软件系统架构是一系列的抽象模式，用于指导软件系统各方面的设计。从要解决的问题出发，确定系统边界，切分系统组件，设立沟通机制，设计实现过程，最后联系组件，合并成为一个整体，输出预期的效果。

一个网站的交互风格会大大地影响系统性能。采用好的架构可以节省软件项目构建与维护的人力成本，使得每次变更都短小简单、易于实施且避免缺陷。作者在绪论中提到他工作的动机是希望理解和评估基于网络的应用的架构设计，通过有原则地使用架构约束获得期待的功能、性能和社会学几方面的属性。

一个好的架构需要具备模块化设计、可重组、高内聚、低耦合等特征。REST本身没有创造新的组件或服务，而是使用Web现有特征和能力，提供连接器语义上的架构约束，强调组件交互的可伸缩性、接口的通用性、组件的独立部署，以及用来减少交互延迟、增强安全性、封装遗留系统中的中间组件。REST既是定义架构改进的一种方法，也是识别架构不匹配的方法。超本文传输协议是REST的引擎。

Web背后的设计原理可以被描述为一种架构风格，而架构风格由应用于架构内元素的一组约束组成。REST的推导从一个空的约束集合开始。

添加客户-服务器约束，以分离用户接口和数据存储两个关注点，同时改善了用户接口的可移植性和系统组件交互的可伸缩性。可以采用按需代码风格。客户端组件从远程服务器获得处理资源的代码，在本地执行。

添加无状态约束，以请求中发送重复数据、增加交互开销的牺牲来改善可见性、可靠性和可伸缩性。

增加缓存约束，请求响应中的数据被显示或隐式地标记为可缓存或不可缓存。缓存会存在一致性的问题。使用缓存时，通常会在HTTP/1.1中添加Cache-Control、Age、Etag和Vary 几个头信息字段。

添加分层的系统约束。每一层为其上一层提供服务，且使用其下一层所提供的服务。限制组件的行为，将架构分解。

客户端、服务器、缓存等都是REST使用的连接器。它们对访问资源和转移资源表述的活动进行封装。

组件是软件指令和内部状态的一个抽象单元，通过其接口提供对于数据的转换。REST要求组件之间要有统一的接口，信息采用标准化的形式来转移。REST对信息的核心抽象是资源，通常用统一资源定位符指向它，它可以是一段文本，一张图片，也可以是一个服务。信息是自描述的。使用一个表述来捕获资源的当前的或预期的状态、在组件之间传递该表述。资源以HTML的形式来展现给用户，而资源的具体表现形式应该在HTTP协议中的Content-Type字段和Accept字段中被描述。REST规定了客户端操作资源的几个动词，即HTTP协议中的Method，包括GET、POST、PUT、DELETE。所有访问方法对资源拥有相同的语义。

基于REST的基本原理，Web开发者能够设计并建造出最为高效的Web应用。