G09_接口设计之美_支持 Agile 敏捷开发

内容：

1. 一般架构设计 vs. 敏捷设计
2. 敏捷设计的主要观点
3. 架构的表述(Representation)与沟通(Communication)
4. 如何表述(Represent)”足够好的架构”呢?
5. 复习：代码造形的角色
6. 敏捷架构师：<一般架构设计>也能敏捷起来
7. 敏捷架构师+敏捷开发者
8. 敏捷架构师的素养
9. 架构师团队+敏捷迭代
10. 结语

1. 一般架构设计 vs. 敏捷设计

一般架构设计：” Design is Code”

在一般的架构设计里，架构设计是完整的，架构师设计完成整体架构之后，在交给开发者来编程，迅速落实为代码(Design is Code)，进行必要的测试，给予回馈。当需求有所变更时，开发者需要同时维护设计与代码，双重负担，开发者敏捷不起来。

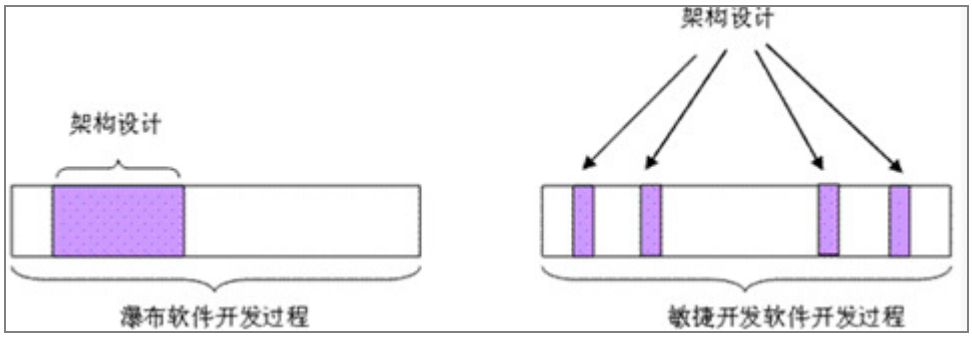
敏捷设计：” Code is Design”

反之，在敏捷设计里，将架构设计被切分为许多部份，分散于整个开发流程之中，包括规划、编程、测试、重构等各阶段；让设计直接表现于代码里(Code is Design)，避免开发者同时维护设计与代码的负担，让开发者敏捷起来。

2. 敏捷设计的主要观点

在敏捷开发里，流行的敏捷设计观点是：“敏捷是把原来架构设计里的详细设计放到了编码的过程中，开发人员要有架构师的思维，架构师的预先架构设计要<正好足够>〃” 例如，2009 年 7 月 6 日陈序明先生在<< IT168>>上发表的文章里，就说到：

敏捷开发把原先软件过程前期的架构设计，分散到了整个敏捷开发软件过程中。



架构设计包含两方面：1)架构；2)设计。其中设计中的详细设计需要大量的时间，包含详细的流程，API，数据结构等设计。但软件开发阶段的 Code 编码阶段，同样蕴含了很多详细设计的内容。换句话说，现在敏捷开发提倡 Code is

design，而以前是 Design is code。但是开发人员维护一套 Design，外加一套 Code，不堪重负，效率低。所以，现在是 Code is Design 盛行，敏捷盛行。基于这两种原因，敏捷中将传统的架构设计分成：**架构 + 设计**：

（1）敏捷开发的架构保留架构部分；

（2）转移设计到 Code 编码阶段、重构阶段、Unit Test 阶段等。

而详细设计阶段，则在 Code 编码和 UT 单元测试阶段进行。这个阶段重构很重要，重构使你的软件架构和组件结构自然呈现。所以在敏捷开发中架构设计的内容发生了变化：敏捷开发中止于架构，轻详细设计。但详细设计不是消失不见了，而是转移到了开发阶段，也即是：Code is design。这样既能拥抱变化，又规避风险，又 Don't Repeat Yourself。

其意味着，敏捷开发将传统的架构设计分成：架构 + 设计。敏捷开发的架构师，将架构部分做到”足够好”即可。然后，将(详细)设计转移到代码撰写阶段、重构阶段、以及单元测试阶段等。简而言之，在敏捷中，架构设计仅止于”足够好”的架构，不进行详细设计。但详细设计仍然存在，只是转移到了开发阶段而已。这些详细设计需要花费大量的时间，包含详细的流程、数据结构等设计；而且在编程阶段，也蕴含了很多详细设计的内容。如果直接将设计表现于代码中(Code is design)，就能避免开发人员同时既要维护一套设计，又要维护一套代码；可大幅减轻开发者的负担，提升效率。

3. 架构的表述(Representation)与沟通(Communication)

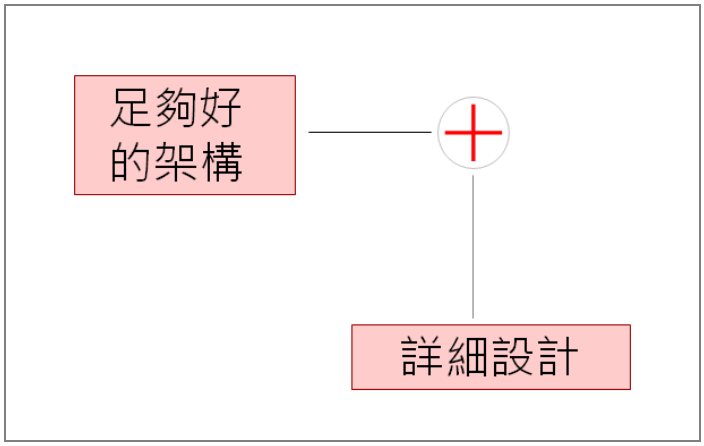
在敏捷开发里，架构师与开发者的沟通，是非常关键的。包括：

- 架构师以 EIT 等代码造形表述设计；让开发者直接对应到代码。代码造形就如同专块，建筑师叙述如何以砖块组合出形形色色的建筑物；施工者就烧出专块，并按步就班组合起来。
- 架构师和开发者都能从简单组合出复杂。亦即：造形很简单，内涵可复杂，重复地组合。
- 让用户获得从简单中叫出复杂的满足感。亦即：优质的用户体验。

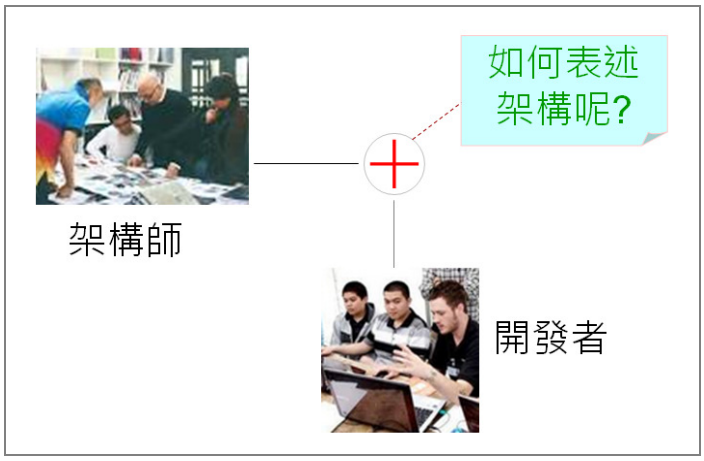
4. 如何表述(Represent)”足够好的架构”呢?

架构设计被切分为”足够好的架构”和”详细设计”两个部分，那么这两部

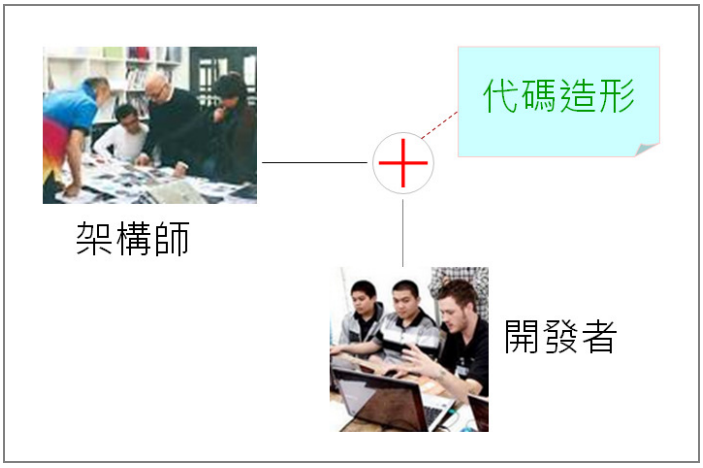
分又如何相互衔接呢?



其实大家都知道，架构师最关键的任务就是：接口(Interface)设计。所以，"足够好架构"的核心部分就是：接口的定义(Definition)和表述(Representation)。那么，有什么有效的方法能清晰而明确地定义和表述接口设计呢?



答案是：代码造形(Code Form)。



代码造形能直接对映到程序语言的结构，具有高度的精确性(Precision)，架构师能准精确地传达设计的涵义。架构师以<代码造形>表述架构，基于共同词汇，提升了共识(Shared Understanding)，开发者很容易理解其架构的设计涵意。所以，代码造形能大幅提升开发者的效率；而且迅速配合需求变更、架构创新(或重构)设计，大幅提升了整体团队的敏捷性。☆

G09_接口设计之美_支持 Agile 敏捷开发

内容：

1. 一般架构设计 vs. 敏捷设计
2. 敏捷设计的主要观点
3. 架构的表述(Representation)与沟通(Communication)
4. 如何表述(Represent)”足够好的架构”呢?
5. 复习：代码造形的角色
6. 敏捷架构师：<一般架构设计>也能敏捷起来
7. 敏捷架构师+敏捷开发者
8. 敏捷架构师的素养
9. 架构师团队+敏捷迭代
10. 结语

5. 复习：代码造形的角色

代码造形的涵意

顾名思义，代码造形就是代码层级的设计造形(Form)。代码造形就是开发者常用的词汇(Vocabulary)，其能直接对映(Map)到程序语言的基本结构，此结构大多定义成为关键词(Key word)。例如，指令(Instruction)、函数(Function)和类(Class)。

代码造形的特征

代码造形的特征如下：

- 单一的造形，架构师以它叙述形形色色的设计，开发者直接将它落实为代码。就如同专块，建筑师叙述如何以砖块组合出形形色色的建筑物；施工者就烧出专块，并按步就班组合起来。
- 让生产者(架构师+开发者)从简单组合出复杂。亦即：造形很简单，内涵可复杂，重复地组合。
- 让用户获得从简单中叫出复杂的满足感。亦即：优质的用户体验。

以<类>(Class)造形为例

- 造形很简单：软件的类(Class)造形，内部只有 2 种要素：函数(Function)和属性(Attribute)；其要素之间的关系也能很清晰。
- 内涵可复杂：在软件系统里，各种不同内涵，可以透过类造形来呈现出来。类造形就像集装箱，可以容纳各种多变得复杂内涵。
- 重复地组合：类造形像积木一样，定义了明确的组合关系(例如”继承(Inheritance)”关系、”结合(Association)”关系等)。于是，人们能将拿类造形来重复组合成为复杂的大系统。

代码造形在沟通上的用处

- 因为代码造形能直接对映到程序语言的结构，具有高度的精确性(Precision)，架构师能准精确地传达设计的涵义。
- 因为代码造形是开发者的词汇，架构师以<代码造形>表述架构，基于共同词汇，提升了共识(Shared Understanding)，开发者很容易理解其架构的设计涵意。
- 所以，代码造形能大幅提升开发者的效率；而且迅速配合需求变更、架构创

新(或重构)设计，大幅提升了整体团队的敏捷性。

- 架构师如同妈妈，使用 kid language 来与小孩交谈，非常有助于小孩语言天份的开发。同样地，架构师以<代码造形>来表述架构，来与开发者交谈；非常有助于开发者设计能力的提升。
- 架构师自由创意去思考架构设计(加法设计)，但是都以一致的<代码造形>来表述架构设计(减法设计)。就如同唐诗的<七言绝句>造形，李白、杜甫、白居易人人都能发挥创意、尽情思考，但都以一致的造形来表述(Representation)。不但没有伤害创意，而且还基于<诗同形>而相互激发创作的氛围。
- 即使架构尚未达到”足够好”，只要能清晰而明确地表述，就能随着敏捷迭代而互相切磋着磨而达到”足够好”的境界了。

回顾代码造形的演进

回顾 1970 年代的 C 语言，函数(Function)和数据结构(Data Structure)是计算机语言的基本模块(Basic Building Block)；因此 IT 系统分析&设计人员就以函数和数据结构去发展建模方法和实现工具。例如当年的结构化(Structured)设计方法和数据库(Data Base)系统等。当时系统分析与设计的基本模块是：单一的函数造形；并且能直接对映(Map)到代码。

到了 1980-90 年代，OOP 成为业界主流，类(Class)成为 C++和 Java 等 OOP 语言的基本模块。因而有了 UML 建模语言和 OOAD(Object-Oriented Analysis and Design) 方法问世。于是，系统分析与设计的基本模块扩大为：单一的类造形；并且能直接对映(Map)到代码。

以此类推，1995 年 Gamma 推出了设计模式(Design Patterns)，成功地成为系统分析和设计的基本模块。但是，设计模式花样繁多，其幕后缺乏单一模式来组合出各种设计模式，无法直接对映到代码。因而，设计模式未能扩大系统分析和设计的基本模块。至今，类(Class)仍然是系统架构师或分析师心中的基本设计模块。

于 2012 年 5 月，高焕堂老师提出了单一的 EIT 造形，来扩大系统分析和设计的基本模块；并且能直接对映(Map)到代码。这 EIT 造形内含 3 种要素：<E>对映到代码的基类(Super-class)、<I>对映到基类的抽象函数(Abstract Function)、而<T>则对映到代码的子类(Subclass)，如下图所示：

其中，接口就是一种抽象类，所以在本质上，EIT 造形就是由 3 个类所构成的，能直接对映到代码(即 3 个类的代码)。EIT 造形就像自然界的原子(Atom)造形或 DNA 螺旋造形一般，自然界的不同物质都具有一致的简单造形，但其内涵却都可以不相同。其中，”EIT”名称是来自汽车的<引擎(Engine)、接口(Interface)、轮胎(Tire)>三种角色，及其之间的关系。

EIT 造形介于类和设计模式之间。它是由 3 个类所构成的单一造形；它又能组合出各种设计模式，以及各种框架(Framework)。☆

G09_接口设计之美_支持 Agile 敏捷开发

内容：

1. 一般架构设计 vs. 敏捷设计
2. 敏捷设计的主要观点
3. 架构的表述(Representation)与沟通(Communication)
4. 如何表述(Represent)”足够好的架构”呢?
5. 复习：代码造形的角色
6. 敏捷架构师：<一般架构设计>也能敏捷起来
7. 敏捷架构师+敏捷开发者
8. 敏捷架构师的素养
9. 架构师团队+敏捷迭代
10. 结语

6. 敏捷架构师：<一般架构设计>也能敏捷起来

在敏捷开发里，把原来架构设计里的详细设计放到了编码的过程中；所以，架构师只要将架构部分做到”足够好”即可;而大部分的设计工作转移到开发者身上。相对上，架构师处于配角地位，来支持开发者，让开发工作更具有敏捷力。

在一般(或传统)软件开发里，则反过来，架构师是主角；相对上，开发者处于配角地位。如果开发者，能够来协助架构师，让架构的关键部分(如平台接口或团队接口的<I>部分)能迅速落实为代码，进行必要的测试，给予回馈。也能大幅提升了架构师团队更具敏捷力。这意味着，敏捷思维不仅适用于开发团队(架构师是配角)；也非常适用于架构师团队(开发者是配角)。

7. 敏捷架构师+敏捷开发者

以上的”Design is Code”和”Code is Design”看似对立的；其实”Design”与”Code”就如同太极的阴阳两仪，其界线并不是泾渭分明的。例如，有些团队里的开发者并没有足够经验和技能，来承担”Code + Design”的双重任务时，若能及时发挥架构师高度敏捷力，藉由 EIT 造形来表述架构，就能强化”Design is Code”来分担”Code is Design”的工作，却是落实敏捷开发的常见途径。因之，架构师和开发者的敏捷力，必须兼而顾之，并且相辅相成的。简而言之：

- 敏捷架构师针对关键部分(如平台接口或团队接口的<I>部分)，请求开发者来协助编程，让设计迅速落实为代码(Design is Code)，进行必要的测试，给予回馈。
- 敏捷开发者，将细节设计被切分为许多部份，分散于整个开发流程之中，包括规划、编程、测试、重构等各阶段；让设计直接表现于代码里(Code is Design)。
- 如果上述理想的”Code is Design”途径难以落实的情况下(常常源于团队组织结构的限制)，只要架构能清晰表述和沟通，采取敏捷架构师的”Design is Code”的途径，也能有效支持敏捷迭代过程的。

8. 敏捷架构师的素养

8.1 架构师的视角

- 以代码造形思考设计；让开发者直接对应到代码。代码造形就如同专块，建筑师叙述如何以砖块组合出形形色色的建筑物；施工者就烧出专块，并按步就班组合起来。
- 设计师从简单组合出复杂。亦即：造形很简单，内涵可复杂，重复地组合。
- 让用户获得从简单中叫出复杂的满足感。亦即：优质的用户体验。

8.2 敏捷架构师关键议题

敏捷架构设计的 2 个关键议题是：

- 架构师团队如何给自己创造重构的自由度，以及支持开发者重构的空间。
- 架构设计如何迅速落实为代码。

敏捷架构师的关键任务：创造重构的自由度

架构师团队如何给自己创造重构的自由度，以及支持开发者重构的空间，是敏捷设计的关键议题。这种自由度，决定于架构师是否能仔细分辨出：关注<未来的决策>与关注<今天决策的未来性>的微妙差异了。愈是能关注<今天决策的未来性>，而不是关注<未来的决策>，就愈能创造未来重构的自由度。

例如，EIT 造形和框架的主角都是接口<I>，愈是关注<目前决策的未来性>时，就愈会想去设计通用性(General)<E>和<I>来包容未来<T>的多变化。而一群<E&I>的巧妙组合，就成为框架了。

由于 EIT 造形具有重复组合的特性，人们可以组合出多层级 EIT 造形体系的结构，进而设计出多层级的框架，就能创造更大的重构自由度。例如，上层 EIT 造形的<I>能包容用户需求<T>的未来变化；而底层框架则能包容系统平台特殊模块<T>的未来变化。用户需求与平台模块之间藉由两层 EIT 造形的通用性<I>来衔接与组合，而创造了弹性的重构空间。

迅速落实为代码

“Design is Code”依赖两项重要的代码层结构：

- EIT 造形(含类造形)
- 框架(Framework)

其中，EIT 造形成为架构思考的简单元素，然后从简单中组合出复杂架构，而框架则是产出的代码层级的架构(亦即，计算机可执行架构)。换句话说，EIT 造形能迅速落实为可执行的代码。可执行的代码有两种：框架和 App。由强龙开发框架代码；而由地头蛇开发 App 代码；这称为<强龙/地头蛇>分工模式。这就是当今 Apple 和 Google 应用商店的分工模式。EIT 造形既适用于强龙团队，也适用于地头蛇团队；也能支持这两种团队的敏捷开发过程。☆

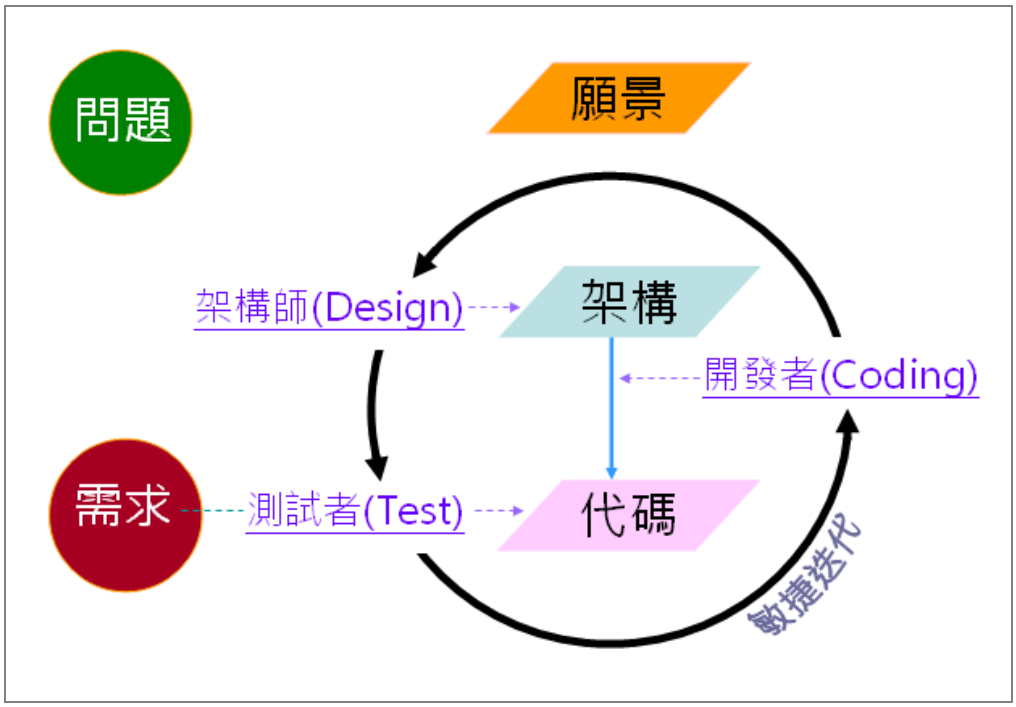
G09_接口设计之美_支持 Agile 敏捷开发

内容：

1. 一般架构设计 vs. 敏捷设计
2. 敏捷设计的主要观点
3. 架构的表述(Representation)与沟通(Communication)
4. 如何表述(Represent)”足够好的架构”呢?
5. 复习：代码造形的角色
6. 敏捷架构师：<一般架构设计>也能敏捷起来
7. 敏捷架构师+敏捷开发者
8. 敏捷架构师的素养
9. 架构师团队+敏捷迭代
10. 结语

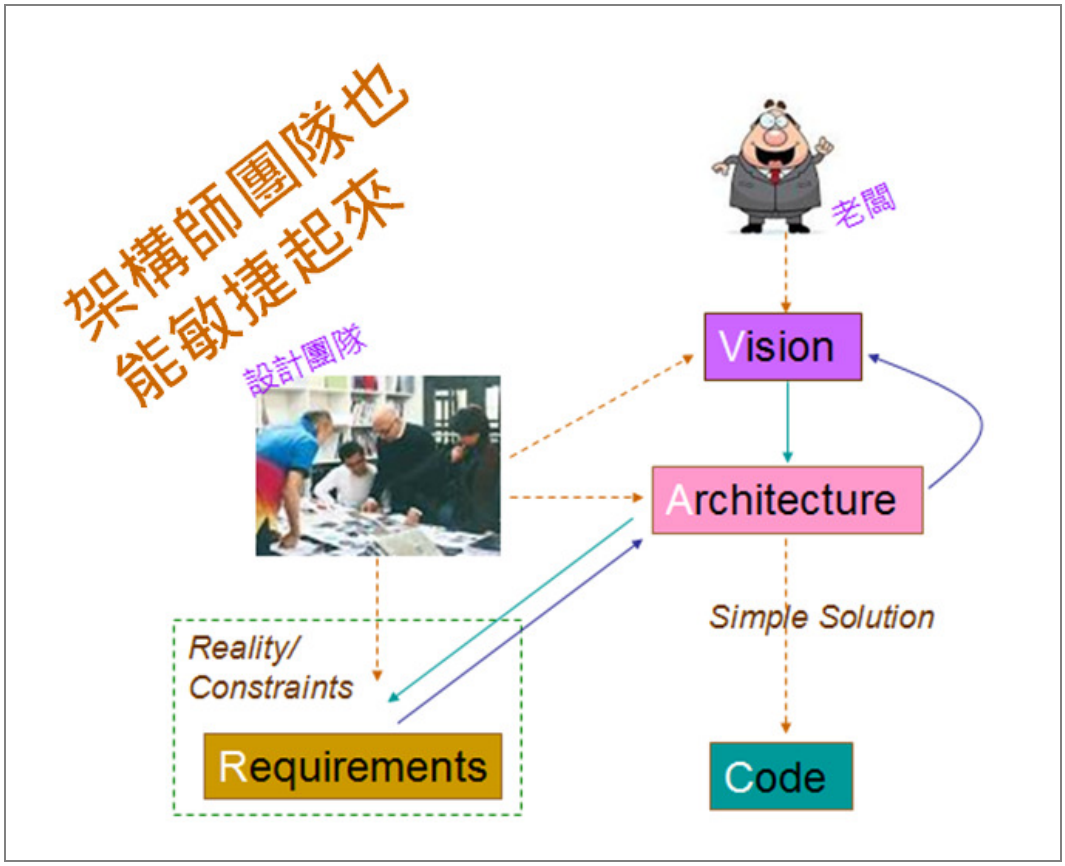
9. 架构师团队+敏捷迭代

基于愿景而设计足够好(Good Enough)架构，做为基础，尽快将设计落实成为代码；然后以需求来进行检测，将测试结果反馈回来，修正和重构设计和代码，持续迭代循环下去。



著名的敏捷专家 Fred George 说道：

- 身为架构师，我先实现系统里最艰难的部分。
  (As an architect, I will implement the most difficult parts of a system.)
- 我称它为”先验过程”，这过程检视我的设计理念是否足够好。
  (I call it "pioneering", the process where I see if an idea in my head actually is a good idea.)
- 在这先验过程的实践中，我会不断修正设计里念，直到有了感觉足够好的设计模式。然后才让开发团队跟进；这项设计模式就是系统架构了。(
  (I will always refine the idea in that first implementation. Then I feel comfortable letting the rest of the programming team follow that pattern. That is the architecture.)



Simple solution 是足够好的架构，它也是来自迭代过程(“Always refine …”)。架构师在设计出 simple solution 时，已经进行了无数次心智内的敏捷迭代了。

10. 结语

10.1 善用代码造形

无论是强龙开发框架，或者地头蛇开发 App，都可搭配敏捷开发过程，来提高效率和质量。在敏捷中，架构设计仅止于”足够好”的架构，不进行详细设计，而将详细设计转移到了开发阶段。

架构设计被切分为”足够好的架构”和”详细设计”两个部分，那么这两部分又如何相互衔接呢？其实大家都知道，架构师最关键的任务就是：接口(Interface)设计；所以”足够好架构”的核心部分就是：接口的定义(Definition)和表述(Representation)。那么，有什么有效的方法能清晰而明确地定义和表述接口设计呢？答案是：代码造形(Code Form)。

10.2 清晰表述”足够好”的架构

在敏捷中，架构设计仅止于”足够好”的架构，不进行详细设计。但详细设计仍然存在，只是转移到了开发阶段而已。代码造形能直接对映到程序语言的结构，具有高度的精确性(Precision)，架构师能准精确地传达设计的涵义。

对于架构师而言，只要能够清晰而明确地表述，即使架构尚未达到”足够好”，也能随着敏捷迭代而互相切磋着磨而达到”足够好”的境界了。由于代码造形是开发者的词汇，架构师以<代码造形>表述架构，基于共同词汇，提升了共识(Shared Understanding)，开发者很容易理解其架构的设计涵意。所以，代码造形能大幅提升开发者的效率；而且迅速配合需求变更、架构创新(或重构)设计，大幅提升了整体团队的敏捷性。

对于开发者而言，架构师就如同妈妈，使用 kid language 来与小孩交谈，非常有助于小孩语言天份的开发。同样地，架构师以<代码造形>来表述架构，来与开发者交谈；非常有助于开发者设计能力的提升。开发者从其熟悉的简单<代码造形>出发，然后像堆积木一样，逐渐培养从简单(造形)中<组合>出复杂(系统)的能力。

10.3 善用 EIT 代码造形提升敏捷性

自从 1996 年 Java 问世之后，接口(Interface)成为 Java 语言的关键词(Key Word)。于是，<接口>的位阶已经提升了，其与<类>是同位阶了，而不再隐藏于类造形里。这意味着，我们可以设计一个更大的代码造形来包容类和接口两种元素。于是，高焕堂 老师将 3 个<类造形>组合起来，成为一个更大的造形；就称为 EIT 造形。

EIT 造形的焦点在于接口<I>，而其<E>和<T>只是配角而已。系统的接口(Interface)本来就存在的，而接口的设计、定义和表述也正是架构师的核心职责。EIT 造形可以协助架构师清晰地定义接口，非常有助于清晰表达架构。

一旦架构师运用 EIT 造形来清晰而明确地表述出<I>；一方面架构师能弹性地配合敏捷跌代(Iteration)而切分系统，缩小工作范围，提高效率；另一方面，开发者能直接对映到代码语言的”Interface”关键词(Key-word)，而大幅开发者效率和敏捷性。

虽然从代码造形来看，<E>、<I>和<T>三者是同位阶的，但从架构师角度上，<I>属于主角，而<E>和<T>是配角。搭配两个配角，才能将<I>表述的完整而清晰。

由于 EIT 造形只是对类造形加以扩大；也就是以类为基础(保留了类的各项功能)，将 3 个类结合在一起，各扮演不同角色。所以，只要利用类造形既有的

组合机制，就能将 EIT 造形组合起来，成为复杂的系统了。EIT 造形提供更宏大的整体观，更易于重构，迅速从简单组合出复杂系统。

10.4 共同的感觉、一致的心灵

EIT 造形是软件产业 30 年来，有机会提升系统设计的基本模块，从类造形扩大为 EIT 造形。分析或设计师以 EIT 造形来思考设计；而开发者将设计迅速落实为代码。EIT 造形的<I>可成为团队分工的界线，强龙将<E&I>部份落实为框架代码和 API 接口；而地头蛇将<T>部份落实为 App 代码。

无论是架构师、开发者或 API 定义者，EIT 造形都是他们心中共同的感觉、一致的心灵；因而成为系统设计与实践的基本模块。

一旦架构师运用 EIT 造形来清晰而明确地表述出<I>；一方面架构师能弹性地配合敏捷跌代(Iteration)而切分系统，缩小工作范围，提高效率；另一方面，开发者能直接对映到代码语言的“Interface”关键词(Key-word)，而大幅开发者效率和敏捷性。◆

~ End ~