Principles of Product Development Flow

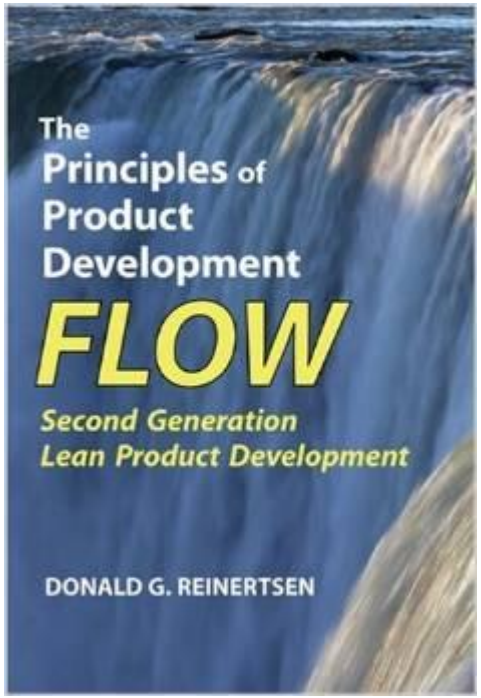
Part 7: Controlling Flow Under Uncertainty

About Me

- Started programming in 1981
- Owner of Enoki Solutions Inc.
  - Consulting and Software Development
- Exposed to several industries
- Running VanDev since Oct 2010

Book:
The Principles of Product Development Flow

- ~$45 on Amazon.ca
- Published in 2009
- Award winning
- Difficult material
- Generally ignored :(
- Awesome IMNSHO



F1 The Principle of Congestion Collapse: When loading becomes too high, we will see a sudden and catastrophic drop in output

- As utilization increases the incremental increase in flow decreases
- At a critical threshold the system collapses
  - High utilization -> High queues -> High Cycle time -> Bad queuing discipline -> Load Throughput
- Ergo: Avoid high utilization

F2 The Peak Throughput Principle: Control occupancy to sustain high throughput in systems prone to congestion.

- Peak throughput occurs immediately before collapse; operate slightly below this
- Occupancy (# of jobs in flight) is easier to control than Speed (time a job takes)
  - WIP Limits

F3 The Principle of Visible Congestion: Use forecasts of expected flow time to make congestion visible.

- Make jobs physically visible (stickies on a whiteboard)
- Forecast time to exit based on where in the process each job is
  - Not a job specific forecast!
  - QA -> Ship in X days 95% ($\mu+2\sigma$)
  - Dev -> Ship in Y days 95% ($\mu+2\sigma$)
  - etc.

F4 The Principle of Congestion Pricing: Use pricing to reduce demand during congested periods.

- Peak season costs more
- Bus zone costs apply during peak hours
- OT (used to) costs more
- Shipping product during high demand seasons costs more

F5 The Principle of Periodic Resynchronization: Use a regular cadence to limit the accumulation of variance.

- Think about two buses
  - Leed bus takes time to pick up people
    - More people accumulate at its future stops
    - Slows down
  - Trailing bus gets closer
    - Ecounters less people
    - Speed up
- Forcing a wait fixes this
- Locally non-optimal -> Globally better

F6 The Cadence Capacity Margin Principle: Provide sufficient capacity margin to enable cadence.

- Having buses wait -> need more buses
- To meet your launch schedule you need enough capacity margin to absorb schedule delays at intermediate milestones.

F7 The Cadence Reliability Principle: Use cadence to make waiting times predicable.

- If builds happen every day
  - How long until the next build?
- If we ship every week
  - How long until the next release is shipped?
- How important is it to modify the existing feature set when you ship again next week?
  - Not very.

F8 The Cadence Batch Size Enabling Principle: Use a regular cadence to enable small batch sizes.

- Regular activities on a short cycle
  - force a smaller batch size
  - reduce transaction cost
- Fixed schedules are cheaper
  - predicable meetings
  - less in flight to consider
- Unpredictable content (features)

F9 The Principle of Cadenced Meetings: Schedule frequent meetings using a predictable cadence.

- More meetings on a regular schedule
  - Less to talk about -> Quicker meetings
  - Less time between action and feedback
  - More opportunity to react to emergent issues
- Increased costs balanced against benefits
  - Reduce costs by co-locating teams
  - Don’t move the meeting
  - Structure feature elimination to remove debate

Cadence vs Synchronization

Cadence:

- Repeats on a fixed schedule

Synchronization:

- Causes two or more events to occur at the same time.

F10 The Synchronization Capacity Margin Principle: To enable synchronization, provide sufficient capacity margin.

- If two or more items are processed together they are limited by the arrival of the slowest item.
- You must provide capacity margin for the process to idle while waiting on the slowest item.

F11 The Principle of Multiproject Synchronization: Exploit scale economies by synchronizing work from multiple projects.

- Synchronize independent work of the same type
  - Setup to do a particular type of work
  - Run several work items through
- Work items wait until the that type of work is being done

F12 The Principle of Cross-Functional Synchronization: Use synchronized events to facilitate cross function trade-offs.

- If an action has to pass several hurdles to success check all of them each time
- Finding failure in series is more expensive than finding failure in parallel

F13 The Synchronization Queueing Principle: To reduce queues, synchronize the batch size and timing of adjacent processes.

- Traffic light synchronization “Green Wave”
- Requires more predictability than generally found in development
- Other areas (operations) that are more predictable can benefit

F14 The Harmonic Principle: Make nested cadences harmonic multiples.

- Outer processes synchronize on a multiple of internal cadences
- hourly -> daily -> weekly -> monthly (ish) -> quarterly -> yearly.

F15 The SJF Scheduling Principle: When delay costs are homogeneous, do the shortest job first.

- Two jobs, A 10u and B 1u
- Do A then B -> delay = 10u+11u = 21u
- Do B then A -> delay = 1u + 11u = 12u
- *If* delay matters

F16 The HDCF Scheduling Principle: When job durations are homogeneous, do the high cost-of-delay job first.

- Two jobs both 1u:
  - A +\$10 at t_0, +\$1 at t_0
  - B +\$20 at t_0, +\$19 at t_1
- A then B = \$10 + \$19 = \$29
- B then A = \$20 + \$1 = \$21

F17 The WSJF Schedule Principle: When job durations and delay costs are not homogeneous, use WSJF.

- W = Cost of Delay / E(t)
- E(t) -> very hard to nail down in software
  - **IFF** based on historically aligned data
    - **and IFF** σ/μ is << 1
    - **only** for determining which job to start next!

F18 The Local Priority Principle: Priorities are inherently local.

- Unimportant tasks in high priority projects should not override important tasks in low priority projects

F19 The Round Robin Principle: When task durations is unknown, time-share capacity.

- Some tasks are impossible
- When do we stop?
- Switching active tasks without completion
  - Task switching has costs
  - Much worse costs with people
- 80% (or more) of tasks should finish without being preempted

F20 The Preemptive Principle: Only preempt when switching costs are low.

- Prefer changing queue discipline to preemption
  - head-of-line (high cost of delay) jobs
- Preempt only low cost of delay jobs
- Include the cost of switching both out of and into the job

F21 The Principle of Work Matching: Use sequence to match jobs to appropriate resources.

- Attempt to sequence work so specialized resources (experts) become involved at the *right* time
- Move experts to the work they align with when that work is present and they are moving onto a new task already

F22 The Principle of Tailored Routing: Select and tailor the sequence of subprocesses to the task at hand.

- Not all projects are the same
- Not all releases are the same
- Not all risks are the same
- So why would the pipeline for every project be the same?
  - Of even stay static for a given project?
- One size does not fit all

F23 The Principle of Flexible Routing: Route work based on the current most economic route.

- Take the toll bridge or drive 25km more to avoid it?
  - What does 25km of driving cost?
  - What does the toll cost?
  - Take the cheaper option
- Performance: O(n)
  - in space (memory) or time (cycles)

F24 The Principle of Alternate Routes: Develop and maintain alternate routes around points of congestion.

- Who needs to review a design?
  - Have more than one option
- Who can do the release?
  - Have more than one option
- Who can do …
  - Have more than one option!

F25 The Principle of Flexible Resources: Use flexible resources to absorb variation.

- Who can cover for whom?
- Why does it have to run on **that** server?
  - or **that** database?
  - or use **that** architecture?
- Options enable choice that enable cost savings

F26 The Principle of Late Binding: The later we bind demand to resources, the smoother the flow.

- Why assign a resource to a task at the start of a project?
- Adapting to change over sticking to a plan
  - Avoid the need for change by leaving the details as light as possible

F27 The Principle of Local Transparency: Make tasks and resources reciprocally visible at adjacent processes.

- Send signals up and down the pipeline
  - Sometime large is coming, prepare
  - We have a gap in our release schedule
- Enables decisions
  - Reserve more machines to handle load
  - Push less features to fill the gap

F28 The Principle of Preplanned Flexibility: For fast responses, preplan and invest in flexibility.

- Pre-plan splits
  - If we need to drop features we do so in this order
- We need to handle more load
  - We have planned and tested scaling
- We need more people
  - We have cross trained people in other groups
    - and kept them informed
- Firedrills!

F29 The Principle of Resource Centralization: Correctly manage, centralized resources can reduce queues.

- Requires low utilization
- Requires projects to vary in the time they use them
  - Projects must be staggered
- Overall not as useful for software
  - Projects tend to bunch up

F30 The Principle of Flow Conditioning: Reduce variability before a bottleneck.

- Time at the bottleneck ~ queue size
- Queue size ~ arrival rate variability
- Arrival rate variability before a bottleneck makes the bottleneck worse
- “Neck down”
  - Pipes go from large to small in stages

Conclusions

- Pretty much a grab bag of ideas
- Focus on flow
- Mostly:
  - → Adapt
  - → Enable adaptation
  - → Pre plan adaptation
  - → “Neck down” around bottlenecks

Q&A