Appendix

Software development in startup companies:

The Green-field Startup Model

Contents

List of Figures

List of Tables

Appendix

A.1 Case study details

This appendix presents the *interview package*[1] used during the case study, dividing the description according to the research design process. Moreover we present all the collected raw codes and categories. Finally we present the engineering elements identified by practitioners in support of the categories defined in the GSM model.

A.1.1 Interview package content

In this subsection we briefly present the content of the interview package. The interview package has been incrementally improved and structured to represent the output of all the activities involved in the interview design process. The package, which contains a set of artifacts (38) clustered in 9 directories, is available for download on Github [1] under MIT licence[2]. The directories accompanied by their identifier are listed below:

- TMPL - Templates.
- SUP - Support material.
- TC - Topic cards.
- CLIST - Check lists.
- HLIST - Hand lists.
- TOOL - Tools.
- FUP - Follow-up.
- RC - Recordings.
- TAN - Data Triangulation.

First of all we created general *templates* to conduct and record information of the sampled companies, shown in Table A.1.

[1] Published at: `https://github.com/adv0r/BTH-Interview-Package`.

[2] MIT license is a free software license (information available `http://opensource.org/licenses/MIT`).

ID	File Name	Description
TMPL.0	Companies Overview	Startups' information related to: name, category, website and brief description of their main product or service.
TMPL.1	Cold Call Script	Dynamic Script with verbatims and reaction, to use during phone calls with companies.
TMPL.2	Ack Mail	Request for interview template email for informing candidate startups about our research and our interest.
TMPL.3	Thanks Mail	Mail to send to startups immediately after the interview.
TMPL.4	Reward Mail	Mail to endorse rewarding to startups after they filled the follow up questionnaire.
TMPL.5	Web call-for-interview	Web-page to inform about the interview and allow spontaneous participation.

Table A.1: Interview Package - Templates

The *support materials* aim to help researchers in executing the interviews, providing tools to control the process and structure the interview conduction (see Table A.2).

ID	File Name	Description
SUP.0	Package Content	List of all available documents to conduct the interview.
SUP.1	Whiteboard Wireframe	Wireframe of the interviews workflow with a graphical representation that can be followed on a whiteboard.
SUP.2	Definition Vocabulary	List of definition of engineering terms used during the interview, constantly updated to ensure consistency.
SUP.3	Company Info	Condensed summary of information about a startup to prepare before the interview containing: respondent and company data; developed product features and qualities.
SUP.4	Package Usage	Model representing the usage timeline of each artifact present in the interview package.

Table A.2: Interview Package - Support Material

Table A.3 lists all the *topic cards* we used during the interview. These artifacts represent the actual script which has been followed in the execution of the interviews.

ID	File Name	Description
TC.0	Kick-off Card	Verbatim to start the interview session introducing the interviewers, clarifying the execution details and a little disclaimer.
TC.1	Opening Questions Card	Script describing the first questions to ask for making the interviewee comfortable.
TC.2	Feature Elicitation Card	Questions to quickly elicitate the main features of the companies product.
TC.3	Non Functional Card	Questions to elicitate the main quality aspects considered during the development process.
TC.4	Process Card	Script to elicitate if standard processes and methodologies has been considered during development and to keep track of topic covered during the interview.
TC.5	Requirements Card	Script to elicitate main methods, tools and measures used during requirements specifications.
TC.6	Analysis Card	Script to elicitate main methods, tools and measures used during analysis of critical parts of the project and feasibility assessment.
TC.7	Design Card	Questions about the high-level architecture decision and low level design choices made during the development process.

Table A.3 – *Continued on next page*

Table A.3 – *Continued from previous page*

ID	File Name	Description
TC.8	Implementation Card	Questions to elicitate the implementation methods, tools and measures.
TC.9	Testing Card	Questions to elicitate the methods, tools and measures for validating and verifying the developed product.
TC.10	Deployment Card	Questions to elicitate methods, tools and measures during the deployment of the product.
TC.11	Cool-down Card	Script for thanking the respondent for his participation and recalling to answer the follow-up questionnaire.

Table A.3: Interview Package - Topic Cards

In order to have control over the engineering elements we created a checklist of well known practices, tools and methodologies which, the company might have used (see Table A.4).

ID	File Name	Description
CLIST.0	Practices List	List of best practices in software engineering.
CLIST.1	Tools List	List of the most used tools by practitioners.
CLIST.2	Methodologies List	List of the most recent software development methodologies described in software engineering.

Table A.4: Interview Package - Check List

Moreover we packed lists to provide to practitioners possible engineering elements they might have used but that were not mentioned during the interviews (see Table A.5).

ID	File Name	Description
HLIST.0	Qualities List	List of qualities attributes as listed in ISO 9126.
HLIST.1	SE Artifacts List	List of well-known engineering artifacts most used from software engineers.

Table A.5: Interview Package - Hand List

We created a tool for managing the interview conduction (*Slider.app*), and we made use of *Prezi* to visualize the workflow of the entire interview (see Table A.6).

ID	File Name	Description
TOOL.0	Slider.app	Web-application for conducting the follow-up questionnaire online and collecting data.
TOOL.1	Prezi.com	Web-tool for conducting the interview remotely and visualizing the workflow and lists.

Table A.6: Interview Package - Tools

Table A.7 presents the artifact which were used to capture results using an online questionnaire.

ID	File Name	Description
FUP.0	Questionnaire	Follow-up questionnaire with both free text form and likert scale question to rate items mentioned during the interview.
FUP.1	Result Summary	Short report of results obtained from the interview.
FUP.2	Repertory Grids	Grid of methods and engineering artifacts retrieved from the interview as elements for scaling..

Table A.7: Interview Package - Follow-up

Table A.8 shows the structure of all the recordings that were stored during the interviews.

ID	File Name	Description
RC.0	Audio	Audio records of the interview.
RC.1	Notes	Written notes obtained during the interview
RC.2	Whiteboard	Workflow of main topics (product, quality, process) discussed during the interview.
RC.3	Others	Other obtained records not mentioned before.

Table A.8: Interview Package - Recordings

In some circumstances we were able to triangulate data with provided artifacts after the interviews (see Table A.9).

ID	File Name	Description
TAN.0	Provided Artifacts	Artifacts discussed during the interview, then delivered by the startups to the interviewers.

Table A.9: Interview Package - Triangulation

We encourage anyone who has interests in pushing this work forward, to fork the repository and contribute to it. If used in a research context, please inform us, in order to be able to track where and how the interview package has been used.

A.1.2 Interview questions

During the interview design process we prepared scripts to use as guidelines in conducting the interviews. We embedded them in the previously described *interview package*, and in Table A.10we present the list of questions grouped by *topic card*.

1. Opening questions	
ID	Question
Q.0.0	When was (company name) founded?

Table A.10 – *Continued on next page*

Table A.10 – *Continued from previous page*

Q.0.1	How was the initial team composed? And now?
Q.0.2	What was your role initially?
Q.0.3	How long did it take to release (product-name) to the public the first version* of your product?
	2. Features elicitation
ID	Question
Q.1.0	Is (product-name) (company name) first product? (yes/no)
Q.1.1	Does (product-name) still represent good part of your current core business? (yes/no)
Q.1.2	Can you briefly describe it? What does it do?
Q.1.3	Could you help me to write a list of the main features of this product, briefly, on this whiteboard? Let's try to write essential functionalities (around 3-5) of your system.
	3. Non-functional attribute
ID	Question
Q.2.0	You have a (product-type) in place ... I guess that in this case, is important to (some-important-non-functional-aspect) . Am I right?
Q.2.1	Why was this important/unimportant?
Q.2.2	How did you realize it? (ask for each quality he mention as important)
	4. Process
ID	Question
Q.3.1	Did you use any specific development methodology?
Q.3.2	Did you use any project management process? How do you schedule the progress of your project?
Q.3.2	If any,who was the manager? (critical decsion)
Q.3.3	What were your tipical working hours?
Q.3.4	Did you had any pressure for deliverying the product fast?
	5. Requirements engineering
ID	Question
Q.4.1	Where does the idea behind (product name) come from?
Q.4.3	Did you discuss the idea with other founders/team-members? Any other stakeholders involved in the discussion? Did you document it?
Q.4.3	Have you formilized the behaviour/requirements you wanted to implement in the first release? How?
Q.4.4	Did you structure/organize the requirements?
Q.4.5	What happened when requirements were changed, added or deleted? How did you manage them? Any tools?
Q.4.6	Did you trace functionalities/requirements during subsequent activities?
	6. Analysis
ID	Question
Q.5.0	Did you analyzed what were the main challenges of the project from the development perspective and how to mitigate them? Did you document it?
Q.5.1	Did you consider the skills and time needed for realizing the project? Did you document it?
Q.5.2	Have you applied any measure to assess feasibility? And potential risks? "What if" analysis?
Q.5.3	Any particular considerations for critical requirements?
	7. Design
ID	Question
Q.6.0	Have you considered any design architecture before start implementing the actual code? If no: have you structured your system in different parts? How do they interact?
Q.6.1	Have you created models or diagrams for documenting those decisions?
Q.6.2	Have you considered measuring your architectural decisions? Such as maintanability effort required to change a component of your system.
Q.6.3	Have you considered well-know standards to adopt (such as design or architectural patterns)? Did you document them?
Q.6.4	Have you utilized any particular tools for designing your system?
	8. Implementation
ID	Question
Q.7.0	How did you approach coding the first days? You had the idea, you had the requirements and the design... and then?
Q.7.1	Have you considered any workflow guidelines before or during the coding phase?
Q.7.2	How did you divide the work between team-members?
Q.7.3	How did you manage the code base?
Q.7.4	What documentation have you produced during coding?

Table A.10 – *Continued on next page*

Table A.10 – *Continued from previous page*

Q.7.5	Have you considered any configuration process for the development environment? Did you document it?
Q.7.6	How did you manage issues and bugs?
Q.7.7	Did you monitor aspects of your development such as team productivity, size/complexity ... ?
Q.7.8	Which are the tools that most have helped you producing code?
Q.7.9	What programming language did you use?
	9. Testing
ID	Question
Q.8.0	Did you perform any kind of tests for the implemented code? Such as acceptance, unit, integration and system tests?
Q.8.1	When did you write the tests? Are they documented?
Q.8.2	Quality assurance was an important concern? (to deduct also from the discussed qualities)
Q.8.3	Have you conducted any verification and validation process? Did anyone tried your product before the first release? Any reports and analysis of results?
Q.8.4	Have you conducted any measurement for assessing the validation and verification results?
Q.8.5	Have you utilized any specific tools for performing testing?
	10. Deployment
ID	Question
Q.9.0	How did you deploy your project? Are you about scalability?
Q.9.1	Have you utilized any specific tools?
	11. Closing questions
ID	Question
Q.10.0	Have you considered improving the development process (in terms of efficiency and effectiveness)?
Q.10.1	Have you experienced a drop-down performance during the development, if any? What could be the reason and when did it happen?
Q.10.2	What is the most valuable improvements you would apply with perfect hindsight?

Table A.10: Grounded Theory - Interviews guiding questions

A.1.3 Interviews - Open coding

In this section we present all the codes conceptualized during the *open coding process*[3]. The following tables shows the 630 which contributed in the formation of the GSM model, divided by thematic area.

[3]During the process we used tags such as *SUGG* and *ERR* to differentiate the conceptualization between what was the current state-of-art and gathered hindsight suggestions for the future or mistakes occurred in the past.

Code stats - Opening Questions

Code
Small team
Recently Founded
Technical Respondent
Technical Founders
Short Product Building Time
Web and Mobile Product
Web Application
Tech/Biz Founders
Very short product building time
Web and desktop

Table A.11: Opening questions

Code stats - Product Priorities

Code
Growing Team
Minimum and essential set of functionalities is important
Limited budget
Automatization of deployment
Tradeoff: Quality and Time-to-Market
UX
Assess usability with user feedbacks is important
Development speed is the most important
Time pressure
Entrepreneurial attitude
Interoperability
Not paying attention to non-functional aspects makes development faster
Value to the user is the most important
Being fast to learn what brings value to user
Delay choices which could limit technological flexibility
Delicate balance between code maintainability and development speed
Desire to roll out something new as fast as possible
Ease of use
ERR: Not using a framework to avoid learning, consequences with low maintainability
Framework to Improve Maintainability
Metrics for assess Usability
Quality was not a priority in early stage
Short Time-to-Market is the main focus
SUG Build scalable product/infrastructure from data 1
SUG Technology is not enough
Usability
Usability more important than functionalities
Build scalable product/infrastructure from day 1
Compliance with third party components
Ease of use (Product usage without interruption) is important
ERR: Not considering UX from day 1 led to unsatisfied initial users
ERR Technical issues for scalability
Idea conception from personal problem
Internal Deadlines
Lack of scalability
Lack of usability expertise
Mobile application compliance with Apple standard is important
Performance
Pivoting from initial prototype
Poor performance drive users away from the product

Table A.12 – *Continued on next page*

Table A.12 – *Continued from previous page*

Code
Portability from web to mobile
Portability in mind from day 1
Proof of concept by prototype
Prototyping for assessing efficiency
Prototyping for assessing usability
Reliability important for infrastructural product
Ruby on Rails increase Maintainability
Scalability realized soon based on past experience
SUG Enhance scalability
Talk with customers
Tradeoff: Effectiveness more important than UX
Tradeoff: High portability and UX for Mobile Apps
Tradeoff: Quality vs. Budget
Tradeoff: Usability functionality
Analysis for communication of different technologies
Attend startup events to prove idea/concept/prototype validity
Close friends feedbacks for increasing usability
Cultural usability expectation
Customer giving spontaneous feedback
Dedicated person for UX portability to Android
Efficiency
Efficiency to enhance UX
Efficiency will emerge only when using a prototype
Find a mentor in early stage
First prototype to solve personal problem
Focus group to assess usability (with potential users)
Hire more people
Interoperability studies beforehand (with third parties)
Lack of people
Lack of reliance on third party services
Limited hw resources
Maintainability becomes important when complexity and size increases
Maintainability enhance testability
Medium Product Building Time
Minimum level of maintainability (decent) from day 1
Mobile application portability covered mainly with iOs and Android
Mobile MVP hard to get accepted on Apple Store
Mobile portability deals with different OS
Mobile portability is important
Mobile/web portability first version using HTML5 to save time
Not perfect usability is not critical
people are usually more attracted by interfaces
Portability of Web application is Browser Compliance
Product changes quickly
Product is not security-critical
Reliability becomes a problem when users increase
Reliability for web-consumer product is not important
Reliability in early stage is not important
Remote team (non co-located)
Scalability issues solved using Elastic Infrastructures (EC2)
Scalability vs UX
Social network integration boost usability
Sometime startup fail because over-engineer the product before launching
Starting as side project cause low attention to maintainability
Studying competitors to differentiate
SUG: Maintainability from day 1 is easier than huge in late stage
SUG: Scrum helps in clarify product vision
SUG: Scrum helps in visualize project progress with the team
SUG Solid product infrastructure increases confidence in the product
SUG system oriented infrastructure instead of monolithic app
SUG working prototype is essential for fund raising
Technical background hinder usability

Table A.12 – *Continued on next page*

Table A.12 – *Continued from previous page*

Code
Tradeoff: Mobile native apps takes longer but more Effective (UX, Efficiency)
Tradeoff: Working overtime → quality code
Tradeoff: Development speed vs Reliability
Two respondents together
Usability assessment cannot be based only on feedback (users do not realize some design detail)
Usability at first designed with personal experience
Usability essential for games
Usability important when founders background is design
Usability improved by studying cutting-edge examples
Usability outsourcing
Users intensively using a system needs a high UX
UX background
Working overtime decreases productivity in long run
You have to find the balance between UX and functionality
Lack of scalability because of budget
Lack of scalability because of people
SUG working product is essential for fund raising

Table A.12: Product priorities

Code stats - General Process

Code
Respondents claim to use light version of scrum partial principles implemented
Communication facilitated by small team size and co-location
Hacker culture
Started from pre-developed technology / prototype
Automatization of deployment
Collaborative online tools for task management
Co-located Team
Critical decision taken by the CEO/CTO which have global overview of the project
Developers usually work overtime to meet goals
Development speed is the most important
Fast and Informal Development approach
Short and flexible iterations ($\lesssim$ 2 weeks)
Time pressures from investors
Agile methodologies are not effective with one/two person teams
Developers are self-organized in choosing tasks
Entrepreneurial attitude
Evolutionary MVP approach helps bring value to customers
Feedback from users is a priority since day 1
Growing team requires more control / management on initial chaos""
Having a deadline for final releasing is necessary to set a limit to improvements
Idea originated from hole in the market
MVP evolutionary approach to (software) development
New features idea proposed by CEO
Reduce wasting time on specifications and analysis and focus on code
Simple products doesn't require formal process
Virtual kanban board style to manage the user stories
Being fast to learn what brings value to user
Delay choices which could limit technological flexibility
Desire to roll out something new as fast as possible
Feedbacks collected via email
File sharing between team-members using online tools
Founder background as scrum-muster drove his attitude towards software development
Informal internal deadlines
Post-its for task management
Small milestones ($\lesssim$ 2 weeks) help developers' awareness of project progress.
SUG: Limit planning because plans will be subject to changes, no matter what
SUG Technology is not enough

Table A.13 – *Continued on next page*

Table A.13 – *Continued from previous page*

Code
Build scalable product/infrastructure from day 1
Co-located working environment does not require tools for know who was doing what
Collaborative online tools for project management
Developers satisfaction is influenced by how the sprint goal is achieved/not achieved
Development team priorities in contrast with company operations
Discussing ideas informally
ERR: Low experience with project management -¿ No schedule
ERR: A key developer acting as manager is much less effective
Flat organizational structure
Focusing on one task at the time improve productivity
Github for managing the code base
Idea conception from personal problem
Implemented some principle of Lean Startup
Internal communication tools, simple chat, are very important for communication
Internal Deadlines
Knowledge sharing with online dedicated tools
No process
Not able to keep documentation updated during the process (lack of time)
Pivoting from initial prototype
Plan with a rough spreadsheet is sufficient
Post-it
Project management (tools) become necessary with growing team and product complexity
Project management is not needed because of co-located work env
Stand-up meetings are simple and worthwhile
Stand-up meetings to discuss task assignment of the day
SUG: for non-collocated teams more prescriptive guidelines needed (process)
SUG: Outsourcing in Low-wages countries helps saving budget
SUG: Scrum can work in the context of startups given that one person has done it before
SUG: Too many tools negatively affect speed
Talk with customers
Team progress was not measured
Time pressure from media coverage (hype)
Uncertainty
Very short time between idea discussion and feature implementation
Whiteboard for task management
Whiteboard to monitor project progress
A team which worked together in the past develops ad-hoc approaches
Agile helped receiving quickly user feedback
Agile Practices such as TDD and PP can cause overhead
Analysis for communication of different technologies
Attend startup events to prove idea/concept/prototype validity
Automated source control tools when team grows
Avoid formalities to release faster
Back-end skills is important for developers in startups
Build fast (asap) without schedule
Comfortable environment negatively affect productivity
Community manager is a very important figure to manage large number of users
Cross functional teams (full stack developers - front-end backend - mobile and web)
Customer Development helped in identifying the market niche
Customer dictated usage scenarios,and developers team extract functionalities
Customer giving spontaneous feedback
Developer responsible for designing, coding and testing a feature
Didn't executed radical pivot of main features
E-mails for communication on distance
ERR: Bug fixing process not integrated in Scrum
ERR: Lack of documentation can lead to poor understanding of the system
ERR: Little or no software development experience
ERR: Losing precious time engineering the product without releasing early
ERR: Scrum without expedite-lane for dealing with emergencies was problematic
ERR: When all developers can modify the kanban wall (add move remove) confusion arises
Feature Meeting when necessary to discuss new features
Final integration (one-time) when single parts completed

Table A.13 – *Continued on next page*

Table A.13 – *Continued from previous page*

Code
Find a mentor in early stage
First prototype to solve personal problem
Good impression on first reference
Graphical design tool
Growing team led to necessity of more meetings
Growing team require tasks assignment
Having a user manual is important
Highly experienced developers
Highly experienced developers made the process faster
Hire more people
Idea started as a side project
Impediment board helps in solving bottlenecks
In the first stage of a startup team productivity is not essential (other priorities)
Informal Meetings
Instagram founders were working.
Introducing a process decreases productivity
Iteratively adjust product tracking metrics
Measure conversion rate of landing page (desktop application)
Meeting for discussing new feature ideas
Show-and-tell to informally share weekly achievements are very useful to boost motivation
Mobile MVP hard to get accepted on Apple Store
No formal schedule
No pressure (time, budget, investors)
No Project management tool
Overhead of request hard to handle after successful launch
Overhead of request hard to handle when tired
Pair-programming helps with new hires
Pivoting from b2b to b2c
Process is not necessary until you start collecting real users feedback
Process perceived as limitation to speed
Product backlog for monitoring project progress
Product changes quickly
Proximity to release brought fear for credibility which led to more formalities
Release step-by-step to an increasing number of friends/user before open the product
Rigid Macro-Milestones, flexible internal smaller milestones
Rigid Weekly Sprints (Scrum) when pressure higher
Scrum works better with skilled and experienced people
Scrumboard (simplified) with only a few boards
Simple products does not require project management tools
Solo developer
Sometime startup fail because over-engineer the product before launching
Source of pressure because they were using the product for their work
Starting as side project cause low attention to maintainability
Stories / Features did not need any prioritization (all necessaries)
Studying competitors to differentiate
SUG: A mentor in the early stage is really useful
SUG: Be flexible in apply only useful Agile practices
SUG: Make some beforehand analysis to define data structure
SUG: Scrum brings advantages: team-building, responsibilities, better code quality
SUG: Scrum practices for software development, together with Lean Startup methodology
SUG: Shorten iteration coding time (from 5d to 3d) dedicate 2 days / week bug fixing
SUG: Small engineering teams (3 persons) are quick, adaptive and responsive
SUG: The only possible approach for startups is using Lean Startup Methodology
SUG: Use a physical kanban wall when co-located
SUG: Use kanban wall to track team velocity
SUG: Working overtime on the code, makes poor code quality
SUG For non-co-located teams more documentation is needed
SUG Solid product infrastructure increases confidence in the product
SUG system oriented infrastructure instead of monolithic app
SUG working prototype is essential for fund raising
SUG: many small customers bring more value than a single big company
SUG: Setting deadlines increase productivity

Table A.13 – *Continued on next page*

Table A.13 – *Continued from previous page*

Code
Team with no or small working history
Technical founders are aware of development team necessities more than non-technical managers
User documentation
Users feedback not necessary for the simple and specific application type (well-defined)
Workflow driven by user feedbacks, new ideas and necessities without scheduling
Working overtime only in the first phase
Young Employees

Table A.13: General Process Codes

Code stats - Requirement Engineering

Code
Dissemination of the idea with informal discussion and support of tools (whiteboard, paper, views)
Automated tools for collecting TODO lists, Requirements,User Stories
Started from pre-developed technology / prototype
Critical decision taken by the CEO/CTO which have global overview of the project
Estimations based on developer experience
Manual tools (whiteboard / paper) for collecting stories
Feedback from uservoice
Idea originated from hole in the market
New features idea proposed by CEO
Stories/features prioritization using personal experience and user feedback
Work break-down in smaller stories to let developers work independently
Idea refinement through networking
Lack of Requirement Specification Documentation
Trello
Automated tools for managing stories/features are necessary when complexity grows
Developers can manage user stories independently (create and assign)
Prototype (rough) to explain features and share vision among team members
Specification written from user perspective (stories)
Stories / List updated throughout the process via automated tools
Stories/features traceability via version control
Whiteboard not updated throughout the process (lack of time)
A prototype can substitute stories / feature documentation
Ability to write detailed specification from day one, thanks to prev. experience
i've never seen a document...
Clarify product vision through important use cases only
Co-working space limit whiteboard utility
Cross divide technology (HTML5) helped building faster prototype
Customer dictated usage scenarios,and developers team extract functionalities
Dissemination of the idea with informal discussion via email
ERR: Don't ask early customer feedback
Features / stories collected in an informal magazine (the startup product itself)
Feedback from prototype to drive user stories / features
Idea coming from prev. founder working experience (hole in market)
Idea refinement through focus group
Idea started from authority request for application open
Lack of trust in customer's feedback (next)
New features idea through focus group
New stories collected and prioritized during weekly meeting discussion
One developer assigned to one task at the time (generally)
Paper prototype (wireframes) to demonstrate the views
Physical wall for task / todo / user stories
Product based on a Contract with public authority (Requirements fixed)
Product support three family of users
Scrum board to manage stories (physical) with post-it notes
Simple semi-automated todo list software (google spreadsheet) to collect stories/features/feedback
Started with more structure and process that increasingly degenerated in chaos/spaghetti
Stories / feature traceability using skype logs

Table A.14 – *Continued on next page*

Table A.14 – *Continued from previous page*

Code
Stories / featured prioritization with 3 simple labels (urgent, todo,ideas) was enough
Stories / Features did not need any prioritization (all necessaries)
Stories / Lists used initially and not really updated
Stories/feature prioritization on a whiteboard is much better than Electronic tools
Stories/Feature prioritization with the help of tools (categorize etc)
Stories/features captured with top-down approach
Stories/features prioritization analyzing which were useless to pursue the vision
Stories/features prioritization documented side to side with business plan
Stories/features prioritization through informal discussion
SUG: Use Customer Development approach to collect user feedback before implementation
Team formation during events (startup weekend)
User feedback to choose among different UI
User views mockups represented the feature list (product as a flow)
Video on Landing page to explain features to users and share vision among teams
Visualize feature idea through mind-mapping tools
Whiteboard to clarify product vision is excellent

Table A.14: Requirement Engineering Codes

Code stats - Analysis

Code
Analysis not important, partially replaced by informal discussions
Analysis of feasibility not important because past experience/knowledge with similar domains
For non-core functionalities user feedback outperforms a formal analysis
Simple product does not require analysis for expert developers
Small informal analysis to make important technological decisions
Competitor informal analysis to investigate improvements to tackle
Using well-known, traditional, tested technologies foster team performance
Analysis does not work with innovative products never done before
Analysis of interoperability with critical third party components
Analysis replaced by experimenting critical technologies before implementation
Analysis replaced by exploring solutions within developers community (ruby gems)
Analyzing competitor feedback to understand what to improve
Create a document to show customers the difference from competitors
Decisions taken with recorded brainstorming for traceability
Disruptive technology
Evaluated and documented pros and cons of decisions made
Evolutionary Prototyping substitute analysis for innovative product
Facebook is still using mysql...
Specifying the product features precisely beforehand makes development cycle shorter
Let users evaluate between different front-end functionalities alternatives
Third party newsletter for external third party APIs updates
Transposed feature list to use cases (Analysis)
Underlying technology (third party) changing quickly (Interoperability) do not allow Analysis
User feedback to decide application name
Using well-known, traditional, tested technologies makes hiring developers easier

Table A.15: Analysis Codes

Code stats - Design

Code
Hacker culture
Well known architectural framework for web application (MVC)
Document communication among components (high-level)
Maintainability through de-coupled modules (modularization)

Table A.16 – *Continued on next page*

Table A.16 – *Continued from previous page*

Code
Design of the architecture conducted through informal and poorly documented discussions
Design of the architecture not documented at all (based on personal experience)
Designed the data-structure
Framework reduce the need of documentation (well-known and structured)
General high level mock-ups of views was the only documented design
Tradeoff: Engineering the product/process vs. Flexibility in first phases
Delay choices which could limit technological flexibility
There is no time to keep documentation updated out-of-the-code
Clear and standard code minimise the need of a design documentation
ERR: Not having an initial design led to problems in later phases
Initially defined modules and communication to enhance efficiency
Not able to keep documentation updated during the process (lack of time)
Simple product do not require formal design (replaced by naive diagrams)
Technical debt
UML complex diagram replaced by naive diagrams
With extremely small teams architectural choices do not need to be documented
Academic background led to more formal design diagrams
Automated tool for managing the product architecture (MVC)
Automatic tools for documentation
Co-location and the high communication volume makes design documentation un-necessary
Design informally made using whiteboard between engineers helps a lot
Electronic tools for UI design
ERR: Academic background led to standard formal UML-like diagrams useless for early stage startup
ERR: Lack of initial analysis/design of data structure led to overhead later on to fix problems
ERR: Small mistake in initial data structuring led to bad consequences
ERR: Traceability of decision taken was a problem
Growing team lead to necessity of refactoring
Growing team lead to waste and trash code (ERR)
Lack of documentation can sustain a growing team at 1-2 employee at the time through training
Pair programming helps in the first phase when structuring the application
Reflect the code structure in the UI using different colors for different user-category
Simple product do not require architectural design (replaced with evolutionary prototyping)
Specification for critical communication between components
Structure the code-base differentiating by user group
SUG: Document at least communication among components
SUG: The fastest MVP prototype is a piece of paper with the view in front of real users
SUG: When working remotely, even with past experienced teams, documentation is important
SUG: Remote teams, design only the final definitive mock-up, to save communication time
SUG: When working with remote teams design documentation is important
SUG: Wireframes will not reflect the actual outcome (limited utility)
SUG: not sure about choice of framework
Tradeoff: Time pressure leads to lack of documentation
Tradeoff: Trash code when growing instead of Write quick code in the beginning
Using ready open source components for UI elements made development faster
With evolutionary prototyping there is no design phase (in the waterfall sense)

Table A.16: Design/Architecture Codes

Code stats - Implementation

Code
Collaborative online tools for task management
Productivity metrics are ignored
Chat tools for internal communication and traceability
Clear code does not need in-text comments
Comments inside the code when necessary
Critical decision taken by the CEO/CTO which have global overview of the project
Estimations based on developer experience
Git / GitHub as version control system for the code-base
Github for issue / bug management

Table A.17 – *Continued on next page*

Table A.17 – *Continued from previous page*

Code
Developers are self-organized in choosing tasks
Framework reduce the need of documentation (well-known and structured)
Growing team requires more code documentation
Project management tool for issue / bug management
Task are assigned by CEO/CTO on personal experience
Virtual kanban board style to manage the user stories
Work break-down in smaller stories to let developers work independently
Extremely small development team did not require Version Control System
Git for branching/merging useful for the codebase
Refactoring the code only when stricly necessary
SUG: Treat issues/bug and user stories/new features equally (same board)
Trello
Well-known framework for the product (RoR)
Code metrics ignored (complexity, size, etc ...)
Documentation perceived as a waste
Framework (RoR) easy to learn and with big advantages
Growing team requires use of Version Control System
Hero developer helped in meet deadlines
IDE
Lack of experience caused some re-work
Need of documenting the code is bad code smell
No workflow / guidelines for implementation
Php
Refactor as-you-go
Team productivity measured naively looking at closed tickets
Versioning system not necessary when no-overlapping between developers' concern
Begin implementation with internal APIs
C++ (desktop application)
Chose technologies familiar to developers (language and framework)
Code documentation only for long-lasting pieces of code
Code n fix.
Code standards
Developers can decide which bug to work on
Critical bugs fixed immediately (no need of issue tracking system)
CSS3
developed locally, tested locally
Developer environment consistent with production environment
Documentation not necessary with clean code Documentation, even in-code, is waste
ERR: Time wasted for not analysing existing technical solutions (libraries)
Github useful for code-reviews
Github useful to see who is doing what
Growing teams require to trace who is doing what
HTML5
Internal APIs to improve portability
Jira used for issue / bug management
Mercurial for version control system
Minimal set of tools for code implementation (electronic board + chat + version control)
New hires training encourage developers to refactor the code
No need of task system (no even manual) thanks to hero developer
Node.js
Non relational database
Pair programming help communication between developers
Paper for issue / bug management
Process perceived as killer for creativity
RoR helps in managing code
RoR for scalability
SUG: Chose the technology according to the nationality of developer you want to hire
SUG: Github useful to see who is doing what (growing team)
SUG: Given good experience TDD is best way of writing software
SUG: Trello for issue management (and task) is efficient
SUG: Version control system automatic tools are well integrated and do not cause overhead
SUG: When working with remote teams, let one person decide tasks assignments

Table A.17 – *Continued on next page*

Table A.17 – *Continued from previous page*

Code
Tickets (stories) not useful in the very early stage (one big story implement the product")"
Track for bugs in the beginning, then stopped.
Version control system using project management tool integration

Table A.17: Implementation Codes

Code stats - Verification and Validation

Code
Absence of automatic testing (replaced by experience and usage)
In house validation by trying the product
Test only critical parts is enough, Secondary bug found by users
Progressively have the product used and tested by increasing number of persons refining it
Feedback from uservoice After releasing a new feature let a trustworthy set of superusers try it and report bugs works very good
Feedbacks collected via email
For non-core functionalities user feedback outperforms a formal analysis
In house validation for core features identifies critical malfunctioning before releasing it
Automatic tools to asses product usage help adjust flaws
Contacting users personally to identify malfunctioning
If code is well tested, documentation can be replaced by test cases
In web applications bug are usually client side, hard to automatically detect
Landing page helps you having feedbacks before releasing the actual product
New features manually tested every week before the weekly roll-out
Release the product to let the users report bugs
RoR test suite framework are well defined
TDD requires a paradigm shift that is easier for newer generation of developers
Testing absent because lack of knowledge
Testing almost non existing so that process can be faster
Tradeoff: Amount of tests (slow down the process) and reliability (user are fault tolerant)
Tradeoff: Testing require hiring VS. Small teams are more flexible
Tradeoff: Time-to-market more important than testing
Unit testing only when strictly necessary
Agile Practices such as TDD and PP can cause overhead
Continuous Integration testing using automated tool (selenium)
Developer responsible for testing his own code makes process faster
Development team were using the product itself for their development process
Didn't found bug in production: Lucky
Email used for bug reports
ERR: Not testing UX with real users
ERR: in the prev. project we did UX ourselves
For iOs/mobile product Apple offer automatic testing
Framework facilitate testing
Growing (remote) team require a tester
Growing team requires increasing number of tests
Growing teams are facilitated by having TDD already in place
Include a on/off switch in new features to de-activate it if something goes wrong
Inspect logs to find bugs
Integration testing executed manually sometimes
Landing page used for idea validation
Maintainability enhance testability
Scalability through testing over databases
Self explanatory tests don't need test cases (cocumber)
Simple projects does not require much testing
SUG: Customer service dedicated to collect feedback is very important
SUG: Developers are Testers and should test someone else's code
SUG: Given good experience TDD is best way of writing software
SUG: User stories itself suggest acceptance tests
TDD, not religious but only on critical parts, helps a lot
TDD helps keeping the focus on the current task

Table A.18 – *Continued on next page*

Table A.18 – *Continued from previous page*

Code
TDD requires experience
Test to assess performance/efficiency
Testing helps innovation by improving confidence in the code (not being afraid of breaking stuff)
Testing necessary when the product become more complex they still do not have testing)
The interviewee not expert in automatic testing techniques
User retention is the most significant metrics to understand how the product is working in the market
We test mainly the most used features

Table A.18: Verification and Validation Codes

Code stats - Deployment

Code
Deploy on third party infrastructure (cloud)
Deployed on Virtual Private Server (VPS)
Automatic tools for staging and deployment
Direct deploy from development machines to production
SUG: Use simple automatic tools (Chef) for managing staging and production
Deploy new features using Git merging feature branches with master
Manual deploy from development machines to staging and then production
Weekly scheduled deploy
Deploy with the help of automatic tools
Extremely frequent new deployments
Heroku speed up the process avoiding infrastructural complexity
SUG: Increasing number of users require a staging environment before deploy in production
we deploy from 5 to 20 times per day

Table A.19: Deployment Codes

Code stats - Closing Questions

Code
Enthusiasm and motivation keep productivity high
Growing team requires more control / management on initial chaos
Working history between developer facilitated execution
ERR: Create a complex and big project for long time without evaluation
Growing team makes high volume of informal communication is a problem
SUG: Project management tools, if well integrated, boost productivity
Time pressure (Beating competitors)
Find product/market fit is a priority
Product/market fit
Productivity drop-down when growing team
Company is now mature and moving towards process and structure
Customers likes small improvements (not-expected small features)
Developers fear notify ticket status (feel monitored)
Developers in startups have big responsibilities
ERR: Not estimating because lack of experience
Even in the early stage at least two developers are required
Frameworks help growing faster
Front-end developers are over-rated
Good developers willing to work in a startup are hard to find
Good technical founders should hire excellent engineers
Growing is making release time longer (fear of break things) + (Releasing more features in one pack)
Growing requires a release plan
Growing requires efficient system for managing big number of small releases
Growing requires the CEO to slowly moving away from the code
Growing will always produce a productivity drop-down

Table A.20 – *Continued on next page*

Table A.20 – *Continued from previous page*

Code
Growing will cause an initial drop-down in productivity but with following improvement of speed
In startups developers multi skilled (generalists) are more useful than gurus in one technology
It's hard to say anything before the product is released
Not having a data schema and hiring a new developer is not a big deal
Past experience with similar product make development much more effective and fast
Productivity will improve when the teams work together for some time (feeling)
Speed and flexibility is the most important factor in the beginning
SUG: Clear business direction help development to go faster and reduce wasted features
SUG: Customer development in parallel to software development is very important
SUG: Get out of the office soon (to verify business assumption)
SUG: if you can't pitch your idea in 5 seconds, something is wrong
SUG: Introduction tutor/course for new hires
SUG: Is better to have a drop-down in productivity when team grows than lose time before
SUG: Record relevant metrics from day 1 to see what's happening with the product
SUG: Release weekly
SUG: To boost motivation start think day 1 how to make profit
SUG: After initial enthusiasm, you need revenue to boost morale
The more users the more feedback to manage (grow)
Time-pressure for media coverage led to productivity boost
Tradeoff - Drop-down productivity for lack of process (win) VS. Introducing early process (not worth it)
Using a rigid process led to problem with emergencies
Very satisfied on development approach

Table A.20: Closing Questions Codes

A.1.4 Interviews - Axial coding

During the generation of the theoretical framework, categories have been grouped together, organized into a tree-like structure. At the highest level of abstraction we identified 6 macro categories, in addition to the core category, that is *"Speed up development"*, considered as the most urgent priority by the totality of the respondents. Table A.21 summarizes categories and subcategories identified in the study.

Category	Subcategory
Speed-up development	Working overtime to meet deadlines Use of standard/known technologies Development aided with well-integrated and simple tools Externalize infrastructural complexity on third party solutions Keep simple and informal workflow
Evolutionary approach	Find the product/market fit quickly Uncertain conditions make long-term planning not viable
Product quality has low priority	UX is the only important qualities Suitable and limited functionalities Efficiency emerges after using the product User is fault-tolerant in innovative beta product Cross-browser and cross-device compliance with aid of automatic tools Product should be reasonably ready-to-scale
Team is the catalyst of development	High-impact of CTO/CEO background Very small and co-located development team Developers have big responsibilities (self-organized) Multi-role and full-stack engineers Skilled developers are essential for high speed Team works under constant pressure Limited need of formalities between team-members

Table A.21 – *Continued on next page*

Table A.21 – *Continued from previous page*

Category	Subcategory
	Access to external expertise (Mentors)
Accumulated technical debt	Minimal Project Management Informal specification of functionalities Rough and quick feasibility study Lack of architectural design Lack of automated testing Tacit Knowledge replaces formal documentation
Growth harms performance	Pay off the accumulated technical debt Need of re-engineer the product Focus shifts to business concerns Company and user size grow
Severe Lack of resources	Time shortage Limited human resources Limited access to expertise

Table A.21: Grounded theory - Categories and sub-categories

In view of the relatively high complexity of the entire categories tree, the lower level sub-groups are not shown in this section, rather they are presented in detail in the GSM model.

A.1.5 Categories and engineering elements

Finally all the engineering elements identified in the follow-up results were mapped on the categories identified in the GSM model.

Engineering elements	Category	Freq.
Very useful		
Analysis of critical/important use case scenarios	Rough and quick feasibility study	1
Asking user feedbacks for little adjustments only	Find the product/market fit quickly	1
Assembla for managing tickets/tasks	Use of well-integrated and simple tools	1
Basecamp for bugs/issues	Use of well-integrated and simple tools	1
Break-down of big tasks in smaller tasks	Lack of requirement engineering	1
Build APIs to export functionalities to mobile	Limited number of suitable functionalities	1
Create ticket on-the-fly without any analysis and design	Rough and quick feasibility study	1
Customer development and Lean startup methodology	Find the product/market fit quickly	1
Deployment on Amazon infrastructure	Externalize complexity to third party solutions	1
Developer responsible for designing,coding and testing a feature.	Multi-role and full-stack engineers	1
Developing the product without having schemas/diagrams	Lack of architectural design	1
Dropbox for sharing documents (x3)	Use of well-integrated and simple tools	3
Electronic Kanban Wall for managing features (Agile Zen)	Ticket-based tools to manage stories/features	1
Evolutionary prototyping/MVP (X4)	Find the product/market fit quickly	4
Focus Group for assessing graphical aesthetic	Find the product/market fit quickly	1
Focus Group for setting the main functionalities	Find the product/market fit quickly	1
No need of formal analysis (past experience)	Rough and quick feasibility study	1
Having Mentors in early stage	Access to mentors expertise	1
Hip-chat for internal communication	Use of well-integrated and simple tools	1
Lack of detailed documentation	Tacit Knowledge	1
Lack of formal and automatic testing	Lack of automated testing	1

Table A.22 – *Continued on next page*

Table A.22 – *Continued from previous page*

Engineering elements	Category	Freq.
List of features (paper notes)	Use of well-integrated and simple tools	1
Mock-ups of the UI	Lack of architectural design	1
Naive diagrams (disposable) instead of UML communication diagrams	Lack of architectural design	1
Postpone potential choices which could "limit"	Uncertain conditions make long-term planning not viable	1
Structure the app in a self-explanatory way with Rails	Tacit Knowledge	2
Collecting initial feedbacks before coding	Find the product/market fit quickly	1
Setting informal deadlines between co-founders	Minimal Project Management	1
Short release time (weekly deployment)	Light lean startup principles	1
Sketches (wireframe)	Lack of architectural design	1
Skype for assigning bugs	Use of well-integrated and simple tools	1
Starting from a previously developed technology	Use of standard/known technologies	1
SVN for the codebase	Use of well-integrated and simple tools	1
Let user test secondary functionality	Lack of automated testing	1
Treating bugs as user stories	Minimal Project Management	1
Trying the product internally to identify bugs/issues	Lack of automated testing	1
Use Cases (Assembla)	Rough and quick feasibility study	1
User feedbacks (by means of the "super circle" of users)	Find the product/market fit quickly	1
UserVoice for collecting users' feedback	Find the product/market fit quickly	1
Using a Whiteboard	Use of well-integrated and simple tools	1
Using Linode to deploy the application	Use of well-integrated and simple tools	1
Extremely useful		
Amazon EC2	Externalize complexity to third party solutions	1
Analyzing competitors' feedbacks	Rough and quick feasibility study	1
Basecamp's tasklist	Lack of requirement engineering	1
CEO solving conflicts in development decisions	Tacit Knowledge	1
Chef for deployment	Use of well-integrated and simple tools	1
Clean-code	Tacit Knowledge	1
Co-located team members	Very small and co-located development team	1
Collecting feedback from pre-launch through landing page	Light lean startup principles	1
Competitor's analysis	Rough and quick feasibility study	1
Consulting available gems before start implementing	Rough and quick feasibility study	1
Multi-role and full-stack engineers (full stack developers)	Multi-role and full-stack engineers	1
Daily stand-ups	Keep simple and informal workflow	1
Database model	Lack of architectural design	1
Deploy workflow (feature brench-¿ pull req -¿ Capistrano)	Use of well-integrated and simple tools	1
Development experience	Skilled developers are essential for high speed	1
Feature meetings	Naive task assignment mechanism	1
Flat hierarchy of the team	Multi-role and full-stack engineers	1
Get early feedback from customers	Find the product/market fit quickly	1
Git for code-base	Use of well-integrated and simple tools	1
Github for having review of the code	Use of well-integrated and simple tools	1
Having senior developers	Skilled developers are essential for high speed	1
Heroku for deployment	Externalize complexity to third party solutions	1
High-experience developers	Skilled developers are essential for high speed	1
HTML5, CSS3	Use of well-integrated and simple tools	1
Hype of media for increasing moral of developers	High enthusiasm boosts productivity	1
Informal meetings for discussing biggest changes only	Keep simple and informal workflow	1
Initial feature list (whiteboard)	Use of well-integrated and simple tools	1
Initial survey to collect user feedbacks	Light lean startup principles	1

Table A.22 – *Continued on next page*

Table A.22 – *Continued from previous page*

Engineering elements	Category	Freq.
Mind mapping instead of text writing communication	Tacit Knowledge	1
MySQL as DBMs	Use of standard/known technologies	1
Only in-line comments to document the code	Tacit Knowledge	1
Oral communication	Tacit Knowledge	1
Personal experience for story estimation	Rough and quick feasibility study	1
Post-it notes for tracing tasks and bugs	Keep simple and informal workflow	1
Prototype an Hybrid Django/Php	Use of standard/known technologies	1
RESTful API	Use of standard/known technologies	1
Scrum board (by means of whiteboard with post-it notes)	Keep simple and informal workflow	1
Self-imposed informal deadlines (Google spread-sheet)	Minimal Project Management	1
Skype for communication	Use of well-integrated and simple tools	1
Using an MVP approach	Find the product/market fit quickly	1
Using whiteboard for main focus of the product/vision	Use of well-integrated and simple tools	1
Whiteboard for tracing the progress (modules implemented)	Minimal Project Management	1
Hindsights		
Crazy egg for ux testing	Adapt to early feedbacks	1
Ruby On Rails that forced me to use an MVC approach on development	Lack of architectural design	1
Basecamp and TODO-list	Lack of Requirement Engineering	1
Delelop and release fast each time we had a new feature.	Find the product/market fit quickly	1
High skilled team.	Informal specification of functionalities	1
I wanted to focus more attention on designing the interface and the analysis of the needs of end users (the tourist)	Find the product/market fit quickly	1
90% of developers were full stack	Multi-role and full-stack engineers	1
The excellence of the whole technical team	Skilled developers are essential for high speed	1
Continuous deployment	Find the product/market fit quickly	1
Also UX/CPO/UI could code	Multi-role and full-stack engineers	1
Developers and the heterogeneity of knowledge	Skilled developers are essential for high speed	1
Motivation to innovate all played an extremely important role	Multi-role and full-stack engineers	1

Table A.22: Questionnaire results to theoretical framework

A.2 Technical debt, potential capability and speed measurement

This appendix discusses how the numerical results related to *potential capability*, *technical debt* and *execution speed* have been obtained to validate the high level relation of the model.

In particular we explain the process we utilized to measure three measures:

- *Potential capability*: a metric that represents the degree to which each company reflected the capability of reaction and flexibility to the dynamic environment during the development process, given by the three categories that (theoretically) mostly affect *speed-up development*.
- *Execution speed*: a metric that represents the development speed of the startup during different phases of the first release, computed by means of a weighted average speed for each phase, by analyzing interview transcripts looking at subcategories of *Speed-up development*.
- *Technical debt*: a metric that represents the extent to which processes are controlled, structured, planned and documented by means of engineering artifacts and practices. It has been computed by means of a weighted average of the debt accumulated in each development phase observing subcategories of *accumulated technical debt*, with consequences on startups' growth.

A.2.1 Potential capability

To define the last measure - *potential capability* - we quantified characteristics related to three theoretical categories: *team is catalyst of development* (CAT4), *product quality has low priority* (CAT3), and *evolutionary approach* (CAT2). According to the framework, these categories contribute respectively to performance, efficiency and effectiveness - and we want to attest the validity of these relations.

Following the example of the SMS, the procedure has been executed simultaneously in pair on the same screen. When necessary we performed an in-depth review of the transcript[4].

To begin the numerical evaluation, we associated a weight to each category (Table A.23), reflecting the importance according to the empirical data. Factors related to the team have the largest impact on the score (0.5). Factors related to the methodology undertaken are slightly more important (0.3) than quality-related concerns (0.2).

ID	Category	Weight
CAT4	Skilled team is the catalyst of development	0.5
CAT2	Evolutionary approach	0.3
CAT3	Product quality has low priority	0.2

[4]If the conflicts persisted after an in-depth review of the transcript, we let a third expert person (i.e. our supervisors) take the final decision.

Table A.23: Capability - Weights

Subsequently we evaluated the three categories separately, assigning a score to each company according to relevant subcategories[5]. In the next subsections we present the detailed evaluation performed on the three categories. Each subcategory has been evaluated using a discrete scale 0 to 2 where 0 represent a null contribution, 1 is average, and 2 is above the average.

Team factors

Selected subcategories from *team is the catalyst of development (CAT4)*:

- T1: High-impact of CTO background.
- T2: Very small and co-located development team.
- T3: Developers have big responsibilities (self-organized).
- Multi-role and full-stack engineers:
  - T4: Engineers are responsible for marketing/sales/development (flat structure).
  - T5: Generalists developers instead of specialists (full-stack).
- T6: Skilled developers are essential for high speed.
- T7: Team works under constant pressure.
- Limited need of formalities between team-members:
  - T8: Positive impact of high co-location.
  - T9: Previous working experience.
  - T10: Knowing each other before starting the company.

The results of the evaluation are reported in Table A.24, where the *weighted score* has been computed by summing up the individual scores obtained in subcategory, multiplying the value by the weight previously defined and finally normalize it on a scale 1 to 5 in order to be able to make a comparison with the *technical debt* and *execution speed*.

[5] We excluded categories equally impacting all companies since contributing 0.

Company	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	**Weighted score**
C1	2	1	2	1	1	1	2	1	2	1	**2.78**
C2	0	1	2	1	2	0	2	1	1	1	**2.18**
C3	1	0	1	0	0	1	1	2	1	0	**1.39**
C4	2	1	2	0	1	2	1	1	1	0	**2.18**
C5	2	1	1	1	1	1	1	1	1	0	**1.98**
C6	1	1	1	1	0	1	1	1	0	0	**1.39**
C7	1	0	1	1	0	0	1	2	1	0	**1.39**
C8	2	1	2	1	1	2	1	2	1	1	**2.78**
C9	1	1	2	0	1	1	1	2	2	2	**2.58**
C10	1	1	1	0	1	2	1	2	1	1	**2.18**
C11	2	1	2	1	1	2	2	2	2	2	**3.37**
C12	1	1	1	1	0	1	1	1	2	2	**2.18**
C13	1	0	1	1	1	1	0	1	0	2	**1.59**

Table A.24: Capability - Team

Development approach factors

Selected subcategories from *Evolutionary approach (CAT2)*:

- E1: Flexibility and Reactiveness are main objectives.
- E2: Build a functioning prototype and iterate on it (MVP).
- E3: Progressively roll-out to larger number of people.
- E4: Focus on minimal set of functionalities (suitability).
- E5: Small iterations (release often).
- E6: Find product-market fit as soon as possible.

The results of the evaluation are reported in Table A.25.

Company	E1	E2	E3	E4	E5	E6	**Weighted score**
C1	1	1	1	1	2	1	**0.83**
C2	1	1	0	2	1	1	**0.71**
C3	1	1	1	1	1	1	**0.71**
C4	2	2	2	2	2	2	**1.43**
C5	1	1	1	1	1	2	**0.83**
C6	2	2	0	2	1	1	**0.95**
C7	2	2	1	2	2	2	**1.31**
C8	2	1	2	2	2	2	**1.31**
C9	1	1	1	1	1	1	**0.71**
C10	1	0	2	1	2	2	**0.95**
C11	2	2	2	1	2	2	**1.31**
C12	0	0	0	0	1	0	**0.12**
C13	0	0	1	0	1	0	**0.24**

Table A.25: Capability - Evolutionary

Quality factors

Selected subcategories from *Product quality has low priority (CAT3)*:

- Q1: UX is the only important quality.
- Q2: Limited number of suitable functionalities.
- Q3: Users are fault-tolerant in innovative beta products.
- Q4: Efficiency emerges after using the product.
- Q5: Product should be reasonably ready-to-scale.

The results of the evaluation are reported in Table A.26.

Company	Q1	Q2	Q3	Q4	Q5	**Weighted score**
C1	1	1	1	1	0	**0.32**
C2	1	1	1	0	0	**0.24**
C3	1	0	1	0	0	**0.16**
C4	2	2	1	1	2	**0.63**
C5	1	1	1	1	2	**0.48**
C6	1	1	0	1	0	**0.24**
C7	2	0	0	0	1	**0.24**
C8	0	2	1	1	1	**0.40**
C9	1	1	1	1	0	**0.32**
C10	1	1	1	1	1	**0.40**
C11	0	2	0	1	1	**0.32**
C12	1	0	0	0	1	**0.16**
C13	1	0	1	0	1	**0.24**

Table A.26: Capability - Quality

Finally the three overall scores for *potential capability* have been computed by summing up the weighted contributions of the three categories. Table A.27 show the final scores of *potential capability*.

Company	Potential capability
C1	3.928571429
C2	3.134920635
C3	2.261904762
C4	4.246031746
C5	3.293650794
C6	2.579365079
C7	2.936507937
C8	4.484126984
C9	3.611111111
C10	3.531746032
C11	5.000000000
C12	2.460317460
C13	2.063492063

Table A.27: Potential capability

In conclusion, all the operations executed in this appendix had the only scope of attesting the correctness of relations between high-level categories of the framework. Although the scores assigned are subject to researchers personal bias, we executed the whole process in pair, and we provided to other researchers the detailed rubrics and instructions necessaries for executing similar evaluations.

A.2.2 Execution speed and Technical debt

Since both *executions speed* and *technical debt* have been computed by summing up weighted contributions phase by phase, they are presented together. First, the phases have been structured outlining the configuration of the interviews[6]:

- S1: idea/vision/objectives dissemination.
- S2: requirements engineering.
- S3: analysis.
- S4: architecture design.
- S5: coding/debugging.
- S6: verification and validation.
- S7: deployment.
- S8: general project management.

Afterwards, for each company we assigned a weight to each phase based on the effort declared by the respondents in the follow-up questionnaire[7]. The weights are shown in Table A.28.

Company	S1	S2	S3	S4	S5	S6	S7	S8	**Sum**
C1	0.02	0.08	0.11	0.04	0.23	0.23	0.08	0.23	1
C2	0.02	0.08	0.08	0.08	0.23	0.15	0.15	0.23	1
C3	0.02	0.11	0.04	0.08	0.45	0.06	0.02	0.23	1
C4	0.03	0.13	0.04	0.09	0.53	0.07	0.02	0.09	1
C5	0.03	0.09	0.09	0.13	0.40	0.16	0.02	0.09	1
C6	0.03	0.09	0.09	0.27	0.27	0.09	0.09	0.09	1
C7	0.03	0.09	0.04	0.04	0.44	0.22	0.04	0.09	1
C8	0.03	0.13	0.04	0.09	0.53	0.07	0.02	0.09	1
C9	0.03	0.18	0.09	0.18	0.27	0.09	0.09	0.09	1

Table A.28 – *Continued on next page*

[6]Observe how we added a new "phase" that was not present initially in the structure of the interview, but emerged from respondents, which typically started to answer our questions of requirement engineering talking about *how they transmitted the initial idea to other team members*.

[7]For the four companies, which didn't filled the survey, we used average values, adjusted according to interviews.

Table A.28 – *Continued from previous page*

Company	S1	S2	S3	S4	S5	S6	S7	S8	**Sum**
C10	0.02	0.14	0.07	0.11	0.25	0.07	0.07	0.27	1
C11	0.02	0.08	0.08	0.12	0.41	0.13	0.07	0.08	1
C12	0.03	0.13	0.13	0.18	0.35	0.07	0.02	0.09	1
C13	0.03	0.12	0.09	0.13	0.34	0.14	0.07	0.09	1

Table A.28: Weights

Figure A.1 is a graphical representation of the weights assigned to each phase presented in Table A.28.

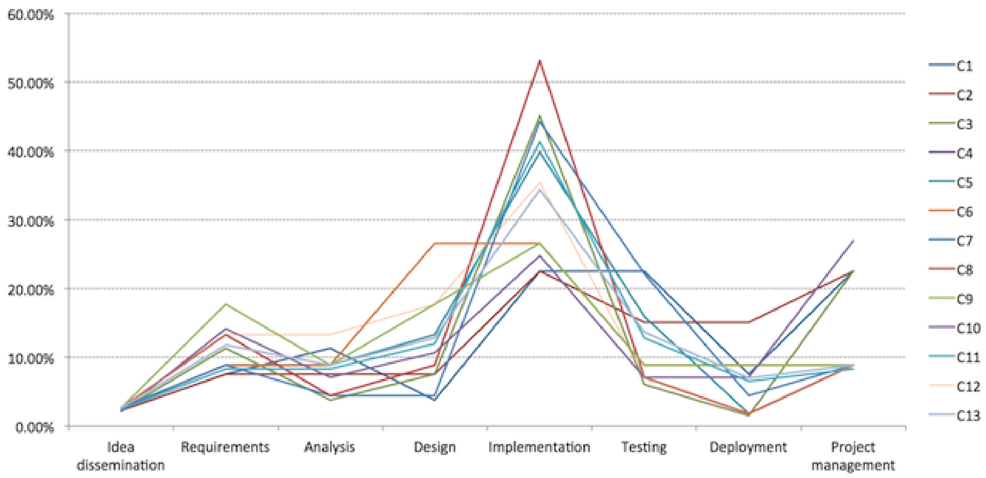

Figure A.1: Execution speed and Technical debt, by phase

As expected, more importance has been assigned to the implementation phase, which occupied most of time and resources of the company. Afterwards we defined a rubric table with extreme values indication phase by phase, used to assign a score from 1 to 5 to each company, for *execution speed* and *technical debt* (see Table A.29).

Idea dissemination (S1)		
Score	Execution speed	Technical debt
1	The vision and objectives formally stated with heavy documentation, which needs to be manually updated.	The vision and objectives are maintained through automatic tools, which promptly inform the team-members.
5	Vision clearly shared among team-members.	Not having any means to share the vision.
Requirements engineering (S2)		
Score	Execution speed	Technical debt
1	The company needs to execute a slow process because of a long list of artifacts and formalized specifications.	Requirements artifacts are complete, up-to-date, accessible, structured, traceable and with shared ownership.
5	Use of highly automated tools and low-precision artifacts to specify the initial list of features/stories.	Features are not documented and shared through oral communication and tacit knowledge

Table A.29 – *Continued on next page*

Table A.29 – *Continued from previous page*

Score	Execution speed	Technical debt
Analysis (S3)		
Score	Execution speed	Technical debt
1	Complete analysis requires formal processes such as risk analysis, feasibility study, critical evaluation of alternative technologies	Pre-defined modus operandi in view of all possible risky situations, fully documented and up-to-date.
5	The only analysis is conducted by logical thinking and reasoning on possible scenarios.	Lack of minimal risk mitigation strategies, and limiting decision taking.
Design (S4)		
Score	Execution speed	Technical debt
1	Formal architecture analysis with detailed diagrams and extended design documentation.	Complete, traceable and up-to-date architectural design artifacts or rigid use of established framework.
5	Little up-front-design supported by well-known framework solutions.	Home-made/No-standard or monolithic architecture design without any support of documentation.
Implementation (S5)		
Score	Execution speed	Technical debt
1	Heavy processes definition with lack of automated/integrated tools and use of out-of-code documentation.	Use of coding standard and clean code accessible by any developer (self-explanatory) with advanced versioning system and task management. Collective ownership and critical decision documented and traceable.
5	Lack of out-of-code documentation to update and minimal workflow guidelines without any burocracy and simple high automated/integrated tools (advanced versioning system, online collaborative ticket-management)	Lack of coding standards and documentation of critical parts in and out of code. Lack of any task management and versioning system
Verification and validation (S6)		
Score	Execution speed	Technical debt
1	Systematic and rigorous quality assurance processes.	High industry standard met for extensive automatic testing systems with documented test cases.
5	Highly automized and quick tests of critical parts of the system only.	Complete lack of both automatic and manual testing systems.
Deployment (S7)		
Score	Execution speed	Technical debt
1	Formal deployment policies with multi-staging system or heavily procedures manually conducted.	Advanced automatic tools or services for multi-staging, easy-to-scale deployment, with a defined documentation for deployment steps.
5	New features directly deployed to production with support of simple and automatic tools.	Manual deployment without any documented procedure to follow.
General project management (S8)		
Score	Execution speed	Technical debt
1	Heavy project management with detailed scheduling, tracing team metrics...	Documented and up-to-date well-defined workflow of activities with support of automatic/online tools.
5	Minimal low-precision tools for task management and internal deadlines for the critical milestones.	Lack of any simple workflow with heavy use of manual tools and oral communications.
Extreme values: 1 = very low; 5 = extremely high		

Table A.29: Rubrics for execution speed and technical debt

We used these guidelines to consistently evaluate in pair the companies, based on the data collected during the case study. The rubrics naturally emerged during the process of evaluating the first companies, and were constantly updated throughout the process. In case of disagreement, conflicts were examined in-depth to reach a final consensus, sometimes by consulting interview transcripts.[8] The results of the evaluation of *execution speed* is shown in Table A.30, and for *technical debt* in Table A.31.

Company	S1	S2	S3	S4	S5	S6	S7	S8	**Weighted score**
C1	5	4	4	4	4	4	4	4	**4.022556391**
C2	3	4	4	4	3	4	4	5	**3.977443609**
C3	3	4	4	3	4	4	5	3	**3.691729323**
C4	4	5	5	5	4	3	3	4	**4.17699115**
C5	5	4	5	5	4	5	4	4	**4.407079646**
C6	3	3	5	4	4	4	4	4	**3.973451327**
C7	4	5	4	4	3	4	5	5	**3.778761062**
C8	4	3	3	4	5	4	4	5	**4.442477876**
C9	5	4	5	3	4	4	4	4	**3.938053097**
C10	3	4	3	4	3	4	4	4	**3.659574468**
C11	5	5	5	4	5	5	4	4	**4.732290708**
C12	3	4	4	4	4	5	5	5	**4.150442478**
C13	3	3	3	4	3	4	4	4	**3.422812193**
Average	3.85	4.00	4.15	4.00	3.85	4.15	4.15	4.23	**4.03**

Table A.30: Execution speed

Company	S1	S2	S3	S4	S5	S6	S7	S8	**Weighted score**
C1	2	4	2	4	2	4	4	4	**3.052631579**
C2	1	2	3	3	3	5	3	3	**3.180451128**
C3	2	3	1	4	2	4	3	3	**2.601503759**
C4	3	4	3	4	3	2	4	4	**3.362831857**
C5	3	3	3	3	3	2	3	3	**3.14159292**
C6	2	4	3	3	3	3	3	3	**3.061946903**
C7	1	2	1	3	3	2	4	4	**3.03539823**
C8	2	2	2	3	4	1	3	3	**3.362831858**
C9	2	3	2	2	3	3	3	3	**2.796460177**
C10	1	2	1	3	2	2	2	2	**2.085106383**
C11	2	3	3	3	4	2	3	3	**3.451701932**
C12	2	2	2	3	4	2	3	3	**3.115044248**
C13	2	3	3	3	3	3	3	3	**2.973451327**
Average	1.92	2.85	2.23	3.15	3.00	3.16	2.69	3.15	**3.02**

Table A.31: Technical debt

A.2.3 Statistical tests

Summarizing the results of the dimensions discussed in the previous subsections, we present the results in Table A.32. Following, we conducted statistical tests using the analysis of variance to assess the existence of relations between *execution speed*, *potential capability* and *technical debt*.

[8] Observe that we tried to define two metrics which can be measured independently from each other and from external factors (team experience, project type, . . .). How these factors can influence the amount of accumulated technical debt or the execution speed can vary case by case. However, it is not in the scope of this thesis exploring those relations.

Company	Execution speed	Potential capability	Technical debt
C1	4.022556391	3.928571429	3.052631579
C2	3.977443609	3.134920635	3.180451128
C3	3.691729323	2.261904762	2.601503759
C4	4.17699115	4.246031746	3.362831857
C5	4.407079646	3.293650794	3.14159292
C6	3.973451327	2.579365079	3.061946903
C7	3.778761062	2.936507937	3.03539823
C8	4.442477876	4.484126984	3.362831858
C9	3.938053097	3.611111111	2.796460177
C10	3.659574468	3.531746032	2.085106383
C11	4.732290708	5.000000000	3.451701932
C12	4.150442478	2.46031746	3.115044248
C13	3.422812193	2.063492063	2.973451327

Table A.32: Quantification results of *execution speed*, *technical debt* and *potential capability*

First we defined two null hypotheses (H0): $H0_1$ = *Startups do not release the product faster when a capable team adopt a more evolutionary approach AND with less quality constraints*; $H0_2$ = *The execution speed does not increase the amount of accumulated technical debt*. Then we tested $H0_1$ and $H0_2$ with an one-tailed test using Pearson's product moment correlation coefficient, with positive association analysis, fixing the level of confidence to 95% which means we reject H0 in case the p-value is lower than 0.05.

We concluded that, in our sample:

1. Higher values of *Execution speed* are strongly associated with *higher* values for *Technical debt* (with a clear statistical significance, p-value: 0.002073).
2. Higher values of *Execution speed* are strongly associated with *higher values* for *Potential capability* (with a clear statistical significance, p-value: 0.004549).

To perform the Pearson's correlation we verified two assumptions: a) data is on interval scale; b) data is normally distributed.

In regard to assumption a) there is a considerable disagreement in the literature whether individual *Likert* items can be considered as interval-level data [2] [3]. However, we provide a symmetric *Likert scale* with a middle category and clearly defined linguistic qualifiers for each item (with the aid of the theoretical framework and rubrics. Then, we made our best to present evaluations that are homogeneously (with same interpretation) distributed across the different companies data. Furthermore we improved the approximation of an interval-level measurement by adjusting weights of scores of categories to equally space the 'distance' between the final scores with the use of validated follow-up questionnaire results.

In regard to assumption b) we validated it by conducting the *Kolmogorov-Smirnov* (K-S) test. Each dimension has been tested separately, comparing them with a normal distribution with the same mean and standard deviation. The output gives the output statistic deviation (D) and then a p-value associated with that statistic. Moreover each dimension's distribution is presented by a histogram and respective normal curve.

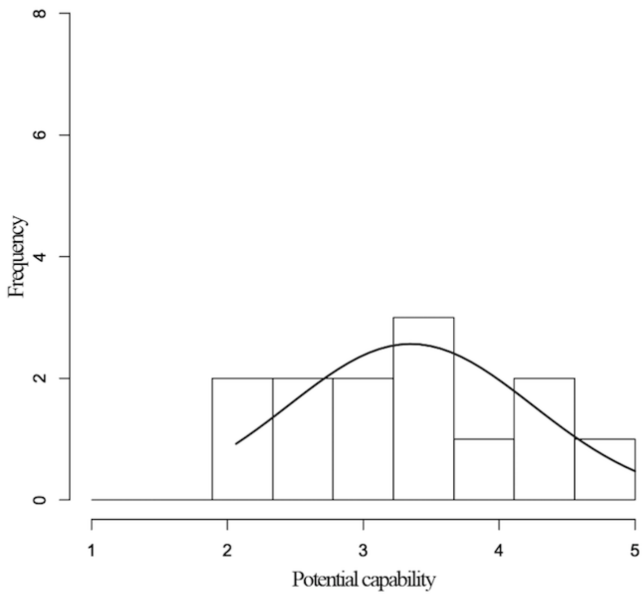

Figure A.2: Distribution of *potential capability*

The output of the *potential capability* K-S test is: D = 0.1117, p-value: 0.9909.

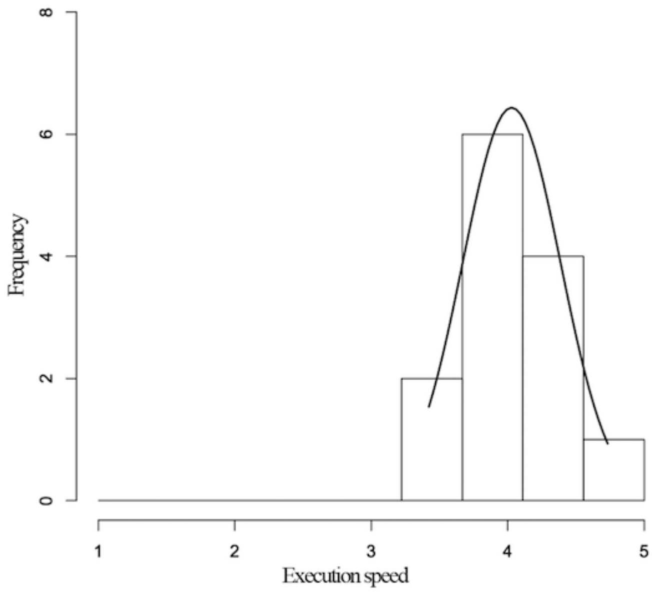

Figure A.3: Distribution of *execution speed*

The output of the *execution speed* K-S test is: D = 0.1223, p-value: 0.9769.

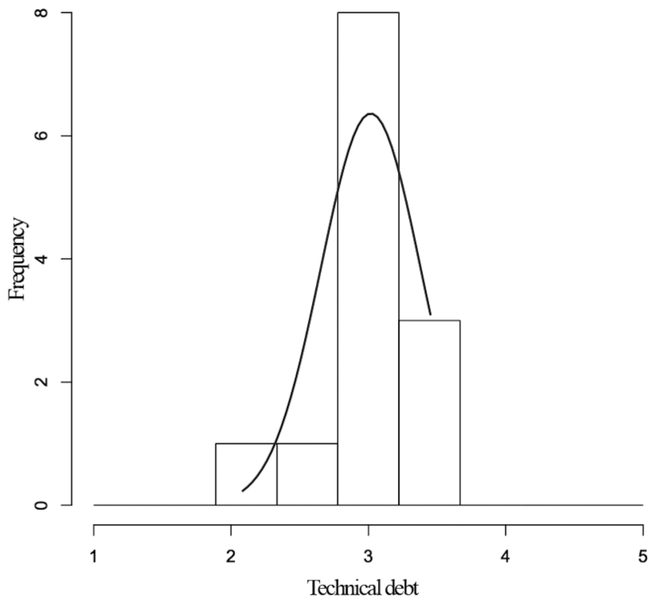

Figure A.4: Distribution of *technical debt*

The output of the *technical debt* K-S test is: D = 0.2214, p-value: 0.4799. As shown above in all the cases the p-values are well above 0.05, so we accept that there is no difference between the observations and a set of random observations drawn from a perfect normal distribution with the same mean and variance.

References

[1] C. Giardino and N. Paternoster. Interview Package. [Online]. Available: https://github.com/adv0r/BTH-Interview-Package, (Aug. 25, 2012).

[2] S. Jamieson, “Likert scales: How to (ab)use them,” *Medical Education*, vol. 38, no. 12, pp. 1217–1218, 2004.

[3] G. Normann, “Likert scales, levels of measurement and the “laws” of statistics,” *Advances in Health Science Education*, vol. 15, no. 5, pp. 625–632, 2010.