

# Principles of Software Design Management, Planning

Robert Lukočka

lukotka@dcs.fmph.uniba.sk

www.dcs.fmph.uniba.sk/~lukotka

M-255

# Project management activities

- Project initiation
- Contract management
- For each management item: time, work, resources (people, money, ... )
  - planning
  - organization
  - controlling
  - monitoring
  - evaluation
- Decision making
- Information management

# Planning

- Defining planed items
- Estimation (work, dependencies)
- Resource allocation
- Scheduling
  - Critical path analysis
- Updating/refining the plan

# Planning

What plans do we need.

- There are various scopes: project plan, release cycle plan, iteration plan
  - This does not imply that you should have detailed long term plan.
  - There are uncertainties involved and you should acknowledge them, it makes little sense to discuss details when huge error bars are involved
  - Often it is right to prefer adaptive planning.
- There are various activities: e.g. Design, Construction, Testing, Contract management; Risk Management; Change management;.

# Organization

- Project team structure.
- Set up environment and processes.
- To obtain, arrange and release resources.
- Attach resources to roles.

# Controlling

- Check and regulate managed objects to achieve specified goal.
- Controlling used materials and resource.
- ...

# Monitoring and evaluation

- Measure relevant metrics, get feedback
- Keep measurement logs
- Report defects
- Use the results to refine plans, improve processes, improve allocation of resources, . . .

# Management

During the rest of the presentation we cover several selected topics of interest:

- Brook's law
- Cross-functional teams
- Issue tracking, product backlog, kanban boards
- Scrum
- Estimations



# Planning: Brook's Law

*"Adding engineers to a late software product makes it later."*

Why?

- It takes some time (and possibly resources) for the people added to a project to become productive.
- More people, more communication overhead.
- Some works are not easily divisible. (It takes one woman nine months to make one baby, nine women can't make a baby in one month. [3])

# Planning: Brook's Law

What can we do about this? How to mitigate the effect

- Correcting the schedule :).
- Add highly skilled programmers and specialists (they become productive faster and require less communication to do so).
- Helping with side tasks (e.g. quality assurance, documentation)
- Adding people early enough (the law applies to projects that are already late).

# Organizational structure

There are two very distinct approaches to the structure of teams

- Traditional team structure - team consist of specialist specialized in the same area (e.g. User Experience/UI Design, Development, Quality, Database, Security, . . . )
- Cross-functional teams - members of each team have competence in various areas of software development, covering (almost) all aspects necessary for the project (this may include even people with background in e.g. marketing).

# Cross-functional vs Traditional

## Traditional teams:

- Allow deep specialization within the teams area of expertise
- Easier substitutability of the team members (e.g. when somebody gets sick, or goes for a vacation)

## Cross-functional teams:

- Team members get competence to perform tasks in neighboring areas.
- This improves decision-making within the team as the members have better understanding of the product as whole.

# Cross-functional vs Traditional

The application structure tends to copy the organizational structure:

- Communication within a team is way easier than inter-team communication.
- Thus if a team member can solve an issue within the team, he often does that, despite the fact that the change creates new dependency with other component/service.
- This may erode the architecture over time.

# Cross-functional vs Traditional

Consider an issue when database is too slow to handle one kind of requests.

- In traditional team structure, the likely solution is that something is improved on the database level. As the team has various experts with deep knowledge in the area, traditional team may be able to do this even if it is hard while cross-functional team might struggle.
- In cross-functional team it is quite likely, that the issue will be dealt with outside the database (do we even need to send that many requests).

# Cross-functional vs Traditional

Responsibility for product.

- It makes a big difference if the responsibility for a product is shared between 6 and 100 people.
- It is hard to account which specific actions have what impact. In these circumstances people tend to attribute the positive effects to the actions of their team and negative effects to the action of other teams.
- It may lead to an uncooperative attitude and loss of motivation.

# Issue tracking

Issue tracking system is a system that helps to maintain and manage a list of issues and foster the collaboration to solve these issues.

- often contains various additional tools for project management (resource allocation, priority management, oversight ...)

Issue:

- Bug
- Feature
- Task
- Missing documentation
- ...

These systems started as bug trackers. People later started successfully use bug trackers to handle non-defect issues.



# Issue tracking

Issue contains:

- Identifier
- Type (bug, feature)
- Status (open, in progress, resolved, closed, reopened)
- Priority
- Assigned to
- Further relevant information obtained while handling the issue (bug description, reference to the requirements documentation, commit number)
- Communication associated with the request
- ...

# Issue tracking

Look how an issue tracking system may look like:

- [Redmine issues of redmine](#)

How an issue may look like:

- [Issue example](#)

# Issue tracking alternatives

Some other popular ways to assign tasks to team members:

- Sprint Backlog ([Example](#))
- Kanban boards, [Example 1](#), [Example 2](#)

In both cases the approaches assume team self-organization., i.e. there is no authority to decide who pick which task, everybody tries to pick the task so that the objectives of the iteration are attained.

- Also assumes cross-functional teams of reasonable sizes.

# Scrum [13]

- Scrum is a management framework for incremental product development using one or more cross-functional, self-organizing teams of about seven people each.
- It provides a structure of roles, meetings, rules, and artifacts. Teams are responsible for creating and adapting their processes within this framework.
- Scrum uses fixed-length iterations, called Sprints. Sprints are no more than 30 days long, preferably shorter. Scrum teams try to develop a potentially releasable (properly tested) product increment every Sprint.

# Scrum [13]

- Scrum - Iterative-Incremental development, an iteration is called **sprint**. Sprint has fixed length, at most 30 days,
- Team has 3-9 members, it is cross-functional, self-organizing, has autonomy on how to develop.
- Specialized roles: Product Owner a Scrum Master.
- Scrum defines a plan of meetings.
- Scrum defines three artefacts: product backlog, sprint backlog, increment. These artefacts do not replace the documentation but exist on top of it. It is team responsibility to decide what should be documented.
- At the end of the sprint we should have a deployable product increment.

# Scrum - Product Owner [13]

- Responsible for maximizing the return on investment of the development effort.
- Responsible for product vision.
- Constantly re-prioritizes the Product Backlog, adjusting any long-term expectations such as release plans.
- Final arbiter of requirements questions.
- Decides whether to release.
- Decides whether to continue development.
- Considers stakeholder interests.
- May work as a team member.

# Scrum - Scrum Master [13]

- Works with the organization to make Scrum possible.
- Ensures Scrum is understood and enacted.
- Creates an environment conducive to team self-organization.
- Shields the team from external interference and distractions to keep it in group flow.
- Promotes improved engineering practices.
- Has no management authority over the team.
- Helps resolve impediments.

# Scrum - Meetings [13]

- Sprint Planning
- Daily Scrum
- Sprint Review
- Sprint Retrospective
- Backlog Refinement (Optional)
- Sprint Cancellation (Optional)



# Scrum - Meetings [13]

## Sprint Planning

- Product Owner required on this meeting.
- Team is responsible to make estimations for Product Backlog tasks.
- Product Owner prioritizes the tasks (taking into account the estimation).
- Team decides to which tasks it commits during this sprint, the selected tasks are moved into Sprint Backlog.
  - Takes into account PO's priorities.
  - Takes into account current project state and technical debt.
  - PO cannot affect the the amount of work the team decides to do within the sprint.

# Scrum - Meetings [13]

## Sprint Planning

- If the top of the Product Backlog has not been refined, a portion of the Planning meeting might be spent doing this.
  - If necessary, Backlog Refinement Meeting may be held before sprint planning.
- Toward the end of the Sprint Planning Meeting, the team determines how it will accomplish the work.
- Duration: max. 8 hours for 30 day sprint.

# Scrum - Meetings [13]

## Daily scrum (a.k.a. stand-up)

- Inspecting progress, resolving issues.
- Stand-up meeting, max. 15 min. Standing up is a popular way to keep it short (which does not always work).

# Scrum - Meetings [13]

## Sprint Review Meeting

- Product Owner required on this meeting.
- Demonstrate the working product.
- Product Owner decides what is done (it is up to PO to decide what is important, even unfinished documentation may cause an item to be declared not done). Unfinished tasks return to the Product Backlog.
- Various stakeholders can attend the meeting (e.g. future users of the product). It is an ability to find out how to adapt the product.
- Scrum Master helps PO and other stakeholders to convert the feedback into new Product Backlog Items. If new items appear fast enough, some Product Backlog Items are never done (which is a good thing).

# Scrum - Meetings [13]

## Sprint retrospective

- Scrum Master is required on this meeting.
- At this meeting, the team reflects on its own process. They inspect their behavior and take action to adapt it for future Sprints.
- There are various obstacles that impede the reflection.
  - Presence of people who conduct performance appraisals.
  - Psychological aspects.
- Scrum Masters should use a variety of techniques to facilitate retrospectives.
- The fact that the Scrum Master does not contribute as a team member allows him to see how the team work from a different perspective.

# Scrum - Scrum Master role

Scrum Master is fixed within a sprint but may change between sprints.

- In some scrum approaches Scrum Master is a rotating position (original approach?).
- Other scrum approaches have devoted Scrum Master (currently more prevalent approach?).

There are differences, e.g.:

- Devoted Scrum Master probably knows more tricks to improve discussion/feedback/...
- A rotating Scrum Master has better chance to find technological improvements in the area he is familiar with.

# Scrum - Artefacts [13]

## Product Backlog:

- Force-ranked (prioritized) list of desired functionality.
- Visible to all stakeholders.
- Any stakeholder (including the Team) can add items.
- Constantly re-prioritized by the Product Owner.
- Constantly refined by the Scrum Team.
- Items at top should be smaller (e.g. smaller than 1/4 of a Sprint) than items at bottom.

# Scrum - Artefacts [13]

## Product Backlog Item:

- Describes the what (more than the how) of a customer-centric feature.
- Often written in User Story form.
- Has a product-wide definition of done to prevent technical debt.
- May have item-specific acceptance criteria.
- Effort is estimated by the Development Team, ideally in relative units.



# Scrum - Artefacts [13]

## Sprint Backlog:

- Consists of selected PBIs negotiated between the team and the Product Owner during the Sprint Planning Meeting.
- No changes are made during the Sprint that would endanger the Sprint Goal.
- Initial tasks are identified by the team during Sprint Planning Meeting.
- Team will discover additional tasks needed to meet the Sprint Goal during Sprint execution.
- Visible to the team.
- Referenced during the Daily Scrum Meeting.

# Scrum - Artefacts [13]

Increment:

- The product capabilities completed during the Sprints.
- Brought to a usable, releasable state by the end of each Sprint.
- Released as often as the Product Owner wishes.
- Inspected during every Sprint Review Meeting.

# Scrum - Artefacts (optional)

Further optional artefacts:

- Sprint task (optional)
- Sprint burndown chart (optional)
- Product/release burndown chart (optional)

# Scrum values

- *Commitment*: Team members individually commit to achieving their team goals, each and every sprint.
- *Courage*: Team members know they have the courage to work through conflict and challenges together so that they can do the right thing.
- *Focus*: Team members focus exclusively on their team goals and the sprint backlog; there should be no work done other than through their backlog.
- *Openness*: Team members and their stakeholders agree to be transparent about their work and any challenges they face.
- *Respect*: Team members respect each other to be technically capable and to work with good intent.

# Scrum - some issues

- The priorities of the tasks are very important to get right. May be too important to be decided by one person: Product Owner. On the other hand, PO may stand as a representative of other process that decides what should be built.
- Scrum does not specify good technical practices, it is left to the team.
  - Good technical practices, especially the ability to cope with changes, are necessary for scrum to succeed.
  - Ignoring technical quality causes (after a short period of increased speed) the slow-down or even stoppage of the development (see the following: [technical debt illustration](#), [flacid scrum](#)).
- Scrum is hard to scale (see e.g. Scrum of scrums).

# Scrum stability of PBI

- Scrum considers important, that the goal of the sprint is stable and the team works 100% to achieve it.
- New stuff can be added into sprint backlog only if it does not endanger the sprint goal.
- If something becomes so important that it needs to be done right away, and is significantly large to endanger the sprint goal, the only way to do it before next sprint is sprint cancellation (thus the Product Owner has to choose wisely if it is that important).

# Sprint cancellation

Typical reasons for sprint cancellation:

- A better technical solution is found that makes the current Sprint's activity throwaway work.
- A major technology change occurs.
- Market forces render the work obsolete.
- Fundamental and urgent external changes invalidate the Sprint Goal.
- Urgent bug fix or feature development requests cannot wait until the normal completion of the Sprint.

After Scrum cancellation all Sprint Backlog items are moved into Product Backlog and new sprint starts with Sprint Planning.

# Estimations

Good estimations are necessary:

- planning (time, resources)
- decision-making
- monitoring
- evaluation



# Estimations - expected value vs deadline

There is a common problem in interpreting the estimations. should the estimation give an expected value or a deadline?

- What is a deadline. To be 100% sure is not realistic nor practical.
  - E.g. consider the following definition of the deadline: time when there is a 95% chance that we are finished.
- Expected value and deadline as defined here can be very different, especially if significant uncertainties are involved.
- If we want to compose estimations, the most meaningful single number parameter is the expected value.
- Deadlines cannot be easily combined by adding them when large uncertainties are involved.

You should distinguish expected values and “deadlines”.

# Estimations - expected value vs deadline

Despite this fact management, customers, executives want deadlines and will often add them.

- the desire for deadlines is not unwarranted (you need it e.g. for contracts).
- If you give deadlines for partial job, the sum of time is way too big (summing is incorrect, but this is what is usually done with the numbers).
- Thus you often need to give some number between the expected value and the “deadline” value to make everybody happy (even considering all other aspects).

A sensible approach is at least two numbers expected value and variance: other parameter that describes the distribution and calculate the deadlines from them.

# Cone of uncertainty

- It is not possible to provide good estimates in the early phases of the development, there are too many unknowns.
- On the other hand, if you contract is fixed price, cost, time, you need it early.
- Good project management can focus on reducing the unknowns, and thus improving the precision of the available estimations faster.
- After we deal with several most significant unknowns, law of big numbers should work in our favor.

# Cone of uncertainty

I can see how I'd do it if I were rewriting that whole controller from scratch, but that would take days ... is there an elegant hack where I can change the inputs to this function in such a way that I don't have to rewrite its code? ... what if I monkeypatch it at the class level? ... wait, maybe there's an API call that almost does what I want, then I can tweak the results - hang on, what if I outsource it via an asynchronous call to the external OS? In that case I can confidently estimate that this will require less than two hours of typing. However, working out what to type is going to take me/us anywhere from one hour to several days. Sorry. [7]

# Estimations

Software estimates are always wrong, because the tasks being estimated are always, to some extent, terra incognita, new and unknown. However, sometimes the errors are in your favor; an obscure API, a third-party library, or an elegant hack condenses what you expected to be a week's worth of work into a single day or less. [7]

The cases where we are way faster than expected are more rare, but present a problem to—we may not be able to use the resources that became available efficiently.

# We are quite bad at estimating stuff, some jokes

## Hofstadter's Law:

- It always takes longer than you expect, even when you take into account Hofstadter's Law. :)

## 90-90 Rule:

- The first 90 percent of the code accounts for the first 90 percent of the development time. The remaining 10 percent of the code accounts for the other 90 percent of the development time. :)

# Estimations

Check this:

- [How to respond when you are asked for an estimate](#)

Besides all this, there is another issue: “How much work does it require?” vs “When will it be done?”.

- It takes two weeks, but as you have process high priority requests first, it might take four weeks.

# Estimation bias

Even if we understand the difference between expected value and deadline and even if we are aware of the uncertainties involved, there are still many reasons for bias.

- positive bias - people tend to think they are good :)
- MCE pressure
  - They want to hear a low number and they are right next to you. It is easy to “get rid of them” by satisfying them.
  - The time depends on who does the job. Does a high estimate (higher than your peer's) mean, that you are just slower at the job or you see something that you see something others do not see.
- It is hard to be intellectually honest about own capabilities.
- Random unexpected events increase the required time way more often then they decrease it.
  - People are generally very bad at accounting for unlikely scenarios.



# Estimation approaches

- Expert estimation
- Formal estimation model
- Combination-based estimation

Expert estimation is the dominant approach. Formal estimation requires detailed specification and good data measured in similar past projects.

# Estimations via decomposition

A natural approach is to decompose the problem into smaller tasks. This approach is useful but has some limitations.

Functionality also not the right key. Consider a concrete example. Suppose you're building an app that logs in to a web service. Don't have individual server-side estimates for "user can create account," "account email address can be confirmed," "user can log in," "user can sign out," and "user can reset password." Have a single "user authentication" task, and estimate that. [7]

- The parts often intersect.
- There is often a correlation between how long the parts will take.
- A lot of work, it may lead to estimation fatigue (A state when you do not really think about the estimations, but you want to end the meeting :))

# Estimations - manhours

The most common measure of required effort is manhours. This is not an universal measure and one needs to be careful when using it

- Brook's law is an example of situation when applying man hours makes no sense at all.
- Not everybody works at the same pace.
- The is a proxy parameter (eg. user story points, function points) is commonly introduced to the estimation process.

*"Let's run that trail. It'll take 30 minutes."* [11]

Also, helps with MCE pressure to some extent.

- Eventually, we can translate the proxy parameter to manhours at appropriate level (e.g. for whole team).

# Estimations - T-shirt method

XS, S, M, L, XL, XXL

- As teams learn to work together, the team members refine their idea of what a given T-shirt means.
- Note that the estimate precision is severely limited by available T-shirts. This shows, that the estimate is not accurate enough anyway to reasonably consider more granularity.
- T-shirt estimates - what can they easily state about our backlog:
  - XL/XXL on the top of the backlog → we need to break stuff into smaller tasks.
  - SSSSS → we were breaking too much, this is inefficient, now we have to put effort to manage a lot of small tasks.
- If you have a roughly even mix of S, M, and L, you've probably structured things so that you'll have pretty good - well, least bad - estimates. [7]

# Story points

- 1, 2, 3, 5, 13, 40, 100, infinity, unknown
- Story points make apparent that the difference between S and M is much smaller than between M and L.

# Planing Poker [10]

- Planning poker (or scrum poker) is a consensus-based, gamified technique for estimating, mostly used to estimate effort or relative size of development goals in software development.
- The rules mitigate various psychological issues that make it hard for a group to obtain a honest estimate.

# Planing Poker [10]

- Each player has its own set of cards, e.g.: 0, 0.5, 1, 2, 3, 5, 8, 13, 20, 40, 100, ?,  $\infty$ , coffee cup.
- A Moderator, who will not play, chairs the meeting.
- The Product Owner provides a short overview of one user story to be estimated. The team is given an opportunity to ask questions and discuss to clarify assumptions and risks. A summary of the discussion is recorded, e.g. by the Moderator.
- Each individual lays a card face down representing their estimate for the story. Units used vary - they can be days duration, ideal days or story points. During discussion, numbers must not be mentioned at all in relation to feature size to avoid anchoring.

# Planing Poker [10]

- Everyone calls their cards simultaneously by turning them over.
- People with high estimates and low estimates are given a soap box to offer their justification for their estimate and then discussion continues.
- Repeat the estimation process until a consensus is reached. The developer who was likely to own the deliverable has a large portion of the “consensus vote”, although the Moderator can negotiate the consensus.
- To ensure that discussion is structured; the Moderator or the Product Owner may at any point turn over the egg timer and when it runs out all discussion must cease and another round of poker is played. The structure in the conversation is re-introduced by the soap boxes.



# Planing Poker

Planing poker addresses various concerns:

- Uses proxy parameter.
- Coffee card against estimation fatigue.
- Designed so that everybody participates, not only the “loud”, “active” player.










# Formal estimation / Combination

- Requires applicable measured past data.
- Given good past data, the Models predicting the development effort may be quite involved.
- Inaccurate measurements are still useful if handled correctly.
- One could combine formal and expert approaches: e.g, turning story points to man hours based on past data.







# Resources |

- How to respond when you are asked for an estimate? - Thomas Owens response
- Wikipedia: Planning poker
- Scrum Reference Card

# References |

-  R. Červenka: Project management
-  Wikipedia: Software project management
-  Wikipedia: Brooks's law
-  Wikipedia: Issue tracking
-  Wikipedia: Hofstadter's law
-  Wikipedia: 90-90 rule
-  Jon Evans: On the dark art of software estimation
-  How to respond when you are asked for an estimate? - Thomas Owens response
-  Wikipedia: Software development effort estimation

# References II

-  [Wikipedia: Planning poker](#)
-  [Mike Cohn: The Main Benefit of Story Points](#)
-  [Richard Clayton: Software Estimation is a Losing Game](#)
-  [Scrum Reference Card](#)
-  [Scrum - Wikipédia](#)
-  [Wikipedia: Kanban](#)