# A Software Entrepreneurship Course - Between two paradigms

Aaen, Ivan; Rose, Jeremy

*Publication date:*
2011

*Document Version*
Accepted author manuscript, peer reviewed version

# A Software Entrepreneurship Course

## Between two paradigms

Ivan Aaen, Jeremy Rose
Department of Computer Science, Aalborg University, Denmark

**Abstract**
Entrepreneurship courses usually build on a traditional plan-based paradigm for starting up new ventures but what if the students are more at home with another paradigm? Our Computer Science students usually favor an agile paradigm for software development. This paradigm is based on incremental and experimental development. We have developed a new entrepreneurship course based on both paradigms and let our students choose among these paradigms when they do a mini-project as part of the course. This paper reports from the first year of teaching two paradigms for entrepreneurship.

## Introduction

How can we teach entrepreneurship to Computer Science students? Many entrepreneurship courses are extracurricular, voluntary, and shared among several different educations; and many are a-theoretical - paying particular attention to the practical side of starting a company. There are good reasons for this, but also obvious drawbacks. Not all students are recruited since some students do not plan to become entrepreneurs at the time such courses are offered. Sometimes these courses are also less esteemed among students that favor theory and scientific evidence in their education. Sometimes such courses are even seen as a *cuckoo in the nest* by faculty staff perceiving them as un-academic and stealing students away from 'real' courses.

We have chosen to make our course mandatory, academic, and tailor-made for ICT based entrepreneurs. All Computer Science students must follow this course and pass the exam, and the course is therefore similar to any other course in the curriculum by having an academic core based on research literature – empirical as well as theoretical - and well-defined concepts.

One question in designing a course for Computer Science is whether ICT students favor a traditional plan-driven paradigm that *per se* appears logical, easy to understand, and systematic although quite different from the preferred paradigm for their main study; or if they favor a similarity between their usual paradigm and the entrepreneurship paradigm even if the latter may appear less systematic to a beginner in the field.

This question is not just a mental thing but also a practical question of how the business and the software solutions related to it - products, processes, and services - will be developed.

Our assumption when designing the course was that many students would favor a similarity – possibly even a correspondence – between business development and solution development over the comfort of using a plan-driven paradigm to make this new territory less alien.

This first year of teaching gave an idea of whether a familiarity with paradigms would matter more to our students than using a fairly orderly, classic structure to support their initial steps into new territory.

## Two paradigms

Historically, software development was dominated by a plan-oriented paradigm where software solutions are built according to specifications; where costs and delivery dates are estimated based on requirements; where development activities are planned in detail based on estimates; and where changes to requirements are assumed to be rare after development has started.

Obviously, these assumptions are rarely met. Software development is usually not very predictable: Requirements change rapidly; estimates are uncertain; software teams change; and so do project conditions in general.

A number of changes in software technologies and competencies have made an alternative – *agile* – paradigm possible, and this new paradigm has been a serious contender to traditional software development for well over a decade. The agile paradigm is based on theoretical and methodological work such as: Takeuchi & Nonaka (Takeuchi and Nonaka, 1986); Schwaber & Beedle (Schwaber and Beedle, 2002); and Beck (Beck, 2000). The paradigm as such was articulated in the Agile Manifesto in 2001 (Beck et al., 2001).

Our students usually favor the agile paradigm and use it in some of their study projects. This paradigm assumes that software development is akin to new product development where not all requirements are known upfront; where early estimates are not reliable; where adaptive rolling-wave planning is required in place of detailed up-front planning; and where creative adaptation to unforeseen changes is needed (Larman, 2004).

When we teach Software Engineering, we systematically balance between these two paradigms. Understanding how paradigms differ is a way to empower our students and make them able to see alternatives in how to work.

In our design of the Entrepreneur course for Computer Science students we have sought to balance between two paradigms similar to those from Software Engineering. We call these two paradigms *Analyze – Design - Enact (ADE)*, and *Consider - Do - Adjust (CDA)*. ADE corresponds to the traditional – plan-based – paradigm (Dollinger, 1999; Kuratko and Hodgetts, 2004), while CDA corresponds to the agile paradigm and is inspired by Saravathy and others (Wiltbank et al., 2006; Sarasvathy, 2008). These two entrepreneurship paradigms are outlined in the below table from the course notes by Jeremy Rose.

## Educational setting and data collection

All studies at Aalborg University are based on Problem-Based Learning (PBL). The students work on problems in project groups. The problem defines the overall challenge to be answered by the project - be that a technology project, or indeed a business development project as is the case here. These projects are done in small self-organizing teams, where theory and practice are integrated. The team chooses theories and methods appropriate for the project.

| | **Analyze – Design - Enact** | **Consider – Do - Adjust** |
|---|---|---|
| **Theoretical inspiration** | Standard entrepreneurship literature | Sarasvathy's theory of effectuation (Sarasvathy, 2008) |
| **Software engineering inspiration** | Traditional development | Agile development |
| **Description** | Promote a software venture by understanding the technical and business environment, designing a logical response and setting it in motion | Promote a software venture by taking iterative and incremental steps forward on the basis of what is achievable now |
| **Understanding of future** | Predictable, non-controllable – adjust to the forces of the market | Unpredictable, controllable – create (a small part of) the future |
| **Attitude to market** | Analyze the whole market and identify obstacles | Imagine a place for a possible entry into the market |
| **Attitude to technology development** | Understand technology trajectories and fit software projects into likely developments | Develop the areas of technology expertise you excel in |
| **Role of business planning** | Conceptualize the venture as completely as possible before starting | Take initial business decisions based on what you can afford and look out for what you need to do next |
| **Software development style** | Decide software engineering practice up-front, likely to emphasize the traditional | Improvise engineering practice and decide according to circumstance, lean practice, agility |
| **Attitude to change** | Avoid change as far as possible - stick to plan to achieve goals | Embrace change as opportunity and act upon new situations and possibilities |
| **Funding approach** | Attract capital and investors before start up - fund to maximize return | Invest what you can afford without considering the possible return and bootstrap |
| **Approach to others working in the same areas** | Avoid competition. Consider competitors a threat | Collaborate with those who demonstrate commitment and network with potential stakeholders |
| **Approach to intellectual property** | Protect IP through copyrights and patents to deter competition, improve market share | Build networks of ideas, sharing, various business models including open source |
| **Partnering and networking** | Control collaboration to avoid loosing intellectual property | Build a network of committed stakeholders, collaborators and partners |
| **Time to market** | Long on the basis of venture capital | Short to maximize returns |

PBL ensures that our students are comfortable with collaboration, self-management, and uncertainties. Pedagogically, this makes miniprojects a relevant platform for our course, but PBL is also relevant per se for teaching entrepreneurship to software people. Niefert et al. found that the software industry is the one where most start-ups (67 percent) are created by teams with a typical team size of three (Niefert et al., 2006, 9.). This matches our choice to allow the students to work on miniprojects in groups of up to three students.

At the end of the course the students should:

- Be able to identify, discuss and relate major theoretical themes and issues in contemporary Software Entrepreneurship research
- Display a paradigmatic understanding of Software Entrepreneurship, and relate individual research or practical contributions to those paradigms
- Understand how to formulate and develop their own entrepreneurial ideas and suggestions and communicate these to knowledgeable listeners
- Be able to relate their own ideas and suggestions to theoretical paradigms, themes and issues

The course was offered on the 9th semester, i.e. half a year before the students graduate as masters. The course was mandatory and graded via individual oral exam based on a miniproject report.

The 3 ECTS course consisted of 11 half days organized in 6 sessions with 1-2 weeks intervals. The sessions were used for lectures, paper presentations, guest lectures, and feedback to miniprojects. The students were required to present papers, comment on papers, and present status for their miniprojects. Most of their work on miniprojects took place between sessions.

In the miniprojects, the students were required to design a software or software-dependent company. The design could include name, product and services, market segment, working practices, business model and value creation strategy, IP strategy, management, funding, growth, and networking and partnership strategy. Low fidelity prototyping techniques (design sketches, mock-ups, mashups) could be used to communicate the central visions, products and services of the company.

We use these written reports as our main empirical data in this paper. We analyzed them looking for paradigmatic fit with the two paradigms.

## Analysis and results

A total of eight miniprojects were made as part of the course. Four of these miniprojects discussed new ventures based on software technologies, while the rest addressed existing companies or marketing existing solutions. The four ventures are analyzed in this paper.

In the table below, the miniprojects are named after stars in the Cassiopeia constellation. Cassiopeia is also the name of organization hosting the embedded incubator at our department.

All four projects are firmly within the Consider-Do-Adjust paradigm suggesting that the preferred paradigm for software development is indeed also preferred for

designing software ventures. All of them use evolutionary business planning; all propose bootstrapping for funding; all rely heavily on partnering and collaboration; all apply and open approach to intellectual property; and all aim for short time-to-market.

| | **Caph** | **Schedar** | **Rucha** | **Segin** |
|---|---|---|---|---|
| **Business idea** | Consulting: Model-based verification | Personalized search engine | Smartphone control for intelligent homes | Consulting: Model-based verification |
| **Description** | Verification of mission-critical software components | Enhancing existing search machines via personal criteria | Smartphone remote control for home automation systems | Modeling and verification services for safety-critical software |
| **Understanding of future** | Unpredictable, controllable via networking | Unpredictable, controllable via technology | Unpredictable, controllable via partnering | Unpredictable, controllable via networking |
| **Attitude to market** | Create a group of core customers | Viral via superior free services to users | Offering competitive edge for partners | Create a group of core customers |
| **Attitude to technology development** | Based on founder excellence | Based on founder excellence | Combining technical know how and domain knowledge | Based on founder excellence |
| **Role of business planning** | Evolutionary | Evolutionary | Evolutionary | Evolutionary |
| **Attitude to change** | Presuming changes in market and technology | Rapid technological change, leading edge strategy | Rolling wave planning | Presuming changes in market and technology |
| **Funding approach** | Bootstrapping | Bootstrapping | Bootstrapping | Bootstrapping |
| **Approach to others working in the same areas** | Collaboration, networking | Partnering and competing | Collaboration and networking | Collaboration, networking |
| **Approach to intellectual property** | Academic collaboration, sharing | Academic collaboration, sharing | Open source, sharing | Commercializing university-based research, open-source development |
| **Partnering and networking** | Core customers and academic collaboration | Partnering with a leading search machine company, academic collaboration | Partnering with companies in the building industry | Core customers and academic collaboration |
| **Time to market** | Short and incremental | Short and incremental | Short and incremental | Short and incremental |
| **Paradigm** | CDA | CDA | CDA | CDA |
| **IT knowledge used** | Research-based | Research-based | Applied | Research-based |

## Evaluation and conclusion

The limited number of cases in this study obviously calls for caution in our evaluation and indeed our conclusions, but the familiarity among the software and

entrepreneurship paradigm seems to be welcomed by our students as all chose to base their ventures on the CDA paradigm.

It seems likely that incremental business development will suit incremental solution development well; i.e. that the experiences from agile software development will influence agile business development, and vice versa.

Three out of four projects are clearly related to ongoing research projects. This may indeed help reduce a problem we have had up to now. Most of the start-ups we have seen in our embedded incubator have been based on *application* of software know-how to various domains. *Research-based* start-ups have not been as common as could be desired.

Our students seem to favor open source, sharing, and collaboration. They are used to this way of working from the university setting, and a paradigm close to how the students work in their studies may help making research-based ventures less daunting to prospective entrepreneurs.

## Acknowledgement

We gratefully acknowledge *Fonden for Entreprenørskab* for supporting the course and this paper.

## References

Beck, K., M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas. “Manifesto for Agile Software Development.” http://www.agilemanifesto.org (accessed June 23, 2011).

Beck, Kent. 2000. *Extreme Programming Explained: Embrace change*. Reading, MA: Addison-Wesley.

Dollinger, M.J. 1999. *Entrepreneurship: strategies and resources*. Upper Saddle River, New Jersey: Prentice hall.

Kuratko, DF, and RM Hodgetts. 2004. *Entrepreneurship: Theory, Process and Practice*. Mason, Ohio: Thomson.

Larman, Craig. 2004. *Agile and iterative development : a manager's guide*. Agile software development series. Boston: Addison-Wesley.

Niefert, Michaela, Georg Metzger, Diana Heger, and Georg Licht. 2006. *Hightech-Gründungen in Deutschland: Trends und Entwicklungsperspektiven*. Studie im Auftrag von "Impulse" und Microsoft Deutschland, Mannheim. ZEW - Zentrum für Europäische Wirtschaftsforschung GmbH.

Sarasvathy, Saras D. 2008. *Effectuation: elements of entrepreneurial expertise*. Cheltenham, UK ; Northampton, MA: Edward Elgar.

Schwaber, Ken, and Mike Beedle. 2002. *Agile software development with scrum*. Upper Saddle River, NJ: Prentice Hall.

Takeuchi, Hirotaka, and Ikujiro Nonaka. 1986. The new new product development game. *Harvard Business Review Harvard Business Review* 64, no. 1: 137-146.

Wiltbank, R, N Dew, S Read, and SD Sarasvathy. 2006. What to do next? The case for non predictive strategy. *Strategic Management Journal* 27, no. 10: 981-998.