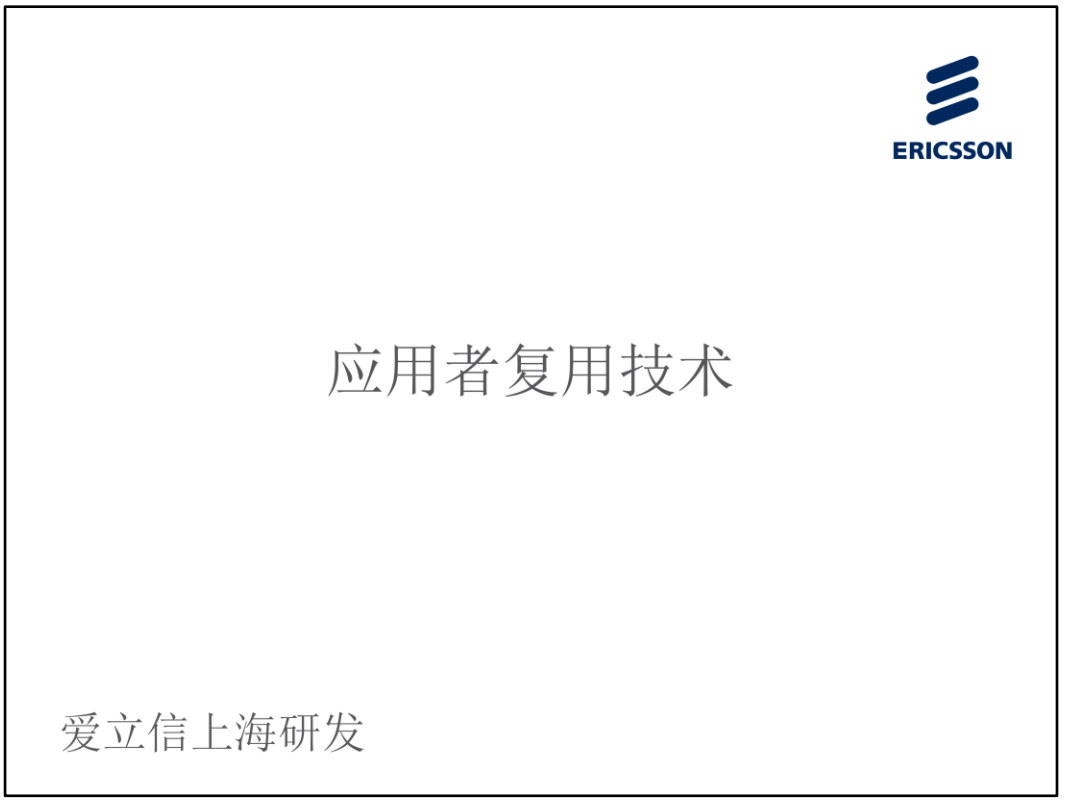

ERICSSON
应用者复用技术
爱立信上海研发

软件业的挑战

更快：满足市场时间要求

更好：满足功能，故障少，提供支持

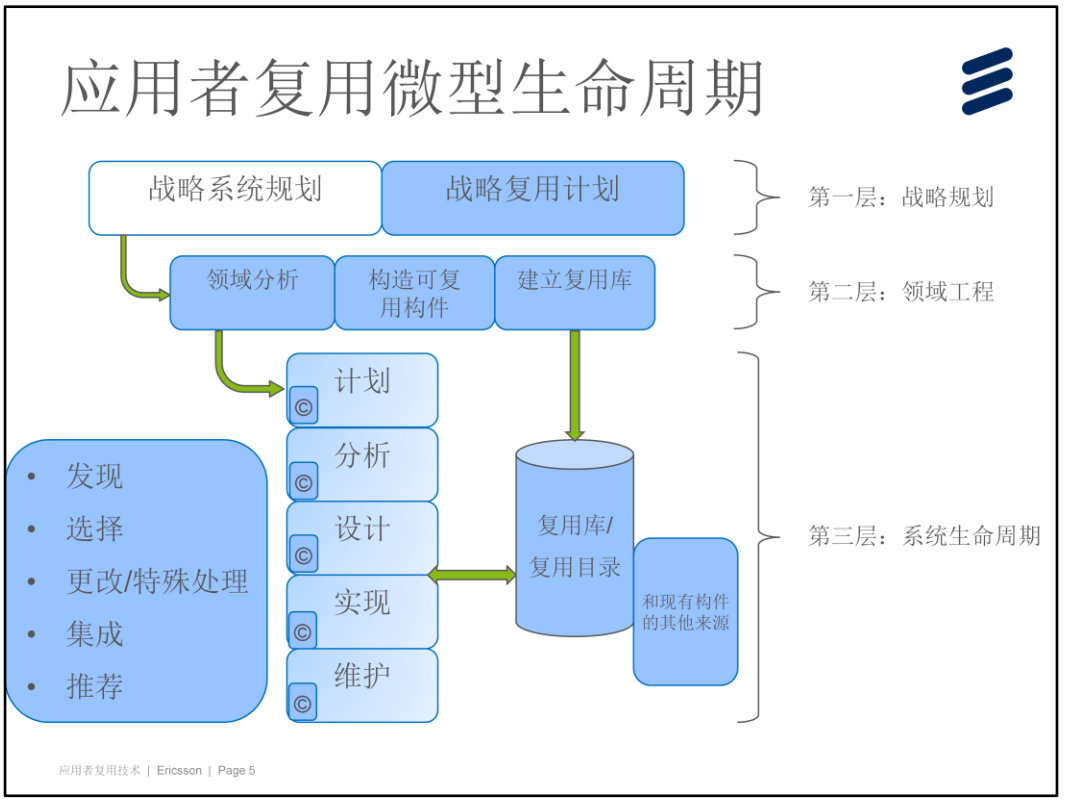
更便宜：生产和维护成本低

引入复用

- 最安全和最经济的引入复用的方式是采取一种渐进式的方法，在该方法中复用被分阶段地引入到公司中
  - 减少文化上的冲击
  - 减少对复用观念的抵制
  - 使复用失败的风险和它带来的任何负面影响降到最低程度

应用者复用

- 应用者复用是复用的一个方面，它考虑如何利用复用来建立系统；也就是说，如何复用现有的构件来建立新系统。
- 分阶段地引入复用的方法是首先引入应用者复用，期望项目团队去使用可以得到的可复用构件，而不是期望他们去建立新的可复用构件
  - 更加容易实施
  - 不会延长项目的工期
  - 在较短时间内见到复用的效益，证明复用的价值，减少失败的风险



- 寻找候选的可复用构件，由它们来产生生命周期每一阶段的交付
- 对候选构件进行评价，选择那些适合于在本系统内复用的构件
- 更改可复用构件或者对它们进行特殊处理，使之能满足本系统的要求
- 将可复用构件集成到本阶段将要产生的系统交付中
- 为可复用构件提出一些更改或者改进之处，增强它们未来的可复用性

应用者复用技术

- 选择应用包
- 选择可复用构件
- 冗余检查
- 标识候选的可复用构件


ERICSSON
选择应用包

应用包

- 应用包是预先定义的，常常已被验证过的，并为商务活动需要的解决方案
- 包是软件复用的特殊情况，即作为一个整体被复用
  - 包的部分价值体现在它包含了商务规则和实践
  - 包采用能被公司复用的形式吸取了专家的意见，知识和经验

选择应用包策略

- 根据将建立或获得的新系统的需求，可以选择应用包作为提供该解决方案可能采取的策略
  - 定义系统解决方案的需求
  - 确定应用包对该系统解决方案是否是一个可交付的策略

评估应用包

- 识别满足系统需求的候选应用包并对它们进行评估
  - 对系统解决方案需求的“适宜度”
  - 技术需求
  - 如何修改才能满足该系统解决方案当前和将来预期的需求
  - 质量：测试用例，模型，文档，代码，设计
  - 维护的工作量
  - 供应商：信誉，价格，将来的计划
  - 对用户的影响

选择应用包的商业考量

- 根据成本预算和预期的商业效益决定是否购买该软件包
  - 购买包的价格和维护的许可证
  - 购买和操作该包需要的所有附加工具的成本
  - 客户化该包的成本
  - 帮助安装和客户化该包的咨询服务成本
  - 培训成本和学习使用该包的时间
  - 建立或修改该包的用户文档的成本
  - 与你的信息和技术体系的其余部分集成的成本


ERICSSON
选择可复用构件

可复用构件

- 可以被复用的软件成分称为可复用构件(reusable component)
  - 它可以是从旧的软件中提取的，也可以是专门为了复用而开发的。
  - 一个软件构件只有在多个系统中能够被使用才称得上名副其实的可复用构件。
- 无论对可复用构件原封不动地使用还是作适当的修改后再使用，都可称为复用
  - 改善系统的质量
  - 降低系统开发和维护的成本
  - 减少项目失败的风险

选择可复用构件

- 寻找和选择合适的可复用构件，它们能被用于开发各种可交付的系统项目
  - 一部分是系统的构件，诸如设计和代码的构件
  - 另一部分是有关的项目管理和文档
- 关键问题
  - 必须有可以复用的对象
  - 要复用的对象必须是有用的，高质量的
  - 需要知道如何去使用被复用对象

规程

- 确定检索的可复用构件的类型
- 确定对在项目中使用的可复用构件的进行检索的来源
- 寻找建立该构件可能复用的现存系统
- 检索复用库和复用目录中能用于该项目的可复用构件
  - 应用模板，类库
  - 系统体系结构
  - 设计级构件
  - 实用功能
  - 文档构件
  - 测试数据，脚本和用例

考虑

- 需要特殊或附加的工具支持?
- 给项目或系统带来限制?
- 优化该项目在后续阶段的复用?
- 需要特殊的开发者或用户培训或咨询支持?
- 需要修改?
- 已经复用了多少次?
- 节省什么?

风险与困难

- 在同一系统中采用多个开发商所提供的构件而引入的兼容性问题
- 采用随处可见的构件可能会使开发出的软件产品丧失技术上的独创性和市场上的竞争力
- 第三方的构件开发商可能停业，这会使购买的构件失去维护服务


ERICSSON
冗余检查

冗余

- 当多个软件构件提供相同的功能，或者服务于同样的目的，或者定义同样的数据时，在一个软件系统中或在一组软件系统中就出现了冗余。
- 不必要的冗余会降低系统的质量和性能，引起更高的开发成本和开发周期。
- 冗余，特别是在大系统或产品家族中，不是有意出现的，因为系统的开发者不可能详细地了解由其他系统开发者完成工作的细节。

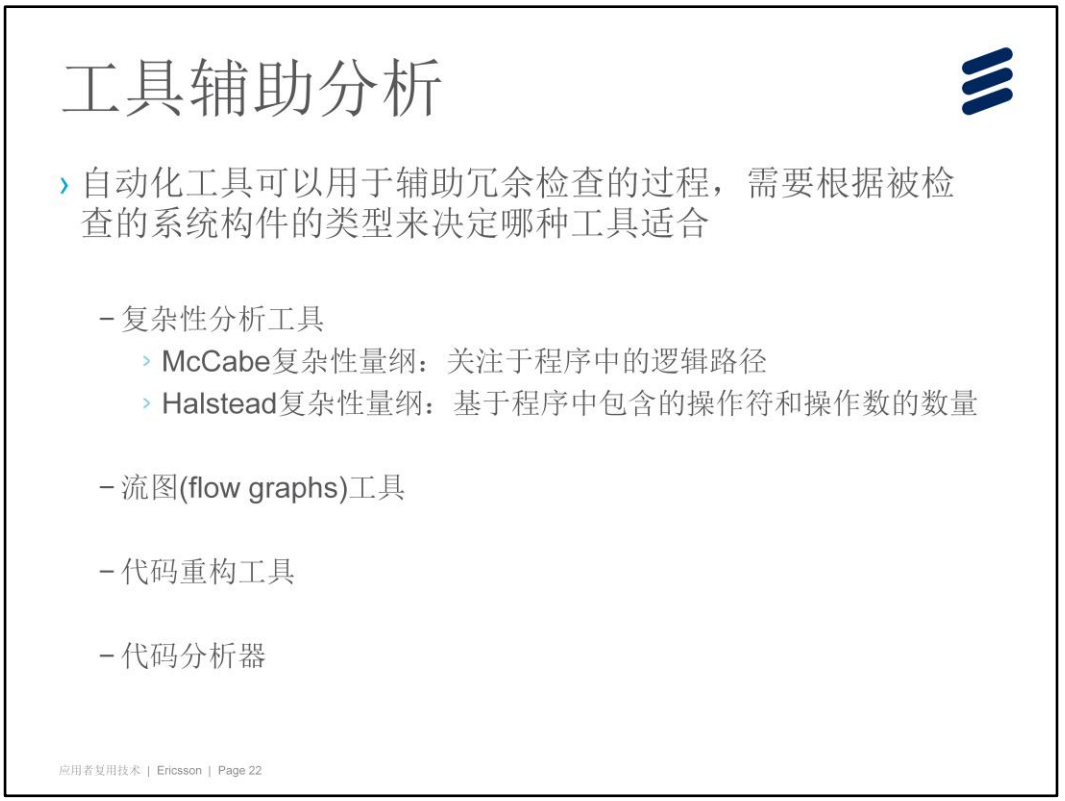
冗余

- 冗余可能出现在系统抽象的所有级别上
  - 分析：测试用例冗余性
  - 设计：类描述，逻辑结构，过程，数据模型
  - 代码
- 在代码级之前识别冗余更容易，那时构件的相似性没有隐藏在实现的细节中。
- 尽早的识别冗余可以避免在后期不必要的工作

冗余检查

- 共性分析在系统开发的各个阶段进行。
- 当冗余被发现并且可以被消除时，将创建一个通用的构件，用于取代所有冗余的构件。
- 冗余构件常常不能和另一个构件100%的匹配，必须能够识别出能由一个适当的通用构件取代的相似的和相同的结构。

生产者复用是在项目开始前分析复用以减少冗余
冗余检查是在项目结束后分析复用以减少冗余



McCabe度量法是一种基于程序控制流的复杂性度量方法。McCabe复杂性度量又称环路度量。它认为程序的复杂性很大程度上取决于程序图的复杂性。这种方法以图论为工具，先画出程序图，然后用该图的环路数作为程序复杂性的度量值。把程序流程图的每一个处理符号都退化成一个结点，原来连接不同处理符号的流线变成连接不同结点的有向弧，这样得到的有向图就叫做程序图。

根据图论，在一个强连通的有向图G中，环的个数V(G)由以下公式给出：
V(G)=m-n+2

其中，V(G)是有向图G中环路数，m是图G中弧数，n是图G中结点数，这样就可以使用上式计算环路复杂性了。

Halstead复杂度是以程序中出现的操作符和操作数为计数对象，以它们出现的次数作为计数目标（直接测量指标），然后据以计算出程序容量、工作量。

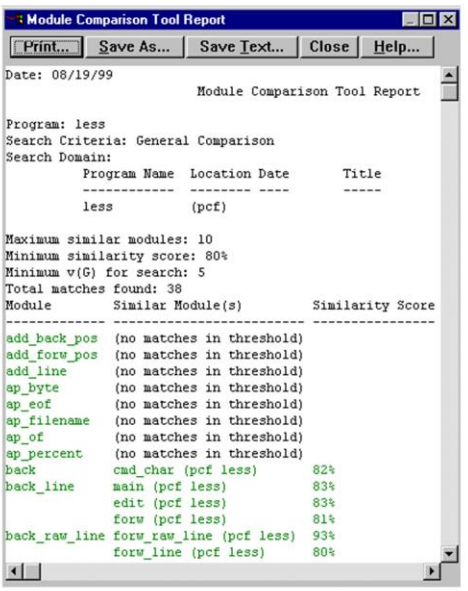
Machine Discovery of Static Software Reuse Potential Metrics (1994)

http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.29.3630

MCCABE COMPARE

McCabe Compare identifies reusable and redundant code

- *Simplify maintenance and re-engineering of applications through the consolidation of similar code modules*
- *Search for software defects in similar code modules, to make sure they're fixed consistently throughout the software*



https://www.rivier.edu/faculty/vriabov/CS699_McCabe_Overview.ppt

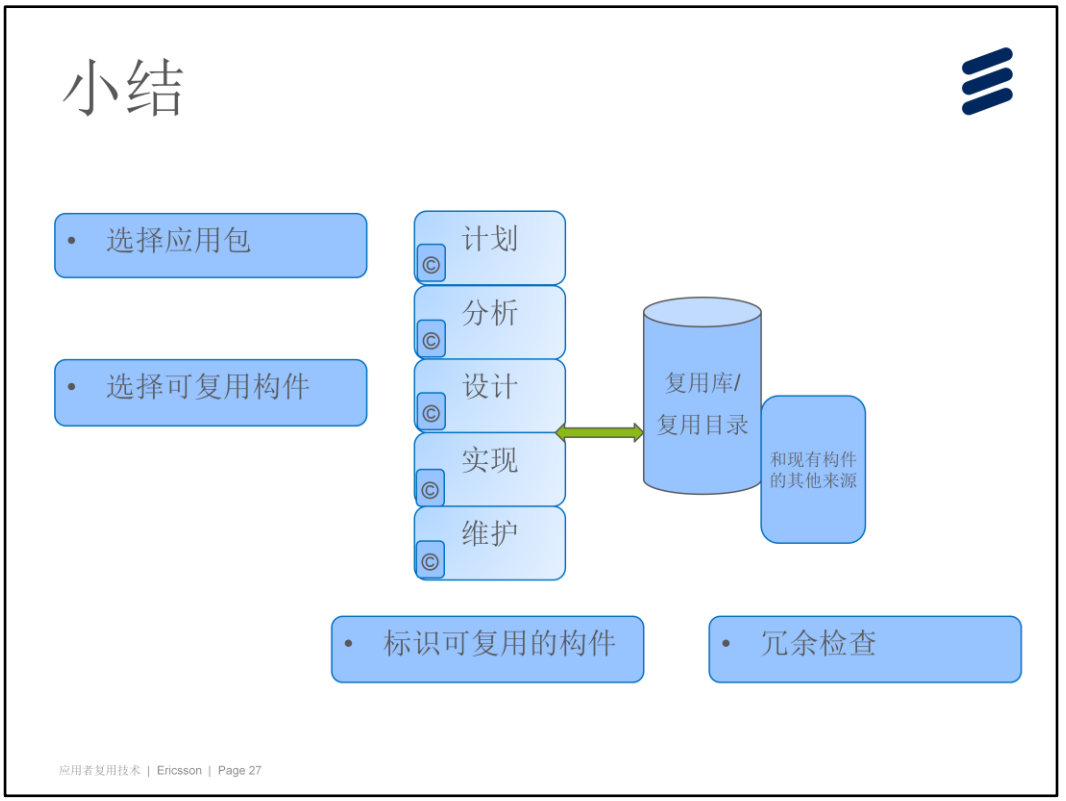

ERICSSON
标识候选的可复用构件

标识候选的可复用构件

- 用于识别在项目期间建立的可复用构件
- 确保在项目中，使有高可能被复用的构件能用复用的思想开发
  - 在构件初始设计和构造时就建立复用，比在已经建立后再试图使它适用于复用要好
- 标识那些复用可能性高的新构件，并交给另一个组开发，是一种较常见的实施

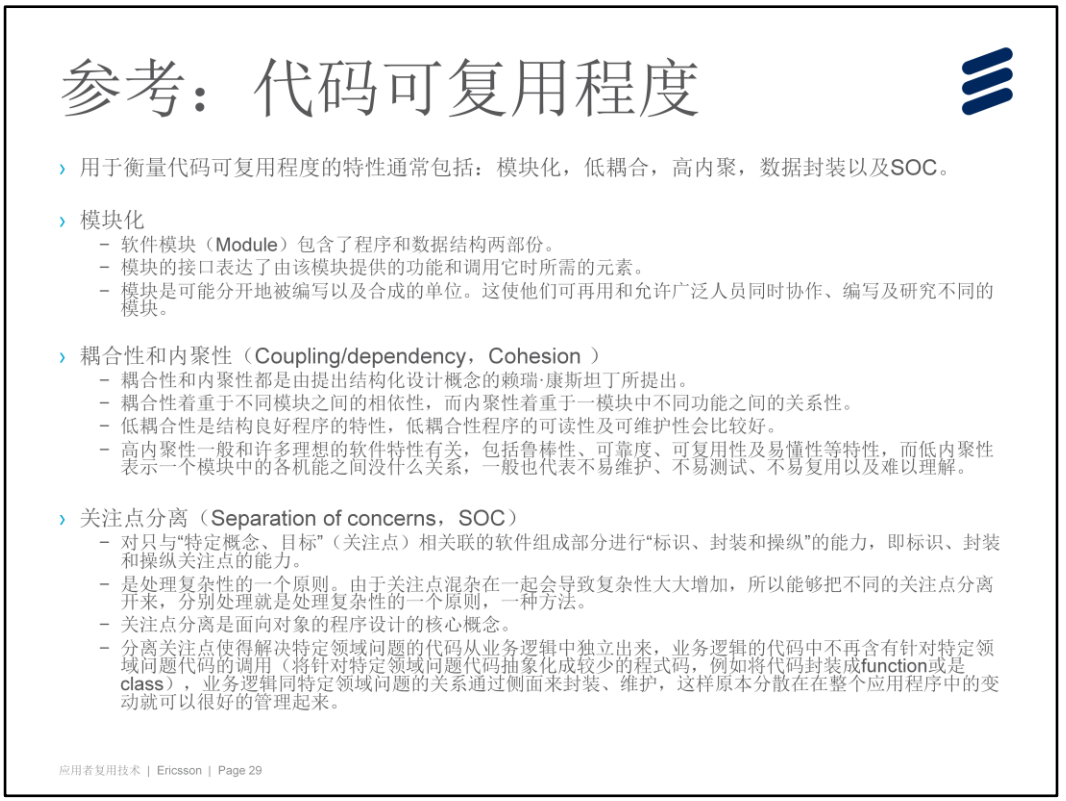
精选复用

- 建立和支持复用构件几乎总要增加相应的时间和成本（多至10倍），所以要精选复用。
- 选出那些对将来复用有最高可能性的构件作为可复用的构件开发
  - 最频繁被复用的
  - 能在多个系统中被复用的
  - 对于商务策略最重要的
  - 最可能收回复用投资的
  - 有已知需求的



- 选择应用包直接完成系统
  - 选择可复用构件的一个特例
- 在生命周期每个阶段开始前选择可复用构件
- 在任何阶段，标识可复用的构件
- 每个阶段结束时，进行冗余检查
  - 标识可复用的构件的特例，即通过冗余来标识/分析


ERICSSON



内聚性是一种非量化的量测，可利用评量规准来确认待确认源代码的内聚性的分类。内聚性的分类如下，由低到高排列：

- 偶然内聚性（Coincidental cohesion，最低）

偶然内聚性是指模块中的机能只是刚好放在一起，模块中各机能之间唯一的关系是其位在同一个模块中（例如：“工具”模块）。

- 逻辑内聚性（Logical cohesion）

逻辑内聚性是只要机能只要在逻辑上分为同一类，不论各机能的本质是否有很大差异，就将这些机能放在同一模块中（例如将所有的鼠标和键盘都放在输入处理副程序中）。

- 时间性内聚性（Temporal cohesion）

时间性内聚性是指将相近时间点运行的程序，放在同一个模块中（例如在捕捉到一个异常后调用一函数，在函数中关闭已打开的文件、产生错误日志、并告知用户）。

- 程序内聚性（Procedural cohesion）

程序内聚性是指依一组会依照固定顺序运行的程序放在同一个模块中（例如一个函数检查文件的权限，之后打开文件）。

- 联系内聚性（Communicational cohesion）

联系内聚性是指模块中的机能因为处理相同的数据，因此放在同一个模块中（例如一个模块中的许多机能都访问同一个记录）。

- 依序内聚性（Sequential cohesion）

依序内聚性是指模块中的各机能彼此的输入及输出数据相关，一模块的输出数据是另一个模块的输入，类似工厂的生产线（例如一个模块先读取文件中的数据，之后再处理数据）。

- 功能内聚性（Functional cohesion，最高）

功能内聚性是指模块中的各机能是因为它们都对模块中单一明确定义的任务有贡献（例如XML字符串的词法分析）。

应用者复用的软件开发过程