计算机科学课程体系规范 2013

(Computer Science Curricula 2013)

ACM 和 IEEE 计算机学会　著

ACM 中国教育委员会、教育部计算机类专业教学指导委员会　译

高等教育出版社·北京

前　言

为了方便中国大陆的计算机相关专业的院系和教师及时了解、学习国外计算机类专业本科培养方案与课程设置的最新成果，ACM 中国教育委员会决定长期开展引进和翻译国外该方面成果的工作。

2014 年 7 月 ACM 中国教育委员会与中国教育部计算机类专业教学指导委员会就 ACM Computer Science Curricula 2013（CS2013）的翻译开展合作，计划在 2014 年末完成此项工作。

在 ACM 中国教育委员会主席张铭教授（北京大学）和计算机类专业教学指导委员会秘书长马殿富教授（北京航空航天大学）的领导下，组织国内的相关教师成立了工作组开展此项工作，教学指导委员会负责初稿翻译，ACM 中国教育委员会负责校对和审核。工作组秘书由李波（西安交通大学）担任。参与人员的分工如表 1 所示。

章节	翻译人	校对审核人
前言及目录 Chapter 1：Introduction Chapter 2：Principles Chapter 3：Characteristics of Graduates	王晓阳 复旦大学	张龙 高等教育出版社
Chapter 4：Introduction to the Body of Knowledge Chapter 5：Introductory Courses Chapter 6：Institutional Challenges	胡斌 兰州大学	李波 西安交通大学
Appendix A：The Body of Knowledge Algorithms and Complexity（AL） Architecture and Organization（AR） Computational Science（CN）	陈思红 中国科学技术大学	张铭 北京大学

续表

章节	翻译人	校对审核人
Discrete Structures (DS) Graphics and Visualization (GV) Human – Computer Interaction (HCI)	胡事民 清华大学	陈文智 浙江大学
Information Assurance and Security (IAS) Information Management (IM) Intelligent Systems (IS)	王万良 浙江工业大学	刘卫东 清华大学
Networking and Communication (NC) Operating Systems (OS) Platform – Based Development (PBD)	罗军舟 东南大学	刘琴 同济大学
Parallel and Distributed Computing (PD) Programming Languages (PL) Software Development Fundamentals (SDF)	谭国真 大连理工大学	吴文俊 北京航空航天大学
Software Engineering (SE) Systems Fundamentals (SF) Social Issues and Professional Practice (SP)	陈越 浙江大学	秦征 英特尔中国

表 1　参与人员分工

参与的教师较为深入地了解了美国同行在培养方案研究和设计方面的动机、思路、方法和方案;也体会到了中、美两国在教学理念和培养要求方面的细微差别。此项工作一定会在中国的院校培养方案修订及教学改革方面起到很好的作用。

在所有参与者的共同努力下,此项工作顺利完成,在此表示由衷的感谢。此项工作还得到了 ACM 计算机学会、ACM 中国的鼎力支持,在此也表示衷心的感谢。由于中、美两国在教学用词方面的差异,加上时间匆忙和经验不足,翻译难免有错,欢迎大家批评指正。

译者

2015 年 1 月

目　录

第1章 引 言

早在40多年前,ACM和IEEE计算机学会就开始致力于计算领域国际化本科课程体系的设计(参见[1]),并大致每十年对课程体系进行一次修改,本卷代表了这一系列努力的最新成果。计算领域不断成长并趋于多样化,课程体系必然随之改变,使得今天整个计算领域的课程体系除含有计算机科学(Computer Science,参见[3])课程体系外,还包括计算机工程(Computer Engineering),信息系统(Information Systems),信息技术(Information Technology)和软件工程(Software Engineering)等课程体系。为保持计算领域课程体系的现代化并与现实紧密相关,所有这些课程体系都需定期更新。上一个完整的计算机科学课程体系于2001年公布(CC2001[2]),之后于2008年进行了中期评估(CS2008[4])。

此次修改针对计算机科学基本要素进行了重新思考,进而对整个计算机科学知识体(Body of Knowledgl)做了重新定义,以此对过去的课程体系进行全面的修正,结果便是称作CS2013的本卷。本卷还给出一些实际课程的示范案例,用于应对各类教育机构的不同情况,在课程结构及专业设置方面提供一些具体的指导。

由于计算机科学自身日新月异的变化及其在多个方向上的快速扩张,其课程体系的设计工作一向具有很强的挑战性,特别是既要面对日益多样化的与计算机科学相关的主题,又要面对计算领域与其他学科的日益融合。编制者需要有很好的平衡技巧才能在顾及计算领域知识的增长对内容扩展要求的同时,建立起切实可行的本科课程体系。为此,CS2013指导委员会做了很大的努力,与计算机科学教育界人士进行了最广泛的交流,更好地了解计算机教育的新机遇及特定需求并对现有的及创新中的课程模式进行肯定。

宪章

ACM和IEEE计算机学会在CS2013工作中遵循下述指导原则:审阅ACM和IEEE计算机学会CC2001课程体系以及中期评估CS2008,根据本学科的最新发展,制定具有持久影响力的2013年修订增强版。

CS2013 工作小组需极力争取计算机科学领域内人员的广泛参与,获取各类人士的意见和建议。CS2013 也需寻求成为一个国际化的体系,使之适用于广泛多样的教育机构,提供课程体系和教学方面的指导。在报告的写作过程中,还需多次征询公众意见并给公众提供监督的机会。

本卷的产生过程遵循了此宪章。

CS2013 编写过程概述

2010 年的下半年,ACM 和 IEEE 计算机学会分别委任了 CS2013 指导委员会联席主席,由联席主席招募指导委员会其他成员。2010 年秋,指导委员会接受了 CS2013 宪章,并开始对计算机科学系系主任们进行了调研工作(见下述)。指导委员会于 2011 年 2 月第一次开会,开始把工作重点放在修改计算机科学的"知识体"(Body of Knowledge,或简称 BoK)上。选择这样的起始点是因为 CS2008 报告以及对计算机科学系系主任的调查结果均指出,需要在 BoK 中增加新的知识领域。

在撰写本卷的整个过程中,指导委员会大致每 6 个月召开一次面对面会议,期间穿插一月一次的电话会议。在确定了新的 BoK 后,指导委员会为每一个知识领域(Knowledge Area,简称 KA)设立一个分委员会,由三位指导委员会成员参与,其中一位任分委员会主席,并由分委员会主席挑选领域内其他专家参加分委会工作。在分委员会对其负责的 KA 撰写文稿的过程中,分委员会委员们会在各类学术会议上进行宣讲,同时直接寻求相关专家的建议,以求广泛征求反馈意见。指导委员会还通过网络方式收集社会各界的意见。作为 CS2013 指导委员会成员的分委员会主席们有一项重要工作是协同合作来解决 KA 之间的冲突,消除冗余,完成适当的分类,并在各 KA 之间进行交叉引用。由此可见,在凝练 CS2013 知识体的过程中,不仅有 CS2013 指导委员会的努力,计算机科学界同行们的广泛参与更是起到了显著作用。为期两年的撰写,最终形成了本卷所呈现的知识体。

从 2012 年夏季会议开始,指导委员会的重点转向了示范课程和课程体系模版。在这方面,广泛的参与再次成为关健,所提供的范例组成了本卷范例部分,呈现在附录 C 中。

调研回馈

为了给 CS2013 奠定一个良好的基础,指导委员会对 CC2001 和 CS2008 的使用情况展开了调研,向在美国国内约 1 500 个计算机科学(和相关专业)系的系主任及本科教学负责人,以及美国以外的 2 000 个系主任,发出调研请求。共收到 201 个回复,回复代表了较广泛的教育机构(教育机构的性质由回复人自我认定):

- 研究型大学(55%)
- 教学型大学(17.5%)
- 本科院校(22.5%)
- 社区大学(5%)

回复调研的各教学机构在规模上也有较好的覆盖性:

- 不到 1 000 名学生(6.5%)
- 1 000 ~ 5 000 的学生(30%)
- 5 000 至 10 000 名学生(19%)
- 超过 10 000 名学生(44.5%)

在回答他们是如何使用 CC2001/CS2008 课程体系报告时,被调查者普遍提到使用了知识体,即他们最经常查看的是此报告中计算机科学本科专业课程体系的知识点提纲。而当被问到什么样的知识点应该加入知识体中时,被调查者一致提到非常有必要考虑“信息安全”及“并行与分布计算”。事实上,在撰写 CS2008 报告时也做过类似调查,而被调查者的反馈意见也是希望把这两个知识点加进来,但当时 CS2008 的指导委员会认为, CS2008 是个中期评估报告,不考虑增加新的知识领域,他们把这个任务推给下一次对课程体系进行完整修改时再考虑。所以这次 CS2013 报告确实增加了一些新的 KA,其中包含“信息保障与安全”及“并行与分布式计算”。

宏观原则

CS2013 报告的撰写是在以下几个原则性指导意见下进行的(详见下一章中的介绍):

- 视计算机科学为一枚“大帐篷”(big tent)。现实情况下的计算机专业越来越多地涉及跨学科工作,比如“计算生物学”(computational biology),“计算工

程学”(computational engineering),以及各种“计算 X 学”。以一个外向型的态度看待计算机科学非常重要,作为一门学科,计算机科学应该积极寻求合作机会,与其他学科融合。

- 控制课程体系的规模。虽然计算机科学的领域迅速扩大而且势头还在继续,但课程体系的规模却不可能按同样趋势随之扩张。鉴于此,CS2013 必须重新审视计算机科学的基本内容,为新增的知识点腾出空间,而不是让教学机构在设计其计算机科学专业时必须拿出比 CS2008 所要求的更多的总学时数。同时,是要在计算机科学本质方面的教学不失其严谨性的前提下,提倡更灵活的课程设置。
- 用实例说话。当年,CS2001 的撰写者给自己提出了一个不小的挑战,即将其定义的所有知识领域融会贯通地设计进六个课程体系模板和 47 门示范课程中,并进行详细地说明。这番努力很有勇气,但回想起来,这种以“定制课程”为引导的方法,似乎在各教学机构的课程设计实践中没有起到太大的作用。CS2013 采用了一个不同的方法:收集了已经取得成功的课程及课程体系,并用它们来描绘各相关知识单元是怎样贯穿在实际教学中的。
- 以教学机构的需求为导向。CS2013 希望能适用于世界各地有着各种文化背景的教学机构,为此必须认识到不同的教学机构有着不同的需求和目标,以及其资源方面的局限。由此,CS2013 以一种层次化的形式对知识体进行布局,核心一级(Core – Tier1)内是对计算机科学而言至关重要的知识点内容,其体量较小,而其他知识点内容都设置在核心二级(Core – Tier2)内。所有的计算机科学课程体系都应该包含核心一级内的内容,但可以灵活选择核心二级内的知识点。本卷第 4 章对此有更详细的描述。

知识领域

CS2013 的知识本体由 18 个知识领域(KA)组成,对应计算领域中的 18 个研究专题。这 18 个知识领域为:

- AL – 算法与复杂度(Algorithms and Complexity)
- AR – 体系结构与组织(Architecture and Organization)
- CN – 计算科学(Computational Science)
- DS – 离散结构(Discrete Structures)
- GV – 图形学与可视化(Graphics and Visualization)
- HCI – 人机交互(Human – Computer Interaction)

- IAS – 信息保障与安全(Information Assurance and Security)
- IM – 信息管理(Information Management)
- IS – 智能系统(Intelligent Systems)
- NC – 网络与通信(Networking and Communications)
- OS – 操作系统(Operating Systems)
- PBD – 基于平台的开发(Platform – based Development)
- PD – 并行与分布式计算(Parallel and Distributed Computing)
- PL – 程序设计语言(Programming Languages)
- SDF – 软件开发基础(Software Development Fundamentals)
- SE – 软件工程(Software Engineering)
- SF – 系统基础(Systems Fundamentals)
- SP – 社会问题与专业实践(Social Issues and Professional Practice)

上述 CS2013 知识领域(KA)有三个出处。第一是由修订 CC2001/ CS2008 中的 KA 而成,当然其中有些经过了相当大的修改。第二是 CC2001 发布以来新增的 KA,由于这些 KA 已经成为计算领域不可分割的部分,故此列入 CS2013。比如“信息保障与安全”领域,由于在过去的 10 年中计算机和网络安全显得越来越重要,故此将其列入 CS2013 中的新增领域。第三是对 CC2001/CS2008 中的 KA 通过重构形成,使其与当代实际情形更为贴切。例如,“软件开发基础”(SDF)领域就是从“程序设计基础”、“软件工程”、“程序设计语言”与“算法与复杂度”等领域中抽取与软件开发有关的基本知识和技能聚集而成。类似的例子还有“系统基础”(SF)领域,此知识领域汇集了分布在多个知识领域中具有基础性、跨领域性的系统概念,为在更高层次和更多方面进行系统层面的工作奠定基础。

对于读者来说,一个很重要的方面是要认识到知识领域的相互关联性,一个 KA 概念可能是建立在另外 KA 之上的,也可能是对另外 KA 的一个补充。读者应把知识体作为一个整体来阅读,而不是孤立地去专注某个特定的知识领域。本卷第 4 章对 KA 有较完整的论述,也描述了本卷新增 KA 的产生动机。

专业实践

计算机科学专业本科生所接受的教育必须为他们今后的工作提供全面而充分的准备,而不是简单的技术细节灌输。事实上,软技能(如团队合作、口头及书面沟通、时间管理、问题解决能力、适应性等)及个人素质(如风险承受能力、

合议共治能力、耐心程度、职业道德、机会识别、社会责任感、对多元化习惯及背景的理解等)在工作场所中将起到关键作用。要想成功地将技术知识运用在实践中,工作者往往需要容忍事物的模糊性,能很好地与不同背景和学科的人一起讨论和工作。在各种职业发展道路上,要想成功地实践专业技能,这些软技能和个人素质都是要最先考虑并会影响一切的重要方面。

学生们一部分的软技能和个人素质(比如耐心,时间管理,职业道德,对多元化的理解等)将通过大学的学习生活经历而获得,而其他部分则可能需要通过具体的课程教育来获得。CS2013 包含了一些课程实例,介绍怎样在本科计算机科学课程中鼓励学生软技能和个人素质的发展。比如,在软件工程(SE)这个知识领域中,项目管理课程的主要课时内就涉及团队精神和风险管理,需求工程课程的一个核心内容就是容忍模糊性的能力。在社会问题与专业实践(SP)这个知识领域中,专业交流部分的一个核心是书面和口头沟通能力;同样在社会问题与专业实践(SP)这个知识领域中,社会背景方面的主要课时内也包含帮助学生们理解关注社会责任所带来影响的内容。终身学习以及专业发展的重要性在社会问题与专业实践(SP)的序言中有所述及,并在本卷第 2 章(原则)和第 3 章(毕业生特征)中有所阐述。

课程体系与课程的样本

CS2013 里包括了实际使用过的、来自不同类型的大学及学院的课程样本,这些样本是用来说明知识领域的各个知识点可以用不同的方式加以涵盖。本卷还以几个教学机构的课程体系为例,展示把多个课程聚集成一个完整课程体系的方法。更重要的是,我们认为对课程和课程体系的介绍,可以促进计算机界内教育理念的共享,同时也鼓励计算机教育工作者能在更广泛的群体内分享自己所属教育机构(或他们熟悉的教育机构)的课程和课程体系。

社区参与和网站

CS2013 得益于计算机界内众多成员的广泛参与,他们审阅了本卷不断更新的各个修改版草案,提出了批评和建议。除指导委员会外,有超过 100 人为本卷的成形提供了有益的指导。CS2013 编写过程的更多信息可在 CS2013 网站上查询:http://cs2013.org

致谢

CS2013 的编写过程得益于许多人,包括:亚历克斯·艾肯(斯坦福大学),珍妮·阿尔布雷希特(威廉姆斯学院),罗斯·安德森(剑桥大学),佛罗伦萨·阿佩尔(圣泽维尔大学),海伦·阿姆斯特朗(科廷大学),科林·阿姆斯特朗(科廷大学),科斯特·阿散诺维奇(加州大学伯克利分校),拉杜楼·巴比切努(阿肯色大学小石城分校),杜安·贝利(威廉姆斯学院),道格·鲍德温(纽约州立大学杰纳西奥分校),迈克·巴克(麻省理工学院),迈克尔·巴克(奈良科学与技术研究所),保罗·比姆(华盛顿大学),罗伯特·贝克(维拉诺瓦大学),马特·毕晓普(加州大学戴维斯分校),艾伦·布莱克威尔(剑桥大学),唐·布拉依塔(朗伍德大学),奥利维尔·博纳凡特(鲁汶天主教大学),罗杰·博伊尔(利兹大学),克来·布来希尔斯(英特尔),博·布林克曼(迈阿密大学),大卫·布罗门(林雪平大学),迪克·布朗(圣奥拉夫学院),金·布鲁斯(波莫纳学院),乔纳森·巴斯(滑铁卢大学),内提娃·卡夫托里(东北伊利诺伊大学),保罗·凯恩斯(约克大学),艾莉森·可里亚(基督城理工学院),科特·克利夫顿(罗斯霍曼和奥姆尼集团),伊冯·科迪(维多利亚大学),史蒂夫·库珀(斯坦福大学),托尼·靠林(谢菲尔德大学),乔伊斯·柯里 – 列托(陶森大学),罗恩·塞托龙(圣路易斯华盛顿大学),梅丽莎·达克(普渡大学),珍妮特·戴维斯(格林内尔学院),玛丽·德斯加丁斯(马里兰大学巴尔的摩县校区),扎卡里·多兹(哈维·马德学院),保罗·朵里希(加州大学欧文分校),丽奈特·爵文(西北大学),思考特·德赖斯代尔(达特茅斯学院),凯希·菲斯勒(伍斯特理工学院),苏珊·福克斯(麦卡利斯特学院),爱德华·福克斯(弗吉尼亚理工大学),埃里克·弗莱登塔尔(德克萨斯州立大学埃尔帕索分校),斯蒂芬·弗罗因德(威廉姆斯学院),林·法齐尔(尼尔森·曼德拉都市大学),格雷格·加涅(维斯敏斯特学院),丹·加西亚(加州大学伯克利分校),朱迪·格史汀(印第安纳大学—普渡大学印第安纳波利斯分校),约兰达·吉尔(南加州大学),迈克尔·格莱契尔(威斯康星大学麦迪逊分校),弗朗西斯·格罗金斯基(圣心大学),安叙尔·古普塔(IBM),马克·古兹迪阿尔(佐治亚理工学院),布莱恩·海(阿拉斯加大学费尔班克斯分校),布伦特·海林佳(威廉姆斯学院),彼得·亨德森(巴特勒大学),布莱恩·亨德森 – 赛勒(悉尼科技大学),马修·赫兹(卡尼修斯学院),汤姆·海尔本恩(安柏瑞德航空大学),托尼·霍斯金(普度大学),约翰·爵林(乌得勒支大学),纪宜明(南卡罗来纳大学别福特分校),玛吉·约翰逊(谷歌),马

特·琼斯(斯旺西大学),弗兰斯·卡休科(麻省理工大学),丽莎·卡西玛希科(ACM 教育委员会),珍妮弗·凯(罗文大学),斯科特·科莱墨(斯坦福大学),吉姆·克鲁斯(马萨诸塞州大学阿默斯特分校),道格·李(纽约州立大学奥斯威戈分校),特里·林克莱特(华盛顿中区大学),大卫·鲁伯克(英伟达),比尔·玛纳里斯(查尔斯顿学院),塞缪尔·曼恩(奥塔哥理工大学),西·黛安·马丁(乔治华盛顿大学),多利安·麦克科里纳罕(IEEE - CS),安德鲁·麦克哥特里克(斯特拉斯克莱德大学),摩根·麦奎尔(威廉姆斯学院),基思·米勒(伊利诺大学斯普林菲尔德分校),汤姆·穆尔塔格(威廉姆斯学院),纳拉扬·穆尔蒂(佩斯大学),卡拉·南司(阿拉斯加大学费尔班克斯分校),托德·奈勒(葛底斯堡学院),里斯·纽曼(辛克莱社区学院),克里斯廷·尼克尔(计算机网络运营、网络安全和信息安全保障中心),詹姆斯·诺布尔(惠灵顿维多利亚大学),彼得·诺维格(谷歌),约瑟夫·奥罗克(史密斯学院),延斯·帕尔斯伯格(加州大学洛杉矶分校),罗伯特·帕诺夫(Shodor. org),苏希尔·普拉萨德(乔治亚州立大学),迈克尔·奎恩(西雅图大学),马特·拉透(多伦多大学),塞缪尔·A·瑞伯尔斯基(格林内尔学院),潘妮·瑞安庚斯(马里兰大学巴尔的摩县校区),卡洛斯·里德尔(卢塞恩应用科学大学),埃里克·罗伯茨(斯坦福大学),阿尼·罗森伯格(东北和科罗拉多州立大学),英格丽·拉塞尔(哈特福德大学),迪诺·史怀哲(美国空军学院),迈克尔·斯科特(罗彻斯特大学),罗伯特·塞奇威克(普林斯顿大学),海伦·夏普(开放大学),罗伯特·斯隆(伊利诺伊大学芝加哥分校),安·索贝尔(迈阿密大学),卡罗尔·斯布拉丁(西北密苏里州立大学),约翰·斯通(格林内尔学院),米歇尔·斯特劳特(科罗拉多州立大学),艾伦·苏斯曼(马里兰大学),布莱尔·泰勒(陶森大学),西门·汤普森(肯特大学),延·迪马诺夫斯基(ACM),辛迪·塔克(布鲁格拉斯社区和技术学院),伊恩·乌汀(肯特大学),格里特·范德伟(荷兰开放大学),约翰·凡尼耶科克(纳尔逊·曼德拉都市大学),克里斯托夫·冯普朗(乔治 - 西蒙 - 欧姆纽伦堡大学),罗索瓦·冯索墨斯(纳尔逊·曼德拉都市大学),亨利·沃克(格林内尔学院),约翰·瓦里聂可(加州大学伯克利分校),查尔斯·威姆斯(马萨诸塞州大学阿默斯特分校),杰罗德·温曼(格林内尔学院),大卫·韦瑟罗尔(华盛顿大学),梅兰妮·威廉姆森(布鲁格拉斯社区和技术学院),迈克尔·鲁英(英特尔)和朱莉·泽蓝斯基(斯坦福大学)。

此外,多个学术会议及其他研讨会也对 CS2013 报告草案的各个部分进行了审议,这些会议包括:第 42 届 ACM 计算机科学教育兴趣小组技术研讨会(SIGCSE - 11),第 24 届 IEEE - CS 软件工程教育与培训大会(CSEET - 11),

2011 年 IEEE 教育前沿大会(FIE－11)，2011 年计算科学联合研讨大会(FCRC－11)，第 2 届人工智能教育进展研讨会(EAAI－11)，ACM 数据通信兴趣小组 2011 年年度大会(SIGCOMM－11)，2011 年 IEEE 国际计算机、信息与系统科学与工程联合会议(CISSE－11)，2011 年系统、程序设计、语言和应用程序方面软件人性化大会(SPLASH－11)，第 15 届学术信息系统安全教育讨论会，2011 年国家学术中心信息辅助教育(CAE/ IAE)原则研讨会，第 7 届 IFIP 第 11.8 技术小组世界信息安全教育大会(WISE)，第 43 届 ACM 计算机科学教育兴趣小组技术研讨会(SIGCSE－12)，SIGCSE－12 上的一个特别讨论分会，2012 年计算机研究协会斯诺博得会议以及 2012 年 IEEE 教育前沿研讨大会(FIE－12)等。

CS2013 的写作过程也得到了一些组织和工作小组的支持，提供了宝贵的反馈意见。这些组织和工作小组包括：ACM 教育委员会和理事会，IEEE－CS 教育活动董事会，ACM 实践工作者理事会，ACM SIGPLAN 教育委员会，ACM 计算机与社会兴趣小组，SIGCHI 执行委员会，文科计算机科学联盟(LACS)，美国国家科学基金会与 IEEE－TCPP 并行与分布式计算联合课程计划委员会，英特尔与美国国家科学基金会赞助的信息安全研讨会以及美国国家科学基金会资助的网络安全课程体系指引项目。我们也感谢所有课程和课程体系案例的作者们。

参考文献

[1] ACM Curriculum Committee on Computer Science. 1968. Curriculum 68: Recommendations for Academic Programs in Computer Science. Comm. ACM 11, 3 (Mar. 1968), 151－197.

[2] ACM/IEEE－CS Joint Task Force on Computing Curricula. 2001. ACM/IEEE Computing Curricula 2001 Final Report. http://www.acm.org/sigcse/cc2001.

[3] ACM/IEEE－CS Joint Task Force for Computer Curricula 2005. Computing Curricula 2005: An Overview Report. http://www.acm.org/education/curric_vols/CC2005－March06Final.pdf

[4] ACM/IEEE－CS Joint Interim Review Task Force. 2008. Computer Science Curriculum 2008: An Interim Revision of CS 2001, Report from the Interim Review Task Force. http://www.acm.org/education/curricula/ComputerScience2008.pdf

第2章　指导原则

工作伊始,CS2013 指导委员会就对以下所列的原则达成了一致,用来指导本卷的编写过程,这些原则与之前设计课程体系时采用的原则在很大程度上重合,尤其是与 CC2001 和 CS2008 所采用的原则相似。与以前的 ACM/IEEE 课程体系一样,CS2013 将面对很多潜在的使用者,包括在各类教学机构工作的教师们,这些教学机构可能分布在全球六大洲的学院、大学和技术学校内,也包括这些教师任职的各计算机专业、院系及研究所,还包括一些鉴定机构和认证委员会,最后还有计算机类图书作者及研究人员等。其他潜在使用者包括大学预科学校和大学先修课程提供者,以及各类计算机科学的研究生专业。指导原则的设立充分考虑了这些潜在使用者的需求,兼顾了学生的学习目的与课程体系的开发及评估过程。注意,原则的重要性程度并不与以下原则的次序相关。

1. 计算机科学课程体系的设计应为学生提供灵活性选择,使其有能力进行跨领域工作。计算机是一个广泛的领域,它与其他许多学科相关,也从这些学科获取成长的动力,这些学科包括数学、电子工程、心理学、统计学、美术、语言学、物理学和生命科学等。应该培养计算机科学专业的学生们具有跨学科工作的灵活性。

2. 计算机科学课程体系的设计应该可以让学生毕业后有能力从事多种职业,这样也可以吸引具有各种才能的人加入学习计算机科学的行列。计算机科学几乎对所有的现代工作都有影响,所以,CS2013 采用了一个泛领域的观点,把各类“计算 - X”(例如,计算金融或计算化学)和“X - 信息学”(例如,生态信息学和生物信息学)也算在了计算机科学之内。如第 3 章(毕业生特征)所述,一个完美的计算机科学毕业生应该达到理论与应用的平衡。

3. CS2013 需要对毕业生在相关知识领域上的掌握程度提供指导,它需要设立具体目标来确定学生的掌握程度是否已达到预期水平,并提供课程和课程体系范例,用实例说明怎样涵盖知识体系里的各知识点领域。

4. CS2013 必须提供真实、实用的建议,提供指导的同时又给予灵活性,使在 CS2013 下,课程体系的设计可以进行各种创新并能跟踪领域的最新发展。该课程体系旨在提供清晰的、可实现的目标,同时也给予教学计划制定的灵活性,以便能对迅速变化的计算机科学做出快速反应。CS2013 旨在指导,而不是给定

一个用来评估各种教学计划的最低标准。

5. CS2013 必须适合各种教学机构。由于教学机构及教学计划的广泛性(需考虑 2 年期,3 年期和 4 年期的教学计划,文科类、技术类及研究类教学机构以及机构规模上的差异),我们不希望也不可能把计算类的课程体系进行严格规定,各教学机构需要评估自己的条件和环境来设定自己的教学计划。

6. 必须控制基本知识的体量。虽然与计算机科学相关的知识点范围有所扩大,可是能用于本科教育的课时数却没有相应地增长。因此,CS2013 必须谨慎选择其包括的知识领域,慎重推荐这些知识领域的基本要素。

7. 计算机科学课程体系的设计应培养其毕业生们有能力在这个迅速变化的领域里取得成功。计算机科学一直在迅速地改变着自己,并在可预见的将来继续快速变化着。课程体系必须包含培养学生终身学习能力的内容,必须包括对学生专业实践能力(如沟通能力、团队合作、职业道德等)的培养。计算机科学专业的学生必须学会把理论和实践相结合,既要认识到抽象的重要性,也能对良好的工程设计具有鉴赏能力。

8. CS2013 应明确给出计算机科学所有毕业生应该具备的基本知识和技能,同时在知识领域的选择上提供最大的灵活性。为此,我们把知识领域分为三个层面来介绍:核心一级、核心二级及可选知识领域。对于核心一级、核心二级及可选领域的完整讨论,请参阅第 4 章(知识体简介)。

9. 在课程及课程体系的组建上,CS2013 应提供最大的灵活性。知识领域的描述不是对特定课程的描述,事实上有许多新颖、有趣、有效的方法把各知识领域植入教学课程中去。

10. CS2013 的制定及审查必须建立在一个广泛的群众基础之上。CS2013 的编写工作必须有许多不同团体的参与,包括工业界、政府以及各类计算机科学教育的高等教育机构,必须考虑这些潜在用户的反馈意见。

第3章　毕业生特征

计算机科学专业的毕业生应掌握第4章所描述知识体中各领域的基本知识,尤其是列入核心知识点的那些知识领域,具备这些领域要求的基本能力。他们还应具备一些在知识体中没有明确列出的基本能力,因为该领域的专业人士通常会表现出一些独特的思维和解决问题的风格特征,这些特征通常是从具体的学习和工作经历中获得。下面描述的这些特征,我们认为计算机科学专业的毕业生们应该至少达到某种初级水平。这些特征可以使他们在计算机领域内获得成功,并有更好的职业发展。当然,其中的一些特征和技巧也适用于其他领域,但我们还是把它们包括在这里,因为在计算机科学专业计划中,应该明确地鼓励学生们努力发展这些技能和特征。以下这个列表基于CC2001和CS2008中出现过的类似列表,但新版列表也包含了一些实质性的修改,这些实质性的修改是在CS2013指导委员会进行调研的结果下做出的。

在一个广义层面上,计算机专业的毕业生应具有以下特征:

对计算机科学在技术层面上的理解

毕业生应该对计算机科学有所掌握,具体体现在掌握知识体中的核心部分。

熟悉通用的主题和原则

毕业生需要对一些反复出现的知识点有所理解,如抽象化、复杂性、渐进式改变等,也要对一些通用原则有所掌握,如共享资源、安全性、并发性等。毕业生应该认识到,尽管这些知识点和原则是在介绍某具体知识领域时引入的,但它们在计算机科学领域有着广泛的应用,并不只限于这些具体的知识领域。

对理论与实践之间的相互作用有所理解

计算机科学的一个基本特点就是要理解理论与实践的相互作用以及它们之间基本的关联性。计算机科学专业的毕业生需要了解理论与实践是如何相互作

用的。

系统分层观点

计算机科学的毕业生需要在多种细节程度、多个抽象层面上思考问题。这种思考方式能够超脱于各部件实现细节之上,达到对计算机系统整体结构的理解,并理解其组建的过程及分析的过程。他们还需要了解一个计算机系统会在怎样的环境中运行,会怎样与人类用户、物理世界进行交互。

解决问题的能力

毕业生要了解如何运用自己学到的知识来解决实际问题,而不仅仅是编写代码或搬动比特位。他们应该能够对一个系统的功能、实用性、性能等方面做出定量和定性的评估,并能设计和改善此系统。他们应该认识到,对一个给定的问题可能有多种解决方案,做出合理的选择并不是一个纯粹的技术问题,因为这些解决方案将对人们的生活产生真正的影响。毕业生也应该能够把自己的解决方案向别人做清晰的表述,解释一个解决方案为什么能解决给定的问题、是怎样解决了这个问题以及是在什么假设下解决这个问题的。

项目经验

为确保毕业生能够成功地应用他们所学到的知识,所有的计算机科学专业毕业生都应该参与过至少一个重要的项目。在大多数的情况下,这个项目可以是一个软件开发项目,但在特殊情况下,其他方面的项目也是适宜的。这些项目应以激励学生发挥综合性能力为主,要求学生对潜在的解决方案进行评估,其规模应大于一般课程下的项目规模。作为项目经验的一部分,学生应该抓住机会开发和训练人际交往中的沟通能力。

致力于终身学习

毕业生应该认识到,计算机领域的进步速度可以用迅猛来形容,毕业生必须拥有一个坚实的基础,使得他们能够而且乐于在领域进步时跟上步伐,保持相关的技能。程序设计语言和技术平台会随时间而改变,因此,毕业生需要认识到在

他们的整个职业生涯中,必须不断地学习和调整他们的技能。要获得这种能力,学生们应该接触多种程序设计语言、工具、范式和技术,并在受教育的全过程中,始终坚持对底层基本原理的学习。此外,与前辈们不同,当代毕业生们需要管理他们自己的职业发展和提升。在寻求职业发展时,毕业生需常常参与专业活动,如获取认证证书、参加管理培训及获得其他领域的知识等。

承担职业责任

毕业生应该认识到计算机学科内含有的社会、法律、伦理和文化问题,他们也必须认识到,不同的国家有不同的社会、法律和道德标准。他们应该熟知,伦理问题、技术问题、审美价值的相互交织,在开发计算系统时将会起到非常重要的作用。从业人员必须了解他们个人和集体所承担的责任及系统故障可能导致的严重后果,但他们也必须了解自己的局限性,以及他们所使用工具的局限性。

沟通和组织能力

毕业生应有能力向各类听众解释清楚技术问题及其解决方案,这可能涉及面对面的、书面的及电子的通信方式。他们应准备好作为团队的一员进行有效的工作,并能够管理好自己的学习和发展过程,包括对时间的管理、重点任务的安排及进度的管理等。

对计算广泛适用性的认识

计算平台的范围很广,小可以是嵌入式微型传感器,大可以是高性能集群和分布式云计算中心。计算机应用几乎影响到现代生活的所有方面,毕业生应了解计算机技术所有可能的应用方向。

对特定领域知识的鉴赏

毕业生应理解计算机技术会与许多不同的领域接触,很多特定领域内问题的解决同时需要计算技能和特定领域知识。因此,毕业生在其职业生涯里,要能够与来自不同领域的专家进行交流,并向其学习。

第4章　知识体概述

本章介绍知识体的结构、基本原理和知识体最本质的创新。知识体并不是一个特定的课程集合或课程设置——这是课程和课程设置范例的作用，相反地本章强调知识体的灵活性，从而使得课程适应学科人才培养的需求以及课程的不断演变。在计算机科学方面，可以认为知识体是教学内容覆盖以及课程设置实现的规格，各种课程设置应能够符合此规格的要求。

知识体设计特点与原则

以下是知识体设计和考虑的要点：

- 知识领域并不与课程体系中的具体课程一一对应：课程体系中应包含涉及多个领域知识点的课程。
- 知识点分为“核心”与“选修”，核心进一步划分为“一级”或“二级”。

➢ 课程体系应当包含所有的核心一级知识点，并确保所有学生都接触到了这些教学内容。

➢ 课程体系应当包含所有或者几乎所有的核心二级知识点，并确保所有学生都接触到了这些教学内容的绝大部分。

➢ 课程体系应当包含重要的选修知识点：对一个完整的课程体系来说只包含核心知识点是不够的。

- 因为知识体是层次结构的大纲，构建课程时必要强调与第三章描述的毕业生特征相关联的关键问题，即如何在一个课程体系内反映毕业生特征。
- 知识体中的学习成果和学时建议对课程目标的涵盖深度提供了一定指导。
- 增加的新知识领域反映了本学科的重要变化。

知识领域并不是必须的课程和重要的课程样例

将每一个知识领域与具体课程相关联是一个很自然的做法。我们通常明确地反对这种做法，尽管许多课程体系中的课程只包含单个知识领域的教学内容，或者正相反，某个知识领域的内容仅仅来源于一个单个课程。知识体的层次结

构是归类相关信息的有效的方式,而不是课程教学内容的组织方式。除了一般的灵活性之外,在很多地方,尤其是以下方面,应期待许多课程从多个知识领域中整合教学内容:

- 入门课程:在计算机科学领域入门课程有很多不同的成功做法。很多专注于软件开发基础的知识点并搭配一些程序设计语言和软件工程的知识点,而将剩下知识领域的大部分知识点留给高级课程。但在入门课程中涵盖其他知识领域的哪些知识点可以有所不同。有些课程采用面向对象程序设计,另有一些采用函数式程序设计,还有一些采用基于平台的开发。相反,这里并没有要求第一门或第二门课程涵盖所有的软件开发基础,但实际上这些早期课程通常都包含软件开发基础的大部分知识点。本卷会在第五章用一个单独的篇章来讨论入门课程。

- 系统课程:系统基础知识领域的知识点可以用于设计涵盖一般系统原理的课程,还可以用于那些特定的系统领域的课程,如计算机体系结构、操作系统、网络或分布式系统。例如,操作系统的课程可能被设计成涵盖一般系统原理,如底层编程、并发和同步、性能度量或计算机安全,还包含更具体的与操作系统相关的内容。因此,这样的课程可能会借鉴几个知识领域的教学内容。一些基本的系统知识点,如延迟或并行可能会出现在很多课程中的许多地方。虽然这些知识点的反复出现是非常重要的,但知识体并没有明确规定这些知识点的具体设置。附录 C 中的课程范例表明有多种途径可以将这种教学内容组织为课程。

- 并行计算:前面的章节讨论了知识体的变化,其中主要包括新知识领域的增加,新知识领域与并行和分布式计算有关。知识体结构的变化导致其他领域相关知识点可能改变或替换:如算法中的并行算法、软件开发领域中的程序构造、计算机体系结构中的多核设计等。既可以在一个课程中讲授基本的并行知识点,也可以贯穿在课程体系中,至少在近期会有致力于并行性的课程体系。

核心一级,核心二级,选修:这些术语是什么意思,什么是必须的。

正如本章开头所述,计算机科学的课程体系应当涵盖所有的核心一级知识点,绝大部分的核心二级知识点,以及许多颇有深度的选修知识点(对计算机科学的本科生来说仅学习核心知识点是不够的)。这里需要回答核心一级,核心二级和选修是什么,为什么要进行区分。

区分核心的动机:早期的课程指导,对每一个知识点只是分为“核心”和“选修”。放弃“每个核心课程的所有内容必须讲授给每个学生”这个严格的要求的原因是:

• 目前很多重要的计算机科学专业的课程体系都或多或少缺少了部分核心知识点,这样很容易造成课程体系满足不了计算机科学本科学位要求的结果。

• 随着学科的发展,核心内容的不断增长使得学生面向兴趣进行个性化选择的压力十分巨大。如果简单地处理这件事根本不可能让学生在短时间内获得本科学位。提供核心知识点覆盖的灵活性可以使课程体系和学生的个性选择成为可能。

相反,允许跳过某些核心知识点,可保证核心知识点的绝大部分是每个学生学习的基本部分。通过保持满足基本教学内容的一个较小核心,进而向课程设计者提供更多的指导和课程组织建议。核心一级包含计算机科学本科培养方案的基础知识点。

核心一级的含义:核心一级的知识点应该是每个计算机科学的课程体系必要的组成部分。虽然核心二级知识点与选修知识点也很重要,但培养方案必须包含核心一级知识点是广泛的共识。虽然大多数核心一级知识点通常被涵盖在入门课程中,但也有可能会涵盖在以后的课程中。

核心二级的含义:核心二级知识点对于计算机科学本科学位一般是必不可少的。大多数课程体系只需满足核心二级知识点的最低限,但也鼓励涵盖所有的核心二级知识点。这就是说,计算机科学的培养方案可以让学生聚焦在一些核心二级知识点不要求的特定区域。必须承认,资源的限制,如教员的缺乏、学位要求的体制限制,可能使得在提供高级的选修教学内容的同时,还要涵盖所有的核心知识点会面临巨大的困难。涵盖核心二级知识点的 90 ~ 100% 应该是一个计算机科学课程体系的目标,80% 是最低限度。

核心一级知识点不是必然处在核心二级知识点之前,尤其是入门课程会包括核心一级以及核心二级(甚至是选修)的内容,而一些核心内容将会被推迟到以后的课程。

选修的含义:一个只涵盖核心教学内容的培养方案提供的广度和深度将会有很多不足。大多数培养方案将不会包含知识体的所有选修内容,少数培养方案会涵盖全部。另一方面,知识体也绝不是穷尽的,高级课程往往超出它的内容和学习成果。尽管如此,在相应的教学内容方面,知识体为计算机科学的本科学位提供了一个有用的指导,所有计算机科学的学生应该通过选修知识点深化对多个领域的理解。

课程体系可能需要在知识体中指定选修教学内容。许多课程体系,尤其是那些有着特定培养方向的,必修课程规定了必须要求覆盖的选修知识点。

核心的规模

核心(一级加上二级)比以前的课程指南要长几个学时,但这可通过对核心的灵活处理来平衡。其结果是,并没有增加课程体系的必修课程数量。事实上,涵盖核心二级90%的课程体系与涵盖CS2008所有核心的课程体系有相同的核心学时,而且一个涵盖了核心二级80%的课程体系的学时总数比涵盖了CC2001核心(从2001至2008核心在增长)的要少。在本章结尾,将有更全面的定量比较。

注意平衡

计算机科学是研究理论、软件、硬件和应用程序四者优雅地相互作用的学科。当独立地看待核心特别是核心一级时,可能会专注于程序设计、离散结构和算法。这种专注的结果就是,这些知识点通常会作为高级课程的先修知识而较早出现。通过学习知识体的选修内容以更多不同的方式获得系统和应用程序的基本经验。所有的课程体系都应包括适当的选修内容,一个完整的课程体系可以并且应当达到适当的平衡。

设计课程体系时的进一步思考

一些跨领域的知识点,尤其是学科中的"big ideas"与知识体同样有用,并且是互补的,通过深入理解课程中的跨学科知识点来完善它是非常重要的。在设计课程体系时,确定课程体系广义要求的学习目标也是有价值的。因此,本卷第三章描述的毕业生特征的一些基本原则都是有必要的。

在过去的几年里,两个正在进行的趋势已经对许多课程体系产生了很深的影响。首先,计算机科学的不断发展,导致很多培养方案在学科内进行了更有特色的专门化设计,出现了描述此变化的许多术语(如Thread,Track和Vector)。其次,计算机学科对几乎所有其他领域的重要性导致建立越来越多的跨学科培养方案(如联合专业和双专业),并将跨学科的教学内容嵌入计算机科学的课程中。我们应当积极欢迎这两个趋势,相信灵活的知识体,包括更为灵活的核心知识点,能支持这两种趋势。另一方面,这样的专业化不是必需的:许多培养方案会继续将计算机作为一门独立又有关联性的学科,提供广泛而深入的内容。

知识体的组织

CS2013知识体由一系列的知识领域(KAs)组成,知识体通过相关知识点而

不是课程被组织起来。每个知识领域进一步组织成一组知识单元(KUs),在每个知识领域的描述部分(首部),这些知识单元被总结在一个表中。可预计到知识领域的知识点将在不同的院系以不同的方式被组织成课程。

课程学时

按照 CC2001/CC2008 的惯例,用授课时间(学时)作为单位来定义知识体中的覆盖,因为作为唯一的单位,在不同的文化背景下都是可理解并可转换的。学时是传统面对面讲课方式讲授课程内容所需要的时间,不包括与讲课相关的其他任何任务(例如,自主学习、实验课和评估)的时间。实际上,教师希望学生在课外花大量的额外时间巩固在课堂上学到的知识。按照以前课程指南坚持的原则,使用授课时间作为计量单位,并不要求或只认可使用传统讲课方法进行教学实践。

知识点学时规定代表了希望课堂内容涵盖的最少时间。院系可以选择在较长的一段时间涵盖相同的教学内容,从而满足学校的个性化需求。

课程

纵观知识体,当提到一个“课程”时是指学校认可的学习单元。通常情况下,全日制在校生可同时修好几门“课程”,每个学生将在每学年选修几门“课程”。在大部分院校,“课程”是最常见的术语,而在某些学校使用如“模块”或“paper”此类的其他名称。

对学习成果的指导

知识领域中的每个知识单元给出了要求的指定知识点和学生的学习成果描述。学习成果大小不等,并没有统一的课程学时要求;有相同学时的知识点可能有完全不同的学习成果。每个学习成果都与掌握级别有关。为了定义这个级别,借鉴了很多其他课程的方式,尤其是已经被计算机科学很好探究过的布卢姆分类法(Bloom's Taxonomy)。指南并没有直接应用布卢姆的分类,一个原因是它是根据教学法而制定,引入过多级别会增加描述的复杂性,另一个原因是掌握级别是象征性的,故不会给使用本指南的用户在理论上进行限制。

本指南使用了三个掌握级别,定义为:

- 熟悉:学生理解一个概念是什么或知道它的意思。这个掌握级别注重对概念有一个基本的认识,而不是期待能真正灵活地运用它。它回答“关于这个

你知道什么?”。

• 运用:学生能够以一种具体的方法使用或应用概念。例如,对概念的运用可能包括,在程序中适当地使用特定概念,使用特定的证明方法或执行特定的分析。它回答“关于怎样做你知道什么”。

• 评估:学生能够从多个角度考虑一个概念并证明能够用特定方法来解决问题。这一掌握级别不只是意味着应用概念,它包括从多种可选办法中选择较为合适办法的能力。它回答“你为什么要那样做?”。

以上方案可通过软件开发中的迭代进行示例。例如 for 循环,while 循环和迭代器。在“熟悉”级别,学生会被预期能对软件开发中的迭代做一个定义,并且知道为什么这是一个很有用的技术。为了显示“运用”级别的掌握程度,学生应该能够正确地运用迭代写出一个程序。在“评估”级别理解迭代要求学生理解迭代的多种方法并且能在不同应用时选择适当的方法。

无论是细节还是重点,都说明指南的学习成果描述可能不会与院校实际情况完全匹配。院校可能对相同的掌握级别和意图给定的知识点有不同的学习成果要求。不过,应该相信通过给予描述性的学习成果,既能明确指南的目的,又能促进对特定课程学习成果意义的解读。

新增知识领域综述

虽然计算机科学包括随时间迅速变化的技术,它仍旧是由基本不变的基本概念、观点和方法定义的。因此知识体的核心仍保持早期的设置不变。然而,随着计算机技术和教学方法的新发展,核心内容的某些方面随着时间也在演变,以前的结构和组织方式不再适用于对当前学科的描述了。故 CS2013 以多种方式对知识体的组织进行了修改,增加了一些新的知识领域(KAs)并重组了其他的一些领域。本节主要描述这部分的变化。

信息保障与安全(IAS)

由于认识到世界对信息技术和计算越来越强的依赖,所以增加了该领域。IAS 是一系列兼顾技术与政策的控制及流程的领域,旨在保护和防护信息和信息系统。IAS 将散落在其他领域的知识点集合在一起,与 IAS 密切相关的知识点在这个领域中深入地探讨,而其他的知识点可通过标记的交叉引用,找到相应包含它们的领域。因此,该领域的描述包含了一个交叉引用到其他知识领域的详细表格。

网络与通信(NC)

CC2001 引入了一个名为“以网络为中心的计算”的知识领域,包含了传统网络技术、Web 开发以及网络安全等知识点。上次的指南报告就指出了这些知识点的增长以及分歧,这次重命名并重新分解了这个知识领域使之专门侧重于网络和通信的知识点。新增的基于平台的开发(PBD)知识领域包含了 Web 应用和移动设备的开发。安全性被包含在新增的信息保障与安全(IAS)知识领域中。

基于平台的开发(PBD)

认识到在入门级和中高级选修课中可以增加使用特定平台编程环境,故增加了该领域。诸如 Web 或移动设备平台可以使学生在限制的环境中学习,这个环境通常与硬件,APIs 和特殊服务(通常是跨学科的背景)相关。这个环境与“通用目的”的程序设计课程非常不同,以保证这个知识领域有新的教学内容,此外该领域的知识点全部是选修。

并行与分布式计算(PD)

先前课程的并行性知识点作为选修分布在不同的知识领域。由于并行计算和分布式计算的重要性大大增加,确定这一领域的基本概念并使之成为核心教学内容尤为关键。为了突出和协调相关教学内容,CS2013 为这一领域设立了专门的知识领域,内容包括程序设计模型、程序设计语用学、算法、性能、计算机体系结构和分布式系统。

软件开发基础(SDF)

该领域将入门的程序设计拓展到更多地专注于软件开发过程,确定了第一年计算机科学课程应该掌握的概念和技能。由于其广泛的用途,该领域包括了以往出现在其他面向软件的知识领域的基本概念和技能(例如,程序设计领域的程序构造,算法和复杂性领域的简单算法分析,软件工程领域的简单开发方法)。同样,这些知识领域相对于 SDF 领域中的基本概念和技能包含更多的高级教学内容。相较于以前的课程指南,本卷将面向对象程序设计、函数式程序设计和事件驱动程序设计等关键程序设计手段纳入了程序设计语言知识领域,并期望任何课程体系的入门课程都要包含其中的一些知识点。

系统基础(SF)

在以前的课程指南中,一个典型计算机系统的交互层,包括硬件构造块、体系架构,操作系统服务、应用程序执行环境(特别是从现代应用视角出发来看并行执行),这些内容分散在许多知识领域。本新领域为其他的知识领域(包括体系结构与组织、网络与通信、操作系统和并行与分布式计算)提出了一个统一的系统观点和共同的概念基础。其组织原则是"程序设计性能":程序员需要明白怎样由底层系统实现高性能模式,特别是在挖掘并行性方面。

知识领域的核心学时

下表是 CS2013 知识体中各个知识领域的核心(一级和二级)学时数的概况。为了进行比较,提供之前 CS2008 和 CC2001 报告的核心学时数。

KA	知识领域	CS2013 核心一级	CS2013 核心二级	CS2008 核心	CC2001 核心
AL	算法与复杂度	19	9	31	31
AR	体系结构与组织	0	16	36	36
CN	计算科学	1	0	0	0
DS	离散结构	37	4	43	43
GV	图形学与可视化	2	1	3	3
HCI	人机交互	4	4	8	8
IAS	信息保障与安全	3	6	– –	– –
IM	信息管理	1	9	11	10
IS	智能系统	0	10	10	10
NC	网络与通信	3	7	15	15
OS	操作系统	4	11	18	18
PBD	基于平台的开发	0	0	– –	– –
PD	并行与分布式计算	5	10	– –	– –
PL	程序设计语言	8	20	21	21
SDF	软件开发基础	43	0	47	38

续表

KA	知识领域	CS2013 核心一级	CS2013 核心二级	CS2008 核心	CC2001 核心
SE	软件工程	6	22	31	31
SF	系统基础	18	9	– –	– –
SP	社会问题与专业实践	11	5	16	16
	总计	165	143	290	280

核心一级 + 核心二级	合计	308
核心一级 +90% 的核心二级	合计	293.7
核心一级 +80% 的核心二级	合计	279.4

由上表可见,CS2013 的核心一级学时数加上核心二级的学时数略高于前期指南报告的核心学时数。然而,应当注意到,CS2013 分级的结构为院系从核心二级选择知识点(至少包含 80%)明显地提供了灵活性。其结果是,与以前的课程指南的学时相比,CS2013 指南的实施是完全可行的。

第5章　入门课程

与许多技术学科不同,几乎所有的计算机科学入门课程都没有一个很好的知识点描述列表。入门课程是不断变化的,先要了解这些课程从 CC2001 到 CS2013 的演变。CC2001 将入门课程系列分为六个通用模型:命令优先,对象优先,函数优先,广度优先,算法优先以及硬件优先。虽然入门课程的这些特征到今天仍然存在,参照最初 CC2001 的模型,这一领域的进步已经使入门课程更加多样化。此外,入门课程采用的方法也处在不断变化的状态。

对入门课程经过数十年的争论后,这些课程的内容仍然是激烈讨论的话题,其重要原因是并不是每个与计算机的研究者相关的东西都可以从头教起(程序设计、软件过程、算法、抽象、性能、安全性及专业性等)。这是一个重要挑战。换句话说,不是每件事都可以排在第一位,因此一些知识点必须推到课程的后面,在某些情况下尤为显著。许多知识点将不会出现在第一门课程,甚至是第二门,这意味着那些不继续后续学习的学生(例如,非计算机专业)将不会接触到这些知识点。最后,当试图决定早期课程应包含什么时,必须要考虑到课程体系中入门课程内容的选择会导致的一系列权衡。

设计维度

将制作入门课程相关的设计维度作为本章的结构,在每个维度上,集中总结相关的权衡考虑。给定的计算机科学入门课程或课程序列代表了在多维设计空间中的一组决定,并希望能达到独特的结果。要注意的是,在这里讨论的重点是入门课程作为计算机科学本科课程的一部分所代表的意义。还需要注意的是,这里不讨论日益增长的"CS0"课程:这些前序课程往往侧重计算机素养或计算思维。其内容主要包括介绍计算机科学的概念或相关教学内容,但并不是知识体的一部分,不在讨论的范围之内。

入门课程的途径

应当认识到入门课程不是抽象的构造,而是面向特定目标受众和相关背景

的。院系最了解他们自身的学科背景,对自己的学生和他们的需求很敏锐。入门课程在不同院系有所不同,尤其在入门课程系列的性质和学时方面(即允许学生进行专业选择前必须上的课程数)。课程系列也可能有不同的入口,以适应来自不同知识背景或之前的计算机经验有显著差异的学生。入门课程系列拥有多种进入和通过的途径可以帮助学生匹配与其能力适合的课程级别。它也可以帮助在两年制和四年制院校之间创建更灵活的衔接,使来自其他学校/专业的学生更加顺利地转到计算机专业。计算的普适性和编程特殊技能,对其他领域的学生来说越来越必不可少。面向非计算机专业的课程与需要多年学习的面向计算机专业的课程可能有区别,也可能区别不大。此外,入门课程的多途径可以给在大学阶段较晚开始选择计算机的学生们提供更多的选择。

针对不同受众建设课程是必要的,而不仅仅是面向已经确定要学习计算机科学专业的学生,这样可以帮助大部分其他专业学生进入计算机领域。需要注意到计算机领域跨学科的重要性,以及程序设计入门课程的吸引力已显著扩大并超越了传统的工程领域。对于不同背景以及不同期望的目标受众,实行以跨学科知识点为重点的入门课程(例如,计算生物学、机器人技术、数字媒体处理等)已成为流行。这样一来,教学内容就和那些有着多学科定位的学生的期望与愿望联系起来了。

权衡:提供了多种进入和通过入门课程系列的途径可以使计算机科学更容易被不同的受众接受,但院系需要更大的投资(在工作和资源方面)来构建这样的路径或给学生提供不同知识点的选项。此外,必须注意引导学生选择合适的入门课程的途径(在这方面对有大量计算机经验知识的学生和完全没有经验的学生是一样的)。

较长的入门课程序列(即更长或更结构化的先修课程序列),可以让教师为每门课程假定更多先修知识,但是,在学生能够参加他们感兴趣领域的高级课程之前,这种冗长的序列降低了灵活性并增加了时间。

侧重于程序设计

大部分入门课程都是以程序设计为中心的,通过学习一门给定的程序设计语言并编写软件制品,可以使学生更好地学习计算机科学的概念(如抽象,分解等)。侧重于程序设计可以为计算机科学专业的学生在早期提供对这一关键技术的训练,并帮助计算机背景不同的学生将能力提升到更平等的水平。即使侧重程序设计,学生编写完整的程序还需要进一步细分——确保能理解各部分是

如何组合在一起的,并给出一个构建项目的完整体验——同样的,学生根据现有程序和框架完成或修改,使学生获得处理现实世界问题的经验,帮助学生编写更大、更复杂的程序。许多不侧重程序设计的入门课程的目的是为了提供对计算机概念更广泛的介绍,而不仅限于学习一门程序设计语言的语法。课程安排有意识的不集中在程序设计方面,这样的观点大致类似于 CC2001 的“广度优先”模式。无论程序设计是否是第一门课程的主要内容,重要的是不要让学生认为计算机科学只是学习特定程序设计语言的细节。在学习程序设计的时候,还需强调更普遍的计算机概念。

权衡:一个以程序设计为重点的入门课程,可以帮助学生在初期培养必要的技能,并提供一种可以描述其他计算机科学概念的通用语言。侧重于程序设计对于来自其他领域学生的学习是有用的,他们希望可以在跨学科工作中将程序设计作为一种工具。然而,在入门课程中过于狭隘的注重程序设计,直接讲授一门程序设计语言,也会给学生一种过于狭隘(或误导性)的看法。对于一些学生来说这种狭隘的观点可能会限制计算机科学对其的吸引力。

程序设计范例和语言的选择

许多入门课程的一个决定性因素是程序设计范式的选择,从而引起程序设计语言的选择。事实上,在 CC2001 列出的六种入门课程模式有一半是通过程序设计范式进行描述的(命令优先,对象优先,函数优先)。这种以范式为基础的入门课程仍然存在,它们的优点也一直在争论。需要注意的是,过去的十年内可成功用于入门课程的程序设计语言大大增加了,而不是随着时间推移渐渐青睐于某一个特殊的范式或语言。但是,似乎有朝着“更安全”或更可管理的语言增长的趋势(例如,从 C 到 Java),以及更多动态语言的使用,如 Python 或 JavaScript。可视化程序设计语言,如 Alice 和 Scratch,也已成为流行的选择,它们向入门课程提供了“简化语法”的程序设计方式,它们通常(但又不仅仅)被非专业人员或在入门课程的初始阶段使用。一些入门课程系列提供了好几种可选的程序设计范式,脚本与过程式程序设计或者函数式与面向对象程序设计都是可选的程序设计范型。这些不同的选择让学生很好地体验到程序设计的不同观点,避免僵化语言的特征,并矫正他们的某些观念:比如存在一种“正确”或“最佳”的程序设计语言。

权衡:在这里有很多的权衡,包括:

(1) 使用专门为入门课程设计的语言或环境能促进学生学习,但会造成其

他课程的使用限制。相反,使用专业用途的语言或环境会使学生过早地接触太多的复杂性设计。

(2) 使用“安全的”或更能管理的语言和环境来帮助非计算机专业学生学习。但是,这样的语言教学可能会提供一个抽象层,从而掩盖了对实际机器运行的理解,使得难以进行评估性能的权衡。是否使用“低级”语言来执行特定的程序模型,使其更接近于由机器实际执行的程序,往往是各个院系需要做的决定。

软件开发实践

虽然程序设计是构造软件的方法,入门课程可以在软件开发的不同阶段选择采纳许多其他的实践方法。例如,使用软件开发的最佳实践可以在不同的入门课程强调不同的内容,这些实践包括单元测试、版本控制系统、集成开发环境(IDEs)和程序设计模式等。引入此类软件开发的实践可以帮助学生较早地体验真实的软件开发项目面临的挑战。另一方面,虽然所有的计算机研究人员都应具有坚实的软件开发技能,但这些技能并不局限在基本的编程入门课程中,特别是如果目标受众不仅仅是计算机科学专业的学生。入门课程从一开始就应该小心平衡软件开发实践的最佳方式,使更多的人理解入门课程。

权衡:在入门课程中纳入软件开发实践可以很早就帮助学生培养实际的软件开发能力。入门课程包含这种实践,在某种程度上受课程目标受众的影响,也会影响目标受众。软件开发实践也会影响程序设计语言及开发环境的选择。

并行处理

传统意义上,入门课程已经假定采用单处理器、单进程和单线程,程序完全按照程序员的指令进行的顺序执行。最近软硬件的发展促使教育工作者重新思考这些假定,即使在入门级课程——多处理器也是普遍存在的:用户界面导致异步事件驱动处理;“大数据”需要并行处理和分布式存储。因此,一些入门课程从一开始就强调并行处理(传统的单线程执行模型被看作是通用并行模式的一个特例)。然而从长远来看,应该相信这是一个有趣的模型。可以预测,未来,“单线程模式”在入门课程中仍然占据主导地位(可能包含基于图形用户界面(GUI – based)或机器人事件驱动编程)。随着越来越多成功的教学方法的开发,新手程序员也可以访问并行处理,并行程序的范例变得更为普遍,从而希望在入门课程看到更多并行程序的元素。

权衡:了解并行处理对计算机专业的学生越来越重要,早期学习这种模式可以让学生在这个领域有更多的实践。另一方面,并行程序在大多数当前的编程环境中仍然存在困难。

平台

虽然许多程序设计入门课程使用传统的计算机平台(例如,台式/笔记本电脑),其结果是,有些平台是与硬件无关的。在过去的几年已经看到引入到课程中的可编程设备越来越多样化。例如,一些入门课程可能选择基于 Web 开发或移动设备编程(如智能手机、平板电脑);也有使用机器人或游戏机等专用平台。这样可能会帮助新手对专业产生更多的热情,强调与外部世界的交互是必要的、自然的。近年来课程的发展导致发明了专门以学习程序设计为目的的小型的、功能受限的计算设备(如 raspberry - pi)。在这些情况下,使用一个特定平台并选择相应的程序设计范式,组件库,APIs 和安全规范。在平台软/硬件的限制条件下进行工作是一个非常有用的软件工程技能,如何选择课程的知识点也可能同样受限于平台的选择。

权衡:使用特定的平台可以给课堂带来令人信服的真实世界环境,平台的教学设计能够让学生产生有益的焦点。但是,它需要相当谨慎,以确保特定平台的细节不会掩盖教学目标。此外,平台的特殊性可能会影响后续课程内容的可转移性。

映射到知识体

实事求是的讲,入门课程系列不应该被理解为只简单地包含软件开发基础(SDF)知识领域的知识点。相反,应鼓励实施 CS2013 指南的教师在考虑上述设计维度的基础上,为入门课程系列选择多个知识领域的教学内容。举例来说,即使是相对简单的通常入门课程系列的教学内容也要来自软件开发基础(SDF)、程序设计语言(PL)和软件工程(SE)领域,其中与 PL 相关的知识点主要与在课程中使用的程序设计语言及相应选择相关。更广泛地说,使用非传统平台的课程将借鉴基于平台的开发(PBD)领域的知识点,可能还包含并行与分布式计算(PD)领域中强调多处理的教学内容。应鼓励读者把 CS2013 知识体作为其建设创新型新入门课程系列的指引,以适合本校学生的需求。

第6章　体制的挑战

虽然知识体规定了什么内容应该被包含在计算机科学本科培养方案中,但并不视其为计算机科学本科课程应简单累加的传授内容。计算机科学是一个快速变化的领域,讲授什么与促进现行的研究、帮助学生构建吸收新知识的框架和促进学生面向职业发展这三个方面是互补的。批判性的思维、解决问题的能力、为终身学习奠定基础是学生们在整个本科学习生涯中需要培养的技能。教育不仅仅是信息的传播,更要激发学生对学科的热情,鼓励学生们去尝试,让他们体验到成功的喜悦。这些方面需要反映在计算机科学课程体系和教学方法中。

本地化 CS2013

任何学校要成功地部署和更新计算机科学课程体系,需要有满足是本校需求的敏感性。CS2013 既不应该被解读为知识领域的简单选取,也不是从课程到知识领域的一一映射。重要的是应当鼓励学校思考一种方式,使知识体可以最好地融合成为一个独特的课程体系,该体系能反映学校的教学目标、师资力量、学生需求和用人单位的要求。事实上本卷指南创建两级核心正是为了精确地提供这种灵活性,为了最好地适合这些要求,保持核心一级教学内容必须的最低限度的同时,使学校在选择核心二级教学内容时有更大的选择余地。

积极推进计算机科学

除了课程,也要强调咨询、指导和培养教师和学生之间关系的重要性。许多学生,尤其是那些来自弱势背景的学生,可能无法体会到一个计算机科学学位可以带来的全部职业选择。宣传和推进计算机专业职业选择的广泛性,特别是适应当地用人单位的需求,有两个益处:首先,它为学生提供他们可能没有考虑到的有关职业选择的信息;其次,可帮助计算机科学的相关专业吸引更多的学生(潜在的来自更广泛多样的背景)。无论目前的入学趋势如何,随着时间的推移,提供一个健康的计算机科学培养方案,需要致力于吸引更多学生到计算机领域(计算机领域注册人数在近十年来有很大的起伏)。

更值得注意的是:许多学生仍然觉得学习计算机科学等同于今后工作时成为“程序员”,从而产生消极、错误的刻板想法,认为计算机科学工作是孤立的、机械的。同时有许多学生相信,如果他们在之前没有大量的程序设计经验,他们在修计算机科学的学位时会没有竞争力。计算机系应该强烈地质疑这两个看法。计算机系通过举办如邀请校友与在校生交流等课外活动展示计算机科学学位潜在的职业道路:学习计算机科学除了“作为一名程序员”还有很多可能性;软件开发是一个极具创造力和协作的过程。在这些努力中,设计多入口进入计算机专业学习的课程体系是非常重要和迫切的,这样可以使有或者没有程序设计经验的学生都可以顺利转入。

扩大参与

毫无疑问现今对拥有计算技能学生的需求是巨大的。事实上,已有预计在未来十年中信息技术工作岗位会有巨大的空缺。因此,迫切需要扩大参与计算机科学的学习人数,不分种族、性别和经济地位全方位地吸引人才到计算机领域。学校应努力使更多学生进入计算机科学培养流水线,并提供相应支撑,帮助学生顺利完成学业。

校园内的计算机科学

一种论点认为计算机科学正在成为21世纪大学教育的核心学科之一,也就是,任何受过教育的个人必须在计算机科学方面达到理解和掌握的水平。这就超越了计算机科学作为一个广泛的跨学科研究的工具和方法的现实。很可能在不久的将来,许多大学的每个本科生都需要进行计算机科学的学习,计算思维成为所有毕业生所期望的基本技能之一。计算机科学教学任务显著的扩展会对院系资源产生巨大的影响,尤其是在教师和实验设备方面。

虽然CS2013为计算机科学本科培养方案提供了指导,仍需认为在广泛的跨学科领域提供计算机教育是非常重要的。为此,计算机院系不妨考虑提供容易参与学习的课程,特别是入门级别的,这样会对许多学科的学生具有吸引力。这也可以达到双重目的,吸引更多起初没有倾向选择计算机专业的学生。

更广泛的说,计算机学科正成为其他学科不可或缺的工具,这有利于计算机科学系对外开放,建立起与其他院系和其课程领域的桥梁。鼓励学生从事多学科的工作,促进跨计算机科学和其他领域学习的培养方案(例如,在“计算X”培

养方案中,X 代表其他学科,如生物或经济学科)。

计算机科学辅修

在确定计算机科学作为大学核心学科之一的同时,院系也可以考虑面向计算机科学提供更多的辅修。辅修应该给学生提供灵活的选择,使学生获得连贯的计算机科学知识,而不是仅学习一两个课程,但这也不可能包含完整的培养方案。事实上,辅修的另一个意义是:可以使主修其他科目的学生获得坚实的计算机基础,今后在和他们领域有交叉的工作中使用。

众所周知,本科学生主修专业的选择是在真实了解了各专业培养方案差异的情况下进行的。结果就是很多学生不选择计算机专业,仅仅是因为之前缺乏对计算机科学的了解,而不知道计算机科学真正需要什么,也不知道他们自己是否喜欢这个学科。辅修计算机科学的学生如果在其后的学术生涯中对计算机产生兴趣或发现计算机带来的新东西,这些辅修的内容可以让他们获得某个计算机相关的文凭。为了能够给予这类学生主修计算机科学的能力,这些“尝试”性的课程应当设置为能让学生尽快适应计算机专业本科的要求。

计算机科学对数学知识的要求

数学和计算机科学的许多领域之间有着深入而密切的联系。几乎在所有的计算机科学课程体系中都包括数学课程,广泛地讲,每个院校都有自己一整套不同的要求。导致这种差异的因素很多,如计算机专业是否在工程学院,这直接影响是否开设微积分及微分方程的课程,即使这些课程包含的内容远远超出了大多数计算机专业所需要的。同样地,一些学校可能限制一种专业所修的课程,例如在许多文理学院,可能会特别地对计算机科学专业设置数学方面的要求。因此,CS2013 仅指定对大多数计算机本科生直接相关的数学要求,这些要求包括集合论、逻辑和离散概率等。知识体对这些数学知识的要求主要是在离散结构(DS)领域。

对所有的计算机科学系学生来说,常用的数学能力是一项重要的要求。不过 CS2013 将基础数学和与计算特定的领域直接相关的、仍旧很重要的数学进行了区分。基础数学可能会影响计算机科学的很多部分,此方面的数学被包括在 CS2013 知识体中。例如,对线性代数的理解在许多计算机领域扮演着重要作用,如在对图形和图形算法的分析中。然而,线性代数不一定在所有的计算机领

域都需要(事实上,许多高质量的计算机培养计划没有对线性代数提出明确要求)。同样,虽然注意到在计算机学科使用概率与统计呈增长趋势(通过知识体中知识点的核心学时数增长反应出来),也相信在未来这种趋势还是会持续,但是仍没有必要对所有的计算机专业在培养方案中安排一门完整的概率论课程。

一般情况下,计算机科学的培养方案肯定会将学生的"数学成熟度"提高一个等级。例如,对算术操作的理解,包括简单求和及其系列,对算法效率分析来讲是必须的。但是面向大学水平的课程,对基本算术知识的详细要求超出了CS2013的范围。也就是说,培养方案使用微积分的要求并不是作为领域知识的一种手段,而是作为一种方法,早早的在大学水平的教育中帮助学生开发数学成熟度和清晰的数学思维。因此,虽然没有规定这种要求,但注意到计算机系的本科生需要有足够多的数学知识,才有基础去构建计算机数学(CS - specific mathematics)(例如,在离散结构知识领域中指定的那些),这其中,重要的是,并没有明确要求微积分、微分方程或线性代数是重要的大学水平课程。

有的学生转到特定的计算机领域学习先进课程,他们可能需要着重学习有关这些领域的数学课程。计算机专业能够帮助促进学生们选择除离散结构之外的数学内容,使计算机专业学生们获得他们所选的特定计算机领域的背景知识。这种课程要求最好是留给学生的个人培养方案或学生所选的特定计算机领域去斟酌决定。

最后,计算机科学培养方案中的数学要求会形成一个先修课程序列,必须注意该序列上的课程数量,事实上,数学类先修课程的结构不是计算机科学院系自身规定的,但是在培养方案设计的时候必须考虑允许之前没有显著数学背景知识的学生主修计算机科学。计算机科学专业要求一系列冗长的数学类先修课程,可能会阻碍学生进入计算机学科,也可能使在大学后期转入计算机专业的学生感到困难,也可能使想在早期学习中学习计算机特定课程的学生们感到更困难,从而可能会打消他们的积极性。

计算机资源

对于学生和教师们来说,计算机科学专业的课程需要充足的计算机资源。计算机科学课程的需求往往超出传统的基础设施(一般的校园计算机实验室),并可能包含专门的硬件和软件,或大规模的计算机基础设施。项目和毕业设计课程拥有充足资源是非常重要的。此外,院系需要考虑计算机设备的多样性(如智能手机、平板电脑),这些设备都可以作为学习课程的平台。

保持教师的活力和健康

一个强大的计算机科学专业建立在拥有足够多经验丰富的教师基础上,才能保持院系的健康和活力。院系招聘教师应不仅满足保持培养方案的可行性,也要允许现有的教师有时间进行专业开发和探索新的想法。为了应对计算机领域的快速变化,教师们必须寻找机会培养新技能,了解新领域,与新技术的发展同步。虽然传授新技术与遵守基本原则是矛盾的,但无论偏重于两者中任何一个方面都不会对学生有利。教师需要时间获取新的思想和技术,并将其纳入课程和课程体系中。通过教师来接受并改进新的教学内容和方法这种方式,院系可以建立专业学习和终身学习的机制模型。

除了专业的发展,计算机科学专业对招生人数的变动保持良好的应对能力也是尤为重要的。事实上,计算机作为一门学科在过去几十年经历了几番繁荣和萧条的波折,导致在世界各地的招生都有重大的变化,这也几乎跨越了所有类型的院校机构。面对招生低迷的情况,院系应该创建相应应对措施来保持适应力,例如,课程内容更容易被广泛接受,与其他院系建立跨学科双修课程,或提供服务课程。

面对持续的招生人数增加(如近几年来有目共睹的),招聘充足的师资力量成为迫切需要。如果能力不足,可以通过扩大选课人数(每个课程有更多教学平行班并有更多的学生反馈评价)或增加教学任务(每一位教师必须讲授更多的课程)给教师们增加压力,但这样可以会导致教学质量的下降和潜在的教师流失。质量的问题会导致学生放弃计算机专业。正如上述讨论的那样,由于需要更多、更有能力的计算机专业毕业生,这种结果是非常不利的。面对日益增长的招生人数,保持强大师资能力必要性的激烈争论已经广泛展开,该争论在计算机专业近期的繁荣[5]和 30 年前[2]都存在。

教职员工

终身教职员工首要的评价标准(从广义上来说)是基于对教学的贡献,可以帮助建立更适合的课程,投入参与课程实验的设计和修订,扩大课程的宣传使更多的学生选择这门学科。随着来自所有院系挑战的加剧,这种的职业标准代表策略和务实的一种平衡。这种教职定义的价值最早由 CC2001 察觉到,并在十几年内都没有减弱,而且最近受到了更多的认可[7]。

大学生助教

虽然研究型大学传统上招募研究生作为本科课程的助教,但在过去 20 多年,越来越多的学校发现为计算机入门课程招收优秀的本科生作为助教的价值。本科生担任助教的益处包括:当他们成为帮助别人的角色时,他们会对知识有更好的理解,可以更合理的安排时间,提高组织责任能力和表达能力[4,6]。对学习入门课程的学生也有好处:学生们有更多可用的课程教学人员,更方便接触助教,可以从相近的同龄人那里得到帮助,这是学生更愿意面对的更熟悉问题和相应解决方案的助教。

网络教育

在初期的网络教育和近期的大规模在线开放课程(MOOCs)的推动下,高等教育掀起了海啸。讨论网络教育的潜能和缺陷远远超出了本文献的研究范围。为此只是简单地指出网络教育的某些方面可能影响到相关院系对本指南的部署。

首先,在线教育的教学内容不必安排成完整学期的课程结构。因此大多数情况是在网络上开设迷你课程或模块(其显著特点是课时少于一个学期),但这些课程仍旧包含 CS2013 知识体相关内容。这样一来,一些院系,特别是教师资源有限的学校,可以选择去寻找和利用其他院系提供的在线教学内容。混合式学习是另一种模式,学习同样一门课,可以同时获得面对面学习和在线学习两种方法的益处。

MOOCs 带来的振奋还包括支持大规模学生的学习,但大规模课程作业的评估和反馈会面临技术挑战,但这对于计算机科学是个重要且崭新的研究机会。MOOC 平台为评估学生的学习有效性所提供的量化功能,可能会改变计算机科学本身的教学。

虽然我们意识到可扩展规模课程的价值,但要注意培养的重点不是关于课程内容或传达信息,例如,教学方法和对学习的支持。其次,虽然 MOOCs 是强大的传递知识的媒体,但要确保计算机科学毕业生的特征仍在不断发展中,这点是非常重要的。

参考文献

[1] Auletta, K. April 30, 2012. "Get Rich U.", The New Yorker.

[2] Curtis, K. Computer manpower: Is there a crisis? National Science Foundation, 1982.

[3] Microsoft Corporation. A National Talent Strategy: Ideas for Securing U. S. Competitiveness and Economic Growth. 2012

[4] Reges, S., McGrory, J., and Smith, J. "The effective use of undergraduates to staff large introductory CS courses," Proceedings of the Nineteenth SIGCSE Technical Symposium on Computer Science Education, Atlanta, Georgia, February 1988.

[5] Roberts, E., "Meeting the challenges of rising enrollments," ACM Inroads, September 2011.

[6] Roberts, E., Lilly, J., and Rollins, B. "Using undergraduates as teaching assistants in introductory programming courses: an update on the Stanford experience," Proceedings of the Twenty - sixth SIGCSE Technical Symposium on Computer Science Education, Nashville, Tennessee, March 1995.

[7] Wolfman, S., Astrachan, O., Clancy, M., Eiselt, K., Forbes, J., Franklin, D., Kay, D., Scott, M., and Wayne, K. "Teaching - Oriented Faculty at Research Universities." Communications of the ACM. November 2011, v. 54 (11), pp. 35 - 37.

附录 A　知识主体

算法与复杂度（Algorithms and Complexity，AL）

算法是计算机科学与软件工程的基础。实际应用中软件系统的性能依赖于：(1) 使用的算法；(2) 各层实现的适合程度以及效率。因此算法设计的好坏对于所有软件系统的性能都至关重要。此外，对于算法的研究也提供了对于问题内在本质的洞察以及独立于程序设计语言、编程范例、计算机硬件以及任何其他执行层面的可能的技术解决方案。

计算的一个重要组成部分是在认识到可能没有合适算法存在的前提下，选择并应用适用于特定用途算法的能力。这依赖于对定义完备的一组重要问题算法范围的理解，并且了解这些算法的长处和弱点以及他们在特定环境下的适用性。效率是贯穿这个领域的普遍知识点。

这一知识领域定义了设计、实现和分析算法用以解决问题所需的核心概念和能力。算法对于计算机科学的所有前沿领域都至关重要，包括人工智能、数据库、分布式计算、图形学、网络、操作系统、编程语言、信息安全等。在以上领域有特定用途的算法会在相关的知识领域中列出。例如密码学会出现在新的信息保障与安全(IAS)知识领域，同时并行和分布式算法会出现在并行与分布式计算(PD)知识领域。

本文涉及的所有知识领域，其知识点的顺序和分组安排，与讲授的顺序不一定相关。不同的教学计划会在不同的课程中讲授这些知识点，按照他们认为最适合学生的顺序来安排。

AL. 算法与复杂度(19 个核心一级学时，9 个核心二级学时)

	核心一级学时	核心二级学时	包含选修
AL/基础分析	2	2	否
AL/算法策略	5	1	否
AL/基础数据结构及算法	9	3	否

续表

	核心一级学时	核心二级学时	包含选修
AL/基础自动机的可计算性及复杂度	3	3	否
AL/高级计算复杂度			是
AL/高级自动机理论及可计算性			是
AL/高级数据结构、算法及分析			是

AL/基础分析

［2 个核心一级学时，2 个核心二级学时］

知识点：

［核心一级］

- 在最优、期望以及最坏情况下，一个算法的行为特征。
- 复杂度上界以及期望界限的渐进分析。
- 大 O 符号：正式的定义。
- 复杂度种类，包括常数级、对数级、线性、二次以及指数级。
- 实际的性能度量。
- 算法的时间与空间的权衡关系。

［核心二级］

- 大 O 符号：使用。
- 小 o，大 Ω，大 Θ 符号。
- 递推关系。
- 迭代算法与递归算法的分析。
- 主定理的一些版本。

学习成果：

［核心一级］

1. 解释什么是“最优”，“期望”和“最坏”情况下一个算法的行为。［熟悉］

2. 在特定算法的情况下，指出会导致不同行为的数据特征及/或其他条件或假设。［评估］

3. 非正式地确定简单算法的时间和空间复杂度。［运用］

4. 阐述大 O 符号的正式定义。［熟悉］

5. 列出及对比标准的复杂度种类。［熟悉］

6. 通过实践验证通过数学分析推导出的关于运算时间的假设。输入运行

不同大小规模的算法并比较性能。[评估]

7. 举例说明对于算法的时间与空间的权衡关系。[熟悉]

[核心二级]

8. 使用大 O 符号给出算法的时间和空间复杂度的渐进上界。[运用]

9. 使用大 O 符号给出算法的时间和空间复杂度的期望界限。[运用]

10. 解释如何使用大 Ω,大 Θ,小 o 等符号来描述一个算法的工作量。[熟悉]

11. 使用递推关系来确定递归定义的算法的时间复杂度。[运用]

12. 求解基本的递推关系,例如使用某种形式的主定理。[运用]

AL/算法策略

[5 个核心一级学时,1 个核心二级学时]

教师可以选择在"基础数据结构及算法"的算法部分中讲授以下算法策略。这两个知识单元的总课时(18)可以以不同的方式划分,不过我们认为 1:2 的比例是合理的。

知识点:

[核心一级]

- 穷举算法。
- 贪心算法。
- 分治(参见 SDF/算法与设计)。
- 递归回溯。
- 动态规划。

[核心二级]

- 分支定界。
- 启发式算法。
- 规约:变换后求解。

学习成果:

[核心一级]

1. 为每一种算法策略(穷举、贪心、分治、递归回溯以及动态规划)找一个适用的实例。[熟悉]

2. 使用一个贪心算法来解决一个合适的问题,并判断选择的贪心规则是否能得到一个最优解。[评估]

3. 使用分治算法来解决一个相应的问题。[运用]

4. 使用递归回溯求解类似迷宫问题。[运用]

5. 使用动态规划来解决一个相应的问题。[运用]

6. 选择适当的算法来解决一个问题。[评估]

[核心二级]

7. 描述各种启发式方法。[熟悉]

8. 使用启发式方法来解决一个相应的问题。[运用]

9. 描述穷举算法和启发式策略之间的权衡。[评估]

10. 描述如何使用分支定界方法来改善启发式方法的性能。[熟悉]

AL/基础数据结构及算法

[9 个核心一级学时，3 个核心二级学时]

本知识单元直接建立在软件开发基础（SDF）的基础上，特别是 SDF/基本数据结构和 SDF/算法与设计中的内容。

知识点：

[核心一级]

- 简单的数值算法，如计算一列数字的平均数，求列表的最小、最大值以及模，近似求一个数的平方根，求最大公约数。
- 序列及二进制搜索算法。
- 最坏情况下二次的排序算法（选择排序，插入排序）。
- 最坏或平均情况下 O（nlogn）的排序算法（快速排序，堆排序，归并）。
- 哈希表，包括避免和解决冲突的策略。
- 二进制搜索树。

➢ 在二进制搜索树上常见的操作，如求最小值、最大值、插入、删除、遍历树等。

- 图以及图算法。

➢ 图的表示（如邻接表、邻接矩阵）。

➢ 深度和广度优先遍历。

[核心二级]

- 堆。
- 图和图论算法。

➢ 最短路径算法（Dijkstra 及 Floyd 算法）。

➢ 最小生成树（Prim 及 Kruskal 算法）。

- 模式匹配及字符串/文本算法（如子串匹配、常规表达式匹配、最长共同子序列算法）。

学习成果：

[核心一级]

1. 实现基本的数值算法。[运用]

2. 实现简单的搜索算法,并解释其时间复杂度上的差异。[评估]

3. 能够实现常见的二次及 O(N logN)的排序算法。[运用]

4. 说明哈希表的实现,包括避免冲突及解决方案。[熟悉]

5. 讨论用于排序、搜索以及哈希的主要算法的时间及内存效率。[熟悉]

6. 讨论除了计算效率之外会影响算法选择的因素,如编程时间、可维护性和在输入数据中特定应用的模式的使用。[熟悉]

7. 解释树的平衡性如何影响各种二叉搜索树的操作效率。[熟悉]

8. 使用基本的图论算法解决问题,包括深度优先和广度优先搜索。[运用]

9. 展示评估算法,从可能的一系列选项中选择,并提供选择的依据,以及在特定的上下文中实现该算法的能力。[评估]

[核心二级]

10. 描述堆的属性以及使用堆来实现一个优先队列。[熟悉]

11. 使用图论算法来解决问题,包括单源和所有对之间的最短路径,以及至少一个最小生成树算法。[运用]

12. 跟踪和/或实现一个字符串匹配算法。[运用]

AL/基础自动机的可计算性及复杂度

[3 个核心一级学时,3 个核心二级学时]

知识点:

[核心一级]

- 有限状态自动机。
- 正则表达式。
- 停机问题。

[核心二级]

- 下文无关文法(参见 PL/语法分析)。
- 介绍 P 问题和 NP 问题的定义,以及他们的联系和区别。
- 介绍 NP 完全问题的定义以及典型例子(例如,SAT 问题,背包问题)。

学习成果:

[核心一级]

1. 讨论有限状态自动机的概念。[熟悉]

2. 设计一个能接受某一种特定语言的确定性有限状态自动机。[运用]

3. 设计一种特定语言所对应的正则表达式。[运用]

4. 解释为什么停机问题没有算法解。[熟悉]

[核心二级]

5. 设计一种特定语言所对应的上下文无关语言。[运用]

6. P 问题和 NP 问题的定义。[熟悉]

7. 阐述 NP 完全问题的重要性。[熟悉]

AL/高级计算复杂度

[选修]

知识点：

- 回顾 P 问题和 NP 问题的定义；介绍 P 空间和 EXP。
- 多项式层次结构。
- NP 完全性（Cook 定理）。
- 经典的 NP 完全问题。
- 归约技术。

学习成果：

1. P 问题和 NP 问题的定义。（交叉参照 AL/基础自动机的可计算性及复杂度）。[熟悉]

2. P 空间的定义以及它和 EXP 空间的定义。[熟悉]

3. 阐述 NP 完全性的重要性。（交叉参照 AL/基础自动机的可计算性及复杂度）。[熟悉]

4. 提供经典的 NP 完全问题的例子。[熟悉]

5. 通过把一个问题归约到已知的经典 NP 完全问题来证明它是一个 NP 完全问题。[运用]

AL/高级自动机理论及可计算性

[选修]

知识点：

- 集合和语言。

➢ 正则语言。

➢ 回顾确定性有限状态自动机（DFAs）。

➢ 非确定性有限状态自动机（NFAs）。

➢ NFAs 和 DFAs 的等价性。

➢ 回顾正则表达式；它们和有限状态自动机的等价性。

➢ 闭包的性质。

➢ 证明语言的非正则性，通过泵原理或者替代的手段。

- 上下文无关语言。

➢ 下推自动机(PDAs)。

➢ PDAs 和上下文无关文法(CFG)的关系。

➢ 上下文无关语言的性质。

- 图灵机,或与它计算性等价的正式模型。
- 非确定型图灵机。
- Chomsky 层次结构。
- Church – Turing 理论。
- 可计算性。
- Rice 定理。
- 不可计算函数的例子。
- 不可计算性的影响。

学习成果:

1. 确定某一种语言在 Chomsky 层次结构中的位置(正则,上下文无关,递归可枚举)。[评估]

2. 等价转换一种语言的表示,包括 DFAs、NFAs 和正则表达式之间的转换,以及 PDAs 和 CFGs 之间的转换。[运用]

3. 解释 Church – Turing 理论及其重要性。[熟悉]

4. 解释 Rice 理论及其重要性。[熟悉]

5. 提供一些不可计算函数的例子。[熟悉]

6. 通过把一个问题归约到已知的经典不可计算问题来证明它的不可计算性。[运用]

AL/高级数据结构、算法及分析

[选修]

很多教学计划希望学生接触更高级的算法或分析方法。下面是选出的一些可用的高级知识点,它们可能在过去和未来的几年一直流行,但并不是所有高级算法都被包含在内。

知识点:

- 平衡树(如 AVL 树、红黑树、伸展树、堆)。
- 图(如拓扑排序、查找强联通片、图匹配)。
- 高级数据结构(如 B 树、斐波那契堆)。
- 串类数据结构和算法(如后缀数组、后缀树、Trie 树)。
- 网络流(如最大流[Ford – Fulkerson 算法]、最大流 – 最小割、最大二分匹

配）。

- 线性规划（如对偶法、单纯形法、内点法）。
- 数值理论算法（如模运算、素数测定、整数分解）。
- 几何相关算法（如点、线段、多边形[性质，交]、查找凸包、空间分解、冲突检测、几何查找/测距）。
- 随机算法（Randomized algorithms）。
- 概率算法（Stochastic algorithms）。
- 近似算法。
- 均摊分析。
- 概率分析。
- 在线算法和竞争分析。

学习成果：

1. 明白实际问题对应的算法解决方案（例如，图问题，线性规划问题等）。[评估]

2. 选择和应用高级算法中的技术（例如，随机，近似）去解决实际问题。[评估]

3. 选择和应用高级算法分析的技术（例如，均摊分析，概率分析等）去评价算法。[评估]

体系结构与组织(Architecture and Organization, AR)

计算机专业人士不应该只把电脑当作一个用神秘的方法来执行程序的黑盒。计算机体系结构与组织这个知识领域建立在系统基础(SF)上,用来加深对计算所依赖的硬件环境以及硬件环境提供给更高的软件层接口的理解。学生应该获取、理解和评价计算机系统的功能部件,包括它们的特点、性能、交互,尤其是利用并行来维持从当前到未来性能提升的挑战。学生需要理解计算机的体系结构,并通过分析编程者对并行和延迟的理解,使程序拥有较高的性能。选择使用一个系统时,学生应该能够在各个部件之间进行权衡,比如 CPU 的内核频率、每条指令执行所需的时钟周期、内存大小以及内存平均访问时间。

这些知识点的学习成果主要对应于核心内容,而且是为 16 个学时的最低体系结构教学计划而设计的。对于想要讲授超过最低学时要求的教学计划,AR 知识点可以再补充一组共两门的课程来提升到一个新的高度。如果教学计划想要覆盖选修知识点,可以通过补充两门后继课程甚至综合性更强的第三门课程来介绍这些知识点。

AR. 体系结构与组织(0 个核心一级学时,16 个核心二级学时)

	核心一级学时	核心二级学时	包含选修
AR/数字逻辑与数字系统		3	否
AR/数据的机器级表示		3	否
AR/汇编级计算机组成原理		6	否
AR/存储系统的组织与结构		3	否
AR/接口和通信		1	否
AR/功能性组成			是
AR/多处理器和可选体系结构			是
AR/性能优化			是

AR/数字逻辑与数字系统

[3 核心二级学时]

知识点:

- 计算机体系结构历史综述。

- 组合逻辑 vs 时序逻辑/现场可编程逻辑门阵列作为基础组合 + 时序逻辑构建模块。
- 多种表示/层的解释(硬件仅仅是另一层)。
- 计算机辅助设计工具:硬件编程与结构表示。
- 寄存器转移标记/硬件描述语言(Verilog/VHDL)。
- 物理约束(门延迟,扇入,扇出,能量/功率)。

学习成果:

1. 描述计算机技术部件的发展:从真空管到超大规模集成电路,从大型计算机体系结构到云数据中心的组成。[熟悉]

2. 理解现代计算机结构向多核发展的趋势以及并行化在所有的硬件系统中是固有的性质。[熟悉]

3. 解释"功耗墙"(power wall)在未来处理器性能的提升以及朝利用并行化的方向发展等方面的含义。[熟悉]

4. 描述清楚有很多等价的计算机功能表示,包括逻辑表达式以及门,能够用数学表达式来描述简单的组合以及时序电路的功能。[熟悉]

5. 设计计算机的基本构建块:算术逻辑单元(门级别),寄存器(门级别),中心处理单元(寄存器转移级别),内存(寄存器转移级别)。[运用]

6. 利用计算机辅助设计(CAD)工具获取、合成、仿真来评估一个简单的计算机设计中的构建块(比如:算术逻辑单元,寄存器以及寄存器之间的活动)。[运用]

7. 在逻辑电路级别,评价一个简单处理器执行的功能以及时序图行为。[评估]

AR/数据的机器级表示

[3 个核心二级学时]

知识点:

- 位、字节和字。
- 数值数据的表示和数值的基。
- 定点和浮点系统符号和补码表示。
- 非数值数据的表示(字符编码和图数据)。
- 记录和数组的表示。

学习成果:

1. 解释为什么包括计算机指令在内的所有信息在计算机内都是数据。[熟悉]

2. 解释在计算机内用其他格式来表示数值数据的原因。[熟悉]

3. 阐述负整数如何用符号位和补码表示。[熟悉]

4. 阐述定长数的表示如何影响计算的准确性和精度。[熟悉]

5. 阐述计算机内部如何表示非数值数据,例如:字符,字符串,记录和数组。[熟悉]

6. 把数值数据从一种格式转换到另一种格式。[运用]

7. 编写简单的汇编/机器层级的程序来做字符串的处理和修改。[运用]

AR/汇编级计算机组成原理

[6 个核心二级学时]

知识点:

- 冯·诺依曼机的基本组成。
- 控制单元;取指令,解码,执行。
- 指令集和类型(数据处理,控制,I/O)。
- 汇编/机器语言编程。
- 指令格式。
- 地址格式。
- 子程序请求和响应机制(参见 PL/语言翻译与执行)。
- I/O 和中断。
- 堆、静态变量、栈、程序段。
- 共享内存的多线程/多核心结构。
- SIMD 和 MIMD 的介绍和 Flynn 分类法。

学习成果:

1. 解释经典冯·诺依曼机的组成和它的主要功能单元。[熟悉]

2. 阐述一条指令是如何在经典冯·诺依曼机中执行的,并把这个执行过程扩展到多线程、多处理器同步和 SIMD 中。[熟悉]

3. 解释指令层级的并行和风险,以及他们在传统处理器流水线中是如何运作的。[熟悉]

4. 总结指令是如何在机器层和使用符号的汇编层表示的。[熟悉]

5. 演示如何建立高层语言的模式和汇编/机器语言符号之间的映射关系。[熟悉]

6. 解释不同的指令格式,比如每条指令的地址以及可变长度与固定长度格式。[熟悉]

7. 解释汇编级如何处理子程序调用。[熟悉]

8. 解释中断和 I/O 操作的基本概念。[熟悉]

9. 写简单的汇编语言程序段。[运用]

10. 展示如何在机器语言的级别实现基本的高级编程结构。[运用]

AR/存储系统的组织与结构

[3 个核心二级学时]

参见 OS/内存管理、OS/虚拟机。

知识点:

- 存储系统以及它们的技术。
- 内存层次结构:时间和空间位置的重要性。
- 主存组成和操作。
- 延迟、周期时间、带宽和交叉。
- 缓存(地址映射、块大小、替换和存储策略)。
- 多处理器缓存的一致性,使用内存系统进行内核同步,原子内存操作。
- 虚拟存储(页表,TLB)。
- 故障处理和可靠性。
- 错误编码、数据压缩以及数据完整性(参见 SF/冗余下的可靠性)。

学习成果:

1. 认识主要的内存技术(如 SRAM、DRAM、闪存、磁盘)以及他们相对的成本和性能。[熟悉]

2. 解释运行时内存延迟的影响。[熟悉]

3. 描述如何运用分级存储体系(缓存、虚存)来减少有效内存延迟。[熟悉]

4. 描述内存管理的原理。[熟悉]

5. 解释运用虚拟内存管理的系统的运作。[熟悉]

6. 计算在各种缓存、内存配置和混合指令及数据引用下的平均内存访问时间。[运用]

AR/接口与通信

[1 个核心二级学时]

关于系统视角的输入/输出过程和管理的讨论,参见操作系统(OS)知识领域。这里的重点是支持设备接口和处理器之间通信的硬件上的机制。

知识点:

- I/O 基本原理:握手,缓存,程序 I/O,中断 - 驱动 I/O。
- 中断结构:矢量中断和优先级中断,中断确认。

- 外部存储,物理组成,驱动。
- 总线:总线协议,仲裁,直接内存存取(DMA)。
- 网络介绍:通信网络是远程访问的另一层。
- 多媒体支持。
- 磁盘阵列(RAID)结构。

学习成果:

1. 解释如何使用中断来实现 I/O 控制和数据传输。[熟悉]
2. 识别计算机系统里各种总线。[熟悉]
3. 描述从磁盘驱动器进行数据访问的过程。[熟悉]
4. 比较常见的网络组织,比如以太网/总线、环网、交换和路由。[熟悉]
5. 识别多媒体访问和显示所需的跨层接口:从远程存储中获取图像,然后通过通信网路传输,暂存入本地存储,最后进行图形显示。[熟悉]
6. 描述 RAID 的优势和不足。[熟悉]

AR/功能性组成

[选修]

注意:本部分在计算机科学专业是选修,在计算机工程专业是核心。

知识点:

- 实现简单的数据通道,包括指令流水线、风险检测与解决。
- 控制单元:硬布线实现和微程序实现。
- 指令流水线。
- 指令级并行(ILP)简介。

学习成果:

1. 比较数据通道可选择的实现方法。[熟悉]
2. 讨论控制点的概念以及硬布线实现和微程序实现生成控制信号的过程。[熟悉]
3. 解释通过流水线实现的基本的指令级并行以及可能产生的主要的风险。[熟悉]
4. 设计并实现一个完整的处理器,包括数据通道和控制。[运用]
5. 对一个给定的处理器和存储系统,确定它的平均指令周期。[评估]

AR/多处理器和可选体系结构

[选修]

这里指 SIMD 和 MIMD 结构的硬件实现。参见 PD/并行体系架构。

知识点:

- 幂律。
- 举例介绍 SIMD 和 MIMD 指令集和结构。
- 互联网络(超立方,洗牌交换,无线网格,交叉开关矩阵)。
- 多处理器共享内存系统和存储一致性。
- 多处理器缓存一致性。

学习成果:

1. 讨论超越了经典冯·诺依曼模型的并行处理的概念。[熟悉]
2. 描述交替并行结构如 SIMD 和 MIMD。[熟悉]
3. 解释互联网络的概念并描述不同的方法。[熟悉]
4. 讨论多处理系统在内存管理方面需要注意的问题,并描述它们是如何解决的。[熟悉]
5. 描述内存底板、处理器内存互联和通过网络的远程存储之间的区别,以及它们对访问延迟和程序性能的影响。[熟悉]

AR/性能优化

[选修]

知识点:

- 超标量体系结构。
- 分支预测、预测执行、无序执行。
- 预读。
- 向量处理器和 GPU。
- 多线程硬件支持。
- 可扩展性。
- 其他结构,如 VLIW/EPIC、加速器以及其他类型的特殊用途处理器。

学习成果:

1. 描述超标量体系结构以及它们的优势。[熟悉]
2. 解释分支预测的概念及其功能。[熟悉]
3. 说明预读的成本和效益。[熟悉]
4. 解释推测执行概念并举出能够证明它的条件。[熟悉]
5. 讨论多线程在一个结构中提供的性能优点以及影响它获得最大收益的因素。[熟悉]
6. 描述可扩展性和性能之间的关联。[熟悉]

计算科学（Computational Science，CN）

计算科学是计算机科学应用的一个领域，也就是将计算机科学应用于解决多个学科领域的计算问题。在“计算科学介绍（Introduction to Computational Science）”[3]这本书中，作者提出以下定义：“计算科学领域将计算机模拟、科学计算可视化、数学建模、计算机编程以及数据结构、网络、数据库设计、符号计算和高性能计算与各学科结合。”计算科学，很大程度上侧重处理数据和信息的算法，包括算法的理论、设计和实现，这可以追溯到四千多年前帮助人类计算的工具。人们创建了各种系统来计算天文位置。Ada Lovelace 的编程目的是为了计算伯努利数字。在十九世纪后期，机械计算器刚刚能用，立即被科学家使用起来。科学家和工程师对计算的需求长时间驱动了计算方法的研究与创新。随着计算机解决问题能力的提升，计算科学在广度和重要性上也在增长。它独自成为了一门学科[2]并且被认为是“正在崛起的五个大学专业之一[1]”。在计算科学的大领域下可喜地出现了各种子领域，包括计算生物学、计算化学、计算力学、计算考古学、计算金融、计算社会学和计算取证。

计算科学的一些基本概念与每一位计算机科学家都有密切关系（如建模和仿真），而计算科学的主题是计算机科学本科教育中非常有价值的组成部分。这个领域介绍了很多有价值的思想和技术，包括数值表示的精度、误差分析、数值方法、并行体系结构及算法、建模和仿真、信息可视化、软件工程和优化。与计算科学相关的主题包括程序设计的基本概念（SDF/程序设计基本概念）、算法设计（SDF/算法与设计）、程序测试（SDF/开发方法）、数据表示（AR/数据的机器级表示）、基本的计算机体系结构（AR/存储系统的组织与结构）。同时，选修了这个领域相关课程的同学有机会将这些技术实践于广泛的应用领域，如分子和流体动力学、天体力学、经济学、生物学、地质、医学和社会网络分析应用等。许多在这些领域应用的技术要求高等数学知识，如微积分、微分方程、线性代数等。这里假定学生已经在其他地方获得了所需的数学背景。

在计算科学领域中，术语“运行”、“修改”和“创建”往往用于描述理解的层次。这一章遵循本卷中其他章节的约定，使用术语“熟悉”、“运用”和“评估”。

参考文献

[1] Fischer, K. and Glenn, D., “5 College Majors on the Rise,” *The Chronicle of Higher*

Education, August 31, 2009.

[2] President's Information Technology Advisory Committee, 2005: p. 13. http://www.nitrd.gov/pitac/reports/20050609_computational/computational.pdf

[3] Shiflet, A. B. and Shiflet, G. W. *Introduction to Computational Science: Modeling and Simulation for the Sciences*, Princeton University Press, 2006: p. 3.

CN. 计算科学(1 个核心一级学时,0 个核心二级学时)

	核心一级学时	核心二级学时	包含选修
CN/建模与仿真引言	1		否
CN/建模与仿真			是
CN/处理			是
CN/交互式可视化			是
CN/数据、信息和知识			是
CN/数值分析			是

CN/建模与仿真引言

[1 个核心一级学时]

抽象化是计算机科学的一个基础概念。计算的一个主要方法就是将真实世界抽象化,建立一个可以在一台机器上模拟的模型。计算机科学的兴起可以追溯对事物进行建模的研究,例如对炮弹轨迹的事物进行建模,对密码协议进行建模,而这两个问题推动了二十世纪四十年代中前期早期计算系统的发展。

对于真实世界中系统的建模和仿真为计算机科学家们提供了基本的知识,为计算科学奠定了基础。任何建模和仿真概论都应包括或要求计算导论方面的知识。此外,一些通用的建模与仿真技术,数据可视化方法和软件测试及评估的机制也很重要。

知识点:

- 对一些情况的抽象模型。
- 作为动态建模的仿真。
- 仿真技术和工具,如物理仿真、人类指导的仿真、虚拟现实。
- 验证模型的基本方法(例如,将仿真的输出与真实数据或另一个模型的输出做比较)。
- 以被建模的系统相关的格式来演示结果。

学习成果:

1. 解释能够允许一台机器来解决一个问题所使用的建模概念及抽象化。[熟悉]

2. 描述建模与仿真之间的关系,即将仿真考虑为动态建模。[熟悉]

3. 对一个真实状况创建一个简单正式的数学模型并将该模型用于仿真。[运用]

4. 区分不同类型的仿真,包括物理仿真、人类指导仿真和虚拟现实。[熟悉]

5. 描述几种验证模型的方法。[熟悉]

6. 创建一个简单的仿真结果演示。[运用]

CN/建模与仿真

[选修]

知识点:

- 建模及仿真包含优化的目的;支持决策、预测、安全考虑、培训与教育。
- 性能、准确性、有效性和复杂性等方面的权衡。
- 仿真过程;确定关键特性或行为,简化假设;结果验证。
- 建立模型:使用数学公式或方程、图、约束;方法和技巧;动态系统中时间步(time stepping)的使用。
- 正式的模型和建模技术:简化假设并避免细节的数学描述。技术的例子包括:

➢ 蒙特卡洛方法。

➢ 随机过程。

➢ 排队论。

➢ Petri 网和着色 Petri 网。

➢ 图结构包括有向图、树、网络等。

➢ 博弈,博弈论,使用博弈论建模。

➢ 线性规划及其扩展。

➢ 动态规划。

➢ 微分方程:常微分方程、偏微分方程。

➢ 非线性技术。

➢ 状态空间及转换。

➢ 在各种环境中评估和评价模型和仿真;核实及验证。[1] 模型和仿真。

➢ 重要的应用领域包括卫生保健和诊断、经济学和金融、城市规划、科学和工程。

➢ 支持仿真与建模的软件;软件包,语言。

学习成果：

1. 解释并举例说明在一系列重要的应用领域中的仿真和建模的好处。[熟悉]

2. 表现将建模与仿真技术应用于一系列问题领域的能力。[运用]

3. 解释一种特定的建模方法的构造和概念。[熟悉]

4. 解释核实[1] 与验证一个模型的区别；对于特定的例子展示该区别。[评估]

5. 核实及验证仿真结果。[评估]

6. 评估一个仿真，指出优缺点。[评估]

7. 对于一个给定的问题或情况选择适当的建模方法。[评估]

8. 比较同一情况下不同的仿真结果并解释其区别。[评估]

9. 从一个系统的仿真结果推断该系统的行为。[评估]

10. 对一个现有的模型进行扩展或改变以适应新的情况。[评估]

CN/处理

[选修]

处理部分包括其他知识领域的众多知识点。具体来说，处理部分应覆盖包括硬件体系结构的讨论，如并行系统、存储器层次结构以及处理器之间的互联。这些知识点在 AR/接口和通信、AR/多处理器和可选体系结构、AR/性能优化等部分覆盖。

知识点：

- 基本的编程概念：

➢ 由有限个定义明确的步骤组成的算法的概念，每一个步骤和整个过程都在有限的时间内完成。

➢ 知名算法的例子，如排序和检索。

➢ 分析的概念：了解问题的要求是什么，如何用一个算法来解决问题，信息可以如何表示使得一台计算机能够处理。

➢ 发展或确认一个工作流。

➢ 将算法转换为机器可执行代码的过程。

➢ 软件流程包括生命周期模型、需求、设计、实现、验证和维护。

➢ 数据计算机算法的计算机表示形式。

[1] 核实（*Verification*）指这个模型的计算正确。例如，假设我们声称计算总时间，而相关计算确实做到这一点。验证（*Validation*）是检查是否这个模型与真实情况相符。

- 数值方法。

➢ 数值拟合算法(如牛顿法)。

➢ 数值计算的体系结构,包括并行体系结构。

- 并行与分布式计算的基本属性:

➢ 带宽。

➢ 滞后时间。

➢ 可扩展性。

➢ 粒度。

➢ 并行性包括任务、数据和事件并行性。

➢ 并行体系结构包括处理器架构,内存和高速缓存。

➢ 并行编程模式包括线程、消息传递、事件驱动技术、并行软件的体系结构和 MapReduce。

➢ 网格计算。

➢ 体系结构对计算时间的影响。

➢ 并行的总时间到科学曲线:事物的连续性

- 计算代价,例如,重新计算一个值的代价与存储和查找的代价。

学习成果:

1. 解释算法的特点和定义属性以及它们与计算机处理的关系。[熟悉]

2. 分析简单的问题说明,识别相关的信息并选择适当的处理方法来解决问题。[评估]

3. 确定或简述一个现有计算过程的工作流,例如基于实验数据创建一个图的过程。[熟悉]

4. 描述将一个算法转换为机器可执行代码的过程。[熟悉]

5. 总结软件开发的各个阶段,比较几种常见的生命周期模型。[熟悉]

6. 说明数据如何在计算机中表示。比较整数和浮点数的表示形式。描述数据表示中的下溢、上溢、四舍五入及截断误差。[熟悉]

7. 使用标准数值算法求解常微分方程和偏微分方程。使用计算系统求解方程组。[运用]

8. 描述带宽、延迟、可扩展性和粒度的基本属性。[熟悉]

9. 描述并行度的级别,包括任务、数据和时间的并行度。[熟悉]

10. 比较和对比不同的并行编程模式,认识他们的优缺点。[评估]

11. 确定会影响计算的正确性和计算效率的问题。[熟悉]

12. 并行计算中的设计、编码、测试和调试。[运用]

CN/交互式可视化

[选修]

这个子领域与建模和仿真相关。大多数知识点都在本指南的其他知识领域中有详细讨论。存在很多方法来展示数据和信息,包括沉浸性、现实主义、可变视角;触觉和平视显示器、可听化及手势映射。

交互式可视化一般需要理解人类感知(GV/基本概念),图形管道、几何表示法和数据结构(GV/基本概念),2D 和 3D 渲染、表面和体绘制(GV/基本绘别、GV/几何建模和 GV/高级绘制),通过 API 使用标准输入组件,如菜单、滑块和按钮以及用于数据显示的标准输出组件,包括图表、图形、表和直方图(HCI/交互设计、HCI/交互系统编程)来开发用户界面。

知识点:

- 数据可视化的原理。
- 图形和可视化算法。
- 图像处理技术。
- 可扩展性问题。

学习成果:

1. 在易用性、易学性及代价等方面,与常见的计算机接口机制比较。[评估]
2. 使用标准的 API 和工具创建数据的视觉表示,包括图形、图表、表格和直方图。[运用]
3. 描述几种使用计算机交互和数据处理的方法。[熟悉]
4. 从一个数据集提取有用的信息。[评估]
5. 对特定问题分析和选择可视化技术。[评估]
6. 描述将数据分析从小型扩展至大型数据集相关的问题。[熟悉]

CN/数据、信息和知识

[选修]

大多数知识点都在本指南的其他知识领域中有详细讨论,具体包括信息管理(IM/信息管理概念、IM/数据库系统和 IM/数据模型),算法与复杂度(AL/基础分析、AL/基础数据结构及算法)及软件开发基础(SDF/程序设计基本概念、SDF/开发方法)。

知识点:

- 内容管理模型、框架、系统、设计方法(如 IM/信息管理基础中)。
- 内容的数字表示,包括数字、文本、图像(如光栅和矢量)、视频(如

QuickTime、MPEG2、MPEG4）、音频（如写分数、MIDI、采样数字化声轨）和动画；复杂/复合/聚合对象；FRBR。

- 数字内容创作/获取和保存，包括数字化、采样、压缩、变换、转换/翻译、迁移/仿真、数据抓取（crawling）、数据采集（harvesting）。
- 内容结构/管理，包括数字化图书馆和以下静态/动态/流等方面：

➢ 数据：数据结构、数据库。

➢ 信息：文档集，多媒体池，超媒体库（超文本、超媒体），目录，存储库。

➢ 知识：本体、三元组存储、语义网络、规则。

- 处理和模式识别，包括索引、搜索（例如，查询和查询语言，集中式/联邦式/P2P）、检索、聚类、分类、分析/挖掘/提取，渲染、报告、事物处理。
- 用户/社群对展示和交互的支持，包括浏览、搜索、过滤、路由、可视化、共享、协作、评价、批注、个性化、推荐。
- 采用相关的系统/软件建模、设计、逻辑和物理实现。

学习成果：

1. 对于一个计算科学应用，确定所有的数据、信息和知识要点和相关的组织。[评估]

2. 描述如何表示用于处理的数据和信息。[熟悉]

3. 描述对于数据、信息和知识的典型用户要求。[熟悉]

4. 选择一个合适的系统或软件来实现管理数据、信息和知识。[评估]

5. 列出并描述一个计算科学应用所需的报告、事务记录及其他处理。[熟悉]

6. 比较与典型计算科学应用相关的数据库管理、信息检索和数字化图书馆系统。[评估]

7. 为一些计算科学用户/社群设计一个数字图书馆，提供适当的内容和服务。[用法]

CN/数值分析

[选修]

参见 AR/机器级的数据表示。

知识点：

- 错误、稳定性、收敛性，包括截断和四舍五入。
- 函数逼近，包括泰勒展开、插值法、外推法和回归。
- 数值微分和积分（辛普森规则，显式和隐式方法）。
- 微分方程（欧拉方法、有限差分法）。

学习成果：

1. 定义错误、稳定性、计算机精度等概念以及计算近似的不确切性。[熟悉]

2. 实现函数逼近的泰勒展开、插值法、外推法及回归算法。[运用]

3. 实现微分和积分算法。[运用]

4. 实现算法求解微分方程组。[运用]

离散结构(Discrete Structures, DS)

离散结构是计算机科学的基本理论。这里所谓基本理论,意味着很少有计算机科学的学者将离散结构作为主要研究方向,但从事计算机专业其他众多方向的学者却需要理解和运用离散结构的相关概念。离散结构涉及到集合论、逻辑、图论和概率论等方向中的一些重要理论。

离散结构的相关理论被广泛应用于数据结构和算法中,在计算机科学的其他方向也有很多应用。例如,构造和理解证明(形式化符号化证明或不够形式化但数学上严格的论断)的能力对于计算机科学的几乎每个方向都很重要,包括(但不限于)形式化规范说明、验证、数据库及密码学等;在网络、操作系统和编译器等方向会用到图论的相关概念;在软件工程和数据库方向会用到集合论的相关概念;在智能系统、互联网及很多计算机应用方向会用到概率论。

既然离散结构为计算机的很多其他方向提供了基础理论,它与其他方向的分界就更值得关注,特别是它与算法及算法复杂性、软件开发基础、程序设计语言和智能系统等方向的分界往往并不鲜明。实际上,不同的教学机构都会组织相关课程覆盖这些理论,但组织形式可能截然不同。有些机构会将这部分内容集中用一、两门名为“离散结构”或“离散数学”的课程进行讲解;而另一些机构则会将这些内容整合到程序设计、算法及人工智能等课程中;另外也有很多机构会将前面两种形式进行混合,例如,先用一门集中介绍性课程覆盖一部分内容,再用更多的高级专题性课程覆盖其他内容。

DS. 离散结构(37 个核心一级学时,4 个核心二级学时)

	核心一级学时	核心二级学时	包含选修课程
DS/集合、关系与函数	4		否
DS/基础逻辑	9		否
DS/证明方法	10	1	否
DS/计数基础	5		否
DS/树和图	3	1	否
DS/离散概率	6	2	否

DS/集合、关系与函数

[4 个核心一级学时]

知识点:

- 集合

➢ 维恩图。

➢ 并集、交集、补集。

➢ 笛卡儿积。

➢ 幂集。

➢ 有限集合的基数。

- 关系。

➢ 自反性、对称性、传递性。

➢ 等价关系、偏序关系。

- 函数。

➢ 满射、单射、双射。

➢ 反函数。

➢ 函数的复合。

学习成果:

1. 通过示例解释函数、关系和集合的基本术语。[熟悉]

2. 使用集合、函数和关系的相关操作。[运用]

3. 为实际例子建立适合的集合、函数或关系模型,并根据上下文解释相关操作和术语。[评估]

DS/基础逻辑

[9 个核心一级学时]

知识点:

- 命题逻辑(交叉参照 IS/知识推理)。
- 逻辑联结词。
- 真值表。
- 范式(合取范式、析取范式)。
- 合式公式的有效性。
- 命题推理定律(肯定前件式和否定后件式的概念)。
- 谓词逻辑。

➢ 全称量词与存在量词。

- 命题逻辑和谓词逻辑的局限(例如,表达能力问题)。

学习成果：

1. 将自然语言的逻辑陈述转换为命题逻辑和谓词逻辑的表达式。[运用]

2. 学会使用符号化的命题逻辑和谓词逻辑等形式化方法，例如，计算公式的有效性、公式的规范化。[运用]

3. 使用命题推理定律进行命题逻辑和谓词逻辑的证明。[运用]

4. 描述如何使用符号化逻辑对真实世界中的场景或应用进行建模，包括一些涉及计算机的场景，例如，软件分析（如程序正确性）、数据库查询以及算法。[运用]

5. 将形式化逻辑证明和/或非形式化的但严格的逻辑推理应用到实际问题中，例如，预测软件行为或解决智力游戏等问题。[运用]

6. 说明命题逻辑和谓词逻辑的长处和局限。[熟悉]

DS/证明方法

[10 个核心一级和 1 个核心二级学时]

知识点：

[核心一级]

- 蕴含、等价、逆命题、否命题、逆否命题、否定和矛盾等概念。
- 数学证明的架构。
- 直接证明。
- 反例证伪。
- 反证法。
- 数学归纳法。
- 结构归纳法。
- 弱归纳法与强归纳法（即，第一类数学归纳法和第二类数学归纳法）。
- 数学上的递归定义。

[核心二级]

- 良序关系。

学习成果：

[核心一级]

1. 识别给定证明中使用的证明方法。[熟悉]

2. 列出本单元描述的每种证明方法的基本框架，包括直接证明、反证法和归纳法。[运用]

3. 正确应用每种证明方法（直接证明、反证法和归纳法）得出合理的论断。[运用]

4. 确定哪种证明方法最适合给定问题。[评估]

5. 解释数学归纳法和结构归纳法思路的关系与递归和递归定义结构之间的相似之处。[评估]

6. 解释弱归纳法与强归纳法之间的关系,并给出正确使用每种方法的例子。[评估]

[核心二级]

7. 陈述良序原理及其与数学归纳法之间的关系。[熟悉]

DS/计数基础

[5 个核心一级学时]

知识点:

- 计数论。
- 集合的基数与计数。
- 加法法则和乘法法则。
- 容斥原理。
- 等差数列和等比数列。
  - 鸽巢原理。
  - 排列与组合。
- 基本定义。
- 帕斯卡恒等式。
- 二项式定理。
  - 求解递推关系(参见 AL/基础分析)。
- 简单递推关系举例,如斐波那契数列。
- 其他例子,体现解法的多样性。
  - 模运算基础。

学习成果:

1. 计数论的应用,包括加法法则与乘法法则、容斥原理、等差和等比数列。[运用]

2. 在形式化证明中使用鸽巢原理。[运用]

3. 计算集合的排列和组合,并在特定应用背景下解释它们的意义。[运用]

4. 为现实问题找到适合的计数方法,例如排列圆桌座位的方法数、满足某些限制条件的座位排列方法数或特定牌型的组合方法数(如满堂红)。[运用]

5. 求解多种基本的递推关系。[运用]

6. 分析问题背后的递推关系。[运用]

7. 计算涉及模运算的算式。[运用]

DS/树和图

[3 个核心一级和 1 个核心二级学时]

参见 AL/基础数据结构及算法,尤其是与图遍历策略的关系。

知识点:

[核心一级]

- 树。

➢ 性质。

➢ 遍历策略。

- 无向图。
- 有向图。
- 带权图。

[核心二级]

- 生成树/生成森林。
- 图的同构。

学习成果:

[核心一级]

1. 通过实例表达图论的基本术语,以及每种类型的树/图的性质和特例。[熟悉]

2. 详细说明树和图的不同遍历方法,包括树的前序、后序及中序遍历。[运用]

3. 使用适当的树和图对计算机专业的多种实际问题进行建模,例如,表达网络拓扑结构、组织层次化的文件系统。[运用]

4. 说明在数据结构、算法、证明方法(结构归纳法)和计数等方面如何应用了树和图的相关概念。[运用]

[核心二级]

5. 解释如何构造图的生成树。[运用]

6. 判定两个图是否同构。[运用]

DS/离散概率

[6 个核心一级和 2 个核心二级学时]

知识点:

[核心一级]

- 有限概率空间、事件。

- 概率公理与概率测度。
- 条件概率、贝叶斯定理。
- 独立性。
- 整数随机变量(伯努利分布、二项分布)。
- 期望,包括期望的线性性质。

[核心二级]

- 方差。
- 条件独立性。

学习成果:

[核心一级]

1. 计算基础问题(如赌博游戏)中事件的概率及随机变量的期望。[运用]
2. 区分独立和非独立事件。[运用]
3. 识别二项分布的实例,利用该分布计算概率。[运用]
4. 使用贝叶斯定理计算问题中的条件概率。[运用]
5. 使用概率相关工具解决问题,例如,分析算法的平均情况、分析散列函数。[运用]

[核心二级]

6. 计算给定概率分布的方差。[运用]
7. 解释为什么独立事件有可能是条件非独立的(反之亦然)。找出现实世界中这样的例子。[运用]

图形学与可视化(Graphics and Visualization, GV)

计算机图形学通常是用来描述用计算机生成和处理图像的术语。它是通过计算使得视觉交流成为可能的一门科学。它的用途包括卡通、电影特效、视频游戏、医疗图像、工程,也包括科学、信息和知识的可视化。传统上,本科阶段的图形学教学内容集中在绘制、线性代数和现象学方法。最近,教学重点也开始包括物理、数字积分、可扩展性和特殊用途的硬件。为了使学生能够熟练使用和生成计算机图形,必须涉及许多基于特定应用的议题,如文件格式、硬件接口和应用程序接口。这些议题变化很快,而随之变化的描述尝试能够避免对它们的过度规范。图形学与可视化覆盖的范围可划分为几个相互关联的领域。

- 基本原理:计算机图形学依赖于对人们如何通过视觉来感知信息和信息如何能够在一个显示设备上绘制出来的理解。每一个计算机科学家应该对图形学在何处和如何被恰当地应用以及牵涉到显示绘制的基本过程有一些理解。
- 建模:需要显示的信息必须以一些形式,通常是一种表示形状和形态数学规范的形式,被编码在计算机内存中。
- 绘制:绘制是显示包含在一个模型内信息的过程。
- 动画:动画是一种使图像看上去在移动的绘制方式以及对模型时间变化的合成或获取。
- 可视化:可视化领域是探索从种类繁多的应用数据中确定和表达相互关联的结构和关系。展示的首要目的应该是交流数据集中的信息以加强理解。
- 计算几何:计算几何是对以几何描述的算法进行研究。

和图形学与可视化有关系的是智能系统知识领域中的机器视觉和图像处理,以及算法与复杂度知识领域中的计算几何的算法。虚拟现实中的知识点可以在人机交互知识领域中找到。

本节描述假设学生熟悉数据表达、抽象和编程实现的基础概念。

GV. 图形学与可视化(2 个核心一级学时,1 个核心二级学时)

	核心一级学时	核心二级学时	包含选修课程
GV/基本概念	2	1	是
GV/基本绘制			是
GV/几何建模			是

续表

	核心一级学时	核心二级学时	包含选修课程
GV/高级绘制			是
GV/计算机动画			是
GV/可视化			是

GV/基本概念

[2 个核心一级和 1 个核心二级学时]

对几乎每一个计算机科学家和软件开发工程师来说,理解人们如何与机器相互影响是非常重要的。虽然这些知识点可能在一个标准的本科生图形学课程中覆盖到,它们也可能被覆盖在入门级的计算机科学和编程课程中。这里包含立即和保留模式的部分动机是这些模式类似于轮询程序对事件驱动程序的重要性。这是计算机科学中的一个基础性问题:是有一个按钮对象抑或仅仅是在屏幕上显示一个按钮?注意到本节中大部分的输出是在知识层面上,它们当中的许多将在后续章节中更深入地涉及。

知识点:

[核心一级]

- 包括用户界面、音视频编辑、游戏引擎、计算机辅助设计、可视化和虚拟现实的媒体应用。
- 人类感知的模拟信号、分辨率和界限的数字化,举例来说,视觉显示的像素、激光打印机的点阵和音频的采样(参见 HCI/基础)。
- 标准应用程序界面的使用,用以构建标准媒体格式的用户界面和显示(参见 HCI/图形用户界面构建)。
- 标准媒体格式,包括无损和有损格式。

[核心二级]

- 加性和减性颜色模型(CMYK 和 RGB)和为什么这些模型能够提供一系列的颜色?
- 存储数据和重新计算数据之间的权衡,矢量和光栅化图像表示的具体化。
- 用一系列静态图像表示的动画。

[选修]

- 双缓冲技术。

学习成果

[核心一级]

1. 向人们展示数字化的常见用途(例如,计算机图形学,声音)。[熟悉]

2. 用通用的术语解释模拟信号如何被离散的采样合理地表达。例如,图像如何能够用像素来表达。[熟悉]

3. 解释人类感知的局限性如何对模拟信号数字表达的选择产生影响。[熟悉]

4. 使用一个标准的应用程序接口来构建一个简单的用户界面。[运用]

5. 描述有损和无损图像压缩技术之间的差异,比如常见图形图像文件格式 JPG、PNG、MP3、MP4 和 GIF 中所反映出的差异。[熟悉]

[核心二级]

6. 描述颜色模型以及他们在图像显示设备中的用法。[熟悉]

7. 描述存储信息和存储足够的信息以再造信息之间的权衡,就好比矢量显示和光栅化显示之间的差异。[熟悉]

[选修]

8. 描述从一系列离散的图像帧中产生连续运动的基本过程(有时称为“闪光融合”)。[熟悉]

9. 描述双缓冲技术如何能够从动画中去除闪烁。[熟悉]

GV/基本绘制

[选修]

本节描述几乎每一个本科生的计算机图形学课程都会覆盖到,同时对图形学进一步学习也很有必要的基本绘制和基础图形学技术。采样和反走样与数字化的效果有关,它们出现在计算的其他领域比如音频采样中。

知识点:

- 真实感绘制,例如,光的发射和散射以及与数值积分的关联。
- 正向和反向绘制(即光线投射和光栅化)。
- 多边形表示。
- 基本辐射度法,相似三角形和投影模型。
- 仿射和坐标系变换。
- 光线跟踪。
- 可见性和遮挡,包括这类问题的解法如深度缓存技术、画家算法和光线跟踪。
- 前向和反向绘制方程。

- 简单三角形光栅化。
- 基于着色器应用程序接口的绘制。
- 纹理映射,包括缩小和放大(例如,三线性 MIP 映射)。
- 应用于绘制的空间数据结构。
- 采样和反走样。
- 场景图和图形流水线。

学习成果

1. 讨论光线传输问题和它与数字积分的关联,即光线是被发射出来,在场景周围散射,并被眼睛测量到。[熟悉]

2. 描述基本的图形流水线以及前向和反向绘制如何作为因素考虑其中。[熟悉]

3. 创建一个显示简单图形图像的三维模型的程序。[运用]

4. 通过将点从(x,y,z)转换到(x/z,y/z,1)的方式,从相似三角形中推导直线透视。[运用]

5. 应用仿射变换获得二维和三维点。[运用]

6. 应用三维坐标系和必要的变化将二维变换操作扩展为处理三维的变换。[运用]

7. 对比前向和反向绘制。[评估]

8. 解释纹理映射、采样和反走样的概念和应用。[熟悉]

9. 解释面向可见性问题的光线跟踪和光栅化的对偶关系。[熟悉]

10. 实现在简单二维图像上执行变换和裁剪操作的简单过程。[运用]

11. 实现一个基于顶点缓存器和着色器、采用光栅化应用程序接口的简单实时绘制。[运用]

12. 比较不同的绘制技术。[评估]

13. 基于分辨率和颜色编码计算空间开销。[评估]

14. 基于刷新频率和光栅化技术计算时间开销。[评估]

GV/几何建模

[选修]

知识点:

- 诸如求交计算和近距离测试的基本几何操作。
- 立体、立体像素和基于点的表达。
- 曲线和曲面参数多项式。
- 曲线和曲面的隐式表达。

- 诸如多项式曲线、贝塞尔曲线、样条曲线曲面、非均匀有理 B 样条和水平集方法的逼近技术。
- 曲面表示技术,包括细分、网格表示、网格光顺、诸如 Delaunay 三角化和 Marching cubes 算法的网格生成。
- 空间剖分技术。
- 诸如分形、生成模型和 L 系统的过程模型。
- 与程序语言相互对照的分形图(生成图片的语法)。
- 弹性变形和自由变形模型。
- 细分曲面。
- 多分辨率建模。
- 重构。
- 构造实体几何(CSG)表达。

学习成果:

1. 同时使用隐式和参数化形式表达曲线和曲面。[运用]
2. 通过曲面细分来创建简单多面体模型。[运用]
3. 从一张隐式曲面中生成一个网格表达。[运用]
4. 采用过程式方法生成一个分形模型或地形。[运用]
5. 从激光扫描仪获取的点云中生成一个网格。[运用]
6. 从诸如立方体和二次曲面的简单基元中构建 CSG 模型。[运用]
7. 从时间空间复杂度和图像质量方面对比建模方法。[评估]

GV/高级绘制

[选修]

知识点:

- 绘制方程的解答和近似,举例来说:

➢ 分布式光线跟踪和路径跟踪。

➢ 光子映射。

➢ 双向路径跟踪。

➢ Reyes(微多边形)绘制。

➢ 蒙特卡洛光传输(MLT)。

- 时间(运动模糊)、镜头位置(焦距)、连续频率(颜色)和它们对绘制的影响。
- 阴影映射。
- 遮挡剔除。

- 双向反射分布函数(BSDF)理论和微平面模型。
- 次表面散射。
- 面光源。
- 分层深度缓存技术。
- 光场,基于图像的绘制。
- 非真实感绘制。
- GPU 架构。
- 包括如下的人类视觉系统:对光的自适应,对噪声的敏感度以及闪光融合。

学习成果:

1. 论证一个算法如何估计绘制方程的解。[评估]
2. 证明一个绘制算法的某些性质,例如,完整性、一致性和公正性。[评估]
3. 分析一个简单算法的带宽和计算需要。[评估]
4. 在光栅化的应用程序接口中实现一个非平凡的阴影算法(例如,卡通渲染或级联阴影图)。[运用]
5. 讨论一个专门的艺术技巧是如何在渲染器中实现的。[熟悉]
6. 解释如何确认用来创建一个独特图像所需的图形学技巧。[熟悉]
7. 实现任何一种在个体像素级别上、采用原始图形系统的指定图形学技术。[运用]
8. 使用一个简单(例如,Phone 模型)双向反射分布函数加反射和折射的场景,实现一个光线跟踪器。[运用]

GV/计算机动画

[选修]

知识点:

- 前向和逆运动学。
- 碰撞检测和响应。
- 使用噪声,规则(群落/群体)和粒子系统的过程动画。
- 蒙皮算法。
- 基于物理的运动,包括面向织物、肉体和头发的刚体动力学,物理粒子系统,质点 - 弹簧网络。
- 关键帧动画。
- 样条。
- 实现旋转的数据结构,比如四元数。

- 照相机动画。
- 运动捕捉。

学习成果:

1. 使用前向运动学方法,计算模型部件的位置和定向。[运用]

2. 使用逆运动学方法,从一个位置和定向中计算模型关节型部件的定向。[运用]

3. 描述旋转的不同表达方法之间的折衷。[评估]

4. 实现用以产生中间位置和定向的样条插值方法。[运用]

5. 实现牛顿力学中粒子动力学物理建模的算法,例如,Witkin&Kass 方法、snakes and worms 方法、辛欧拉方法、Stormer/Verlet 方法或者中点欧拉法。[运用]

6. 讨论面向弹道轨迹建模的一些流体动力学方法背后的基本理念,例如,液体泼溅、灰尘、火焰或者烟雾。[熟悉]

7. 使用常见动画软件基于元球和骨架来创建简单器官形式。[运用]

GV/可视化

[选修]

可视化与人机交互以及计算科学知识领域有着很强的关联。读者可以参考人机交互和计算科学知识领域以获得更多与用户群体和界面评估有关的知识点。

知识点:

- 二维/三维标量场的可视化:颜色映射,等势面
- 直接体数据渲染:光线投射,传递函数,分割
- 以下内容的可视化:
  - ➢ 矢量场和流数据。
  - ➢ 时变数据。
  - ➢ 高维数据:维度约减,并行坐标。
  - ➢ 非空间数据:多变量,树/图结构,文本。
- 驱动视觉抽象的感知和认知基础。
- 可视化设计。
- 可视评价方法。
- 可视化应用。

学习成果:

1. 描述标量和矢量可视化的基本算法。[熟悉]

2. 描述可视化算法在精度和性能方面权衡。[评估]

3. 为一个数据特性和应用任务的特别组合提出合适的可视化设计。[评估]

4. 分析一个为特别任务所定制的可视化的有效性。[评估]

5. 设计用以评估可视化算法或系统效用的一个过程。[评估]

6. 确认一系列包括科学、医学和数学数据表达的可视化应用:流场可视化和空间分析。[熟悉]

人机交互(Human – Compater Interaction,HCI)

人机交互是研究如何设计人类活动和计算系统间的交互,以及如何构建人机界面来支持交互的学科。

用户和计算机之间的交互发生在用户界面上。用户界面既包括软件,也包括硬件。因此,用户界面设计在软件产品的生命期中处于较早的阶段;同时,系统核心功能的设计和实现也可能会影响用户界面的使用,效果有好有坏。

人机交互既要考虑人也要考虑计算机系统。因此,作为一个知识领域,人机交互学科需要综合考虑文化、社会、组织、认知和感知等多方面的问题。所以说,人机交互是多学科交叉的学科,涉及心理学、人体工程学、计算机科学、图形和产品设计、人类学和工程学等。

HCI. 人机交互(4 个核心一级学时,4 个核心二级学时)

	核心一级学时	核心二级学时	包含选修课程
HCI/基础	4		否
HCI/交互设计		4	否
HCI/交互系统编程			是
HCI/以用户为中心的设计和测试			是
HCI/新型交互技术			是
HCI/协同和通信			是
HCI/人机交互中的统计学方法			是
HCI/人因和安全			是
HCI/面向设计的人机交互			是
HCI/混合、增强和虚拟现实			是

HCI/基础

[4 个核心一级学时]

动机:对终端用户而言,界面就是系统。因此,人机交互的设计必须围绕交互开展,遵循以人为中心的设计原则。相比于本大纲中的其他课程,学生需要掌握一套完全不同的技术体系来解决这个问题。

知识点:

- 人机交互的情景(所有和用户界面相关的事情,例如:网页、商业应用、移动应用和游戏等)。
- 以用户为中心的开发流程,例如:较早的关注用户、经验测试、迭代设计等。
- 不同的评测要素,例如:实用性、效率、学习性、用户满意度等。
- 可用性启发原则和可用性评测原理。
- 与交互设计有关的物理能力,例如:颜色感知、人体工程学。
- 与交互设计有关的认知模型,例如:注意力、感知和认知、运动和记忆;期望和执行的差异。
- 与交互设计有关的社会学模型,例如:文化、通信、网络和组织。
- 好设计和好设计师的原则;工程方案的权衡。
- 可访问性,比如,对不同残疾类型人群的界面(例如:盲人、运动能力受损的人)。
- 适合不同年龄人群的界面(例如:儿童、年龄大于 80 岁的人群)。

学习成果:

1. 讨论为什么以用户为中心的软件开发是重要的。[熟悉]
2. 总结心理和社会交互的基本规则。[熟悉]
3. 开发使用以软件来分析人交互行为的概念词汇:功能可见性、概念模型、反馈等。[运用]
4. 定义一个以用户为中心的设计过程,该过程需要明确考虑到用户、开发者以及他们熟悉的人具有明显的不同。[运用]
5. 针对一个存在的应用程序,创建一个简单的可用性测试。[评估]

HCI/交互设计

[4 个核心二级学时]

动机:计算机专业学生需要一套必不可少的标准方法和工具来构建界面。

知识点:

- 用户图形界面相关原则。
- 视觉设计元素(布局、颜色、字体、标签等)。
- 任务分析,包括生成任务分析模型的定性方面。
- 低保真度(纸面)建模。
- 量化评估技术,比如击键评估。
- 用户帮助及文档。

- 处理用户出错/系统失效。
- 用户界面标准。

学习成果:

1. 对于已标识的用户组,进行并记录需求分析。[评估]
2. 创建简单的应用程序,包括采用帮助和文档来支持用户图形界面。[运用]
3. 执行量化评估,讨论/报告其结果。[运用]
4. 讨论至少一项国内或国际用户界面设计标准。[熟悉]

HCI/交互系统编程

[选修]

动机:以用户体验为中心进行软件开发,学习其中涉及的方法和技术。

知识点:

- 软件架构模式。例如,模型 - 视图 - 控制器(MVC)模式;命令对象,在线,离线(参见 PL/事件驱动及响应编程,MVC 用于事件驱动型编程环境)。
- 交互设计模式:视觉架构、导航距离。
- 事件管理及用户交互。
- 几何管理(参见 GV/几何建模)。
- 选择交互风格及交互技术。
- 呈现信息:导航、表示、操作。
- 界面动画技术(比如场景图)。
- 窗口小部件类和库。
- 现代用户图形界面库(如 iOS、安卓、JavaFX)图形界面创建工具及界面编程环境(参见 PBD/移动平台)。
- 说明性界面规范:Stylesheet 及 DOM。
- 数据驱动型应用(以数据库为后端的网页)。
- 跨平台设计。
- 适于资源受限设备的设计(比如小型移动设备)。

学习成果:

1. 解释 MVC 模式对于界面设计的重要性。[熟悉]
2. 创建采用现代图形用户界面的应用程序。[运用]
3. 辨别不同平台用户界面的异同点。[熟悉]
4. 解释并使用用户图形界面编程的概念:事件处理、基于受限资源的布局管理等。[熟悉]

HCI/以用户为中心的设计和测试

[选修]

动机:保证在设计流程的所有阶段,从开始设想到最终实现,终端用户的需求都被全面考虑在内。

知识点:

- 设计流程的方法和典型特征。
- 功能性和使用性需求(可参考需求工程学)。
- 收集需求的方法(例如:采访、调查、人类学研究、上下文问询)。
- 分析和展示需求的方法和工具(例如:报告、角色模型)。
- 原型设计的方法和工具(例如:草图、故事版、低保真原型构建、线框图)。
- 无需用户参与的测评方法,包括定性和定量两方面(例如:走查、GOMS、专家分析、启发式方法、指导性和标准测评方法)。
- 用户参与的测评方法(例如:观察用户行为、表达思维报告、采访、调查、实验)
- 对测评有效性的进一步测试(例如:取样、普适性验证)。
- 报告测评结果。
- 国际化,考虑跨文化的用户设计方案。

学习成果:

1. 解释以用户为中心的设计如何补充了其他软件过程模型。[熟悉]
2. 使用低保真原型构建方法来收集和报告用户响应。[运用]
3. 对特定的用户界面设计选择适当的方法。[评估]
4. 使用多种技术来评估一个指定的用户界面。[评估]
5. 对比不同评估方法的优缺点。[评估]

HCI/新型交互技术

[选修]

动机:随着技术发展,新的交互手段成为可能。需要建立一种可扩展的知识体系来追踪这些新兴交互技术。

知识点:

- 选择交互风格和交互技术。
- 将信息呈现给用户:导航、表达、操控。
- 设计、实现并且评估非鼠标交互的方法。

➢ 触摸和多触点的交互界面。

➢ 共享、虚拟人或者大型交互界面。

- 新型输入模态(例如:传感器和位置信息)。
- 新型窗口界面(例如:iPhone、Android)。
- 语音识别和自然语言处理(参见 IS/自然语言处理技术)。
- 可穿戴界面和有形界面。
- 诱导交互和情感交互。
- 普适和情景感知交互技术(普适计算)。
- 贝叶斯推论(例如:输入文本和指点预测)。
- 外围显示和交互。

学习成果:

1. 描述非鼠标界面的适用场景。[熟悉]
2. 理解在鼠标指点界面之外的交互手段。[熟悉]
3. 讨论非鼠标交互的优缺点。[评估]

HCI/协同和通信

[选修]

动机:计算机界面不仅帮助用户实现他们的个人目标,同时也帮助他们与其他人交互,不论这种交互是任务相关的(如工作或游戏)还是任务无关的(如社交网络)。

知识点:

- 异步的组通信,如电子邮件、论坛、社交网络。
- 同步的组通信,如聊天室、会议、网络游戏。
- 社交媒体、社交计算和社交网络分析。
- 在线协作、智能空间、工作流技术的社会协同层面。
- 在线社区。
- 软件角色和智能代理、虚拟世界和替身(交叉参照 IS/代理)。
- 社会心理学。

学习成果:

1. 描述同步通信和异步通信的区别。[熟悉]
2. 将人机交互领域中的个人交互问题与组交互问题相比较。[评估]
3. 讨论若干由协作软件引发的社会问题。[熟悉]
4. 讨论体现人类意图的软件中的人机交互问题。[熟悉]

HCI/人机交互中的统计学方法

[选修]

动机:很多人机交互研究依赖于对数据恰当的使用、理解和应用。这些知识

对于具有心理学背景的学生往往不成问题,但计算机背景的学生往往缺少这方面的基础。

知识点:

- t 测试(t – test)。
- 方差分析(ANOVA)。
- 随机(非参数)测试,组内设计 vs 组间设计。
- 效应量计算。
- 探索性数据分析。
- 统计数据的展示。
- 定性结果和定量结果的结合。

学习成果:

1. 解释基本的统计概念和他们应用的场景。[熟悉]
2. 提取和说明论文中定量汇报结果时使用的统计参数。[运用]
3. 设计一个可以产出定量结果的用户研究。[运用]
4. 进行一项研究并汇报结果,同时使用定性和定量评估。[运用]

HCI/人因和安全

[选修]

动机:有效的界面设计需要具备安全心理学的基本知识。很多攻击并不具有技术基础,而是利用人们的癖好和弱点。“只有业余的人才攻击机器,专家瞄准的是人”(Bruce Schneier,

https://www. schneier. com/blog/archives/2013/03/phishing_has_go. h.)。

知识点:

- 应用心理学和安全政策。
- 安全经济学。
- 监管环境 – 责任、可靠性和自我决定。
- 组织弱点和威胁。
- 可用性设计和安全。
- 借口、扮演和欺骗。例如,网络钓鱼和鱼叉式网络钓鱼(交叉参照 IAS/威胁与攻击)。
- 信任、隐私和欺骗。
- 生物特征识别(相机、声音)。
- 身份管理。

学习成果:

1. 解释网络钓鱼和鱼叉式网络钓鱼的概念,以及如何区分它们。[熟悉]

2. 描述界面设计中的信任问题,并分别列举一个高信任和低信任系统。[评估]

3. 为一家安全机构设计界面。[评估]

4. 解释身份管理的概念和其重要性。[熟悉]

5. 分析一项安全政策和/或安全措施,以展示他们如何考虑或如何没有考虑到人因影响。[运用]

HCI/面向设计的人机交互

[选修]

动机:人机交互的起源和应用于特定历史、学科和文化环境,一些课程希望强调对人机交互工作本身的范式和价值的理解。

知识点:

- 对技术及其界面的智能风格和观点。
- 人机交互作为一门设计学科的考虑。

➢ 草图。

➢ 参与式设计。

- 批判反思式的人机交互。

➢ 批判性技术实践。

➢ 为政治活动服务的技术。

➢ 用户体验的哲学。

➢ 人种学和民族方法学。

- 应用的象征域。

➢ 可持续性。

➢ 表达艺术的技术。

学习成果:

1. 解释“人机交互是一门面向设计的学科”这句话的含义。[熟悉]

2. 详述符合特定设计目标的设计过程。[熟悉]

3. 将一系列设计方法运用到给定问题中。[运用]

HCI/混合、增强和虚拟现实

[选修]

动机:针对创建和开发沉浸式环境(尤其是游戏)所需接口组成部分的详尽考虑。

知识点:

- 输出。

➢ 声音。

➢ 立体显示。

➢ 力反馈模拟、触摸设备。

- 用户输入。

➢ 用户视点、物体追踪。

➢ 姿态和手势识别。

➢ 加速计。

➢ 基准标志。

➢ 用户接口。

- 物理建模和渲染。

➢ 物理模拟:冲突检测和响应,动画。

➢ 可见性计算。

➢ 实时渲染,多细节层次(LOD)。

- 系统结构。

➢ 游戏引擎。

➢ 移动增强现实。

➢ 飞行模拟器。

➢ CAVEs 虚拟系统。

➢ 医学成像。

- 网络。

➢ 点对点技术、客户端 – 服务器、航位推测法(dead reckoning)、数据加密、同步系统。

➢ 分布式协作。

学习成果:

1. 描述计算机制图系统实现的用于合成立体视图的光学模型。[熟悉]

2. 描述不同的用户视点追踪技术的原理。[熟悉]

3. 描述基于几何与基于图像的虚拟现实之间的区别。[熟悉]

4. 描述在网络环境中用户操作同步和数据一致性的问题。[熟悉]

5. 对于面向一个特定应用的虚拟现实系统,确定其在接口、硬件和软件配置上的基本需求。[运用]

6. 描述游戏引擎的几种可能的用途,包括它们的潜能和限制。[熟悉]

信息保障与安全(Information Assurance and Security IAS)

信息技术已经成为当今世界的重要支撑,在计算机科学教育中也起着关键的作用。认识到这点后,CS2013 将信息安全保障作为知识领域(KA)增加到知识体系中。信息安全保障的范围涵盖了为确保信息和信息系统的机密性、完整性及可用性而在技术和政策上所进行的保护、防卫的控制和处理过程,以及为此提供的证明方法和不可抵赖性手段。保障这一概念本身就包含了对当前以及过去的处理过程以及数据合法性的认证。从全面的观点看,保障和安全两方面对信息及信息系统缺一不可。因此,保护信息系统安全、确保信息系统过去和现在的运行状态及数据正确无误的人才成为社会的急需。从这个角度说,信息保障和安全教育的内容就是为培养这类人才所应该具备的知识、技能和能力而所做的一切努力。在计算机科学这一学科中,安全的概念以及相关知识内容的重要性已经成为计算机学科的核心要求,可以和前些年计算机性能概念的重要性相媲美。

所有的知识领域中,信息保障与安全有独特的特点,它的知识点广泛渗透到其他知识领域中。因此,本章只列出与信息保障与安全(IAS)直接相关的知识点,而其他相关知识点仅在本知识领域中进行说明和交叉引用。在 IAS 知识领域中,多数知识点都在 9 个课时的核心一级和核心二级中给出。这样的课时安排,是让学生在 IAS 领域中初步熟悉这些知识点,而更深入的掌握则放在了应用这些知识点的相关知识领域中。散布在所有其他知识领域的对 IAS 知识点的广泛应用(共 63.5 课时)足以让计算机科学专业学生对它们有深入地了解和掌握。IAS 知识领域主要包括两个部分:(1) 信息保障与安全 IAS 特有的一些概念;(2) 被整合到更能反映其本质的其他知识领域的 IAS 知识点,以及对信息安全有着重要作用的知识点。为了保持知识领域的完整性,IAS 全部课时的分布如下表所示。

IAS. 信息保障与安全"核心"及分布于其他领域的课时

	核心一级学时	核心二级学时	包含选修
信息保障与安全 IAS	3	6	是
IAS 在其他领域	32	31.5	是

IAS. 信息保障与安全(3 个核心一级学时,6 个核心二级学时)

	核心一级学时	核心二级学时	选修
IAS/安全基本概念	1		否
IAS/安全性设计准则	1	1	否
IAS/防错性程序设计	1	1	是
IAS/威胁与攻击		1	否
IAS/网络安全		2	是
IAS/密码学		1	否
IAS/Web 安全			是
IAS/平台安全			是
IAS/安全策略和管理			是
IAS/数字取证			是
IAS/安全软件工程			是

CS2013 的许多其他知识领域中,安全问题或者被作为知识单元的基础(如 OS/安全和防护、SE/软件构建等),或者被作为知识点的应用案例(如 HCI/基础、NC/路由和转发、SP/知识产权)而成为其讨论内容。下表给出了这些知识领域/知识单元中与 IAS 明确相关的课时数。

IAS. 信息保障与安全(分布在其他知识领域)(32 个核心一级学时,31.5 个核心二级学时)

知识领域及知识点	核心一级学时	核心二级学时	选修
AR/汇编级计算机组成原理		1	
AR/存储系统的组织与结构		0.5	
AR/多处理器和可选体系结构			是
HCI/基础	1		
HCI/人因和安全			是
IM/信息管理概念	0.5	0.5	
IM/事务处理			是
IM/分布式数据库			是
IS/不确定性推理方法			是

续表

知识领域及知识点	核心一级学时	核心二级学时	选修
NC/引言	1		
NC/网络应用程序	0.5		
NC/可靠数据传输		1.5	
NC/路由和转发		1	
NC/局域网		1	
NC/资源分配		0.5	
NC/移动性		1	
OS/操作系统概述	2		
OS/操作系统原理	1		
OS/并发		1.5	
OS/调度和分发		2	
OS/内存管理		2	
OS/安全和防护		2	
OS/虚拟机			是
OS/设备管理			是
OS/文件系统			是
OS/实时与嵌入式系统			是
OS/容错性			是
OS/系统性能评估			是
PBD/Web 平台			是
PBD/移动平台			是
PBD/工业平台			是
PD/并行基础	1		
PD/并行分解	0.5		
PD/通信和协调	1	1	是
PD/并行结构	0.5		是
PD/分布式系统			是
PD/云计算			是

续表

知识领域及知识点	核心一级学时	核心二级学时	选修
PL/面向对象程序设计	1	3	
PL/函数式程序设计	1		
PL/基本类型系统	0.5	2	
PL/语言翻译与执行		1	
PL/运行时系统			是
PL/静态分析			是
PL/并发和并行			是
PL/类型系统			是
SDF/程序设计概念基础	1		
SDF/开发方法	8		
SE/软件过程	1		
SE/软件项目管理		1	是
SE/工具和环境		1	
SE/软件构建		2	是
SE/软件验证和确认		1	是
SE/软件演化		1.5	
SE/软件可靠性	1		
SF/跨层通信	3		
SF/并行性	1		
SF/资源分配与调度技术	0.5		
SF/虚拟化与隔离		1	
SF/冗余下的可靠性		2	
SP/社会环境	0.5		
SP/分析工具	1		
SP/职业道德	1	0.5	
SP/知识产权	2		是
SP/隐私和公民自由	0.5		
SP/安全策略、法律和计算机犯罪			是

IAS/安全基本概念

[1 个核心一级学时]

知识点:

- CIA(机密性、完整性、可用性)。
- 风险、威胁、漏洞及攻击向量的概念(参见 SE/软件项目管理/风险)。
- 认证和授权、访问控制(强制性 vs 自主性)。
- 信任和可信度的概念。
- 伦理(责任披露)(参见 SP/职业道德责任,责任和赔偿责任)。

学习成果:

1. 关键安全属性的权衡及分析(机密性,完整性,可用性)。[运用]

2. 描述风险、威胁、漏洞和攻击向量的概念(包括没有绝对安全的事实)。[熟悉]

3. 解释认证、授权、访问控制的概念。[熟悉]

4. 解释信任和可信度的概念。[熟悉]

5. 描述伦理问题在计算机安全方面的重要性,比如是否修补漏洞和是否披露漏洞等道德问题。[熟悉]

IAS/安全性设计准则

[1 个核心一级学时、1 个核心二级学时]

知识点:

[核心一级]

- 最小权限及隔离(参见 OS/安全性和防护/政策/分割机制、SF/虚拟化与隔离/保护的理论基础/可预测性能、PL /语言翻译与执行/内存管理)。
- 故障自动防护(参见 SE/软件构建/编码实践:技术、术语/模式、开发高质量程序的机制和 SDF /开发方法/编程正确性)。
- 开放性设计(参见 SE/软件演化/大背景下的软件开发,已有的代码库)。
- 端到端的安全(参见 SF/冗余下的可靠性/长距离通信时错误如何增加;端到端的原则)。
- 深度防御(例如,防御性编程,分层防御)。
- 安全的设计(参见 SE/软件设计/系统的设计原则)。
- 安全和其他设计目标之间的冲突关系。

[核心二级]

- 完整的调解。
- 使用审查过的安全组件。

• 经济机制(减少可信计算基,最小化攻击面)(参见 SE/软件设计/系统设计原则和 SE/软件构建/开发环境:“绿场”对比现有代码库)。

• 可用的安全(参见 HCI /基础/通知交互设计的认知模型)。

• 安全可组合性。

• 预防、检测和威慑(参见 SF/冗余下的可靠性/缺陷与故障之间的区别及 NC/可靠数据传输/错误控制和 NC/可靠数据传输/流控制。

学习成果

[核心一级]

1. 描述如何将最小特权和隔离应用于系统设计的原理。[熟悉]

2. 总结故障自动防护原则和默认拒绝的原则。[熟悉]

3. 讨论依赖于开放式设计中的安全保密性的影响。[熟悉]

4. 解释端到端数据安全的目标。[熟悉]

5. 讨论多层次防御系统的好处。[熟悉]

6. 对生命周期中各个阶段的产品,描述安全方面有哪些因素需要评估。[熟悉]

7. 描述在产品设计中加入安全带来的成本和折衷因素。[熟悉]

[核心二级]

8. 描述调解的概念和完整的调解原则。[熟悉]

9. 描述安全操作标准组件,并解释重用他们而不是重新设计基本操作的好处。[熟悉]

10. 解释可信计算的概念,包括可信计算基和攻击面,以及最小化可信计算基的原理。[熟悉]

11. 讨论安全机制设计对可用性的重要性。[熟悉]

12. 描述出现在多个组件之间的边界安全问题。[熟悉]

13. 辨识预防机理和检测/预防机制的不同作用。[熟悉]

IAS/防错性程序设计

[1 个核心一级学时、1 个核心二级学时]

防错性程序设计问题一般不在隔离中考虑,但其可应用于其他知识点,特别是在 SDF,SE 和 PD 的知识领域。

知识点:

[核心一级]

• 输入验证和数据清洁(参见 SDF/开发方法/程序的正确性)。

• 选择编程语言和类型安全的语言。

• 输入验证和数据清洁发生错误的例子(参见 SDF/开发方法/程序的正确性和 SE/软件构建/编码实践)。

➢ 缓冲区溢出。

➢ 整数的错误。

➢ SQL 注入。

➢ XSS 漏洞。

• 竞争条件下(参见 SF/并行性/并行程序设计、PD/并行结构/共享和分布式内存、PD/通信和协调/共享内存和 PD/并行基础/串行程序设计中未发现的编程错误)。

• 异常和意想不到的行为的正确处理(参见 SDF/开发方法/程序正确性)。

[核心二级]

• 第三方组件的正确使用(参见 SDF/开发方法/程序正确性和操作系统的原则/应用程序接口(API)的概念)。

• 有效的部署安全更新(参见 OS/安全和防护/安全保护方法及设备)

[选修]

• 信息流控制。

• 以安全为目的正确产生随机性。

• 用于检测和减轻输入以及数据清洁错误的机制。

• 模糊测试。

• 静态分析和动态分析。

• 程序验证。

• 操作系统的支持(例如,地址空间随机化,堆栈 canaries 探测技术)。

• 硬件支持(例如,数据执行保护 DEP,可信赖平台模块 TPM)。

学习成果

[核心一级]

1. 解释为什么输入验证和数据清洁在输入通道的对抗性控制上是必要的。[熟悉]

2. 解释为什么你会选择一种类型安全的语言来开发程序,如 Java,而不是采用不安全的编程语言如 C /C + + 。[熟悉]

3. 常见的输入验证错误的分类,并写出正确的输入验证码。[运用]

4. 演示如何使用一个高层次的编程语言进行防止竞争条件的发生和如何处理异常。[运用]

5. 演示如何识别并妥善地处理错误情况。[运用]

[核心二级]

6. 解释滥用接口与第三方代码的风险以及如何正确使用第三方代码。[熟悉]

7. 讨论更新软件以修复安全漏洞的必要性以及软件修复的生命周期管理。[熟悉]

[选修]

8. 列出直接和间接的信息流动的例子。[熟悉]

9. 解释除加密学(如密码生成、随机算法)外随机数在安全中的作用。[熟悉]

10. 解释用于检测和减少不同类型数据清洁错误的机制。[熟悉]

11. 演示程序是如何测试并检测出输入的错误。[运用]

12. 使用静态和动态的工具来识别程序故障。[运用]

13. 描述存储结构是如何保护避免运行时的攻击。[熟悉]

IAS/ 威胁与攻击

[1 个核心二级学时]

知识点:

[核心二级]

- 攻击者目标、能力和动机(如地下经济、数字间谍、网络战、内部威胁、黑客犯罪和高级持续威胁)。
- 恶意软件实例(例如,病毒,蠕虫,间谍软件,木马和僵尸网络)。
- 拒绝服务(DOS)和分布式拒绝服务(DDoS)。
- 社会工程(例如,网络钓鱼)(参见 SP/社会环境/在网络世界中计算对和社会影响和 HCI/交互设计/处理人员/系统故障)。

[选修]

- 针对隐私和匿名的攻击(参见 HCI/基础/通知交互设计的社会模型),如文化、通信、网络和组织(参见 SP/隐私和公民自由/隐私保护的技术解决方案)。
- 恶意软件/不必要的沟通,如隐蔽通道和信息隐藏。

学习成果:

[核心二级]

1. 描述针对特定系统攻击者的大致类型。[熟悉]

2. 讨论恶意软件策略(如基于签名的检测、行为检测)的局限性。[熟悉]

3. 区分社会工程攻击和拒绝服务攻击的例子。[熟悉]

4. 讨论如何认定拒绝服务攻击以及减轻其危害的措施。[熟悉]

[选修]

5. 描述常见应用中隐私和匿名的风险性。[熟悉]

6. 讨论隐蔽通道和其他数据泄露过程的概念。[熟悉]

IAS/网络安全

[2 个核心二级学时]

对网络安全的讨论依赖于事先对网络基本概念的理解,包括协议(如 TCP/IP)和网络结构(交叉引用 NC/网络通信)等。

知识点:

[核心二级]

- 网络面临的特定威胁和攻击类型(例如,拒绝服务攻击,欺骗,探测和流量重定向,中间人,消息完整性攻击,路由攻击和流量分析)。
- 用于数据和网络安全的密码。
- 安全网络架构(例如,安全信道,安全路由协议,安全 DNS,VPN,匿名通信协议,隔离)。
- 防御机制与对策(例如,网络监控,入侵检测,防火墙,欺骗和 DOS 保护,蜜罐技术,回溯)。

[选修]

- 无线和蜂窝网络安全(交叉参照 NC/移动/蜂窝网络协议、NC/移动/802.11)。
- 其他非有线网络(例如,Ad hoc 网络,传感器网络和车载网络)。
- 审查抵抗。
- 可行的网络安全管理(例如:配置网络访问控制)。

学习成果:

[核心二级]

1. 描述网络威胁和攻击的不同类别。[熟悉]

2. 描述公有和私有密钥的架构,以及公有密钥基础设施(PKI)如何支持网络安全。[熟悉]

3. 描述网络协议栈每一层的安全技术的优点和局限性。[熟悉]

4. 针对给定的网络威胁,确定适当的防御机制,并分析其局限性。[熟悉]

[选修]

5. 讨论其他非有线网络的安全属性和局限性。[熟悉]

6. 了解非有线网络面临的其他威胁。[熟悉]

7. 描述使用安全通信通道可以/或者不可以防护的威胁。[熟悉]

8. 综述抗网络审查的技术。[熟悉]

9. 画图来描述网络安全。[熟悉]

IAS/密码学

[1 个核心二级学时]

知识点:

[核心二级]

• 覆盖与不同(通信)双方相关概念的基本的密码学术语,安全/不安全的通道,攻击者和他们的能力,加密,解密,密钥和特点,签名。

• 密码的类型(如凯撒密码、仿射密码)与典型的攻击方法(如频谱分析)。

• 支持数字签名和加密的 PKI(公钥基础设施)及其面临的挑战。

[选修]

• 密码学的基本数学预备知识,包括线性代数、数论、概率论、统计学等。

• 密码学原语。

➢ 伪随机发生器和流密码。

➢ 分组密码(伪随机置换),如 AES。

➢ 伪随机函数。

➢ 哈希函数,如 SHA2、抗碰撞。

➢ 消息认证码。

➢ 密钥推导函数。

• 对称密钥加密。

➢ 完全保密性和一次性密码本。

➢ 操作模式的语义安全性和认证加密(例如,encrypt - then - MAC,OCB,GCM)。

➢ 消息完整性(例如,CMAC,HMAC)。

• 公有密钥。

➢ 陷门置换,例如,RSA。

➢ 公共密钥加密,例如,RSA 加密、EI Gamal 加密。

➢ 数字签名。

➢ 公有密钥基础设施和证书。

➢ 难度设定,如 Diffie - Hellman、整数分解。

• 认证密钥交换协议,例如,TLS。

• 密码协议:尝试 - 响应认证,零知识协议,承诺,未察觉的传输,安全两方或多方计算,秘密共享以及应用。

- 现实世界中概念的应用,例如:电子现金,客户和服务器之间安全的通道,安全的电子邮件,实体认证,设备配对,投票系统。
- 密码学中安全的定义和攻击:

➢ 目标:分辨性,不可伪造性,持续碰撞。

➢ 攻击者的能力:选择消息攻击(签名),生日攻击,侧通道攻击,故障注入攻击。

- 加密标准和参考实现。
- 量子密码。

学习成果:

[核心二级]

1. 描述密码学的目的和列出其在数据传输的应用方式。[熟悉]

2. 定义下列术语:密码,密码分析,加密算法,密码学;描述两种将纯文本转换成密文的基本方法。[熟悉]

3. 讨论密码学中素数的重要性并说明其在密码算法中的应用。[熟悉]

4. 解释公有密钥基础设施如何支持数字签名和加密,讨论其存在的不足/局限性。[熟悉]

[选修]

5. 运用密码学原语,描述它们的基本性质。[运用]

6. 说明如何测度熵以及如何形成加密的随机性。[运用]

7. 运用公共密钥原语及开发相关应用。[运用]

8. 解释密钥交换协议工作原理及失效方式。[熟悉]

9. 讨论密码协议及其属性。[熟悉]

10. 描述密码学原语和协议的现实应用。[熟悉]

11. 总结在密码学原语中与攻击相关的安全定义,包括攻击者的能力与目标。[熟悉]

12. 在一个场景下适当地应用已知的密码学技术。[运用]

13. 评估自行设计加密方法的危险性。[熟悉]

14. 描述量子密码学和量子计算对加密算法的影响。[熟悉]

IAS/Web 安全

[选修]

知识点:

- Web 安全模型。

➢ 浏览器安全模型,包括同源策略等。

➢ 客户机/服务器信任边界,例如,不能依赖客户端能安全执行。

• 会话管理,认证。

➢ 单点登入。

➢ HTTPS 和证书。

• 应用程序的漏洞和防御。

➢ SQL 注入。

➢ XSS。

➢ CSRF。

• 客户端安全。

➢ Cookie 安全政策。

➢ HTTP 安全拓展,例如,HSTS。

➢ 插件,扩展和 Web 应用程序。

➢ Web 用户跟踪。

• 服务器端的安全工具,如 Web 应用防火墙(WAFS)和模糊器。

学习成果:

1. 描述包含同源政策的浏览器安全模型以及网络安全中的威胁模型。[熟悉]

2. 讨论 Web 会话的概念,安全的通信信道(如 TLS)和安全认证证书的重要性,单点登录认证,如 OAuth 和 SAML。[熟悉]

3. 描述常见 Web 应用程序中的漏洞和攻击类型以及相应的防御方式。[熟悉]

4. 在应用程序开发中运用客户端的安全功能。[运用]

IAS/平台安全

[选修]

知识点:

• 编码完整性和编码签名。

• 安全启动,测试引导,以及信任根目录。

• 鉴定。

• TPM 和安全协处理器。

• 外围设备的安全威胁,例如:DMA,IOMMU。

• 物理攻击:硬件木马,内存探针,冷启动攻击。

• 嵌入式设备安全,例如:医疗设备、汽车。

• 信任的路径。

学习成果

1. 解释编码完整性与编码签名的概念以及它的应用范围。[熟悉]
2. 讨论信任根目录的概念以及安全启动和安全引导的过程。[熟悉]
3. 描述系统完整性的远程证明机制。[熟悉]
4. 总结 TPM 的目的及主要原语。[熟悉]
5. 确定把配件插入设备造成的威胁。[熟悉]
6. 辨识物理攻击及其对策。[熟悉]
7. 辨识对非 PC 硬件平台的攻击。[熟悉]
8. 讨论可信路径的概念及其重要性。[熟悉]

IAS/安全策略和管理

[选修]

参见一般交叉引用的 SP /安全政策,法律和计算机犯罪。

知识点:

- 隐私策略(交叉参照 SP/社会环境/网络世界中计算对社会的影响;参见 SP/职业道德/责任和义务、SP/隐私和公民自由 /隐私权保护的法律基础)。
- 推理控制/统计披露限制。
- 备份策略,密码更新策略。
- 违反信息披露策略。
- 数据收集和保留策略。
- 供应链策略。
- 云安全的权衡。

学习成果:

1. 描述隐私的相关概念,包括个人私有信息,由安全机制引起的侵犯隐私行为,描述隐私保护策略如何与安全机制发生矛盾。[熟悉]
2. 描述攻击者如何通过与数据库交互来推断秘密。[熟悉]
3. 解释如何设定数据备份与密码更新的策略。[熟悉]
4. 讨论如何设置违反信息披露策略。[熟悉]
5. 描述数据保留策略的后果。[熟悉]
6. 识别依靠外包制造的风险。[熟悉]
7. 识别外包到云的风险及益处。[熟悉]

IAS/数字取证

[选修]

知识点:

- 数字取证的基本原理和方法学。
- 设计需要取证的系统。
- 证据规则——一般概念以及司法与监督链之间差异。
- 搜查和扣押证据:法律和程序的要求。
- 数字证据的方法和标准。
- 保存数据的标准和技术。
- 法律和报告的相关问题,包括作为专家证人工作。
- OS /文件系统取证。
- 申请取证。
- Web 取证。
- 网络取证。
- 移动设备取证。
- 计算机/网络/系统的攻击。
- 攻击检测和调查。
- 反取证。

学习成果:

1. 描述什么是数字取证,数字证据的来源,数字取证的局限性。[熟悉]
2. 解释如何设计软件以支持取证。[熟悉]
3. 描述使用截获的数据的法律要求。[熟悉]
4. 描述从确认数据处理开始的证据获取过程。[熟悉]
5. 描述如何完成数据收集,并正确地存储原始数据和取证的副本。[熟悉]
6. 在硬盘上采集数据。[运用]
7. 当作为证据检验员作证时的责任与义务。[熟悉]
8. 基于给定的搜索项从影像系统中恢复数据。[运用]
9. 从应用轨迹中重建应用的历史。[运用]
10. 通过 web 痕迹重建 web 的浏览历史。[运用]
11. 捕获和解读网络流量数据。[运用]
12. 讨论与移动设备取证相关的挑战。[熟悉]
13. 检查系统(网络,计算机,或应用程序)的恶意软件或恶意活动[熟悉]
14. 应用取证工具检测安全漏洞。[运用]
15. 辨识反取证方法。[熟悉]

IAS/安全软件工程

[选修]

在软件开发基础(SDF)和软件工程(SE)知识领域中已经涉及的安全编程实践基础,例如,软件工程/软件构造,软件验证与确认。

知识点:

- 将安全加入到软件设计生命周期中(参见 SE/软件过程)。
- 安全设计原则及模式。
- 安全的软件规范和需求。
- 安全的软件开发实践(参见 SE/软件构建)。
- 安全测试——测试安全需求是否满足的过程(包括静态和动态分析)。
- 软件质量保证和以标准测试集来测量。

学习成果:

1. 描述将安全集成到软件开发生命周期的要求。[熟悉]
2. 将保护机制的设计准则、软件安全准则[2]以及安全设计准则[1]应用到软件开发计划中。[运用]
3. 开发一个软件开发计划书,完整描述功能需求,并且确认预期的执行路径。[运用]
4. 描述在程序代码中最大限度地减少漏洞的最佳软件开发实践。[熟悉]
5. 对软件的应用进行静态或者动态的安全验证与评估。[运用]

参考文献:

[1] Gasser, M. Building a Secure Computer System, Van Nostrand Reinhold, 1988.

[2] Viega, J. and McGraw, G. Building Secure Software: How to Avoid Security Problems the Right Way, Addison - Wesley, 2002.

信息管理(Information Management IM)

信息管理主要关注于信息的捕捉、数字化、表示、组织、转化和展示,以及高效的访问和更新存储信息的算法,数据建模和抽象,文件存储技术。学生应能够建立数据的概念模型和物理模型,针对特定问题确定合适的信息管理方法和技术,并能够在选择和实现合适的信息管理解决方案,解决相关的设计问题,包括可扩展性、可访问性和可用性等。

注明:本章信息管理(IM)部分内容参见上章信息保障与安全(IAS)中安全基本概念。

IM. 信息管理(1 个核心一级课时;9 个核心二级课时)

	核心一级课时	核心二级课时	包括选修课
IM/信息管理概念	1	2	否
IM/数据库系统		3	是
IM/数据模型		4	否
IM/索引			是
IM/关系数据库			是
IM/查询语言			是
IM/事务处理			是
IM/分布式数据库			是
IM/物理数据库设计			是
IM/数据挖掘			是
IM/信息存储与检索			是
IM/多媒体系统			是

IM. 信息管理相关知识点(分布在其他知识领域)(1 个核心一级课时; 2 个核心二级课时)

	核心一级 /课时	核心二级 /课时	包括选修课
IAS/安全基本概念	1	2	否

注:参见信息保障与安全(IAS)知识领域对这个主题领域的描述。

IM/信息管理概念

[1 个核心一级课时;2 个核心二级课时]

知识点:

[核心一级]

- 信息系统作为社会技术系统。
- 基本信息存储和检索(IS&R)的概念。
- 信息获取和表示。
- 支撑人的需求:搜、检索、链接、浏览、导航。

[核心二级]

- 信息管理应用。
- 声明和导航式查询,使用链接。
- 分析和索引。
- 质量问题:可靠性,可扩展性,效率和有效性。

学习成果:

[核心一级]

1. 描述人们如何获得信息和数据来支持他们的需求。[熟悉]
2. 描述中央式数据组织控制的优点和缺点。[评估]
3. 辨识与信息管理有关的职位/角色(如数据库管理员、数据建模人员、应用程序开发人员及最终用户)。[熟悉]
4. 比较和对比数据和知识的信息。[评估]
5. 展示如何使用显式存储的与数据关联的元数据/数据模式。[运用]
6. 确定一个组织的数据持久性问题。[熟悉]

[核心二级]

7. 从是否满足用户信息需求角度评价一个信息应用。[评估]
8. 解释声明式查询的使用。[熟悉]
9. 给出一个导航式查询的声明式版本。[熟悉]
10. 描述解决信息隐私、信息完整、信息安全和信息保存问题的几种技术方案。[熟悉]
11. 解释提高效率(吞吐量,响应时间)和有效性(召回,精度)的措施。[熟悉]
12. 描述提升信息系统规模的方法。[熟悉]
13. 找出常见的信息系统存在的漏洞和故障情况。[运用]

IM/数据库系统

[3 个核心二级课时]

知识点:

[核心二级]

- 数据库系统的演变和方向。
- 数据库系统的组成。
- 数据库管理系统的核心功能的设计(例如,查询机制,事务管理,缓冲区管理,访问方法)。
- 数据库的结构和数据的独立性。
- 声明式查询语言的使用。
- 支持结构化与/或流内容的系统。

[选修]

- 用于管理大数据的方法(如 NoSQL 数据库系统,使用 MapReduce 技术)。

学习成果:

[核心二级]

1. 解释使用数据库和直接用数据文件编程两者之间的本质区别。[熟悉]
2. 描述包括查询优化器、查询执行器、存储管理、访问方法和事务处理器等核心数据库系统组件的最常见的设计。[熟悉]
3. 指出数据库系统的基本目标、功能及模型。[熟悉]
4. 描述一个数据库系统的组成,并给出数据库应用实例。[熟悉]
5. 确定 DBMS 的主要功能,描述其在数据库系统中的作用。[熟悉]
6. 解释数据独立性概念及其在数据库系统中的重要性。[熟悉]
7. 使用声明式查询语言,从数据库提取信息。[运用]
8. 描述数据库提供的支持结构化和/或流(序列)数据(例如文本)的功能。[熟悉]

[选修]

9. 描述保存和处理大量数据的主要方法。[熟悉]

IM/数据模型

[4 个核心二级课时]

知识点:

- 数据模型。
- 概念模型(例如:实体关系图,UML 图)。
- 电子表格模型。
- 关系数据模型。
- 面向对象的模型(交叉参照 PL/面向对象程序设计)。

- 半结构化数据模型(例如:使用 DTD 或 XML 模式进行表示)。

学习成果:

1. 比较和对比不同类型数据的数据模型,包括内部结构。[评估]
2. 描述建模符号的概念(例如,实体关系图或 UML)以及它们的使用方法。[熟悉]
3. 定义用于关系数据模型的基本术语。[熟悉]
4. 描述关系数据模型的基本原理。[熟悉]
5. 应用关系数据模型的建模概念和表示法。[运用]
6. 描述面向对象模型的主要概念,如对象的身份、类型的构造函数、封装、继承、多态和版本控制。[熟练]
7. 描述关系数据模型和半结构化数据模型之间的差异。[评估]
8. 对于一个给定的关系模式给出它的一个半结构化等价表示(例如,DTD 或 XML 模式)。[运用]

IM/索引

[选修]

知识点:

- 索引对查询性能的影响。
- 索引的基本结构。
- 在内存中保持数据的缓冲区。
- 用 SQL 创建索引。
- 对文本进行索引。
- 对网站进行索引(例如,Web 爬取)。

学习成果:

1. 为一批资源生成索引文件。[运用]
2. 解释倒排索引在文档查找中的作用。[熟悉]
3. 解释词干和停用词是如何影响索引的。[熟悉]
4. 对于给定的关系模式确定适当的索引和查询集。[运用]
5. 分别在使用索引和不使用索引的情况下估计信息检索的时间。[运用]
6. 描述 Web 抓取的主要挑战,例如,检测重复的文件,确定抓取的边界。[熟悉]

IM/关系数据库

[选修]

知识点:

- 概念模式到关系模式的映射。
- 实体和引用完整性。
- 关系代数与关系演算。
- 关系数据库设计。
- 函数依赖。
- 模式分解;无损连接和分解的依赖保持属性。
- 候选码、超级码和属性的闭包。
- 巴斯 - 科德范式(BCNF)。
- 多值依赖(第四范式 4NF)。
- 连接依赖(PJNF,第五范式 5NF)。
- 表示理论。

学习成果:

1. 用从实体 - 关系模型建立的概念模型建立关系模式。[运用]

2. 解释并举例说明和展示实体完整性约束和参照完整性约束的概念(包括外键概念的定义)。[运用]

3. 说明数学集合论中的关系代数运算(包括 并集、交集、差集、笛卡儿积)以及专门为关系数据库开发的关系代数运算(选择(限定)、投影、连接、分割)。[运用]

4. 用关系代数写出查询。[运用]

5. 写出元关系演算查询。[运用]

6. 确定属于一个关系的同一个子集的两个或多个属性之间的功能依赖。[评估]

7. 建立表示为主键和外键的约束与函数依赖的联系。[运用]

8. 在给定的函数依赖集下计算一组属性的闭包。[运用]

9. 对于一个给定的函数依赖关系,确定一组属性能否形成一个超键与/或候选键。[评估]

10. 评估一个给定的分解,说明它是否为无损连接并满足原函数依赖。[评估]

11. 描述 BCNF、PJNF、5NF 等范式的性质。[熟悉]

12. 解释规范化对数据库操作,特别是查询优化的效率影响。[熟悉]

13. 描述什么是多值依赖以及它所确定的约束类型。[熟悉]

IM/查询语言

[选修]

知识点:

- 数据库语言概述。
- SQL 语言(数据定义、查询表示、数据更新、约束、完整性)。
- 选择。
- 投影。
- 选择 - 投影 - 连接。
- 聚合和分组。
- 子查询。
- QBE 和第四代环境。
- 在传统编程语言中调用非过程式查询语句的不同方法。
- 其他主要查询语言简介(例如:XPATH、SPARQL)。
- 存储过程。

学习成果:

1. 用 SQL 创建关系数据库的模式,定义表的主键,定义实体完整性和参照完整性等约束。[运用]

2. 用 SQL 来创建表和从数据库中用 SELECT 语句来检索数据。[运用]

3. 评价查询处理的策略,选择优化的策略。[评估]

4. 填充关系模板来实现非过程式的查询实例,并返回所需要的查询结果。[运用]

5. 在 C + + 或 Java 等独立的语言中嵌入面向对象的查询语句(例如:SELECT Col. Method( ) FROM Object)。[运用]

6. 编写带有参数的存储过程,使其具有一定的控制流程,实现给定的功能。[运用]

IM/事务处理

[选修]

知识点:

- 事务。
- 出错和恢复。
- 并行控制。
- 与存储,特别是缓冲相关的事务管理的交互。

学习成果:

1. 把 SQL 语句嵌入到应用程序中形成事务。[运用]

2. 解释隐式事务提交的概念。[熟悉]

3. 描述高效事务执行的有关问题。[熟悉]

4. 解释何时和为什么需要回滚操作以及日志如何保障正确的回滚。[评估]

5. 解释并发控制机制的不同隔离级别带来的影响。[评估]

6. 选择正确的隔离级别来实现给定的事务协议。[评估]

7. 在应用程序中确立正确的事务边界。[评估]

IM/分布式数据库

[选修]

知识点:

- 分布式 DBMS。

➢ 分布式数据存储。

➢ 分布式查询处理。

➢ 分布式事务模型。

➢ 同构和异构的解决方案。

➢ 客户机 - 服务器分布式数据库(交叉参照 SF/计算范式)。

➢ 并行数据库管理系统。

➢ 并行数据库系统的体系结构:共享内存,共享磁盘,无共享。

➢ 加速和规模扩展,例如,使用 MapReduce 处理模型(交叉参照 CN/处理、PD/并行分解)。

➢ 数据复制和弱一致性模型。

学习成果:

1. 解释在分布式数据库的设计过程中数据分片、数据复制和资源分配等技术。[熟悉]

2. 评估执行分布式查询的简单策略,从中选择能最大限度地减少数据传输量的策略。[评估]

3. 解释两阶段提交协议是如何提交一个访问存储在多个节点上数据库的事务的。[熟悉]

4. 描述基于区分复制技术和投票法的分布式并发控制。[熟悉]

5. 描述客户端 - 服务器模型软件系统的三个层次。[熟悉]

IM/物理数据库设计

[选修]

知识点:

- 存储和文件结构。

- 索引文件。
- 哈希(也称“散列”)文件。
- 签名文件。
- B 树。
- 带有稠密索引的文件。
- 可变长记录文件。
- 数据库效率和性能调优。

学习成果:

1. 解释记录、记录类型、文件的概念以及将文件记录放置在磁盘的不同技术。[熟悉]
2. 给出主索引、辅助索引以及聚类索引的应用实例。[熟悉]
3. 区分非稠密索引和稠密索引。[评估]
4. 使用 B – 树实现动态多级索引。[运用]
5. 解释内部和外部散列技术的理论与应用。[熟悉]
6. 使用哈希技术实现动态文件扩展。[运用]
7. 描述散列、压缩和高效的数据库搜索之间的关系。[熟悉]
8. 评价不同的散列方案的成本和收益。[评估]
9. 解释物理数据库设计将如何影响数据库的事务效率。[熟悉]

IM/数据挖掘

[选修]

知识点:

- 数据挖掘的用途。
- 数据挖掘的算法。
- 关联和序列模式。
- 数据聚类。
- 购物篮分析。
- 数据清洁。
- 数据可视化(交叉参照 GV/可视化和 CN/交互式可视化)。

学习成果:

1. 比较和对比数据挖掘技术在研究和应用两个不同领域的用法。[评估]
2. 解释发现关联性在市场购物篮数据中的价值。[熟悉]
3. 归纳可以通过关联规则挖掘发现的模式类型的特征。[评估]
4. 描述如何扩展关系数据系统,以利用关联规则完成模式发现。[熟悉]

5. 从方法学角度评价有效的数据挖掘应用研究。[评估]

6. 识别和表征数据中的噪声源、冗余、异常值。[评估]

7. 识别用于实现数据挖掘的闭环过程的机制(在线聚集、随时行为、交互式可视化)。[熟悉]

8. 描述为什么各种闭环过程能提高数据挖掘的效率。[熟悉]

IM/信息存储和检索

[选修]

知识点:

- 文件、电子出版、标记和标记语言。
- 查找树、倒排文件、PAT 树、签名文件、索引。
- 形态分析、词干、短语、停用词表。
- 词频分布、不确定性、模糊性、权重。
- 向量空间、概率、逻辑和先进的模型。
- 信息需求、关联性、评价、有效性。
- 叙词表、本体、分类和类型、元数据。
- 书目信息、文献计量学、引文。
- 路由和(社区)过滤。
- 多媒体搜索、信息搜寻行为、用户建模、反馈。
- 信息的摘要和可视化。
- 分面搜索(如使用引用、关键词、分类模式)。
- 数字图书馆。
- 数字化、存储、交换、数字对象、组合和包。
- 元数据与编目。
- 命名、资料库、档案。
- 归档和保存、完整性。
- 空间(概念、地域、二维/三维、虚拟现实)。
- 架构(代理、总线、封装/中介)、互操作性。
- 服务(搜索、链接、浏览等等)。
- 知识产权管理、隐私和保护(水印)。

学习成果:

1. 解释信息存储和检索的基本概念。[熟悉]

2. 描述哪些问题与高效的信息检索相关。[熟悉]

3. 给出采用不同搜索策略的应用,并解释为什么该搜索策略适用于该应

用。[评估]

4. 设计并实现一个小型或中型的信息存储和检索系统或数字图书馆。[运用]

5. 描述一些在数字图书馆中与归档和保存信息相关的技术解决方案。[熟悉]

IM/多媒体系统

[选修]

知识点:

- 输入和输出设备、设备驱动程序、控制信号和协议、DSP。
- 标准(如音频、图形、视频)。
- 应用、媒体编辑器、写作系统、写作。
- 流/结构、捕获/表示/变换、空间/域、压缩/编码。
- 基于内容的分析、索引、音频检索、图像、动画、视频。
- 表示、渲染、同步、多模态融合/接口。
- 实时传递、服务质量(包括性能)、容量规划、音频/视频会议、视频点播。

学习成果:

1. 描述媒体和支持常用的多媒体信息系统的相关设备。[熟悉]

2. 展示基于内容的信息分析在多媒体信息系统中的使用。[运用]

3. 评价音频、视频、图形、颜色以及其他信息表示概念在多媒体演示中的使用。[评估]

4. 使用写作系统实现一个多媒体应用。[运用]

5. 针对媒体和多媒体中的每一种标准,使用非技术性语言来描述制定标准的目的,并阐述人类视听感知对标准的局限性存在的敏感度。[熟悉]

6. 描述多媒体计算机系统的要求(包括识别支撑工具和适当标准),使之能够容载一系列多媒体应用。[熟悉]

智能系统(Intelligent Systems IS)

人工智能研究的是用传统方法解决起来很困难或者不现实的问题的求解。它被广泛用于支撑日常应用,如电子邮件、文字处理和搜索,以及感知环境并合理地与环境交互的自治代理的设计与分析。

这些问题的求解依赖于一系列通用和专门化的知识表示方案、问题求解机制和学习技术。它们处理感知(例如,语音识别、自然语言理解、计算机视觉),问题求解(例如,搜索、规划),行动(例如,机器人),以及支持它们所需的架构(例如,代理、多代理)。人工智能的学习将使得学生能够确定一种人工智能方法是否适用于一个给定的问题,发现合适的表示和推理机制,实现并对其进行评估。

IS. 智能系统(10 个核心二级学时)

	核心一级学时	核心二级学时	选修
IS/基础问题		1	是
IS/基本搜索策略		4	否
IS/基本知识表达和推理方法		3	否
IS/基本机器学习方法		2	否
IS/高级搜索方法			是
IS/高级知识表达和推理方法			是
IS/不确定性推理方法			是
IS/代理			是
IS/自然语言处理技术			是
IS/高级机器学习方法			是
IS/机器人学			是
IS/感知和计算机视觉			是

IS/基础问题

[1 个核心二级学时]

知识点:

- 人工智能问题概述,成功的人工智能应用最新实例。
- 什么是智能行为。

➢ 图灵测试。

➢ 理性与非理性推理。

• 问题的特征。

➢ 完全 vs 部分可观察。

➢ 单个 vs 多个代理。

➢ 确定性 vs 随机性。

➢ 静态 vs 动态。

➢ 离散 vs 连续。

• 代理的本质。

➢ 自治 vs 半自治。

➢ 反身性,基于目标的和基于效用的。

➢ 感知和与环境交互的重要性。

• 哲学和伦理问题。[选修]

学习成果

1. 描述图灵测试和“中文房间”思想实验。[熟悉]

2. 区分最优推理/行为和类人推理/行为的概念。[熟悉]

3. 确定一个智能系统一定能解决的问题的特征。[评估]

IS/基本搜索策略

[4 个核心二级学时]

交叉参照 AL/基础分析,AL/算法策略,AL/基础数据结构及算法

知识点:

• 问题空间(状态、目标和算子),通过搜索求解问题。

• 因素化表示(将状态构造为变量)。

• 非启发式搜索(广度优先、深度优先、迭代加深式深度优先)。

• 启发式搜索(爬山法,通常的最好优先搜索,A^*)。

• 搜索的空间和时间效率。

• 双人博弈(介绍极小极大搜索)。

• 约束满足(回溯和局部搜索方法)。

学习成果:

1. 对用自然语言(例如,英语)描述了初始状态和目标状态以及运算方法的一个问题,给出其问题空间的形式化表示。[运用]

2. 描述启发式方法的作用,并说明完全性、最优性、时间复杂度和空间复杂度之间的权衡。[熟悉]

3. 描述搜索空间的组合爆炸问题及其后果。[熟悉]

4. 对一个问题,选择并实现一个合适的非启发式搜索算法,并分析它的时间和空间复杂度。[运用]

5. 对一个问题,通过设计必要的启发式评估函数,选择并实现一个合适的启发式搜索算法。[运用]

6. 对于一个给定问题,评估一种启发式策略是否可给出/能够保证最优解。[评估]

7. 构造一个用自然语言(例如,英语)描述的约束满足问题,并使用时序回溯算法或随机局部搜索实现它。[运用]

8. 比较和对比采用博弈的基本搜索问题。[熟悉]

IS/基本知识表达和推理方法

[3 个核心二级学时]

知识点:

- 总结和评价命题、谓词逻辑(交叉参照 DS /基础逻辑)。
- 归结法和定理证明(仅限命题逻辑)。
- 正向链接、反向链接法。
- 总结和评价概率推理和贝叶斯定理(交叉参照 DS /离散概率)。

学习成果:

1. 将自然语言(例如,英语)句子翻译为谓词逻辑语句。[运用]

2. 将一个逻辑语句转化为子句形式。[运用]

3. 在逻辑语句集合上应用归结法以回答查询。[运用]

4. 对实际问题进行概率推理,使用贝叶斯定理来确定某假设的概率。[运用]

IS/基本机器学习方法

[2 个核心二级学时]

知识点:

- 各种机器学习任务的定义和实例,包括分类。
- 归纳学习。
- 简单统计学习,如朴素贝叶斯分类器、决策树。
- 过拟合问题。
- 测量分类器的精度。

学习成果:

1. 列出三种主流学习方式之间的差异:监督学习、强化学习和非监督学习。

[熟悉]

2. 识别分类任务的实例,包括可用的输入特征和将预测的输出。[熟悉]

3. 解释归纳和演绎学习的区别。[熟悉]

4. 通过具体问题来描述过拟合。[熟悉]

5. 在分类任务中应用简单的统计学习算法,如朴素贝叶斯分类器,并测量分类器的精度。[运用]

IS/高级搜索方法

[选修]

注意:关于分支定界与动态规划的知识点列在 AL/算法策略里面。

知识点:

- 构建搜索树、动态搜索空间、搜索空间的组合爆炸。
- 随机搜索。

➢ 模拟退火。

➢ 遗传算法。

➢ 蒙特卡罗树搜索。

- A * 搜索的实现、定向搜索。
- 期望最大搜索(Expectimax search,MDP 求解)和机会节点。

学习成果:

1. 设计并实现遗传算法来求解问题。[运用]

2. 设计并实行模拟退火调度以避免局部极小。[运用]

3. 设计并实现 A * /定向搜索来求解问题。[运用]

4. 在极小极大搜索中应用 α - β 剪枝来修剪双人博弈中的搜索空间。[运用]

5. 比较和对比遗传算法与经典搜索技术。[评估]

6. 比较和对比各种启发式搜索算法对于给定问题的适用性。[评估]

IS/高级知识表达和推理方法

[选修]

知识点:

- 知识表示问题.

➢ 描述逻辑。

➢ 本体工程。

- 非单调推理(例如,非经典逻辑、缺省推理)。
- 论证。

- 关于动作和变化的推理(例如,情景和事件演算)。
- 时间与空间推理。
- 基于规则的专家系统。
- 语义网络。
- 基于模型和基于案例的推理。
- 规划。

➢ 偏序和全序规划。

➢ 规划图。

➢ 分层规划。

➢ 规划和执行,包括条件规划和持续规划。

➢ 移动代理/多代理规划。

学习成果:

1. 比较和对比用于表示结构化知识的最常见模型,并突出它们的长处和短处。[评估]

2. 识别非单调推理的构成及其作为信念系统的一个代表性机制的有效性。[熟悉]

3. 比较和对比表征不确定性的基本技术。[评估]

4. 比较和对比定性表示的基本技术。[评估]

5. 应用情景和事件演算求解与动作和变化有关的问题。[运用]

6. 说明时间和空间推理之间的区别,以及它们怎么相互关联。[熟悉]

7. 说明基于规则、基于事件、基于模型的推理技术之间的差异。[熟悉]

8. 定义规划系统的概念,说明它与经典的搜索技术的不同。[熟悉]

9. 描述规划即搜索、基于算子的规划以及命题规划之间的区别,并给出每种方法最适用领域的实例。[熟悉]

10. 说明单调和非单调推理之间的区别。[熟悉]

IS/不确定性推理方法

[选修]

知识点:

- 总结和评价基本概率(参见 DS/离散概率)。
- 随机变量及概率分布.

➢ 概率公理。

➢ 概率推理。

➢ 贝叶斯规则。

- 条件独立性。
- 知识表示。

➢ 贝叶斯网络。

- 精确推理及其复杂性。
- 随机抽样(蒙特卡罗)方法(如 Gibbs 抽样)。

➢ 马尔可夫网络。

➢ 关系概率模型。

➢ 隐马尔可夫模型。

- 决策理论。

➢ 偏好和效用函数。

➢ 预期效用最大化。

学习成果:

1. 采用贝叶斯规则确定给定证据假设的概率。[运用]
2. 说明为什么条件独立性断言可以使概率系统的效率更高。[评估]
3. 识别适用不确定性推理的知识表示实例。[熟悉]
4. 陈述精确推理的复杂性。识别近似推理方法。[熟悉]
5. 设计并实现至少一种不确定性推理的知识表示。[运用]
6. 描述时间概率推理的复杂性。[熟悉]
7. 设计并实现一个隐马尔可夫模型作为时间概率系统的一个实例。[运用]
8. 描述偏好和效用函数之间的关系。[熟悉]
9. 说明效用函数和概率推理为何可以结合做出合理的决定。[评估]

IS/代理

[选修]

交叉参照 HCI/协同和通信。

知识点:

- 代理定义。
- 代理体系结构(例如:反应、层级、认知)。
- 代理理论。
- 合理性、博弈论。

➢ 决策论代理。

➢ 马尔可夫决策过程(MDP)。

- 软件代理、个人助理和信息存取。

➢ 协同代理。
➢ 信息收集代理。
➢ 可信代理(虚拟人物,在代理中的情绪建模)。
• 学习代理。
• 多代理系统。
➢ 合作代理。
➢ 代理团队。
➢ 竞争代理(例如,拍卖、投票)。
➢ 群体系统和生物启发式模型。

学习成果:

1. 列出智能代理的定义特征。[熟悉]
2. 描述标准代理体系结构的特征,并进行对比。[评估]
3. 描述代理理论在软件代理、个人助理和可信代理等领域的应用。[熟悉]
4. 描述的学习代理使用的主要范式。[熟悉]
5. 使用适当的例子演示多代理系统怎样支撑代理的交互。[运用]

IS/自然语言处理技术

[选修]

交叉参照 HCI /新型交互技术

知识点:

• 确定和随机文法。
• 语法分析算法。
➢ 控制流图和图表分析器(例如 CYK)。
➢ 概率控制流图和加权 CYK。
• 表示意义/语义。
➢ 基于逻辑的知识表示。
➢ 语义角色。
➢ 时序表征。
➢ 信念、愿望、意图。
• 基于语料库的方法。
• N - gram 和隐马尔可夫模型。
• 平滑和退避。
• 使用的例子:词性标注和形态。
• 信息检索(交叉参照 IM/信息存储与检索)。

➢ 向量空间模型。

- TF &IDF。

➢ 准确率和召回率。

- 信息提取。
- 语言翻译。
- 文本分类,归类。

➢ 词袋模型。

学习成果:

1. 定义和对比确定和随机文法,并举例说明它们的充分性。[评估]

2. 仿真、应用或者实现用于自然语言语法分析的经典和随机算法。[运用]

3. 认识表示意义的挑战。[熟悉]

4. 列出使用标准语料库的优点。举出几个支持一系列自然语言处理任务的现有语料库的例子。[熟悉]

5. 认识信息检索、语言翻译和文本分类的技术。[熟悉]

IS/高级机器学习方法

[选修]

知识点:

- 各种机器学习任务的定义和实例。
- 基于统计的通用学习,参数估计(最大似然估计)。
- 归纳逻辑程序设计(ILP)。
- 监督学习。

➢ 学习决策树。

➢ 学习神经网络。

➢ 支持向量机(SVMs)。

- 集成。
- 最近邻算法。
- 无监督学习和聚类。

➢ EM。

➢ K 均值。

➢ 自组织映射。

- 半监督学习。
- 学习图模型(交叉参照 IS/不确定性推理方法)。
- 效果评价(如交叉验证、ROC 曲线下的面积)。

- 学习理论。
- 过拟合问题、维度灾难。
- 强化学习。

➢ 探索和利用的权衡。

➢ 马尔可夫决策过程。

➢ 值和策略迭代。

- 机器学习算法在数据挖掘中的应用(交叉参照 IM/数据挖掘)。

学习成果:

1. 说明三个主要类型的学习之间的差异:监督学习、强化学习、无监督学习。[熟悉]

2. 实现监督学习、强化学习和非监督学习的简单算法。[运用]

3. 对于特定的问题域,确定这三个学习方式中哪种最为合适。[运用]

4. 比较和对比下列技术,并给出每一种策略适用的实例:决策树、神经网络和信念网络。[评估]

5. 评估简单学习系统在实际数据集上的效果。[评估]

6. 描述学习理论的现状,包括取得的成果和存在的不足。[熟悉]

7. 说明过拟合问题,包括检测与管理这一问题的技术。[运用]

IS/机器人学

[选修]

知识点:

- 概述:问题与进展。

➢ 机器人系统最新动态,包括它们的传感器和传感器处理概述。

➢ 机器人控制体系结构,如协商与无功控制和 Braitenberg 车辆。

➢ 世界建模与世界模型。

➢ 感知与控制中固有的不确定性。

- 配置空间和环境映射。
- 解释不确定的传感器数据。
- 定位和映射。
- 导航与控制。
- 运动规划。
- 多机器人协同。

学习成果:

1. 列出当今最先进的机器人系统的能力与局限性,包括它们的传感器和通

知这些系统的关键传感器处理。[熟悉]

2. 将传感器、执行器和软件集成到一个执行特定任务的机器人中。[运用]

3. 使用协商、反应和/或混合控制体系结构对机器人进行编程,使之完成简单任务。[运用]

4. 在机器人的配置空间中实现基本运动规划算法。[运用]

5. 描述常见机器人传感器和执行器的不确定性;陈述减少这些不确定性的策略。[熟悉]

6. 列出机器人对于外部环境的表示差异,包括它们的优点和缺点。[熟悉]

7. 比较和对比至少三种机器人在已知或未知环境中的导航策略,包括它们的优点和缺点。[评估]

8. 至少描述一种协调多个机器人动作与感知使其完成单个任务的方法。[熟悉]

IS/感知和计算机视觉

[选修]

知识点:

- 计算机视觉。

➢ 图像采集、表示、处理和性质。

➢ 形状表示,对象识别和分割。

➢ 运动分析。

- 音频和语音识别。
- 识别中的模块性。
- 模式识别方法(交叉参照 IS/高级机器学习方法)。

➢ 分类算法和分类效果的度量。

➢ 统计技术。

学习成果:

1. 总结图像和对象识别在人工智能的重要性,并指出这种技术的几个重要的应用。[熟悉]

2. 列出至少三种图像分割方法,如阈值、基于边缘和基于区域的算法,以及界定它们的特征、优势和劣势。[熟悉]

3. 实现基于轮廓和/或基于区域形状表示的二维物体识别。[运用]

4. 区分声音识别、语音识别和说话人识别的目标,并确定在这些情况下应如何对原始音频信号进行不同的处理。[熟悉]

5. 举出至少两个将源数据从一个感官域转换到另一个感官域的实例,例

如,触觉数据解释为单波段的二维图像。[熟悉]

6. 实现真实数据上的特征抽取算法,如图像的边角检测器或描述一小段音频信号的傅里叶系数的向量。[运用]

7. 实现一个合并特征形成更高层次的感知的算法,例如,视觉基本实体中的轮廓或多边形或音频信号中的音素假说。[运用]

8. 实现一个分割输入感知到输出类别的分类算法,并定量评价分类结果。[运用]

9. 针对(8)的分类算法,如果将其中特征提取替换成一个其他可能方法(不论实现与否),请评估其效果。[评估]

10. 描述至少三种分类方法,分析它们适用的前提以及优缺点。[熟悉]

网络与通信(Networking and Communication,NC)

互联网和计算机网络已无处不在,其正确运行为目前不断增长的计算应用提供了重要支撑。在当前与未来的计算环境中,固定和移动网络都将是其关键组成部分。如果没有网络的存在,众多应用程序将无法运行。而随着应用的发展,这种依赖性很有可能会继续增强。

本知识领域的学习目标总体上可归纳为如下几点。

- 理解网络世界。网络互联程度越来越高,并且其使用也将不断普及。为了灵活地运用网络,学生必须理解网络组成和运行的关键原理。
- 为后续学习提供基础。由于网络领域是不断快速演化的,网络课程应该重视基础,为后续高级课程(例如,网络设计、网络管理、传感器网络等)做准备。
- 理论与实践相结合。网络存在于现实环境中,许多网络设计方案受实际情况的约束。因此,学生在网络实验、使用工具和编写网络软件过程中应该考虑这些实际情况的约束。

目前存在两类规划网络课程的方式。第一类是自顶向下的方式,即课程从上层应用开始,然后讲授下层的可靠传输、路由和转发等内容;另一类是自底向上的方式,课程从最底层开始,然后介绍网络层、传输层和应用层。

NC. 网络与通信(3 个核心一级学时、7 个核心二级学时)

	核心一级学时	核心二级学时	是否选修
NC/引言	1.5		否
NC/网络应用程序	1.5		否
NC/可靠数据传输		2	否
NC/路由和转发		1.5	否
NC/局域网		1.5	否
NC/资源分配		1	否
NC/移动性		1	否
NC/社交网络			是

NC/引言

[1.5 个核心一级学时]

参见“信息保障与安全(IAS)/网络安全”中网络安全及其应用的相关内容。

知识点:

- 互联网的组织(包括互联网服务提供商 ISP、内容提供商 CP 等)。
- 交换技术(例如,基于电路的交换、基于报文的交换)。
- 网络的物理组成,包括主机、路由器、交换机、ISPs、无线网、局域网(LAN)、接入点和防火墙。
- 分层原理(封装和复用)。
- 网络层次(应用层、传输层、网络层、数据链路层、物理层)。

学习成果:

1. 清楚地表达互联网的组成。[熟悉]
2. 列出和定义恰当的网络术语。[熟悉]
3. 描述一个典型网络架构的分层结构。[熟悉]
4. 识别不同复杂度网络的类型(如边界、核心网络等)。[熟悉]

NC/网络应用程序

[1.5 个核心一级学时]

知识点:

- 命名和地址方案(DNS、IP 地址、统一资源标识符等)。
- 分布式应用程序(客户端/服务器、点对点、云等)。
- HTTP 应用层协议。
- TCP 和 UDP 的复用。
- 网络套接字(Socket)APIs。

学习成果:

1. 列出网络中名字与地址之间的区别和联系。[熟悉]
2. 理解命名方案和资源定位的原理。[熟悉]
3. 实现一个简单基于 Socket 的客户端 - 服务端应用。[运用]

NC/可靠数据传输

[2 个核心二级学时]

这个知识单元与系统基础(SF)有关。参见 SF/状态与状态机和 SF/冗余下的可靠性两个部分。

知识点:

- 错误控制(重传技术、定时器)。
- 流控制(应答、滑动窗口)。
- 性能问题(流水线技术)。

- TCP 协议。

学习成果：

1. 描述可靠传输协议的运行过程。[熟悉]
2. 列出影响可靠传输协议性能的因素。[熟悉]
3. 设计和实现一个简单的可靠传输协议。[运用]

NC/路由和转发

[1.5 个核心二级学时]

知识点：

- 路由与转发的对比。
- 静态路由。
- IP 协议。
- 扩展性问题(分层寻址)。

学习成果：

1. 描述网络层的结构。[熟悉]
2. 描述 IP 报文的转发机制。[熟悉]
3. 列出分层寻址在可扩展性方面的优势。[熟悉]

NC/局域网

[1.5 个核心二级学时]

知识点：

- 多路访问。
- 多路访问的实现方法(指数退避、时分复用等)。
- 局域网。
- 以太网。
- 交换技术。

学习成果：

1. 描述以太帧的转发机制。[熟悉]
2. 描述 IP 网络和以太网的区别。[熟悉]
3. 描述 IP 网络和以太网的联系。[熟悉]
4. 描述多路访问实现方法的步骤。[熟悉]

NC/资源分配

[1 个核心二级学时]

知识点：

- 资源分配的需求。

- 固定分配(TDM、FDM、WDM)与动态分配的对比。
- 端到端与网络辅助方法的对比。
- 公平性。
- 拥塞控制的原理。
- 解决拥塞的方法(例如,内容分发网络 CDN)。

学习成果:

1. 描述网络资源的分配机制。[熟悉]
2. 描述大规模网络的拥塞问题。[熟悉]
3. 比较和对比固定与动态分配技术。[评估]
4. 比较和对比当前解决拥塞问题的不同方法。[评估]

NC/移动性

[1 个核心二级学时]

知识点:

- 蜂窝网的原理。
- 802.11 网络。
- 节点的移动性支持(家代理)。

学习成果:

1. 描述无线网络的组成。[熟悉]
2. 描述无线网络中移动用户的支持机制。[熟悉]

NC/社交网络

[选修]

知识点:

- 社交网络概述。
- 社交网络平台示例。
- 社交网络图的结构。
- 社交网络分析。

学习成果:

1. 讨论社交网络的关键原理(如会员、信任机制)。[熟悉]
2. 描述现有社交网络的运行机制。[熟悉]
3. 根据网络数据构建一个社交网络图。[运用]
4. 分析一个社交网络并确定关键性用户。[运用]
5. 给定相关数据,评价社交网络问题的合理性。[评估]

操作系统(Operating Systems OS)

操作系统定义了对硬件行为的抽象并管理计算机用户之间的资源共享。本知识领域的内容涵盖了操作系统的基本知识,包括建立操作系统的网络接口、讲授内核和用户模式之间的区别、发展操作系统设计和实现的关键方法等。本知识领域和系统基础(SF)、网络与通信(NC)、信息保障和安全(IAS)以及并行和分布式计算(PD)等知识领域相辅相成,其中系统基础(SF)和信息保障和安全(IAS)也开始面临着一些与时俱进的问题。例如,系统基础中的性能、虚拟化与隔离以及资源分配与调度等问题;并行和分布式计算中的并行基础问题;信息保障和安全中的深度取证与安全问题。总之,操作系统领域中很多课程的内容都取材于这些交叉知识领域。

OS. 操作系统(4 个核心一级学时,11 个核心二级学时)

	核心一级学时	核心二级学时	是否选修
OS/操作系统概述	2		否
OS/操作系统原理	2		否
OS/并发		3	否
OS/调度和分发		3	否
OS/内存管理		3	否
OS/安全和防护		2	否
OS/虚拟机			是
OS/设备管理			是
OS/文件系统			是
OS/实时与嵌入式系统			是
OS/容错性			是
OS/系统性能评估			是

OS/操作系统概述

[2 个核心一级学时]

知识点:

- 操作系统的角色和目标。

- 典型操作系统的功能。
- 支持客户端 - 服务器模型和手持设备的机制。
- 设计要素(效率、健壮性、灵活性、可移植性、安全性、兼容性)。
- 安全、网络、多媒体和视窗系统的影响。

学习成果:

1. 解释现代操作系统的目标和功能。[熟悉]
2. 分析操作系统设计中的权衡。[运用]
3. 描述现代操作系统在易用性、效率和演化能力等方面的功能。[熟悉]
4. 讨论网络操作系统,基于客户端 - 服务器模式的操作系统、分布式操作系统与单用户操作系统的区别。[熟悉]
5. 了解操作系统的潜在隐患以及相应的安全防范机制。[熟悉]

OS/操作系统原理

[2 个核心一级学时]

知识点:

- 架构方法(单内核、层次式、模块式、微内核)。
- 抽象、进程和资源。
- 应用编程接口(APIs)的概念。
- 软硬件技术和应用需求的演化。
- 设备的组织。
- 中断机制及其实现。
- 用户/系统状态及其保护的概念、内核态迁移。

学习成果:

1. 解释逻辑层的概念。[熟悉]
2. 解释分层模式中建立抽象层的优势。[熟悉]
3. 描述 APIs 和中间件的价值。[评估]
4. 描述应用软件的资源使用机制和系统软件的资源管理机制。[熟悉]
5. 对比操作系统的内核态和用户态。[运用]
6. 讨论使用中断处理的优劣。[熟悉]
7. 解释设备列表和驱动 I/O 队列的使用。[熟悉]

OS/并发

[3 个核心二级学时]

知识点:

- 状态和状态图(参见 SF/状态与状态机)。

- 架构(就绪列表、进程控制块等)。
- 分发及上下文切换。
- 中断的功能。
- 操作系统对象的原子访问管理。
- 同步原语的实现。
- 多处理器相关问题(自旋锁、重入)(参见 SF/并行性)。

学习成果:

1. 描述操作系统框架对并发的需求。[熟悉]
2. 展示多个独立任务同时运行所引发的潜在问题。[运用]
3. 总结操作系统中并发的实现机制并描述其优势。[熟悉]
4. 解释任务历经的不同状态以及实现支持多任务管理的相应数据结构。[熟悉]
5. 总结实现操作系统同步的技术(描述基于操作系统原语的信号实现机制等)。[熟悉]
6. 描述如何通过中断、分发、上下文切换等机制实现操作系统的并发支持。[熟悉]
7. 构建一个针对简单问题域的状态转移图。[运用]

OS/调度和分发

[3 个核心二级学时]

知识点:

- 抢占和非抢占调度(参见 SF/资源分配和调度技术与 PD/并行性能)。
- 调度器和策略(参见 SF/资源分配和调度技术与 PD/并行性能)。
- 进程和线程(参见 SF/计算范式)。
- 截止期限和实时性问题。

学习成果:

1. 比较和对比操作系统中抢占和非抢占调度的常用算法,如优先级、性能对比和公平分享机制等。[运用]
2. 描述调度算法和应用领域的联系。[熟悉]
3. 描述处理器调度的类型,包括短期、中期、长期以及 I/O 等。[熟悉]
4. 描述进程和线程的区别。[运用]
5. 比较和对比实时调度的静态与动态方法。[运用]
6. 讨论抢占调度和截止期限调度的必要性。[熟悉]
7. 掌握调度算法的逻辑并应用于磁盘 I/O、网络调度、工程调度等其他领

域。[运用]

OS/内存管理

[3 个核心二级学时]

知识点:

- 物理内存和内存管理硬件的概述。
- 工作集和抖动。
- 缓存(参见 AR/存储系统的组织与结构)。

学习成果:

1. 解释内存架构和性能代价间的权衡。[熟悉]
2. 归纳虚拟内存在缓存和分页中的应用原理。[熟悉]
3. 评估内存大小和处理器速度间的权衡。[评估]
4. 阐明任务内存分配的不同方法,并陈述每种方法的相对优缺点。[评估]
5. 描述缓存的必要性和使用方法(性能和接近程度,缓存给隔离和虚拟机抽象带来的复杂性)。[熟悉]
6. 讨论抖动发生的原因及其识别和管理技术。[熟悉]

OS/安全和防护

[2 个核心二级学时]

知识点:

- 系统安全概述。
- 隔离策略和隔离机制。
- 安全方法和设备。
- 保护、访问控制和认证。
- 备份。

学习成果:

1. 清楚地表达操作系统在安全和防护方面的需求。(参见 IAS/安全策略和管理/多类系统的操作系统安全)[评估]
2. 总结操作系统在安全和防护方面的特征及局限性。(参见 IAS/安全策略和管理)[熟悉]
3. 解释操作系统中资源访问控制的现有机制。(参见 IAS/安全策略和管理/访问控制/IT 系统的安全系统配置)[熟悉]
4. 按照一定的安全策略实现简单的系统管理,如创建账户、设定权限、应用补丁、常规备份等。(参见 IAS/安全策略和管理)[运用]

OS/虚拟机

[选修]

知识点:

- 虚拟化的类型(包括硬件/软件、操作系统、服务器、服务、网络)。
- 分页和虚拟内存。
- 虚拟文件系统。
- 虚拟机管理程序。
- 便携式虚拟化;模拟与隔离的对比。
- 虚拟化的成本。

学习成果:

1. 解释虚拟内存的概念以及它在硬件和软件层面的实现方式。[熟悉]
2. 区分模拟和隔离。[熟悉]
3. 评估虚拟化的权衡。[评估]
4. 讨论虚拟机管理程序以及不同类型虚拟机管理程序协作的必要性。[运用]

OS/设备管理

[选修]

知识点:

- 串行和并行设备的特点。
- 抽象设备的差异。
- 缓存机制。
- 直接内存访问。
- 故障恢复。

学习成果:

1. 解释串行和并行设备的主要差异,并分别定义它们适用的环境。[熟悉]
2. 定义操作系统维护的物理设备和虚拟设备间的关系。[运用]
3. 解释缓存的概念,并描述实现机制。[熟悉]
4. 区别将一系列设备(包括手持设备、网络、多媒体)接入电脑时所使用的各种机制,并说明其对操作系统设计的影响。[运用]
5. 描述直接内存访问的优缺点,并讨论其适用场景。[运用]
6. 定义故障恢复的需求。[熟悉]
7. 实现一个适用于多种设备的简单设备驱动。[运用]

OS/文件系统

[选修]

知识点:

- 文件(数据、元数据、操作、组织、缓存、时序、非时序)。
- 目录(内容与结构)。
- 文件系统(分区、挂载/卸载、虚拟化文件系统)。
- 标准化实现技术。
- 内存映射技术。
- 专用文件系统。
- 命名、搜索、接入、备份。
- 日志与日志结构的文件系统。

学习成果:

1. 描述在系统设计过程中需要注意的事项。[熟悉]
2. 比较和对比文件组织的不同方法,并指出它们各自的优缺点。[运用]
3. 总结硬件发展是如何改变文件系统设计和管理中的优先级。[熟悉]
4. 概括日志的作用以及日志结构文件系统对系统容错能力带来的提升。[熟悉]

OS/实时与嵌入式系统

[选修]

知识点:

- 进程和任务调度。
- 实时环境中的内存/硬盘管理要求。
- 故障、风险和恢复。
- 实时系统中的特殊问题。

学习成果:

1. 了解实时系统的定义。[熟悉]
2. 解释实时系统中时延的存在并描述其特点。[熟悉]
3. 总结实时系统存在的一些特殊问题,如风险,以及如何处理这些问题。[熟悉]

OS/容错性

[选修]

知识点:

- 基本概念:可靠、可用的系统(参见 SF/冗余下的可靠性)。
- 时空冗余(参见 SF/冗余下的可靠性)。
- 容错的实现方法。

- 操作系统容错机制(检测、恢复和重启)的实例,包括如何使用这些技术支撑操作系统自身服务。

学习成果:

1. 解释容错性、可靠性、有效性的联系。[熟悉]
2. 概述操作系统中容错机制的实现方法。[熟悉]
3. 解释操作系统错误恢复机制。[熟悉]

OS/系统性能评估

[选修]

知识点:

- 系统性能评估的必要性(参见 SF/性能/性能指标图)。
- 评估对象(参照 SF/性能/性能指标图)。
- 系统性能策略,例如,缓存、分页、调度、内存管理和安全。
- 评估模式(确定型、分析型、仿真或者特定实现)。
- 如何收集评估数据(数据分析和跟踪记录机制)。

学习成果:

1. 描述系统运行的性能指标。[熟悉]
2. 解释系统的主要评估模型。[熟悉]

基于平台的开发(Platform – Based Developcnent PBD)

基于平台的开发是指特定软件平台上的软件设计与开发。与通用目的的编程相比,基于平台的开发需考虑特定平台的约束。例如,Web 编程、多媒体开发、移动计算、应用软件开发和机器人开发都需考虑相应平台所提供的特定服务、API 或硬件的约束。而这些平台是从不同的机器层抽象而来,具有特定的 API 和显著区分的分发/更新机制。基于平台的开发可被用于构建软件生态系统(ecosystem),拥有广泛的应用领域。

由于部分平台特性比较突出,如 Web 开发平台,而 CS2013 课程大纲要求不能指定特定平台,本知识领域虽然强调了许多当前的平台,但并没有将它们列入核心课程中。而因为通用开发技能在其他的知识领域(例如,"软件开发基础(SDF)")中多有涉及,所以本知识领域关注这些技能在特定平台开发上的进一步强化。

PBD. 基于平台的开发(选修)

	核心一级学时	核心二级学时	是否选修
PBD/引言			是
PBD/Web 平台			是
PBD/移动平台			是
PBD/工业平台			是
PBD/游戏平台			是

PBD/引言

[选修]

本知识单元描述基于平台的开发与传统软件开发之间的基本差别。

知识点:

- 平台概述(例如,Web、移动、游戏、工业)。
- 基于指定平台 API 的编程。
- 平台语言概述(例如,Objective – C、HTML5)。
- 平台约束编程。

学习成果:

1. 描述基于平台的开发和通用目的编程的区别。[熟悉]
2. 列出平台语言的特性。[熟悉]
3. 编写和执行一个简单的基于平台的程序。[运用]
4. 列出平台约束编程的优缺点。[熟悉]

PBD/Web 平台

[选修]

知识点:

- Web 编程语言(例如,HTML5、Java Script、PHP、CSS)。
- Web 平台约束。
- 软件即服务(SaaS)。
- Web 标准。

学习成果:

1. 设计与实现一个简单的 Web 应用。[运用]
2. 描述 Web 给开发者带来的约束。[熟悉]
3. 比较和对比 Web 编程与通用目的编程。[评估]
4. 描述软件即服务与传统软件产品的区别。[熟悉]
5. 讨论 Web 标准对软件开发的影响。[熟悉]
6. 根据当前 Web 标准评价一个现有的 Web 应用。[评估]

PBD/移动平台

[选修]

知识点:

- 移动编程语言。
- 移动无线通信的挑战。
- 位置感知的应用。
- 性能/能耗的权衡。
- 移动平台约束。
- 新技术。

学习成果:

1. 在给定移动平台上设计与实现一个移动应用。[运用]
2. 讨论移动平台给开发者带来的约束。[熟悉]
3. 讨论性能与能耗的权衡。[熟悉]
4. 比较和对比移动编程与通用目的编程。[评估]

PBD/工业平台

[选修]

本知识单元涉及“智能系统(IS)/机器人学”。

知识点:

- 工业平台的类型(例如,数学、机器人、工业控制)。
- 机器人软件及其架构。
- 领域特定语言。
- 工业平台约束。

学习成果:

1. 在给定平台上设计与实现一个工业应用。(例如,使用 Lego Mindstorms、Matlab)[运用]
2. 比较和对比领域特定语言与通用目的编程语言。[评估]
3. 讨论给定工业平台给开发者带来的约束。[熟悉]

PBD/游戏平台

[选修]

知识点:

- 游戏平台的类型(例如,Xbox、Wii、PlayStation)。
- 游戏平台语言(例如,C + +、Java、Lua、Python)。
- 游戏平台约束。

学习成果:

1. 在游戏平台上设计与实现一个简单的应用。[运用]
2. 描述游戏平台给开发者带来的约束。[熟悉]
3. 比较和对比游戏编程与通用目的编程。[评估]

并行和分布式计算(Parallel and Distributed Computing ,PD)

过去十年,多处理器计算,包括多核处理器和分布式数据中心方面,呈现爆炸式的发展。这使得并行和分布式计算从过去的选修课程变为本科计算机专业的核心内容。并行计算和分布式计算都要求在逻辑上同时执行多个进程,其计算操作以复杂的方式交叠进行。并行和分布式计算的基础涉及许多领域,包括对基础系统概念的理解,例如,并发和并行执行、状态/内存操作的一致性和延迟,进程之间的通信和同步来源于信息传递和内存共享等的计算模型,以及原子性、一致性和条件等待等算法概念。实际运用中为获得更高的运行速度需要理解并行算法、问题分解的策略、系统结构、策略的具体实现及性能分析和优化。分布式系统强调安全性和容错等问题,重视复制状态的维护,并引入和计算机网络相关的问题。

因为并行计算和如此众多的计算领域相关,至少包括算法、语言、系统 、网络和硬件,很多课程体系会把这些内容分别安排到不同的课程当中,而不是专门的一门课程。虽然我们认为计算机科学正在向这个方向发展,并最终会实现课程的集中,但是从 2013 年的情况看这个过程尚未明朗。我们觉得本知识领域的内容可以指导课程的设计者实现一门完整的集成并行计算知识点的课程。需要指出的是,并发和互斥的知识要点会在系统基础知识领域(System Fundamentals, SF)中出现。所以课程体系可以选择在同一门课程中对并行和并发进行介绍。进一步地,本知识领域的一些知识点和学习成果只包含在选修课范围内作简要介绍。目前,这些知识点存在太多的差异性,共性的东西很少,例如,并行科学计算、过程代数、非阻塞数据结构等,推荐这些内容应当在选修课程中讲授。

因为并行和分布式计算的术语和概念在不同的研究团体中理解不尽相同,这里对这些重要的术语进行简要的描述。以下定义可能并不完整,但是有助于厘清相关概念。

- 并行性(Parallelism):同时使用多个计算资源,通常为了实现更高的计算速度。
- 并发性(Concurrency):有效及正确地管理对资源的并发访问 。
- 活动性(Activity):可以和其他计算任务同时执行,例如,程序、进程、线程、并行执行的硬件部件等。
- 原子性(Atomicity):用于定义一个动作是否从外部观察上是不可分割的

规则和特性,例如,设置一个字节的每一位,传输单个报文,或是完成一次数据事务。

• 共识性(Consensus):两个或多个活动对于一个给定的谓词可以达成一致,例如,一个计数器的数值、一个锁的所有者、线程的终止等。

• 一致性(Consistency):用于规定多个活动中更新的变量值、产生的消息规则和特性要求,防止可能出现的数据竞争问题;例如,顺序一致性是指并行程序访问共享内存时所有变量的状态等价于一个单独的程序交叉地访问所有内存时所产生的结果。

• 组播(Multicast):一个消息发送给很多的接受者,通常对接收者收到消息的顺序没有任何约束条件。组播事件是指发送给指定的监听者或是订阅者的组播消息。

近年来,随着多处理器计算技术的不断发展,并行和分布式计算在本科计算机教学的重要性也随之加大。除了本指南给出的材料以外,建议感兴趣的读者可以参考如下文档:"NSF/TCPP Curriculum Initiative on Parallel and Distributed Computing – Core Topics for Undergraduates",可以在 http://www. cs. gsu. edu/ ~ tcpp/curriculum/上找到。

通用交叉引用的注释:Systems Fundamentals 也包括对并行计算内容的介绍(SF/计算模型, SF/对并行的系统支持,SF/性能)。

SF 知识领域对并行概念的介绍与 PD 知识领域是互补的,在教学顺序上没有先后关系。SF 的教学目标是建立一个统一的视图,帮助学生理解在系统的不同抽象层次(在计算系统的各个级别:逻辑门、处理器、操作系统和服务,都有内在并行度)对同步运行计算任务的支持。而 PD 的教学目标则是聚焦于促进学生理解并行是计算的基本原语,了解并行和分布式编程的相关问题。由于两者的教学目标不同,所以分配给每个知识领域的课时不是冗余的:层次式系统的视图和高级计算的概念都分别安排了不同的核心课时。

PD. 并行和分布式计算 [5 个核心一级学时;10 个核心二级学时]

	核心 1 级学时	核心 2 级学时	是否选修
PD/并行基础	2		否
PD/并行分解	1	3	否
PD/通信和协同	1	3	是
PD/并行算法、分析和编程		3	是

续表

	核心 1 级学时	核心 2 级学时	是否选修
PD/并行体系结构	1	1	是
PD/并行性能			是
PD/分布式系统			是
PD/云计算			是
PD/形式模型和语义学			是

PD/并行基础

[2 个核心 1 级学时]

在学生熟练掌握基本的并行概念(已经包括在 SF 部分)的基础上,讲授由这个概念引发的相关问题,例如:竞争条件和活性。

交叉引用 SF/计算模型和 SF/对并行的系统支持

知识点:

- 多个同时执行的计算。
- 并行的目标(如吞吐率)vs 并发性(如控制共享资源的访问)。
- 并行、通信和协调。

➢ 协调多个并发计算的程序结构。

➢ 同步的需要。

- 在串行编程中无法发现的编程错误。

➢ 数据竞争(共享状态下的同时读与写或者同时写)。

➢ 高级竞争(违反了程序意图的交叉行为,不希望发生的非确定性)。

➢ 缺乏活性和进展(死锁、互斥等待)

学习成果:

1. 区分使用计算资源快速获取解答和有效管理共享资源的不同(交叉引用 GV/基础概念的学习成果 5)。[熟悉]

2. 区分多种支持同步操作的程序结构,它们具有交互实现并能互补的优势。[熟悉]

3. 区别数据竞争和高级竞争。[熟悉]

PD/并行分解

[1 个核心 1 级学时,3 个核心 2 级学时]

(交叉引用 SF/对并行系统的支持)

知识点:

[核心 1 级]

- 通信和协调/同步的必要性。
- 独立性和分割。

[核心 2 级]

- 并行分解的基本知识(交叉引用 SF/对并行的系统支持)。
- 基于任务的分解。

➢ 实现策略,如线程。

- 数据并行分解

➢ 分解策略,例如,SIMD 和 Map – Reduce。

- 实体和反应进程,如请求句柄。

学习成果:

[核心 1 级]

1. 解释为什么在并行程序中同步是必要的。[使用]
2. 懂得什么样的串行程序可以分解为独立的并行模块。[熟悉]

[核心 2 级]

3. 设计一个正确而且可扩展的并行算法。[使用]
4. 运用基于任务的分解方法实现并行算法。[使用]
5. 运用数据并行分解的方法实现并行算法。[使用]
6. 使用实体或反应进程设计程序。[使用]

PD/通信和协调

[1 个核心 1 级学时,3 个核心 2 级学时]

交叉引用 OS/并发机制实现的问题。

知识点:

[核心 1 级]

- 共享内存。
- 一致性及其在防止数据竞争编程中的作用。

[核心二级]

- 消息传递。

➢ 点对点 vs 组播(或基于事件)的消息。

➢ 阻塞式 vs 非阻塞式的消息发送。

➢ 消息缓冲(交叉引用 PF/核心数据结构/队列)。

- 原子性

➢ 定义和测试原子性及安全性需求。

➢ 原子访问和更新的粒度,如何使用诸如临界区、事务等来描述它们。

➢ 互斥机制,包括:锁、信号量、监控和其他构造方法。

■ 失去活性和死锁的可能性(原因、条件、预防措施)。

➢ 组合。

■ 使用同步来组成大粒度的原子访问。

■ 事务类型,包括:乐观或保守的方式。

[选修]

• 一致性。

➢ 循环阻碍、计数器和其他相关构造。

• 条件动作

➢ 条件等待(如使用条件变量)。

学习成果:

[核心 1 级]

1. 使用互斥方法避免给定竞争条件的发生。[使用]

2. 给出一个例子,说明并发活动(如具有数据竞争的程序)的访问次序不是顺序一致性的。[熟悉]

[核心 2 级]

3. 给出一个例子,说明阻塞式消息发送可能会导致死锁。[使用]

4. 解释何时和为什么组播或基于事件的消息传递比其他消息传递更好。[熟悉]

5. 设计一个程序,能够在一组并发任务完成后正确地停机。[使用]

6. 使用正确同步的队列来缓冲活动传递的数据。[使用]

7. 解释为什么要检查先决条件,并且这些条件必须共享相同的原子单元才能有效。[熟悉]

8. 设计一个测试程序展示并发编程的错误,例如,当两个并发活动都要增加一个变量值的时候,其中的一次数据更新就会被遗漏。[使用]

9. 描述至少一种设计技术,可以使用多个锁或者信号量来防止程序失去活性。[熟悉]

10. 描述乐观和保守的并发控制在不同更新冲突率下的价值。[熟悉]

11. 给出一个例子,说明乐观性更新动作在某些情形下可能永远无法完成。[熟悉]

[选修]

12. 使用信号量或者条件变量来阻塞线程的执行,直到预设的条件满足。[使用]

PD/并行算法、分析和编程

[3 个核心 2 级学时]

知识点:

[核心 2 级]

- 关键路径、Work - Span 算法以及和 Amdahl 定律的关系(交叉应用 SF/性能)。
- 加速比和可扩展性。
- 天然(不费力的、非常容易的)并行算法。
- 并行算法的模式(分治法、Map - Reduce、主 - 从等)。

➢ 特定的并行算法(如并行合并排序算法)。

[选修]

- 并行图算法(例如,并行最短路径、并行生成树)(交叉引用 AL/算法策略/分治法)。
- 并行矩阵运算。
- 生产者 - 消费者和管道算法。
- 非可扩展并行算法的例子。

学习成果:

[核心 2 级]

1. 定义“关键路径”、“时间总量 Work”、“时间跨度 Span”。[熟悉]

2. 学会计算算法运行时间的总量和跨度,并找到并行执行图中的关键路径。[使用]

3. 定义“加速比”和解释算法可扩展性的概念。[熟悉]

4. 确定一个并行化程序中的相互独立的任务。[使用]

5. 刻画计算负载的相关特性,这些特性决定是否可以并行地执行该负载[熟悉]

6. 实现并行的分而治之(图算法),并测量它相对其串行版本的经验性能。[使用]

7. 使用 Map - Reduce 模式来分解一个问题(例如,计算一个文档中一些词汇出现的次数)。[使用]

[选修]

8. 给出一个符合生产者 - 消费者模式的计算问题实例。[熟悉]

9. 给出管道模式作为并行化方法的计算问题实例。[熟悉]

10. 实现并行矩阵算法。[使用]

11. 识别生产者 - 消费者算法带来的问题,以及相关的解决机制。[熟悉]

PD/并行体系结构

[1 个核心 1 级学时,2 个核心层 2 级学时]

这里列出的知识点与体系结构和组织(AR)知识领域中的知识单元有关(AR/汇编级计算机组成原理、AR/多处理器和可选体系结构)。这里专注于从应用的角度来看并行体系结构,而体系结构和组织的知识领域从硬件的角度论述这个话题。

[核心 1 级]

- 多核处理器。
- 共享和分布式内存。

[核心 2 级]

- 对称多处理(SMP)。
- 单指令多数据流(SIMD),向量处理。

[选修]

- 图形处理单元(GPU),共处理。
- 佛林(Flynn)分类法。
- 对并行程序设计的指令级支持。

➢ 原子指令,如比较和设置

- 内存问题

➢ 多处理器缓存和高速缓存一致性。

➢ 非均匀存储器访问(NUMA)。

- 拓扑

➢ 互联。

➢ 集群。

➢ 资源共享(如总线和互联)。

学习成果:

[核心 1 级]

1. 解释共享和分布式内存之间的区别。[熟悉]

[核心 2 级]

2. 描述 SMP 体系结构,并注意其关键特性。[熟悉]

3. 描述自然匹配 SIMD 机器的计算任务的特征。[熟悉]

[选修]

4. 描述 GPU 与 CPU 各自的优点和局限性。[熟悉]

5. 解释弗林(Flynn)分类法中每个分类的特性。[熟悉]

6. 描述面向原子操作的汇编级支持。[熟悉]

7. 描述维持高速缓存一致性的挑战。[熟悉]

8. 描述在不同内存和分布式计算机系统拓扑中的关键性能挑战。[熟悉]

PD /并行性能

[选修]

知识点:

- 负载均衡。
- 性能测试。
- 调度和竞争(交叉引用 OS/调度和分发)。
- 评估通信开销。
- 数据管理。

➢ 距离不均造成的通信成本(交叉引用 SF/资源分配与调度技术)。

➢ 缓存的影响(比如伪共享)。

➢ 保持空间局部性。

- 能耗的使用和管理。

学习成果:

1. 检测和纠正负载不均衡。[使用]

2. 计算 Amdahl 定律对一个特定并行算法的影响(交叉引用 SF/资源分配与调度技术)。[使用]

3. 描述数据的分布和布局对算法通信成本的影响。[熟悉]

4. 检测并纠正错误的实例共享。[使用]

5. 解释调度对并行性能的影响。[熟悉]

6. 解释数据本地化的性能影响。[熟悉]

7. 解释并行性能中能量使用的冲突和平衡。[熟悉]

PD/分布式系统

[选修]

知识点:

- 故障(交叉引用 OS/容错性)。

➢ 基于网络(包括分区)和基于节点的故障。

➢ 影响整个系统的保障性(如可用性)。

- 分布式信息传输。

➢ 数据转换与传输 。

➢ 套接字。

➢ 消息队列。

➢ 信息缓存、重试、删除。

- 分布式系统设计权衡。

➢ 延迟和吞吐量。

➢ 一致性,可用性,分区容错性。

- 分布式服务设计。

➢ 有状态和无状态的协议与服务。

➢ 会话(基于连接)设计。

➢ 反应(IO 触发)和多线程的设计。

- 核心的分布式算法。

➢ 选择、发现。

学习成果:

1. 从不同类型的失败中区分网络故障。[熟悉]

2. 解释在分布式故障中,为什么简单的锁等同步结构没有作用。[熟悉]

3. 编写一个程序,执行所需的封送处理并转换成单元消息,例如,两台主机之间通过数据包通信。[使用]

4. 在设定的网络条件下,测量观察主机间的吞吐量和响应时间。[使用]

5. 解释为什么没有分布式系统能够同时具备一致性、可用性与分区容错性。[熟悉]

6. 实现一个简单的服务,例如,一个拼写检查服务。[使用]

7. 解释在给定的服务条件下如何根据服务的开销、可伸缩性和容错性,选择有状态和无状态的设计方案。[熟悉]

8. 描述在一个服务需要逐步被用于更多用户时或者一个服务只是暂时性的有很多客户时,可扩展性所遇到的挑战。[熟悉]

9. 举例说明领导者选举算法这类一致性算法的必要性。[掌握]

PD/云计算

[选修]

知识点:

- 互联网大规模计算。

➢ 任务划分(交叉引用 PD/并行算法、分析、编程)。

➢ 数据访问。

➢ 集群、网格和栅格。

• 云服务

➢ 基础设施即服务。

■ 弹性资源。

■ 平台 API。

➢ 软件即服务。

➢ 安全。

➢ 成本管理。

• 虚拟化(交叉引用 SF/虚拟化和隔离、OS/虚拟机)

➢ 资源共享管理。

➢ 迁移过程。

• 基于云的数据存储。

➢ 弱一致性数据存储共享访问。

➢ 数据同步。

➢ 数据分区。

➢ 分布式文件系统(交叉引用 IM/分布式数据库)。

➢ 复制。

学习成果:

1. 探讨弹性与资源管理在云计算中的重要性。[熟悉]

2. 解释为一组设备提供共享数据公共视图的同步策略。[熟悉]

3. 解释使用虚拟化基础设施的优点和缺点。[熟悉]

4. 部署一个使用云基础设施来计算或管理数据的应用。[使用]

5. 正确地区分客户端应用与资源性应用。[使用]

PD/形式化模型和语义学

[选修]

知识点:

• 进程和消息传递的形式化模型,包括 CSP 和 Pi - 演算等代数系统。

• 并行计算的形式化模型,包括并行随机访问自动机(PRAM)和其他模型,如块同步并行模型(BSP)。

• 计算依赖性的形式化模型。

• (弱)共享内存一致性模型和它们与编程语言规范的关系。

• 算法正确性准则,包括可线性化。

- 算法过程模型,包括非阻塞保障和公平性保障。
- 规范和检查正确性特征的技术,例如,原子性和无数据竞争。

学习成果:

1. 使用形式化模型,例如 Pi-演算来给并发进程建模。[使用]
2. 解释特定形式化并行模型的特征。[熟悉]
3. 形式化地建模共享内存系统,证明其是否满足一致性。[使用]
4. 使用形式化模型显示并行算法的进度保障。[使用]
5. 使用形式化方法证明一个并行算法具有安全性和活性。[使用]
6. 确定一个具体的任务执行是否可以线性化。[使用]

程序设计语言(Programming Languages PL)

程序设计语言是程序员用来精确地描述概念、规划算法和问题求解推理的媒介。在一个计算机科学家的职业生涯中,需要单独或者联合使用多种语言进行工作。软件开发者需要懂得多种语言背后所隐含的程序模型,并且充分了解多语言支持互补方法,做出恰当的设计选择。计算机科学家将经常需要学习新的语言以及程序设计结构,并且理解程序设计语言定义、组合和实现背后所蕴含的原理。有效地使用程序设计语言并且评估他们的缺陷,需要程序设计语言的基础知识、静态分析,以及诸如内存管理等运行时的管理要素。

RL. 程序设计语言(8 个核心 1 级课时,20 个核心 2 级课时)

	RL/核心 1 级课时	核心 2 级课时	是否选修
RL/面向对象程序设计	4	6	否
RL/函数式程序设计	3	4	否
RL/事件驱动和反应性程序设计		2	否
RL/基本类型系统	1	4	否
RL/程序表示		1	否
RL/语言翻译与执行		3	否
RL/语法分析			是
RL/编译语义分析			是
RL/代码生成			是
RL/运行时系统			是
RL/静态分析			是
RL/高级程序构造			是
RL/并发与并行			是
RL/类型系统			是
RL/形式语义			是
RL/语言语用学			是
RL/逻辑式程序设计			是

注:

• 在课程的绪论部分,前三个单元(面向对象程序设计、函数式程序设计、事件驱动和反应性程序设计)中的一个或多个知识点很可能可以与软件开发基础知识领域的一些知识点合并。

• 学习成果中的一些最重要的核心部分与面向对象程序设计、函数式程序设计以及事实上所有的程序设计相关联。这些学习成果将在面向对象程序设计和功能程序设计知识单元重复。我们并非有意多次重复这些内容,因为无法恰当地把他们安排在一个知识单元里。

PL/面向对象程序设计

[4 个核心一级学时,6 个核心二级学时]

知识点:

[核心一级]

• 面向对象设计。

➢ 分解为带有状态与行为的对象

➢ 建模的类结构设计

• 定义类:域、方法和构造。

• 子类、继承,以及方法的重载。

• 动态调用:定义调用方法。

[核心二级]

• 子类型(参见 PL/类型系统)。

➢ 子类型的多态性、类型语言的隐式类型转换。

➢ 行为替代者的概念:像超级类型一样工作的子类型。

➢ 子类型与继承之间的关系。

• 面向对象封装术语。

➢ 类成员的私有性与可见性。

➢ 显示方法签名的接口。

➢ 抽象基类。

• 利用集合类,迭代程序,以及其他通用库组件。

学习成果:

[核心一级]

1. 设计并实现类。[运用]

2. 利用子类设计简单类结构使代码能够为不同子类重用。[运用]

3. 利用动态派遣方法对控制流进行正确分析。[运用]

4. 比较和对照:(1) 过程/函数式程序设计方法(为每个操作定义一个函

数,并且函数体为每个数据变量提供一个实例);(2) 面向对象方法(为每个数据变量定义一个类,并且为每个操作定义一个方法)。通过定义关于矩阵的变量与操作来理解上述两点。[评估]这部分内容也在 PL/函数式程序设计中出现。

[核心二级]

5. 解释面向对象继承(代码共享与重载)与子类型的关系。(子类型在超级类型上下文环境中可用的思想)[熟悉]

6. 利用面向对象封装机制,例如接口与私有成员。[运用]

7. 在多个程序语言中定义并使用迭代子和其他聚合操作,包括那些把函数作为参数的操作,并为每种语言选择最合适的说法。[运用]这部分内容也在 PL/ 函数式程序设计中出现。

PL/函数式程序设计

[3 个核心 1 级学时,4 个核心 2 级学时]

知识点:

[核心一级]

- 无副作用程序设计。

➢ 函数调用无副作用,可以促进复合式推理。

➢ 变量不可变,阻止其他代码对程序数据造成不可预知的修改。

➢ 数据可以自由地改名和复制,而不会导致其他未预期的效果。

- 函数中通过对每个数据的变化情况分别对待的方式来处理结构化数据(如:树)。

➢ 关联的语言构造,如可识别联合类型(Union)和在不同数据上的模式匹配。

➢ 围绕复合数据定义函数,以应用于处理他们的构成要素。

- 一阶函数(获取、返回、存储)。

[核心二级]

- 函数闭包(利用包含词法环境变量的函数)。

➢ 基本含义与定义——运行期间通过获取环境创建闭包。

➢ 规范术语:回调,迭代子参数,通过函数参数的复用代码。

➢ 利用闭包将数据封装在环境当中。

➢ 函数柯里化与部分应用。

- 定义高阶聚合操作,例如,Map,Reduce/Fold 和过滤。

学习成果:

[核心一级]

1. 撰写基本算法,避免指派可变状态或考虑引用平等。[运用]

2. 撰写调用和返回其他函数的可用函数。[运用]

3. 比较和对照(1) 过程/函数式编程方法(为每个操作定义一个函数,并且函数体为每个数据变量提供一个实例);(2) 面向对象方法(为每个数据变量定义一个类,并且为每个操作定义一个方法)。通过定义关于矩阵的变量与操作来理解上述两点。[评估](这部分内容也在 PL/面向对象程序设计中出现。)

[核心二级]

4. 利用函数闭包对一个程序中的变量与词法范围进行正确分析。[运用]

5. 运用函数封装机制例如闭包和模块化接口等。[运用]

6. 在多个程序语言中定义并使用迭代子和其他聚合操作,包括那些把函数作为参数的操作,并为每种语言选择最合适的说法。[运用](这部分内容也在 PL/面向对象程序设计中说明。)

PL/事件驱动与反应性程序设计

[2 个核心二级学时]

知识点:

- 事件和事件句柄。
- 规范使用如 GUIs、移动设备、机器人、服务器等。
- 利用反应性框架。

➢ 定义事件句柄/监听者。

➢ 不在事件句柄 - 写方控制下的主要事件循环。

- 外部产生事件和程序产生事件。
- 模型、视角、控制器的区分。

学习成果:

1. 为反应性系统撰写事件句柄,例如 GUIs。[运用]

2. 解释为什么事件驱动程序设计风格可以自然地运用在定义需要对外部事件进行反应的软件中。[熟悉]

3. 以模型、视角、控制器的形式对交互式系统进行描述。[熟悉]

PL/基本类型系统

[1 个核心一级学时,4 个核心二级学时]

知识点:

[核心一级]

- 类型是一组数值和一组操作。

➢ 原型类型(如数字,布尔数据等)。

➢ 利用其他类型组成的合成类型(如记录、联合、数组、链表、函数、引用

等)。

- 将变量、参数、结果和域与类型相关联。
- 类型安全及使用与类型不一致数据所产生的错误。
- 静态类型的目标与局限性。

➢ 在不运行程序的情况下消除一些类型的错误。

➢ 不确定性意味着静态分析必须采用保守估计的策略来刻画程序行为。

[核心二级]

- 类属类型(参数化多态)。

➢ 定义。

➢ 支持类属库,例如:集合类型。

➢ 比较和对比 ad hoc 多态(重载)与子类型多态。

➢ 比较和对比前期错误与后期错误/避免错误。

➢ 比较和对比在代码开发和维护阶段强制类型不变与在原型开发阶段推迟类型的决定,并允许灵活的编程模式,比如异构的集合类型。

➢ 比较和对比避免代码的错误使用与允许更多代码重用。

➢ 比较和对比检测完整程序与允许不完整程序。

学习成果:

[核心一级]

1. 对于原始类型和合成类型,非正式地描述这个类型的数值。[熟悉]

2. 对于一个拥有静态类型的系统,描述静态禁止的操作,例如,给一个函数或方法传递错误的数据类型。[熟悉]

3. 举例描述利用类型系统检测到的程序错误。[熟悉]

4. 对于多语言编程,分辨动态检查和静态检查得到的程序性质。[运用]

5. 给出在特定语言中不包含类型检查的示例程序,使其在运行时不出现错误。[熟悉]

6. 利用类型和类型错误信息撰写和调试程序。[运用]

[核心二级]

7. 解释为什么类型规则定义的操作集对于一个类型是合法的。[熟悉]

8. 写下类型规则,管理一个特定的合成类型。[运用]

9. 解释为什么不确定性需要类型系统来以保守模式来模拟程序行为。[熟悉]

10. 定义和使用利用类属类型(包括集合类型)的程序片段(例如:函数、类和方法)。[运用]

11. 讨论类属,子类型和重载之间的不同点。[熟悉]

12. 解释静态类型在撰写、维护和调试软件过程中的优点和局限性。[熟悉]

PL/程序表示

[1 个核心二级学时]

知识点:

- 以其他程序作为输入的程序,如:解释器、编译器、类型检查器、文档生成器等。
- 抽象语法分析树:与具体语法相对照。
- 表示代码执行、翻译与传送的数据结构。

学习成果:

1. 解释程序以其他程序作为输入数据的过程。[熟悉]
2. 描述小规模语言的抽象语法分析树。[运用]
3. 描述与以源代码字符串作为输入相比利用程序表示的好处。[熟悉]
4. 撰写一段程序来处理一段代码表示,如:解释器、表达式优化器,或文档生成器。[运用]

PL/语言翻译与执行

[3 个核心二级学时]

知识点:

- 比较和对比解释、编译到本地代码以及编译到可移植的间接表示三种语言翻译。
- 语言翻译流程:语法分析、可选的类型检查、翻译、链接、执行。
  - ➢ 作为本地代码执行或在虚拟机上执行。
  - ➢ 其他选项如动态加载或动态代码生成。
- 构造核心语言的运行表示如对象(方法表)和一阶函数(闭包)。
- 内存的运行态布局:堆栈调用、堆调用、静态数据调用。
  - ➢ 实现循环调用、递归调用、队尾调用。
- 内存管理。
  - ➢ 手动内存管理:分配、释放和堆内存重用。
  - ➢ 自动内存管理:利用可达概念来实现垃圾内存数据的自动回收。

学习成果:

1. 从特定的语言实现中区分语言的定义(比较和对比编译器与解释器、数据对象的运行表示符)。[熟悉]

2. 区分语法与语义和执行的词法分析。[熟悉]

3. 概述语言核心构造,如对象和闭包在底层运行时的表示。[熟悉]

4. 解释程序设计语言如何将内存管理成全局数据、文本、堆栈和堆栈单元并且解释如递归和内存管理等功能如何映射到内存模型中。[熟悉]

5. 识别并解决内存泄露和悬挂指针的解引用。[运用]

6. 讨论内存回收的(包括可达性概念)的好处及局限。[熟悉]

PL/语法分析

[选修]

知识点:

- 利用正则表达式方法实现扫描(词法分析)
- 自顶向下句法分析策略(例如,递归下降,Earley 句法分析,LL)和自底向上句法分析策略(例如,回溯式句法分析,LR);上下文无关文法的作用。
- 从定义规则中生成扫描器和分析器。

学习成果:

1. 利用形式文法来定义语法。[运用]

2. 利用定义工具生成分析器与扫描器。[运用]

3. 识别语法定义中的关键要素:模糊性、关联性、优先性。[熟悉]

PL/编译语义分析

[选修]

知识点:

- 高级程序表示如抽象语义树。
- 作用域的范围和绑定的解析。
- 类型检查。
- 定义规则,如属性语法。

学习成果:

1. 实现上下文相关的源代码级静态分析,如:类型检查或标识符识别来确定他们的绑定。[运用]

2. 利用属性语法来描述语义分析。[运用]

PL/代码生成

[选修]

知识点:

- 过程调用与方法派遣。
- 分别编译:连接。

- 指令选择。
- 指令调度。
- 寄存器分配。
- 窥视孔优化。

学习成果：

1. 辨识自动将源代码转换为汇编或其他低级语言的所有关键步骤。[熟悉]

2. 为现代语言的函数和方法调用生成低级代码。[运用]

3. 讨论为什么分别编译需要正则调用惯例。[熟悉]

4. 讨论为什么分别编译会因为未知调用作用而限制优化的实施。[熟悉]

5. 讨论由简单翻译引入的优化机会和优化策略，例如：指令选择、指令调度、寄存器分配和窥视孔优化。[熟悉]

PL/运行时系统

[选修]

知识点：

- 动态内存管理途径与技术：分配/释放、垃圾回收（标记清除、拷贝、引用计数）、区域。
- 为对象和活动记录进行数据布局。
- 及时编译与动态重新编译。
- 虚拟机的其他通用特征，如类加载、线程和安全性。

学习成果：

1. 利用碎片、地址、内存开销来比较不同内存管理框架的优点。[熟悉]

2. 讨论自动内存管理方法的优点与局限。[熟悉]

3. 解释如何在对象和活动记录运行时表示元数据的使用，例如：类指针、数组长度、返回地址和框架指针。[熟悉]

4. 讨论实时编译与动态重编译的优点、缺点以及难点。[熟悉]

5. 识别现代语言运行系统提供的服务。[熟悉]

PL/静态分析

[选修]

知识点：

- 相关程序表示，如基本块、控制流图、用户定义链和静态单独分配。
- 程序分析的不确定性和后果。
- 对流不敏感的分析，如类型检查、可扩展的指针和别名分析。

- 对流敏感的分析,如前向和后向数据流分析。
- 路径敏感分析,如软件模型检查。
- 定义分析的工具与框架。
- 程序优化的静态分析角色。
- 验证与错误检查中的静态分析。

学习成果:

1. 以概念框架如数据流分析的形式定义有用的静态分析。[运用]
2. 解释为什么非平凡的可靠静态分析只能是近似分析。[熟悉]
3. 讨论为什么一个程序分析是正确的(满足可靠和可停机的性质)。[运用]
4. 区分"可能"和"一定"的程序分析。[熟悉]
5. 解释为什么潜在的别名现象限制了可靠的程序分析和别名现象如何能够提供帮助。[熟悉]
6. 利用静态分析的结果完成程序优化和部分程序改错。[运用]

PL/高级程序结构

[选修]

知识点:

- 延迟求值与非确定流。
- 控制抽象:异常句柄、连续性、元。
- 面向对象抽象:多继承、混合、特征、多方法。
- 元编程:宏、产生式编程、基于模型的开发。
- 模块系统。
- 通过模式匹配(正则表达式)的串操作。
- 动态代码执行("eval" 语句)。
- 语言支持断言检查、不变性质和前置/后置条件。

学习成果:

1. 正确使用各种高级编程结构。[运用]
2. 讨论高级编程构造如何改进程序结构、软件质量和程序员的开发效率。[熟悉]
3. 讨论高级编程结构如何与其他的语言功能的定义和实现相互作用。[熟悉]

PL/并发与并行

[选修]

对并发的支持是编程语言的根本问题,涉及到编程语言设计、语言的实现和

语言理论等丰富的内容。由于其他的知识领域对这部分内容已经有说明,所以这里只侧重补充一些相关知识点。编程语言课程非常适合讲授并发部分的知识内容。

交叉参照 PD/并行和分布式计算、SF/并行性

知识点:

- 构造线程共享的变量和共享内存的同步。
- Actor 模型。
- Futures 并发设计模式。
- 支持数据并行的语言。
- 串行进程之间消息通信的模型。
- 内存一致性模型对编程语言的语义和正确代码生成的影响。

学习成果:

1. 使用多种编程模型撰写正确的并发程序,包括:共享内存、Actors、Futures 和数据并行的原语。[运用]

2. 利用消息传递模型来分析通信协议。[运用]

3. 解释为什么编程语言在数据竞争的情况下无法保证顺序一致性,以及程序员如何应对。[熟悉]

PL/类型系统

[选修]

知识点:

- 复合类型的构造子,例如:点积类型(为集合)、和类型(为联合)、函数类型、量化类型和递归类型。
- 类型检查。
- 类型安全性质:类型保持及演进。
- 类型推理。
- 静态重载。

学习成果:

1. 精确地、组合式地定义类型系统。[运用]

2. 对于不同的基础类型构造子,辨识它们表示的数值和所约束的不变性质。[熟悉]

3. 精确地定义健壮的类型系统所保持的不变性质。[熟悉]

4. 从类型保持和类型演进定理的角度,证明一个简单语言的类型安全性。[运用]

5. 给一个简单语言实现基于同一机制(unification)的类型推理算法。[应用]

6. 解释静态重载和关联解析算法如何影响程序的动态行为。[熟悉]

PL/形式语义

[选修]

知识点:

- 比较和对比语法和语义。
- Lambda 演算。
- 形式语义方法:操作语义、指称语义、公理语义。
- 面向语义的归纳证明法。
- 类型系统的形式化定义和证明(参见 PL/类型系统)。
- 参数化 (参见 PL/类型系统)。
- 使用形式化语义进行系统建模。

学习成果:

1. 给出一个小规模语言的形式语义。[运用]

2. 撰写 lambda 演算程序,并展示其运算生成正常形态的过程。[运用]

3. 讨论语义的不同方法:操作语义、指称语义和公理语义。[熟悉]

4. 使用归纳法证明一个语言所有程序的性质。[运用]

5. 使用归纳法,基于形式化定义的类型系统,证明一个具有良好类型定义的语言程序的性质。[运用]

6. 使用参数化方法在只给定类型的条件下,确定代码的行为。[运用]

7. 使用形式语义(而不是程序设计语言)来建立软件系统的形式模型。[运用]

PL/语言语用学

[选修]

知识点:

- 语言设计原则,如:正交性。
- 执行顺序、优先级和关联性。
- 比较和对比积极执行与延迟执行。
- 定义控制和迭代的构造方法。
- 外部调用和系统函数库。

学习成果:

1. 讨论编程语言设计中的正交化和缺省等概念的作用。[熟悉]

2. 使用简洁和客观的标准来评价语言设计。[运用]

3. 给出这样的样例程序,使其在不同的执行顺序、优先级和关联规则下得到不同的运行结果。[运用]

4. 给出延迟执行的用途,例如,用户定义的抽象控制。[熟悉]

5. 讨论允许外部调用和系统库的必要性,以及给语言实现带来的后果。[熟悉]

PL/逻辑式程序设计

[选修]

知识点:

- 数据结构和算法的字句表示。
- 子句合一。
- 回溯和搜索。
- 截断操作(Cuts)。

学习成果:

1. 使用逻辑式语言来实现一个传统的算法[运用]

2. 使用逻辑式语言中的子句、关系和截断来实现一个隐式搜索算法[运用]

软件开发基础(Software Development Fundamentals SDF)

熟练掌握软件开发的过程是学习计算机专业大多数科学知识的先决条件。为了有效地使用计算机解决问题,学生必须胜任用多种程序设计语言进行阅读及书写程序。然而,除了程序设计技巧之外,他们必须能够设计并分析算法,选择合适的编程范式,并且利用现代的开发方法和测试工具。这一知识领域涵盖了那些与软件开发过程相关的基本概念和技巧。基于此,它为其他基于软件的知识领域,特别是,程序设计语言(PL)、算法和复杂性(AC)、软件工程(SE)提供了基础。

需要特别指出的是,这一知识领域和 CC2001 课程体系规范中规定的程序设计基础的知识领域是明显不同的。CC2001 对应的知识领域仅仅关注计算机科学课程导论中需要的编程技巧,而新的知识领域面向更宽泛的目标。后者关注整个的软件开发过程,明确那些在计算机科学学科第一年需要掌握的概念和技巧。包括算法的设计及简单分析、基本的程序设计概念和数据结构以及基本的软件开发方法和工具。由于其宽泛的目标,软件开发基础知识领域也包括其他面向软件的知识领域所涵盖的基本概念和技巧(如程序设计语言(PL)的编程构造,包含于算法与复杂性(AC)的简单算法分析,来源于软件工程(SE)的基本开发方法)。同样地,所有这些知识领域均包含构建于此处列出的基础概念和技巧之上的更先进的知识。

由于比上一个版本的程序设计基础更宽泛,这一知识领域还使得第一学年的课程组织更加灵活。例如,基本程序设计概念这一单元仅给出所有的程序设计模式共有的那些概念。它期望每位教师自己选择一个或更多的程序设计模式(如面向对象的程序设计、函数式的程序设计、脚本式设计)来给出这些程序设计概念,并且将程序设计语言(PL)知识领域的某一特定模式的内容添加、充实到课程当中。同样地,教师可以选择提前强调形式化分析方法(例如,Big - Oh 复杂性分析、可计算性),或者设计方法学(例如,团队项目、软件生命周期),从而把用于程序设计语言(PL)、算法和复杂性(AC)、软件工程(SE)的学时结合起来。这样这个知识领域的 43 学时的课程素材可以从其他相关知识领域的内容中得到加强,给一年级的学生提供完整和一致的学习体验。

需要指出的是,每一个知识单元分配的学时只是反映了课堂讲授所需要的最少时间。很多软件开发方面的知识点还会在后续的知识领域介绍中得以重复和加深(例如,在处理表数据时采用循环构造子)。另外,要掌握这个知识领域

的概念和技能,只靠课堂学习是不够的,还需要大量的软件开发实践。

SDF. 软件开发基础(43 核心一级学时)

	核心一级学时	核心二级学时	是否选修
SDF/算法与设计	11		否
SDF/程序设计基本概念	10		否
SDF/基本数据结构	12		否
SDF/开发方法	10		否

SDF/算法与设计

[11 个核心一级课时]

这个单元是掌握算法与复杂性分析(AC)核心概念的基础,对基本分析和算法策略单元尤其重要。

知识点:

- 算法的概念和属性。
- 算法效率的比较(例如,操作次数)。
- 问题求解过程中算法的角色。
- 问题求解策略。
- 循环和递归的数学基础。
- 数据结构的循环和递归遍历。
- 分而治之策略。
- 设计的基本概念和原理。
- 抽象。
- 问题分解。
- 封装和信息隐藏。
- 行为和执行分离。

学习成果:

1. 讨论问题求解过程中算法的重要性。[熟悉]
2. 讨论一个问题如何能用多个不同属性的算法求解。[熟悉]
3. 设计算法求解简单的问题。[运用]
4. 使用一种编程语言实现、测试和调试算法用以简单问题求解。[运用]
5. 实现、测试和调试简单的递归方程。[运用]

6. 判定一个问题是否适用于用递归或者循环方法求解。[评估]

7. 利用分而治之思想求解问题。[运用]

8. 运用分解思想将一个问题分解为更小的问题求解。[运用]

9. 分辨多个抽象数据类型的数据构件和行为。[运用]

10. 实现一个数据构件和行为松耦合的一致抽象数据类型。[运用]

11. 分辨一个问题的多种设计和实现方式的优劣。[评估]

SDF/程序设计基本概念

[10 个核心一级课时]

这个单元是掌握程序设计语言核心概念的基础,对面向对象程序设计、过程程序设计以及事件驱动和反应型程序设计内容尤其重要。

知识点:

- 高级语言的基本语义和语法。
- 变量和基本数据类型(例如,数字,字母,布尔型变量)。
- 表达式和赋值语句。
- 简单输入/输出,包括文件输入/输出。
- 条件和循环控制结构。
- 函数和值传递。
- 递归概念。

学习成果:

1. 分析和解释简单程序的行为,包括基本程序构件如变量、表达式、赋值语句、输入/输出,控制结构,函数,值传递和递归。[评估]

2. 辨析基本数据类型的使用。[熟悉]

3. 使用基本数据类型编写程序。[运用]

4. 利用标准条件、递归控制结构和函数修改和扩展简短程序。[运用]

5. 使用以下基本程序设计结构设计、实现、测试和调试一个程序:基本计算、简单输入/输出、标准条件和循环结构、函数定义和值传递。[运用]

6. 编写一个程序,使用文件输入/输出使得多个执行间保持一致。[运用]

7. 选择合适的条件和循环结构完成一个程序设计任务。[评估]

8. 描述递归概念并举例递归的使用。[熟悉]

9. 辨析递归问题的基本模式和通用模式。[评估]

SDF/基本数据结构

[12 个核心一级课时]

这个单元是掌握算法与复杂度(AL)中核心概念的基础,对基础数据结构及

算法、基础自动机的可计算性及复杂度单元尤其重要。

知识点：

- 数组。
- 记录/结构体(异类整合)。
- 字符串和字符串操作。
- 抽象数据类型和实现。

➢ 栈。

➢ 队列。

➢ 优先队列。

➢ 集合。

➢ 映射。

- 引用和别名。
- 链表。
- 选择合适数据结构的策略。

学习目标：

1. 讨论内置数据结构的合理使用。[熟悉]

2. 描述以下数据结构的一般性运用：栈、队列、优先队列、集合、映射。[熟悉]

3. 使用以下数据结构编写程序：数组、记录/结构体、链表、队列、集合、映射。[运用]

4. 从性能的角度比较使用同一数据结构的不同实现。[评估]

5. 描述引用如何能让对象被多种方式访问。[熟悉]

6. 比较动态和静态数据结构实现的各自优劣。[评估]

7. 选择一个合适的数据结构来为给定的问题建模。[评估]

SDF/开发方法

[10 个核心一级课时]

这个单元是掌握软件工程(SE)知识领域核心概念的基础，对软件过程、软件设计和软件演化内容尤其重要。

知识点：

- 程序理解。
- 程序正确性。

➢ 出错类型(意义、逻辑、运行中)。

➢ 规范概念。

- 防错性编程(例如,安全编码和异常处理)。
- 代码检查。
- 测试基础和测试用例产生。
- 代码协定的作用和使用,包括先置和后置条件。
- 单元测试。
  - 简单重构。
  - 现代编程环境。
- 代码搜索。
- 使用库构件和 API 编程。
  - 调试策略。
  - 编写文档和程序风格。

学习目标:

1. 跟踪不同程序段的执行过程并且总结它们的计算过程。[评估]

2. 解释为什么正确创建程序构件对于高质量的软件生产至关重要。[熟悉]

3. 辨析导致程序安全隐患的常见编码错误(例如,缓存溢出、内存泄漏和恶意代码)并且运用策略避免这些错误。[运用]

4. 基于一个给定的检查列表,对程序构件进行单人检查(重点关注常见编码错误)。[运用]

5. 参与一个小团队的代码检查,关注构件的正确性。[运用]

6. 描述如何使用代码协定来规定一个程序构件的行为。[熟悉]

7. 利用过程抽象来重构程序。[运用]

8. 利用不同的策略来测试和调试简单程序。[运用]

9. 利用一种集成开发环境 IDE 和及其工具,如单元测试工具、可视化调试工具等来构造、执行和调试代码。[运用]

10. 利用一种编程语言的内置标准库来构造和调试代码。[运用]

11. 分析另一个程序员的代码在多大程度上满足文档和编程风格标准。[评估]

11. 运用一致的文档和编程风格实现软件的可读性和可维护性。[运用]

软件工程（Software Engineering，SE）

在每一个计算应用领域，专业、品质、进度和成本是生产软件系统的关键。因此，软件工程的各个元素在所有计算领域的软件开发都适用。自从人们第一次意识到软件工程应该是一种规范以来，各种各样的软件工程实践已得到开发和利用。这些不同的实践之间有很多的权衡因素也已确定。在工作中的软件工程师必须针对给定的开发工作选择并运用适当的技术和实践方法以使开发工作价值最大化。要了解如何做到这一点，他们就要学习软件工程的各个元素。

软件工程是这样一种规范，它关注理论、知识和实践的应用，以有效且高效地构建能满足客户和用户需要的、可靠的软件系统。本规范适用于小型、中型和大型系统。它囊括一个软件系统生命周期的所有阶段，包括需求的获取、分析和描述，设计，构建，验证与确认，部署，以及运行和维护。无论系统是大型或小型，遵循传统的计划驱动的开发流程，或是采用敏捷方法，或其他方式，软件工程关注的是以最佳途径建设好的软件系统。

软件工程采用工程化的方法、流程、技术和评估。通过使用工具获得收益，这些工具用于管理软件开发，软件构件的分析和建模，质量的评估和控制，以及规范的保证、软件进化和重用的控制方法。软件工程的工具箱已经发展进化了很多年。例如，带有需要和保证条款以及类不变条件的契约的使用，就是一个很好的、已经越来越普及的做法。软件开发，其可能涉及个人开发者、一个团队或者多个开发团队，需要针对一个给定的开发环境选择最合适的工具、方法和途径。

学生和教师需要理解软件工程方法的专门化带来的影响。例如，专门化的系统包括以下几种。

- 实时系统。
- 客户端 - 服务器系统。
- 分布式系统。
- 并行系统。
- 基于 Web 的系统。
- 高安全性系统。
- 游戏。
- 移动计算。
- 特定领域的软件（如科学计算或商业应用）。

所有这些特殊系统带来的问题，需要在软件工程的各个阶段给予特殊的处理。学生必须意识到软件工程的一般技术和原则与解决特殊系统具体问题所需的技术和原则之间的差异。

专门化带来的一个重要影响是，在进行软件工程应用的教学中，可能需要选择不同的素材，如在不同的过程模型之间、不同的系统建模方法之间或执行任意关键活动时不同技术之间选择。这反映在核心和选修素材的分配上，核心知识点和学习结果关注各种选择之下隐含的基本原理，而做选择时各种可能性的细节内容则被分配给选修素材。

软件工程实践的另一个分支是在以下两种类型之间：即关注那些开发能够正确执行功能的系统的基本需求和那些系统其他方面质量以及平衡这些质量所需要做出的折衷为重点的实践。这种划分也反映在核心和选修素材的分配上，有关开发系统的基本方法的知识点和学习成果分配给核心部分，有关其他方面的质量以及折衷问题分配给选修素材。

一般来说，学生可以通过参与一个项目来更好地学习运用软件工程知识领域中定义的很多素材。这些项目应该要求学生们进行团队工作，通过参与尽可能多的生命周期环节来开发一个软件系统。软件工程很大程度上是致力于使团队成员和利益相关者之间进行有效的沟通。通过采用项目团队形式，项目可以具有充分挑战性，从而要求学生运用有效的软件工程技术，开发和锻炼他们的沟通技巧。同时，因为在学术框架内组织和运行有效的项目是具有挑战性的，所以学习运用软件工程理论和知识的最好方法是在实际项目的环境中进行。大家可能会困惑，对于一些知识单元，按照本文中规定的最少学习可能无法达到相应级别的学习成果。其实，这些成果是通过项目经验来实现的，而项目经验甚至可能来自该知识单元中的后续课程。

此外，有越来越多的证据表明，迭代的方法会使学生更有效地学习运用软件工程原理。在迭代过程中，学生有机会经历一个开发周期，评估他们的工作，然后应用他们在评估中获得的知识，去进行下一个开发周期。敏捷和迭代生命周期模型天然具备这样的机会。

本文中的软件生命周期的术语是基于更早一点的文档，如软件工程知识体（SWEBOK）和 ACM/ IEEE－CS 软件工程 2004 课程指导（SE2004）。虽然有些术语最初是在计划驱动的开发流程的背景下定义的，但在此他们被视为是通用的，因此同样适用于敏捷过程。

注："SDF/开发方法"的知识单元包括了介绍软件工程某些方面的 9 个核心一级学时。这里说明的软件工程知识领域中的知识单元、知识点和核心学时，应

被理解为以先读过“SDF/开发方法”中的材料为假设前提。

SE. 软件工程(6 个核心一级学时;21 个核心二级学时)

	核心一级学时	核心二级学时	是否包括选修
SE/软件过程	2	1	是
SE/软件项目管理		2	是
SE/工具和环境		2	否
SE/需求工程	1	3	是
SE/软件设计	3	5	是
SE/软件构建		2	是
SE/软件验证与确认		4	是
SE/软件演化		2	是
SE/软件可靠性		1	是
SE/形式化方法			是

SE/软件过程

[2 个核心一级学时;1 个核心二级学时]

知识点:

[核心一级]

- 系统级别的考虑,即软件与其目标环境的相互作用(交叉参照 IAS/安全软件工程)。
- 软件过程模型的介绍(如瀑布、增量、敏捷)。

➢ 软件生命周期中的活动。

- 大规模编程对比个人编程。

[核心二级]

- 软件过程模型的评估。

[选修]

- 软件质量的概念。
- 过程改进。
- 软件过程能力成熟度模型。
- 软件过程的度量。

学习成果:

［核心一级］

1. 描述软件如何与各种系统交互及参与，包括信息管理、嵌入式、过程控制和通信系统。［熟悉］

2. 描述几个主要过程模型的相对优势和劣势（如瀑布、迭代以及敏捷）。［熟悉］

3. 描述作为各种过程模型重要组成部分的不同做法。［熟悉］

4. 区分软件开发的各个阶段。［熟悉］

5. 就理解大型代码库、阅读代码、理解构建以及理解变更背景等方面，描述大型编程是如何区别于个人工作。［熟悉］

［核心二级］

6. 解释软件生命周期的概念，并通过一个例子说明其各个阶段，包括各阶段产生的可交付产品。［熟悉］

7. 就常见的几种过程模型对开发特定类别的软件系统所具有的价值进行比较，要充分考虑如需求的稳定性、规模以及非功能性特性等问题。［运用］

［选修］

8. 定义软件质量，并且描述质量保证活动在软件过程中扮演的角色。［熟悉］

9. 描述不同过程改进方法的目的和之间的基本相似性。［熟悉］

10. 比较几种过程改进模型，如 CMM、CMMI、CQI、计划 - 执行 - 检查 - 行动或 ISO9000。［评估］

11. 评估开发工作，并通过参与过程改进（使用一种模型，如 PSP），或参与项目回顾，来给出潜在改变的推荐。［运用］

12. 解释过程成熟度模型在过程改进中扮演的角色。［熟悉］

13. 描述几个评估和控制项目的过程指标。［熟悉］

14. 用项目指标来描述一个项目的当前状态。［运用］

SE/软件项目管理

［2 个核心二级学时］

知识点：

［核心二级］

- 团队参与。

➢ 团队过程，包括任务对应的职责、会议结构以及工作调度。

➢ 软件开发团队中的角色和责任。

➢ 团队冲突的解决。

➢ 虚拟团队的(沟通、感知、结构)伴随风险。

• 工作量估算(个体级别)。

• 风险(交叉参考 IAS/安全软件工程)。

➢ 风险在生命周期中扮演的角色。

➢ 风险类别,包括安全、保险、市场、金融、技术、人员、质量、结构和过程。

[选修]

• 团队管理。

➢ 团队组织和决策。

➢ 角色认定和分配。

➢ 个人和团队绩效考核。

• 项目管理。

➢ 调度与跟踪。

➢ 项目管理工具。

➢ 成本/效益分析。

• 软件度量和估算技术。

• 软件质量保证以及度量所扮演的角色。

• 风险。

➢ 风险识别和管理。

➢ 风险分析与评估。

➢ 风险容忍度(如风险规避、风险中性、风险偏爱)。

➢ 风险规划。

• 全系统处理风险的方法,包括处理与工具相关的危害。

学习成果:

[核心二级]

1. 讨论有助于团队有效运作的一般行为。[熟悉]

2. 建立并遵循一个小组会议的议程。[运用]

3. 确定和论证软件开发团队中的必要角色。[运用]

4. 了解团队冲突的来源、危害以及可能带来的好处。[运用]

5. 在团队设置中应用一种冲突解决策略。[运用]

6. 使用一种特事特办的方法来估算软件开发工作(如时间),并与实际需要的工作量比较。[运用]

7. 举出若干软件风险的例子。[熟悉]

8. 描述风险对一个软件开发生命周期的影响。[熟悉]

9. 描述软件系统中不同类别的风险。[熟悉]

[选修]

10. 通过参与团队项目，演示团队建设和团队管理的核心要素。[运用]

11. 描述过程模型的选择如何影响团队的组织结构和决策过程。[熟悉]

12. 创建一个团队，确定合适的角色并将角色分配给团队成员。[运用]

13. 评估并向团队和个人反馈其在团队环境中的绩效。[运用]

14. 使用特定的软件过程，描述项目需要进行规划和监控的方面（如估算规模和工作量、进度、资源分配、配置控制、变更管理以及项目的风险识别和管理等）。[熟悉]

15. 使用合适的项目度量跟踪项目某个阶段的进展情况。[运用]

16. 比较简单的软件规模和成本估算技术。[运用]

17. 使用项目管理工具，辅助软件开发项目中的任务分配和跟踪。[运用]

18. 描述风险容忍度对软件开发过程的影响。[评估]

19. 识别风险，并描述管理风险的方法（规避、接受、转移、减缓），且刻画每种方法的优势和不足。[熟悉]

20. 解释风险在软件开发过程中如何影响决策。[运用]

21. 识别软件系统的安全风险。[运用]

22. 在某种特定情况下，展示一种系统化的方法，以完成识别危害和风险的任务。[运用]

23. 在各种简单（包括安全状况）的场景中，应用风险管理的基本原理。[运用]

24. 针对某种风险缓解方法进行成本/效益分析。[运用]

25. 识别并分析一些源于除软件之外的其他方面、全系统的风险。[运用]

SE/工具和环境

[2 个核心二级学时]

知识点：

- 软件配置管理和版本控制。
- 发布管理。
- 需求分析和设计建模工具。
- 测试工具，包括静态和动态分析工具。
- 实现程序构建过程的部分自动化（例如，自动生成）的编程环境。

➢ 持续集成。

- 工具集成的概念和机制。

学习成果：

1. 描述集中式和分布式软件配置管理之间的差异。[熟悉]

2. 描述版本控制如何用于帮助实现软件版本的管理。[熟悉]

3. 识别配置项，并在一个小型的基于团队的项目中使用源代码控制工具。[运用]

4. 描述现有的静态和动态测试工具如何被集成到软件开发环境中。[熟悉]

5. 描述在为一个特定的软件系统选择一套工具(包括需求跟踪、设计建模、实现、自动化构建以及测试)时要考虑的重要的问题。[熟悉]

6. 演示使用软件工具支持中等规模的软件产品开发的能力。[运用]

SE/需求工程

[1 个核心一级学时；3 个核心二级学时]

需求工程的目的，是为了建立关于需要、优先级以及软件系统相关限制的共识。许多软件的失败源于对要开发软件的需求理解不完全，或对那些需求的管理不完善。

需求规格说明形式的规范性从完全非正式(如口语)到严格的数学形式(如用形式化的规范语言，如 Z 语言，或用一阶逻辑书写)都有。在实践中，成功的软件工程工作使用需求规格说明书，以减少不确定性，并提高开发团队对预期软件愿景理解的一致性和完整性。计划驱动的方法倾向于产生带编号需求的正式文件。敏捷方法则倾向于不那么正式的规格说明，包括用户故事、用例和测试用例。

知识点：

[核心一级]

- 使用用例或用户故事等来描述功能性需求。
- 需求特性，包括一致性、有效性、完整性和可行性。

[核心二级]

- 软件需求获取。
- 使用类图或实体 - 关系图等来描述系统数据。
- 非功能性需求及其与软件质量(交叉参照 IAS/安全软件工程)的关系。
- 评估和需求规格说明书的使用。

[选修]

- 需求分析建模技术。
- 有关软件/系统行为的确定性/不确定性因素的可接受性。

- 原型法。
- 形式化需求规格说明的基本概念。
- 需求规格说明。
- 需求确认。
- 需求跟踪。

学习成果：

［核心一级］

1. 针对一个用例或者系统需要的某种行为的类似描述，列出其重要组成部分。［熟悉］

2. 描述需求工程的过程是如何支持行为需求的获取和确认。［熟悉］

3. 解释一个简单的软件系统给定的需求模型。［熟悉］

［核心二级］

4. 描述需求获取面临的基本挑战以及常用的技术。［熟悉］

5. 列出一个数据模型（如类图或 E－R 图）的重要组成部分。［熟悉］

6. 在一个软件系统给定的需求规格说明书中，识别功能性和非功能性需求。［运用］

7. 对一系列的软件需求进行审查，以根据良好需求的特性确定需求的质量。［运用］

［选修］

8. 应用需求获取和分析的关键要素和常用方法，为一个中型软件系统制作一组软件需求。［运用］

9. 把计划驱动和敏捷方法比作需求规格说明和验证，并分别描述各自相关的利益和风险。［熟悉］

10. 使用一种常用的、非形式化方法对一个中等大小的软件系统进行建模和需求确定。［运用］

11. 将用一种形式化规格描述语言写成的软件需求规格说明（如一个软件组件合同）翻译成自然语言。［运用］

12. 创建一个软件系统的原型，以降低需求中的风险。［运用］

13. 区分向前和向后跟踪，并解释他们在需求验证过程中所扮演的角色。［熟悉］

SE/软件设计

［3 个核心一级学时；5 个核心二级学时］

知识点：

[核心一级]

- 系统设计原则:抽象层次(架构设计和详细设计)、关注点分离、信息隐藏、耦合和内聚、标准结构的重用。
- 设计范例,如结构化设计(自顶向下的功能分解)、面向对象的分析和设计、事件驱动的设计、组件级设计、数据结构为中心、面向方面、面向功能、面向服务。
- 软件设计的结构模型和行为模型。
- 设计模式。

[核心二级]

- 需求和设计之间的关系:模型转化、契约设计、不变式。
- 软件架构的概念和标准体系结构(如客户端 - 服务器、n - 层、转换为中心、管道 - 过滤器)。
- 使用设计模式重构设计。
- 设计中对组件的使用:组件的选择、设计、组件的改造及组装、组件与模式、组件与对象(例如,使用标准控件集建立一个图形用户界面)。

[选修]

- 内部设计质量及其模型:效率和性能、冗余和容错、需求可追溯性。
- 外部设计质量及其模型:功能性、可靠性、性能和效率、可用性、可维护性、可移植性。
- 设计质量的度量与分析。
- 质量不同方面之间的权衡取舍。
- 应用程序框架。
- 中间件:中间件中的面向对象范例,对象请求代理和汇集,事务处理监控器,工作流系统。
- 安全设计和编码原则(交叉参照 IAS/安全设计原理)。

➢ 最小特权原则。

➢ 默认故障安全原则。

➢ 心理可接受性原则。

学习成果:

[核心一级]

1. 说出设计原则,包括关注点分离、信息隐藏、耦合和内聚以及封装。[熟悉]

2. 使用一种设计范例来设计一个简单的软件系统,并解释系统设计原则是

如何被应用其中。[运用]

3. 用最适合的设计范例来建立一个简单的软件系统的设计模型。[运用]

4. 就某个单一的设计范例，描述一个或多个可用于设计简单软件系统的设计模式。[熟悉]

[核心二级]

5. 对适合于某一给定脚本的简单系统，讨论并选择一种合适的设计范例。[运用]

6. 根据需求规格说明书，为软件产品的结构和行为建立适当的模型。[运用]

7. 用适当的模型解释软件产品的需求和设计之间的关系。[评估]

8. 对于单一设计范例框架内一个简单的软件系统的设计，描述该系统的软件架构。[熟悉]

9. 给定一个高层次的设计，通过鉴别常用的软件架构，如 3 - 层、管道 - 过滤器以及客户端 - 服务器，来识别该软件架构。[熟悉]

10. 调查软件架构的选择对一个简单系统设计的影响。[评估]

11. 将模式的简单例子应用于一个软件设计。[运用]

12. 描述重构的一种形式，并讨论其可能适用于何时。[熟悉]

13. 选择适当的组件，用于软件产品的设计中。[运用]

14. 解释合适的组件可能需要做哪些适应性改变，以用于软件产品的设计中。[熟悉]

15. 为一个典型的小型软件组件设计一种契约，以用于一个给定的系统。[运用]

[选修]

16. 讨论并选择合适的软件架构，应用于适合某一给定脚本的简单系统。[运用]

17. 应用设计软件组件的内部和外部质量的模型，来达到各种冲突的质量因素之间一种可接受的折衷。[运用]

18. 从某种重要的内部质量属性的角度，分析一种软件设计。[评估]

19. 从某种重要的外部质量属性的角度，分析一种软件设计。[评估]

20. 解释对象在中间件系统中扮演的角色及其与组件之间的关系。[熟悉]

21. 将面向组件的方法应用于某一范围内软件的设计，如将组件用于并发和事务、用于可靠的通信服务、用于数据库交互，包括远程查询和数据库管理或用于安全通信和访问。[运用]

22. 重构一个现有的软件实现,以提高其设计的某些方面。[运用]

23. 陈述并应用最小特权原则和默认故障安全原则。[熟悉]

SE/软件构建

[2 个核心二级学时]

知识点:

[核心二级]

- 编码实践:技术、套路/模式、构建优质程序的机制(参见 IAS/防御性编程、SDF/开发方法)。

➢ 防御性编码实践。

➢ 安全编码实践。

➢ 使用异常处理机制,使程序更加健壮、容错。

- 编码标准。
- 集成策略。
- 开发环境:“绿场”对比现有代码库。

➢ 变更影响分析。

➢ 变更实现。

[选修]

- 程序中潜在的安全问题。

➢ 缓冲区和其他类型的溢出。

➢ 竞态条件。

➢ 不正确的初始化,包括特权的选择。

➢ 检查输入。

➢ 假设成功和正确性。

➢ 验证的假设。

学习成果:

[核心二级]

1. 描述技术、编码套路和实现设计的机制,以达到所需的性能,如可靠性、效率以及健壮性。[熟悉]

2. 使用异常处理机制构建健壮的代码。[运用]

3. 描述安全编码和防御性编码实践。[熟悉]

4. 在一个小型软件项目中选择并使用一种定义的编码标准。[运用]

5. 对照比较各种整合策略,包括自顶向下、自底向上以及三明治集成。[熟悉]

6. 描述对某特定项目开发的代码库进行分析和实现变更的过程。[熟悉]

7. 描述对某大型现有代码库进行分析和实现变更的过程。[熟悉]

[选修]

8. 重写一个简单的程序，以消除常见的漏洞，如缓冲区溢出、整数溢出以及竞态条件。[运用]

9. 写一个软件组件，执行一些非平凡的任务，且可从输入和运行时的错误中恢复。[运用]

SE/软件验证与确认

[4 个核心二级学时]

知识点：

[核心二级]

- 验证与确认的概念。
- 检查、审查、审计。
- 测试类型，包括人机界面、可用性、可靠性、安全性、与规格说明一致性（交叉参照 IAS/安全软件工程）。
- 测试基础（交叉参照 SDF/开发方法）。

➢ 单元、集成、验证以及系统测试。

➢ 测试计划的创建和测试用例生成。

➢ 黑盒和白盒测试技术。

➢ 回归测试和测试自动化。

- 缺陷跟踪。
- 测试在特定领域的局限性，如并行系统或安全攸关系统。

[选修]

- 验证的静态方法和动态方法。
- 测试驱动的开发。
- 计划确认，确认文档。
- 面向对象的测试，系统测试。
- 非代码资料（文档、帮助文件、培训资料）的验证和确认。
- 故障记录、故障跟踪和此类活动的技术支持。
- 故障估算和测试终止，包括缺陷播种。

学习成果：

[核心二级]

1. 区分程序的确认与验证。[熟悉]

2. 描述工具在软件确认中所扮演的角色。[熟悉]

3. 作为团队活动的一部分,承接一个中等规模代码段的检查。[运用]

4. 描述并区分不同类型和层次的测试(单元、集成、系统以及验收)。[熟悉]

5. 描述识别用于集成、回归和系统测试的重要测试用例的技术。[熟悉]

6. 为一个中等规模的代码段创建并归档一组测试。[运用]

7. 描述如何选择好的回归测试以及如何自动化执行。[熟悉]

8. 在一个小规模软件项目中使用一种缺陷跟踪工具来管理软件缺陷。[运用]

9. 讨论测试在某个特定领域的局限性。[熟悉]

[选修]

10. 为一个中等规模的代码段评估一个测试套件。[运用]

11. 比较静态和动态的验证方法。[熟悉]

12. 指出测试驱动开发方法的基本原则,并解释自动化测试在这些方法中扮演的角色。[熟悉]

13. 讨论涉及面向对象软件的测试问题。[运用]

14. 描述非代码资料的验证和确认技术。[熟悉]

15. 描述故障估算方法。[熟悉]

16. 基于故障密度和故障播种,估算一个小型软件应用中故障的数量。[运用]

17. 针对一个小型或中型的软件项目进行软件源代码检查或审查。[运用]

SE/软件演化

[2 个核心二级学时]

知识点:

- 在大型的、预先存在的代码库背景下进行软件开发。
  - 软件变更。
  - 关注及关注的位置。
  - 重构。
- 软件演化。
- 可维护软件的特性。
- 系统再造工程。
- 软件重用。
  - 代码段。

➢ 库和框架。

➢ 组件。

➢ 产品线。

学习成果：

1. 指出与软件演化相关的主要问题，并解释它们对软件生命周期的影响。[熟悉]

2. 估算变更请求对一个中等规模现存产品的影响。[运用]

3. 在修改软件组件的过程中使用重构。[运用]

4. 讨论在不断变化的环境中演化的系统所面临的挑战。[熟悉]

5. 概述回归测试的过程及其在发布管理中扮演的角色。[熟悉]

6. 讨论不同类型的软件重用的优点和缺点。[熟悉]

SE/软件可靠性

[1 个核心二级学时]

知识点：

[核心二级]

- 软件可靠性工程的概念。
- 软件可靠性、系统可靠性和失败行为（交叉参照 SF/冗余下的可靠性）。
- 故障生命周期的概念和技术。

[选修]

- 软件可靠性模型。
- 软件容错技术和模式。
- 软件可靠性工程实践。
- 基于度量的软件可靠性分析。

学习成果：

[核心二级]

1. 解释在达到非常高水平可靠性的过程中所存在的问题。[熟悉]

2. 描述软件可靠性如何有助于系统的可靠性。[熟悉]

3. 列出可以应用于软件生命周期各个阶段的尽可能减少故障的方法。[熟悉]

[选修]

4. 比较三种不同的可靠性建模方法的特点。[熟悉]

5. 展示对软件系统应用多种方法开发可靠性估算的能力。[运用]

6. 指出各种可使软件架构实现的方法，这种架构应达到某种指定级别的可

靠性。[运用]

7. 指出在一个中型应用程序中应用冗余实现容错的各种途径。[运用]

SE/形式化方法

[选修]

下面列出的知识点对离散结构(DS)知识领域的核心素材有很强的依赖性,特别是知识单元 DS/函数关系及集合、DS/基本逻辑以及 DS/证明技巧。

知识点:

- 形式化的规格说明与分析技术在软件开发周期中的作用。
- 程序断言语言和分析方法(包括编写并分析前置和后置条件的语言,如 OCL、JML)。
- 软件建模与分析的形式化方法。
  - ➢ 模型检测器。
  - ➢ 模型查找器。
- 支持形式化方法的工具。

学习成果:

1. 描述形式化的规格说明与分析技术在复杂软件开发中所扮演的角色,并像比较测试的验证和确认技术一样比较它们的用法。[熟悉]

2. 将形式化的规格说明与分析技术用于软件设计和低复杂度的程序。[运用]

3. 解释使用形式化规格说明语言的潜在好处和缺点。[熟悉]

4. 为从简单到复杂的各种行为创建并评估程序断言。[运用]

5. 使用一种常见的形式化规格说明语言,为一个简单的软件系统制定规格说明,并从规格说明中导出测试用例。[运用]

系统基础(Systems Fundamentals, SF)

底层硬件和软件基础设施以及在其上构建的应用程序被统称为“计算机系统”。计算机系统横跨操作系统、并行和分布式系统、通信网络以及计算机体系结构等子学科。传统上,这些领域是通过独立的课程以非集成的方式讲授的。然而,这些子学科在各自的核心内容中越来越多地共享一些重要的共同基本概念。这些概念包括计算范例、并行、跨层通信、状态和状态转移、资源分配与调度等等。系统基础知识领域的设计,是从集成的视角以尽管简单但统一的风格去看待这些基本概念,为不同的特殊机制及适合于特定领域的政策提供共同基础。

SF. 系统基础 [18 个核心一级学时;9 个核心二级学时]

	核心一级学时	核心二级学时	是否包括选修
SF/计算范式	3		否
SF/跨层通信	3		否
SF/状态与状态机	6		否
SF/并行性	3		否
SF/评估技术	3		否
SF/资源分配与调度技术		2	否
SF/临近技术		3	否
SF/虚拟化与隔离		2	否
SF/冗余下的可靠性		2	否
SF/定量评估			是

SF/计算范式

[3 个核心一级学时]

这里提出的观点是跨层系统从硬件构建模块到应用程序组件的多种表示法,以及每种表示中可用的并行性,交叉参照 PD/并行基础。

知识点:

- 一台计算机的基本构建模块和组件(门、触发器、寄存器、互连,数据通路 + 控制 + 存储器)。
- 硬件计算范例:基本的逻辑构建模块,逻辑表达式、最小化,与或式。

- 应用层的顺序处理,单线程。
- 简单应用层的并行处理:请求级别(Web 服务/客户端 - 服务器/分布式),单服务器单线程、多服务器多线程。
- 流水线的基本概念,重叠的处理阶段。
- 缩放的基本概念:速度更快还是处理更大规模的问题。

学习成果:

1. 列出常见的计算组织的模式。[熟悉]

2. 描述计算机的基本构建块,以及它们在计算机体系结构的历史发展中所扮演的角色。[熟悉]

3. 以现实生活中的例子(如烹饪食谱、多取款机线路以及夫妇的美食购物)为由,说出单线程与多线程的区别、单台服务器与多服务器模型之间的区别。[熟悉]

4. 说出强缩放与弱缩放的区别,即对比问题的规模对性能的影响、与通过资源的规模缩放来解决问题。以简单的、现实生活中的例子为由。[熟悉]

5. 使用逻辑设计的基本构建模块来设计一个简单的逻辑电路。[运用]

6. 使用捕捉、合成以及仿真工具来评估一个逻辑设计。[运用]

7. 编写一个简单的顺序执行的程序以及该程序的简单并行版本。[运用]

8. 针对不同的问题规模,评估一个程序简单顺序和并行版本的性能,并且能够描述所取得的速度提升。[评估]

SF/跨层通信

交叉参照 NC/引言、OS/操作系统原理。

[3 个核心一级学时]

知识点:

- 编程抽象、接口、库的使用。
- 应用程序和操作系统服务之间的区别,远程过程调用。
- 应用程序 - 虚拟机的交互。
- 可靠性。

学习成果:

1. 描述计算系统是如何在分离关注点、定义良好的接口、对高层隐藏低层细节的基础上一层层搭建起来的。[熟悉]

2. 描述硬件、虚拟机、操作系统以及应用程序是如何成为解释/处理的附加层的。[熟悉]

3. 描述错误是如何被检测、回信号,并通过这些层进行处理的机制。[熟

悉]

4. 构建一个简单程序,使用分层、错误检测与恢复以及跨层错误状态反射等方法??。[运用]

5. 使用程序跟踪、单步执行以及调试工具在一个分层的程序中找错误。[运用]

SF/状态与状态机

[6 个核心一级学时]

交叉参照 AL/基础自动机的可计算性及复杂度、OS/状态与状态图、NC/协议。

知识点:

- 数字与模拟/离散与连续系统。
- 简单的逻辑门、逻辑表达式、布尔逻辑化简。
- 时钟、状态、测序。
- 组合逻辑、时序逻辑、寄存器、存储器。
- 计算机和网络协议作为状态机的例子。

学习成果:

1. 将计算描述为一个系统,其特征可以用从一个唯一的配置(状态)转换到另一个(状态)的一组已知的配置来刻画。[熟悉]

2. 描述两种系统之间的区别,一种是输出只是输入的函数(组合的),另一种是带存储/历史的(顺序的)。[熟悉]

3. 将计算机描述为一个解释机器指令的状态机。[熟悉]

4. 解释一个程序或网络协议如何也可以被表示为状态机,并且对于相同计算的不同表示也可以存在。[熟悉]

5. 为简单问题陈述的解决方案(如红绿灯测序、模式识别)制定状态机的描述。[运用]

6. 从状态机的表示导出状态机的时序行为。[评估]

SF/并行性(Parallelism)

[3 个核心一级学时]

参见 PD/并行基础。

知识点:

- 顺序与并行处理。
- 并行编程与并发编程。
- 请求并行与任务并行。

- 客户端 - 服务器/Web 服务、线程(分叉 - 汇合)、流水线。
- 多核架构和同步的硬件支持。

学习成果:

1. 对于一个给定的程序,区分其顺序和并行的执行及其对性能的影响。[熟悉]

2. 在执行时间轴上展示并行的事件和操作可以同时进行(即在相同的时间)。如果这可以被利用,解释工作如何能在更短的运行时间内进行。[熟悉]

3. 解释并行的其他用途,例如,用于执行的可靠性或冗余。[熟悉]

4. 定义指令并行、数据并行、线程并行/多任务、任务/请求并行等概念之间的区别。[熟悉]

5. 写出一个以上的并行程序(例如,在一个以上的并行编程范例中的一个简单的并行程序;通过同步原语管理共享资源的简单的并行程序;通过任务并行执行对分区数据同时操作的简单的并行程序(例如,并行搜索词);通过消息传递执行渐次流水线处理的一个简单的并行程序)。[运用]

6. 使用性能工具,就问题规模和资源数量两方面来度量使用并行程序所获得的速度提升。[评估]

SF/评估技术

[3 个核心一级学时]

交叉参照 PD/并行基础。

知识点:

- 性能特性指标。
- 工作量和有代表性的基准以及性能特性指标的收集和分析方法。
- CPI(每条指令的周期)方程作为理解指令集设计、处理器流水线以及内存系统组织中的权衡的工具。
- Amdahl 定律:不能加快的那部分计算限制了可以加快的那部分的效果。

学习成果:

1. 解释系统架构的组件是如何为改善其性能做出贡献。[熟悉]

2. 描述 Amdahl 定律,并讨论其局限性。[熟悉]

3. 设计并实施一个面向性能的实验。[运用]

4. 使用软件工具来刻画和度量程序的性能。[评估]

SF/资源分配与调度技术

[2 个核心二级学时]

知识点:

- 资源的种类(如处理器份额、内存、磁盘、网络带宽)。
- 调度的种类(如先到先得、优先级)。
- 公平调度的优势,抢占式调度。

学习成果:

1. 定义有限的计算机资源(处理器份额、内存、存储和网络带宽)是如何通过其精心分配到现有实体而进行管理的。[熟悉]

2. 描述将资源分配给竞争实体的调度算法,以及评估这些算法的特性指标,如公平性。[熟悉]

3. 实现简单的调度算法。[运用]

4. 使用另外种类调度实现的特性指标。[评估]

SF/临近技术

[3 个核心二级学时]

交叉参照 AR/内存管理、OS/虚拟内存。

知识点:

- 光速与计算机(每纳秒一英尺对比 1 GHz 的时钟)。
- 计算机系统中的延迟:内存、磁盘延迟、网络存储器的对比。
- 缓存及时空局部性对处理器和系统的性能的影响。
- 数据库、操作系统、分布式系统以及计算机体系结构中的缓存与缓存一致性。
- 介绍处理器内存层次结构和平均内存访问时间的公式。

学习成果:

1. 解释局部性对决定性能的重要性。[熟悉]

2. 描述为什么在空间上接近的东西需要较少的访问时间。[熟悉]

3. 计算平均内存访问时间,并且就容量、错失/命中率以及访问时间来描述内存层次结构性能的权衡。[评估]

SF/虚拟化与隔离

[2 个核心二级学时]

知识点:

- 保护和可预测性能的基本原理。
- 通过用于管理物理内存资源的虚拟内存显示间接的层级。
- 虚拟内存和虚拟机的实现方法。

学习成果:

1. 解释为什么隔离和保护个体程序的执行和共享公共基础资源的环境是

很重要的。[熟悉]

2. 描述间接的概念如何可以产生拥有一台专享机器及其资源的错觉,即使其实际上是在多个程序和环境之间共享。[熟悉]

3. 度量在不同的虚拟机中运行的两个应用实例的性能,并确定性能隔离的效果。[评估]

SF/冗余下的可靠性

[2 个核心二级学时]

知识点:

- 缺陷和故障的区别。
- 检查和重试产生的冗余。
- 冗余编码产生的冗余(纠错码、CRC、FEC)。
- 复制/镜像/副本。
- 其他达到容错性和可用性的方法。

学习成果:

1. 解释程序错误、系统错误以及硬件故障(如坏的内存)和异常(如企图除以零)的区别。[熟悉]

2. 说出检测、处理以及故障恢复之间的区别,并说出实现方法。[熟悉]

3. 描述纠错码在为内存、存储器以及网络提供错误检查和纠正技术中的作用。[熟悉]

4. 应用简单的算法,以数据校正为目的来利用冗余信息。[运用]

5. 就其数据开销、实现的复杂性以及编码、检测和纠正错误的相对执行时间,比较不同的错误检测和纠正的方法。[评估]

SF/定量评估

[选修]

知识点:

- 指导定量评价的分析工具。
- 规模分析的数量级(大 O 符号)。
- 系统的慢速和快速路径分析。
- 影响性能的事件(如指令停顿、缓存未命中、页面错误)。
- 了解分层系统、工作负载和平台、其对性能的影响以及对评估带来的挑战。
- 微基准测试的陷阱。

学习成果:

1. 解释某给定的系统性能指标图如何有用。[熟悉]

2. 解释基准作为系统性能度量的不足之处。[熟悉]

3. 使用极限的研究或简单的计算,产生在特定情况下给定的性能指标数量级的估算。[运用]

4. 实施一个分层系统的性能实验,以确定某个系统参数对系统性能图的影响。[评估]

社会问题与专业实践(Social Issues and Professional Practice,SP)

尽管计算类课程是以技术问题为中心的,但技术问题并不构成该领域教育计划的全部。学生还必须接触到更大的计算的社会环境,以培养对相关的社会、伦理、法律和专业问题的理解。这种对非技术问题的研究纳入 ACM 课程的需要,在 1991 年被正式承认。以下摘自参考文献[2]。

本科生也需要了解计算学科内在的基本的文化、社会、法律以及伦理问题。他们应该了解本学科过去、现在在哪里以及将来往何处去。他们也应该了解在这个过程中他们各自的角色,同时也会将哲学问题、技术问题以及审美价值作为学科发展的重要组成部分来欣赏。

学生还需要培养对计算的社会影响提出严肃问题的能力,并能够评估对这些问题的建议性的答复。未来从业者必须能够预见将某一产品引入某种给定的环境所带来的影响。该产品将增强还是降低生活质量?对个人、团体以及机构将产生什么影响?

最后,学生们需要意识到软件和硬件厂商及用户的基本的法律权利,也需要欣赏作为这些权利基础的道德价值观。未来从业者必须了解他们将要承担的责任以及失败可能导致的后果。他们必须了解自己的局限性,同时也了解他们工具的局限性。所有从业人员必须在自己选择的专业以及在作为一个整体的计算机学科中,长期投入保持与时俱进。

技术的进步不断显著影响着我们的生活和工作方式,社会问题和专业实践的重要性也与日俱增;新的基于计算机的产品和领域每年都带来前所未有的挑战性的问题。学生必须有意识地带着对发现和解决这些问题的关注进入职场和学术界。

计算机科学教育工作者可以选择在独立的课程中传授这一核心及选修内容,或将其融入传统的技术性和理论性的课程中,或者作为毕业设计和专业实践课程中的特殊单元。这部分需要熟悉的内容可以通过一门必修课和其他课程中若干短小模块共同组合来更好地覆盖。另一方面,一些被列为核心一级的单元(特别是社会背景、分析工具、职业道德以及知识产权)不容易被其他传统课程覆盖。如果没有一门独立的课程,很难恰当地覆盖这些知识点。另一方面,如果道德和社会因素只包括在独立的课程中,而不是“在环境下”,这将强化一种错误概念,即这些其他相关问题对技术流程不起作用。由于这种广泛相关性,在若

干传统的课程、技术主题的背景下,包括对伦理、法律、社会和职业因素进行分析的案例研究模块是很重要的。

如软件工程、数据库、计算机网络、信息保障与安全以及计算导论等方面的课程提供了分析伦理问题明显的背景。然而,课程设置中的几乎任何课程都可以开发出与道德相关的模块。如果只设一门独立的课程,也就显然违反了本建议的精神。要跟计算机从业者们讲他们有责任通过道德和技术手段积极解决这些问题,那么过一遍这方面所有的问题是有必要的。在任何课堂上讨论的道德问题应该与该课堂的主题内容直接相关并且是从中自然产生的。例如在数据库课程中讨论数据聚合或数据挖掘,又或在软件工程课程中讨论在对客户的义务和对用户及其他可能被他们的工作所影响的人的义务之间可能产生的冲突。围绕应用布置的编程作业,如在眼科手术过程中控制激光的移动,可帮助讲到计算对专业、道德和社会的影响。那些不熟悉应用伦理学的内容和/或教学的计算机教师,应敦促其从 ACM、IEEE - CS、SIGCAS(计算机与社会学的特殊兴趣团体)以及其他组织获取相当多的资源优势。

应该指出的是,伦理分析的应用贯穿计算机的"社会与专业"知识领域的每一个小节。ACM 的道德与职业操守守则(http://www.acm.org/about/code - of - ethics)为我们专业工作的实施基础提供了方针。"一般性道德责任"提供了关于我们对个人责任、职业操守以及我们的领导角色的承诺的一种理解。

SP. 社会问题与专业实践 [11 个核心一级学时;5 个核心二级学时]

	核心一级学时	核心二级学时	是否包括选修
SP/社会环境	1	2	否
SP/分析工具	2		否
SP/职业道德	2	2	否
SP/知识产权	2		是
SP/隐私和公民自由	2		是
SP/专业交流	1	1	是
SP/可持续性	1		是
SP/历史			是
SP/计算经济性			是
SP/安全政策、法律和计算机犯罪			是

SP/社会环境

[1 个核心一级学时;2 个核心二级学时]

在过去的 75 年里,计算机和互联网或许比任何其他技术都更多地改变了社会,显著增加了人类生产力;爆炸性的增长了人们对于新闻、娱乐以及通信等的选择;以及在科学和工程界中几乎每一个分支的根本性突破。“社会环境”为所有其他 SP 知识单元,尤其是“职业道德”提供了基础。另请交叉参照知识领域“人机交互”(HCI)和“网络与通信”(NC)。

知识点:

[核心一级]

- 计算在网络世界中的社会寓意(交叉参照 HCI/基础/社会模式、IAS/安全基本概念/社会问题)。
- 社交媒体上的个人主义、集体主义和文化的影响。

[核心二级]

- 互联网的增长和控制(交叉参照 NC/引言/互联网的组织)。
- 通常被提到的数字鸿沟,在获取数字技术资源中以及由于性别、阶级、种族、地域和/或欠发达国家而产生的后果中的差异。
- 可访问性问题,包括法律要求。
- 情境感知计算(交叉参考 HCI/非鼠标接口设计/普适与情境感知)。

学习成果:

[核心一级]

1. 描述计算机技术(网络、移动计算、云计算)在个人层面上从正负两方面对社会互动模式的改变。[熟悉]

2. 指出开发者假设以及嵌入在硬件和软件设计中的价值,特别是当它们涉及针对不同人群的可用性,包括弱势群体和残疾人。[熟悉]

3. 对给定的设计和实施解读其社会情境。[熟悉]

4. 使用经验数据评估给定的设计和实施的效能。[评估]

5. 总结社交媒体对个人主义与集体主义以及文化的影响。[运用]

[核心二级]

6. 讨论上网如何成为生活在压抑状态下的人们一种解放的力量;解释网络限制如何成为政治和社会压迫的工具。[熟悉]

7. 分析在实现民主(如提供社会服务、电子投票)中依赖计算的利弊。[评估]

8. 描述不同人群中的弱势群体对计算机行业(如企业文化、产品的多样性)

的影响。[熟悉]

9. 解释情境感知对普适计算系统的意义。[熟悉]

SP/分析工具

[2 个核心一级学时]

伦理学理论和原理是伦理分析的基础,因为它们提供了观点,从这些观点出发,沿此路线可以获得指导以得出判断。每个理论强调的观点不同,如预测后果并遵循其对他人应负的责任,以得到一项以道德为指导的决定。但是,为了使一种道德理论成为有用的,这个理论必须指向一个共同的目标集合。道德原则是每个理论都努力要实现以获得成功的共同目标。这些目标包括慈善、最小伤害、尊重自主以及正义。

知识点:

- 伦理的论证。
- 伦理学理论与决策。
- 道德假设和价值观。

学习成果:

1. 评估在特定情况下的利益相关者的立场。[评估]
2. 在一场争论中分析基本的逻辑谬误。[评估]
3. 分析一场争论以识别前提和结论。[评估]
4. 在道德争论中演示实例和比喻的运用。[运用]
5. 评估技术决策中的道德/社会权衡。[评估]

SP/职业道德

[2 个核心一级学时;2 个核心二级学时]

计算机伦理是实践哲学的一个分支,是处理计算专业人士应该如何就职业和社会行为做出决定的。有三种主要的影响:1) 一个个体自己的个人守则;2) 关于工作场所道德行为的任何非正式的守则;3) 对正式的道德守则的接触。交叉参照知识领域"信息保障与安全"(IAS)。

知识点:

[核心一级]

- 社区价值以及我们生活的法则。
- 专业精神的本质,包括关心、关注和纪律、信托责任以及指导。
- 作为计算机专业人士,在熟悉程度、工具、技能、法律及专业框架方面与时俱进,并且有自我评估以及在计算领域不断进步的能力。
- 专业认证、道德守则、行为以及实践,如 ACM/ IEEE - CS、SE、AITP、IFIP

和国际性的学会(交叉参照 IAS/安全基本概念/道德问题)。

• 责任制、责任和义务(如软件的正确性、可靠性和安全性,以及网络安全专业人员的合乎道德的保密)。

[核心二级]

• 计算专业人士在公共政策中扮演的角色。
• 保持对后果的意识。
• 伦理异议和举报。
• 地域文化与道德困境的关系。
• 骚扰和歧视的处理。
• 专业资格认证的形式。
• 在工作场所计算的可接受的使用政策。
• 人体工程学和健康的计算环境。
• 上市时间与成本的考虑相对于质量的专业标准。

学习成果:

[核心一级]

1. 识别在软件开发中出现的伦理道德问题,并确定如何在技术上和伦理上处理这些问题。[熟悉]

2. 解释保证软件的正确性、可靠性和安全性的伦理责任。[熟悉]

3. 描述现有的专业人士与时俱进的典型机制。[熟悉]

4. 描述相关专业守则作为专业化表达和决策指导的优势和弱点。[熟悉]

5. 分析一个全球性的计算问题,观察专业人士和政府官员在管理这个问题时扮演的角色。[评估]

6. 从 ACM、IEEE 计算机学会以及其他组织来评估伦理的专业守则。[评估]

[核心二级]

7. 描述专业人员可贡献于公共政策的途径。[熟悉]

8. 描述不当专业行为的后果。[熟悉]

9. 识别举报事件中的各渐进阶段。[熟悉]

10. 举例说明区域文化如何与伦理困境相互影响。[熟悉]

11. 调查骚扰和歧视的形式以及援助的渠道。[运用]

12. 检查各种形式的职业资格认证。[运用]

13. 解释计算环境中的人体工程学和人们健康之间的关系。[熟悉]

14. 制定一个带强制措施的计算机的使用/可接受的使用政策。[评估]

15. 描述有关企业促使对产品上市时间的关注相对于强化产品质量的专业标准的问题。[熟悉]

SP/知识产权

[2 个核心一级学时]

知识产权指的是资产所有权包含的一系列无形权利,如软件程序。每个知识产"权"本身就是一种资产。法律根据其类型提供了不同的方法来保护这些所有权。与软件相关的知识产权基本上有四种类型:专利、版权、商业秘密和商标。每一个对应一种不同类型的法律保护。交叉参照知识领域"信息管理"(IM)。

知识点:

[核心一级]

- 知识产权的哲学基础。
- 知识产权(交叉参照 IM/信息存储与检索/知识产权与保护)。
- 无形数字知识产权(IDIP)。
- 知识产权保护的法律基础。
- 数字版权管理。
- 版权、专利、商业秘密、商标。
- 剽窃。

[选修]

- 开源运动的基础。
- 软件盗版。

学习成果:

[核心一级]

1. 讨论知识产权的哲学基础。[熟悉]
2. 讨论知识产权受法律保护的理由。[熟悉]
3. 描述针对数字版权侵权的立法。[熟悉]
4. 批评针对数字版权侵权的立法。[评估]
5. 找出当前的无形数字知识产权的例子。[熟悉]
6. 使用有版权资料的合法化。[评估]
7. 评估各种剽窃检测机制所固有的伦理问题。[评估]
8. 解释软件许可的目的及实施。[熟悉]
9. 讨论涉及软件专利保障的问题。[熟悉]
10. 刻画并对比版权、专利和商标的概念。[评估]

[选修]

11. 识别开源运动的目标。[熟悉]

12. 识别软件盗版的全球性。[熟悉]

SP/隐私和公民自由

[2 个核心一级学时]

电子信息共享强调平衡隐私保护与信息访问的必要。对很多类型数据的数字化访问很容易,这使隐私权和公民自由变得更复杂,在全球各种文化之间均不相同。交叉参照知识领域“人机交互”(HCI)、“信息保障与安全”(IAS)、“信息管理”(IM)以及“智能系统”(IS)。知识点:

[核心一级]

- 隐私权(交叉参照 IS/基础问题/哲学问题)的哲学基础。
- 隐私保护的法律基础。
- 针对事务型数据库的广泛的数据收集、数据仓库、监控系统以及云计算(交叉参照 IM/数据库系统/数据独立性、IM/数据挖掘/数据清理)的隐私问题。
- 差分隐私的后果
- 基于技术的隐私保护解决方案(交叉参照 IAS/威胁与攻击/对隐私和匿名的攻击)

[选修]

- 在实践领域中隐私权的立法
- 公民自由和文化差异
- 言论自由及其局限性

学习成果:

[核心一级]

1. 讨论个人隐私的法律保护的哲学基础。[熟悉]

2. 评估对事务型数据库和数据仓库隐私威胁的解决方案。[评估]

3. 描述数据收集在无处不在的监控系统(如 RFID、人脸识别、收费系统、移动计算)的实施中扮演的角色。[熟悉]

4. 描述差分隐私的后果。[熟悉]

5. 调查技术解决方案对隐私问题的影响。[运用]

[选修]

6. 批评各种形式的隐私立法的意图、潜在价值和实施。[评估]

7. 找出确保适当言论自由的策略。[熟悉]

SP/专业交流

[1 个核心一级学时]

专业交流向各种不同的听众传递技术信息,这些听众可能对这些信息抱有非常不同的目标和需要。对技术信息进行有效的专业交流不是一种天赋,而是需要在贯穿整个本科课程体系的环境下进行讲授的。交叉参照知识领域“人机交互”(HCI)和“软件工程”(SE)。

知识点:

[核心一级]

- 阅读、理解并总结技术材料,包括源代码和文档。
- 编写有效的技术文档和材料。
- 动态的口头、书面和电子的团队与小组沟通(交叉参照 HCI/协同和通信/团队沟通、SE/软件项目管理/团队参与)。
- 与利益相关者进行专业地沟通。
- 利用协作工具(交叉参照 HCI/协同和通信/网络社区、IS/代理/合作代理)。

[选修]

- 处理跨文化的环境(交叉参照 HCI/以用户为中心的设计和测试/跨文化评估)
- 软件项目中竞争性风险的权衡,如技术、结构/流程、质量、人员、市场和金融(交叉参照 SE/软件项目管理/风险)。

学习成果:

[核心一级]

1. 书写清楚、简洁、准确的技术文档,遵循明确定义的关于格式以及包含适当的图表和参考文献的标准。[运用]

2. 评估编写的技术文档,以检测各种问题。[评估]

3. 制作并提供一份高质量的正规的演示报告。[评估]

4. 规划与他人的互动(如虚拟的、面对面的和文档共享),使他们能够让自己的观点与别人交流,且即使不同意也能够仔细聆听和欣赏别人的观点,并能够将自己所闻传达给别人。[运用]

5. 描述各种沟通方式(如虚拟的、面对面的和文档共享)的优势和弱点。[熟悉]

6. 检查用于与项目中利益相关者进行沟通的适当措施。[运用]

7. 比较并对比各种协作工具。[评估]

[选修]

8. 讨论影响跨文化团队绩效和结果的途径。[熟悉]

9. 就技术、结构/流程、质量、人员、市场和金融等方面,检查软件项目中风险的权衡和共同来源。[运用]

10. 评估个人作为跨国团队的一分子进行远程工作的长处和弱点。[评估]

SP/可持续性

[1 个核心一级学时;1 个核心二级学时]

可持续性的特点按联合国[1]的描述为“这种发展既满足当代人的需求,又不损害后代人满足其自身需求的能力。”可持续发展在 CS2008 课程指导中被首次推出。这个新兴领域的知识点可以被自然地融入其他熟悉的领域和单元,如人机交互和软件演化。交叉参照知识领域“人机交互”(HCI)和“软件工程”(SE)。

知识点:

[核心一级]

- 作一个可持续发展的实践者,将实施的决定(如组织的政策、经济可行性以及资源消耗)对文化和环境的影响纳入考虑。
- 探索计算机的使用和废弃(电子垃圾)对全球社会和环境的影响。

[核心二级]

- 在如算法、操作系统、网络、数据库或人机交互等特定领域中设计选择对环境的影响(交叉参照 SE/软件演化/软件演化、HCI/面向设计的人机交互/可持续性)

[选修]

- 可持续设计标准的准则。
- 复杂的以计算机为媒介的现象(如远程办公或网上购物)所产生的系统性的影响。
- 普适计算;融入日常对象和活动中的信息处理,如智能能源系统、用以促进可持续行为的社交网络和反馈系统、交通运输、环境监测、公众科学与公民参与。
- 研究计算在环境问题上的应用,如能源、污染、资源使用、回收和再利用、餐饮管理、农业及其他。
- 软件系统与社会系统可持续发展的相互依存关系,包括其用户的知识和技能、组织流程和政策以及其社会环境(如市场力量、政府政策)。

学习成果:

[核心一级]

1. 找出成为可持续发展实践者的途径。[熟悉]

2. 展示计算机的使用和废弃(电子垃圾)对全球社会和环境的影响。[运用]

[核心二级]

3. 描述在与算法设计、操作系统设计、网络设计、数据库设计等相关的计算领域中设计选择对环境的影响。[熟悉]

4. 通过项目调查新的系统设计对社会和环境的影响。[运用]

[选修]

5. 指出可持续的 IT 设计和部署的指导方针。[熟悉]

6. 列出远程办公或网上购物的可持续效应。[熟悉]

7. 在如智能能源系统、社交网络、交通、农业、供应链系统、环境监测和公民参与等领域调查普适计算。[运用]

8. 开发计算机应用程序,并通过环境问题(如能源、污染、资源使用、回收和再利用、餐饮管理、农业)相关的研究领域进行评估。[评估]

SP/历史

[选修]

讲授计算的历史,是要提供计算的迅速变化如何在全球范围内影响社会的认知。经常会在某些基本概念的背景下讲授历史,如系统基础和软件开发基础。

知识点:

- 史前,1946 年之前的世界。
- 计算机硬件、软件、网络的历史(交叉参照 AR/数字逻辑与数字系统/计算机体系结构的历史)
- 计算的先驱者。
- 互联网的历史。

学习成果:

1. 指出计算领域历史中显著持续的趋势。[熟悉]

2. 指出若干先驱者在计算领域的贡献。[熟悉]

3. 讨论几种编程语言范式的历史背景。[熟悉]

4. 比较个人电脑和互联网问世之前和之后的日常生活。[评估]

SP/计算经济性

[选修]

计算经济学涵盖围绕计算机信息系统的人事和财务管理的指标及最佳

实践。

知识点：

- 垄断及其经济意义。
- 熟练劳动力的供给和需求对计算产品质量的影响。
- 计算领域的定价策略。
- 外包与离岸外包软件开发现象及其对就业和经济的影响。
- 对计算机科学专业而言全球化的后果。
- 对计算资源访问的不同，及其可能产生的影响。
- 职位的成本/效益分析，考虑制造、硬件、软件以及工程的影响。
- 与总成本相关的成本估算与实际成本的对比。
- 创业：前景与陷阱。
- 网络效应或需求方规模经济。
- 工程经济学在财务处理中的使用。

学习成果：

1. 总结反垄断努力的理由。[熟悉]
2. 指出若干种信息技术企业受到劳动力供给短缺影响的途径。[熟悉]
3. 指出计算货物和服务定价策略的演化。[熟悉]
4. 讨论离岸和外包的得失与影响。[熟悉]
5. 调查解决限制访问计算的途径并为之辩护。[运用]
6. 描述网络效应的经济效益。[熟悉]

SP/安全政策、法律和计算机犯罪

虽然安全政策、法律和计算机犯罪是重要的课题，也必须在其他“社会与专业”的知识单元基础上看待它们，如知识产权、隐私和公民自由、社会背景以及职业道德。在过去 75 年中，计算机和互联网也许比任何其他技术都更多地改变了社会。与此同时，它们也带来了对隐私权前所未有的威胁、全新类别的犯罪和反社会行为、对组织的重大破坏以及信息系统中风险的大规模集中。交叉参照知识领域“人机交互”(HCI)和“信息保障与安全”(IAS)。

知识点：

- 计算机犯罪和计算机罪犯的法律平反的例子(交叉参照 IAS/数字取证/举证规则)。
- 社会工程学，身份盗窃与恢复(交叉参照 HCI/人因和安全/信任、隐私与欺骗)。
- 围绕访问的滥用以及安全性中的违规问题。

- 网络恐怖主义和犯罪性黑客攻击的动机和后果,“破解”。
- 恶意软件的影响,如病毒、蠕虫和特洛伊木马。
- 犯罪预防策略。
- 安全政策(交叉参照 IAS/安全政策和管理/政策)。

学习成果:

1. 列出造成社会影响的计算机犯罪与社会工程事故的典型例子。[熟悉]
2. 指出适用于计算机犯罪的法律。[熟悉]
3. 描述网络恐怖主义和犯罪性黑客攻击的动机和后果。[熟悉]
4. 检查围绕访问的滥用以及安全性中的违规产生的伦理和法律问题。[运用]
5. 讨论在安全性和相关权衡中专业人士扮演的角色。[熟悉]
6. 调查个人和包括政府在内的机构均可以采取的措施,以防止或减轻计算机犯罪和身份盗窃造成的不良影响。[运用]
7. 编写一份全公司的安全政策,其中包括用于密码管理和员工监察的流程。[运用]

参考文献

[1] “Our Common Future.” http://grawemeyer.org/worldorder/previous-winners/1991-theunited-nations-world-commission-on-environment-and-development.html

[2] Tucker, A. (ed), B. Barnes, R. Aiken, K. Barker, K. Bruce, J. Cain, S. Conry, G. Engel, R. Epstein, D. Lidtke, M. Mulder, J. Rogers, E. Spafford, A. Turner, *Computing Curricula* 1991: *Report of the Joint Curriculum Task Force*, ACM Press and IEEE-CS Press, 1991.