MINISTRY OF EDUCATION AND SCIENCE OF THE REPUBLIC OF KAZAKHSTAN

JSC "Kazakh-British Technical University"
Department of Computer Engineering

ADMITTED TO DEFENCE
Head of Computer
Engineering Department

_____

"____" _____ 2013

Student _____

Vadim Kotov

MASTER'S THESIS
XXXXXX – Information Systems

Theme: **"Game Mechanics for Stimulating High Performance of Project Participants"**

Almaty, 26th of May 2013

ASSIGNMENT
for graduation work planning

Student: V. Kotov

Major: Information Systems

Theme: "Mini-Game 'Simulation Of The Image Enhancement' And Visualization Of The Learning Scripts To The Lectures 'Computer Vision'"

Approved by: KBTU, act # 148-P dated 8th of October, 2010

Submission deadline: 23$^{\text{th}}$ of May 2011

Initial data to the project:
International standards (e.g. IEEE 1063-1987, ISO 12207, ANSI/IEEE 983, State Standard 34.201, etc.).

List of questions for graduate work development:

Analytical review, perspective on edutainment and usage of video games in education, examples, "DBB-Crackers" game mechanics

Designing, analysis of image enhancement techniques used in Computer Vision, opportunities for PDF-rendering in Unity game development environment, designing prototypes of "Image Enhancement Tool", "PDF-Reader", "PDF-Converter"

Development, the "Image Enhancement Tool" with following functionality:

- Custom LUT/transfer function based image modification (with complex logical functions available)
- Threshold
- Histogram equalisation

"PDF-Reader" and "PDF-Converter" implementation

Application and experiments, testing of "Image Enhancement Tool", application of thesis results to production: possible challenges, benefits and opportunities

List of diploma project advisers in connection with the diploma paper sections:

Section	**Adviser, department**
The economic part	Yanovskaya O.A., "Department of Economics and Management"
Labour protection part	Rakhmanova Zh. T., "Department of Petrouleum Engineering"

Date of assignment receipt: 10th of January 2010

Supervisors — Prof. Dr. Nailja Luth
c.t.s., docent R.M. Duzbayeva

Student — Vadim Kotov

"____" _____ 2011

MINISTRY OF EDUCATION AND SCIENCE OF THE REPUBLIC OF KAZAKHSTAN

JSC "Kazakh-British Technical University"

Department of Computer Engineering

ADMITTED TO DEFENCE

Head of Computer
Engineering Department
c.t.s., assistant professor

_____

B. K. Dlimbetov

"____" _____ 2011

SCHEDULE
for graduation work

Student: V. Kotov

Major: Information Systems

Theme: "Mini-Game 'Simulation Of The Image Enhancement' And Visualization Of The Learning Scripts To The Lectures 'Computer Vision'"

Supervisors: Prof. Dr. Nailja Luth, Senior Lecturer R. M. Duzbayeva

Type of work	**Deadline**
1. Diploma title and supervisor settlement.	October

Type of work	Deadline
1. Arrival at HAW-AW[1] university 2. Introduction to the "DBB-Crackers" game. Discussion of the project assignment 3. Definition of goals and objectives of the project. Clarification of goals and objectives priority 4. Formulation of research objectives and its characteristics 5. Analytical review: perspective on edutainment and usage of video games in education, examples, "DBB-Crackers" game mechanics 6. Familiarisation with software (Unity) and game prototype	January
1. Analysis of image enhancement techniques and algorithms of used in Computer Vision, opportunities for PDF-rendering in Unity game development environment 2. Designing prototypes of "Image Enhancement Tool", "PDF-Reader", "PDF-Converter" 3. Gaining necessary background information of the thesis papers	April

[1]University of Applied Sciences Amberg-Weiden

Type of work	**Deadline**
1. Development of algorithms, specific for the target platform 2. Testing the software on possible logical errors 3. Experimenting and comparing the results of the work of Image Enhancement Tool with such software, as Adobe Photoshop. Summing up appropriate conclusion	March
1. Submission of the results of the project to the University of Applied Sciences Amberg-Weiden. Finding possible issues and benefits. 2. Preparation the graphic material for the thesis report 3. Preparation of the explanatory note 4. Presentation of the thesis project	May

Head of Computer Engineering Department

_____

B.K.Dlimbetov

Abstract

Here you should write your Abstract. Use this command to see the no of the last page: 76. Compact list:

- Item 1
- Item 2

Contents

List of Figures

List of Tables

Introduction

Dean Spitzer's report on work attitudes.

Motivation for IT/creative people (Drive). Odds of old motivational strategy. Experiment with children and drawing.

A list of problems, connected to startups / estimation, etc.

Market of productivity apps.

The reason to make another app, despite the hype: a lot of similar apps, several "make things different", project managers' "secret knowledge".

1. Project management techniques for small teams and startups

In the past project management was different. Not only because the formal discipline was mostly applied to the large projects lasting several years and costing millions of dollars, but also used a different approach, evolved from ancient military regimes, where relatively few people directed large number of others [1].

Project definition, as a temporary endeavour with a defined beginning and end [2], undertaken to meet unique goals and objectives [3], makes people see the world as a system of projects. Thus making project management techniques essential to get things done, especially personal ones.

1.1 Agile methods

Agile methods are a reaction to traditional ways of developing software and acknowledge the "need for an alternative to documentation driven, heavyweight software development processes".

It promotes adaptive planning, evolutionary development and delivery, a time-boxed iterative approach, and encourages rapid and flexible response to change. It is a conceptual framework that promotes foreseen interactions throughout the development cycle.

In 2001 the Manifesto for Agile Software Development [4] was published to define the approach now known as agile software development.

Some of the manifesto's authors formed the Agile Alliance (http://www.agilealliance.org/), a nonprofit organization that promotes software development according to the manifesto's principles.

The Agile Manifesto:

> We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:
>
> - Individuals and interactions over processes and tools
> - Working software over comprehensive documentation

- Customer collaboration over contract negotiation
- Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Individuals and interactions in agile development, self-organization and motivation are important, as are interactions like co-location and pair programming.

Working software working software will be more useful and welcome than just presenting documents to clients in meetings.

Customer collaboration requirements cannot be fully collected at the beginning of the software development cycle, therefore continuous customer or stakeholder involvement is very important.

Responding to change agile development is focused on quick responses to change and continuous development.[7]

According to Kent Beck,[8] the Agile Manifesto is based on twelve principles:

1. Customer satisfaction by rapid delivery of useful software
2. Welcome changing requirements, even late in development
3. Working software is delivered frequently (weeks rather than months)
4. Working software is the principal measure of progress
5. Sustainable development, able to maintain a constant pace
6. Close, daily cooperation between business people and developers
7. Face-to-face conversation is the best form of communication (co-location)
8. Projects are built around motivated individuals, who should be trusted
9. Continuous attention to technical excellence and good design
10. Simplicitythe art of maximizing the amount of work not doneis essential
11. Self-organizing teams

12. Regular adaptation to changing circumstances

Large-scale agile software development remains an active research area.[39][40]

Agile development has been widely seen as being more suitable for certain types of environment, including small teams of experts.[41][42]:157

Positive reception towards Agile methods has been observed in Embedded domain across Europe in recent years.[43]

Some things that may negatively impact the success of an agile project are:

Large-scale development efforts (¿20 developers), though scaling strategies[40] and evidence of some large projects[44] have been described.

Distributed development efforts (non-colocated teams).

Strategies have been described in Bridging the Distance[45] and Using an Agile Software Process with Offshore Development[46]

Forcing an agile process on a development team[47]

Mission-critical systems where failure is not an option at any cost (e.g. software for air traffic control).

The early successes, challenges and limitations encountered in the adoption of agile methods in a large organization have been documented.[48]

In terms of outsourcing agile development, Michael Hackett, Sr. Vice President of LogiGear Corporation has stated that "the offshore team ... should have expertise, experience, good communication skills, inter-cultural understanding, trust and understanding between members and groups and with each other."[49]

Agile methods have been extensively used for development of software products and some of them use certain characteristics of software, such as object technologies.[50] However, these techniques can be applied to the development of non-software products, such as computers, motor vehicles, medical devices, food, and clothing; see Flexible product development.

Table 1.1: Suitability of different development methods

Agile	**Plan-driven**	**Formal methods**
Low criticality	High criticality	Extreme criticality
Senior developers	Junior developers	Senior developers
Requirements change often	Requirements do not change often	Limited requirements, limited features

1.2 Scrum

http://scrum.org These all are from Scrum Guide

Scrum is a framework within which people can address complex adaptive problems, while productively and creatively delivering products of the highest possible value.

The Scrum framework consists of Scrum Teams and their associated roles, events, artifacts, and rules. Each component within the framework serves a specific purpose and is essential to Scrums success and usage.

1.2.1 Scrum theory

Scrum is founded on empirical process control theory, or empiricism. Empiricism asserts that knowledge comes from experience and making decisions based on what is known. Scrum employs an iterative, incremental approach to optimise predictability and control risk.

Transparency

Significant aspects of the process must be visible to those responsible for the outcome.

Transparency requires those aspects be defined by a common standard so observers share a common understanding of what is being seen.

For example:

- A common language referring to the process must be shared by all participants; and,
- A common definition of Done must be shared by those performing the work and those accepting the work product.

Inspection

Scrum users must frequently inspect Scrum artifacts and progress toward a goal to detect undesirable variances. Their inspection should not be so frequent that inspection gets in the way of the work. Inspections are most beneficial when diligently performed by skilled inspectors at the point of work.

Adaptation

If an inspector determines that one or more aspects of a process deviate outside acceptable limits, and that the resulting product will be unacceptable, the process or the material being processed must be adjusted. An adjustment must be made as soon as possible to minimize further deviation.

Scrum prescribes four formal opportunities for inspection and adaptation, as described in the Scrum Events section of this document.

- Sprint Planning Meeting
- Daily Scrum
- Sprint Review
- Sprint Retrospective

1.2.2 The Scrum team

The Scrum Team consists of a Product Owner, the Development Team, and a Scrum Master. Scrum Teams are self-organizing and cross-functional. Self-organizing teams choose how best to accomplish their work, rather than being directed by others outside the team. Cross-functional teams have all competencies needed to accomplish the work without depending on others not part of the team. The team model in Scrum is designed to optimize flexibility, creativity, and productivity.

Scrum Teams deliver products iteratively and incrementally, maximizing opportunities for feedback. Incremental deliveries of "Done" product ensure a potentially useful version of working product is always available.

1.2.3 Scrum events

Prescribed events are used in Scrum to create regularity and to minimize the need for meetings not defined in Scrum. Scrum uses time-boxed events, such that every event has a maximum duration. This ensures an appropriate amount of time is spent planning without allowing waste in the planning process.

Other than the Sprint itself, which is a container for all other events, each event in Scrum is a formal opportunity to inspect and adapt something. These

events are specifically designed to enable critical transparency and inspection. Failure to include any of these events results in reduced transparency and is a lost opportunity to inspect and adapt.

1.2.4 Scrum artifacts

Scrums artifacts represent work or value in various ways that are useful in providing transparency and opportunities for inspection and adaptation. Artifacts defined by Scrum are specifically designed to maximize transparency of key information needed to ensure Scrum Teams are successful in delivering a Done Increment.

1.3 RAD – Rapid Application Development

RAD is an integrated set of techniques, guidelines and tools that facilitate deploying a customer's software needs within a short period of time. This predefined timeframe is called a "timebox". The software product evolves during the RAD development process based on continued customer feedback. In addition, the whole software product is not delivered at once, but is delivered in pieces by order of business importance.

1.3.1 Phases of RAD

1. Requirements Planning phase combines elements of the system planning and systems analysis phases of the Systems Development Life Cycle (SDLC). Users, managers, and IT staff members discuss and agree on business needs, project scope, constraints, and system requirements. It ends when the team agrees on the key issues and obtains management authorization to continue.

2. User design phase during this phase, users interact with systems analysts and develop models and prototypes that represent all system processes, inputs, and outputs. The RAD groups or subgroups typically use a combination of Joint Application Development (JAD) techniques and CASE tools to translate user needs into working models. User Design is a con-

tinuous interactive process that allows users to understand, modify, and eventually approve a working model of the system that meets their needs.

3. Construction phase focuses on program and application development task similar to the SDLC. In RAD, however, users continue to participate and can still suggest changes or improvements as actual screens or reports are developed. Its tasks are programming and application development, coding, unit-integration and system testing.

4. Cutover phase resembles the final tasks in the SDLC implementation phase, including data conversion, testing, changeover to the new system, and user training. Compared with traditional methods, the entire process is compressed. As a result, the new system is built, delivered, and placed in operation much sooner.

1.4 TDD – Test-Driven Development

Test-driven development (TDD) is a software development process that relies on the repetition of a very short development cycle: first the developer writes an (initially failing) automated test case that defines a desired improvement or new function, then produces the minimum amount of code to pass that test, and finally refactors the new code to acceptable standards. Kent Beck, who is credited with having developed or 'rediscovered' the technique, stated in 2003 that TDD encourages simple designs and inspires confidence.

1.4.1 TDD Cycle

Add a test

In test-driven development, each new feature begins with writing a test. This test must inevitably fail because it is written before the feature has been implemented. (If it does not fail, then either the proposed "new" feature already exists or the test is defective.) To write a test, the developer must clearly understand the feature's specification and requirements. The developer can accomplish this through use cases and user stories to cover the requirements and exception conditions, and can write the test in whatever testing framework

is appropriate to the software environment. This could also be a modification of an existing test. This is a differentiating feature of test-driven development versus writing unit tests after the code is written: it makes the developer focus on the requirements before writing the code, a subtle but important difference.

Run all tests and see if the new one fails

This validates that the test harness is working correctly and that the new test does not mistakenly pass without requiring any new code. This step also tests the test itself, in the negative: it rules out the possibility that the new test always passes, and therefore is worthless. The new test should also fail for the expected reason. This increases confidence (though does not guarantee) that it is testing the right thing, and passes only in intended cases.

Write some code

The next step is to write some code that causes the test to pass. The new code written at this stage is not perfect, and may, for example, pass the test in an inelegant way. That is acceptable because later steps improve and hone it.

At this point, the only purpose of the written code is to pass the test; no further (and therefore untested) functionality should be predicted and 'allowed for' at any stage.

Run the automated tests and see them succeed

If all test cases now pass, the programmer can be confident that the code meets all the tested requirements. This is a good point from which to begin the final step of the cycle.

Refactor code

Now the code can be cleaned up as necessary. By re-running the test cases, the developer can be confident that code refactoring is not damaging any existing functionality. The concept of removing duplication is an important aspect of any software design. In this case, however, it also applies to removing any duplication between the test code and the production codefor example magic numbers or strings repeated in both to make the test pass in step 3.

Repeat

Starting with another new test, the cycle is then repeated to push forward the functionality. The size of the steps should always be small, with as few as 1 to 10 edits between each test run. If new code does not rapidly satisfy a new test, or other tests fail unexpectedly, the programmer should undo or revert in preference to excessive debugging. Continuous integration helps by providing revertible checkpoints. When using external libraries it is important not to make increments that are so small as to be effectively merely testing the library itself,[3] unless there is some reason to believe that the library is buggy or is not sufficiently feature-complete to serve all the needs of the main program being written.

1.5 FDD – Feature-Driven Development

Feature-driven development (FDD) is an iterative and incremental software development process. It is one of a number of Agile methods for developing software and forms part of the Agile Alliance. FDD blends a number of industry-recognised best practices into a cohesive whole. These practices are all driven from a client-valued functionality (feature) perspective. Its main purpose is to deliver tangible, working software repeatedly in a timely manner.

1.5.1 Phases of FDD

Develop overall model

The project started with a high-level walkthrough of the scope of the system and its context. Next, detailed domain walkthroughs were held for each modeling area. In support of each domain, walkthrough models were then composed by small groups, which were presented for peer review and discussion. One of the proposed models, or a merge of them, was selected which became the model for that particular domain area. Domain area models were merged into an overall model, and the overall model shape was adjusted along the way.

Build feature list

The knowledge that was gathered during the initial modeling was used to identify a list of features. This was done by functionally decomposing the domain into subject areas. Subject areas each contain business activities, the steps within each business activity formed the categorized feature list. Features in this respect were small pieces of client-valued functions expressed in the form "¡action¿ ¡result¿ ¡object¿", for example: 'Calculate the total of a sale' or 'Validate the password of a user'. Features should not take more than two weeks to complete, else they should be broken down into smaller pieces.

Plan by feature

After the feature list had been completed, the next step was to produce the development plan. Class ownership has been done by ordering and assigning features (or feature sets) as classes to chief programmers.

Design by feature

A design package was produced for each feature. A chief programmer selected a small group of features that are to be developed within two weeks. Together with the corresponding class owners, the chief programmer worked out detailed sequence diagrams for each feature and refines the overall model. Next, the class and method prologues are written and finally a design inspection is held.

Build by feature

After a successful design inspection a per feature activity to produce a completed client-valued function (feature) is being produced. The class owners develop the actual code for their classes. After a unit test and a successful code inspection, the completed feature is promoted to the main build.

1.5.2 Best practices

Feature-Driven Development is built around a core set of industry-recognized best practices, derived from software engineering. These practices are all driven

from a client-valued feature perspective. It is the combination of these practices and techniques that makes FDD so compelling. The best practices that make up FDD are shortly described below. For each best practice a short description will be given.

- Domain Object Modeling. Domain Object Modeling consists of exploring and explaining the domain of the problem to be solved. The resulting domain object model provides an overall framework in which to add features.

- Developing by Feature. Any function that is too complex to be implemented within two weeks is further decomposed into smaller functions until each sub-problem is small enough to be called a feature. This makes it easier to deliver correct functions and to extend or modify the system.

- Individual Class (Code) Ownership. Individual class ownership means that distinct pieces or grouping of code are assigned to a single owner. The owner is responsible for the consistency, performance, and conceptual integrity of the class.

- Feature Teams. A feature team is a small, dynamically formed team that develops a small activity. By doing so, multiple minds are always applied to each design decision and also multiple design options are always evaluated before one is chosen.

- Inspections. Inspections are carried out to ensure good quality design and code, primarily by detection of defects.

- Configuration Management. Configuration management helps with identifying the source code for all features that have been completed to date and to maintain a history of changes to classes as feature teams enhance them.

- Regular Builds. Regular builds ensure there is always an up to date system that can be demonstrated to the client and helps highlighting integration errors of source code for the features early.

- Visibility of progress and results. By frequent, appropriate, and accurate progress reporting at all levels inside and outside the project, based on completed work, managers are helped at steering a project correctly.

1.6 RUP – Rational Unified Process

The Rational Unified Process (RUP) is an iterative software development process framework created by the Rational Software Corporation, a division of IBM since 2003.[1] RUP is not a single concrete prescriptive process, but rather an adaptable process framework, intended to be tailored by the development organisations and software project teams that will select the elements of the process that are appropriate for their needs. RUP is a specific implementation of the Unified Process.

1.6.1 RUP building blocks

RUP is based on a set of building blocks, or content elements, describing what is to be produced, the necessary skills required and the step-by-step explanation describing how specific development goals are to be achieved. The main building blocks, or content elements, are the following:

- Roles (who) A Role defines a set of related skills, competencies and responsibilities.
- Work Products (what) A Work Product represents something resulting from a task, including all the documents and models produced while working through the process.
- Tasks (how) A Task describes a unit of work assigned to a Role that provides a meaningful result.

Within each iteration, the tasks are categorized into nine disciplines:

- Six "engineering disciplines"
- Business Modeling
- Requirements

- Analysis and Design
- Implementation
- Test
- Deployment
- Three supporting disciplines
- Configuration and Change Management
- Project Management
- Environment

1.6.2 Four project lifecycle phases

The RUP has determined a project life cycle consisting of four phases. These phases allow the process to be presented at a high level in a similar way to how a 'waterfall'-styled project might be presented, although in essence the key to the process lies in the iterations of development that lie within all of the phases. Also, each phase has one key objective and milestone at the end that denotes the objective being accomplished. The visualization of RUP phases and disciplines over time is referred to as the RUP hump chart.

Inception Phase

The primary objective is to scope the system adequately as a basis for validating initial costing and budgets. In this phase the business case which includes business context, success factors (expected revenue, market recognition, etc.), and financial forecast is established. To complement the business case, a basic use case model, project plan, initial risk assessment and project description (the core project requirements, constraints and key features) are generated. After these are completed, the project is checked against the following criteria:

- Stakeholder concurrence on scope definition and cost/schedule estimates.
- Requirements understanding as evidenced by the fidelity of the primary use cases.

- Credibility of the cost/schedule estimates, priorities, risks, and development process.
- Depth and breadth of any architectural prototype that was developed.
- Establishing a baseline by which to compare actual expenditures versus planned expenditures.

If the project does not pass this milestone, called the Lifecycle Objective Milestone, it either can be cancelled or repeated after being redesigned to better meet the criteria.

Elaboration Phase

The primary objective is to mitigate the key risk items identified by analysis up to the end of this phase. The elaboration phase is where the project starts to take shape. In this phase the problem domain analysis is made and the architecture of the project gets its basic form.

The outcome of the elaboration phase is:

- A use-case model in which the use-cases and the actors have been identified and most of the use-case descriptions are developed. The use-case model should be 80% complete.
- A description of the software architecture in a software system development process.
- An executable architecture that realises architecturally significant use cases.
- Business case and risk list which are revised.
- A development plan for the overall project.
- Prototypes that demonstrably mitigate each identified technical risk.
- A preliminary user manual (optional).

This phase must pass the Lifecycle Architecture Milestone criteria answering the following questions:

- Is the vision of the product stable?
- Is the architecture stable?
- Does the executable demonstration indicate that major risk elements are addressed and resolved?
- Is the construction phase plan sufficiently detailed and accurate?
- Do all stakeholders agree that the current vision can be achieved using current plan in the context of the current architecture?
- Is the actual vs. planned resource expenditure acceptable?

If the project cannot pass this milestone, there is still time for it to be cancelled or redesigned. However, after leaving this phase, the project transitions into a high-risk operation where changes are much more difficult and detrimental when made.

The key domain analysis for the elaboration is the system architecture.

Construction Phase

The primary objective is to build the software system. In this phase, the main focus is on the development of components and other features of the system. This is the phase when the bulk of the coding takes place. In larger projects, several construction iterations may be developed in an effort to divide the use cases into manageable segments that produce demonstrable prototypes.

This phase produces the first external release of the software. Its conclusion is marked by the Initial Operational Capability Milestone.

Transition Phase

The primary objective is to 'transit' the system from development into production, making it available to and understood by the end user. The activities of this phase include training the end users and maintainers and beta testing the system to validate it against the end users' expectations. The product is also checked against the quality level set in the Inception phase.

If all objectives are met, the Product Release Milestone is reached and the development cycle is finished.

1.6.3 Best practises

Six Best Practices as described in the Rational Unified Process is a paradigm in software engineering, that lists six ideas to follow when designing any software project to minimise faults and increase productivity. These practices are:

Develop iteratively

It is best to know all requirements in advance; however, often this is not the case. Several software development processes exist that deal with providing solution on how to minimize cost in terms of development phases.

Manage requirements

Always keep in mind the requirements set by users.

Use components

Breaking down an advanced project is not only suggested but in fact unavoidable. This promotes ability to test individual components before they are integrated into a larger system. Also, code reuse is a big plus and can be accomplished more easily through the use of object-oriented programming.

Model visually

Use diagrams to represent all major components, users, and their interaction. "UML", short for Unified Modeling Language, is one tool that can be used to make this task more feasible.

Verify quality

Always make testing a major part of the project at any point of time. Testing becomes heavier as the project progresses but should be a constant factor in any software product creation.

Control changes

Many projects are created by many teams, sometimes in various locations, different platforms may be used, etc. As a result it is essential to make sure

that changes made to a system are synchronized and verified constantly.

1.7 Lean

Lean development can be summarized by seven principles, very close in concept to lean manufacturing principles:

1. Eliminate waste
2. Amplify learning
3. Decide as late as possible
4. Deliver as fast as possible
5. Empower the team
6. Build integrity in
7. See the whole

1.7.1 Lean manufacturing principles

Eliminate waste

Everything not adding value to the customer is considered to be waste (muda). This includes:

1. unnecessary code and functionality
2. delay in the software development process
3. unclear requirements
4. insufficient testing, leading to avoidable process repetition
5. bureaucracy
6. slow internal communication

In order to be able to eliminate waste, one should be able to recognize it. If some activity could be bypassed or the result could be achieved without it, it is waste. Partially done coding eventually abandoned during the development process is waste. Extra processes and features not often used by customers are waste. Waiting for other activities, teams, processes is waste. Defects and lower quality are waste. Managerial overhead not producing real value is waste. A value stream mapping technique is used to distinguish and recognize waste. The second step is to point out sources of waste and eliminate them. The same should be done iteratively until even essential-seeming processes and procedures are liquidated.

Amplify learning

Software development is a continuous learning process with the additional challenge of development teams and end product sizes. The best approach for improving a software development environment is to amplify learning. The accumulation of defects should be prevented by running tests as soon as the code is written. Instead of adding more documentation or detailed planning, different ideas could be tried by writing code and building. The process of user requirements gathering could be simplified by presenting screens to the end-users and getting their input.

The learning process is sped up by usage of short iteration cycles each one coupled with refactoring and integration testing. Increasing feedback via short feedback sessions with customers helps when determining the current phase of development and adjusting efforts for future improvements. During those short sessions both customer representatives and the development team learn more about the domain problem and figure out possible solutions for further development. Thus the customers better understand their needs, based on the existing result of development efforts, and the developers learn how to better satisfy those needs. Another idea in the communication and learning process with a customer is set-based development this concentrates on communicating the constraints of the future solution and not the possible solutions, thus promoting the birth of the solution via dialogue with the customer.

Decide as late as possible

As software development is always associated with some uncertainty, better results should be achieved with an options-based approach, delaying decisions as much as possible until they can be made based on facts and not on uncertain assumptions and predictions. The more complex a system is, the more capacity for change should be built into it, thus enabling the delay of important and crucial commitments. The iterative approach promotes this principle the ability to adapt to changes and correct mistakes, which might be very costly if discovered after the release of the system.

An agile software development approach can move the building of options earlier for customers, thus delaying certain crucial decisions until customers have realized their needs better. This also allows later adaptation to changes and the prevention of costly earlier technology-bounded decisions. This does not mean that no planning should be involved on the contrary, planning activities should be concentrated on the different options and adapting to the current situation, as well as clarifying confusing situations by establishing patterns for rapid action. Evaluating different options is effective as soon as it is realized that they are not free, but provide the needed flexibility for late decision making.

Deliver as fast as possible

In the era of rapid technology evolution, it is not the biggest that survives, but the fastest. The sooner the end product is delivered without considerable defect, the sooner feedback can be received, and incorporated into the next iteration. The shorter the iterations, the better the learning and communication within the team. Without speed, decisions cannot be delayed. Speed assures the fulfilling of the customer's present needs and not what they required yesterday. This gives them the opportunity to delay making up their minds about what they really require until they gain better knowledge. Customers value rapid delivery of a quality product.

The just-in-time production ideology could be applied to software development, recognizing its specific requirements and environment. This is achieved by presenting the needed result and letting the team organize itself and divide the tasks for accomplishing the needed result for a specific iteration. At

the beginning, the customer provides the needed input. This could be simply presented in small cards or stories the developers estimate the time needed for the implementation of each card. Thus the work organization changes into self-pulling system each morning during a stand-up meeting, each member of the team reviews what has been done yesterday, what is to be done today and tomorrow, and prompts for any inputs needed from colleagues or the customer. This requires transparency of the process, which is also beneficial for team communication. Another key idea in Toyota's Product Development System is set-based design. If a new brake system is needed for a car, for example, three teams may design solutions to the same problem. Each team learns about the problem space and designs a potential solution. As a solution is deemed unreasonable, it is cut. At the end of a period, the surviving designs are compared and one is chosen, perhaps with some modifications based on learning from the others - a great example of deferring commitment until the last possible moment. Software decisions could also benefit from this practice to minimize the risk brought on by big up-front design.

Empower the team

There has been a traditional belief in most businesses about the decision-making in the organization the managers tell the workers how to do their own job. In a Work-Out technique, the roles are turned the managers are taught how to listen to the developers, so they can explain better what actions might be taken, as well as provide suggestions for improvements. The lean approach favors the aphorism "find good people and let them do their own job," encouraging progress, catching errors, and removing impediments, but not micro-managing.

Another mistaken belief has been the consideration of people as resources. People might be resources from the point of view of a statistical data sheet, but in software development, as well as any organizational business, people do need something more than just the list of tasks and the assurance that they will not be disturbed during the completion of the tasks. People need motivation and a higher purpose to work for purpose within the reachable reality, with the assurance that the team might choose its own commitments. The developers should be given access to the customer; the team leader should provide support

and help in difficult situations, as well as ensure that skepticism does not ruin the teams spirit.

Build integrity in

The customer needs to have an overall experience of the System this is the so-called perceived integrity: how it is being advertised, delivered, deployed, accessed, how intuitive its use is, price and how well it solves problems.

Conceptual integrity means that the systems separate components work well together as a whole with balance between flexibility, maintainability, efficiency, and responsiveness. This could be achieved by understanding the problem domain and solving it at the same time, not sequentially. The needed information is received in small batch pieces not in one vast chunk with preferable face-to-face communication and not any written documentation. The information flow should be constant in both directions from customer to developers and back, thus avoiding the large stressful amount of information after long development in isolation.

One of the healthy ways towards integral architecture is refactoring. As more features are added to the original code base, the harder it becomes to add further improvements. Refactoring is about keeping simplicity, clarity, minimum amount of features in the code. Repetitions in the code are signs for bad code designs and should be avoided. The complete and automated building process should be accompanied by a complete and automated suite of developer and customer tests, having the same versioning, synchronization and semantics as the current state of the System. At the end the integrity should be verified with thorough testing, thus ensuring the System does what the customer expects it to. Automated tests are also considered part of the production process, and therefore if they do not add value they should be considered waste. Automated testing should not be a goal, but rather a means to an end, specifically the reduction of defects.

See the whole

Software systems nowadays are not simply the sum of their parts, but also the product of their interactions. Defects in software tend to accumulate during the development process by decomposing the big tasks into smaller tasks,

and by standardizing different stages of development, the root causes of defects should be found and eliminated. The larger the system, the more organisations that are involved in its development and the more parts are developed by different teams, the greater the importance of having well defined relationships between different vendors, in order to produce a system with smoothly interacting components. During a longer period of development, a stronger subcontractor network is far more beneficial than short-term profit optimizing, which does not enable win-win relationships.

Lean thinking has to be understood well by all members of a project, before implementing in a concrete, real-life situation. "Think big, act small, fail fast; learn rapidly" these slogans summarize the importance of understanding the field and the suitability of implementing lean principles along the whole software development process. Only when all of the lean principles are implemented together, combined with strong "common sense" with respect to the working environment, is there a basis for success in software development.

1.8 Methodologies comparison

A table from RAD wikipedia article

1.9 Project management goals for small teams and start-ups

Productivity

Why should startups and small-teams should be sales-oriented or problem-solving oriented, but not technology oriented at first place?

The general goal of project management is collecting feedback in order to manage available resources to be in time. Why? – Flow, will be considered later.

1.9.1 PERT and Monte-Carlo simulation

The Program (or Project) Evaluation and Review Technique, commonly abbreviated PERT, is a statistical tool, used in project management, that is designed to analyze and represent the tasks involved in completing a given

project. First developed by the United States Navy in the 1950s, it is commonly used in conjunction with the critical path method (CPM).

PERT was developed primarily to simplify the planning and scheduling of large and complex projects. It was developed for the U.S. Navy Special Projects Office in 1957 to support the U.S. Navy's Polaris nuclear submarine project. It was able to incorporate uncertainty by making it possible to schedule a project while not knowing precisely the details and durations of all the activities. It is more of an event-oriented technique rather than start- and completion-oriented, and is used more in projects where time is the major factor rather than cost. It is applied to very large-scale, one-time, complex, non-routine infrastructure and Research and Development projects. An example of this was for the 1968 Winter Olympics in Grenoble which applied PERT from 1965 until the opening of the 1968 Games.

Conventions

- A PERT chart is a tool that facilitates decision making. The first draft of a PERT chart will number its events sequentially in 10s (10, 20, 30, etc.) to allow the later insertion of additional events.
- Two consecutive events in a PERT chart are linked by activities, which are conventionally represented as arrows (see the diagram above).
- The events are presented in a logical sequence and no activity can commence until its immediately preceding event is completed.
- The planner decides which milestones should be PERT events and also decides their proper sequence.
- A PERT chart may have multiple pages with many sub-tasks.

Advantages

- PERT chart explicitly defines and makes visible dependencies (precedence relationships) between the work breakdown structure (commonly WBS) elements
- PERT facilitates identification of the critical path and makes this visible

- PERT facilitates identification of early start, late start, and slack for each activity,
- PERT provides for potentially reduced project duration due to better understanding of dependencies leading to improved overlapping of activities and tasks where feasible.
- The large amount of project data can be organised and presented in diagram for use in decision making.

Disadvantages

- There can be potentially hundreds or thousands of activities and individual dependency relationships
- PERT is not easily scalable for smaller projects
- The network charts tend to be large and unwieldy requiring several pages to print and requiring special size paper
- The lack of a timeframe on most PERT/CPM charts makes it harder to show status although colours can help (e.g., specific colour for completed nodes)
- When the PERT/CPM charts become unwieldy, they are no longer used to manage the project.

Uncertainty in project scheduling

During project execution, however, a real-life project will never execute exactly as it was planned due to uncertainty. It can be ambiguity resulting from subjective estimates that are prone to human errors or it can be variability arising from unexpected events or risks. The main reason that PERT may provide inaccurate information about the project completion time is due to this schedule uncertainty. This inaccuracy is large enough to render such estimates as not helpful.

One possibility to maximise solution robustness is to include safety in the baseline schedule in order to absorb the anticipated disruptions. This is called

proactive scheduling. A pure proactive scheduling is a utopia, incorporating safety in a baseline schedule that allows to cope with every possible disruption would lead to a baseline schedule with a very large make-span. A second approach, reactive scheduling, consists of defining a procedure to react to disruptions that cannot be absorbed by the baseline schedule.

1.9.2 Teams without project managers

Importance for collaboration software (like basecamp) to have it's own methodology and provide subtle education for its users.

The bad: hard to adopt a different methodology. No room for creativity?

The good: potentially more effective for beginners.

2. Team-motivation strategies and personal productivity

They way to create an intention to develop / tailor appropriate methodology is to begin with personal productivity and motivation behind doing the project. Why?

Remind the problem of motivation, Dean Spitzer's report. Is it appropriate for startups?

2.1 "Action Method" application and concept behind

The Action Method begins with a simple premise: everything is a project. This applies not only to the big presentation on Wednesday or the new campaign youre preparing, but also to the stuff you do to advance your career (a career development project), or to employee development (each of your subordinates represents a single project in which you keep track of performance and the steps you plan to take to help him or her develop as an employee). Managing your finances is a project, as is doing your taxes or arranging the upcoming house move.Like most creative people, Im sure you struggle to make progress in all of your projects, with the greatest challenge being the sheer number of projects before you! But once you have everything classified as a project, you can start breaking each one down into its primary components: Action Steps, References, and Backburner Items.

Every project in life can be reduced into these three primary components.

Action Steps

are the specific, concrete tasks that inch you forward: redraft and send the memo, post the blog entry, pay the electricity bill, etc.

References

are any project-related handouts, sketches, notes, meeting minutes, manuals, websites, or ongoing discussions that you may want to refer back to. It is important to note that references are not actionablethey are simply there for reference when focusing on any particular project. Finally, there are

Backburner Items

things that are not actionable now but may be someday. Perhaps it is an idea for a client for which there is no budget yet. Or maybe it is something you intend to do in a particular project at an unforeseen time in the future.

Every project in life can be reduced into these three primary components.

Lets consider a sample project for a client. Imagine a folder with that clients name on it. Inside the folder you would have a lot of Referencesperhaps a copy of the contract, notes from meetings, and background information on the client. The Action Stepsthe stuff you need to docould be written as a list, attached to the front of the folder. And then, perhaps on a sheet stapled to the inside back cover of the folder, your Backburner list could keep track of the non-actionable ideas that come up while working on the projectthe stuff you may want to do in the future.

With this hypothetical folder in mind, you can imagine that the majority of your focus would be on the Action Steps visible on the front cover. These Action Steps are always in plain view. They catch your eye every time you glance at the project folder. And, as you review all of your project folders every day, what youre really doing is just glancing over all of the pending Action Steps.

We call it the Action Method because it helps us live and work with a bias toward action. The actionable aspects of every project pop out at us, and the other components are organized enough to provide peace of mind while not getting in the way of taking action.

Personal projects can also be broken down into the same three elements. If you take some time to look around your desk, you might find some notes or reminders that youve left for yourself. Perhaps you see a household bill that requires payment (an Action Step in the project Household Management), or a copy of your car insurance certificate (a Reference in the project Insurance). Maybe it is a cutout of a great vacation spot you want to visit someday (a Backburner Item in the project Vacation Planning).

Consider a few projects in your lifesome work-related and some personal. The components of these projects are either in your head or all around yousentences in emails, sketches in notebooks, and scribbles on Post-it notes. The Action Method starts by considering everything around you with a project lens and then breaking it down.

The Action Method starts by considering everything around you with a project lens and then breaking it down.

Perhaps you have an idea for a screenplay that youd like to write someday. If so, make it a Backburner Item in the New Screenplay Ideas project or perhaps in a more general Bold Ideas project that you may review only a couple of times every year. (In ActionMethod Online, Backburner Items can be captured by creating an Action Step without a due date, which will automatically be displayed at the bottom of each project.) While some projects realistically wont get much of your focus, they will help store the Backburner Items and References that you generate.

Of course, your hope is that someday a few of these Backburner Items will be converted into real Action Stepswhich will, in turn, lead to a new and more active project, like your screenplay. Action Steps are the building blocks of accomplishment. But sometimes, at certain periods of life, you cant afford to take certain actions. For this reason, it is okay to have dormant projects filled with References and Backburner Items. The time will come when some of these projects return to the surface with some Action Steps.

As you go about your day, you should think in terms of which project is associated with what you are doing at any point in time. Whether in a meeting, brainstorming session, chance conversation, article, dream, or eureka moment in the shower, you are generating Action Steps, References, and Backburner Items at a fast clip. Everything is associated with a project. Sadly, much of this output will be lost unless you capture it and assign it properly.

In the sections ahead, we will explore the three primary components of projects in more detail and how they should be managed. But the key realization should be that everything in life is a project, and every project must be broken down into Action Steps, References, and Backburner Items. Its that simple.

The key realization is that everything in life is a project, and every project must be broken down into Action Steps, References, and Backburner Items.

Of course, in the digital era, information comes to us in many forms. Projects are not always kept in folders. In fact, projects are managed across many mediums. And the components of projects come to us in the form of emails, status updates, files as downloads, and a barrage of links that we save

daily. Nevertheless, the Action Method still applies; everything belongs to a project. With the Action Method in mind, we can make better use of online and offline tools that organize information.

Action Steps are the most important components of projectsthe oxygen for keeping projects alive. No Action Steps, no action, no results. The actual outcome of any idea is dependent on the Actions Steps that are captured and then completed by you or delegated to someone else. Action Steps are to be revered and treated as sacred in any project.The more clear and concrete an Action Step is, the less friction you will encounter trying to do it. If an Action Step is vague or complicated, you will probably skip over it to others on your list that are more straightforward.

To avoid this, start each Action Step with a verb:

* Call programmer to discuss . . . * Install new software for . . . * Research the possibility of . . . * Mock up a sample of the . . . * Update XYZ document for . . .

Verbs help pull us into our Action Steps at first glance, efficiently indicating what type of action is required. For similar reasons, Action Steps should be kept short.

The more clear and concrete an Action Step is, the less friction you will encounter trying to do it.

Imagine you and I are having a conversation in a meeting. I describe to you what I want to accomplish and show you some diagrams that further describe the idea. You reply by saying, I see what youre trying to do. Theres a guy I know who designed a great website with the same type of functionality. Upon saying this, I record an Action Step to follow up with you regarding that website:

* Follow up with [name] re: guys website w/ similar functionality.

A colleague might say, Lets revisit that old draft and consider the initial plan that we hadmaybe it was better? Let me know what you think. In that case, your Action Step would be:

* Print out old draft, follow up with [colleagues name] re: alternative plan.

Sometimes you will find yourself waiting for a response to an email or a phone call. It is easy to forget something when it is in someone elses court! To trigger yourself to follow up if you dont hear back, you may want to create a

separate Action Step.

Action Steps arise from every idea exchange. Even the smallest of Action Steps, when captured, will make a big difference because they create momentum. A missed Action Step can cause miscommunications, more meetings, and could be the difference between success and failure in any project.

Here are some key practices:

Capture Action Steps everywhere. Ideas dont reveal themselves only in meetings, and neither should Action Steps. Ideas come up when you are reading an article, taking a shower, daydreaming, or getting ready for bed. If you think of someone that you met with a month ago regarding a certain project but have not yet followed up with, create an Action Step to follow up with XYZ regarding . . . If you are opening your mail and come across a wedding invitation, your Action Step is to RSVP.

Think of Action Steps expansivelyas anything you might want to doand capture all of them, not only the ones that arise during meetings.

Ideas dont reveal themselves only in meetings, and neither should Action Steps.

Having some sort of pad or recording device handy will enable you to capture actions as they come to mind. Our team developed the iPhone version of Action Method Online because users wanted a quick and anytime, anywhere way to capture Action Steps and assign them to a project. Whatever medium you choose to use for capturing Action Steps, it should always be readily available. Your system should also make it easy to return to your Action Steps at a later time and distinctly recall what you were thinking. And, most important, you must always be able to distinguish Action Steps from Referencesthe regular notes and non-actionable ideas that you may have also written down.

An unowned Action Step will never be taken.

Every Action Step must be owned by a single person. While some Action Steps may involve the input of different people, accountability must reside in one individuals hands at the end of the day. Some people who lead teams or have assistants will capture Action Steps and delegate them to others. However, even when the onus to complete an Action Step has been delegated to someone else, the Action Step must still be owned by the person ultimately responsible.

Every Action Step must be owned by a single person.

The reason comes down to accountability. The practice of simply emailing someone a task to complete does not provide any assurance that it will be completed. For this reason, Action Steps that you are ultimately responsible for should remain on your list until completedeven when you have delegated them to others. Simply marking that the Action Step has been delegated and to whom is sufficient:

* Print out old draft, follow up with Alex re: other plan (Oscar is doing).

Treat managerial Action Steps differently.

Aside from the Action Steps that you and only you can do, there are three other types of Action Steps you should keep in mind as the leader of a project. The first type is delegated Action Steps, which we just discussed above. The second type is Ensure Action Steps. Sometimes you will want to create an Action Step to ensure that something is completed properly in the future. Rather than being a nag to your team, you can create an Action Step that starts with the word ensure. For example, Ensure that Dave updated the article with the new title. If you use a digital tool to manage your Action Steps, you can always search by the word ensure (to only view Action Steps that start with ensure) and spend some time verifying that these items have been done. Creating Ensure Action Steps is a better alternative then sending numerous reminder emails to your team when you are worried about something slipping through the cracks.

The last type of managerial Action Step is the Awaiting Action Step. When you leave a voicemail for someone, send a message to a potential customer, or respond to an email and clear it from your inbox, youre liable to forget to follow-up if the person fails to respond. By creating an Action Step that starts with Awaiting, you can keep track of every ball that is out of your court. When I respond via email to a potential client, I create an Action Step like Awaiting confirmation from Joe at Apple re: consultation, saved in the project Consulting Work. In my online task manager I will set a target date for one week later. After a week passes, I will be reminded to follow up. Sometimes I will search all my Action Steps, across projects, with the word awaiting and dedicate an hour to follow up on everything.

Foster an action-oriented culture.

Your team needs an action-oriented culture to capitalize on creativity. It

may feel burdensome or even a bit aggressive to ask people to capture an Action Step on paper, but fostering a culture in which such reminders are welcome helps ensure that Action Steps are not lost. Some of the most productive teams I have observed are comfortable making sure that others are capturing Action Steps. Aside from friendly questioning along the lines of Did you capture that? some teams take a few minutes at the end of every meeting to go around the table and allow each person to recite the Action Steps that he or she captured. Doing so will almost always reveal a missed Action Step or a duplication on two peoples lists. This simple practice can save time and prevent situations in which, weeks later, people are wondering who was doing what or how something got lost in the shuffle.

Your team needs an action-oriented culture to capitalize on creativity.

Attraction breeds loyalty.

When it comes to the mechanics of capturing action steps, you should find the solution that fits you best. Keep in mind that the design of your productivity tools will affect how eager people are to use them. Attraction often breeds commitment: if you enjoy your method for staying organized, you are more likely to use it consistently over time. For this reason, little details like the colors of folders you use or the quality of the paper can actually help boost your productivity.

In her book The Substance of Style, journalist Virginia Postrel shares an anecdote about usability guru Donald Normans assertion that attractive things work better. When the first color computer monitors became available commercially, Norman wanted to justify the value of buying the expensive monitors instead of the standard black-and-white displays. Nowadays, this decision might seem obvious, but back in the day before the World Wide Web and color printers, the value of a color monitor for functions like word processing was unproven.

I got myself a color display and took it home for a week, Norman recalled. When the week was over, I had two findings. The first finding was that I was right, there was absolutely no advantage to color. The second was that I was not going to give it up. In her analysis of Normans findings, Postrel explains, The difference lay not in information processing but in affect, in how full-color monitors made people feel about their work. In other words, the aesthetics of

the tools you use to make ideas happen matter.

As you move through your day of meetings, brainstorms, and other occasions of creativity, you will start to accumulate Action Steps, References, and Backburner Items. Handouts, random pages of notes, emails, and social network messages will build up all around you. Often these items will get buried in notebooks, pockets, online inboxes, and computer files almost as soon as they are created or received. Ideally, in your written notes you will have kept your

Action Steps separate from everything else. However, you will still need time for processing going through all of your days notes and communications, and distilling them all down to the primary elements. For those who still take paper notes and appreciate tangible project management, you will want to use a tangible inboxa general pile of stuff that has yet to be classified. Most productivity frameworkslike David Allens Getting Things Donesuggest such a central clearinghouse for all of the stuff that you accumulate but cant immediately execute or file. This inbox is not a final destination, but rather a transit terminal where items await processing. During a busy day of meetings, you will not have time to start taking action or filing things away.

How about all of the digital stuff that flows in every day? Your email inbox is the primary landing spot, but information also flows into other online applications. While your tangible inbox, sitting on your desk, is singular, the digital equivalent is becoming more of a collective. Ideally, you should set your social network profiles to forward messages to your email inbox for the sake of aggregation. When you commit time for processing, youll want to limit the number of places you need to visit.

Ideally, you should set your social network profiles to forward messages to your email inbox for the sake of aggregation.

If you cant aggregate the flow of emails and other digital communications in the same place, then you need to define the various pieces of your collective digital inbox. For example, my collective digital inbox includes my email program (which receives messages from all other networks), a Twitter aggregator, and the inbox in my task management application (where I accept/reject stuff sent from my colleagues who use the same application and then manage this information by project). When the time comes for processing, these are the three digital places I need to visit, along with the tangible inbox full of papers

on my desk.

As you can see, the inbox of the 21st century varies for everyone. You must concretely define your collective inbox before you start processing. Peace of mind and productivity starts when you know where everything is. The combined inbox says, Dont worry, all of your stuff (and the Action Steps, Backburner Items, and References contained within) are in a defined place, waiting for you and ready to be sorted.

Peace of mind and productivity starts when you know where everything is.

If you live a digital lifestyle, your ability to process your inbox may be at particular risk without some sense of discipline. The reason: in the era of mobile devices and constant connectivity, it has become all too easy for others to send us messages. As such, our ability to control our focus is often crippled by the never-ending flow of incoming phone calls, emails, text messages, and in-person interruptionsnot to mention messages from other online services. Thus it is important that you avoid the trap of what I have come to call reactionary workflow.

The state of reactionary workflow occurs when you get stuck simply reacting to whatever flows into the top of an inbox. Instead of focusing on what is most important and actionable, you spend too much time just trying to stay afloat. Reactionary workflow prevents you from being more proactive with your energy. The act of processing requires discipline and imposing some blockades around your focus. For this reason, many leaders perform their processing at night or at a time when the flow dies down.

Time spent processing is arguably the most valuable and productive time of your day. While processing, you will sort everything and distinguish Action Steps, Backburner Items, and References. With Action Steps, you will decide what can be done quickly and what must be tracked over time by projectand possibly delegated. With other materials, you will make judgments about what can be thrown away and what must be filed.

As you start to tackle your collective inbox, you will realize that any inbox, on its own, is a pretty bad action management tool. It is difficult to keep your Action Steps separate from References and other noise. The constant stream of email certainly doesnt help. In addition to email, you may also receive other types of incoming communications in the form of Tweets, Facebook messages,

etc. Some are actionable, or contain actionable elements, while others are simply for reference (or for fun).

Time spent processing is arguably the most valuable and productive time of your day.

Given the unyielding flow of communications, you will want to capture and manage your Action Steps separately. Despite the many tricks involving action subfolders and other ways to manage and prioritize Action Steps within an email system, there is nothing better than giving Action Steps their own sacred space to be managed by project.

The Action Method suggests that Action Steps should be managed separately from communications. The solution can be as simple as a spreadsheet or to-do list where all Action Steps are tracked (and can be sorted by project name or due date). You can also make use of more advanced project management applications that manage Action Steps and support delegation and collaboration. What you want to avoid is a mishmash of actionable items amidst hundreds of verbose emails and other messages scattered in various places.

2.2 Gamification

Following the success of the location-based service Foursquare, the idea of using game design elements in non-game contexts to motivate and increase user activity and retention has rapidly gained traction in interaction design and digital marketing. Under the moniker "gamification", this idea is spawning an intense public debate as well as numerous applications ranging across productivity, finance, health, education, sustainability, as well as news and entertainment media. Several vendors now offer "gamification" as a software service layer of reward and reputation systems with points, badges, levels and leader boards.

This commercial deployment of 'gamified' applications to large audiences potentially promises new, interesting lines of inquiry and data sources for human-computer interaction (HCI) and game studies and indeed, "gamification" is increasingly catching the attention of researchers [24,48,58].

Whereas "serious game" describes the design of full-fledged games for non-entertainment purposes, "gamified" applications merely incorporate elements

of games (or game "atoms" [10]). Of course, the boundary between "game" and "artifact with game elements" can often be blurry is Foursquare a game or a "gamified" application? To complicate matters, this boundary is empirical, subjective and social: Whether you and your friends 'play' or 'use' Foursquare depends on your (negotiated) focus, perceptions and enactments. The addition of one informal rule or shared goal by a group of users may turn a 'merely' 'gamified' application into a 'full' game. Within game studies, there is an increasing acknowledgement that any definition of 'games' has to go beyond properties of the game artifact to include these situated, socially constructed meanings [19,67]. For the present purpose, this means that (a) artifactual as well as social elements of games need to be considered, and (b) artifactual elements should be conceived more in terms of affording gameful interpretations and enactments, rather than being gameful. Indeed, the characteristic of 'gamified' applications might be that compared to games, they afford a more fragile, unstable 'flicker' of experiences and enactments between playful, gameful, and other, more instrumental-functionalist modes.

As can be seen, this level model distinguishes interface design patterns from game design patterns or game mechanics. Although they relate to the shared concept of pattern languages [26], unlike interface design patterns, neither game mechanics nor game design patterns refer to (prototypical) implemented solutions; both can be implemented with many different interface elements. Therefore, they are more abstract and thus treated as distinct.

So to restate, whereas serious games fulfill all necessary and sufficient conditions for being a game, gamified applications merely use several design elements from games. Seen from the perspective of the designer, what distinguishes gamification from regular entertainment games and serious games is that they are built with the intention of a system that includes elements from games, not a full game proper. From the user perspective, such systems entailing design elements from games can then be enacted and experienced as games proper, gameful, playful, or otherwise this instability or openness is what sets them apart from games proper for users.

Similar to serious games, gamification uses elements of games for purposes other than their normal expected use as part of an entertainment game. Now normal use is a socially, historically and culturally contingent category. How-

ever, it is reasonable to assume that entertainment currently constitutes the prevalent expected use of games. Likewise, joy of use, engagement, or more generally speaking, improvement of the user experience represent the currently predominant use cases of gamification (in the definition proposed in this paper, gameful experiences are the most likely design goal). Still, we explicitly suggest not delimiting gamification to specific usage contexts, purposes, or scenarios. Firstly, there are no clear advantages in doing so. Secondly, the murkiness of the discourse on serious games can be directly linked to the fact that some authors initially tied the term to the specific context and goal of education and learning, whereas serious games proliferated into all kinds of contexts [61]. Thus, in parallel to Sawyers taxonomy of serious games [61], we consider different usage contexts or purposes as potential subcategories: Just as there are training games, health games, or newsgames, there can be gameful design or gamification for training, for health, for news, and for other application areas.

To summarize: "Gamification" refers to the use (rather than the extension) of design (rather than game-based technology or other game- related practices) elements (rather than full-fledged games) characteristic for games (rather than play or playfulness) in non-game contexts (regardless of specific usage intentions, contexts, or media of implementation).

This definition contrasts gamification against other related concepts via the two dimensions of playing/gaming and parts/whole. Both games and serious games can be differentiated from gamification through the parts/whole dimension. Playful design and toys can be differentiated through the playing/gaming dimension (Figure 1). In the broader scheme of trends and concepts identified as related, we find gamification or gameful design situated as follows: Within the socio-cultural trend of ludification, there are at least three trajectories relating to video games and HCI: the extension of games (pervasive games), the use of games in non-game contexts, and playful interaction. The use of games in non-game contexts falls into full-fledged games (serious games) and game elements, which can be further differentiated into game technology, game practices, and game design. The latter refers to gamification (Figure 2).

2.2.1 Game mechanics list

2.2.2 Appropriate game mechanics for the basic project management

Summarizing the conducted results, the following list of requirements for successful productivity application was made: Achievement game mechanics supports desire to perform tasks Pride game mechanics supports retention of joy from executed tasks Avoidance game mechanics helps a person return to application Cascading Information Theory allows to introduce difficult concepts of GTD (Getting Things Done) and increase a persons activity in the application Communal Discovery is used in collaborative tasks / task delegation Progression Dynamic helps visualize improvement and progress to a goal. Techniques used in animation to create believable characters can help to establish an emotional connection to the person

KPI – Key Performance Indicator

2.3 Goal commitment formula

Our lack of adequate emphasis on motivation at work has, in my view, retarded our attempt to maximize performance. In their review of leadership studies, Hogan Curphy and Hogan (1994) found that only about 30 percent of line managers are able to adequately motivate the people who report to them. They imply that in most circumstances, motivation accounts for about half of all performance results.

The late Tom Gilbert, one of the clearest thinkers in performance improvement, was fond of saying that when two people had equal abilities, the enthusiastic member of the pair would achieve about 70 percent more than the unenthusiastic person. Even more troubling is that evidence that a majority of the published studies of organizational development strategies that report measured increases in motivation are fatally flawed (Newman, Edwards and Raju, 1989; Roberts and Robertson, 1992). Strategies that may not work as powerfully or as consistently as claimed include popular employee empowerment strategies, contests, job redesign, leaderless teams and various performance recognition techniques.

If you doubt the importance of motivation in performance, check your an-

swers to the following questions. Why is enthusiastic commitment to work goals so difficult to achieve with many people, even when we pay people well? Why is it onerous, and sometimes impossible, to convince people at work to persist at vital work goals when they encounter interesting but much less important alternative goals? Why do employee reward programs and empowerment strategies sometimes fail or backfire? Do we have to pay people more to get them to work harder? Are people from different cultural backgrounds motivated differently? Is the motivation of knowledge work similar to the motivation of physical work? Is team motivation different than individual motivation? Why is it that committed people often fail to invest enough effort to fully achieve work goals even though they believe the goals to be important to them and to their organizations? These are some of the questions that trouble human performance consultants.

In the CANE model, motivation is defined as two interlinked processes. The first process leads us to make a commitment to a performance goal and persist the face of distractions from appealing but less important alternative goals. The second motivation process is concerned with the amount and quality of the mental effort people invest in achieving the knowledge component of performance goals. These two motivation processes, committed, active and sustained goal pursuit on the one hand, and necessary mental effort to tackle goal-related problems, on the other hand, are the primary motivation goals in the CANE model.

In todays complex work environments the variety of job tasks that confront all of us change constantly over time. We cannot commit ourselves equally to all tasks. We must prioritize and focus on important tasks in order to be successful. Commitment problems happen when people resist assigning adequate priority to important job tasks. Research on motivation suggests that people with commitment problems may avoid a task altogether and/or argue that the task is less important than some other set of tasks.

Three factors have been found to increase (or decrease) work goal commitment. The first factor is task assessment. All of us will analyze any task we are assigned to determine whether we can successfully complete the task. We all tend to ask ourselves two questions about new tasks - Can I do it? and Will I be permitted to do it?. If we think that we have the ability to accomplish the

goal and that we will be permitted to accomplish it, our commitment will increase (Bandura, 1997; Ford, 1992). If we doubt our ability or the organizations willingness to let us use our skills, commitment will decrease.

Emotion and commitment. The second factor influencing commitment is our mood or emotions. All positive emotions facilitate commitment and all powerful, negative emotions discourage goal commitment (Bower, 1983; 1995; Ford, 1992). This may seem like a minor issue but for temperamental people or in organizations where pressure is high and/or change is constant, negative emotional undercurrents can be strong. Angry or depressed people find it nearly impossible to make a commitment to work goals.

Values and commitment. The final factor that influences the strength of goal commitment is our personal value in the goal. It is my experience that values are the most important element in increasing or decreasing the strength of our commitments. Psychologists now have good evidence that the most important value at work is our belief about whether the achievement of a work goal will increase our personal control or effectiveness (Shapiro et al, 1996; Locke and Latham, 1990). The more we believe that achievement of a work goal will make us more successful, the higher our level of commitment to the goal. The reverse is also true. Few of us will give a high priority to tasks that we sincerely believe will lead us to fail or be perceived as incompetent Utility, Interest and Importance Values.

Task Assessment Solutions

Solving task assessment problems require that we convince people that they can do a job and that existing barriers to their performance will be removed. Pointing out familiar, past examples of job performance that are similar to the new task helps increase confidence. In addition, job aids can bolster confidence. Involving staff in the elimination of any procedural or policy barriers to performance reduces resistence based on task assessment. The service technicians had excellent job aids which increased their confidence about the form task. The key element here is to persuade or empower people to believe that they can succeed at the task they are avoiding. Bandura (1997) provides extensive examples of solutions in this area.

Mood solutions.

Mood problems often take more time to develop than task assessment or

value problems. I find mood problems to be key elements in organizations where a major culture or job change is occurring. This is particularly true in organizations that are changing from a civil service to a business culture.

Solutions that have been found to change mood states have included listening to positive mood music; writing or telling about a positive mood-related experience; watching a movie or listening to stories that emphasize positive mood states; and emotion control training through environmental control strategies including the choice of how we complete work tasks, adjusting work space and positive self talk.

Value solutions

The solution to most commitment problems and opportunities is to convince people that completing the task they are resisting will make them more effective and/or perceived as more effective. People simply will not do what they believe will make them less effective or less successful. Many people are suspicious of change simply because they feel that they will be perceived as less effective under novel, negative or uncertain conditions. They must be convinced that if they commit themselves to the avoided task(s) they will become significantly more effective or successful. The specific solution that accomplishes this goal may be quite different for different individuals and work cultures. Some organizations have adopted various employee empowerment solutions to value problems. In many empowerment settings staff are asked to choose their own work goals in order to get them to value their work. There is good research evidence that this is not necessary.

In cases where participatory goal setting is not possible, they find that value for the goal is enhanced if people perceive the goal to be: 1) assigned by a legitimate, trusted authority with an inspiring vision that reflects a convincing rationale for the goal (importance value), and who; 2) provides expectation of outstanding performance (importance value) and gives: 3) ownership to individuals and teams for specific tasks (interest value); 4) expresses confidence in individual and team capabilities (interest value) while; 5) providing feedback on progress that includes recognition for success and supportive but corrective suggestions for mistakes (utility value).

Motivation Solutions

Value problems are often multi-level issues in an organization. In this orga-

nization, there were a number of beliefs and patterns that had to be considered. The mangers of the technicians had their own motivational issues to handle. For example, the senior manager acting as sponsor for the motivation project placed a number of constraints on a value solution for the technicians.

Two types of motivation are important at work, persistence and mental effort. Commitment (persistence at a task) is increased by convincing people that: a) the organization will remove unnecessary barriers; b) that achievement of the work goal will make the person more personally effective; and c) that the manager requesting the goal is credible, trustworthy, optimistic (about the person or teams ability to achieve the goal), able to clearly communicate the vision connected to the goal and willing to give ownership for the accomplishment. Mental effort is enhanced by insuring that the goal assigned is very challenging. Managers must work with people to adjust their confidence level whenever they become over confident (and thus refuse to take responsibility for errors or poor performance) or under confident (and thus find an excuse to procrastinate or avoid the goal altogether).

2.4 Motivation in Daniel Pink's "Drive"

"THIS IS A BOOK about motivation. I will show that much of what we believe about the subject just isnt soand that the insights that Harlow and Deci began uncovering a few decades ago come much closer to the truth. The problem is that most businesses havent caught up to this new understanding of what motivates us. Too many organizationsnot just companies, but governments and nonprofits as wellstill operate from assumptions about human potential and individual performance that are outdated, unexamined, and rooted more in folklore than in science."

Societies, like computers, have operating systemsa set of mostly invisible instructions and protocols on which everything runs. The first human operating systemcall it Motivation 1.0was all about survival. Its successor, Motivation 2.0, was built around external rewards and punishments. That worked fine for routine twentieth-century tasks. But in the twenty-first century, Motivation 2.0 is proving incompatible with how we organize what we do, how we think about what we do, and how we do what we do. We need an upgrade.

When carrots and sticks encounter our third drive, strange things begin to happen. Traditional if-then rewards can give us less of what we want: They can extinguish intrinsic motivation, diminish performance, crush creativity, and crowd out good behavior. They can also give us more of what we dont want: They can encourage unethical behavior, create addictions, and foster short-term thinking. These are the bugs in our current operating system.

Carrots and sticks arent all bad. They can be effective for rule-based routine tasks because theres little intrinsic motivation to undermine and not much creativity to crush. And they can be more effective still if those giving such rewards offer a rationale for why the task is necessary, acknowledge that its boring, and allow people autonomy over how they complete it. For nonroutine conceptual tasks, rewards are more perilous particularly those of the if-then variety. But now that rewardsnoncontingent rewards given after a task is completecan sometimes be okay for more creative, right- brain work, especially if they provide useful information about performance.

Motivation 2.0 depended on and fostered Type X behaviorbehavior fueled more by extrinsic desires than intrinsic ones and concerned less with the inherent satisfaction of an activity and more with the external rewards to which an activity leads. Motivation 3.0, the upgrade thats necessary for the smooth functioning of twenty-first-century business, depends on and fosters Type I behavior. Type I behavior concerns itself less with the external rewards an activity brings and more with the inherent satisfaction of the activity itself. For professional success and personal fulfillment, we need to move ourselves and our colleagues from Type X to Type I. The good news is that Type Is are made, not bornand Type I behavior leads to stronger performance, greater health, and higher overall well-being.

Our default setting is to be autonomous and self-directed. Unfortunately, circumstancesincluding outdated notions of managementoften conspire to change that default setting and turn us from Type I to Type X. To encourage Type I behavior, and the high performance it enables, the first requirement is autonomy. People need autonomy over task (what they do), time (when they do it), team (who they do it with), and technique (how they do it). Companies that offer autonomy, sometimes in radical doses, are outperforming their competitors.

While Motivation 2.0 required compliance, Motivation 3.0 demands engagement. Only engagement can produce masterybecoming better at something that matters. And the pursuit of mastery, an important but often dormant part of our third drive, has become essential to making ones way in the economy. Mastery begins with flowoptimal experiences when the challenges we face are exquisitely matched to our abilities. Smart workplaces therefore supplement day-to-day activities with Goldilocks tasksnot too hard and not too easy. But mastery also abides by three peculiar rules. Mastery is a mindset: It requires the capacity to see your abilities not as finite, but as infinitely improvable. Mastery is a pain: It demands effort, grit, and deliberate practice. And mastery is an asymptote: Its impossible to fully realize, which makes it simultaneously frustrating and alluring.

Humans, by their nature, seek purposea cause greater and more enduring than themselves. But traditional businesses have long considered purpose ornamentala perfectly nice accessory, so long as it didnt get in the way of the important things. But thats changingthanks in part to the rising tide of aging baby boomers reckoning with their own mortality. In Motivation 3.0, purpose maximization is taking its place alongside profit maximization as an aspiration and a guiding principle. Within organizations, this new purpose motive is expressing itself in three ways: in goals that use profit to reach purpose; in words that emphasize more than self-interest; and in policies that allow people to pursue purpose on their own terms. This move to accompany profit maximization with purpose maximization has the potential to rejuvenate our businesses and remake our world.

For example, in April 2008, Vermont became the first U.S. state to allow a new type of business called the low-profit limited liability corporation. Dubbed an L3C, this entity is a corporationbut not as we typically think of it. As one report explained, an L3C operate[s] like a for-profit business generating at least modest profits, but its primary aim [is] to offer significant social benefits.

Meanwhile, Nobel Peace Prize winner Muhammad Yunus has begun creating what he calls social businesses. These are companies that raise capital, develop products, and sell them in an open market but do so in the service of a larger social missionor as he puts it, with the profit-maximization principle replaced by the social-benefit principle.

Motivation 2.0 suffers from three compatibility problems. It doesnt mesh with the way many new business models are organizing what we dobecause were intrinsically motivated purpose maximizers, not only extrinsically motivated profit maximizers. It doesnt comport with the way that twenty-first-century economics thinks about what we dobecause economists are finally realizing that were full-fledged human beings, not single-minded economic robots. And perhaps most important, its hard to reconcile with much of what we actually do at workbecause for growing numbers of people, work is often creative, interesting, and self-directed rather than unrelentingly routine, boring, and other-directed. Taken together, these compatibility problems warn us that somethings gone awry in our motivational operating system. But in order to figure out exactly what, and as an essential step in fashioning a new one, we need to take a look at the bugs themselves.

One of Lepper and Greenes early studies (which they carried out with a third colleague, Robert Nisbett) has become a classic in the field and among the most cited articles in the motivation literature. The three researchers watched a classroom of preschoolers for several days and identified the children who chose to spend their free play time drawing. Then they fashioned an experiment to test the effect of rewarding an activity these children clearly enjoyed.

The researchers divided the children into three groups. The first was the expected- award group. They showed each of these children a Good Player certificateadorned with a blue ribbon and featuring the childs nameand asked if the child wanted to draw in order to receive the award. The second group was the unexpected-award group. Researchers asked these children simply if they wanted to draw. If they decided to, when the session ended, the researchers handed each child one of the Good Player certificates. The third group was the no-award group. Researchers asked these children if they wanted to draw, but neither promised them a certificate at the beginning nor gave them one at the end.

Two weeks later, back in the classroom, teachers set out paper and markers during the preschools free play period while the researchers secretly observed the students. Children previously in the unexpected-award and no-award groups drew just as much, and with the same relish, as they had before the experiment. But children in the first groupthe ones whod expected and then received an

awardshowed much less interest and spent much less time drawing. 2 The Sawyer Effect had taken hold. Even two weeks later, those alluring prizesso common in classrooms and cubicleshad turned play into work.

To be clear, it wasnt necessarily the rewards themselves that dampened the childrens interest. Remember: When children didnt expect a reward, receiving one had little impact on their intrinsic motivation. Only contingent rewardsif you do this, then youll get thathad the negative effect. Why? If-then rewards require people to forfeit some of their autonomy. Like the gentlemen driving carriages for money instead of fun, theyre no longer fully controlling their lives. And that can spring a hole in the bottom of their motivational bucket, draining an activity of its enjoyment.

Lepper and Greene replicated these results in several subsequent experiments with children. As time went on, other researchers found similar results with adults. Over and over again, they discovered that extrinsic rewardsin particular, contingent, expected, if-then rewardssnuffed out the third drive.

These insights proved so controversialafter all, they called into question a standard practice of most companies and schoolsthat in 1999 Deci and two colleagues reanalyzed nearly three decades of studies on the subject to confirm the findings. Careful consideration of reward effects reported in 128 experiments lead to the conclusion that tangible rewards tend to have a substantially negative effect on intrinsic motivation, they determined. When institutionsfamilies, schools, businesses, and athletic teams, for examplefocus on the short-term and opt for controlling peoples behavior, they do considerable long-term damage.

As one leading behavioral science textbook puts it, People use rewards expecting to gain the benefit of increasing another persons motivation and behavior, but in so doing, they often incur the unintentional and hidden cost of undermining that persons intrinsic motivation toward the activity.

This is one of the most robust findings in social scienceand also one of the most ignored. Despite the work of a few skilled and passionate popularizersin particular, Alfie Kohn, whose prescient 1993 book, Punished by Rewards, lays out a devastating indictment of extrinsic incentiveswe persist in trying to motivate people this way.

Of course, all goals are not created equal. Andlet me emphasize this point-goals and extrinsic rewards arent inherently corrupting. But goals are more

toxic than Motivation 2.0 recognizes. In fact, the business school professors suggest they should come with their own warning label: Goals may cause systematic problems for organizations due to narrowed focus, unethical behavior, increased risk taking, decreased cooperation, and decreased intrinsic motivation. Use care when applying goals in your organization.

Offer a rationale for why the task is necessary.A job thats not inherently interesting can become more meaningful, and therefore more engaging, if its part of a larger purpose. Explain why this poster is so important and why sending it out now is critical to your organizations mission. Acknowledge that the task is boring.This is an act of empathy, of course. And the acknowledgment will help people understand why this is the rare instance when if-then rewards are part of how your organization operates.

Allow people to complete the task their own way. Think autonomy, not control. State the outcome you need. But instead of specifying precisely the way to reach ithow each poster must be rolled and how each mailing label must be affixedgive them freedom over how they do the job.

Heres what you shouldnt do: Offer an if-then reward to the design staff. Do not stride into their offices and announce: If you come up with a poster that rocks my world or that boosts attendance over last year, then youll get a ten-percent bonus. Although that motivational approach is common in organizations all over the world, its a recipe for reduced performance. Creating a poster isnt routine. It requires conceptual, breakthrough, artistic thinking. And as weve learned, if-then rewards are an ideal way to squash this sort of thinking. Your best approach is to have already established the conditions of a genuinely motivating environment. The baseline rewards must be sufficient. That is, the teams basic compensation must be adequate and fairparticularly compared with people doing

Similar work for similar organizations. Your nonprofit must be a congenial place to work. And the people on your team must have autonomy, they must have ample opportunity to pursue mastery, and their daily duties must relate to a larger purpose. If these elements are in place, the best strategy is to provide a sense of urgency and significanceand then get out of the talents way. But you may still be able to boost performance a bitmore for future tasks than for this onethrough the delicate use of rewards. Just be careful. Your efforts will

backfire unless the rewards you offer meet one essential requirement. And youll be on firmer motivational footing if you follow two additional principles.

The essential requirement: Any extrinsic reward should be unexpected and offered only after the task is complete.

Holding out a prize at the beginning of a projectand offering it as a contingencywill inevitably focus peoples attention on obtaining the reward rather than on attacking the problem. But introducing the subject of rewards after the job is done is less risky. In other words, where if-then rewards are a mistake, shift to now that rewardsas in Now that youve finished the poster and it turned out so well, Id like to celebrate by taking you out to lunch.

As Deci and his colleagues explain, If tangible rewards are given unexpectedly to people after they have finished a task, the rewards are less likely to be experienced as the reason for doing the task and are thus less likely to be detrimental to intrinsic motivation.

First, consider nontangible rewards. Praise and positive feedback are much less corrosive than cash and trophies.

Second, provide useful information.

Amabile has found that while controlling extrinsic motivators can clobber creativity, informational or enabling motivators can be conducive to it.

2.5 Mihaly Csikszentmihalyi's concept of "Flow"

The author has been studying for over 20 years the states of optimal experience–those times when people report feelings of concentration and deep enjoyment. These investigations have revealed that what makes experience genuinely satisfying is a state of consciousness called flow–a state of concentration so focused that it amounts to absolute absorption in an activity.

Everyone experiences flow from time to time and will recognize its characteristics: people typically feel strong, alert, in effortless control, unselfconscious, and at the peak of their abilities. Both a sense of time and emotional problems seem to disappear, and there is an exhilarating feeling of transcendence. Flow: The Psychology of Optimal Experience describes how this pleasurable state can be controlled, and not just left to chance, by setting ourselves challenges–tasks that are neither too difficult nor too simple for our abilities. With such goals,

we learn to order the information that enters consciousness and thereby improve the quality of our lives.

The studies have suggested that the phenomenology of enjoyment has eight major components. When people reflect on how it feels when their experience is most positive, they mention at least one, and often all, of the following: 1. We confront tasks we have a chance of completing; 2. We must be able to concentrate on what we are doing; 3. The task has clear goals; 4. The task provides immediate feedback; 5. One acts with deep, but effortless involvement, that removes from awareness the worries and frustrations of everyday life; 6. One exercises a sense of control over their actions; 7. Concern for the self disappears, yet, paradoxically the sense of self emerges stronger after the flow experience is over; and 8. The sense of duration of time is altered. The combination of all these elements causes a sense of deep enjoyment that is so rewarding people feel that expending a great deal of energy is worthwhile simply to be able to feel it.

A Challenging Activity that Requires Skills Optimal experiences are reported to occur within sequences of activities that are goal-directed and bounded by rules–activities that require the investment of psychic energy (attention) and that could not be done without skills. Please note that activities do not need to be physical and skills also need not be physical skills. For instance, the most frequently mentioned enjoyable activity the world over was reading, followed closely by being with other people. For those who do not have the right skills, an activity is not challenging; it is simply meaningless. Challenges of competition were found to be stimulating and enjoyable. But when beating the opponent takes precedence in the mind over performing as well as possible, enjoyment tends to disappear. Competition is enjoyable only when it is a means to perfect one's skills; when it becomes an end in itself, it ceases to be fun.

The Merging of Action and Awareness One of the most universal and distinctive features of optimal experience is the people become so involved in what they are doing that the activity becomes spontaneous, almost automatic; they stop being aware of themselves as separate from the actions they are performing. It often requires strenous physical exertion, or highly disciplined mental activity to enter a continuous flow.

Clear Goals and Feedback Unless a person learns to set goals and to rec-

ognize and gauge feedback in their activities, she will not enjoy them. For activities that are creative or open-ended in nature, a person must develop a strong sense of what she intends to do or negotiate goals and rules during the activity. These goals and rules provide benchmarks for feedback. The kind of feedback we work toward is in, and of itself, often unimportant. What makes feedback valuable is the symbolic message it contains: that I have succeeded in my goal.

Concentration on the Task at Hand One of the most frequently mentioned dimensions of the flow experience is that, while it lasts, one is able to forget all the unpleasant aspects of life. The task requires such concentration that only a very select range of information can be allowed into awareness.

The Paradox of Control The flow experience is typically described as involving a sense of control–or more precisely, as lacking the sense of worry about losing control that is typical in many situations of normal life. What people enjoy is not the sense of being in control, but the sense of exercising control in difficult situations. However, when a person becomes dependent on the ability to control an enjoyable activity then he loses the ultimate control: the freedom to determine the content of consciousness. While experiences are capable of improving the quality of existence by creating order in the mind, they can also become addictive, at which point the self becomes captive of a certain kind of order, and is then unwilling to cope with the ambiguities of life.

The Loss of Self-Consciousness When in a flow experience, what slips below the threshold of awareness is the concept of self, the information we use to represent to ourselves who we are. And being able to forget temporarily who we are seems to be very enjoyable. When not preoccupied with our selves, we actually have a chance to expand the concept of who we are. Loss of self-consciousness can lead to self-transcendence, to a feeling that the boundaries of our being have been pushed forward.

The Transformation of Time One of the most common descriptions of optimal experience is that time no longer seems to pass the way it ordinarily does. Generally, after the experience we do not know where the time went; however, during the actual experience, time seems to stand still. The key element of an optimal experience is that it is an end in itself. It is an autotelic experience. The term "autotelic" derives from two Greek words, "auto" meaning self, and

"telos" meaning goal. It refers to a self-contained activity, one that is done not with the expectation of some future benefit, but simply because the doing itself is the reward. Teaching children in order to turn them into good citizens is not autotelic, whereas teaching them because one enjoys interacting with children is. Most enjoyable activities are not natural; they demand an effort that initially one is reluctant to make. But once the interaction starts to provide feedback to the person's skills, it usually begins to be intrinsically rewarding. Flow in the family context has five characteristics: Clarity: children know what parents expect from them; Centering: children know that their parents are interested in what they are doing in the present; Choice: children feel that they have a variety of possibilities from which to choose; Commitment: trust that allows the child to feel comfortable enough to set aside the shield of defenses and become unself-consciously involved; and Challenge: providing increasingly complex opportunities for action.

2.6 General guidelines by XXXXXX

3. Productivity mobile applications analysis

3.1 Popular applications and their description

Habit List. http://habitlist.com/ Habit List is an application to create good habits and break unhealthy ones. Habit List allows users to track habits and provide visual representations on progress. This encourages more success by seeing the results of hard work or lack thereof. The app is quick to load and the design is simple but effective. Its essentially a list of items that a user would like to complete on a certain schedule. Each habits status is communicated through a green or red indicator. Inside the colored indicator is the streak of days that a habit has been completed or not been completed. Carrot. http://meetcarrot.com/ Carrot encourages users toget tasks done in a timely matter and they're rewarded with points that lead to unlockable features. But if they don't accomplish tasks then the app will get angry. Clear. http://www.realmacsoftware.com/clear/ Clear uses the hierarchy of color, which is utilized to represent priority within the app. The idea is that to-do list becomes a heat map, where the warmest areas (i.e. the ones with the strongest shade of red) are the tasks should be attended first. Remember the milk. http://www.rememberthemilk.com/ Remember the milk offers a lot of GTD functionality: tag clouds, custom task lists, geo location, Gmail extension, Google Calendar integration, etc. Wunderlist 2. https://www.wunderlist.com Wunderlist 2 comes with a number of features: online syncing, filtered lists, and Facebook integration, collaboration. The design spans across the multitude of different devices Wunderlist works on, giving it a single, unified interface. Flow. http://www.getflow.com/ Flow provides all the collaboration tools needed to manage a project (files, deadlines, tasks and discussion) online in one centralized place for everyone. It allows to discuss projects, set deadlines, take notes, and keep everyone on the same page online, even when theyre out of the office. Things. http://culturedcode.com/things/iphone/ Things features a daily review to plan a day on the go: to-dos that are scheduled for today automatically appear at the top of Today list, together with to-dos that have become due; instant action to find the right tasks for the current context using tags;

predefined lists, such as today, next, scheduled, someday.

3.2 Game mechanics in use

Achievement definition: A virtual or physical representation of having accomplished something. These are often viewed as rewards in and of themselves. Pride definition: the feeling of ownership and joy at an accomplishment. Avoidance definition: the act of inducing player behavior not by giving a reward, but by not instituting a punishment. Produces consistent level of activity, timed around the schedule. Communal Discovery definition: The game dynamic wherein an entire community is rallied to work together to solve a riddle, a problem or a challenge. Immensely viral and very fun. Cascading Information Theory definition: The theory that information should be released in the minimum possible snippets to gain the appropriate level of understanding at each point during a game narrative. Progression dynamic definition: this is a dynamic in which success is granularly displayed and measured through the process of completing itemized tasks. Most of the listed applications made a different approach to these mechanics. But only one application tried to develop an emotional connection to a person. It is Carrot, which uses natural language and color-coded emotions in order to response to actions of a person. If a person didnt achieve anything in a period of time, Carrot becomes angry. In addition, this application used the most of introduced game mechanics.

3.3 What to learn from productivity apps

Summarizing the conducted results, the following list of requirements for successful productivity application was made: Achievement game mechanics supports desire to perform tasks Pride game mechanics supports retention of joy from executed tasks Avoidance game mechanics helps a person return to application Cascading Information Theory allows to introduce difficult concepts of GTD (Getting Things Done) and increase a persons activity in the application Communal Discovery is used in collaborative tasks / task delegation Progression Dynamic helps visualize improvement and progress to a goal. Techniques used in animation to create believable characters can help to establish an emotional

connection to the person

4. Methodology for stimulating high-performance of project participants

4.1 Team-wide productivity

Motivating a team is often more challenging than moti- vating a single individual. Individuals within teams operate with different goals, values, beliefs, and expectations. Yet the variety of team member personalities can be a positive force if each performer con- tributes his or her unique capabilities when and where needed.

The first critical issue in team motivation is to be clear about the definition of a team. Nearly everyone who studies teams emphasizes that it is unnecessary to use team motivation strategies when teams are defined as any group of two or more people with similar skills who are simply working together to achieve a common goal (Bandura, 1997). For a team to exist (for motivational purposes), team members must play different roles or bring different skills to the table. Those different skills must be required to achieve team goals. So a team is an inter- dependent group of individuals, each possessing a different set of skills but who collectively possess all of the skills required to achieve team goals.

Foster Mutual Respect for the Expertise of All Members Teams on which one or more members believe that they are working with people who lack adequate skills to achieve team goals have a major motivational problem. In some cases, this belief is simply incorrect. Highly competitive people sometimes distort the real situation and develop the self-protective view that one or more people on their team are inadequate. Competitive spirit is good. But bolstering self-confidence at the expense of others is immature and destructive. Bandura (1997) describes many studies in a variety of fields where weak link doubts about team mem- ber expertise have significantly reduced team effectiveness. Even though all team members vary in their expertise levels, when individuals respect and support one another, less-able team members tend to perform significantly better and work hard over time to increase their skills. Since individual team members tend to be self-focused and so think more about their own

contributions and ability, team members need to be reminded about the skills of other members. One effec- tive way to accomplish this task is to actively attribute suc- cesses to each team members expertise.

Help Weaker Members Believe That Their Effort Is Vital to Team Success Occasionally teams must accommodate members who are novices or who for some reason are not able to do the best job for the team. When teams cant replace weaker members, what works best to preserve team motivation? Jackson and LePine (2003) have recent and solid evidence that when team members believe that their weakest member is merely inexperienced or has faltered for some uncontrollable rea- son (for example, illness, accident, or a family crisis) and can improve, they will give support provided that the per- son is investing effort to do so. The biggest motivational challenge on a team is faced by the weakest member. That individual must believe that what he or she contributes to the team is vital to the teams success and that the other members expect him or her to improve and succeed. Feedback to members who are working to improve must emphasize effort, not ability. When they make progress, it is best to attribute the progress to effort. When no progress is forthcoming, they need to be urged to get busy, get serious and work harder. Avoid attributing suc- cess or failure to ability. Belief that performance is due to ability tends to discourage hard work.

In many teams the motivational challenge is not a weak link, but instead a lack of cooperation and collaboration.

Support a Shared Belief in the Teams Cooperative Capabilities Healthy teams are made up of team players who cooperate with each other. One un-cooperative person can damage the motivation of even the most capable team. The obvious example is the arrogant, self-focused prima donna who invests most of his or her effort trying to look good with managers and clientsat the expense of the team. Less obvi- ous but equally destructive is the outwardly supportive but silently devious back-stabber, whose primarily goal is to make his or her own work highly visible.

Hold Individual Members Accountable for Contributions to the Team Ef-fort One of the first team motivation studies (described in Williams, Karau, Bourgeois, 1993), performed just after the turn of the century, established the principle that has been called social loafing. When people pulled as hard as pos-

sible against a rope connected to a strain gage, their best effort was recorded. When another person was added to the rope and two people pulled together, each person invested less effort in a collaborative effort than he or she did when alone. As more people were added to the rope, each person pulled less forcefully. When interviewed, most people seem unaware that they are not working as hard in a group situation as they did when alone.

Direct the Teams Competitive Spirit Outside the Team and the Organization Competition can be highly motivating for individuals or teams. Salespeople seem to thrive on it, and many people who are raised in Western cultural traditions seem to like a bit of it. One of the most common motivational team-building exercises favored by organizational consultants is a field experience where teams compete with other teams to bond and build team spirit. These events are scheduled off site and are ideally held in unfamiliar settings to interrupt habitual patterns formed at work for relating to others. Teams are challenged to do something highly novel, such as build structures or navigate difficult terrain to reach a tar- get sooner or more effectively than other teams. Individuals are asked to notice how hard they are working, how much they are collaborating, and whether they have a real desire to win.

Teams are defined as collections of individuals with differ- ent skill sets working together to achieve goals that require members to collaboratively apply their different skills. Collections of individuals with similar skills who tackle problems do not require team motivation strategies. In addi- tion to motivational strategies that work with individuals, interdependent teams are most motivated when they trust both the expertise and collaborativeness of other team mem- bers as well as the determination of weaker members on Direct the Teams Competitive Spirit Outside the Team and the Organization Competition can be highly motivating for individuals or teams. Salespeople seem to thrive on it, and many people who are raised in Western cultural traditions seem to like a bit of it. One of the most common motivational team-build- ing exercises favored by organizational consultants is a field experience where teams compete with other teams to bond and build team spirit. These events are scheduled off site and are ideally held in unfamiliar settings to interrupt habitual patterns formed at work for relating to others. Teams are challenged to do something highly novel, such as build structures or navigate difficult terrain to reach a tar-

get sooner or more effectively than other teams. Individuals are asked to notice how hard they are working, how much they are collaborating, and whether they have a real desire to win. In general, team-building exercises have been found to be very effective, but they also have a potentially ugly, unin- tended side effect. Druckman and Bjork (1994) reviewed all studies of team building for the US National Academy of Sciences. The variety of team-building methods shared the common goal of attempting to get members of work teams to bond, collaborate, and work efficiently toward common goals by competing with other teams. The researchers con- cluded that many different approaches worked, but they were surprised to find that after team-building exercises, a significant number of teams were competing in a nearly sui- cidal fashion with other teams in their own organization. Stories include misguided team members who were found to be modifying or deleting the electronic files, intentionally misplacing or rerouting team resources, and spreading negative rumors about members of other teams in their orga- nizations. Apparently, fostering constant, intense rivalry can help when it is directed at the organizations competi- tion, but it can also engender a destructive level of internal competition and focus attention and energy away from orga- nizational goals. The obvious motivational issue in this sit- uation is to make certain that team-building exercises focus the teams competitive energy on competing organizations not on other teams within the same organization.

Summary

Teams are defined as collections of individuals with different skill sets working together to achieve goals that require members to collaboratively apply their different skills. Collections of individuals with similar skills who tackle problems do not require team motivation strategies. In addition to motivational strategies that work with individuals, interdependent teams are most motivated when they trust both the expertise and collaborativeness of other team members as well as the determination of weaker members on their team to invest maximum effort to build their expertise. In addition, team members must believe that their own con- tributions to the team effort are being constantly and fairly evaluated along with the performance of the entire team. Finally, team competitiveness must be focused on opposing organizations that are struggling for the same customer base, not on teams in their own organization.

4.2 Personal productivity

4.3 Automatisation

5. Building productivity mobile application

5.1 Abstractions and games

what part does mobile application could take in the scheme of performance

5.1.1 Concept of time limitation

Describe the idea of resource limit as the mobile screen.

5.2 Game mechanics in use

Describe the mechanics of the app and then check game mechanics.

5.3 Feedback

Feedback is important, as a part of a flow.

5.4 Prototype

Screens of the application, presentation

5.5 Development plan

Next steps: how to fill the scheme

Conclusion

Bibliography

[1] Dan Brandon. *Project Management for Modern Information Systems*. IRM Press, 2006.

[2] Carl Chatfield. *A short course in project management*, *http://office.microsoft.com/en-us/project-help/a-short-course-in-project-management-HA010235482.aspx*. Microsoft.

[3] Sebastian Nokes. *The Definitive Guide to Project Management*. ISBN 978-0-273-71097-4. Financial Times, Prentice Hall, 2007.

[4] Agile manifesto, http://www.agilemanifesto.org/.