通用的软件概要设计说明书国家有标准。例如概要设计说明书（GB8567-88）包括如下内容：

1 引言
1.1 编写目的
1.2 背景
1.3 定义
1.4 参考资料
2 总体设计
2.1 需求规定
2.2 运行环境
2.3 基本设计概念和处理流程
2.4 结构
2.5 功能器求与程序的关系
2.6 人工处理过程
2.7 尚未问决的问题
3 接口设计
3.1 用户接口
3.2 外部接口
3.3 内部接口
4 运行设计
4.1 运行模块组合
4.2 运行控制
4.3 运行时间
5 系统数据结构设计
5.1 逻辑结构设计要点
5.2 物理结构设计要点
5.3 数据结构与程序的关系
6 系统出错处理设计
6.1 出错信息
6.2 补救措施
6.3 系统维护设计

发表者付：
做软件到一定层次了，就要考虑到设计了，设计了很久，就是不系统，系统的设计需要一个记录，记录就用文档，那么对项目所有包括技术上的设计都记录下来，我们就可以理解为软件的概要设计了。

在需求明确、准备开始编码之前，要做概要设计，而详细设计可能大部分公司没有做，有做的也大部分是和编码同步进行，或者在编码之后。因此，对大部分的公司来说，概要设计文档是唯一的设计文档，对后面的开发、测试、实施、维护工作起到关键性的影响。

一、问题的提出

概要设计写什么？概要设计怎么做？

如何判断设计的模块是完整的？

为什么说设计阶段过于重视业务流程是个误区？

以需求分析文档还是以概要设计文档来评估开发工作量、指导开发计划准确？

结构化好还是面向对象好？

以上问题的答案请在文章中找。

二、概要设计的目的

将软件系统需求转换为未来系统的设计；

逐步开发强壮的系统构架；

使设计适合于实施环境，为提高性能而进行设计；

结构应该被分解为模块和库。

三、概要设计的任务

制定规范：代码体系、接口规约、命名规则。这是项目小组今后共同作战的基础，有了开发规范和程序模块之间和项目成员彼此之间的接口规则、方式方法，大家就有了共同的工作语言、共同的工作平台，使整个软件开发工作可以协调有序地进行。

总体结构设计：

功能（加工）—>模块：每个功能用那些模块实现，保证每个功能都有相应的模块来实现；

模块层次结构：某个角度的软件框架视图；

模块间的调用关系：模块间的接口的总体描述；

模块间的接口：传递的信息及其结构；

处理方式设计：满足功能和性能的算法

用户界面设计；

数据结构设计：

详细的数据结构：表、索引、文件；

算法相关逻辑数据结构及其操作；

上述操作的程序模块说明（在前台？在后台？用视图？用过程？・・・・・・）

接口控制表的数据结构和使用规则

其他性能设计。

四、概要设计写什么

结构化软件设计说明书结构（因篇幅有限和过时嫌疑，在此不作过多解释）

任务：目标、环境、需求、局限；

总体设计：处理流程、总体结构与模块、功能与模块的关系；

接口设计：总体说明外部用户、软、硬件接口；内部模块间接口（注：接口≈系统界面）

数据结构：逻辑结构、物理结构，与程序结构的关系；

模块设计：每个模块“做什么”、简要说明“怎么做”（输入、输出、处理逻辑、与其它模块的接口，与其它系统或硬件的接口），处在什么逻辑位置、物理位置；

运行设计：运行模块组合、控制、时间；

出错设计：出错信息、处错处理；

其他设计：保密、维护；

OO 软件设计说明书结构

1 概述

系统简述、软件设计目标、参考资料、修订版本记录

这部分论述整个系统的设计目标，明确地说明哪些功能是系统决定实现而哪些时不准备实现的。同时，对于非功能性的需求例如性能、可用性等，亦需提及。需求规格说明书对于这部分的内容来说是很重要的参考，看看其中明确了的功能性以及非功能性的需求。

这部分必须说清楚设计的全貌如何，务必使读者看后知道将实现的系统有什么特点和功能。在随后的文档部分，将解释设计是怎么来实现这些的。

2 术语表

对本文档中所使用的各种术语进行说明。如果一些术语在需求规格说明书中已经说明过了，此处不用再重复，可以指引读者参考需求说明。

3 用例

此处要求系统用用例图表述（UML），对每个用例（正常处理的情况）要有中文叙述。

4 设计概述

4.1 简述

这部分要求突出整个设计所采用的方法（是面向对象设计还是结构化设计）、系统的体系结构（例如客户/服务器结构）以及使用到的相应技术和工具（例如 OMT、Rose）

4.2 系统结构设计

这部分要求提供高层系统结构（顶层系统结构、各子系统结构）的描述，使用方框图来显示主要的组件及组件间的交互。最好是把逻辑结构同物理结构分离，对前者进行描述。别忘了说明图中用到的俗语和符号。

4.3 系统界面

各种提供给用户的界面以及外部系统在此处要予以说明。如果在需求规格说明书中已经对用户界面有了叙述，此处不用再重复，可以指引读者参考需求说明。如果系统提供了对其它系统的接口，比如说从其它软件系统导入/导出数据，必须在此说明。

4.4 约束和假定

描述系统设计中最主要的约束，这些是由客户强制要求并在需求说明书写明的。说明系统是如何来适应这些约束的。

另外如果本系统跟其它外部系统交互或者依赖其它外部系统提供一些功能辅

助，那么系统可能还受到其它的约束。这种情况下，要求清楚地描述与本系统有交互的软件类型以及这样导致的约束。

实现的语言和平台也会对系统有约束，同样在此予以说明。

对于因选择具体的设计实现而导致对系统的约束，简要地描述你的想法思路，经过怎么样的权衡，为什么要采取这样的设计等等。

5 对象模型

提供整个系统的对象模型，如果模型过大，按照可行的标准把它划分成小块，例如可以把客户端和服务器端的对象模型分开成两个图表述。在其中应该包含所有的系统对象。这些对象都是从理解需求后得到的。要明确哪些应该、哪些不应该被放进图中。所有对象之间的关联必须被确定并且必须指明联系的基数。聚合和继承关系必须清楚地确定下来。每个图必须附有简单的说明。

6 对象描述

在这个部分叙述每个对象的细节，它的属性、它的方法。在这之前必须从逻辑上对对象进行组织。你可能需要用结构图把对象按子系统划分好。

为每个对象做一个条目。在系统对象模型中简要的描述它的用途、约束（如只能有一个实例），列出它的属性和方法。如果对象是存储在持久的数据容器中，标明它是持久对象，否则说明它是个临时对象（transient object）。

对每个对象的每个属性详细说明：名字、类型，如果属性不是很直观或者有约束（例如，每个对象的该属性必须有一个唯一的值或者值域是有限正整数等）。

对每个对象的每个方法详细说明：方法名，返回类型，返回值，参数，用途以及使用的算法的简要说明（如果不是特别简单的话）。如果对变量或者返回值由什么假定的话，Pre-conditions 和 Post-conditions 必须在此说明。列出它或者被它调用的方法需要访问或者修改的属性。最后，提供可以验证实现方法的测试案例。

7 动态模型

这部分的作用是描述系统如何响应各种事件。一般使用顺序图和状态图。

确定不同的场景（Scenario）是第一步，不需要确定所有可能的场景，但是必须至少要覆盖典型的系统用例。不要自己去想当然地创造场景，通常的策略是描述那些客户可以感受得到的场景。

7.1 场景（Scenarios）

对每个场景做一则条目，包括以下内容：

场景名：给它一个可以望文生义的名字

场景描述：简要叙述场景是干什么的以及发生的动作的顺序。

顺序图：描述各种事件及事件发生的相对时间顺序。

7.2 状态图

这部分的内容包括系统动态模型重要的部分的状态图。可能你想为每个对象画一个状态图，但事实上会导致太多不期望的细节信息，只需要确定系统中一些重要的对象并为之提供状态图即可。

8 非功能性需求

五、概要设计怎么做

结构化软件设计方法：

详细阅读需求规格说明书，理解系统建设目标、业务现状、现有系统、客户需求的各功能说明；

分析数据流图，弄清数据流加工的过程；

根据数据流图决定数据处理问题的类型（变换型、事务型、其他型）；

通过以上分析，推导出系统的初始结构图；

对初始结构图进行改进完善：所有的加工都要能对应到相应模块（模块的完整性在于他们完成了需求中的所有加工），消除完全相似或局部相似的重复功能（智者察同），理清模块间的层次、控制关系，减少高扇出结构，随着深度增大扇入，平衡模块大小。

由对数据字典的修改补充完善，导出逻辑数据结构，导出每种数据结构上的操作，这些操作应当属于某个模块。

确定系统包含哪些应用服务系统、客户端、数据库管理系统；

确定每个模块放在哪个应用服务器或客户端的哪个目录、哪个文件（库），或是在数据库内部建立的对象。

对每个筛选后的模块进行列表说明。

对逻辑数据结构进行列表说明。

根据结构化软件设计说明书结构对其他需要说明的问题进行补充说明，形成概要设计说明书。

OO 软件设计方法：

在 OOA 基础上设计对象与类：在问题领域分析（业务建模和需求分析）之后，开始建立系统构架。

第一步是抽取建立领域的概念模型，在 UML 中表现为建立对象类图、活动图和交互图。对象类就是从对象中经过“察同”找出某组对象之间的共同特征而形成类：

对象与类的属性：数据结构；

对象与类的服务操作：操作的实现算法；

对象与类的各外部联系的实现结构；

设计策略：充分利用现有的类；

方法：继承、复用、演化；

活动图用于定义工作流，主要说明工作流的 5W（Do What、Who Do、When Do、Where Do、Why Do）等问题，交互图把人员和业务联系在一起是为了理解交互过程，发现业务工作流中相互交互的各种角色。

第二步是构建完善系统结构：对系统进行分解，将大系统分解为若干子系统，子系统分解为若干软件组件，并说明子系统之间的静态和动态接口，每个子系统可以由用例模型、分析模型、设计模型、测试模型表示。软件系统结构的两种方式：层次、块状

层次结构：系统、子系统、模块、组件（同一层之间具有独立性）；

块状结构：相互之间弱耦合

系统的组成部分：

问题论域：业务相关类和对象(OOA 的重点)；

人机界面：窗口、菜单、按钮、命令等等；

数据管理：数据管理方法、逻辑物理结构、操作对象类；

任务管理：任务协调和管理进程；

第三步是利用“4＋1”视图描述系统架构：用例视图及剧本；说明体系结构的设计视图；以模块形式组成包和层包含概要实现模型的实现视图；说明进程与线程及其架构、分配和相互交互关系的过程视图；说明系统在操作平台上的物理节点和其上的任务分配的配置视图。在 RUP 中还有可选的数据视图。

第四步是性能优化（速度、资源、内存）、模型清晰化、简单化（简单就是享受）。

六、概要设计的原则

总体原则和方法：由粗到细的原则，互相结合的原则，定性分析和定量分析相结合的方法，分解和协调的方法和模型化方法。

要系统考虑系统的一般性、关联性、整体性和层次性。

分解协调：目的是为了创造更好的系统。系统分解是指将一个复杂的系统分解为若干个子系统，系统协调一是系统内协调，即根据系统的总结构、总功能、总任务和总目标的要求，使各个子系统之间互相协调配合，在各个子系统局部优化基础上，通过内部平衡的协调控制，实现系统的整体优化；

屏蔽抽象：从简单的框架开始，隐含细节；

一致性：统一的规范、统一的标准、统一的文件模式；

每个模块应当有一个统一命名的容易理解的名字；

编码：由外向内（界面—>核心）；

面向用户：概要设计是对于按钮按下后系统“怎么做”的简要说明；

模块、组件的充分独立性、封闭性；

同时考虑静态结构与动态运行；

每个逻辑对象都应当说明其所处物理对象（非一一对应）；

每个物理对象都有合适的开发人员，并且利于分工与组装。（详细说明见本人另一篇文章：系统构架设计应考虑的因素）；

确立每个构架视图的整体结构：视图的详细组织结构、元素的分组以及这些主要分组之间的接口；

软件构架与使用的技术平台密切相关，目前常用的平台有 J2EE、.NET、CORBA 等等，因此具体的软件构架人员应当具备使用这些平台的软件开发经验；

通过需求功能与设计模块之间的列表对应，检查每个需求功能是否都有相应的模块来实现，保证需求功能的可追溯性和需求实现（模块）的完整性，同时可以检查重复和不必要的模块。

在需求调研分析过程中对业务处理过程了解的完整性和准确性非常重要。调查了解清楚所有的业务流程才能设计出适合各流程业务节点用户业务特点和习惯的软件，使开发出来的软件更受欢迎。当然在进行软件概要设计时，要尽量排除业务流程的制约，即把流程中的各项业务结点工作作为独立的对象，设计成独立的模块，充分考虑他们与其他各种业务对象模块的接口，在流程之间通过业务对象模块的相互调用实现各种业务，这样，在业务流程发生有限的变化时（每个业务模块本身的业务逻辑没有变的情况下），就能够比较方便地修改系统程序模块间的调用关系而实现新的需求。如果这种调用关系被设计成存储在配置库的数据字典里，则连程序代码都不用修改，只需修改数据字典里的模块调用规则即可。

七、概要设计的重要输出

编码规范：信息形式、接口规约、命名规则；

物理模型：组件图、配置图；

不同角度的构架视图：用例视图、逻辑视图、进程视图、部署视图、实施视图、数据视图（可选）；

系统总体布局：哪些部分组成、各部分在物理上、逻辑上的相互关系；

两个不可忽视的输出：

与需求功能的关系：对于需求中的每一个功能，用哪一层、哪个模块、哪个类、

哪个对象来实现（一对多关系）；反过来，应当说明将要创建的系统每一层、每个模块、每个对象、每一个类“做什么”，他们是为了帮助实现哪些功能（一对多关系）。（需求的颗粒度在一开始往往是比较粗的，因此根据功能点对于整体项目规模的估计或得到项目 WBS 其误差范围也是比较大的。更为重要的原因是，需求往往不是编码工作分解的准确依据，因为一个需求的功能点可能对应多个代码模块，而多个需求的功能点也可能只对应一个或少数代码模块，同时还有软件复用等因素要考虑，因此只有在概要设计完成以后才能准确地得到详细设计或编码阶段的二次 WBS，并估计较为准确的整体项目规模。）

逻辑与物理位置：每个对象在逻辑上分别落在哪一层、哪个模块、哪个类；在物理上每个模块、每个对象、每一个类放在哪个应用服务器或客户端的哪个目录、哪个文件（库），或者是建立在数据库管理系统中的什么东东（过程、函数、视图、触发器等等）。

八、结构化与面向对象方法特点比较

1．从概念方面看，结构化软件是功能的集合，通过模块以及模块和模块之间的分层调用关系实现；面向对象软件是事物的集合，通过对象以及对象和对象之间的通讯联系实现；

2．从构成方面看，结构化软件＝过程＋数据，以过程为中心；面向对象软件＝（数据＋相应操作）的封装，以数据为中心；

3．从运行控制方面看，结构化软件采用顺序处理方式，由过程驱动控制；面向对象软件采用交互式、并行处理方式，由消息驱动控制；

4．从开发方面看，结构化方法的工作重点是设计；面向对象方法的工作重点是分析；但是，在结构化方法中，分析阶段和设计阶段采用了不相吻合的表达方式，需要把在分析阶段采用的具有网络特征的数据流图转换为设计阶段采用的具有分层特征的结构图，在面向对象方法中则不存在这一问题。

5．从应用方面看，相对而言，结构化方法更加适合数据类型比较简单的数值计算和数据统计管理软件的开发；面向对象方法更加适合大型复杂的人机交互式软件和数据统计管理软件的开发；

参考文献：

《实用软件工程》第二版，郑人杰、殷人昆、陶永雷等著
《微软项目：求生法则》Steve McConnell 著，余孟学译
《软件工程：实践者的研究方法》（第 5 版）Roger S.Pressman 著
《软件构架实践》SEI 软件工程译丛，林·巴斯著
《RUP2000》电子版；
《UML 与系统分析设计》张龙祥著；
《面向对象的分析与设计》杨正甫著；