

<http://tadsessions.espace.com.eg>

Agile Estimating and Planning

Part I: The problem and the goal

By: Ali Maher



- Remember that the authors of the Agile manifesto wrote that they 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

Where is the problem?

Difference between building a house
and building a software



	House	Software
Raw materials	High cost	No raw materials
Analysis and Design Cost	Relatively very low	Relatively equivalent to the implementation
Requirement Changes	Very limited	frequent and major
Uncertainty	Limited	Very high
Tasks ordering	Almost visible and fixed	Variable and project dependent
Multitasking	Limited	Opened
Measuring and testing	Clear and fixed	Project dependent
Tasks dependability	Tasks are independent	Tasks are dependent
Nature	Physical output	We are manufacturing logic!
Further Development	Slow	Rapid

- Estimating and Planning are critical, Yet are difficult and error prone.
- Estimates given early in a project are far less accurate than those given later.
- The team that does no planning cannot answer the most basic questions, such as “When will you be done?”
- The team that overplans deludes themselves into thinking that any plan can be “right.” Their plan may be more than thorough, but that does not necessarily mean it will be more accurate or useful.

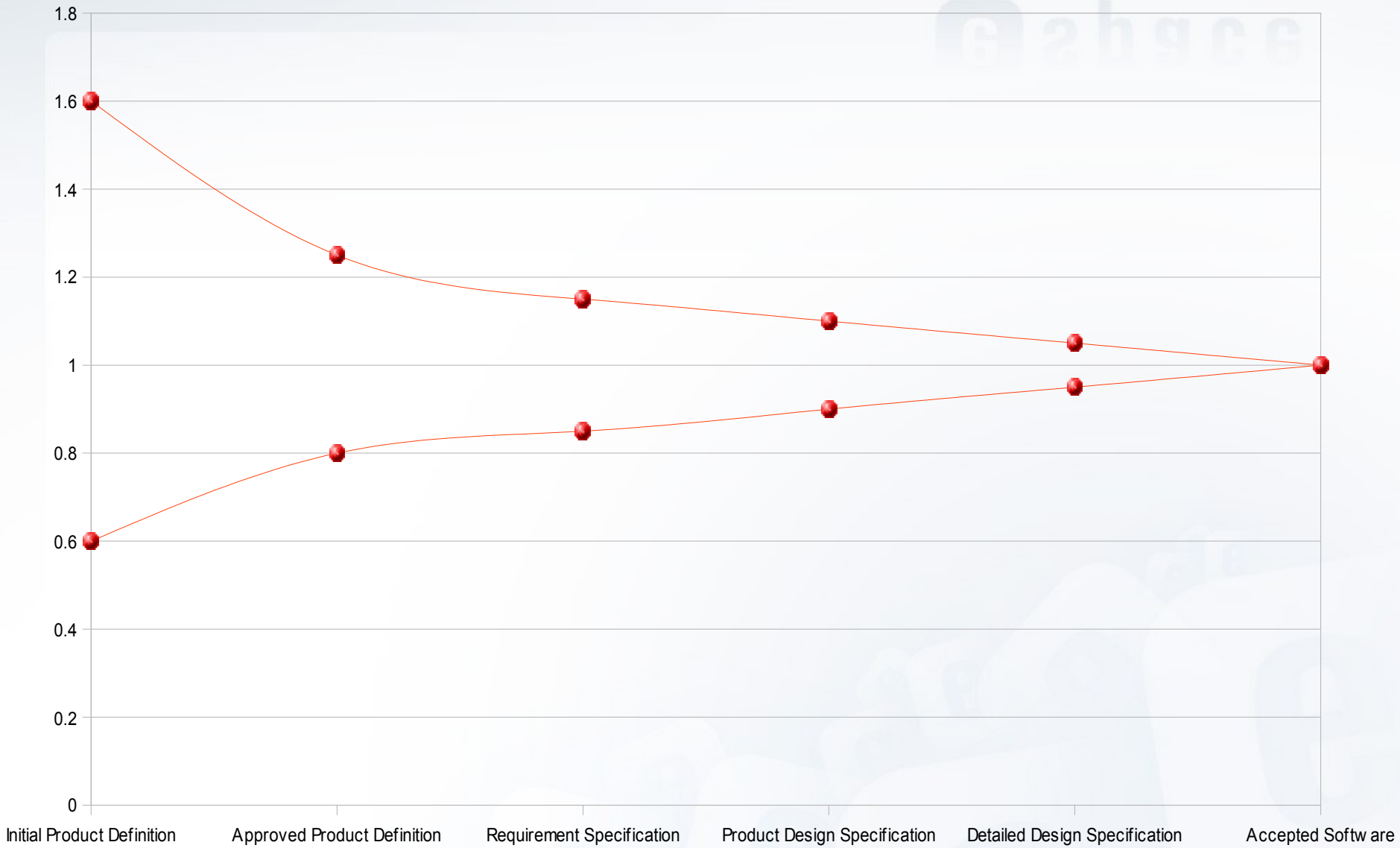
Should software estimation be called “Engineering” [Theoretical limits for estimation]



- Software engineering community is currently divided into:
 - “process” camp: who believe that quality software can be developed on time if a particular software process or programming technology is used.
 - “Problem solving” camp: who believe the programming is fundamentally a process of solving problems.
- “The creation of genuinely new software has far more in common with developing a new theory of physics than it does with producing cars or watches on an assembly line” [Bollinger]
- Using Algorithmic Complexity one can prove that *objective* estimation of software complexity is incorrect.

- Planning is everything. Plans are nothing.
- Planning is an ongoing iterative process.
- Cone of uncertainty

Planning vs Plan - [Cone of uncertainty]



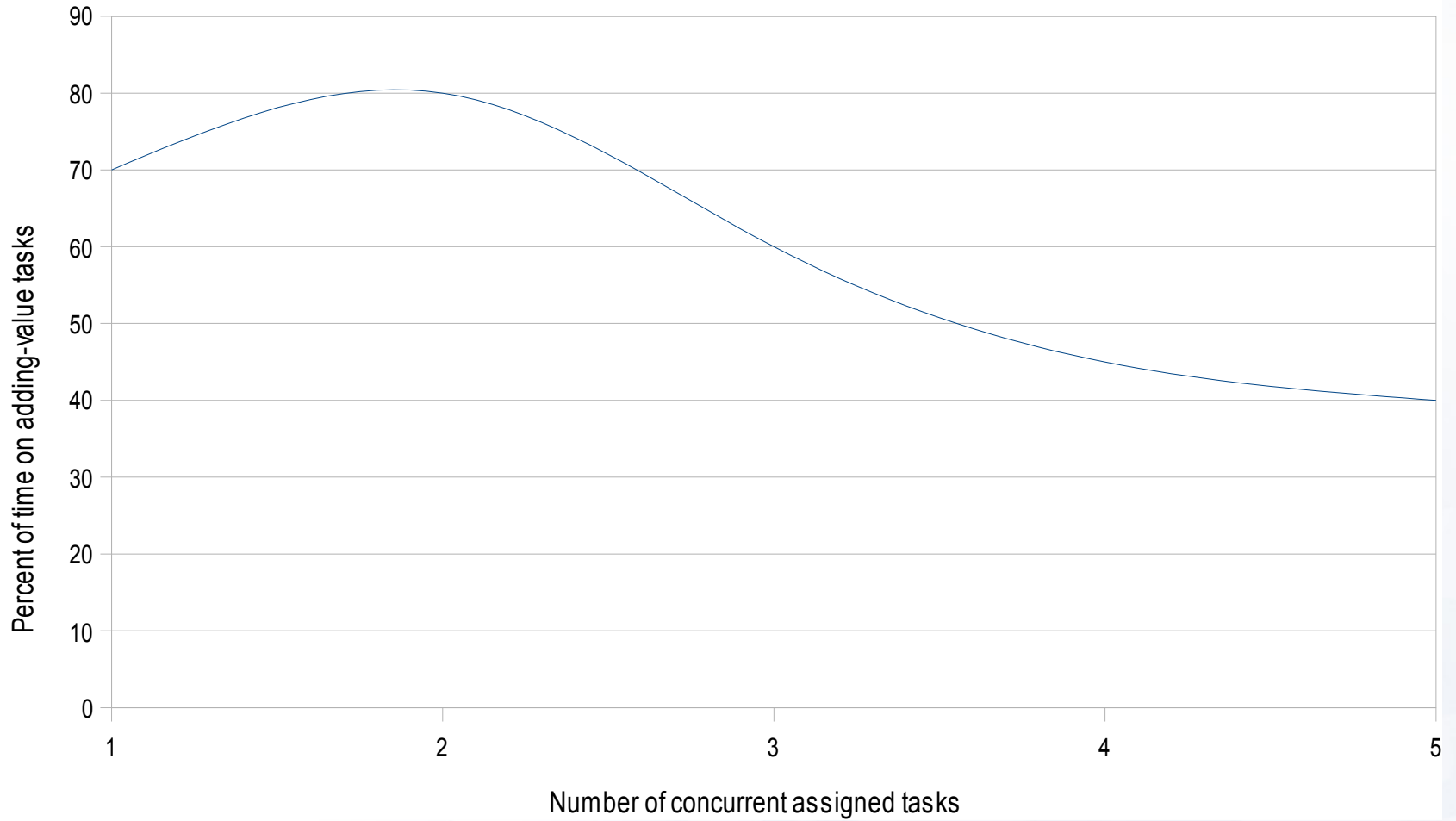
- Planning is an attempt to find an optimal solution to the overall product development question: What should we build?
- To answer this question, the team considers *features*, *resources* and *schedules*.
- The question cannot be answered all at once
- A good planning process supports this by:
 - Reducing risk
 - Reducing uncertainty
 - Supporting better decision making
 - Establishing trust
 - Conveying information

- Nearly two-thirds of projects are significantly overrun their cost estimates (Lederer and Prasad 1992)
- Sixty-four percent of the features included in products are rarely or never used (Johnson 2002)
- The average project exceeds its schedule by 100% (Standish 2001)

- Planning is by activity rather than feature
- Lateness is passed down the schedule
- Activities are not independent
- Features are not developed by priority
- We ignore uncertainty
 - The best way of dealing with uncertainty is to iterate
- Multitasking causes further delays

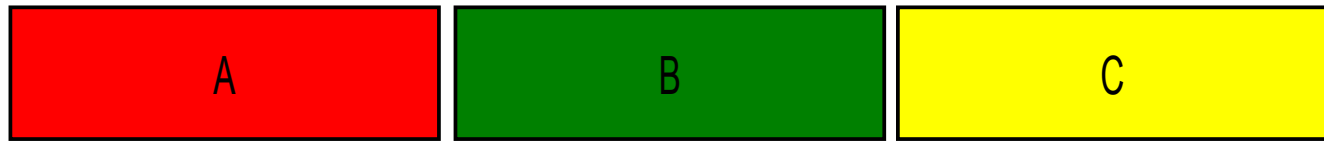
- Multitasking has negative effects on productivity.
- Rapid context switching has negative effect on quality and productivity.
- Loading every one to 100% of capacity has the same effect as loading a highway to 100% of capacity: No one can make any progress.
- Increasing the number of concurrent assigned tasks rapidly decreases the time spent on *value-adding tasks*.
- The following curves illustrate these facts.

Effect of multitasking on productivity

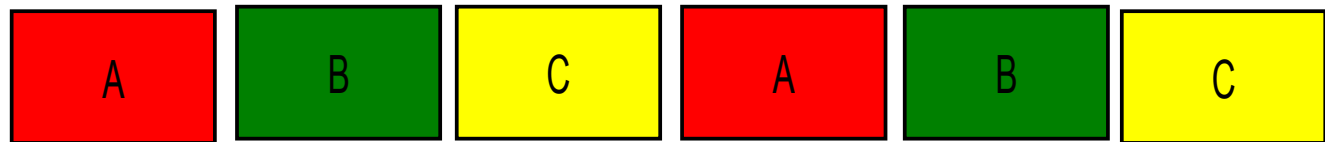


Multitasking leaves work in process longer

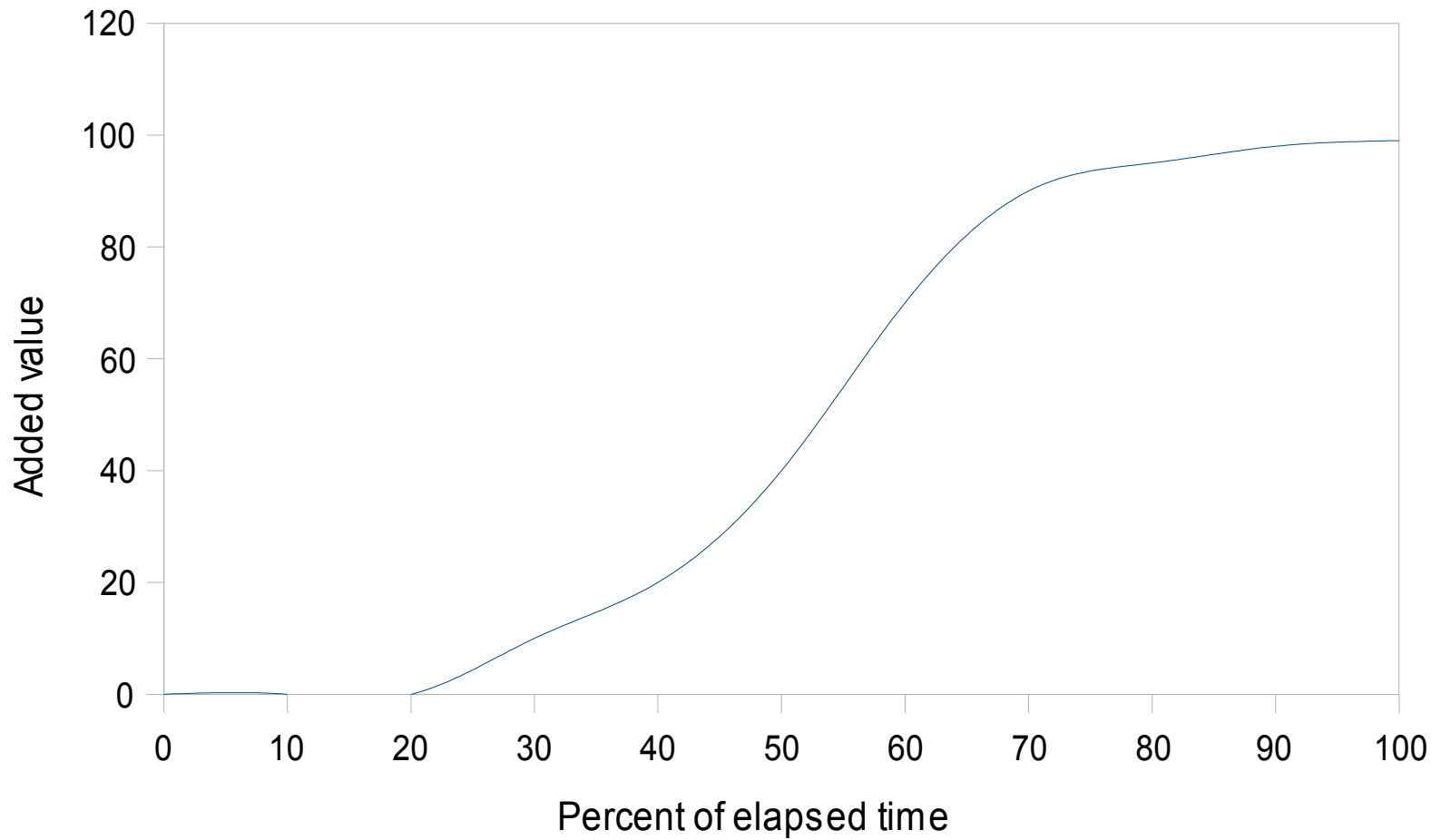
Focus



Multitasking



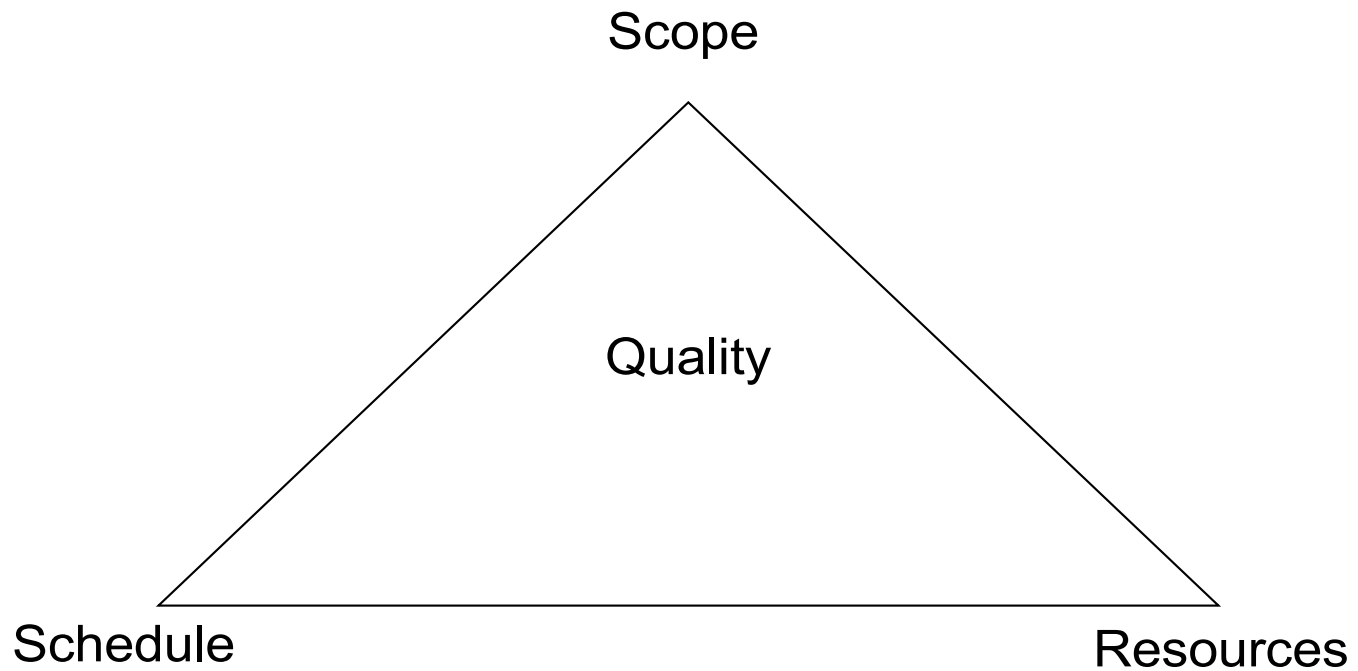
Adding value time for task



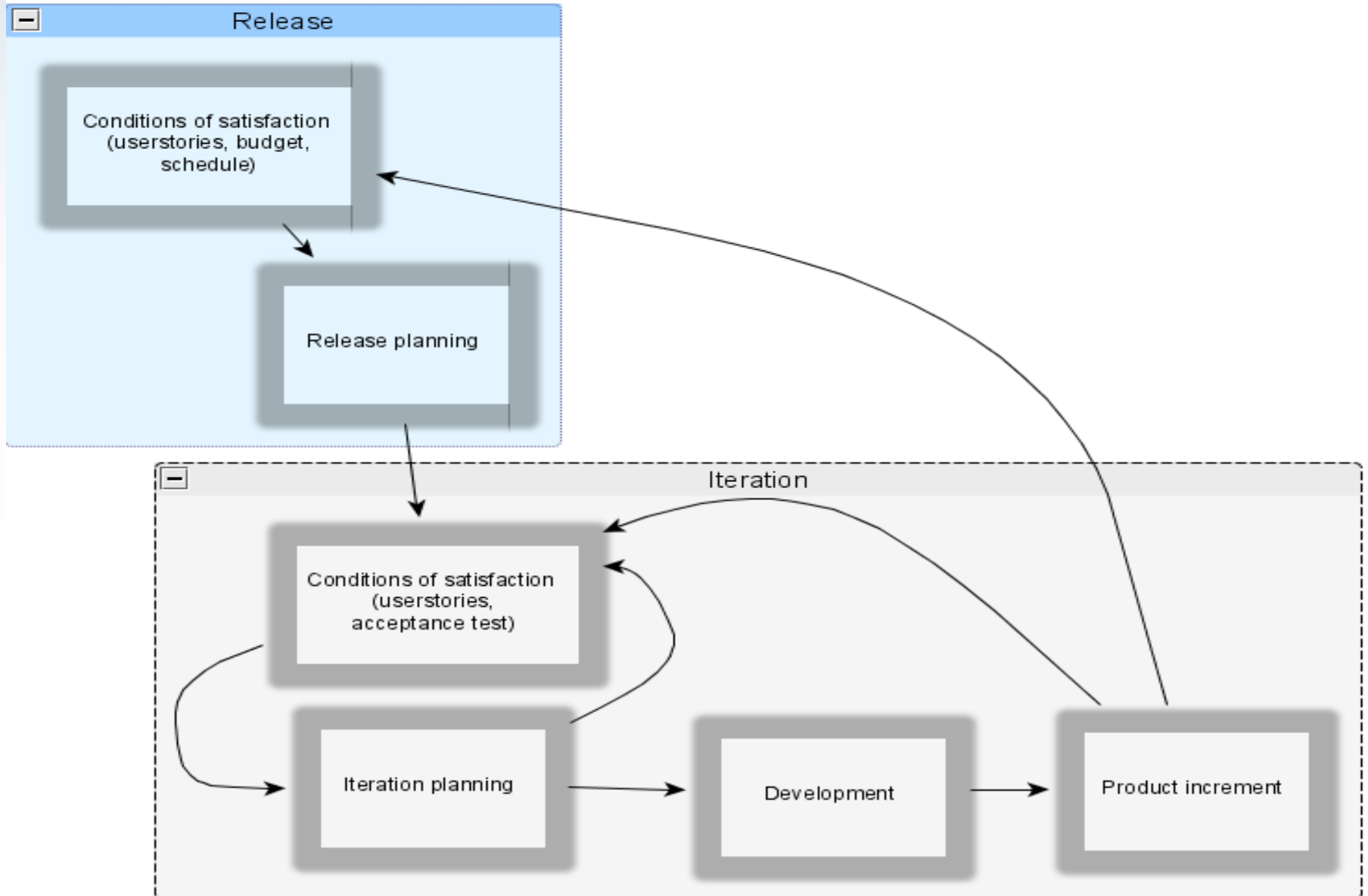
- Is focused more on the planning than on the plan
- Encourages change
- Results in plans that are easily changes
- Is spread throughout the project

- We should not view a project solely as the execution of a series of steps. Instead, it is important that we view a project as rapidly and reliably generating a flow of useful new capabilities and new knowledge.
- Illustrative example: 10-km race vs sixty min race.
- Planning becomes a process of setting and revising goals that lead to a longer-term objectives.

- Day: day to-do list.
- Iteration: next high-priority set of features.
- Release: Scope, schedule, resources of a project.
- Product: owner's looking ahead the immediate release.
- Portfolio: selection of the products that will best implement the vision.
- Strategy: establishing the vision.



Conditions of satisfaction



- Agile teams separate estimates of **size** from estimates of **duration**.
- Schedule is derived from size estimates.
- Two methodologies for size estimates:
 - Story points
 - Ideal days

- Story points: are a unit of measure for expressing the overall size of a user story.
- Story points are relative.
- **Velocity**: is a measure of a team's rate of progress. It is calculated by summing the number of story points assigned to each user story that the team completed during the iteration. Its unit points/iteration.
- Velocity corrects estimation errors.
- It separates the estimation of effort from the estimation of duration.

- **Ideal time** is totally different of **elapsed time**.
- Ideal time is the amount of time that something takes when stripped of all peripheral activities.
- Elapsed time is the amount of time that passes on a clock (or perhaps a calendar).

- Considerations favoring Story points:
 - Story points help drive cross-functional behavior.
 - Story-point estimates do not decay.
 - Story points are pure measure of size.
 - Estimating in story points typically is faster.
 - My ideal days are not your ideal days.
- Considerations favoring ideal days:
 - Ideal days are easier to explain outside the team.
 - Ideal days are easier to explain at first.

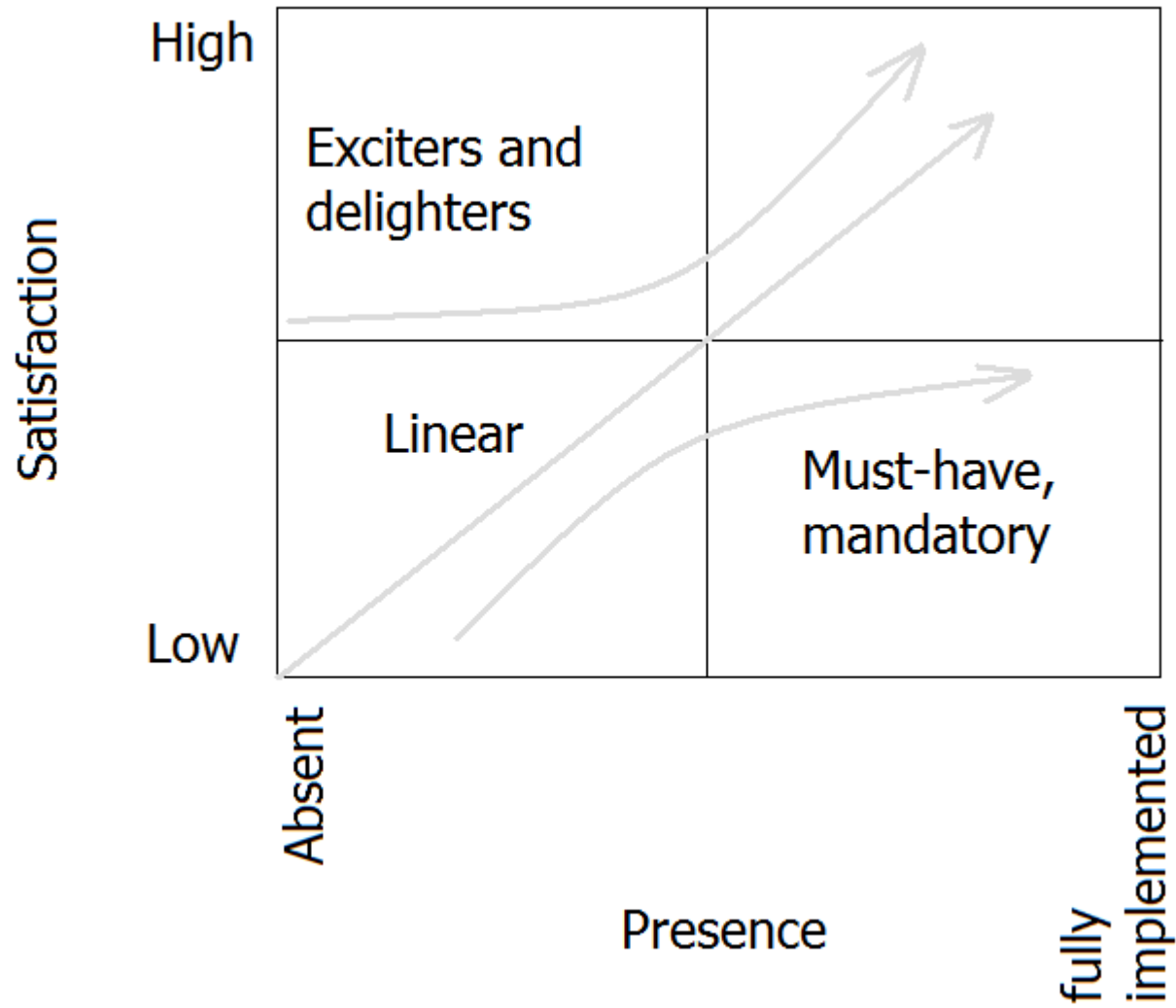
- Theme: is a set of user stories or features that defines a discrete set of user- or customer-valued functionality.
- After each iteration we acquire two type of new knowledge:
 - Knowledge about the product. (What will be developed)
 - Knowledge about the project. (How it will be developed)

Prioritizing Themes (or Stories)

High Risk Low Value	High Risk High Value
Low Risk Low Value	Low Risk High Value

- Separate features into three categories:
 - Threshold, or must-have, features
 - Linear features
 - Exciters and delighters
- See Kano diagram in the next slide.
- Keep in mind that features tend to migrate down the Kano diagram over time. Exciters becomes linear and linear becomes threshold.

Kano diagram



- The nature of software projects is different of the nature of many other engineering projects.
- planning is every thing. Plans are nothing.
- Planning is an attempt to find an optimal solution to the question: What should we build?
- Multitasking, overloading, interruptions and context-switching negatively affects the productivity.
- Agile planning is flexible and spread though out the project.
- Agile teams has at least three levels of planning: Release, iteration and day.

- Release planning have three factors beside the quality: scope, cost, schedule.
- Agile teams use user stories as a base for estimation and planning.
- Agile teams separate estimates of size from estimates of time using story points or ideal days
- Velocity is a measure of a team's rate of progress.
- Features are grouped into themes. Each theme adds a new value to the customer's business.
- Themes should be prioritized based on their risk, value and desirability.

Thank you



Thank You