CPP-Summit 2019

全球C++软件技术大会

C++ Development Technology Summit

Boolan

高端IT互联网教育平台

关注“博览Boolan”服务号

发现更多 会议·课程·活动

CPP-Summit 2019

温　昱
代码重构|架构优化|首席顾问

C/C++嵌入式软件质量守护与重构

议程

1

咨询心得分享

分享点：嵌入式软件重构的几点经验

2

如何诊断深、诊断准

分享点：做到两快三深

3

代码质量守护

分享点：超越圈复杂度 的 路径畅度 守护

4

可插拔，应对需求变更 和 型号差异

分享点：编译期可插拔 与 运行时可插拔

案例心得分享 10分钟

议程

1 **咨询心得分享**

分享点：嵌入式软件重构的几点经验

2 **如何诊断深、诊断准**

分享点：做到两快三深

3 **代码质量守护**

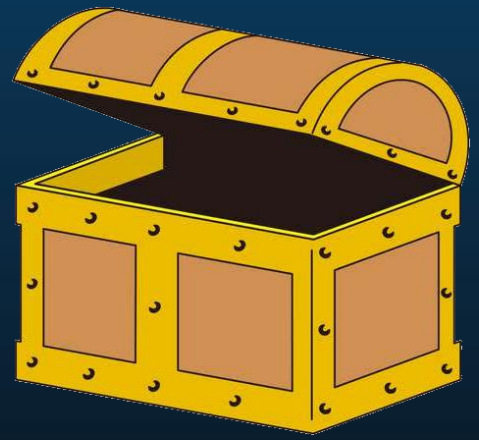
分享点：超越圈复杂度 的 路径畅度 守护

4 **可插拔，应对需求变更 和 型号差异**

分享点：编译期可插拔 与 运行时可插拔

企业问：
最熟代码的 是我们自己
但如何诊断深、诊断准

【两快三深】



1.四类需求变更的 快速分析

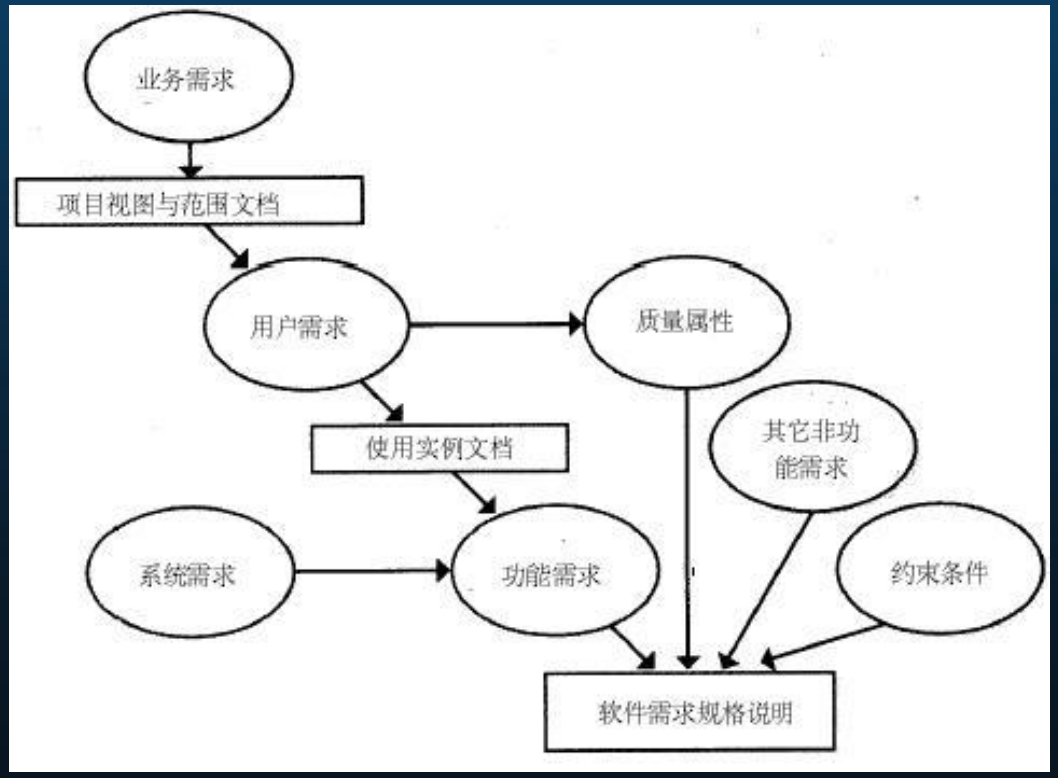
2.模块化及分层的 快速分析

3.任务任务间通信 重点诊断

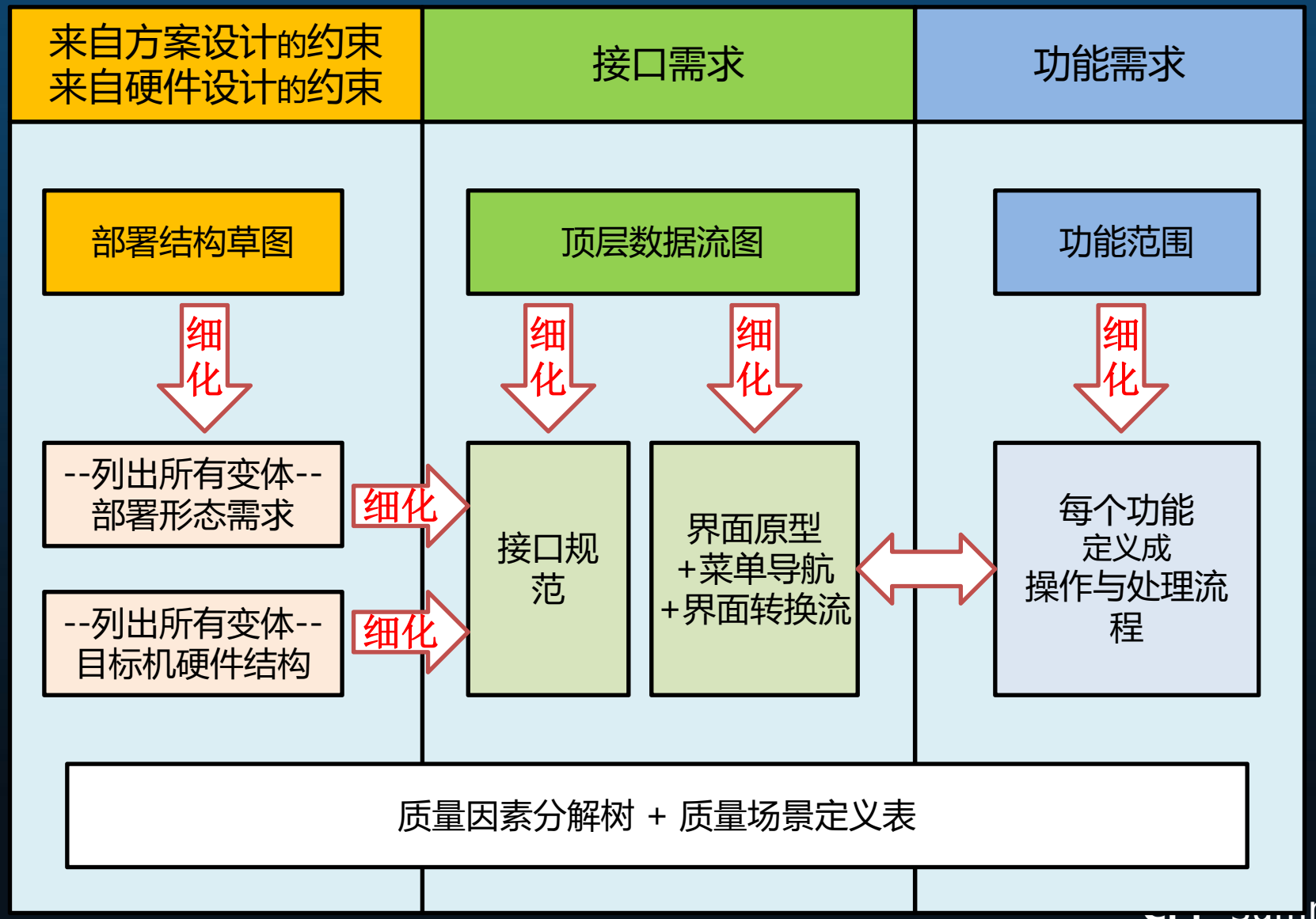
4.模块信息隐藏度 重点诊断

5.语句路径畅度的 重点诊断

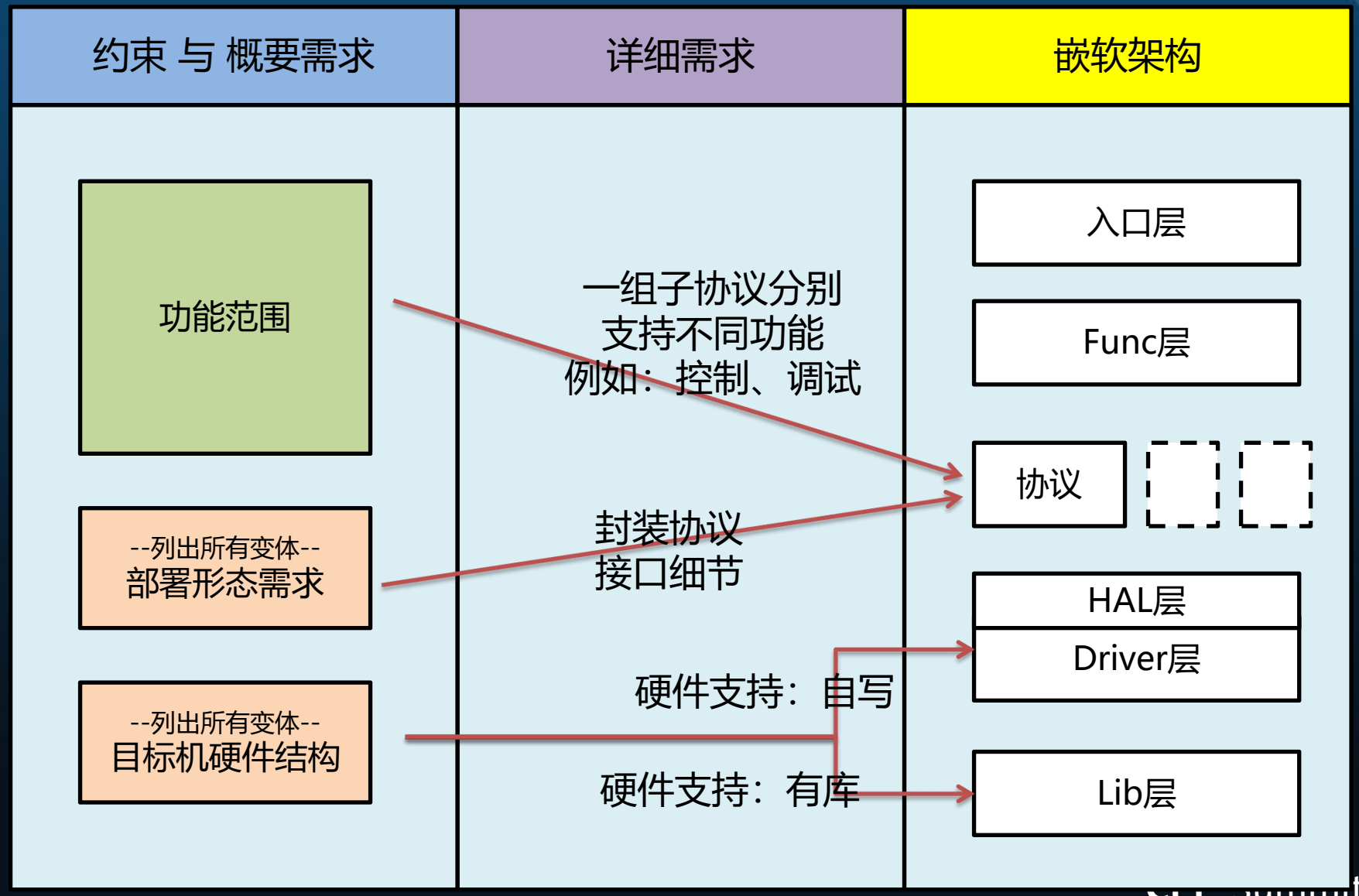
诊断——变更分析：老掉牙了


业务需求
项目视图与范围文档
用户需求
质量属性
使用实例文档
其它非功能需求
系统需求
功能需求
约束条件
软件需求规格说明

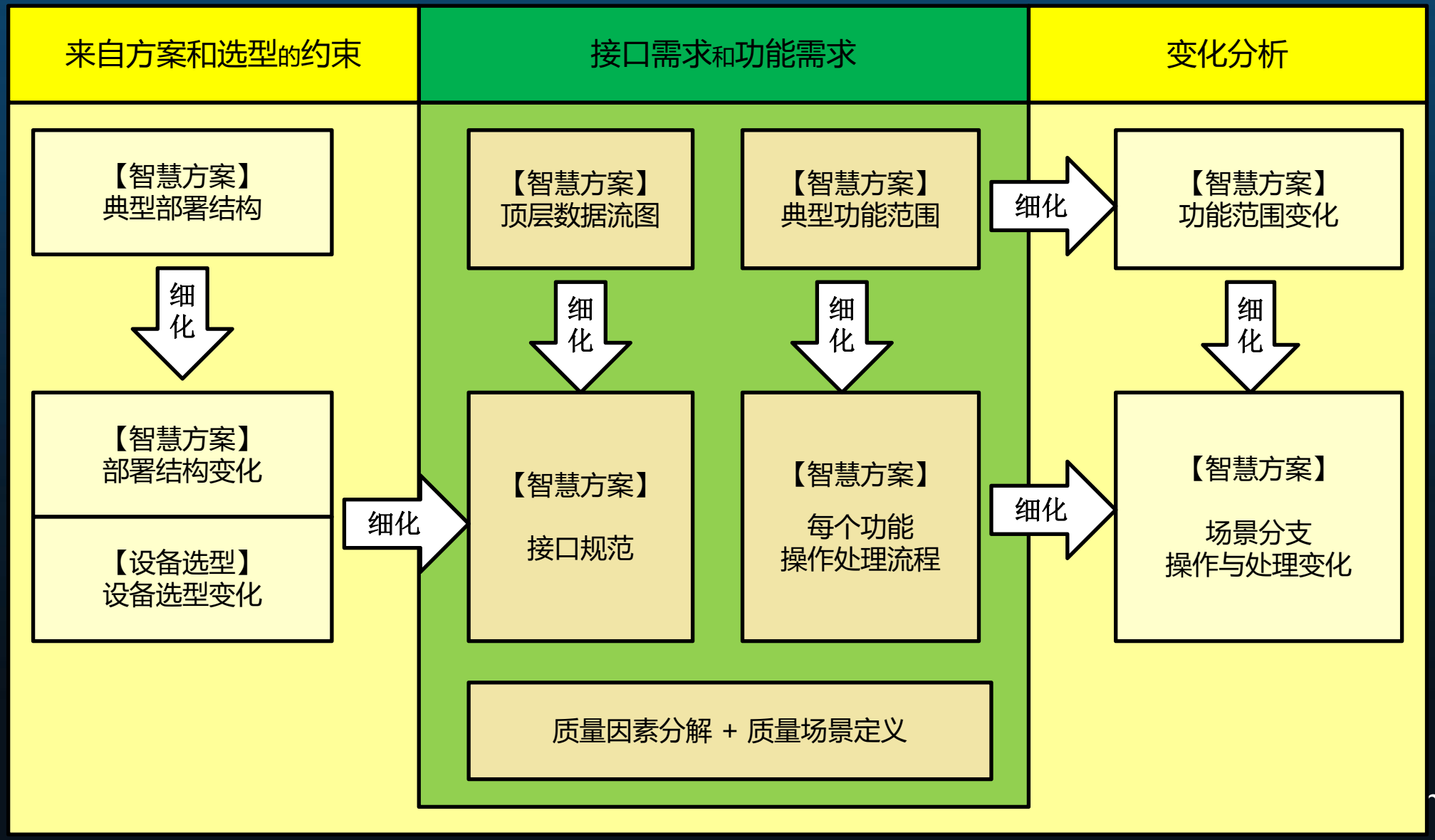
诊断——变更分析：嵌软人


来自方案设计的约束
来自硬件设计的约束
接口需求
功能需求
部署结构草图
顶层数据流图
功能范围
细化
细化
细化
细化
--列出所有变体--
部署形态需求
--列出所有变体--
目标机硬件结构
细化
细化
接口规范
界面原型
+菜单导航
+界面转换流
每个功能
定义成
操作与处理流程
质量因素分解树 + 质量场景定义表

诊断——变更分析：嵌软人


约束 与 概要需求
详细需求
嵌软架构
功能范围
--列出所有变体--
部署形态需求
--列出所有变体--
目标机硬件结构
一组子协议分别
支持不同功能
例如：控制、调试
封装协议
接口细节
硬件支持：自写
硬件支持：有库
入口层
Func层
协议
HAL层
Driver层
Lib层

诊断——嵌软需求变更分析法



诊断——变更分析：案例A

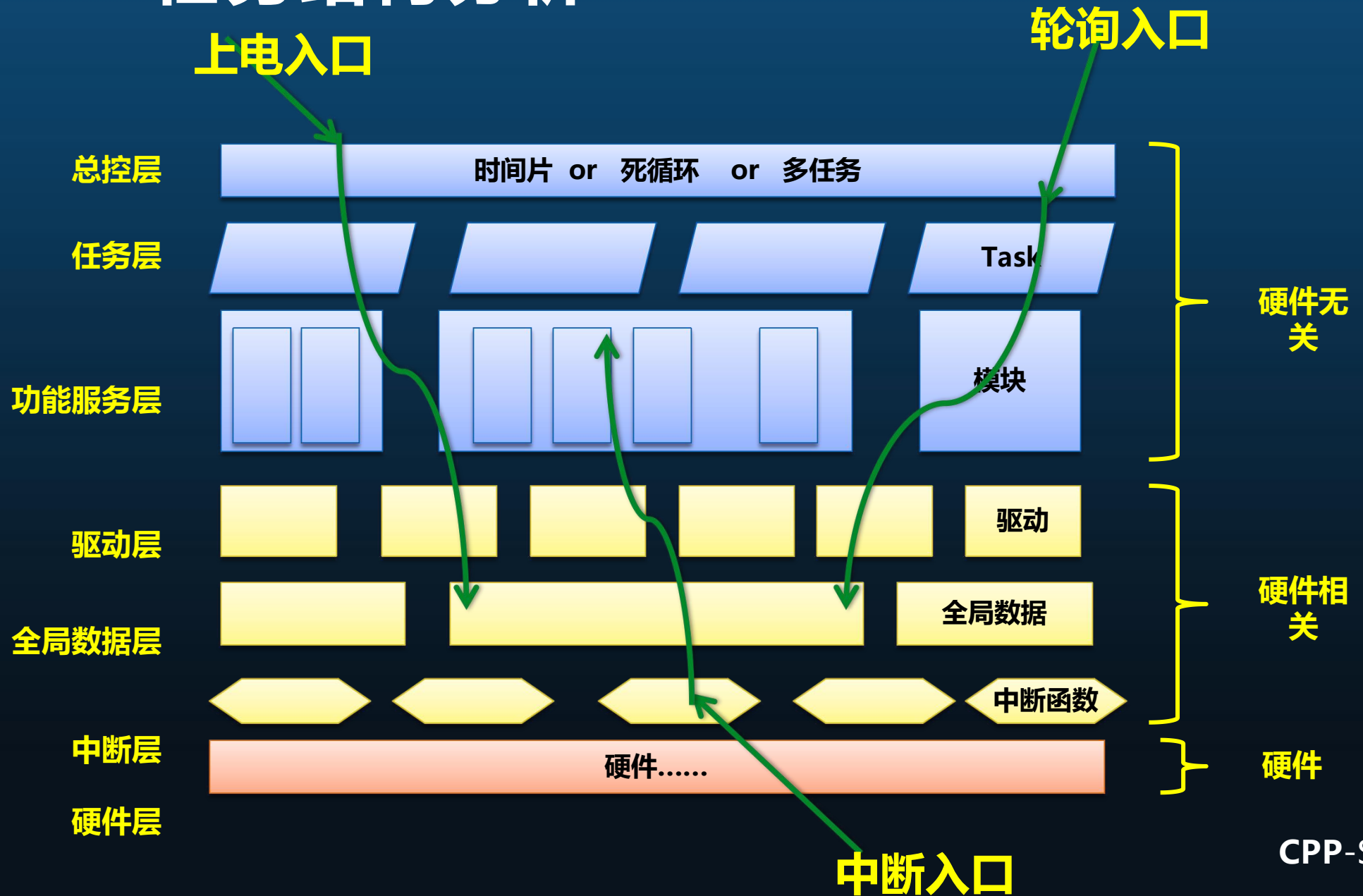
诊断——变更分析：案例A

诊断——变更分析：案例B

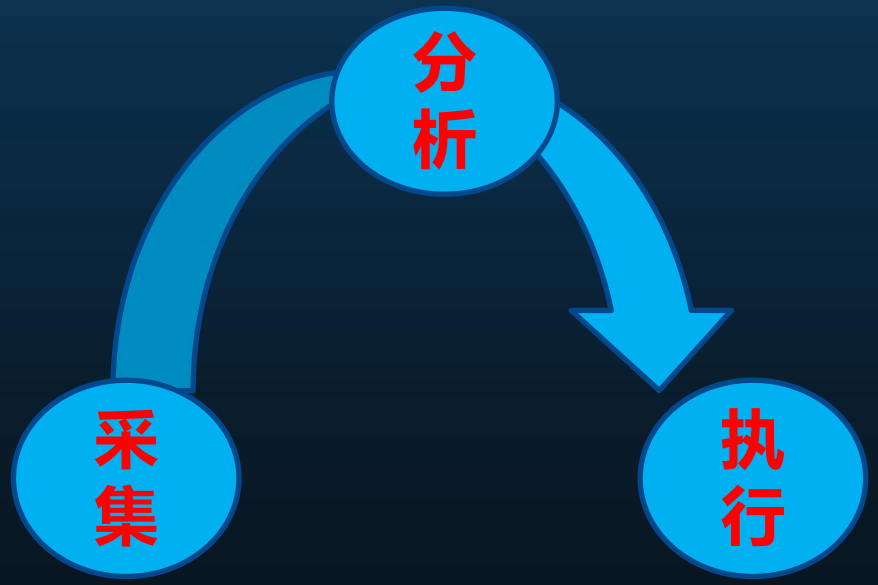
	S6503	S6506	S6506R
GEPON特性	支持802.3ah；支持 1 :32光分路比；支持DBA,DBA带宽分配粒度为1Mbps；20km		
最大ONU数	768路	1408路	1408路(业界集成度最高)
背板带宽	640Gbps	1.6Tbps	1.6Tbps(可扩展性强)
交换容量	96Gbps	384Gbps	768Gbps
包转发率	72Mpps	216Mpps	432Mpps
槽位数量	4	7	8（双主控引擎）
I/O槽位数	3	6	6
端口密度	24 GEPON端口	48 GEPON端口	48 GEPON端口
	3 10GE	6 10GE	6 10GE
	144 GE	288 GE	288 GE
	144 FE	288 FE	288 FE
最长匹配路由	64K	64K	64K
Vlan个数	4K	4K	4K
Vlan路由接口	1K	1K	1K
MAC地址	32K	64K	64K
丰富L2/3协议	802.3x,802.1p,802.1q,IGMP Snooping,PIM-DM/SM,STP,802.1x,BGP,OSFP,IS-IS,RIP,Vlan trunk,802.1x Server,DHCP relay/server		

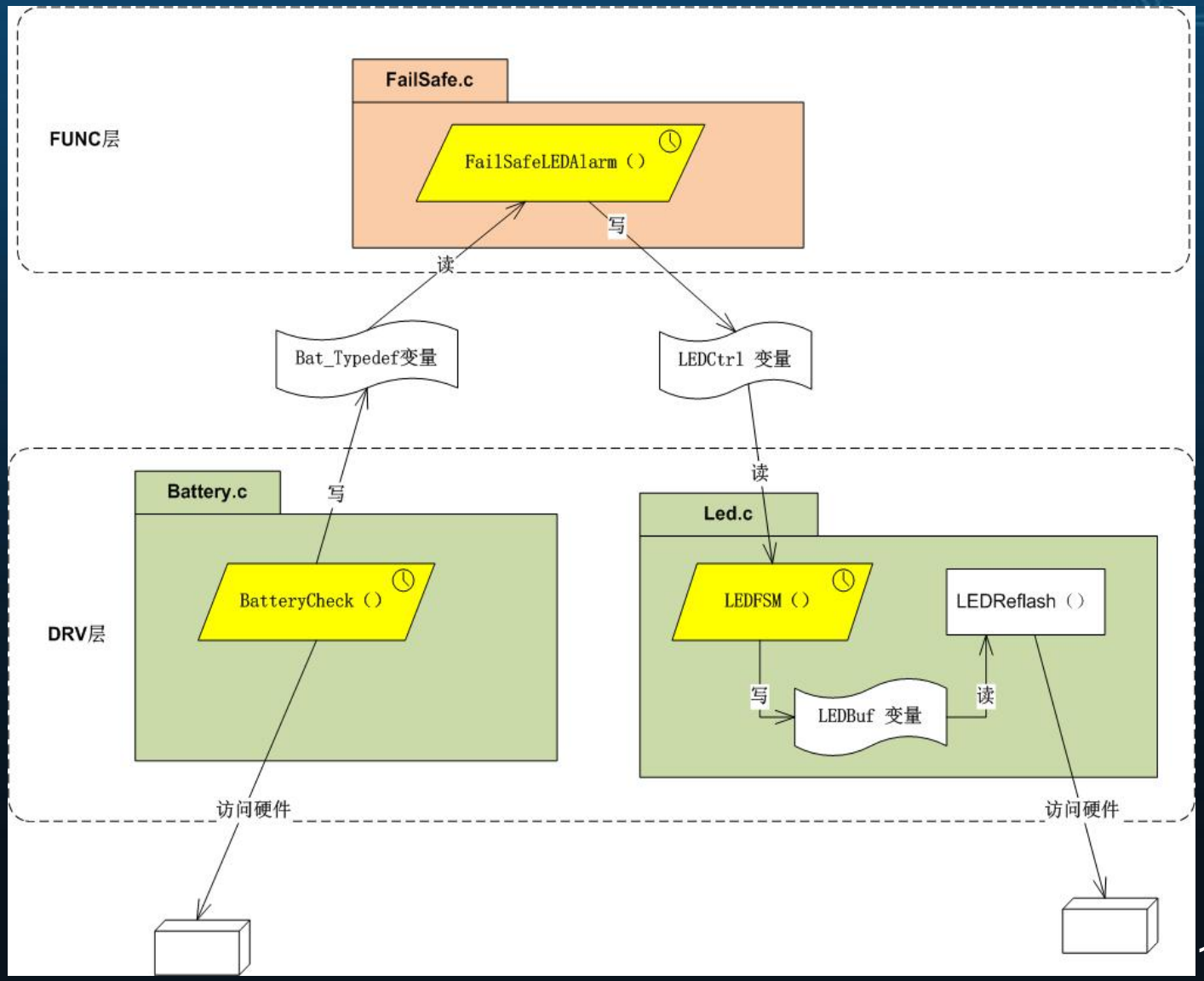
诊断——变更分析：案例B

诊断——任务结构分析


上电入口
轮询入口
总控层
时间片 or 死循环 or 多任务
任务层
Task
功能服务层
模块
驱动层
驱动
全局数据层
全局数据
中断层
中断函数
硬件层
硬件……
硬件无关
硬件相关
硬件
中断入口

诊断——任务结构分析：案例


分析
采集
执行


FUNC层
FailSafe.c
FailSafeLEDAlarm ()
写
读
Bat_Typedef变量
LEDCtrl 变量
读
写
Battery.c
Led.c
DRV层
BatteryCheck ()
LEDFSM ()
LEDReflash ()
写
LEDBuf 变量
读
访问硬件
访问硬件

议程

1 **咨询心得分享**

分享点：嵌入式软件重构的几点经验

2 **如何诊断深、诊断准**

分享点：做到两快三深

3 **代码质量守护**

分享点：超越圈复杂度 的 路径畅度 守护

4 **可插拔，应对需求变更 和 型号差异**

分享点：编译期可插拔 与 运行时可插拔

企业问：
代码规范早就有
为何还是高CC

定理1

拆分 = 分散复杂性

改写 = 减小复杂性

推论：

不是：拆分不行才想改写

而是：逻辑改写简化为主

定理2

模块与接口抽象不良

则局部逻辑优化无效

推论：

不是：闻气味 小步改 不停改

而是：诊断清 设计明 小步走

我很奇怪 为什么
80%的工程师迷信拆分

既不符：定理1

也不符：定理2

代码守护——设计规范>代码规范 例子

代码守护——设计规范>代码规范 例子

代码守护——设计规范>代码规范 例子

代码守护——设计规范>代码规范 例子

首选改设计 

设计没问题时，局部改代码

原来如此，一语惊醒梦中人！ 

议程

1 **咨询心得分享**

分享点：嵌入式软件重构的几点经验

2 **如何诊断深、诊断准**

分享点：做到两快三深

3 **代码质量守护**

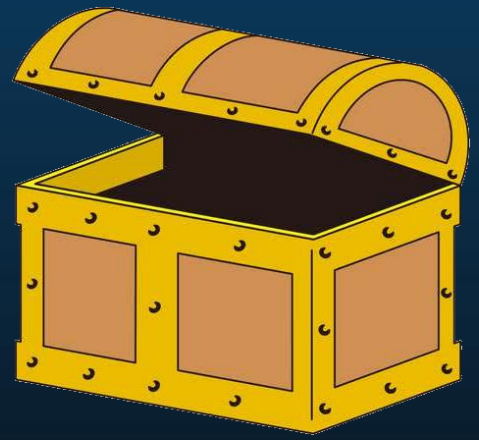
分享点：超越圈复杂度 的 路径畅度 守护

4 **可插拔，应对需求变更 和 型号差异**

分享点：编译期可插拔 与 运行时可插拔

企业问：
很烦紧耦合 如何可插拔
应对需求变更与型号差异

【嵌软好架构】



先有：

任务间数据流清楚

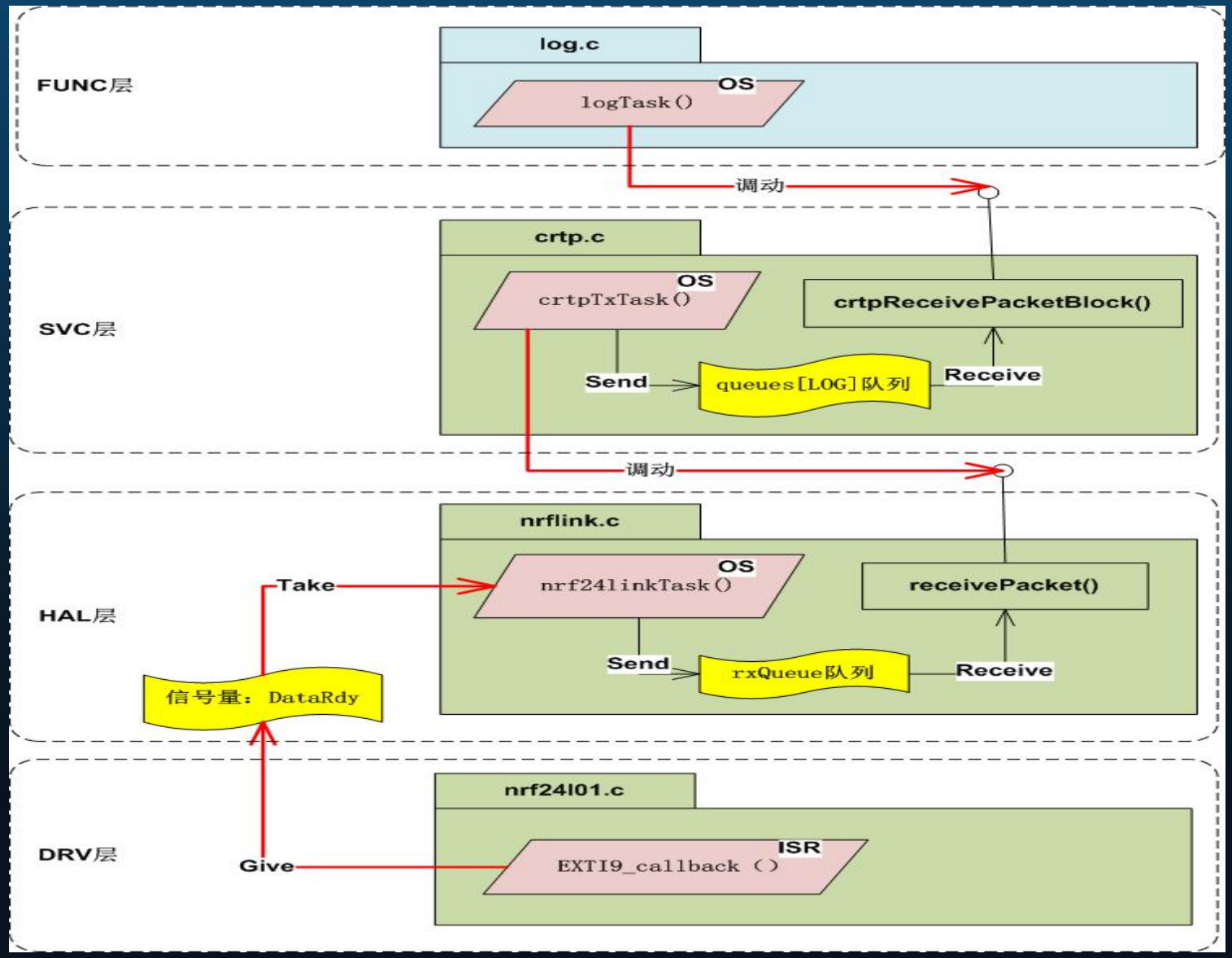
后有：

编译期(真)可插拔

运行时(真)可插拔

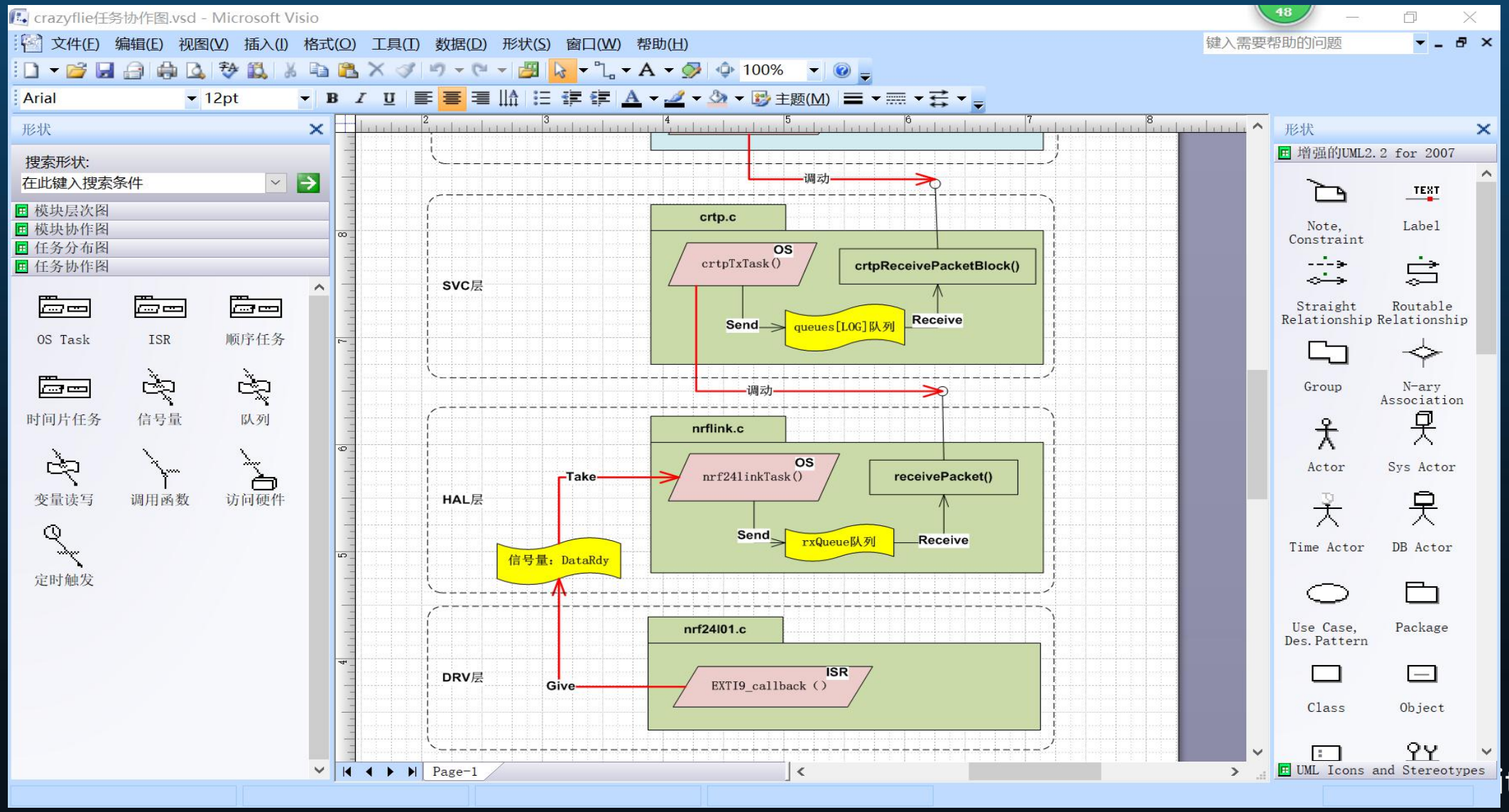
运行时(假)可插拔

架构守护——架构可视化：一个例子



架构守护——架构可视化：一套模板

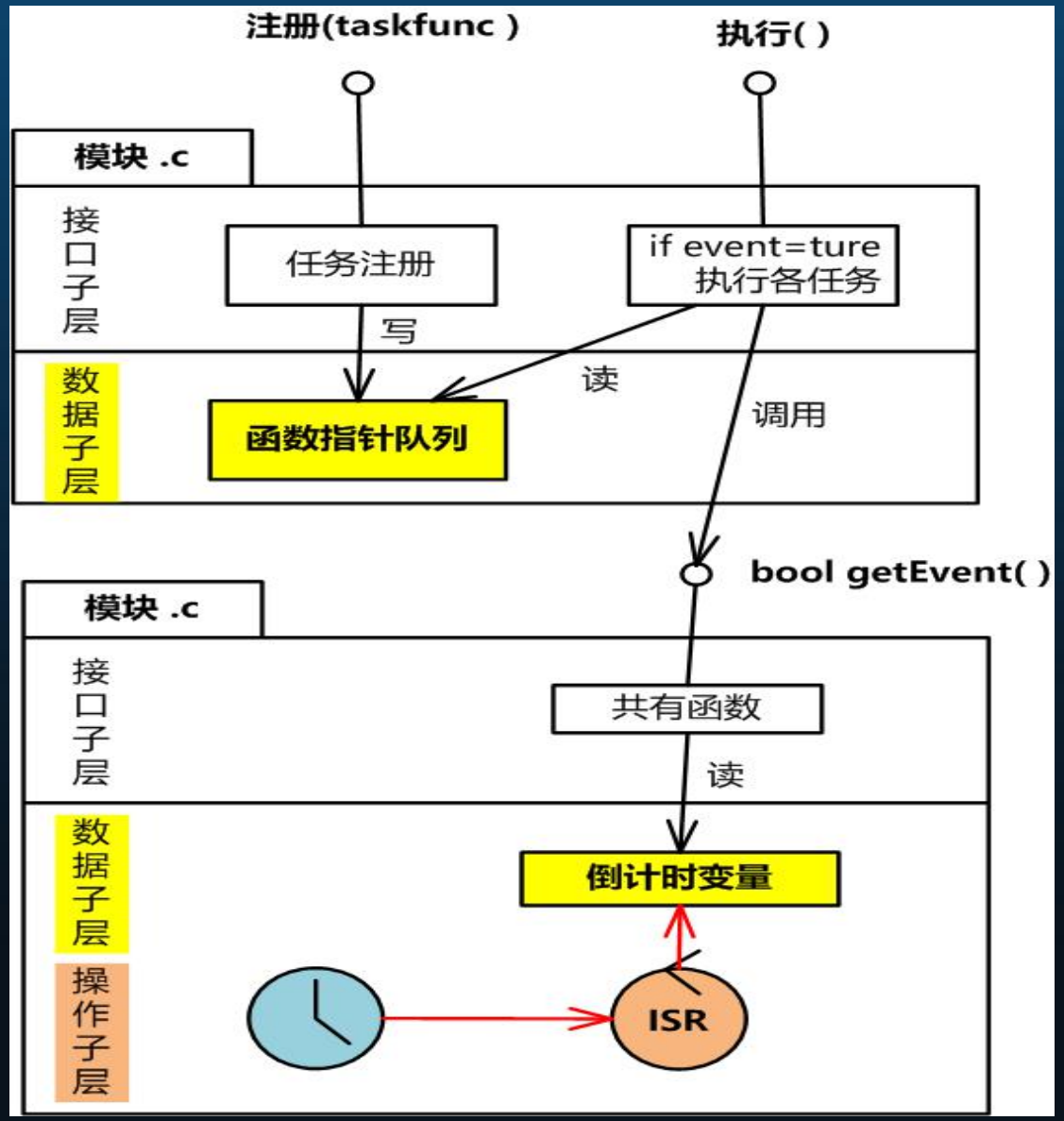
架构守护——架构可视化：一套模板



架构守护——可拔插：编译期(真)可插拔

架构守护——可拔插：编译期(真)可插拔

架构守护——可拔插：运行时（真）可插拔



由任务或中断调用

由时钟中断触发

架构守护——可拔插：运行时（假）可插拔

```
////////////////////////////////////////////////////
//          job framework
////////////////////////////////////////////////////

typedef int  (*JudgeFunc) (void);
typedef void (*LocalJob)  (void);
typedef void (*RemoteJob) (void);

typedef struct {
    char*     name;
    JudgeFunc judge;
    LocalJob  localJob;
    RemoteJob remoteJob;
} JobDefine;

void JobRun(JobDefine jobs[], int single)
{
    for (int i=0; NULL != jobs[i].name; i++)
    {
        if( jobs[i].judge() )
        {
            if(NULL != jobs[i].localJob)  jobs[i].localJob();
            if(NULL != jobs[i].remoteJob) jobs[i].remoteJob();
            if(single) return;
        }
    }
}

#define SINGLE_JOB_RUN(jobs) JobRun(jobs, 1);

#define MULTI_JOB_RUN(jobs) JobRun(jobs, 0);
```

架构守护——可拔插：运行时（假）可插拔

```
////////////////////////////////////////////////
//          programming by config
////////////////////////////////////////////////

int judgeMotorStatus()
{
    return 1;
}

void doBraking()
{
    printf("\n  this is a job");
}

void reportResult()
{
    printf("\n  this is a job");
}

JobDefine  brakeFrontendTask[] = {
  {"brake job",   judgeMotorStatus,  doBraking,   reportResult},
  {"only report", judgeMotorStatus,  NULL,        reportResult},
  {"only do",     judgeMotorStatus,  doBraking,   NULL},
  {NULL,          NULL,              NULL,        NULL},
};

interrupt CpuTimer0ISR()
{
    MULTI_JOB_RUN(brakeFrontendTask);
    return 0;
}
```

谢谢