Agile Monitoring and Control

Burndown Chart

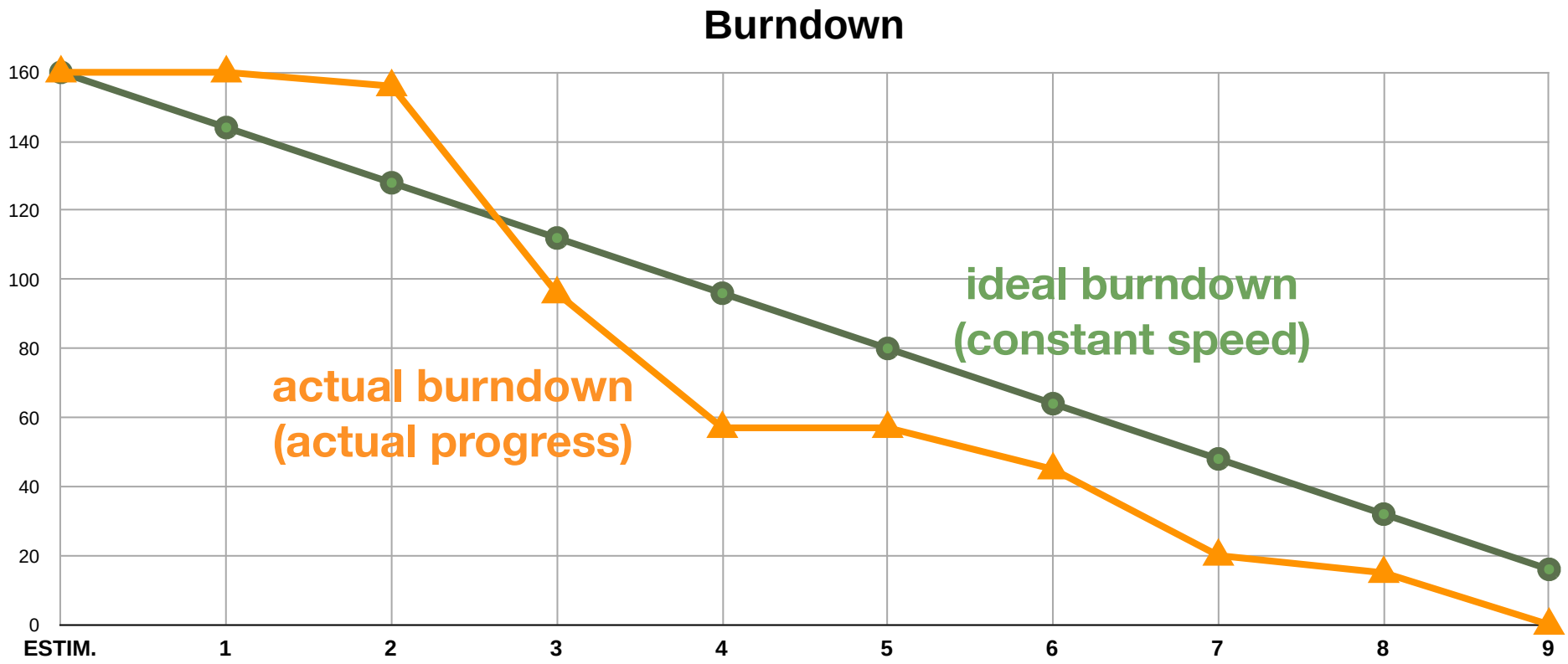
Agile Estimation Practices

- Simplifying a bit, SCRUM and other agile methodologies:
  - Move away from the effort as a perfect measure
  - Structure development in sprints of fixed duration
  - Focus on remaining work, rather than work to be performed
- Examples:
  - Extreme Programming estimates using “ideal programming hours”
  - SCRUM estimates using “points” (an abstract measure, which deliberately moves away from traditional effort estimations)

Agile Estimation Practices

- The estimation process assigns a number of ideal hours or points to each feature to be implemented
- Features are then assigned to a sprint, determining the number of points to be **burned**
- During the sprint points are “burned” as features are delivered.
- One management goal is measuring the **speed** at which features are delivered and ensuring the estimation process keeps the speed constant between sprints

Agile Progress Monitoring



Agile Earned Value Analysis

Agile EVM

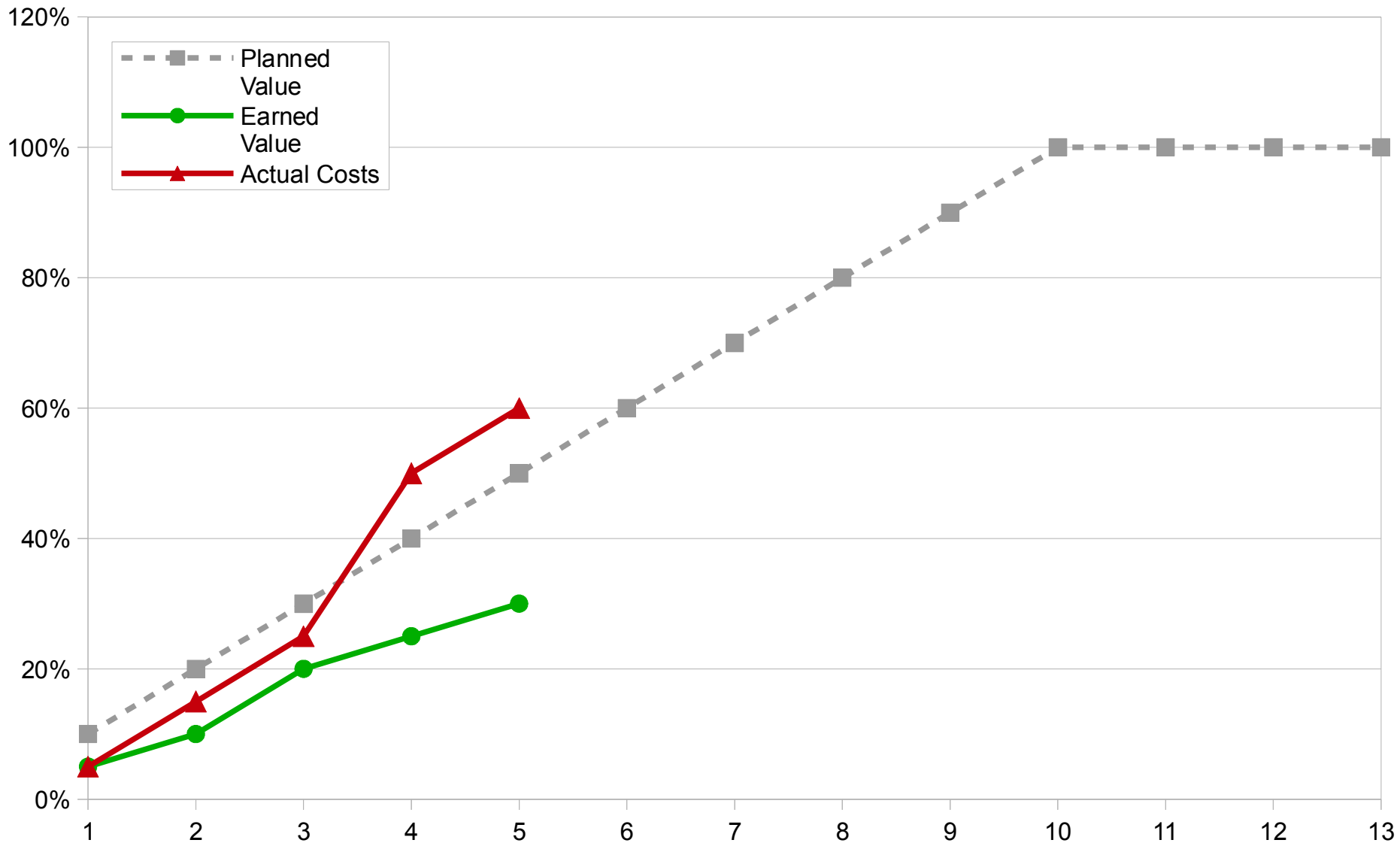
- A mix of Agile and Earned Value Management
- Motivations (*)
  - Substantial:
    * Are we making enough progress, but only by blowing the budget with overtime?
    * If we are running late, was it due to problems in the project, or was it due to other projects “borrowing” our people?
  - Substantial/Formal:
    * Certain projects require the application of EVM (e.g. defense projects in the US > 20M$)
- At least two different methodologies have been proposed (we will look at one)

(*) Source: http://www.agilekiwi.com/EarnedValueForAgileProjects.pdf

Agile Earned Value: an Approach

- The following method proposed by: John Rusk - Earned Value for Agile Development http://www.agilekiwi.com/EarnedValueForAgileProjects.pdf
- It works with SCRUM
- Simple and informal:
  - No special terms; it just assigns colors to the different curves (simpler)
  - It uses percentages rather than money
- Three lines:
  - The grey line (= **planned value**) shows the progress that we expect to make, as a percentage of the total project. It starts at zero and slopes up to 100% at the end of the project.
  - The green line (= **earned value**) shows how much of the product we have built, as a percentage of the total product size.
  - The red line (= **actual costs**) shows the cost we have incurred so far, **as a percentage of the total project budget**.

Example


120%
100%
80%
60%
40%
20%
0%
1
2
3
4
5
6
7
8
9
10
11
12
13
Planned Value
Earned Value
Actual Costs

Understanding the Plot

- In a perfect world, the three lines are perfectly aligned
- The **green** line measures the schedule:
  - If **green** above the **grey** line: ahead of schedule
  - If **green** below the **grey** line: behind schedule
- The **red** line measures efficiency:
  - If **red** above the **green** line: spending the budget faster than we're building the software. Cost overrun!
  - If **red** below the **green** line: building the software faster than we are spending the money. Under budget!

Computing the Lines

- **Planned Value** is computed from the backlog (sum of story points). Velocity is assumed constant and the result is a straight line (simplification, but good enough)
- **Actual Costs** are computed from **costs of hours worked as a percentage of the total project budget**.
- **Earned value** is the **sum of completed story points**. No points for partial completion of a task. This practice is consistent with that normally used on agile burn charts and is referred to as the **0/100 rule in EVM**.
  Only working software features earn value: you only score points when it is designed, built and tested.

Some Remarks

- The chart works best when it covers a period of time that is big enough to feel like the “big picture” (e.g., 3 to 6 months)
- Different levels of granularity are possible (to show daily progress, for instance)
- The technique requires scope to be available up-front (the backlog has to be known before starting the project)
- Delivery of value must be linear (e.g. testing is not at the very end)... otherwise the 0-100% rule does not allow one to measure progress.

Managing Changes

- Two types of changes:
  - Nuances in requirements
    - * Approach: do not account them (they are “sunk” in the known requirements)
  - Actual scope changes (e.g.: new ideas)
    - * Approach: Measure scope changes and revise EVM plot