Learn-by-doing-collaboratively across the curriculum: Integrative projects at UCenfotec

Ignacio Trejos
Member, IEEE
Universidad Cenfotec
Montes de Oca, Costa Rica
itrejos@ucenfotec.ac.cr

Álvaro Cordero
Universidad Cenfotec
Montes de Oca, Costa Rica
acordero@ucenfotec.ac.cr

***Abstract*— Projects typically appear late in Computing and Engineering curricula, as 'capstones' at the end of a degree program. There are alternatives: since year 2000, UCenfotec offers Computing curricula with 'backbone' integrative projects, and students learn to work collaboratively in teams with defined roles and processes. Students acquire relevant knowledge just-in-time, in combination to previous learnings, perform technical activities. They learn: work as teams using systematic engineering processes pertinent to their Computing subdiscipline; communicate effectively; be accountable as teams and individuals; research and apply diverse technologies; effectively manage individual and team work, and critically evaluate process artifacts. Each curriculum has a sequence of incremental projects distributed in 9 terms, that students experience learn-by doing- collaboratively technological artifacts of increasing complexity, difficulty, diversity, heterogeneity and realism, where 'hard skills' are intermingled with 'soft skills' that develop from the individual to the collective, each member growing in autonomy and responsibility, via experience-driven learning.**

Keywords—experience-driven learning; integrative projects; learn-by-doing; collaboration; teamwork; soft skills; Engineering education; Computing education.

I. INTRODUCTION

Most Computing and Engineering curricula are subject-based. Subjects are developed typically in thematic courses, sometimes calling for sequences of courses in order to cover themes in adequate breadth or depth. Integrative courses, when they exist, are usually implemented in one or two semesters towards the very end of the degree program, often called *capstone* projects. Students usually work in groups to design and implement such projects, which may involve real-world issues including cost, safety, efficiency and suitability for the intended user [1]; the work to be developed may involve on- or off-campus clients. The latest Curriculum guidelines for degree programs in Software Engineering [2] recommends that students complete a significant project, preferably working in teams during a year, so that they practice the knowledge and skills learned in previous courses, together with team skills valuable in professional environments, in contexts which have a real-world customer.

This paper presents an approach that has evolved at Universidad Cenfotec since its foundation as a two-year college in year 2000: integrative projects form a *backbone* in the curricula: they start very early, are incremental, progressively integrate knowledge and skills required for problem solving using systematic engineering processes, and incorporate the concurrent development of both 'hard' and 'soft' skills required for teamwork and future professional endeavors. Experience-driven learning-by-doing computing products or services is implemented through projects that require actual teamwork in which members collaborate, performing roles for which they are accountable, employing emotional and social intelligences as cornerstones. The paper emphasizes the origins of the approach.

II. CONTEXT: THE CHALLENGES

A. Rapid growth of the IT sector, worldwide

Between years 1997 and 2000, the IT sector was experiencing an explosive growth [3], [4]; the North American edition of Computer World reported an unprecedented increase of unfilled job positions related to Information Technology: from 160,000 positions in 1997 to 800,000 in year 2000 [5]. Costa Rica was no exception, with software companies and start-ups exporting successfully to Latin America and growing at a rate of 40% to 60% in headcount during the same period [6], [11].

B. The IT Skills Set Gap

The IT skills gap refers to the unavailability of sufficient(ly) qualified candidates to fulfill open positions in IT-related occupations. The problem manifests itself in diverse ways [7]:

- Insufficient numbers of educated persons to fulfill job openings;
- Candidates who are inadequately prepared for certain IT positions;
- New occupations generated by technological change and innovation, or new business models and services;
- Need for retraining personnel in order to assimilate acquired technologies;
- Maintaining and keeping operational information systems built on older or non-fashionable technology;
- Underrepresentation of women and ethnic groups in the IT workforce impairs diversity, thus negatively impacting creativity, innovation, productivity and prosperity;

DOI 10.14684/EDUNINE.1.Vol.2.2017.11-16

- Educational institutions are slow in creating and innovating curricula to prepare the future IT workforce and developing programs to retrain those already employed.

C. Individual and social skills development

At the turn of the 21st Century, several industry, professional and academic forums recognized the need to better prepare for professional life in the workplace. Some Engineering educators started to call for a more holistic and integrative approach, as opposed to the prevailing model rooted on analysis (science) [8]. In 2004, the National Academy of Engineering asserted these attributes of engineers in year 2020 [9] (we highlight those related to so-called 'soft skills'): Strong analytical skills; Practical ingenuity; **Creativity**; **Communication**; Business and management; **Leadership**; **High ethical standards**; **Professionalism**; **Dynamism**, **agility**, **resilience**, and **flexibility**; **Lifelong learning**.

The above is very much in agreement with what employers express in surveys presented in [7] and [10], where the non-technical attributes that were mentioned by 50% or more of the respondents include: Leadership; Ability to work in a team; Communication skills (written and verbal); Problem-solving skills; Work ethic; Initiative; Analytical/critical/quantitative skills; Flexibility/adaptability; Interpersonal skills; Organizational ability; Strategic planning skills.

D. Prior art

By year 2000, capstone projects appeared rather infrequently in Computing *undergraduate* curricula; Chamillard and Braun report on their experience and present an overview of then-current relevant literature [15]. Interesting *graduate* precedents include [16], where Horning and Wortman explain the organization of a Software Engineering project course in the form of a game, and Wang Institute's Master's in Software Engineering [17]. In Costa Rica, capstone software development projects started to appear in 1986, at the Costa Rica Institute of Technology's BEng in Computing [18]; they were not team-based nor did follow a particular software engineering process. A study on the relevance of Computer Science and Software Engineering education was reported in May 2000 by Lethbridge [19], where respondents identify the lack of experience-based learning prior to graduating and joining the workforce. Mary Shaw has called for engineering approaches within computing degrees [20, 21].

Such a context sets the scene for Cenfotec's curricular innovations, which started in year 2000 with a Software Engineering program.

III. Starting Cenfotec

Cenfotec was founded in year 2000 by a group of software development entrepreneurs interested in contributing to Costa Rica's knowledge-based economic development. Founders understood that the number and profiles of graduates from existing Costa Rican universities were insufficient and put the industry's sustainability at risk; hence, they sought to prepare specialized personnel in short, very-focused higher-education programs, to join the digital technologies industry, aimed at growing domestically and internationally.

A. Elements of needs analysis and curriculum design

We list the key elements considered during the initial curriculum design:

- Investigate needs from an employer's standpoint. A team of three professors, experienced in software development, interviewed over 30 software professionals in order to understand the gap between knowledge and skills of recent college graduates and junior software development positions in industry.
- Understand the actual use of software engineering practices and technology mixes.
- Interpret trends in technology, practices and standards, and application areas.
- Incorporate the development of individual and social skills jointly to that of technological competencies.
- Emphasize *learning* via experiences rather than teaching 'theory'. Most people learn inductively since childhood.
- Emulate *realistic* software production environments.
- Keep the curricula *pertinent* – well-aligned with needs, foresighting developments in technology and practice.
- Keep the programs shorter than two years: focused, intense and demanding.

B. Competencies to be developed

Interaction with colleagues from industry produced a desiderata of competencies to develop in the initial curricula:

- Practice systematic processes of software engineering.
- Construct applications on modern software architectures.
- Design and construct Web business applications.
- Integrate enterprise software technology: relational databases, components, objects, platforms.
- Elicit, validate and specify functional requirements.
- Design enterprise software systems using patterns, components and services.
- Program, build and validate software applications and components.
- Perform detailed database design.
- Develop technical documentation.
- Experience project discipline, teamwork, business communications, and personal development.

IV. Initial curriculum (years 2000 -2003)

The design team iterated seven times until reaching an overall curriculum architecture that seemed workable and resilient. The first two programs were designed to prepare software developers: one targeted at high-school graduates to educate them as junior software developers, the other offering complementary education in software development to professionals from other disciplines. Each term of the original programs included a software engineering project, 3 technical courses and one English for IT course. Younger students had a fourth term with an internship practicum, plus two technical electives. Study required 60 to 70 hours a week on lectures and work on courses and projects. Besides, since the start-up in year 2000, continuing education and professional development programs have been offered to the practitioner community.

Table 2 provides a birds-eye view of the 4-term curriculum used at the start of classes in September 2000.

Table 2: First working curriculum (Software development, year 2000)

Term 1	Term 2	Term 3	Term 4
Programming 1 (algorithms in Java)	Programming 2 (objects in Java)	Component design and construction	Elective 1
Introduction to IT	Data structures	Software engineering processes	Elective 2
Business processes	Databases	Conceptual software design	
Software development project 1 (simple Web portal with database)	Software development project 2 (3-tiered Web + local app / MS technology)	Software development project 3 (3-tiered Web application / Java technology)	Enterprise practicum (internship)
Intermediate English	English for IT 1	English for IT 2	

A. *Principles*

Some principles were set to govern course and project design, development and teaching:

- Concept motivation and contextualization: pertinence.
- Practice interleaved with concept understanding.
- Exercises, laboratories, assignments for reinforcing and deepening knowledge.
- Exploit the power of example; recognize and reflect on counter-examples.
- Integrative projects that comprise: Engineering/technical processes; Management/coordination processes; Human development processes; Communication processes; Software technology.
- Discipline to be worked incrementally, progressively building autonomy.
- Facilitation, guidance, coaching and reflection for the emergence of attitudes and values.
- 'A social laboratory': actual role performance, with coaching and peer assessment.
- Responsibility reinforcement throughout studies: for learning, for project work, for artifact design and construction.
- Frequent follow-up by lecturers and tutors.
- Very intense and focused.

B. *Overview of original terms 1 to 3*

Term 1 (Foundations):

- Algorithmic thinking
- Problem solving
- Basic modeling
- Context of IT and business
- Self-discipline and work organization
- Collaboration and teamwork
- First team project software development experience / Iterative-evolutionary process

Term 2 (Detailed design, reuse):

- Modeling and building classes and objects.
- Modeling and building abstract data types.
- Modeling and implementing entities and relationships.
- Building a multi-tiered software architecture.
- Constructing tier components.
- Database modeling, creation and manipulation.
- Second team software project / Iterative-incremental process.

Term 3 (Further design, process, requirements):

- Modeling (functional) interfaces.
- Applying frameworks.
- Design, analysis and application patterns.
- Process and project perspectives.
- Dealing with customers and implementation.
- Introduction to non-functional requirements.
- Third team software project / Iterative process.

V. ORIGINAL PROJECT ARCHITECTURE AND EXPERIENCE

A key differentiator of Cenfotec's original curricula, which has evolved during 16 years to date is *learn-by-doing-collaboratively via projects*. Although students learn practically in courses, those are mostly *solo* undertakings. Projects are *team* experiences, whose general characteristics are:

- **collaborative**, process is performed by a team, which encourages cooperation based on strengths and skills of the students, who assume shared responsibilities;
- projects' **processes** structure and guide the search for knowledge and encourage work discipline, values, skills and attitudes framed within formal activities in which the students perform well defined roles to create or assess relevant artifacts;
- **integration** of knowledge and experience: the learning of various concepts is applied in activities and tasks that develop a product or service, also calling to previously acquired knowledge or skills;
- **ownership**: students learn how to (better) learn;
- **pertinence** and **realism**, work focused on performing according to the reality of the industry, good practices and world trends in IT products and services;
- **human development**, emphasizes communication, team work and value systems for obtaining quality results in demanding environments.

A. *Team roles structure*

A *role* defines a set of responsibilities that a person must assume to contribute to a project's progress and coordinate their work with other members that rely on them. Roles are assigned to persons able to perform certain activities and develop required artifacts.

Team structure is inspired in the roles for software development proposed by Watts Humphrey in [12]. Every member is a software *developer*, who performs analysis, design, construction and quality control of software. Most software development projects require additional activities that do not directly produce software: plans, coordination, manage change, define standards, support, assess options or artifacts, integrate products, etc. These are the roles assigned to members:

- Team coordinator.
- Project planning and follow-up coordinator.
- Development coordinator.
- Quality coordinator.
- Process coordinator.
- Support coordinator.

B. *Project processes and characteristics*

Project 1:

- Offers a first end-to-end, complete software development teamwork experience.
- Iterative process, *evolving* artifacts.
- Activities and artifacts standardized.
- Roles defined and assigned.
- Fundamental teamwork.
- Rigorous documentation (templates provided).
- Given Gantt-chart planning, milestone-driven. Team assigns and balances work.
- Tutored project control. Frequent oversight and assessment to assure learning and meeting requirements or deadlines.
- Requirements provided (and 'invented') by lecturers.
- Requirements specification and use cases.
- Basic UML (use cases, system interactions, activity diagrams).
- Quality control (checklists, reviews, test plans, test designs, test records).
- Web page design and construction.
- Forms and input validation with scripting language.
- Basic data base manipulation via Server Pages in simplified multi-tier architecture.
- Simple artifact management: identification and backup.
- Oral and written communication.
- Public presentation of work.

Typical student testimonials after experiencing Project 1 include:

- "Responsibility-driven work is new and hard"
- "Everyone's commitment is key"
- "Meeting commitments (deadlines, goals) is crucial for progress"
- "One must learn to endure workloads and persevere"
- "Learning and project work requires dedication"
- "Trust is difficult to build, but essential for teamwork"
- "Empathy for peers and users encourages better systems"
- "Orderly work is needed for reliance and knowing project and artifact status"
- "Motivation helps teams and individuals keep going"
- "We evolved from teammates to become friends"

Project 2:

- Built on Project 1, and courses of levels 1 & 2.
- Iterative-incremental process.
- More autonomy in project planning and control: from one milestone to the next.
- Simplified, but realistic, functional requirements for transactional information system.
- 'Mock-up' user (realistic but fictitious).
- UML modeling of domain, software and database.
- Pre-defined architecture: multi-tiered, to be populated with reusable components.
- Components reused for local ('rich desktop') and Web-based user interfaces.
- Use of component framework and middleware.
- Basic user interface design.
- Basic software configuration (version) management.
- Technical peer reviews.

Project 3:

- Built on Project 2, and courses of levels 2 & 3.
- Domain modeling and prototyping, followed by iterative design-build-validate-integrate process.
- Autonomous planning.
- Autonomous construction of multi-tiered architecture.
- Process and product audits.
- Real user for functional requirements; lecturers guide non-functional requirements.
- Basic requirements management.
- User training and product implementation.
- Quality planning and management.
- Configuration management.
- Risk management.
- Further learning of human development, business communications, user interface design.

C. *Development of individual and social skills*

In years 2000-2003, the *Human development* program proceeded from Project 1 through Project 3. The accent was put first on developing skills required of individuals to become productive members of teams and to succeed in a demanding learning environment; inspiration is drawn from the works of Goleman and Cherniss on Emotional intelligence [13] and Covey's 7 Habits for effectiveness [14]: study habits, time management, self-awareness, self-regulation, motivation, proactivity, goal-setting, priority-setting, and basic team-work. On Project 2, intra-team social skills are emphasized: empathy, influence, reaching agreements, synergy, service orientation, and group dynamics. On Project 3, relationships external to the team are stressed: negotiation, customer service, resistance to change, ethics in software development, relating to superiors, stakeholder management.

The *Business communications* program also goes across all projects: oral communications, effective presentations, written communications, effective meetings, relationships with users and customers, user training, selling ideas, formal presentation, business etiquette, preparing a résumé, preparing for job interviews.

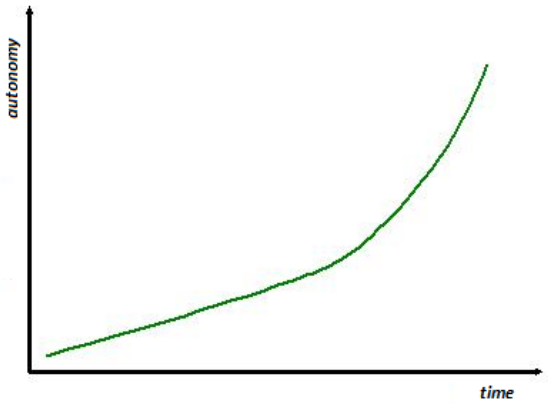
VI. Evolution and Impact

Though fundamental ideas prevail, the original design has evolved during 16 years with incremental adjustments in years 2002, 2005, 2008, 2010, 2012 and 2016. Feedback has been a key to improvements: from employers, graduates and internships (where students get a job offer in more than 90% of the cases). Table 3 in the Appendix summarizes the key elements of projects in Universidad Cenfotec's 9-term Bachelor of Software Engineering program as of 2016 (which introduced software innovation at the end of the project sequence since 2005). The overarching principles have been transferred and adapted to each of three other undergraduate programs, in addition to Software Engineering, considering the diverse competencies to be developed in accordance to their corresponding professional profiles:

- Information Technology.
- Front-end Web development.
- Networking.

In all these curricula an intentional progression has been designed. As students advance in their chosen curriculum, they grow and mature in several dimensions, as illustrated in Figure 1 with the 'autonomy + conscience of interdependence' dimension.

Fig. 1. Autonomy development over time



These are the dimensions developed in the span of 9 terms, involving four integrative projects and one practicum, in addition to 30 or more other courses for each curriculum:

- Integrative and incremental: previous knowledge reused in new contexts, just-in-time acquisition of new knowledge.
- Inductive learning: from the concrete and particular towards the abstract and general.
- Increasing complexity and difficulty.
- Increasing heterogeneity and diversity.
- Social expansion: from the individual to the collective.
- Increasing autonomy, with conscience of interdependence: degrees of freedom expand related to decision making - in architecture, design and construction, risk management, project planning.
- Increasing realism (from local and controlled organizations, domains and requirements to external organizations with real users and environments, plus unknown and changing requirements).

Universidad Cenfotec's *learn-by-doing-collaboratively* via projects is expounded at greater length in [22].

ACKNOWLEDGMENT

This work has been sponsored by Universidad Cenfotec, Costa Rica, and communicates the contributions and accomplishments of numerous people, most notably: José Aurelio Sánchez, Franco Quirós, José Álvaro Romero, Álvaro Cordero, Pablo Monestel, Gerardo Parajeles, Juan Matías, Andrés Labera, and Laura Coto.

REFERENCES

[1] ACM/IEEE-CS Joint Task Force on Computing Curricula. *Information Technology 2008: Curriculum Guidelines for Undergraduate Programs in Information Technology*. ACM Press and IEEE Computer Society Press, 2008.

[2] ACM/IEEE-CS Joint Task Force on Computing Curricula. *Software Engineering 2014. Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering*. ACM Press and IEEE Computer Society Press, 2008.

[3] D. Hoch and G. Purkert. *Secrets of software success*. McKinsey & Company, 2000.

[4] National Research Council. *Building a workforce for the Information Economy*. National Academy Press, 2001.

[5] Computer World, USA/North American editions. Years 1997-2000.

[6] F. Mata and A. Jofré. *Estudio de oferta y demanda del recurso humano en la industria de software*. [Study of supply and demand of human resources by the software industry]. Pro-Software (Inter-American Development Bank, Caprosoft, Procomer, FunCenat), 2001.

[7] M. Sabin, B. Viola, J. Impagliazzo, R. Angles, M. Curiel, P. Leger, J. Murillo, H. Nina, J.A. Pow-Sang & I. Trejos. "Latin American Perspectives to Internationalize Undergraduate Information Technology Education". *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE 2016). ITiCSE '16 Working Group Reports*, July 09-13, 2016, Arequipa, Peru. A. Clear, E. Cuadros-Vargas, J. Carter, Y. Tupac (Eds.). ACM, 2016.

[8] E. Smerdon. "An Action Agenda for Engineering Curriculum Innovation". *11th IEEE-USA Biennial Careers Conference*, November 2-3, 2000, San Jose, California.

[9] National Academy of Engineering. *The Engineer of 2020: Visions of Engineering in the New Century*. National Academies Press, 2004.

[10] National Association of Colleges and Employers. *Job Outlook 2016: Attributes Employers Want to See on New College Graduates' Resumes*. Retrieved from www.naceweb.org/s11182015/employers-look-for-in-new-hires.aspx on 2016/12/30.

[11] L. Brenes and V. Govaere. "La industria del software en Costa Rica" [The software industry in Costa Rica]. *Comercio Exterior*, vol. 58, no. 5, pp. 303-311, may 2008.

[12] W. Humphrey. *Introduction to the team software process*. Addison-Wesley, 2000.

[13] C. Cherniss and D. Goleman. *The emotionally intelligent workplace*. Jossey-Bass, 2001.

[14] S. Covey. *The seven habits of highly effective people*. Free press, 1989.

[15] A. T. Chamillard, K. Braun. "The software engineering capstone: Structure and tradeoffs". *Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education*, 2002, Cincinnati, Kentucky, USA, February 27 - March 3, 2002.

[16] J.J. Horning, D.B. Wortman, "Software Hut: A Computer Program Engineering Project in the Form of a Game", *IEEE Transactions on Software Engineering*, vol. 3, no. , pp. 325-330, July 1977.

[17] R. Fairley, N. Martin. "Software engineering programs at the Wang Institute of Graduate Studies". *ACM '83: Proceedings of the 1983 annual conference on Computers : Extending the human resource*, December 1982.

[18] I. Trejos. *Software development in Costa Rica: unlocking the potential*. Technical report. Instituto Tecnológico de Costa Rica, June 1997.

[19] T. Lethbridge. "What Knowledge Is Important to a Software Professional?". *IEEE Computer*. vol. 33, no. 5, pp. 44-50, June 2000.

[20] M. Shaw, "Prospects for an Engineering Discipline of Software", *IEEE Software*, vol. 7, no. 6, pp. 15–24, 1990.

[21] M. Shaw. "Software engineering education: a roadmap". *ICSE '00 Proceedings of the Conference on The Future of Software Engineering*. Limerick, Ireland, June 04 - 11, 2000.

[22] I. Trejos, Á. Cordero, J.Á. Romero and M.E. Ucrós. "Proyectos integradores en la Universidad Cenfotec". [Integrative projects at Universidad Cenfotec.]. *I Simposio Internacional sobre Innovaciones Curriculares*. Universidad de Costa Rica, Programa de Posgrado en Planificación Curricular. San José, 05-07 Octubre, 2016.

APPENDIX

Table 3: Bachelor of Software Engineering projects framework at Universidad Cenfotec (year 2016)

	Teamwork	**Process, method**	**Project management**	**Quality**	**Configuration**	**Technology**
Project 1	Groups of 5 to 6 students, formed by lecturers. Roles assigned by lecturers.	Iterative evolutionary. Two phases, with stages of analysis, design, construction and testing. *Invented simple requirements.*	Project concept, stakeholders, activities and tasks, organization and responsibilities. Requirements principles.	Quality and customer concepts. Basic testing: types, cases, documentation, execution, analysis.	Item version control using folders and cloud storage (e.g. Dropbox or Google Drive).	Client side: JavaScript, HTML 5, CSS 3. Server side: PHP. Database: MySQL or Microsoft Access.
Project 2	Groups of 4 to 5 students, formed by lecturers. Roles assigned by lecturers.	Iterative incremental. Two phases, with stages of analysis, design, construction and testing. *Realistic e-business requirements.*	Requirements prioritization. Constraints: time, cost, scope. Short-term Project tasks planning.	Test planning and risks. Systematic Unit testing and test case design.	Version control management with *Team Foundation Sever*.	Mostly Microsoft technology-centric: .NET (C#, Visual Basic.NET), SQL Server. Web forms. Win forms. JavaScript, HTML 5, CSS 3.
Project 3	Groups of 4 students, proposed by students. Roles assigned by students.	Agile, based on Scrum. First phase of user-story based requirements. Then two development iterations of 4 sprints each. *Real user and requirements.*	Activity identification and organization with WBS, activity network, creation of project management program. Risk management. Effort estimation techniques.	Test automation. Performance tests using tools. Quality assurance and measurement.	Version management with Git.	Java Enterprise Edition. Data access tier independent of database engine. Advanced JavaScript, HTML 5, CSS 3.
Project 4	Groups of 4 students, proposed by students. Roles assigned by students.	Agile, chosen and justified by students. Innovation-oriented software development. *Real business or start-up requirements.*	Control, estimation and management of agile projects. Students inquire about techniques and justify their choices.	Tests in agile environments. Students define or adapt standards to be used.	Students choose version control tool and standard to be used.	Service-based architecture. Mobile clients. Students have freedom to select technology stack.