软件架构师是IT 行业里独一无二的职业，既要精通软件开发技术，又要掌握业务知识，还要周旋于公司不同部门之间，协调各种予盾。简洁的总结下，希望对读者有帮助！

**1．客户需求重于个人简历**   
客户需求至上。为了自己的简历更炫而采用新技术是沽名钓誉，往往事与愿违。

**2．  简化根本复杂性 ，消除偶发复杂性**   
根本复杂性指的是问题与生俱来的、无法避免的困难。偶发复杂性是人们解决根本复杂性的过程中衍生的。分析问题好比拨云见月、水落石出。架构师的责任在于解决问题的根本复杂性，同时避免引入偶发复杂性。

**3．  关键问题可能不是出在技术上**   
大多数项目是由人完成的，人才是项目成败与否的基础。学会尊重他人，给予团队成员充分的信任，是聪明的架构师获得成功必须掌握的核心技能。团队同心，其利断金。

**4．  以沟通为中心，坚持简明清晰的表达方式和开明的领导风格**   
沟通应当言简意赅、详略得当，别拖泥 带水。

**5．  架构决定性能**   
种瓜得瓜，种豆得豆，架构设计也是一 样道理。它是影响应用性能和可伸缩性的决定因素，性能参数是无法简单地通过更换软件，或者“调优”底层软件架构来改善，必须遵循分布式计算和物理学的基本原理，必须在架构的设计（或重新设计）上投入更多精力。

**6．  分析客户需求背后的意义**   
抽丝剥茧，洞见症结。不要被表面需求 迷惑。

**7．  起立发言**   
让沟通事半功倍，起立发言是最简单、有效的方法。

**8．  故障终究会发生**   
系统必然存在不同形式的故障隐患，无论如何都无法彻底消灭，应该提前设计预防措施，限制故障。

**9．  我们常常忽略了自己在谈判**     
工程师应该适时转换角色，学习谈判的 技巧。绝不能在与对方谈判的第一个要求上就妥协让步。

**10． 量化需求**   
没有规矩，不成方圆。在记录需求的过程中，对一些模糊的描述比如“灵活”，“可维护性”，“性能”等通过简单的问题来量化需求，比如“数量多少”，“有多频繁”，“不能超过多长时间”等。如果缺乏客观的标准，就只能任凭挑剔的用户摆布。

**11． 一行代码比五百行架构说明更有价值**   
可工作的代码才是目标，设计只是达成 目标手段。摩天大楼的建筑师如果一味追求美观而无视物理定律，迟早会自食其果。

**12． 不存在放之四海皆准的解决方案**   
软件世界没有放之四海皆准的解决方案。架构师必须培养和训练情境意识，才能更好地设计架构和解决问题。

**13． 提前关注性能问题**   
尽早展开性能测试。尤其是对性能要求苛刻的系统，可以验证架构和设计是否符合实际性能要求。

**14． 架构设计要平衡兼顾多方需求**   
平衡兼顾项目的技术需求和相关各方的业务需求。

**15． 草率提交任务是不负责任的行为**    （ Niclas Nilsson ）   
要设法杜绝开发人员草率提交任务的念头。

**16． 不要在一棵树上吊死**    （ Keith Braithwaite ）   
为客户提供多样化的解决方案。

**17． 业务目标至上**  （ Dave Muirhead ）   
技术决策不能脱离业务目标和现实条件的约束。

**18． 先确保解决方案简单可用，再考虑通用性和复用性**    （ Kevlin Henney ）   
 如果存在多个可实施方案难以取舍，“先简单后通用”原则可以成为最终的评判标准。通过经验提练的简单方案，远胜过不切实际的通用性。

**19． 架构师应该亲历亲为**  （ John Davies ）   
身先士卒才能赢得同事的信任。

**20． 持续集成**  （ David Bartlett ）   
持续集成确保当前的开发不会出现意外，借助它可以取得更稳定、更符合要求的开发成果。

**21． 避免进度调整失误**  （ Norman Carnovale ）   
不惜一切代价拒绝调整项目进度的要求。为提高交付速度而改变计划会带来一系列的问题。

**22． 取舍的艺术**  （ Mark Richards ）   
架构不可能满足所有需求。鱼和熊掌不可兼得，应牢记瑞典和波兰战争中瑞典瓦萨号（Vasa）战舰的故事。

**23． 打造数据库堡垒**  （ Dan Chak ）   
一开始就要定义好数据模型。数据模型的设计必须做到能够拒绝无效数据，阻止无意义的关系。

**24． 重视不确定性**  （ Kevlin Henney ）   
推迟决策，建设性地利用不确定性。推迟决定不是故意拖延，而是强调作出的决定应该基于足够的事实，不能仅凭假定和猜测。

**25． 不要轻易放过不起眼的问题**  （ Dave Quick ）   
别忘了温水煮青蛙的故事。

**26． 让大家学会复用**  （ Jeremy Meyer ）   
重复利用已有资源，首先要改变大家的观念。

**27． 架构里没有大写的“I ”**  （ Dave Quick ）   
不要让自己变成自大狂。辩解容易，难的是学会停止辩解，恃才傲物容易，难的是拥有自知之明。

**28． 使用“ 一千英尺高” 的视图**  （ Erik Doernenburg ）   
选择合适的架构视图。不能太抽象，也不能细节太多，看清整个架构。

**29． 先尝试后决策**  （ Erik Doernenburg ）   
越晚做出决策，可利用的信息就越多。可先通过尝试，对问题的最佳解决方案有了共识，再组织协调制定决策。

**30． 掌握业务领域知识**  （ Mark Richards ）   
掌握业务领域知识有助于架构师选择合适的架构模式，更好地制定针对未来的扩展计划，适应不断变化的产业趋势。

**31． 程序设计是一种设计**  （ Einar Landre ）   
软件开发也分成设计和生产两个阶段。程序设计属于设计范畴，而不是生产范畴，软件的生产则是自动化的，由编译器、构建工具和测试代码共同完成。

**32． 让开发人员自己做主**  （ Philip Nelson ）   
架构师的工作重点是仔细观察整个系统是怎样组合在一起的，确保系统良好地协调运作，应该给予团队成员足够的自主权，让他们发挥自己的创意和能力。

**33． 时间改变一切**  （ Philip Nelson ）   
选择值得投入精力的工作，漂亮的解决方案搭配正确的任务，才能经受时间的考验。别跟以前的工作过不去。回顾过去的工作，永远会觉得以前的设计和自己的期望有差距，把这些问题当作今后的工作标准，克制自己回过头去修改的冲动。

**34． 设立软件架构专业为时尚早**  （ Barry Hawkins ）

 这个自己分析吧！

**35． 控制项目规模**  （ Dave Quick ）   
分而治之，将大项目分解成独立的小项目，设置优先级，尽快交付，实现最重要的需求，尽快获得客户的反馈，越快越好。

**36． 架构师不是演员，是管家**  （ Barry Hawkins ）   
架构师的职责和管家相似，承担着管理他人资产的责任，架构师应该尽可能为客户利益着想，计算可用的时间和人力，综合考虑成本和复杂性等因素，设计出折中的解决方案，不能存有私心，卖弄时髦的软件框架和流行的技术词汇，只会把系统变得更复杂，给公司造成损失。

**37． 软件架构的道德责任**  （ Michael Nygard ）   
架构师的决定会影响许多人，务必慎重。虽然设计必填项从表面上看并无不妥之处，但实际上是架构师在强迫用户接受自己的意图。必填项迫使用户准备更多的信息，这个过程常常会耽搁工作，让人非常沮丧。

**38． 摩天大厦不可伸缩**  （ Michael Nygard ）   
但软件可以。无论是开发新项目还是替换已有的系统，都应该逐个部署系统组件，一来可以将风险分散到各个时间段，每次砌好“一块砖”，二来迫使我们设计清晰的组件间接口。

**39． 混合开发的时代已经来临**  （ Edward Garson ）   
混合编程是指在同一套软件系统中同时采用多种核心编程语言。新的技术变革正逐步瓦解我们以往积累的技术成果，架构师应拥抱这种变化，跳出原有的思维模式，充分挖掘软件的多元化特性。

**40． 性能至上**  （Craig Russell ）

性能，减少软件响应时间，提高人机交互效率！   
**41． 留意架构图里的空白区域**  （ Michael Nygard ）   
空白区域“充满”了各种软件和“硬件”。隐藏着一系列复杂的事件，应该考虑各种隐藏的细节，才能顺利地解决空白区域里的问题。

**42． 学习软件专业的行话**  （ Mark Richards ）   
架构师必须掌握基本的架构模式和设计模师，学会识别不同的模式，并借助它们和同行及开发人员进行交流。模式可分类四大类：   
企业架构模式定义架构的全局框架结构。   
应用架构模式指出了全局架构下的子系统及局部应用的设计方法。   
集成模式研究怎样在系统的组件、应用和子系统之间传递信息，共享功能。   
设计模式研究架构中每个组件的构造方法。

**43． 具体情境决定一切**  （ Edward Garson ）   
 架构不存在设计理念，具体情境决定一切，根据具体情境设计尽量简单的解决方案。

**44． 侏儒、精灵、巫师和国王**  （ Evan Cofsky ）   
开发团队不应该同质化。软件架构师好比国王，应该熟悉各种人的性格特点，招聘不同性格的人加入自己的团队，为不同性格的团队成员安排合适的任务，轻构化解各种难。由一帮性格相同的人设计的架构只能吸引同样性格的人加入团队，只能解决一类问题。

**45． 向建筑师学习**  （ Keith Braithwaite ）   
借鉴建筑行业的经验。架构师应该蕴含适当的艺术成分！

**46． 避免重复**  （ Niclas Nilsson ）   
两条公认的软件开发的真理：   
（1） 复制是魔鬼。   
（2） 重复性的工作拖累开发进度。   
消灭重复的内容是架构师的责任，为此应该重新研究框架，创造更完善的抽象机制，或者使用代码生成器。

**47． 欢迎来到现实世界**  （ Gregor Hohpe ）   
现实世界比软件世界复杂。不要抱怨现实世界中用户需求带来的麻烦，不妨从中寻找解决问题的灵感。

**48． 仔细观察，别试图控制一切**  （ Gregor Hohpe ）   
选择合适的抽象层次能提供有效的信息，也方便用基本的验证规则检查模型，架构师应仔细观察，提取模型，然后检查验证，别妄想掌控一切。

**49． 架构师好比两面神**  （ David Bartlett ）   
架构师应该像两面神一样，眼观六路、耳听八方。

**50． 架构师应关注边界和接口**   （ Einar Landre ）   
寻找自然的边界，分而治之。架构师应关注和绘制“有界情境”和“情境地图”，有界情境是用以唯一定义一个模型或概念的区域，通常以一朵云或一个气泡来表示。情境地图则包含了一系列有界情境及它们之间的接口。

**51． 助力开发团队**  （ Timothy High ）   
优秀团队是成功的保障，要尽量助力开发团队。

**52． 记录决策理由**  （ Timothy High ）   
记录架构决策背后的理由，长期有用，又无须为之付出过多精力维护，具有极高的投资回报价值。

**53． 挑战假设, 尤其是你自己的**  （ Timothy High   ）   
臆断是事情搞砸的主要根源。一定要拿出相关的经验证据，仔细验证假设，才能做出最终决策，务必要确保软件基石坚实可靠。

**54． 分享知识和经验**  （ Paul W. Homer ）   
讨论有助于发现不足，只有能非常容易地做出解释，才表明你真正理解了。只有不断解释和讨论，才能把经验凝聚成知识。帮助周围的人不断改善，他们也会帮助我们发挥出全部的潜力。

**55． 模式病**  （ Chad La Vigne ）   
不要让一展设计模式功力的欲望，遮蔽了务实的真知。使用模式解决适用的问题才是最重要的。

**56． 不要滥用架构隐喻**  （ David Ing ）   
不要耽溺于系统隐喻之中，只将之用于探索性的沟通，不要反让它拖了后腿。

**57． 关注应用程序的支持和维护**  （ Mncedisi Kasper ）   
应用程序的支持和维护，永远都不应该是事后才考虑的事情。由于应用程序超过80%的生命周期都是在维护上，在设计时就应该多多关注支持和维护的问题。考虑生产环境修误错误的压力，生产环境中的日志记录级别要比开发环境中的低很多，考虑开发人员与支持人员具有不同的技能等。

**58． 有舍才有得**   
珍惜需要权衡的时机，远胜毫无约束和限制。

**59． 原则、公理和类比胜于个人意见和口味**  （ Michael Harmer ）   
清楚的架构原则，能够使那些不熟悉某项特别技术或组件的人，明白其中的缘由，更透彻地理解他们本不熟悉的技术。

**60． 从“ 可行走骨架” 开始开发应用**  （ Clint Shank ）   
“可行走骨架”是对系统的最简单实现，它贯穿头尾，将所有主要的架构组件连接起来。从“ 可行走骨架” 开始，增量培育系统成长 。

**61． 数据是核心** （ Paul W. Homer ）   
从“数据是核心”这个角度去认识系统，能大大降低理解复杂度 。

**62． 确保简单问题有简单的解**  （Chad La Vigne ）   
    试图猜测未来的需求时，错的概率是50%，错得很离谱的概率是49%。今天只需要解决今天的问题就好。   
    把应用发布出去，从反馈中生成真实的需求。

**63． 架构师首先是开发人员**  （Mike Brown ）   
碰到麻烦时，架构师可不能只会干吹烟圈却束手无策。获得开发人员信任的最快捷方式：你的代码就是你的资本。作为架构师，主要目标应该是创建可行、可维护的解决方案，当然，也一定要能够解决当前的问题。

**64． 根据投资回报率（ROI ）进行决策** （ George Malamidis ）   
  在判断每个决策选项是否务实或恰当时，将架构决策视为投资，并将相关的回报率也一并考虑在内。

**65． 一切软件系统都是遗留系统** （ Dave Anderson ）   
软件很快便会过时，修改维护无可避免。需考虑以下几点：   
（1） 清晰性：各个组件和类的角色清晰。   
（2） 可测性：系统易于验证。   
（3） 正确性：结果和设计或期望的一致。   
（4） 可跟踪：能迅速诊断出故障并立马解决问题。

**66． 起码要有两个可选解决方案** （ Timothy High ）   
 软件架构是要在所有给定的约束条件下，寻找到解决问题的最佳方案。期望第一个解决方案即满足全部的需求和约束，几乎是不可能的。如果对手头的问题只有一个解决方案，这意味着将没有进行折衷权衡的余地。这个唯一的解决方案可能无法令系统投资方满意。

**67． 理解变化的影响**  （ Doug Crawford ）   
清楚认识变化类型及其影响。变化有多种不同的表现形式：   
（1） 功能需求的变化   
（2） 可扩展性需求的演进   
（3） 系统的接口被修改   
（4） 团队人员流动等。   
管理变化并非架构师的职责，但架构师要确保变化是可控的，有许多工具和技术可以用以减轻变化的影响。   
（1） 进行小规模的增量渐变。   
（2） 构建可重复运行的测试用例，并经常运行。   
（3） 让测试用例更易编写。   
（4） 跟踪好依赖关系。   
（5） 系统性的行动，根据反馈信息作出进一步反应。   
（6） 自动化重复的任务。

**68． 你不能不了解硬件** （ Kamal Wickramanayake ）   
硬件容量规划，是和软件架构同等重要的事情。水平伸缩能力是关键，可以在需要时才添加硬件，而无须一开始就过量采购。

**69． 现在走捷径，将来需付息** （ Scot Mcphee ）   
及时还清技术债务。可能觉得单元测试并不直接产生价值，于是就让开发人员跳过这些严格的测试工作，这将导致所交付的系统在未来更难修改，而且在修改时信心不足。 除了避免在开发初期走捷径，发现有不当的设计决策时就要尽快修正，搁置越久，为之付出的利息也将越高。

**70． 不要追求“完美”，“足够好”就行** （ Greg Nyberg ）   
避免过度设计。不要屈服于企图使设计或实现达到完美的诱惑。把目的设定在“足够好”就行，当已经达成目标时，就停下来。

**71． 小心“好主意”**  （ Greg Nyberg ）   
 “骆驼的鼻子”是 一个隐喻，指一旦允许一些不期望但很小的情况发生，后面就会招致巨大而无法避免的情况发生。“好主意”就如“骆驼的鼻子”，它将导致范围膨胀，复杂度上升，竭力把和业务需求无关的东西塞入应用中，这纯粹是浪费精力。

**72． 内容为王**  （ Zubin Wadia ）   
内容即网络，即界面。在联系日益紧密的今日世界，内容质量很快成为了成败的关键。优秀的内容指的是其内容之间互相关联，而不是空洞割裂。系统的成功取决于其内容，在设计过程中，要对内容价值的评估给予足够重视。

**73． 对商业方，架构师要避免愤世嫉俗** （ Chad La Vigne ）   
过度自信，会让你在业务领域头破血流。不要让自己成为愤世嫉俗的“天才”，总是以一副自作聪明，居高临下的语调，力求证明公司当前的运营是多么的糟糕不堪，以期触动业务方。成为优秀架构师的秘诀之中有一个关键要素，那就是要对工作满怀激情，但不要是那种带着愤怒和火气的“激情”。

**74． 拉伸关键维度，发现设计中的不足** （ Stephen Jones ）   
单独拉伸每个维度，然后综合起来看待，便可暴露出那些隐藏于最初设计中的潜在限制与不足。架构师可以通过 拉伸关键维度，对解决方案进行校核：   
（1） 了解基础设施的规划是否能够应付增长的需求，圈出限制范围。   
（2） 确认在预期的吞吐量下，系统不但能在当天内及时完成全天的任务处理，同时具备峰值储备，才能应 对“特别忙碌的日子”，才能在停电后“加班补上”.   
（3） 对当前的数据访问方案进行校核，确保其在系统伸缩扩展时依然有效。   
（4） 确保在系统工作负载上升时，（如果需要）能够以增加硬件和转换处理路径的方式进行系统的伸缩扩展。   
（5） 数据量持续上升，导致数据必须在更多的基础设施间进行分割时，要确保应用程序仍然可用。

**75． 架构师要以自己的编程能力为依托** （ Mike Brown ）   
为项目设计架构时，对实现每个设计元素所需的工作量，要做到心中有数;如果以前曾开发过其中某种设计元素，那么估算所需工作量将会容易得多。

**76． 命名要恰如其分** （ Sam Gardiner ）   
弄清楚要做的究竟是什么。设计就是要去实现各种意图，例如，快速、廉价、灵活、而名字便是用来承载和传达这些意图的。

**77． 稳定的问题可以获得高质量的解决方案** （ Sam Gardiner ）   
架构师必须从整体上看待杂乱无章的数据、概念、数据和处理逻辑，架构师要能够将它们作为整体看待，并将它们分解为更小的片段或块。这些问题块必须是稳定的，只要问题是稳定的，就可以创建出一个拥有稳定设计的系统。稳定的设计可以让你集中精力，打造出高质量的应用程序。

**78． 天道酬勤** （ Brian Hart ）   
勤奋体现在很多方面，但归根结底是指具备坚强的毅力，并且对系统的每项任务和每个架构目标，都能投入足够的精力。真正做好那些看似简单的任务，坚守承诺。很多时候，不是能力不足导致项目的失败，而是由于勤奋不够，紧迫感不强。

**79． 对决策负责** （ Yi Zhou ）   
有效的架构决策包含：   
（1） 无论是以敏捷的形式还是传统的形式做出架构决策，都必须对决策过程有充分的认识。   
（2） 要定期对架构决策进行复审;对照检查决策的实际效果和预期效果;   
（3） 要贯彻架构决策，只有全程跟进实施过程，才能够确保最到位地实现其设计意图。   
（4） 并有所谓的全能技术天才，可以将一些决策委托给相应问题域的专家。

**80． 弃聪明，求质朴** （ Eben Hewitt ）   
别去理会什么流行风潮，只有真正睿智的架构师才懂得如何保持质朴。在质朴的方案中，每个组件只做一件事。

**81． 精心选择有效技术，绝不轻易抛弃** （ Chad La Vigne ）   
架构师工作很大的一部分，是要选择用以攻克难题的合适技术。精心选择熟悉的武器，不到万不得已绝不轻易抛弃它们。同时，以审慎的态度更新你的技术武器库。

**82． 客户的客户才是你的客户！（**  Eben Hewitt ）   
假设你正在编写一个电子商务应用程序，那么该仔细考虑的应当是那些会在那个网站上购物的人的需求。

**83． 事物发展总会出人意料**  （ Peter Gillard-Moss ）   
设计是在不断变化的世界中持续进行探索试验的过程。无论研究得多么深入透彻，无论设计是如何深思熟虑，事情最后总会变得和你想的不一样，我们会发现通常无法预知的新信息。

**84． 选择彼此间能和谐共处的框架**  （ Eric Hawthorne ）   
当心“无所不能”型的框架。系统应该由多个相互独立的框架组成，其中每个框架都精专于各自的领域，但同时又非常简洁、包容和灵活。

**85． 着重强调项目的商业价值** （ Yi Zhou ）   
可通过下面五步，成功将架构提案打造为典型的商业项目：   
（1） 形成价值陈述，用以说明组织的业务为何要采用某种特定的软件架构，重点应放在说明其在提高生产力、改进业务效率方面的能力。   
（2） 建立量化的度量标准，量化得越具体越到位，项目也将越具有说服力。   
（3） 回过头来关联传统商业的衡量方式，如果能将技术分析转化为财务数据，则会更为完美。   
（4） 知道该在哪里停止。准备好一张路线图，用以捕获远景目标，清楚地知道每一个里程碑将带来的商业价值。让利益相关者自己决定在何处停止。   
（5） 寻找恰当的时机，如果时机不对，可能仍然无法成功推销你的点子。

**86． 不仅仅只控制代码，也要控制数据**  （ Chad La Vigne ）   
数据库的变化不应该影响构建活动的连续性。要把数据库也作为一个构建单元包含在内，做到一次性构建整个应用程序。

**87． 偿还技术债务**  （ Burkhardt Hufnagel ）   
在速度和架构间进行权衡，保持平衡。“脏但快速”的路线也许适合将修改快速纳入产品中，但由于“脏但快速”的修复有隐性成本，记得安排开发人员回头去妥善修复解决，纳入下一个计划发布的版本中。

**88． 不要急于求解** （ Eben Hewitt ）   
不要立即着手去解决摆在面前的问题，而要看看自己是否可以改变问题。

**89． 打造上手的系统** （ Keith Braithwaite ）   
我们构建的系统，用户体验应该是“上手的”，一定要能够帮助人们做事，这是成功的决定性因素。

**90． 找到并留住富有激情的问题解决者**  （ Chad La Vigne ）   
以正确的方式有效地经营开发团队。应确保团队具有打不垮的首发阵容，而一旦已经拥有“常胜铁军”，就要竭力维持。

**91． 软件并非真实的存在**  （ Chad La Vigne ）   
需求文档不是蓝图，软件并非真实的存在，虚拟世界中的软件是柔韧可变的。

**92． 学习新语言**  （ Burkhardt Hufnagel ）   
防止沟通不畅和误解 。架构师要能够以业务人员可理解的术语向业务人员解释其中的问题，以程序员可理解的术语向程序员转述业务上的问题。

**93． 没有永不过时的解决方案** （ Richard Monson-Haefel ）   
 今天做出的选择，在未来很大程度上会是错误的，只要选择能满足当前需求的最佳解决方案就行了。

**94． 用户接受度问题** （ Norman Carnovale ）   
减轻用户接受度问题带来的风险。人们并不总是满心欢喜地接受新系统或系统的重大升级，架构师的目标，是要去了解和衡量接受度问题带来的威胁，并朝能减轻这些威胁的方向开展工作。最有效的解决办法，是运用系统设计本身来消解个中忧虑，包括培训、定期安排系统的演示，并分享用户从新系统中将会获得的知识。

**95． 清汤的重要启示**  （ Eben Hewitt ）   
软件架构设计需要不断的精炼浓缩。反思各种构想，直至可以确定系统中每个需求的本质。

**96． 对最终用户而言，界面就是系统**  （ Vinayak Hegde ）   
最终用户通过用户界面访问系统，用户交互应和产品健壮性和性能同等重要，好的用户界面能够帮助客户提高生产力，能够帮助人们更加高效，也有利于产品自身的业务收益。

**97． 优秀软件不是构建出来的，而是培育起来的** （ Bill de hÓra ）   
要抵制试图设计出庞大完整的系统来“满足甚或超出”已知需求的特性期望的想法，要有宏伟的远景，   
但不要有庞大的设计。设计尽可能小的系统，帮助成功交付，并推动它向宏伟远景目标不断演化。