TiD 2017
质量竞争力大会

终端软件重构之旅

TiD2017

目录

- 作者简介
- 终端背景描述
- 重构问题
- 原因分析
- 思路&措施
- 效果展现
- 演示内容

作者简介

缪伟：

中兴通讯，PON终端产品软件架构师。

有多年的团队敏捷管理和技术实践经验。近年来一直从事固网终端产品的软件架构研究，提出软件包化思路，重构后的产品焕发新生。

王玲：

中兴通讯，软件开发工程师。

对敏捷软件开发、特性团队管理、UT/FT测试有深入研究。近年来，一直从事敏捷开发团队转型相关工作研究，在特性团队运作、TDD、持续集成等实践方面有丰富经验。

终端背景描述

终端产品背景描述。

- ◆ 市场分别广，需求差异大。
- ◆ 海量发货量，维护周期长。
- ◆ 开发时间短，琐碎定制多。
- ◆ 受限运营商环境，无法主动升级。

为什么要重构

现象一. 硬件方案切换工作量大、周期长。

例如：换方案需要1个月～3个月。

现象二. 业务定制继承移植周期长。

例如：某运营商定制重复移植，最长3个月以上。

现象三. 简单开发也要耗费很多人和时间。

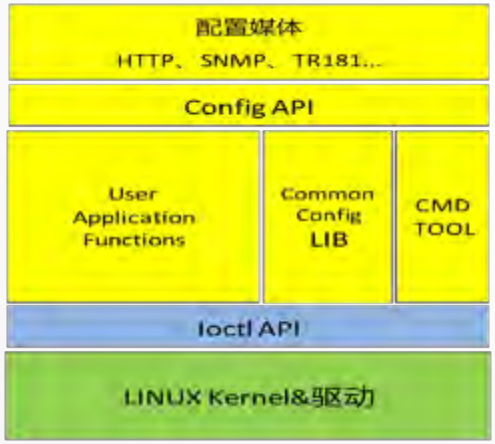
例如：简单页面开发，经常要耗费大量人力。

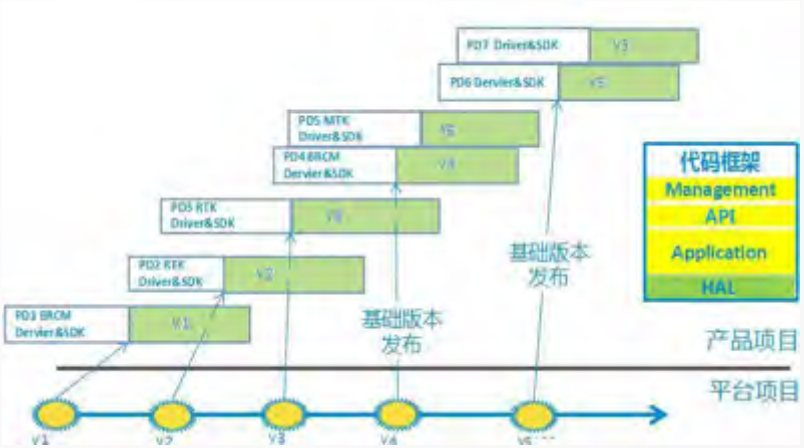
注：时间有限，现象三不作重点描述。

找问题-从终端系统架构上分析

终端产品的系统架构图和产品运作模式。

- 完整LINUX系统。
- 功能放置在系统各个部分。
- 平台阶段性发布版本。
- 产品取最新平台版本滚动发展。

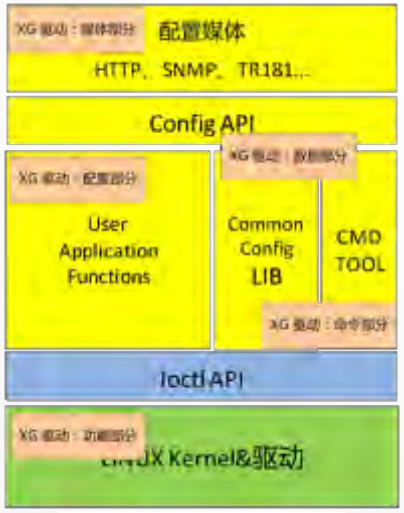




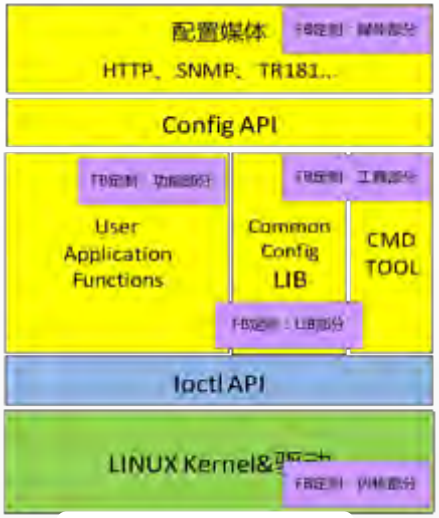
找问题-系统耦合

业务耦合，难以移植，驱动耦合，方案切换困难。
功能霰弹式修改，快速移植基本不可能。

范例：
请把XG功能从A移植到B，请把FBT定制从B移植到A。



A 产品



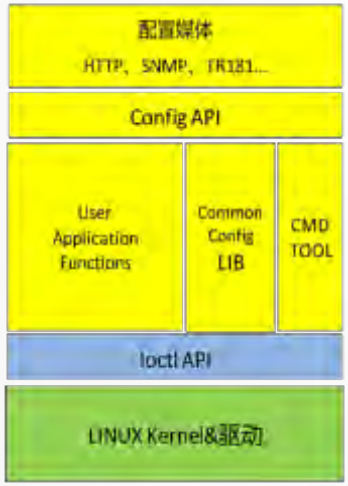
B 产品

找问题-从基线管理模式分析

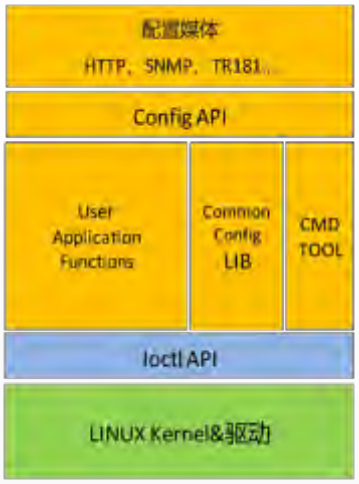
多基线，产品对应平台接口差异，功能移植相当于重新开发。

范例：

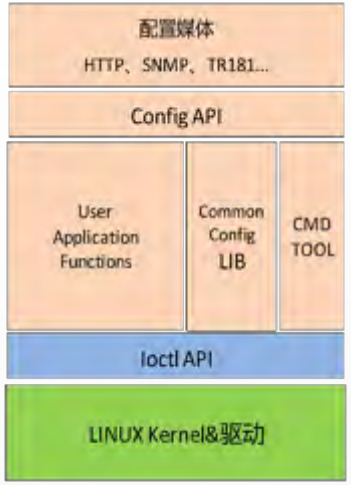
请把BT功能分别移植到基线1，2，3上。



基线1 V1.版本



基线2 V2版本



基线3 V3 版本

找问题-为什么有这么多基线

- 原因1. 定制相互干扰，不敢用共基线。
- 原因2. 开发时间短，任务量大，拉基线直接修改速度最快。
- 原因3. 老基线因为无法自主部署替换版本，无法消亡。
- 原因4. 平台产品关系模式，使用平台新功能而增加基线。

解决思路

◆ 高内聚低耦合-建立运营商码包，驱动码包，建立码包库。
目的：提高移植效率，避免相互干扰。

◆ 共基线建设，基本功能解耦=》码包化。
目的：模块解耦，解决共基线模块相互干涉的问题。

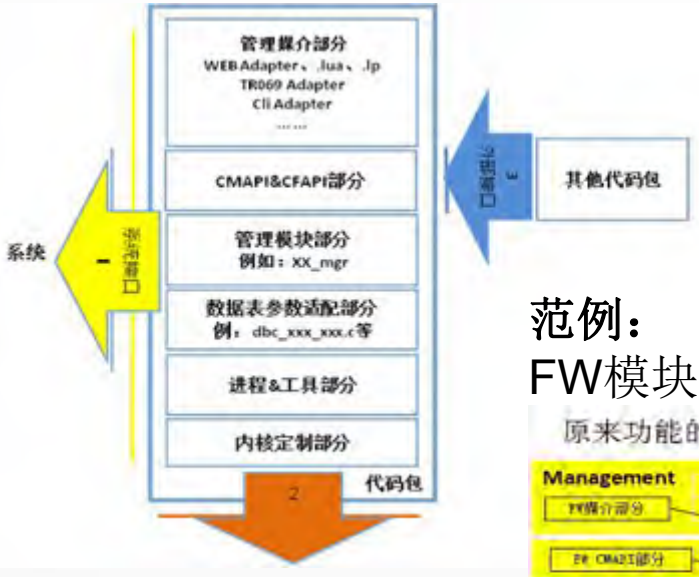
◆ 平台产品关系重构，版本发布调整为基础功能码包化共建。
目的：减少基线数目。

◆ DEVOPS应用，静态检查/UT/FT/快速部署/CRT，快速反馈。
目的：弥补解决防护问题

措施.代码包模型方案

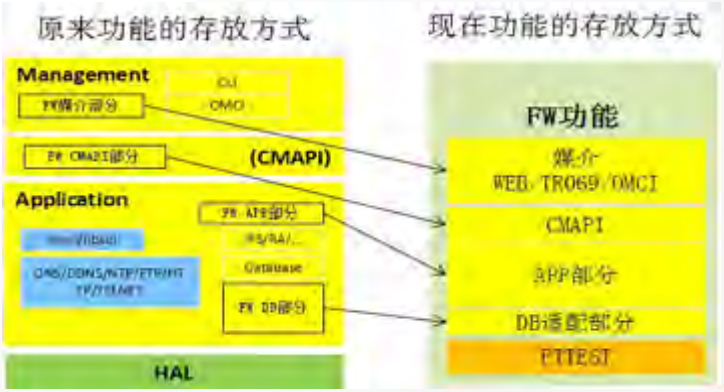
高内聚低耦合，功能以业务为单位存放。

- 文件隔离。
- 编译独立。
- 接口独立。



范例：

FW模块的调整。



解决：

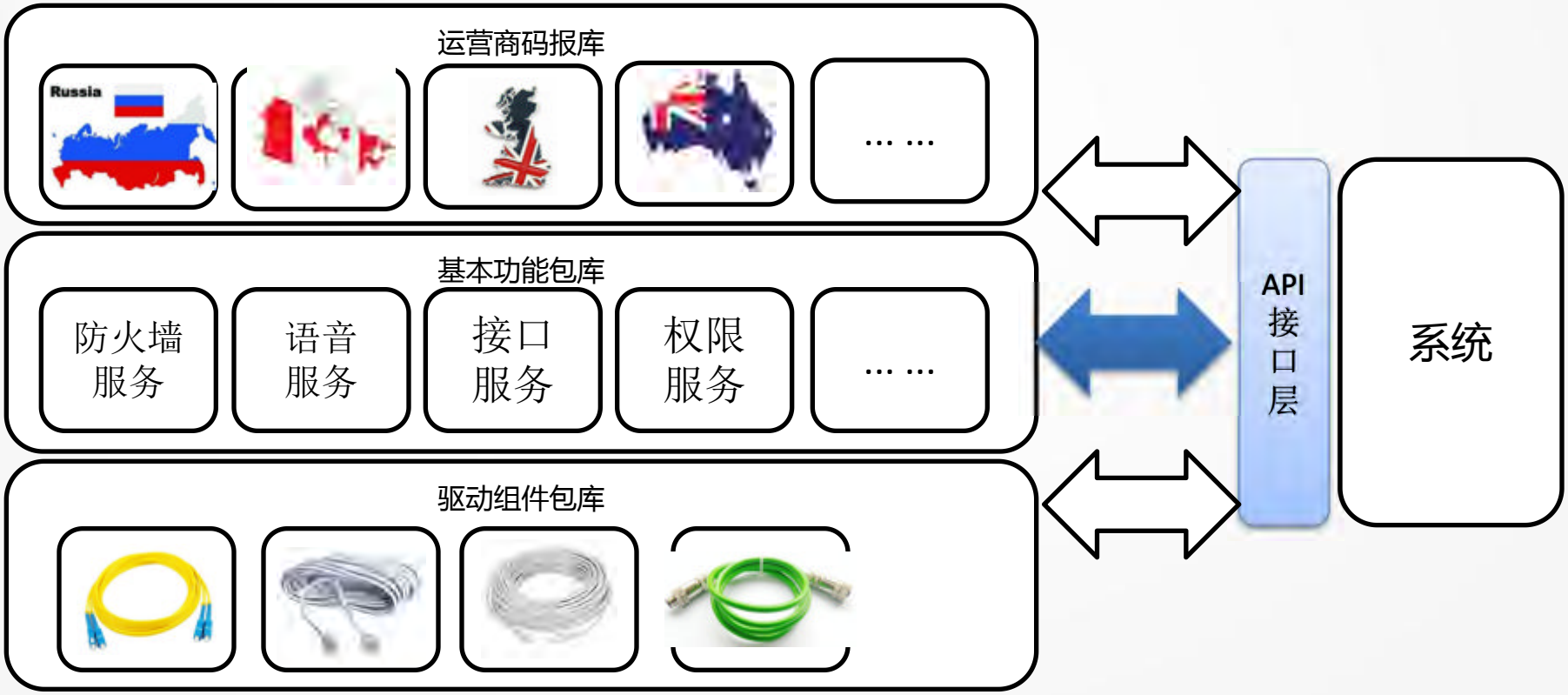
高效移植的问题。

定制位于不同码包，易于隔离。

系统复杂性降低，便于基本功能升级。

措施.建设码包库

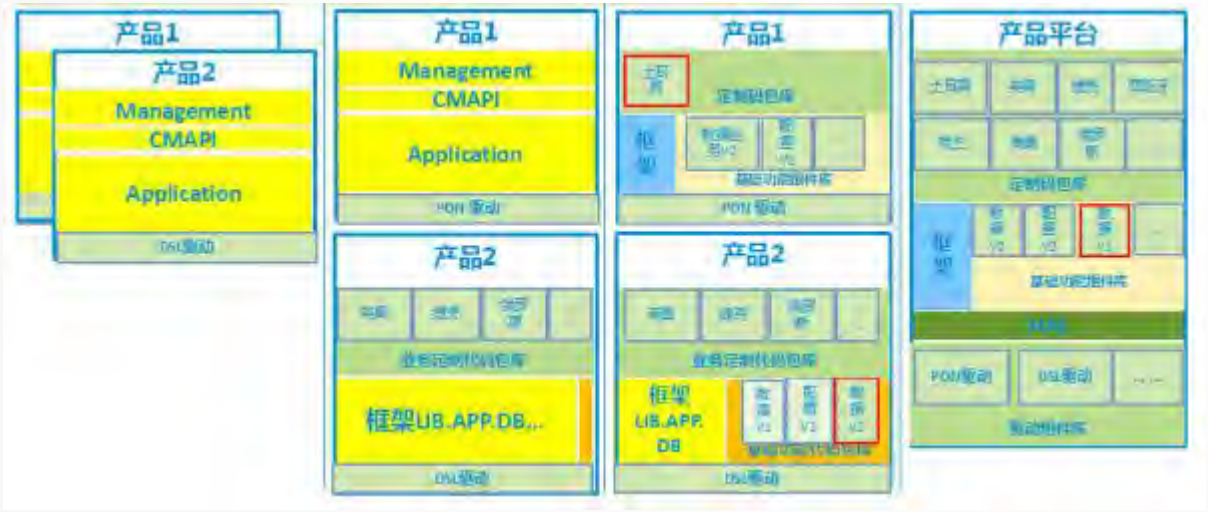
API接口层，衔接基本功能和业务定制，驱动组件。



措施-解耦和共基线建设

码包模型应用，解耦，共基线演变。

演变过程



解耦细节技术

- 功能HOOK挂接。
- 类库API接口。
- 注册回调-通知链。
- XML拼接。

共基线常见阻碍

阻碍1：DEVOPS不完备，质量如何保障。-》解耦差异共存。
先以的文件迁移实现简单解耦，容忍“坏味道”，差异共存，逐步完善。

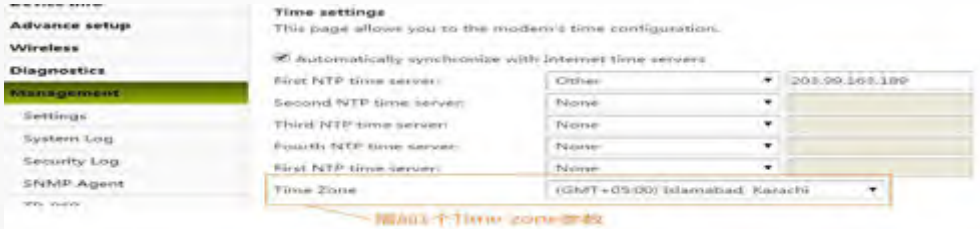
阻碍2：巨大存量市场，如何解决模块重构后的一致性问题。
建立码包版本选择机制，新老版本并存。

阻碍3：各产品功能模块差异，无法共基线。
断臂求生，模块代码包化解耦，保证主体共基线。

找问题-从软件模块架构上分析

业务定制抽象不够，简单定制修改大量代码。

范例：请增加1个媒介参数。



需要修改大量代码。

1.媒介后端
编写CONFIG-LIB适配函数接口。
编写LUA函数调度接口。

2.媒介前端
编写Html/JavaScript脚本。

3.数据处理模块
编写参数保存和读取。

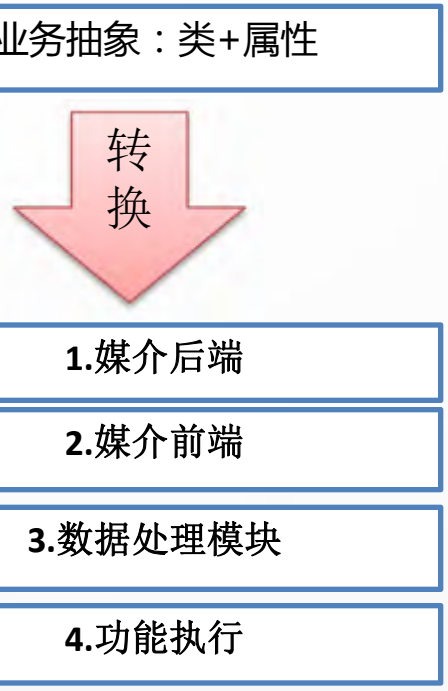
4.功能执行
功能运行及生效。

措施-DSL思想应用

需要一双发现问题的眼睛,应用DSL思想进行抽象改进。细节提效无处不在。

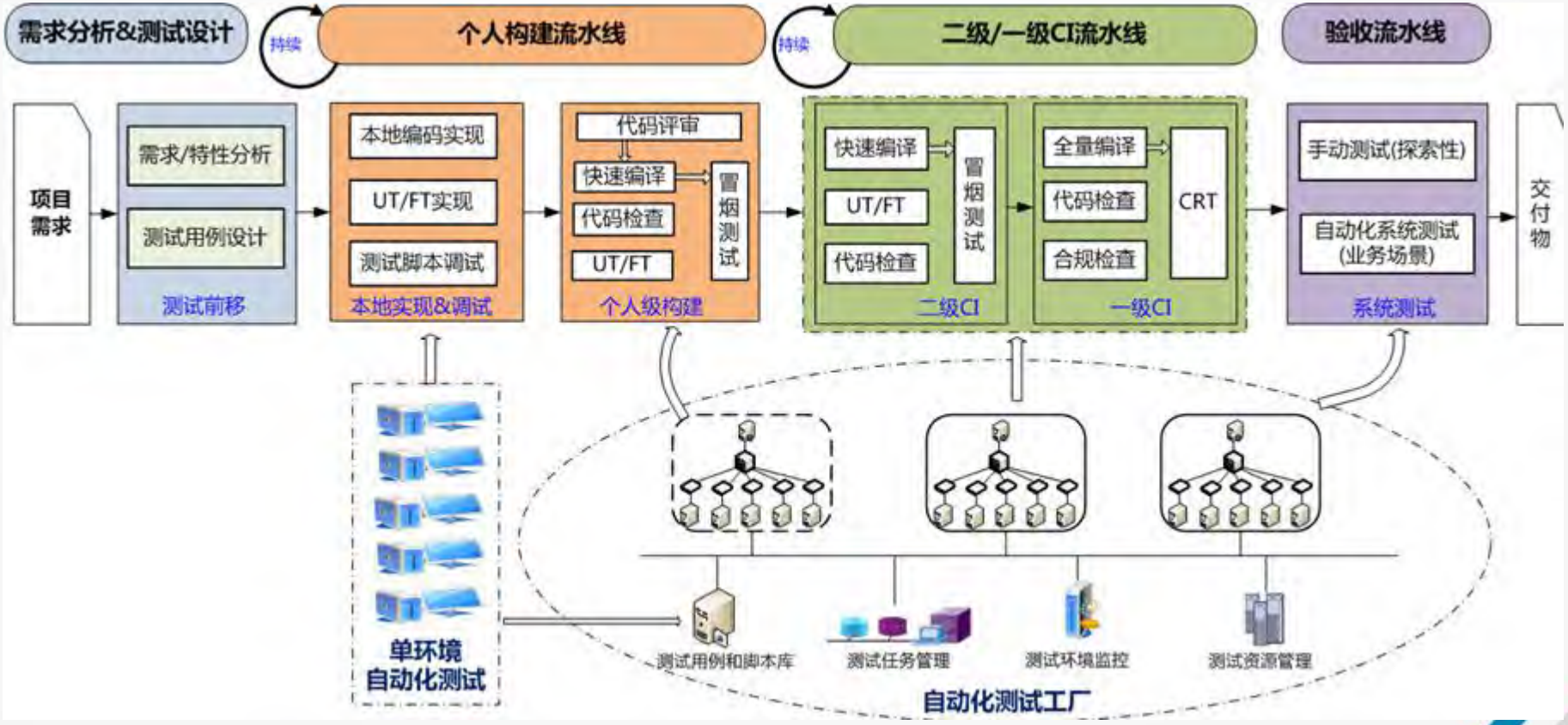
例如：
利用python转换的机制。
创建类/属性定义抽象。

自动的翻译成
Html解析脚本，
Javascript控制脚本，
数据库表及参数，
数据接口实现。
… …



措施-DEVOPS

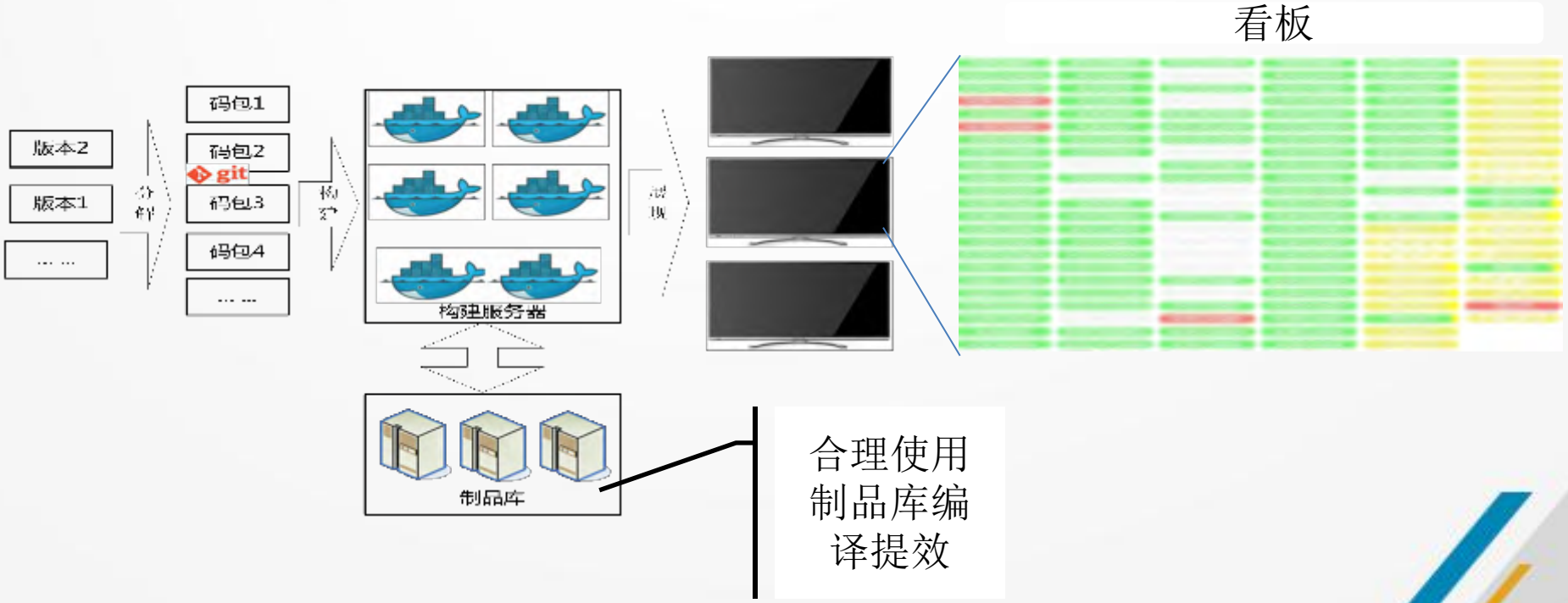
DEVOPS建立流水线：快速反馈，构建、检查、UT/FT/CRT，验收。



DEVOPS-构建-终端海量版本

海量终端版本数量，构建耗尽资源。

- 建立完整看板系统。
- 构建提效：解耦+容器技术+制品库-》构建提效。
- 策略选择构建频率，优先级，关键>通用>定制。

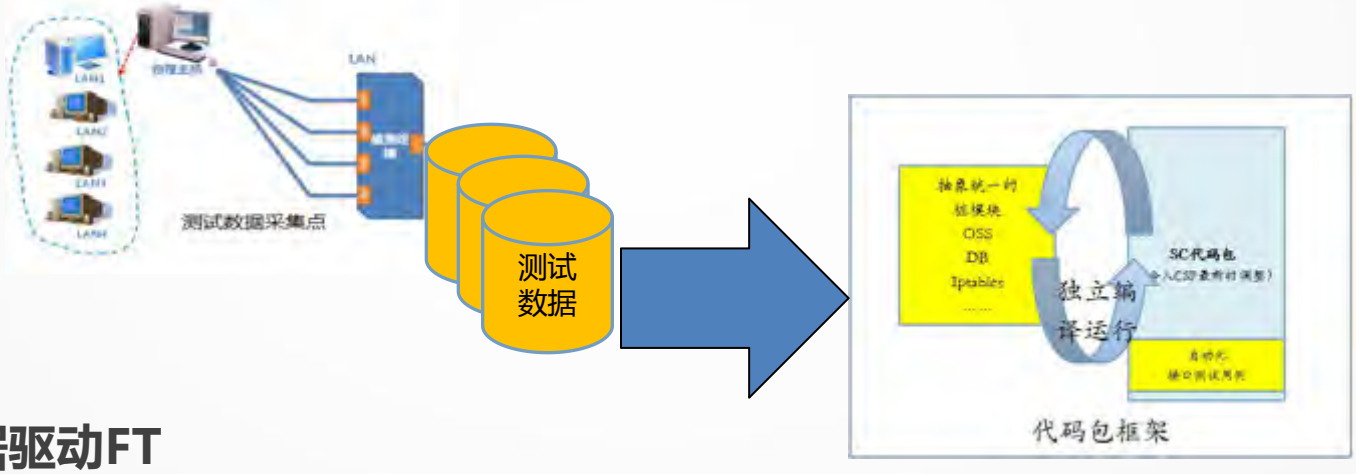


DEVOPS-FT提效（演示）

现有系统，FT太难写，待补的工组量太大，改进措施：

◆ 学习型FT

设备测试运行，产生大量数据，记录转义，再驱动FT测试。



◆ 数据驱动FT

学习型的FT，数据不透明，进一步改进数据格式，

实现测试数据可配置，驱动FT建立防护网，如下学习出来的测试数据，可进一步修改。

[编号]:613004261772 [入库原因]: [redacted]支持LAN侧http_https端口号可配置

[pon_g_ronghe/packages.git] / onci / onciprocess / test / data / sc_vlan300tovlan600 / T_CM_ETH_PORT_VLAN_TRANSLATION_.xml

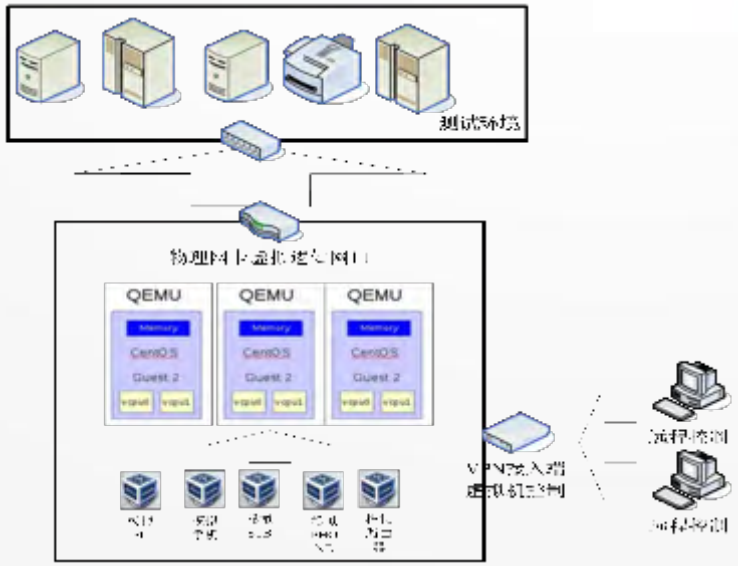
```
<Tbl  name="T_CM_ETH_PORT_VLAN_TRANSLATION_"  RowCount="0">
<Row  No="0">
<DM  name="tIFName"  val="IGD.LD1.ETH1"/>
<DM  name="tVlan_en"  val="1"/>
<DM  name="tPonVlan"  val="500"/>
<DM  name="tUniVlan"  val="300"/>
<DM  name="tVlanAction"  val="2"/>
<DM  name="tPon_pri"  val="0"/>
<DM  name="tUni_pri"  val="3"/>
<DM  name="tPri_en"  val="1"/>
</Row>
</Tbl>
```

DEVOPS-自动化效率

终端设备测试耗费大量设备，用例测试周期太长。

- 云测试-多单板复用相同环境。

- 引入云/虚拟测试模式（演示）



效果展现

- 未使用码包架构，未共基线
  方案移植1～3个月；业务定制1～3个月；30%左右再做重复工作。

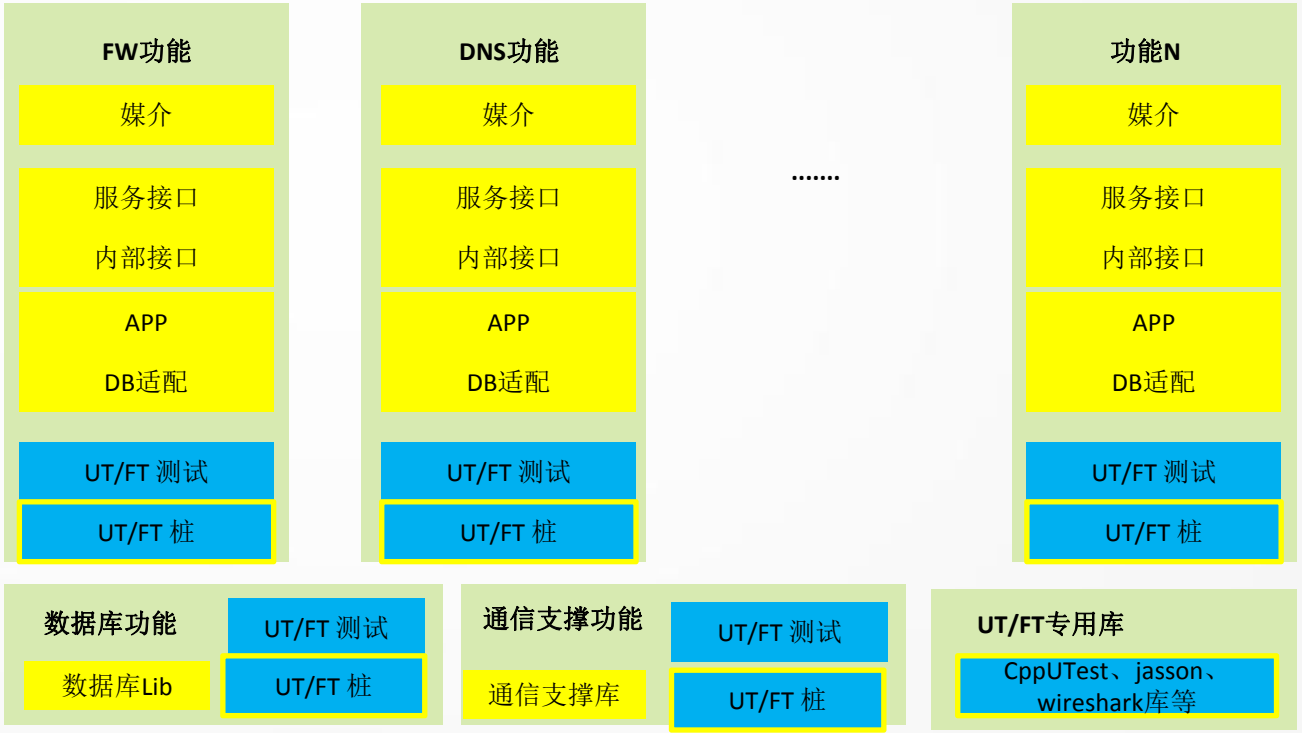
- 未使用码包架构，共基线，无DEVOPS
  项目处于死亡边缘，版本基本做不出来，相互干涉的质量问题无法保证。

- 使用码包架构，共基线，DEVOPS应用
  和谐共存（上百个地区运营商，几十种硬件方案）
  方案切换小于1周
  业务移植小于3天。

UT/FT的测试框架

测试代码以模块为单位

- 独立编译。
- 独立运行。
- 独立结果展示。

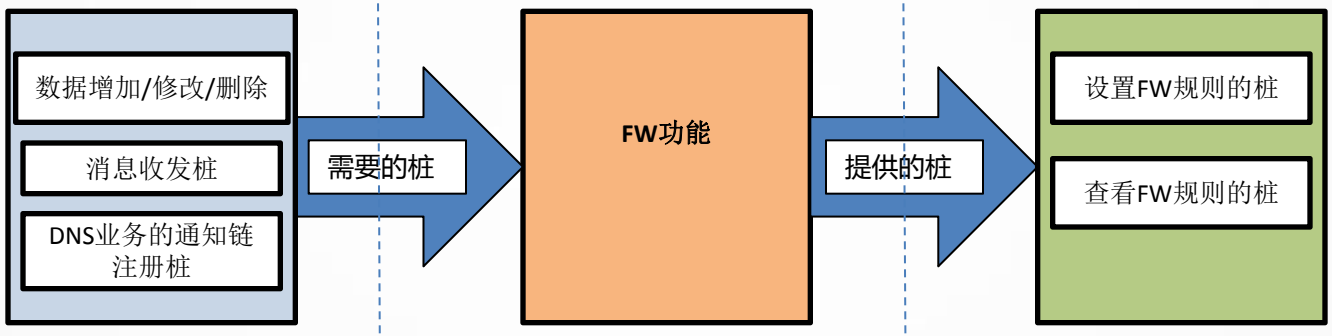


细节

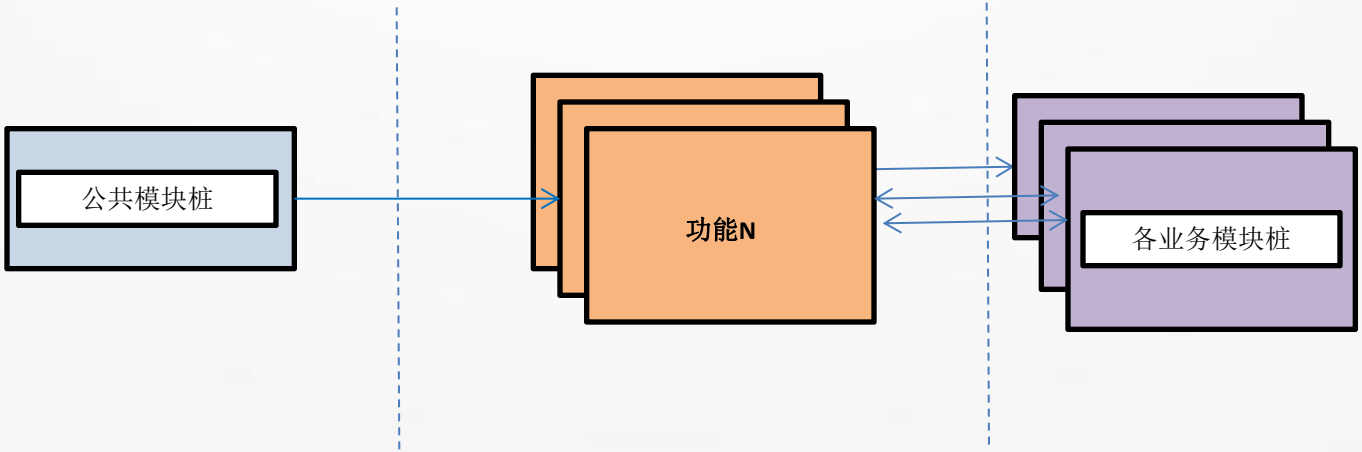
- 桩HOOK挂接。
- 实测数据驱动。

UT/FT桩迭代增加

◆ 各业务模块即是桩的使用者，也是提供者



◆ 桩体系逐步丰富



TiD2017测试用例实践-新接口

- 新接口，使用TDD方式实现

```
TEST_GROUP(MainTestGroup)
{
    void setup(){}
    void teardown(){}
}

Test(MainTestGroup, WebConfig)
{
    /*新接口TDD，先写一个不通过的用例*/
    int iRet = web_setInst("DEV1");
    CHECK_EQUAL(iRet,0);

    int iRet = web_setInst("Dev2");
    CHECK_EQUAL(iRet,0);

    int iRet = web_setInst("DEV11111111111.1111111");
    CHECK_EQUAL(iRet,-1);

    int iRet = web_setInst("");
    CHECK_EQUAL(iRet,-1);
}
```

TDD

```
int web_setInst(char * INST_ID)
{
    if(NULL==INST_ID)
    {
        return -1; //参数错误
    }

    if(strlen(INST_ID)>MAX_IDLEN)
    {
        reutrn -1;  //参数错误
    }

    return dbSetView(DBVIEW,INST_ID);
}
```

- 写一个不通过的用例。
- 实现功能代码，用例通过。
- 重构
- 迭代进行，直至边界测试完成

有依赖关系的接口

- 初始化时设置环境参数

```
TEST_GROUP(MainTestGroup)
{
    void setup(){
        dbstart();
        RunMsg(FW_PID,EV_START,NULL,0)  }
    void teardown(){
        DBExit();  }
}

Test(MainTestGroup, WebConfig)
{
    OSS_EVENT tEvent = {0};

    SubScribNotifyHook = FWSubScribNotifyHook;
    ioctlHook = FWioctlHook;
    socketHook = FWsocketHook;
    CreateSocketHook = FWCreateRawSocketHook;
    sendtoHook = FWsendtoHook;

    tEvent.id = EV_CMAPI_SET;
    OssRunMsg(DHCP4C_PID, &tEvent, buffer, sizeof(T_FWINFO));
}
```

TDD

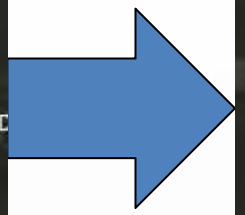
```
int web_setInst(char * INST_ID)
{
    VIEW_INIT(tFWComm);
    iRet = dbAPIGetView(DBVIEW_FW_COMM, "DEV

    return dbSetView(DBVIEW,INST_ID);
}
```

- setup()中模块初始化。
- 通过hook函数决定接口的执行轨迹。
- tearDown()中释放模块资源。

测试用例-旧接口

- 稳定的旧接口，测试用例和功能代码互相验证

```
Test(MainTestGroup, getaddressinfo_Server_OK)
{
    struct hostent *hp = NULL;

    InitFakeServer();
    PrePareDefServers("./testdata/test_Server_OK.txt");
    hp = getaddressinfo("www.sina.com.cn", NULL, NOWAIT_D
    CHECK_EQUAL(CheckResult(hp),0);
    FreeFakeServerAndCfg();
}
```



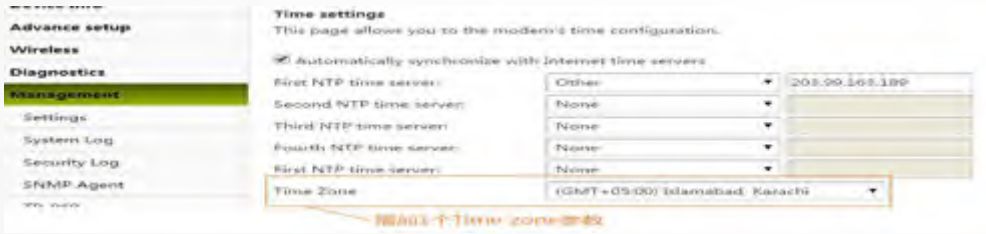
```
int getaddressinfo(char * INST_ID)
{
    struct hostent *csp_gethostbynameMix (const C
{
    static struct hostent h;
    static CHAR           buf[sizeof(CHAR *) * ((
                              sizeof(struct in_ad
                              sizeof(struct in6_a
                              sizeof(CHAR *) * (A
    DWORD                 dwBuffLen = sizeof(CHAR
                                      sizeof(stru
                                      sizeof(stru
                                      sizeof(CHAR

    return getaddresinfo_Thread(name, &h , buf, d
}
}
```

- 初始化过程中预制了DNS服务器状态，和应答报文。
- 验证IniteFakeServer等初始化函数是否正确。

UT/FT困惑

- 手工设置测试边界，工作量较大。
- 开发人员思维定势，影响测试全面性。
- 人工构造报文，不够逼近实测环境。

范例：请增加1个媒介参数。



1.手工设置测试边界
编写新Test。
设置边界值。

2.多少测试用例才算够

10？50？

3.协议交互报文构造
手工构造报文过于简单。

数据驱动型FT

- 采集实测的配置数据，回放，自动设置边界值。
- 收集实测报文，回放，更接近实际环境。

TiD2017

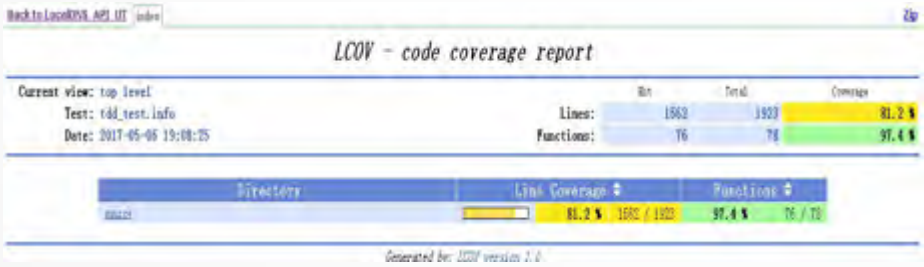
数据驱动型FT

- 收集testdata信息。
- 在测试框架里回放。
- 测试效果逼近实测。

```
Test(MainTestGroup, getaddressinfo_Server_OK)
{
    struct hostent *hp = NULL;

    InitFakeServer();
    PrePareDefServers("./testdata/test_getaddressinfo_Server_OK.txt");
    hp = getaddressinfo("www.sina.com.cn", NULL, NOWAIT_DNSSERV_PRI_DISA
    CHECK_EQUAL(CheckResult(hp),0);
    FreeFakeServerAndCfg();

    InitFakeServer();
    PrePareDefServers("./testdata/test2_getaddressinfo__Server_OK.txt");
    hp = getaddressinfo("www.sina.com.cn", NULL, NOWAIT_DNSSERV_PRI_ENAB
    CHECK_EQUAL(CheckResult(hp),0);
    FreeFakeServer();

    /*cache test*/
    printf("cache test\n");
    InitFakeServer();
    PrePareDefServers("./testdata/test_getaddressinfo_cache_Server_OK.tx
    hp = getaddressinfo("www.sina.com.cn", NULL, NOWAIT_DNSSERV_PRI_ENAB
    CHECK_EQUAL(CheckResult(hp),0);
    FreeFakeServerAndCfg();
}
```

FT结果显示

- 测试代码是版本不可缺少的一部分，如同测试报告。
- 在VerifyCI流水线中执行，快速反馈给相关开发人员。
- 覆盖率问题。




```
csp/component/firewall/cspd/test make gcov Failure!
See more details: http://
Build step 'Execute shell' marked build as failure
```

谢谢！