证要多花 16%（1.16 美元），网络版许可证要多花 33%（1.33 美元），全球版许可证要多花 100%（2 美元）。

为所有开发者提供单机版许可证将花费 30 x 美元 = 30 美元（在移动版许可证的情况下为 34.8 美元）。如果公司转向网络版许可证，成本将是：10 个网络版许可证（推荐），覆盖每个站点的 15 个开发人员。网络版许可证只允许用于本地站点，因此每个站点的成本为 10 x（1.33 美元）= 13.3 美元，总计 26.6 美元，与单机版许可证相比，成本降低了约 13%。下一步是转移到全球版许可证，这将意味着两个站点共有 12 个许可证（建议使用，但如果站点没有重叠的工作时间，可以更少），成本为 12 x（2 美元） = 24 美元。这与网络版许可证相比，成本降低了 11%，与单机版许可证相比，成本降低了 25%。这个例子没有考虑管理和需要更换损坏的工作站或丢失的加密狗的问题。请注意，每年的支持和更新协议的 20% 也将遵循成本降低的百分比。

为你的企业找到合适的许可证类型可以产生重大影响，不仅可以减少 25% 的总许可证成本，而且还可以促进许可证的管理。

10. 总结

控制产品的开发时间对于控制成本和实现交付目标至关重要。这些目标可以通过使用旨在帮助工程师快速和有效地创建产品的工具和服务来实现。

在产品设计中，使用具有前期成本的商业工具与为降低使用特定芯片的门槛而提供的“免费”工具相比，可以成为保持进度和减少开发产品的总体成本的有效方法。公司和开发团队“购买”的是更快的上市时间，并确保交付高质量的产品。最后，这些用例为使用专业解决方案（如 IAR 解决方案）的开发人员提供了清晰的投资回报率（ROI）和总拥有成本（TCO）。

ROI 和 TCO 的总结

资料来源：IAR

用例	节约和改进	节约
为什么要关注代码体积和基准？	每个芯片 1 至 4 美元	每个产品系列 1 万至 4 万美元
应用的性能如何影响 BOM（物料清单）？	每个芯片 0.5 至 0.9 美元	每个产品系列 5000 到 9000 美元
缩短编译时间，提高生产力	通过减少高达 50% 的构建时间来提高生产力 (Linux)	在云服务中节省高达 50% 的实例/时间
缩短调试时间以加快上市时间	释放人时约为 1000 小时	每年 2.4 万美元
缺陷的代价以及为什么代码质量很重要	每 1000 LOC 1900 美元	每 1000 至 5000 LOC 1.9 万至 9.5 万美元
获得技术支持可以让你的开发工具得到回报	每月 2400 美元	每个项目 4.3 万美元
在一个 IDE 中提供最广泛的芯片支持，提高开发人员的生产力	每个开发人员 2000 美元	每个项目 1 万美元
使用经过认证的工具加快安全认证之路	每个项目最多 14.5 万美元	每个项目 14.5 万美元
找到适合你的许可证模式	最大化使用并确保投资，具体取决于开发人员的数量	节省高达 25% 的总许可费用

如需了解有关如何提高开发人员效率的持续建议，请关注微信公众号“IAR 爱亚系统”。

作者

作者

Rafael Taubinger
高级产品营销经理

撰稿人

Anders Holmberg
首席技术官

David Källberg
FAE 经理（欧洲、中东和非洲）

Shawn Prestridge
FAE 经理（美国）

Hyun-Do Lee
销售经理

iar.com

Tomorrows intelligence
delivered today

联系我们

爱亚系统开发（上海）有限公司

IAR上海办公室
电话：021-6375 8658
邮箱：marketing.cn@iar.com
地址：上海市静安区南京西路1717号会德丰国际广场801A

IAR深圳办公室
电话：0755-3304 3249
邮箱：marketing.cn@iar.com
地址：深圳市福田区中心西路1号嘉里建设广场2座15楼